RPEN
A NEW 3D POINTING DEVICE

RPEN
A NEW 3D POINTING DEVICE

by

MOHAMMAD BADRUL ALAM, B.Sc.

A Thesis
Submitted to the School of Graduate Studies
in Partial Fulfilment of the Requirements
for the degree
Master of Science in Computer Science

McMaster University

MASTER OF SCIENCE (2009) McMaster University
(Computer Science) Hamilton, Ontario

TITLE: Rpen: A New 3D Pointing Device
AUTHOR: Mohammad Badrul Alam, B.Sc. (North South University)
SUPERVISORS: Dr. Ryan Leduc
Dr. Skip Poehlman
NUMBER OF PAGES: x, 171

Rpen: A New 3D Pointing Device

Mohammad Badrul Alam
Master of Science in Computer Science
McMaster University, Hamilton, 2009

Abstract

Most often the development of human-computer interfaces has been driven by the technology available at the time, and not by human factor issues. The popular 2D mouse is such a device. Through this effort we have developed an input pointing device which builds on tools and skills that humans have acquired through evolution and experience.

Major trends of graphical input apparatus have gone from indirect computer input (mouse or trackball) to direct input (tablet and touch screen). The current and future trends are more towards 3D interaction and ubiquitous input concepts. Our current effort with the **rpen** falls somewhere in between. While it is desirable to take advantage of a mouse's device acquisition technique, we need the rpen to be a direct and more natural interactive input device. At the same time, the rpen is a 3D spatial input device, seeking to take advantage of the human knowledge and skills, naturally.

The rpen is implemented on a Linux platform with the help of a kernel level device driver. The device uses an alternating current, electromagnetic motion tracker to provide six degrees-of-freedom absolute data. It also uses filtering and smoothing techniques to reduce the effects of electromagnetic distortions in the sensed data.

With the rpen, a user can define a touch screen anywhere. The touch screen can be of any size and of any orientation, horizontal, vertical or tilted. A drafting board, drawing board or sketch board can be a touch sensitive surface. The basic concept of the rpen is implemented in software and is not dependent on the physical sensor used. So, multiple input modes can be implemented through the software interface and the rpen can switch between these modes seamlessly.

This thesis presents a new 3D pointing device, the rpen, capable of doing both 2D and 3D interaction. As a byproduct of the current study it has been determined that the electromagnetic tracker is not suited to function as a motion tracker for any rpen-like generic pointing device.

Acknowledgments

No intellectual effort can be accomplished in isolation. The current study is only possible because of the help, time and guidance extended towards me by many generous and intelligent individuals. I want to acknowledge the contributions made by these individuals and extend my gratitude to them.

First, I would like to start by thanking my academic supervisor, Dr. Ryan Leduc, who is the originator of the rpen concept, for giving me the opportunity to work on this concept. He not only generated the ideas but also guided me to make it a reality. He encouraged me to develop new thoughts and new ways to look at things. He also helped in solving the minute details of the project. His ability to think out of the ordinary and the spirit to try out new ideas will always inspire me in all my future endeavors.

I feel very fortunate to have not one but two mentors. My co-supervisor, Dr. Skip Poehlman, is not only an excellent advisor but also a constant motivator. His ways of appreciating any effort, kept me going during the most trying times. His guidance and advice helped me to limit the scope of the study and focus on achieving the ultimate goal, which for me was to finish the thesis. His realistic and measured approach to research gave me a perfect balance to my research and learning. I am also grateful to my Master's defense committee members, Dr. Wolfram Kahl and Dr. Kamran Sartipi, for their interest and patience in reviewing my thesis.

During the course of the current effort, I have come across many difficult problems. Receiving unstable data from the hardware was one of them. Being a student of Computer Science, my knowledge was very limited in signal processing and noise filtering. I an extremely grateful to Mohammad Tauhiduzzaman for helping me handle these issues, with his time, efforts and knowledge, even though, being very busy himself. I would also like to thank my colleagues and friends in the Department of Computing and Software, who have helped me with their advice and contribution.

Lastly, I wish to thank my family (especially, my wife and little daughter) for their patience, love and unconditional support.

Table of Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

1.1 Motivation

The writing pen is a natural interaction tool for the human being. From the time humans learned to write, the pen has been used to accomplish that need. The pen is also naturally suited for pointing at something, as natural as pointing a finger. We seek naturalness in Human Computer Interaction (HCI) devices for two reasons. One, to reduce the gap between the user's intentions and the actions necessary to carry out those intentions, while interacting with the computer. This situation is well described as the "Gulf of Execution" in [HHN85]. The other reason is to increase the user's performance (speed of input) by providing naturalness. The main idea in such interaction techniques or devices, is to build on tools and skills, the humans have acquired through evolution and experience. As pointed out in [Shn83], direct manipulation interfaces are successful, especially, with new users, because they show resemblance to the existing human skills rather than trained knowledge.

Thus, working with pen-like devices as input apparatus for the computer should be a logical conclusion in the dominant HCI research field. This results in a mature and widely used natural interaction device. Actually, this is not the case. Nowadays, the stylus is the only pen shaped interaction device. Also in most cases, this is not used as the active pointing device but rather as a subsidiary to touch sensitive pointing devices, like the touchpad, touch screen, etc. Some of the 3D interaction devices come with a stylus option, but their primary goal is to track the motion of the device, not aid in HCI design principles.

In the two dimensional world of graphical user interface (GUI), the light pen, a pen based model, was the first interaction device. When it was developed the main focus of the HCI research studies were technology not the interaction techniques. So, afterwards, when the mouse came out, the light pen lost its popularity and the mouse has quickly become the de facto standard 2D input device [PV89]. Even though, the light pen was intuitive and provided the user with a familiar experience. However, when it comes to the interaction with a 2D interface, it falls short, not in terms of technology rather in terms of human factor issues related to fatigue, ergonomics, etc. A major concern with the light pen interface was that it had been using a virtically mounted touch sensitive

screen, providing little support for the user's hand. In contrast, the mouse used the horizontal desktop to allow the user to rest his/her hand on it, while in operation.

The huge popularity of the 2D mouse and its acceptance has led the researcher to stay with a similar kind of apparatus and even try for a 3D mouse, which are sometimes refereed to as a flying mouse or wand. As described in [Zha98], these devices have their own limitations. The standardization of the mouse has meant that often the researchers have tried to define the interaction design principles, for similar devices as well as three dimensional input devices, on the basis of its characteristics. In fact, many of the attributes of an input device are expressed in relation to the mouse. Given the status of the mouse in a 2D world, not much has been investigated by researchers that could be counted as a competitor, but rather worked on providing an ergonomic design to it. Given the current state of advancements in HCI, especially in the areas of human factor issues, pen-like devices have yet to receive a serious consideration as a viable alternative to the mouse. Not only that, there has not been a constructive study to identify the limitations of such a device.

Recent studies in HCI, more specifically 3D interaction, have emphasized a different kind of approach to computer interaction. This approach is more direct, as if manipulating the real object. The researchers argue that the user needs to be able to perceive the interaction, naturally, which not only increase one's satisfaction of interaction but also will increase the "bandwidth" of input from the user end. This interaction style is at odds with the mouse since it uses indirect interaction and is more in line with pen, which uses direct interaction. This means employing the current HCI design standards, and considering the human factor issues associated with such input device, it is possible to come up with a more intuitive and ergonomic pen input device which can position itself as a replacement for today's standard mouse and make a feasible 3D input device.

There is another piece to this puzzle. Touch screens have been the most intuitive interaction device in the 2D world. There also have been quite a few new inventions which forward the idea of touch based input devices, like Jeff Han's Multi-touch and Microsoft Surface from Microsoft. Using the human finger as a touch tool has raised quite a few ergonomic concerns. Due to this fact as well as the finger not being a precise pointing tool, various forms of the stylus are used with touch sensitive pointing apparatus. But in all these touch sensitive systems the stylus is basically a passive pointer, the screen being the main pointing device. Although being very intuitive, touch screens are not as accessible as a mouse. The mouse is a pure pointing device, independent of the screen, whereas, touch screens are both a pointing device and a display system. If a stylus, rather than being a passive device could become the active device and work similarly as it does with a touch sensitive device, then, there is a potential to obtain the best of both

worlds.

There is a considerable amount of challenges to make the pen input device a successful interaction device. A pen based active device will basically work as a 3D spatial device. Thus, the difficulties that a multi-dimensional input device would face are more or less applicable to such an input device. It has to meet ergonomic issues of grip, weight and balance. It has to meet usability issues like fatigue. There has to be ample feedback. The device has to have buttons to improve the quality of system control, which the current stylus devices lack. If these challenges are met successfully there is potential to produce an intuitive input device which is able to work in both 2D and 3D environments.

1.2 Research Goals

Most often the development of human-computer interfaces has been driven by the technology and not by human factor issues. Researchers and developers have been making what is technologically possible and then check what needs to be done to accommodate the human factor issues [JLM93]. The mouse is a prime example of this kind of development. [JLM93] argues it should be the opposite. Our current effort to develop a new pen like device, which we call *'rpen'*,[1] follows this theme. We thought of the things that are natural and intuitive to a human and could be utilized to produce a human-computer input device. Gesture and handwriting is natural to a human. Working with pen in a 3D environment is also natural to the human. A pen not only provides the affordance for handwriting but also naturally serves as a pointing tool. So, now that we knew that the pen can be a method for pointing, we want to prove that it can be a feasible pointing device for the computer as well.

The start of human-computer interaction on graphical user interface (GUI) was with the Light Pen on a Sketchpad. This device developed by Ivan E. Sutherland of MIT, was simple to use. The device quickly became very popular [PV89]. But, after the introduction of the mouse, the light pen lost its edge and eventually lost almost all of the market share. There are two important points to it. Why a pen like interface was the first mode of human-computer interaction in the graphical era and the other is why it has failed. The Light Pen resembled the real life, real object manipulation experience. Its direct and intuitive nature of interaction technique was what motivated the developer as well as the general public. But, the lack of understanding of human factor issues has failed the device. The field of HCI, especially the human factors issues related to human-computer interaction device, is more mature now than it was during the time of light pen. Thus,

[1]The "r" stands for "radio" as we originally envisioned the rpen using radio triangulation to determine its position in 3D space.

it is important to investigate the usability issues related to a pen like device, in light of today's knowledge of HCI.

Studies have shown users prefer direct interaction rather than indirect interaction, for which touch screen like devices always have been popular. Also, usability studies have shown relative position movements are an acquired skill, whereas absolute position is what we are more accustomed to in real life. Thus, we want to develop an absolute positioning device which can work in both direct and indirect interaction mode, i.e., like a mouse as well as a touch screen.

Presently, there is a spectrum of pointing devices in use. The mouse is the most prominent of them all. Others include trackball, isometric (force-based) joystick, isotonic (displacement-based) joystick, indirect tablet (using stylus only), touchpad, and touch screen. All of these pointing devices have different functionalities and have distinct fields of applicability. Our goal is to develop a pointing device which is able to provide versatile functionalities. It can work like a mouse in a confined space, provides the intuitiveness of touch screen and can also work as an indirect tablet. With orientation information, the proposed device could also imitate the functionality of a joystick.

The rpen works as a 2D input device mimicking the mouse. But, actually it is a 3D spatial device. As such, it is able to work both in two dimensions and three dimensions. In this sense the rpen is a hybrid interface [HPG94]. Research study on such interfaces is rare. It has been observed that mapping a more than 2 DOF (degree of freedom) gaming application to a 2 DOF mouse creates dissatisfaction among users and, as such, they usually prefer a keyboard to a mouse in such cases [SWK04]. It would be interesting to observe the opposite, using a higher DOF interactive device to do interaction, usually efficiently done by the 2 DOF mouse.

Designing a 3D interaction device is difficult. There are various design issues that needs to be successfully addressed to develop such a device. Even though, a considerable amount of work has already been done on 3D interaction devices, so far, these efforts have yet to produce a 3D device which has a wide range of applicability. Most of the 3D devices are application orientated. Through the current effort, we hope to produce a spatial pen based input device which could have wide range of applicability both in 3D and 2D.

1.3 Scope and Limitations

The primary goal of the current study is to provide proof of the rpen concept. We are to produce a working prototype of the rpen. To achieve this goal we require both hardware, to track the absolute position, and software to interface the hardware with a host system. The software interface will not only carry the

hardware data to the host system but also will interpret it through the concept and design of rpen. So, the software contains the basic architecture of the rpen and the hardware is there only to provide accurate and reasonably precise 3D absolute position information.

As the hardware could be any 3D absolute position tracking system, commercially available or a new product developed for rpen, in this study we have only focused on the software, making the choice to go for commercially available hardware. This choice inherently comes with the limitation that the hardware design, related to various human factor issues, can not be resolved, if needed. It also means that we have to rely on the hardware vendor on reliable delivery of accurate and precise 3D absolute position data. Small tuning could be done on the hardware data but an extensive exploration on hardware device data, having to do with its accuracy and delivery, is bound to put enormous burden on the time constraint and available human resources.

Hardware experts, specifically with expertise on motion tracking technologies, are not part of the research group. Even though, the rpen is independent of the hardware device in the sense that the main concept is implemented in software, nevertheless, hardware is an essential part of the total device. Rpen's success in many parts dependent on the accuracy, precision and fast data communication abilities of the hardware. An expert in this field not only could help in choosing the best suited for the rpen device, but also could have helped in case of unforeseen problems arising due to the hardware.

Issues related to designing the rpen as a 3D spatial device is part of the thesis. In this sense, identifying the hardware design principles is part of the thesis. Ergonomics and other human factor issues influencing rpen is raised here as well. But, due to the time constraint many of these issues could not be explored fully. Also, some issues arising due to the design principle of rpen, could only be explored, identified and addressed, partially, by the current study. Money was also an important constraint, which forced us to make a cost conscious hardware choice, lacking accuracy, precision and speed.

1.4 Thesis Organization

This thesis is organized in five chapters. At the end of the fifth chapter there is an appendix and the document ends with references.

- **Chapter 1**: This is the introductory chapter of the thesis containing the motivation to pursue the current study. This also provides the goal of the research and defines the scope and limitations of the study.
- **Chapter 2**: This chapter provides the background of the research work. It

contains a survey of motion tracking technologies along with commentary on their strengths and weaknesses. The chapter also contains information about the input device; their attributes and types of input device. It describes design issues related to 3D and spatial input devices. Lastly, the chapter provides a summary and critique of related research and products.

- **Chapter 3**: This chapter presents the design basis of rpen. The chapter starts off with the methodology used for the current research. Then it gives a background on Linux device drivers, more specifically, USB device drivers and input device drivers and how the rpen is implemented as a device driver in Linux. It also provides a brief description on rpen hardware. Before going on to detailing the implementation details of the rpen, the chapter provides a summary on the filtering and calibration techniques, usually used by the electromagnetic devices and what the rpen has implemented and why. Chapter 3 ends with details on the approach taken for rpen in regards to usability testing.

- **Chapter 4**: Chapter 4 presents the experimental results of current study. The chapter also summarizes the user feedback from the usability test.

- **Chapter 5**: This chapter presents the concluding remarks. A brief analysis of the results is also presented in this chapter. Contributions to knowledge from the current study is highlighted towards the end of the chapter. At the end, a discussion of future works is given.

- **Appendix**: Appendix A contains the questionnaire used in the usability test as well as the user instruction manual provided to the participant users as part of the test. Appendix B contains the source code for the rpen device driver, rpen related applications and daemon processes.

CHAPTER 2

Background and Literature Review

2.1 Motion Tracking Technology

In this section we examine different tracking technologies, which could be used with the new pointing device, the rpen. First, we establish a set of criteria for evaluating the tracking technologies. Then, we present the most widely used motion tracking technologies. This survey by no means is exhaustive or complete.

2.1.1 Measuring Tracker Performance

To choose a motion tracking technology, a performance measuring framework is required. This framework helps the user to compare different tracking technologies as well as trackers and can decide upon a specific tracker, which meets the user's needs. [MAB92] provided a general purpose performance measuring framework, which was also adopted in [Bac00][Sko96]. The following are the five components of this framework:

1. **Resolution**: This measures how well the tracker or a tracking technology is in detecting small changes in motion. The more resolution a tracker has, the better it is in detecting minute changes in the tracking system.

2. **Registration**: This is mainly the accuracy of the system. It measures the correlation between the reported position and orientation of a system and the true position and orientation.

3. **Robustness**: This measures the tracker's performance in the presence of noise and other interference in the working environment. A system needs to provide a consistent and predictable output in such conditions, so that, a correction measure can be taken to compensate for the errors.

4. **Responsiveness**: This property is the measurement of the sample rate of a tracker, which is also linked to latency or time lag. A user's satisfaction in using such a tracking system, which employ a tracking device, is very much dependent on this criterion. This gives the feel of controllability to the user.

5. **Sociability**: This measures how good a system is in tracking multiple objects or users, i.e., working with multiple sensors.

This framework is more suited in the context of virtual reality, but, most of it, is also true for all systems. For a motion tracker to be used with the rpen, the criterion of sociability is not needed in this thesis as it only tracks a single object or sensor. For rpen like devices, the working volume is an important factor. It is the volume within which the tracker can successfully track an object or sensor. There are other criteria as well, to compare the tracking systems. For example, a tracking system should not require a special environment to operate. The wearable part of the system or the part of the system to hold in hand should be light in weight and comfortable to use.

Other then these performance factors, there are some other aspects to take into consideration before opting for a motion tracking system. Most important of these are: cost, availability and ease of use. Certainly, in the context of the current study, cost and availability are very important factors.

A motion tracking system for the rpen needs to meet some minimum requirements. It must have a reasonably fast sample rate so that any changes in the position could be translated to the display immediately. There is no acceptable amount of room for latency for an interactive device like the rpen. The tracker system should have high resolution, so that any tiny changes of the tracker could be reflected in the display. Position accuracy is vital for the rpen, as the basic functionality of the device is dependent on it. The tracking system also needs to work in a regular office or home environment, as we envision it as a general purpose pointing device. All in all, as the rpen is set to do the tasks of a mouse, a touch screen and other related pointing operations, it should provide at least a set of performance measurements comparable to the mouse. The users of the device will be expecting it.

2.1.2 Types of Motion Trackers

Here we present some of the major motion tracking technologies presently in use.

2.1.2.1 Mechanical

Mechanical trackers are the oldest motion tracking technology. They are comparatively also the cheapest motion trackers. Mechanical trackers can measure the position and orientation of an object by attaching it to a movable mechanical arm. The arm has a reference point and some other moveable joints, which can move and rotate. These movements and rotations relative to the reference point are measured to produce the position and orientation data. Potentiometers or

optical encoders are usually used for the measurements. Mechanical trackers are mainly ground based systems. Phantom from Sensable Technologies [Inc09d] is an example of this kind of motion tracker. [MPC04] also proposes a mechanical motion tracker named "DigiTracker." The article claims the device could work both in absolute and relative mode. In fact, absolute position and orientation data can be inferred from the original reference point, while, to produce relative movements a clutch mechanism has to be used.

There is another version of a mechanical tracker which basically tracks human body movements, thus, an exoskeleton based system. Gypsy-6 from Animazoo UK Ltd. [Ltd09] is such a system. There is yet another type of mechanical tracker which measures the force extended on a device.

Mechanical trackers usually provide accurate position and orientation data, but are limited by the range of the mechanical arm. In case of exoskeleton based systems, they require an external position tracker to ascertain the user's body position within the work area. Mechanical trackers are good as haptic (force feedback) systems. In fact, the Phantom motion trackers are force feedback systems.

2.1.2.2 Inertial

Inertial tracking systems are usually used to measure the orientation of a tracked object. Mechanical gyroscopes are used to measure the orientation of an object in space. At least two gyroscopes are required to find the 3 DOF orientation information. These sensors suffer problems associated with drift. This problem can be minimized if only the relative orientation is considered rather than the absolute orientation [RBG00]. InertiaCube from InterSense Inc. [Inc09a] is an inertial tracking system which provides 3 DOF orientation data.

The use of accelerometers provide relative position data for these kinds of tracker. The accelerometer produce rate of acceleration, and the position is derived from this data, assuming the starting condition (position and speed) of the target is known.

An advantage of this tracking system is that it does not need any external reference to measure the movements of an object. This means these trackers are not influenced by occlusion or noise. Also as they do not have much inherent latency, these trackers can be sampled at a high frequency. These trackers are usually light weight, small and cheap, particularly, the accelerometer. But, these devices tend to accumulate error over time and need to be calibrated from time to time.

2.1.2.3 Acoustic

Acoustic trackers use ultrasonic waves to determine the position and orientation of the target. There are two methods to track an object with ultrasonic waves, one is Time of Flight (TOF) method and the other is phase coherent (PC) method. In the TOF method, the position and orientation of an object is calculated by measuring the time taken by the ultrasonic waves to travel from a set of transmitters to a set of receivers. Usually one transmitter and three receivers are required to produce 3 DOF data and three transmitters and three receivers are needed for 6 DOF information. The transmitters are mounted on the object and the receivers are set at known fixed positions in the tracked environment.

On other hand the PC method tracks position and orientation by calculating phase differences of the emitted acoustic waves and the phase of a reference wave. This technique uses an incremental method; thus, it requires knowing the starting location of the tracked object. This also means there will be error accumulated over time and thus the device needs to be recalibrated from time to time [Bac00].

Both of these methods suffer from the effects of occlusion or shadowing [Sko96]. Also, echoes and reflections of sound waves are major concerns for both of these methods [Eric 2000]. The position is determined by the velocity of sound in air, which is greatly influenced by air temperature, pressure, turbulence and humidity [RBG00]. IS-900 MiniTrax Wireless Wand from InterSense Inc. [Inc09a] is an example of an acoustic motion tracker. Acoustic motion trackers are usually inexpensive, such as the IS-900 MiniTrax Wireless Wand which costs around $\$3,000$ USD.

2.1.2.4 Optical

Optical motion tracking is the most active research area amongst all other motion tracking systems. Optical trackers with their high resolution, good level of accuracy, and huge working volume tend to provide this high interest by the researchers. Diverse research activities have led to wide range of technologies being used for optical tracking systems. These technologies broadly, can be categorized as using outside-in or inside-out configuration [RBG00].

As can be seen in figure 2.1, the outside-in configuration works by placing the sensors (in the case of optical trackers this is the camera) at reference points around the working volume. These cameras track the target objects having some sort of markers. This is the most widely used technique for optical trackers [RBG00]. There are two methods used to track the objects with outside-in configuration, one is pattern recognition and the other is image based [Bac00].

In image based methods, the visual information captured by the sensors is processed through the image processor and triangulation is used to determine the

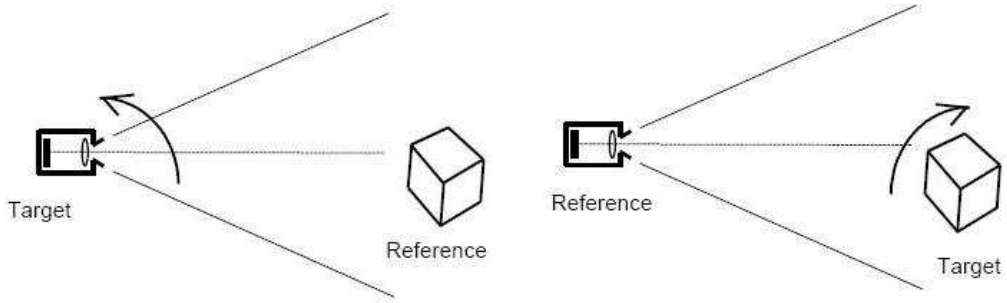


Figure 2.1: The inside-out and outside-in configuration [RBG00].

position of individual markers. The markers used in this method can be either active, emitting light sources, in most cases infrared LEDs, or the markers could be passive (retro-reflective) markers [Sko96][Won07]. The active marker option is better as it gives higher sampling rate and individual markers can be uniquely identified [Won07]. Vicon Motus from Vicon [Sys09] uses retro-reflective markers, whereas, Impulse System from PhaseSpace [Inc09c] can track up to 128 active LED markers.

Pattern recognition systems determine position and orientation by comparing the known patterns with sensed patterns. Infrared LEDs are usually used as markers [Bac00]. Pattern recognition systems require complex algorithms to decipher image content [Sko96]. The helmet tracker from Honeywell uses this technique to calculate helmet orientation [Fer91]. Pattern recognition techniques are also used for inside-out configurations. In an inside-out configuration, the sensor is on the target and the emitters are on fixed positions around the working volume. The HiBall developed by UNC Chapel Hill uses this technique [WBV99]. The major limitation of this system is that it cannot determine full orientation data and it requires a specially configured ceiling where the LEDs are set in specific places [Bac00].

There is yet another optical tracker technology which uses structured light, a laser, to scan the target object. It can scan specific points, the entire scene or random positions to determine the position information [MAB92]. OPTOTRAK optical tracker developed by NDI [Inc09b] uses laser scan technology.

Even though, there is a wide veriety of optical tracker systems, they all share several common limitations, the most important of which is occlusion. It is required to maintain a clear line of sight, to track the target successfully. Depending on the light source being used, other noise sources could be optical noise, spurious light and ambiguity of sensed surface [RBG00]. These tracker systems are unable to determine orientation data directly. They have to be determined from the neighboring markers. In comparison to other tracking technologies, op-

tical trackers require greater amounts of processing time [Won07]. Lastly, the cost and availability of these systems precludes them from being considered as a viable option.

2.1.2.5 Electromagnetic

The electromagnetic trackers operate by measuring electromagnetic signals. Such magnetic fields can be generated by circulating electricity in a coil. Every electromagnetic tracker has a transmitter, which generate these electromagnetic signals. Each transmitter containes within itself three orthogonal coils. These three coils are energized sequentially in each measurement cycle, to produce the magnetic fields. When a receiver/sensor is placed in this field, a magnetic coupling or induced flux is formed between the receiver and the emitting coils. The flux is a function of the distance and orientation of the receiver in relation to the emitting coils [RBG00]. For each measurement cycle, nine signal measurements are needed to derive the 6DOF data of the receiver [NMF98].

Electromagnetic trackers are relatively inexpensive, light-weight, and compact. Electromagnetic trackers are the most widely used tracking system. They do not have the problem of occlusion, like the acoustic and optical trackers. They have a reasonable update rate, good resolution and in favorable conditions, have reasonable accuracy.

There are two types of electromagnetic trackers. One is the Alternate Current (AC) and the other is Direct Current (DC) electromagnetic tracker. The AC magnetic tracker continuously creates changing magnetic fields. The three orthogonal passive coils contained in the receiver is induced by these fields [NMF98]. Invented by Polhemus [Inc09e], AC magnetic trackers have been in the market for nearly 40 years. The AC tracker generates Eddy currents in the surrounding metallic and other electromagnetic objects. This Eddy current is the main reason for the distortion of measurements in AC tracker systems.

On the other hand, DC trackers invented by Ascension Inc. [Cor09] uses rectangular pulse magnetic fields and take measurements of the magnetic fields after a steady state has been reached. At each measurement cycle, Eddy currents are also generated in DC systems but they wait for the currents to die out before taking measurements [RBG00]. The DC trackers also compensate for the Earth's magnetic fields [NMF98]. The problem with DC tracker is when operating near ferrous metals, such as carbon steel and iron alloys, where interference effects occur.

The electromagnetic (EM) trackers have a limited working volume. Increasing power of the magnetic field to increase the working volume will result in noise from the power source [Sko96]. Also, [NMF98] describes the error in the position and orientation information will increase at a rate of the fourth power of

transmitter and receiver separation.

2.1.2.6 Radio Frequency (RF)

Radio frequency (RF) systems are yet to be used as small scale motion tracking systems, for virtual reality or similar applications. This technology is already in use for tracking ships, planes, missiles and other applications such as Long Range Navigation (Loran) and Global Positioning System (GPS). So far, they cannot be used for small-scale object tracking due to the system's errors in signal processing. However, recent advancement in RF technologies could make them a viable alternative for small scale motion tracking [Bac00].

2.1.2.7 Hybrid

This category of motion trackers basically uses a combination of the above mentioned technologies. All of the motion tracking technologies developed so far have their limitations. Some are good with global absolute position tracking, like the optical trackers, whereas, some others like the inertial tracker are good with orientations. So, a clever fusion of any of these technologies has the potential of making a hybrid tracking system, which could eliminate the limitations of the individual tracking technologies. In fact, most of the recent motion tracking systems involves such hybrid devices [Won07]. IS-900 motion tracking system from InterSense [Inc09a] is an inertial-acoustic hybrid tracker. [Won07] also uses a hybrid tracker combining inertial and optical tracker. [SHC96] used a magnetic-optical hybrid tracker. Also, in [Bac00] an inertial-magnetic hybrid tracker was used. A version of hybrid tracking technology, a combination of inertial (both gyroscope and accelerometer) and optical trackers, is very popular nowadays. WII remote as well as the proposed Apple's new gaming apparatus is such a tracker.

2.1.3 Comparison and Summary

Even though, mechanical trackers are usually very good with their accuracy, resolution and robustness, they are confined to a very small working area. Also, they are mostly used to produce relative position and orientation information. Similar to a mouse, the use of these devices has to be learned as they are not intuitive. Although, mechanical trackers are good for haptic feedback, an important component for 3D interaction, we have not considered these trackers due to their very small work area, ease of use and failure to directly produce absolute placement information. Similarly, inertial trackers are good with orientation information but cannot produce absolute position data. Even though, initially we were interested in motion tracking via radio frequency (RF) technology, this is

Table 2.1: Comparative data of three major motion tracking systems.

Property	Electromagnetic	Optical	Acoustic
Accuracy	Moderate	High	High
Resolution	Moderate	High	High
Time Complexity	Low - Moderate	Moderate - High	Moderate
Working Range	Low	High	High
Cost	Moderate - High	High	Moderate - High
Major Limitations	Electromagnetic field distortions	Occlusion	Occlusion, air temperature, humidity, pressure and ultrasonic noise

not a viable option with the current state of development. So, we were left choosing among optical, acoustic and electromagnetic trackers. All of these trackers produced absolute position and orientation data directly.

As table 2.1 specifies, the acoustic tracker is susceptible to various noise elements which is not a good option for a general purpose device like the rpen. Similar to optical trackers, acoustic trackers are also affected by occlusion. This is a major concern for motion tracking devices. Optical trackers usually try to solve this problem by using multiple sensors and other techniques which results in increased computation time as well as cost/price of the product. Taking into account of these limitations and cost considerations, we opted for the electromagnetic motion tracking system. Also, the use of these devices in many virtual reality systems and research, has influenced us to make this choice. Lastly, there is no hybrid tracker available in the market that could be used with the rpen. It could have been a good option, if we were to develop our own hardware. However, initial negotiations with appropriate Electrical and Computer Engineering faculty to this end did not come to fruition so this was not possible.

2.2 Input Device

2.2.1 Input Device Attributes

An input device has several attributes. Depending upon these attributes, they usually differ in functionality or applicability. An input device's movements are generally either absolute or relative (defined below). An input device might employ both modes, but only one can be used at a time for a specific functionality. An input device might provide only visual feedback, which is usually the de facto standard for any input device, while some others provide tactile, haptic (force feedback), auditory or special visual feedback. The sensors of the device might

measure displacement or strain. Then again the displacement might translate to relative movement or absolute movement. Input devices also differ by their footprint. Devices that measure displacement (e.g. a mouse) have a variable footprint, whereas, the Spaceball (trackball) or the EGG (elastic) developed by Zhai [Zha95] have small footprints and measure strain on the device.

Important attributes an input device may possess are listed below with brief explanations.

- **Property sensed**: Most of the traditional devices sense position, motion or force. Some others, like the rotary devices, can sense angle and torque [Hin08].
- **Transfer Function**: A device can modify the signals it senses using a mathematical transformation to provide appropriate, smooth and effective operations. A transfer function can be used to map force-to-velocity, position-to-position or velocity-to-velocity. In the case of the rpen, we are required to use position-to-position mapping inside the transfer function.
- **Sampling rate**: The sampling rate determines how often measurements from the physical sensors embedded in the input device are taken and sent to the computer system. Increased sampling rates produce finer control over the input [Hin08]. Higher sampling rates are important for touch screen otherwise there might be selection errors.
- **Resolution**: Resolution is a metric involving the number of unique measurements that the input device can send to the computer (pixel array).
- **Latency**: Latency, or lag, is the time that elapses between the physical actuation of the input device and the resulting on-screen feedback. Latencies above 100ms interfere with cognitive performance [MKS01].
- **Noise**: Noise is the result of sensing errors due to hardware malfunctions or design inadequacies. Increased noise leads to sampling problems and loss of accuracy.
- **Position mode**: The position mode can be either absolute or relative. For an absolute input device, each position on the sensing surface corresponds to a specific location on the screen. In a relative positioning mode, each input is a functional translation from the current point. A touch screen is an absolute input device, whereas a mouse is a relative input device. In terms of position, the rpen works as an absolute positioning device, but is not limited to the screen as the working volume. The rpen position is determined in arbitrary 3D space units, and then a defined frame of reference is used to interpret it's movements. This allows the device to be

used at locations comfortable to the user, or to suit specific purposes such as mimicking a mouse or touchscreen, or to define different usage contexts.

- **Gain**: Gain is also referred to as the Control-Display (C-D) ratio. C-D is the ratio of the physical movement of the input device in relation to the distance that the on-screen cursor moves. An decreased gain allows for a smaller footprint, i.e., less space is necessary for the input device. The function that controls the gain (C-D ratio) is frequently configurable through software. A high C-D ratio affords greater precision, while a low one allows more rapid operation and takes less desk space [Jac96]. Acceleration and deceleration mechanisms, depending on the movement of the device by the human operator, can increase a user's satisfaction level with the input device.

- **Degrees of freedom (DOF)**: Degrees-of-freedom is a measure of the number of dimensions that the input device senses. For navigation through 3D space or only for 3D space position information, a 3D device must have 3 DOF. To manipulate 3D objects in a volume, the input device usually needs 6 DOF [Zha98].

- **Direct versus indirect**: If the input surface is also the display surface, then the input device is direct. An example of such a device is a touchscreen. Most other input devices are indirect in that the on-screen cursor is controlled through an intermediary device such as a mouse, joystick, or touchpad. In the case of our device, it can operate with both a direct or indirect input device as we can define the work/input surface for our device to be the display surface or any arbitrary surface located anywhere on the 3D space having any orientation (horizontal or vertical or angles thereof).

- **Footprint**: Footprint refers to the amount of space that is required for the operation of the device. For example, a mouse has a large and variable footprint, whereas a trackball has a smaller but fixed footprint.

- **Device acquisition**: Device acquisition defines how easy it is to acquire (locate and hold) a device. Device acquisition and persistence is an important factor for 3D spatial input devices. Such a device needs to be easy to acquire and when released must hold its position. This criterion is most often overlooked and is a major reason why the mouse has dominated the 2 DOF input device paradigm rather than pen-like input device [Zha98]. As the rpen is a 3D spatial device, we are required to give special attention to this aspect.

- **Feedback**: An input device usually provides visual feedback only. But, most studies have suggested that it is better for the usability of the device to provide auditory or some combination of tactile and haptic feedback.

The above categorization is based on manually-operated (devices operated using the human hand) continuous pointing device. The non-manually-operated devices include the human hand or body tracking devices, eye trackers, speech recognizers and other hybrids. These devices are mostly computer vision based devices, i.e., using cameras. As our device is basically a manually-operated continuous pointing device, we need not be concerned with non-manually-operated devices. As well, there are some approaches which attempt to classify the input devices according to their ergonomic differences, to help guide the selection of the appropriate device for a task [MCR90][BS90].

The rpen is an absolute position device, which we expect will be more natural, intuitive, and ergonomic, to use. We initially intend to make it work as a replacement for indirect 2D and 3D mice, as well as function as a touch screen. As we have seen above, a 2D or a 3D mouse is usually a relative device. Thus, it has a smaller footprint and lower C-D gain ratio. It is also an indirect pointing device. On the other hand, a touch screen is an absolute position device with a C-D gain ratio of 1. Its footprint depends on the size of the screen. It employs direct interaction techniques. So, the rpen is a free moving (in 3D space) input device using absolute positioning and both direct and indirect interaction techniques. The C-D gain ratio is 1 if the work surface is the same as the display surface but will be different for other resolutions of the work surface.

2.2.2 Types of Input Device

Here in this section we present some of the traditional input pointing devices as well as some of the new trends.

2.2.2.1 Traditional Pointing Devices

Mouse: The mouse is the prime choice as a pointing input device for desktop graphical interfaces. The regular mouse is a 2D input pointing device. The relative position movement of the device is reflected on the screen with cursor movements. A mouse has a good device acquisition mechanism, i.e., the cursor holds its position on the screen as the device is released. The buttons on a mouse are easier to operate as the force given by the user to activate the button is perpendicular to the mouse's plane of motion [Hin08]. The arm, finger, wrist, and shoulder muscles are utilized to operate the mouse, which allows fast and crude movements as well as smooth and precise movements [ZMB96]. However, the indirect motions must be learned and are awkward for such natural operations as drawing and handwriting, as anyone who has tried this with a mouse can attest to.

Trackball: The trackball is a mechanical ball that can roll in one place.

It senses the relative motion of the ball in 2D and translates it to 2D cursor movements. As the ball only moves in one place the trackball has a very small footprint. It uses different muscle groups and many users find it comfortable. The trackball cannot accommodate buttons on to itself. Thus, it is seen that the buttons are sometimes housed on a platform, similar to the mouse. However, performing a button click while operating the ball is difficult and awkward.

Joysticks: There are various forms of joystick. They can be categorized mainly as isometric or isotonic. Isometric devices sense the force or pressure on the device [Zha95]. These devices usually do not move but if they move, they usually come back to the initial position when released. Example of an isometric joystick is IBM's Trackpoint. "The rate of cursor movement is proportional to the force extended on the stick" [Hin08]. Isometric joysticks have a tiny footprint, thus can be integrated with the keyboard as can be seen on IBM's notebooks. On the other hand, isotonic devices are those which are free moving displacement devices [Zha95]. Isotonic joysticks sense angle of deflection. Thus, they move more than that of the isometric once. They may or may not have the mechanism to return to the initial position. If the rpen sensor can detect 6DOF, then the rpen can be used as an isotonic joystick by holding the tip stationary, and using the orientation information.

Tablet: A tablet can sense touch on the screen and provide a 2D absolute position. Touch tablets sense the bare finger. On the other hand, graphics tablets and digitizing tablets usually use stylus based pen-like devices to sense the input. In most cases tablets work in absolute mode, having 1 to 1 correspondence of touch to control movements. In relative mode, the tablet responds to stylus movements, i.e., there is a continuous motion of the cursor. Thus, the tablet supports many forms of interaction in respect to other forms of input devices like the mouse [Hin08].

Touchpad: The touchpad is a touch sensitive input device similar to the tablet but it does not support a display. Touchpad is relatively much smaller than the display, and thus, is required to work in the relative mode. But, some of the touchpads also support an absolute mode for scrolling. It senses clicks by figure tapping. Touchpads require frequent clutching and could be difficult to use while holding a key on the keyboard [Hin08].

Touch screen: A touch screen is a touch sensitive transparent display. It is also a two dimensional device. Most often touch screens are made to sense a bare finger but some are also developed only to sense special types of stylus. The touch screen is an absolute device, and thus, has a C-D gain of 1. It is also a direct interactive tool, and hence, very intuitive. But, there are a few design and ergonomic issues with this device. Depending on how the screen is set, specially if it set vertically or near vertical, then operation on the device might result in arm and neck fatigue [SPS92]. One solution is to set the touch screen horizontally.

But, this solution has its own problems. Another problem is that there is no "hover" state for a tool tip [Hin08], or a right button for context menus.

2.2.2.2 New Trends of Input Devices

Haptic input devices: One of the findings of [Zha95] was that a rich set of feedback is important for the user to feel the interface and quickly learn the controls. In is not enough for some of the input devices just to provide visual feedback. In some cases, tactile and haptic feedback may increase user performance, particularly, when 3D input devices are involved. But because of the nature of haptic feedback, unfortunately only mechanical input devices can implement this form of feedback. These are tethered multi-DOF devices and usually have huge processing requirements, and thus are usually very expensive.

Multi-touch input: Multi-touch is a version of the touch screen or touchpad, which can recognize multiple touches on the work surface. Many different techniques are used to detect multiple contacts. Recently, this interactive style is used in products like, iPhone and iPod Touch from Apple, Multi-Touch Collaboration Wall from Perceptive Pixel, Microsoft Surface from Microsoft and several others.

Pen based gesture input: Before the use of pen like stylus devices in Palm Pilot and Tablet PCs, researchers have been studying pen based sketching interfaces [Her76]. Pen based gesture input techniques are also used in digital handwriting and digitizers. With pen based interfaces like these, one problem is when and how to differentiate between the gesture of command, and regular ink or editing tasks [Hin08].

Voice input: Human to computer interaction is very slow in comparison with the computer to the human. This gap can be minimized if human speech can be interpreted by the computer. But, until now speech recognition has had limited success. A small amount of commands or words can be understood by the computer. Also, the amount of error grows with the vocabulary and the complexity of structure of the command. As well, it is difficult to use speech to specify a location in space, and thus, it cannot eliminate the use of a pointer [Hin08].

Eye movement: Eye movement-based input is another way to increase the input rate from human to computer. It is a very natural way of communication. Eye tracking mechanisms usually track the eye gaze, that is, as the human eye fixates the object it is looking at, within the fovea. Eye gaze tracking has a limitation of accuracy of about one degree [ZMI99]. Eye movements are subconscious. So, the movements have to be tracked carefully, otherwise, the user will be annoyed with incorrect system response. Thus current eye tracking systems are expensive and have limitations.

Head movement: Applications in virtual reality and augmented reality have been using head movement tracking for quite a while now. Depending on the orientation of the user's head, a head-mounted display can change its display content. Some of the head tracker and head-mounted displays could be awkward for the user. In these kinds of applications tracker accuracy and low latency are critical as they might create "simulator sickness".

Hand gesture input: It is natural for the human to do gestures using the hand. Human hand gesture can be categorized as semiotic, ergotic, or epistemic. Empty handed semiotic gestures, which carry meaningful information, are usually the focus of the interaction research [Hin08]. Many virtual reality applications and research studies have been using data glove, marker-based as well as marker-less hand tracking technologies to capture hand gesture input. This technology also has been used to interpret sign language. In [FH93], the researchers used a data glove and an electromagnetic 6 DOF motion tracker to interpret a pre-defined set of hand gestures. One of the major challenges of hand gesture input is the gap between the user's objectives and the computer's interpretation. There could be false hand movements from the user as well [Hin08].

Binomial input: In the real 3D world, humans perform their tasks using both of their hands. While writing a person would write with their dominant hand and use their other hand to keep balance or adjust the paper. This assignment follows Guiard's kinematic chain theory, which is the non-preferred hand precedes the preferred hand in a task and the non-preferred hand, sets a frame of reference for the preferred hand to work upon [Hin08]. Two handed input, even for single-object manipulation, can offer the user the greater ease in use, natural object manipulation and less strain on both hands [HPG94]. Researchers have been using this technique to arrive at a good HCI interface, especially with 3D object manipulation.

2.2.2.3 Summary

It can be seen from the trends of input apparatus, the traditional ones used for 2D interaction have gone from indirect computer input (mouse or trackball) to direct input (tablet and touch screen). The current and future trends as can be seen above, are more towards 3D interaction and ubiquitous input concepts. Our current effort with the rpen falls, in between. While we want to take advantage of a mouse's device acquisition technique, we want the rpen to be a direct and natural interactive input device. At the same time, the rpen is a 3D spatial input device, trying to take advantage of human knowledge and skills, naturally.

2.3 Design Issues

Although the rpen is mainly modeled to manipulate a cursor in a 2D monitor display, the generated events can be interpreted in a 3D environment as well. Moreover, the rpen in essence is a free moving 3D interactive device, which according to some literature [HPG94] should be distinguished as a spatial device. The reason for this distinction is that the 3D interaction/input devices not only include free moving 3D devices but also those devices that are manipulated from a fixed reference position, which are usually rate controlled, force-based or displacement-based (mechanical tracker). So, the design challenges that affect a 3D spatial device will also apply to the rpen.

Before we move into the design issues pertaining to 3D spatial input devices, we need to address a couple of vital design decisions taken for the rpen. First, the choice made in favor of 3D input device to accomplish mostly 2D activities, needs to be addressed. In the last section, we mentioned a few of the limitations of the various 2D interfaces, especially the mouse and touch screen. It can also be noted in the current and future trends that researchers and interface developers are going towards 3D interactive devices, as they are more natural and intuitive for the users. As it is noted in [Han97], some of the tasks that we do in a 2D graphical interface: selection, rotation, translation, etc. do have direct correspondence to real world actions. And the real world object manipulation is done in 3D.

Also it is to be noted that even though our display metaphor is most often not three dimensional, more and more applications and games nowadays are made for multidimensional interactions. The standard 2D devices are falling short to deliver satisfactory interaction with these applications and games. [DTR07] investigated different input devices for a 3D data visualization environment. The study found that a 3D wand device, in comparison to a PC-tablet and voice interface, was much better in terms of performance and ease of use. In fact as a displacement device, a free flowing 3D device is much faster and precise than any of its 2D counterparts for moving the cursor from one place to another and pointing at the target without losing the hovering motion. All in all, as [Han97] pointed out, a 3D input device is "very direct, expressive and natural".

The second decision we made was to use the touch screen metaphor. This was because touch screens are the most direct representation of the display screen. [VB05] describes direct manipulation with pointing and clicking to be the most widely used form of the interaction interface. The paper also went on to describe that a touchable display mimics the real world interaction as it enables the user to feel the touch through finger or a pen tapping. To use this advantage of direct manipulation, rpen utilizes the touch screen metaphor to define its work surface, which is called an *'rframe'*. The benefit of rpen's work surface is that it is virtual, and thus can be anywhere and of any size. Also, the rpen being three dimensional

means that the work surface is basically a rectangular cube, where the virtual touch surface has a thickness and is inside this cube.

The reason to opt for pen shaped input device has already been mentioned. The pen being natural at pointing makes it good as a tool to be used for precision pointing as well as gestures. Now, the important thing is to look at the design aspect of this pen like device which needs to be ergonomic, effective at pointing and balanced for use. The pen has been used by humans long enough to come up with various ergonomic designs such as an improved grip for longer and comfortable usage, it easily rests in the hand (between the fingers), has good balance while held in the hand and excellent at precision work as it has a distinctive pointing direction. Even though the pen used in hand writing, it is not totally comparable to a pen-like pointing device. However they do share these common properties.

The main difference between the two is that while using a conventional pen most parts of the hand can be rested on the desktop. However, while using a pen based spatial device, that may not be possible much of the time. This raises an important human factor of fatigue that needs to be addressed in the rpen design. According to [KWR06], fatigue is not an issue for high speed placements and gestures, but rather it becomes distracting when fine and precise object manipulation is required. The rpen addresses this issue by offering both direct and indirect methods. Its direct method allows one to use the rpen like an intuitive touchscreen, and its indirect method allows one to use it more like an absolute positioning trackpad, allowing the user to rest their hand on the table surface. The clutch mechanism the rpen has, also provides the user with an opportunity to rest their hand, while the cursor holds its position.

Both [KWR06] and [SK04] have presented the case for designing a pen-based device. Both the papers have emphasized an ergonomic grip. [KWR06] described the grip to be good for both power and precision tasks. The paper also prescribed an ambidextrous form of grip, which makes the pen-based device neutral in terms of handedness. The grip should also account for the weight of the device. On the other hand, [SK04] suggested the pen-based device should have "aesthetic and coherent" form beside being workable and ergonomic. The paper stated a pen-like device requires a distinctive pointing nose, as one of the important findings. Both papers suggest a sufficient number of buttons for the device to have improved controllability.

2.3.1 Design Issues for Spatial Device

There are many design issues related to any spatial input device and many of which have been listed in [HPG94]. Here in this section we discuss a subset of these issues, which are more relevant and may have major impact on the rpen.

Firstly, there is an important notion in 3D interaction that a a single type of 3D interface/input device cannot be used to accomplish a wide range of 3D tasks. In fact, as described by Jacob and Sibert in [JS92], if two tasks in three dimensions are perceived to be different then they require two different input devices. So, it is important for us to identify the area where the rpen is applicable and design the device to accomplish that task. As the rpen is not (at present) to be used for virtual reality system or as a gesture based system, the design issues related to these applications, is not part of the rpen design considerations.

Input device state: It is important for an input device to produce all relevant events or states to support the full set of basic graphical user interactions. According to practitioners, an input device should support three possible states: out-of-range, tracking and dragging. It can be noted here that some of the popular interfaces like the touch screen and mouse do not support all of these three states [Hin08].

Fatigue: Fatigue is a major concern for spatial input devices. It is very difficult for such a device to avoid it completely, but enough care should be taken to reduce fatigue whenever possible. Fatigue affects user performance, user satisfaction and eventually may result in injury to the user [HPG94].

Weight: Weight is also an important design factor for spatial devices. Since these devices are operated in the air, weight will affect the usability of the device.

Clutching mechanism: Clutching mechanism is a software mode which allows the movement of the 3D device without effecting the cursor movement [HPG94]. As spatial devices can be tiring, it is important that they have some sort of clutching mechanism, so that when the device is in that mode, it is not tracked and the user can rest their arm. This mechanism is also important for the "out of range" [Hin08] input device state. It is also important to design the clutch mechanism properly. Poorly designed clutch mechanisms give rise to some of the most confusing and difficult problems of an input device [HPG94].

Multi-sensory feedback: Operating a spatial device the user can easily become confused and might loose control of the device. If there is no physical object to manipulate, it is important to provide the look and feel of virtual object manipulation. For a device like the rpen, visual feedback is imperative but not exclusive. Some combination of auditory feedback, tactile and force feedback will increase the user's level of feel for the control of the input device.

In [Zha98], Zhai proposed a general set of usability criteria for 6 DOF input devices which can impact the user's performance. These criteria should be addressed while designing the device. The six aspects of usability proposed by Zhai are as follows:

- Speed

- Accuracy
- Ease of learning
- Fatigue
- Coordination
- Device acquisition

Some of these aspects have already been mentioned. The concept of device acquisition is also discussed as it is similar to the clutch mechanism. According to Zhai, 6 DOF position control devices lack coordination as they are limited by the rotation and reach of the human hand.

2.4 Related Works

The kinds of input parameters and the types of motion tracking technologies we have considered for the rpen, are very unique for such a 3D input device. These motion trackers and the use of absolute position and orientation data are mostly utilized in virtual reality or augmented reality applications. Also, recent gaming systems are using such devices as their game control, mostly to interpret user gestures. In the following we present some of the recent endeavors and research studies related to 3D interaction.

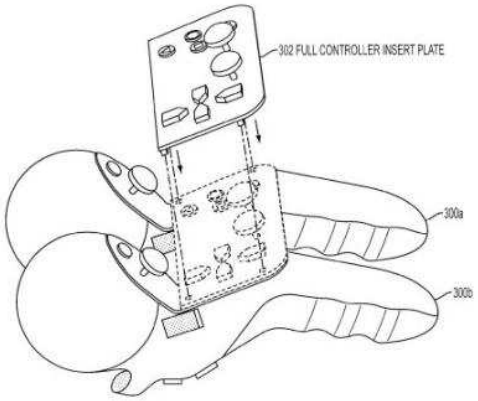


Figure 2.2: PlayStation3 3D motion controller.

Sony's PS3 motion controller: Sony Computer Entertainment is currently working on a new 3D motion tracking controller for its PlayStation3 game console. They have recently unveiled the first working model. The controller

is basically a pair of ice cream cone shaped devices. These two pieces can be used separately and also in a combined configuration. Figure 2.2 shows the PS3 controller, as presented in the submitted patent document (*Patent: US 2008/0261693 A1*).

The new PS3 controller is a hybrid motion tracker, as is the case with most of the current endeavors in this area. It uses inertial motion tracker for the three degree of rotation information. Even though, they have listed both gyroscope and accelerometer as possible options in the inertial tracker, they most likely are using the accelerometers due to cost considerations. The controller also uses LED-based optical tracker for the X and Y coordinate movements, where the x-axis positions represent the vertical movements and the y-axis positions are represented as the horizontal movements. An acoustic tracker is used to track the z-movements, i.e., back and forth movements from the display/screen. Ultrasound is also used to determine the distance between any two controllers. This overall combination of trackers, produces 6 DOF absolute motion data.

There are a few important points to be noted here:

- The main purpose of the new PS3 controller is to capture the user's gestures, which can be approximated. They thus have a higher tolerance level for noise which allows the resolution of the device be lowered.
- The controller is developed for games. The precision and accuracy requirements are not as high as it is for the rpen.
- The new controller is using a hybrid of three motion trackers to produce the 6 DOF motion data at each instance. The process involved to combine this data would be time consuming. So, to reduce time lag they likely are using some sort of stochastic process, such as a Kalman filter. This process is used to combine the multisensor data, filtering and motion prediction. The system uses the predicted data, and then the measured data (which is slower and comes at a later time) is used to refine the prediction. Stochastic processes (usually the Kalman filter) are mostly employed in virtual reality systems, to avoid "simulator sickness" [Wel09].

Apple's 3D remote control: Apple is also pursuing a new 3D remote motion sensing device, which is similar to the Nintendo's Wii Remote (discussed next). The device is still under development. Figure 2.3 shows a working sketch of the Apple Remote, as presented in the submitted patent document (*Patent: US 2008/0106517 A1*).

Similar to the Wii Remote, the Apple Remote Controller contains a photo sensor and accelerometer or gyroscope. Thus, the proposed controller is a hybrid of an optical motion tracker (used to detect the 3 DOF absolute position

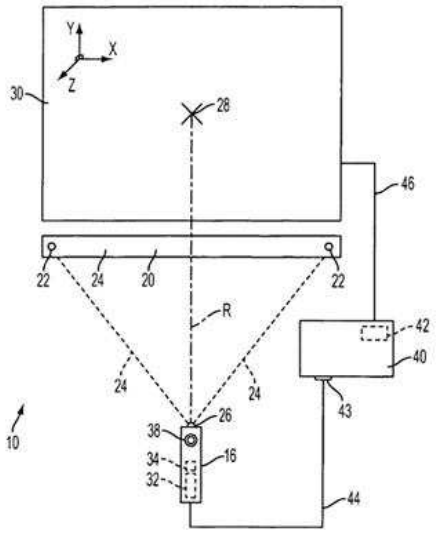


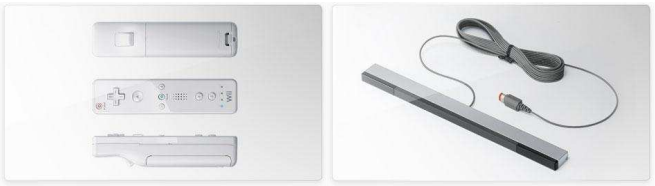
Figure 2.3: Apple's 3D remote.

information) and inertial tracker (to detect relative position on the z-axis). The patent document also suggests that gyroscopes or accelerometers could be used for multiple orthogonal axes for 3 DOF orientation data.

The uniqueness of the device, in comparison to the Wii Remote, is that it combines absolute position data with relative position information to get the average position information and as well as the current change in position (relative to last position) information. It is done only for the z-axis. This method will help the controller to provide a more precise zoom-in and zoom-out functionality.

Even though, the applicability of the device is yet to be fully known, it can be inferred to be used as a controller for a game or in the context of a virtual world. But whatever the application may be, Apple Remote may not have the precision, resolution and accuracy to function as an absolute pointing device like the rpen. It also is to be noted here that, Apple Remote, like its counterparts, Wii Remote and a PS3 controller, works as a distant pointing device. This means, unlike the rpen, the position that these controllers point to, is to be derived geometrically rather than taking the controller position as it is, such as is done in the rpen.

Wii Remote: Unlike the PS3 controller and the Apple Remote, the Wii Remote has been in the market since late 2005. Being the first of its kind, the Wii Remote has captivated game lovers with its free flowing motion capture and gesture recognition capabilities. The remote uses an infrared light source for its optical sensor, which tracks the 3D position information. The device also utilizes



(a) Wii remote (b) Sensor bar

Figure 2.4: Wii remote and its sensor bar [Wii09].

accelerometers to sense acceleration information. The orientation information is derived by the optical sensor, from the sensed relative angle of two dots of light with respect to the ground.

As a game controller it can be operated from as far away as ten metres. But as a pointer it can be as far as five metres. The location of the Sensor Bar is important for the remote to sense the position and orientation information properly. The bar must be centered in respect to the television and either it has to be in line with the front of the TV if placed above the TV, or it must be in line with the front of the surface that the TV is placed up on, if placed below the tv. Figure 2.4 shows the Wii Remote and the Sensor Bar.

The comments made for the Apple Remote apply similarly for the Wii Remote. A patent assigned to Hillcrest Laboratories Inc. for a similar 3D remote control device also exists. The only difference it seems is that the patented device from Hillcrest Laboratories is using accelerometers to provide orientation information as well as the acceleration on all orthogonal axes (*Patent: US 7158118 B2*).

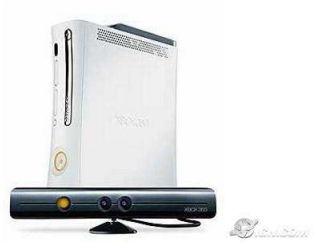


Figure 2.5: Project Natal's sensor device [Sen09].

Project Natal from Microsoft: Project Natal [Pro09b] is a new gaming interface from Microsoft. It is used with Microsoft's gaming console, the Xbox 360. Microsoft is pursuing a completely different 3D interaction mode than its competitors. While its competitors are pursuing 3D control based gaming interfaces, Microsoft is moving towards a "control-free gaming experience" [Pro09a].

Project Natal basically detects a user's whole body movements. It also has voice recognition capabilities, which can recognize a specific set of voice commands from multiple users. Figure 2.5 shows the Project Natal's sensor device.

A Project Natal system uses optical motion trackers to sense user's 3D motion. The optical motion tracker used for the system is not a single device/sensor, but rather, a combination of an 'RGB camera' and 'depth sensor'. It not only provides motion capture capabilities but also facial recognition features. The system also contains a multi-array microphone, which is used for voice recognition capabilities. The system has ambient noise suppression capabilities, and it can also be operated under ambient light conditions [Pro09a]. Project Natal is a gesture recognition system, which can extract 48 interest points on a human body at a frame rate of 30Hz.

While the technology is very impressive, it is not comparable to rpen technology, because they are applicable to two different application areas. The Project Natal is an interface for virtual reality and the rpen is a 3D pointing device. As the Project Natal system has to process data from various points of a human body at a rate of 30Hz, the processing time will be much longer than ideal. Also it uses two different optical systems for tracking purposes. So, as most virtual reality systems do, the Project Natal system must be using some sort of stochastic process to predict user's movements, to reduce the visual time lag. Also, as said for the PS3 controller, the requirements for a gesture-based system are very different from that of a pointing system.

Figure 2.6: Sixense's 3D wireless controller [Con09].

3D wireless controller from Sixense: Sixense Entertainment, a company from California, USA, developed a 6 DOF motion tracking controller (see figure 2.6) using an ultra low power electromagnetic (EM) sensor. As the EM sensor used is very low powered, it creates a very weak magnetic field. This might eliminate some of the metal related noise, but, it also means that the working volume could be very limited. There is not much information available about the device, particularly in relation to its technical specifications. The company's official website [Six09] also only contains information about some of the demonstrations involving the device. In any case, it is evident from the demonstrations

that the device is created as a game controller and not as a pointing device.

g-speak: Oblong Industries developed 'g-speak', a spatial operating environment. Multiple users can use the system at the same time. The system is capable of tracking two-handed gestures. The users interact with the system through free hand (without any object in hand) gestures. The system uses marker based data gloves with its optical tracker, capable of 6 DOF input data.

The system does distant freehand pointing, similar to the technique used in [VB05]. In [VB05], a ray casting technique is used to point at distant objects. The g-speak works with wall sized projection screens, desktop monitors, table-top screens and handheld devices [Obl09].

g-speak is a total 3D operating environment having powerful motion tracking capabilities. The system can work with multiple display systems, where display and work surface are the same. In contrast, the rpen is dependent on the primary screen for its resolution information but the work surface can be anywhere in the space with any orientation.

SixthSense: SixthSense is an augmented reality application, developed by Pranav Mistry, a PhD student in MIT Media Lab [MIT09]. The interface can interpret hand gestures. It uses a camera, which is an optical tracker, to track color coded fingers and the gestures the fingers make. It uses a tiny projector to project information on various surfaces, which can be then manipulated through hand gestures. The whole interface is small enough to be able to wear around one's neck.

Miscellaneous Devices: BurstGlove™[Mot09] is an optical tracker based absolute 6 DOF motion capturing device from Motion4u. It uses a marker based optical tracker. The tracker has a plugin for 3D animation software Maya®. Using the plugin, the tracker can be used to produce animation in 3D space very easily. It is said, the device can be used as a 3D mouse, i.e., the marker will be tracked to be translated to a 3D cursor position only on 1 to 1 basis.

[FS06] also presented an optical tracker based input device. The device uses a camera to sense the position of the emitter, which can be a light emitting point or a finger. 3D relative movements are derived from direct and reflective images of the pointer, i.e., the pointer is operated on a reflective mirror.

Summary: As can be seen from the above discussions, most of the motion tracking devices are either using hybrid tracker or optical tracker technologies. Even the hybrid trackers are using optical systems as one of the motion tracking devices; thus, these devices will have the same limitations that an optical tracker has, the most important of which is occlusion.

Also, almost all these devices are interpreting human gestures as input. Even though some of them could be used as an absolute position pointing device, none of these systems use work surface like the rpen. Most of the devices are confined

to either the display screen or the display system (with multiple displays) as their pointing surface as well as their work surface. In contrast, the rpen translates the cursor movements to the display screen and the work surface could be the same as the screen or different. Also, the work space can be placed anywhere, and at any angle.

CHAPTER 3

Methodology

3.1 Research Methodology

The aim of this research is to develop a working prototype of a new pointing device, '**rpen**', using an absolute position tracking device. The focus of the current study is not centered about any position tracking hardware or comparative study into different types of position tracking technologies; rather, to explore the possibilities of a new kind of pointing device and its usefulness.

The concept of rpen is independent of any position tracking hardware. However, the effectiveness and efficiency of rpen as a device will depend on the hardware being chosen for the prototype. An initial assumption is that a commercially available hardware device providing absolute position data should work with rpen without having to put too much focus on the hardware.

To demonstrate the usefulness of the new pointing device, physical experimentations have been carried out. The experiment involved the tracking of sensor movements to actual cursor movements and appropriate events. They have been done on different work surface resolutions. We will refer to an rpen work surface as an '**rframe**'. Through these experimentations and usability testing, effects of rframe resolution on rpen usage have also been analyzed in this thesis.

Through analysis of previous research and studies of related issues, decisions have been made related to the design and characteristics of rpen. Though hardware design and related experimentation are not part of the current study, design suggestions are made based on the literature reviews as well. Human factor issues are addressed using a combination of analysis and experimentations. Usability testing has been carried out with a limited number of users.

The choice of hardware and tracking technology has been made through analysis. Even though initially it was not intended to put effort in tracking technologies and related issues like noise filtering techniques and smoothing, due to the choice of hardware, a considerable amount of experimentation and data analysis had to be made on these issues.

The current research is a proof-of-concept; this had to involve technical development. No human computer interaction research is complete without experiments showing how the interface works with its human operator. Elaboration

of ideas and logical support for the research, are developed through analysis of theories and results already present in the current literature. The current study is a fusion of these methods.

3.2 Linux

The design decision to implement rpen on Linux comes with its advantages. Linux is a UNIX like open source operating system (OS). Like UNIX, Linux's simplistic design principles make it easy for the developers to understand how it works as well as use its relatively smaller number of APIs and system calls effectively. Linux is open source, meaning that the source code is free and available for developers and academics to learn from the code; test and play with modifications as desired.

Recent improvements and modifications made to the device model in Linux meant that implementing device drivers and interfacing with other kernel core components is easier, and simpler. In [Lov05] Robert Love said, In UNIX everything is a file. In fact, most of the data and device operations can be done using simple interfaces like, *open()*, *close()*, *read()*, *write()*, etc.

Another advantage with Linux is that it has an ample amount of documentation in writing (books) as well as on the web. The Linux source tree with its documentation works as a training guide with examples. One could view existing drivers and use them as the basis of his/her code. One could also modify the windowing system if needed, as well as exisiting application to add extra support, say for gestures, if one wished.

Rpen is conceived to be a general purpose device. It means it has to work with the OS, which in this case is Linux, and also with any applications installed on it. As it is not a special purpose device to meet specific requirements, like the tracking systems used for virtual reality systems, the choice for developing a device driver for rpen was obvious. The developed device driver works as a go between the hardware and Kernel, which in turn will communicate with other applications and user interfaces (see figure 3.1).

Even though, many of the devices can be operated utilizing the I/O scheme, hardware like the rpen, where timely service by the processor is important, utilizes the interrupt concept. It is an asynchronous communication between the kernel and the hardware. Here, whenever the hardware requires a communication with the system it issues an interrupt and the kernel, using the interrupt identification number, executes the specific interrupt handler. Interrupt handler runs in a privileged mode, called the interrupt mode. In this mode/context, the handler is solely allowed to run on the processor and quickly responds to the interrupt.

Linux is a monolithic kernel. It means it runs as a single large process in

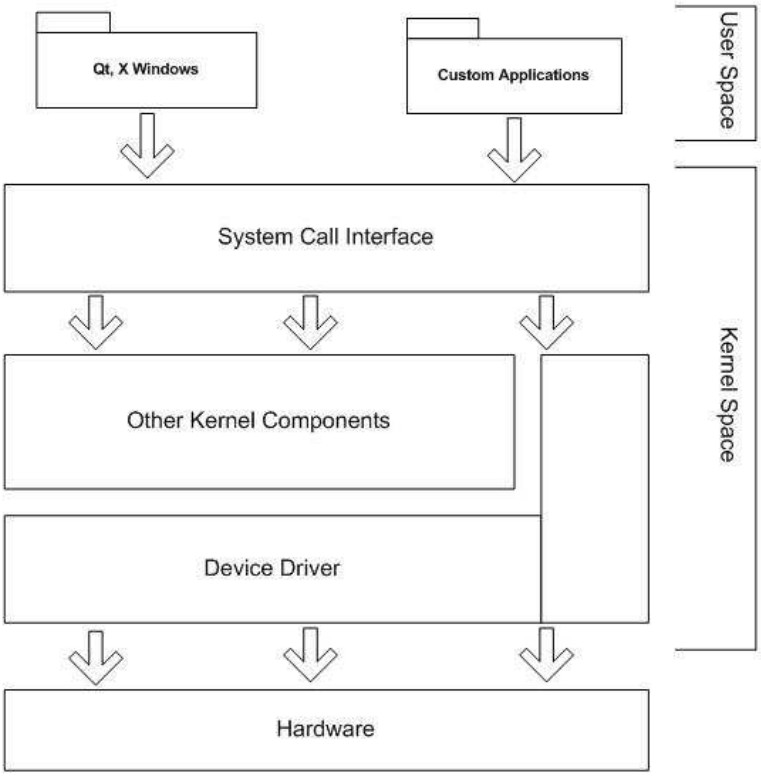


Figure 3.1: Relationship between device driver, kernel and other user space applications [Lov05].

a single address space, in Kernel mode. But it is also modular, i.e., it allows individual code segment to be inserted and removed from the kernel at run-time. Many of the Kernel components are developed as modules, including the device drivers. This means a device driver can be developed out of the source tree of the Linux kernel, totally independently, and also can be loaded whenever needed without having to shutdown the kernel. This gives the developer of a device driver the advantages to run a device in kernel mode as well as develop it without having to worry about the rest of the kernel.

Taking into account the above discussion, it was decided to opt for a device driver in the Linux kernel to operate rpen.

3.2.1 Linux Device Drivers

A major role of an operating system (OS) or kernel is to manage the hardware devices attached to the host system. The kernel uses device drivers to handle this task. These device drivers implement the intricacies of how a device operates as well as communicate and provide a common and simplified interface to applications. The interfaces provided by the device drivers are standardized system calls, similar to file operations. The architecture of how user space applications operate on devices can be understood from the Linux device model, which is discussed in the next section.

As mentioned earlier, device drivers are implemented as modules. In fact, the standardized interface structure specified above, enables them to be modularized. Each device driver, as such, is a compact piece of code having its functions, data, entry, and exit points grouped together. This modularity helps the Linux Kernel to have a minimal loadable image as well as enable devices to be hot pluggable. A module can be dynamically linked to a running kernel by calling '*insmod*' program, which initialize the module by calling the function identified by the *module_init()* macro, inside the module. Similarly, a module can be unlinked from a system dynamically by calling '*rmmod*' program which calls the function identified by the *module_exit()* macro.

Linux devices broadly can be classified into three fundamental device types:

- Character Device
- Block Device
- Network Device

It is possible to develop a module implementing different drivers for more than one of these device types, but this is not good for scalability and extendability.

Character Device: Character (char) devices are accessed as a stream of bytes and a character device driver implements this characteristic of the device. The console and serial ports are char devices. The char devices are assessed through '*/dev*' file system nodes. Character device drivers must at least implement the *open()*, *read()*, *write()*, and *close()* system calls. Most of the commonly known input devices are char devices.

Block Device: A Block Device can host a file system. In Linux, a block device is implemented similar to a char device, in terms of data access. It is also accessible via the '*/dev*' node directly. Other than being able to hold a file system, a block device differs with a char device in the way data is managed internally by the kernel. Hard disks and CDROM players are examples of block devices.

Network Devices: Any device that is able to exchange data with another host is a network device, more appropriately a network interface. These interfaces are not at all similar to char or block devices. They are not handled through '*/dev*' but rather given unique names like *eth0*. The network drivers handle data packets only.

There are other ways to classify the device drivers. One is the kind of bus they use. Thus, a driver could be a USB or serial or SCSI or PCI module. Another major classification is made on the type of interaction the device does, like an input device. A device will fall under more than one of these classifications. For example, rpen is conceived to be an interrupt based, USB, input device, which falls under the broad category of char device. But, because of the hardware choice for our prototype (the Wintracker II discussed in section 3.3), we decided to implement it as a poll based, USB, input device, utilizing bulk data communication usually used for block devices.

3.2.1.1 Interrupts

As mentioned earlier, most of the hardware peripherals attached to a computer use the interrupt concept for its I/O. The reason is that a processor usually can process its tasks much faster than a hardware peripheral with which it works. Thus, rather than waste processor time by waiting for a hardware to respond, it is better that the hardware let the processor know when it needs the service of the processor. It not only saves the processor's time from being wasted but it is also the fastest way to service the hardware.

An interrupt is an electronic signal produced by the hardware attached to the computer. It means the hardware has to be configured to work in interrupt mode. The interrupt signal is received by the interrupt controller. The signal is then passed on to the processor by the controller. After the processor detects the signal, it interrupts the currently executing process to handle the interrupt. The

processor notifies the kernel of the interrupt and the kernel finds the appropriate handler with which to handle the interrupt. A interrupt handling process is depicted in figure 3.2.

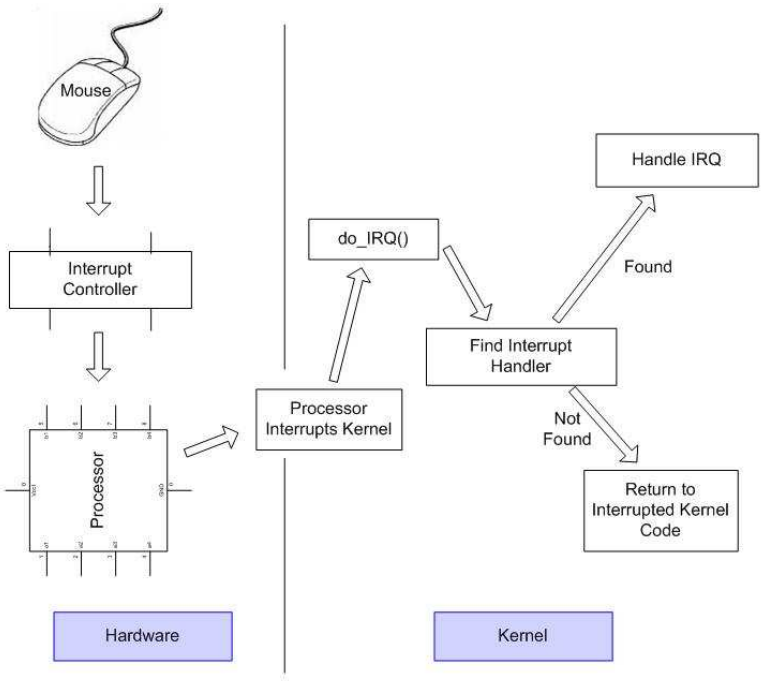


Figure 3.2: Interrupt handling process [Lov05].

Each interrupt is given a unique number, either hard coded or dynamically allocated. This number is called the Interrupt Request (IRQ) number or line. Through IRQ number, the kernel can differentiate between different interrupts. An interrupt handler is associated with an interrupt using this IRQ number.

Another important aspect of the interrupt is that it runs in an interrupt context. A Linux kernel executes its tasks either in process context or interrupt context. In process context, a process can go to sleep or invoke the scheduler. In interrupt context it is not allowed to sleep and it cannot be preempted by the scheduler. As such, an interrupt handler is able to execute its tasks without any pause, stopping other processes running on the processor. So, it is important that the interrupt handler does not use too much time or memory.

The interrupt handler is a part of the device driver. The request for an IRQ number as well as associating it with a handler is done as part of the initialization of a device driver.

As mentioned, rpen is conceived to be an interrupt based input device. Thus,

```
/sys/
|--  block
|--  bus
|--  class
|--  dev
|--  devices
|--  firmware
|--  fs
|--  kernel
|--  module
\--  power
```

Figure 3.3: Top level sysfs directories.

the device driver is designed with an interrupt handler function in mind. However, the chosen hardware for our prototype does not have interrupt capacity. The current device driver rather employs a mechanism to poll/request the hardware device for data at defined regular intervals which is a process based solution rather than an interrupt based one. The initial interrupt handler function design was modified to account for the change. So, in future, if rpen is implemented with interrupt based hardware, small modifications in the device driver should be sufficient.

It also should be mentioned here that an input device such as rpen should always be interrupt based for the effective interactive experience with the system. The faster hardware manipulation (movement) is accurately reflected by the system, the better satisfied the user will be.

3.2.1.2 Linux Device Model

It is important here to discuss the 'Linux Device Model' to gain insight into the relationship between the device, the device driver, kernel, and the user space. It is also very important for rpen, as the basic concept of the device is implemented by taking advantage of the model. The rframe and other basic parameters for rpen are defined through an application in user space. Using the nodes presented through the device model, these parameters are communicated to the device driver. The device driver, through the device model nodes, not only communicates with the user space applications about the devices generated events, but also provides device driver specific data to other rpen applications in the user space; thus, Linux device model is an important part of the rpen design.

The new Linux device model, implemented in kernel version 2.6, is a complicated data structure. It is designed to implement many important tasks/features

Table 3.1: Sysfs functions/macros used in rpen device driver.

Function/Macro	Purpose
int sysfs_create_group()	This function creates an attribute group under the given kobject.
void sysfs_remove_group()	This function removes the group all together.
DEVICE_ATTR()	This macro helps define a device attribute.
ssize_t rpenParam_show()	This function writes to sysfs node.
ssize_t rpenParam_store()	This function reads from sysfs node.

including communication with user space, hotpluggable devices, device classes, device lifecycle etc. Even though it is very important to the devices, in most cases, a device driver does not need to explicitly deal with the device model. Similarly for the hotpluggable device mechanism for rpen, the USB and input subsystems assign the major and minor numbers for the device to be uniquely identifiable to the system and export this information to the user space through the sysfs node. This information is then used by the user space daemon called '*udev*' to dynamically create a '*/dev*' node for rpen. Through this node, rpen reports the events generated by the device for user space applications to respond to, accordingly. In the case of rpen the sysfs nodes are manipulated since it was necessary to feed in and out some basic parameters to the device driver, to allow appropriate behavior of the device.

'*Sysfs*' is the method through which the device model framework is implemented. Sysfs is basically a file system. The top level sysfs directories are shown in figure 3.3. Each active device in the system has sysfs entries, which represent the device model, i.e., information regarding its bus, device, device driver, and class. The files created here are typically ASCII files with usually a single value per file. These files are also known as attributes. Most of these attributes are default values and contain information on the four layers of the device model, as mentioned previously.

At each of the four layers of the device model, sysfs provides an interface for the addition of attributes. However, most of the device drivers define either device or driver level attributes. In the case of rpen, we have implemented device level attributes. The attributes thus appear under the '*/device*' directory. As we had to define multiple attributes, we implemented them as a group, and as such, they can easily be added or removed with a single function call. Table 3.1 lists the functions/macros used by the rpen device driver.

3.2.1.3 User Space vs Kernel Space Device Driver

Rpen is implemented as a kernel space device driver. In fact, most of the device drivers in Linux are in kernel space, although Linux does provide the option to implement a user space device driver. A driver developer needs to consider the characteristics of the device, s/he wants to operate as well as advantages and disadvantages of developing the driver in user space.

In Linux, not all devices are supported for a user space device driver. USB devices are supported through a component like '*libusb*'. The device driver for rpen using a USB hardware device (Wintracker II) could have been developed for user space, but, the characteristics of rpen make a kernel space driver more desirable.

Rpen, being an input pointing device, is conceived to be an interrupt based device. Also it needs fast responses from the kernel. If it employed a user space device driver, this can not be guaranteed. Contexts switches, process privilege issues, risk of then being put to sleep and not getting locks in time, make user space device drivers slow to respond and thus unsuitable.

On the other hand, there are advantages of a user space device driver. A user space device driver can use the full C library. Code debugging is also easier. Most importantly in the context of rpen, a device driver in user space can readily use floating point operations which is not possible with the C library in kernel space. In the current implementation of rpen, the device driver needs floating point calculations in several places. The most critical places are the filter and position transformation for a generic rframe. We examined the roundup error effects of the current device driver, in this study. Also, the implemented filter is specific to our current choice of hardware. As the filters are usually implemented in hardware (if required), it will most likely not be an issue for future versions of rpen.

3.2.1.4 USB Device Drivers

The universal serial bus (USB) is a channel between a host system and its peripherals. Structurally, the USB is not a bus but rather a tree topology, and thus has hubs. Devices and other hubs can be connected to a hub. A hardware known as a USB host controller implements such a hub inside any host system and this is the first root hub. In Linux all the USB devices are controlled by the kernels USB subsystem. The USB subsystem comprises of [Ven08]:

1. The USB core
2. USB host controller drivers
3. Hub driver

4. Khubd - a helper Kernel thread
5. USB device drivers
6. USB file system (usbfs)

A USB device is made up of configurations, interfaces and endpoints (see figure 3.4). A USB device may have one or more configurations but only one can be enabled at any particular time. Each configuration often has one or more interfaces. A USB device driver can control one interface. Thus any device having multiple interfaces will require multiple drivers to operate that device. USB interfaces may also have alternate settings, which are basically different choices for parameters for the interface.

Endpoints are the most basic form of communication in USB devices. There could be zero or more endpoints in an interface, with a maximum of 128 endpoints. Endpoints can carry data in one direction only; from the host to device or from device to host. An endpoint (for a specific implementation) can handle either of the following four data transfer types:

1. Control transfers - used for configuration and control information
2. Bulk transfers - transfer large quantities of data
3. Interrupt transfers - exchange small quantities of time sensitive data
4. Isochronous transfers - for data at predictable rates

Each channel attached to an endpoint is called a pipe and has an integer encoding. It is a combination of endpoint address, direction of data transfer and type of data transfer. To create this pipe, the USB core has provided the following macro:

```
usb_[rcv/snd][ctrl/int/bulk/isoc]pipe(struct usb_device *udev,
__u8 endpointaddress);
```

The rpen device driver has two pipes. One for outgoing commands and other is for incoming sensor data. To create these two pipes, we used the above macro as following:

```
in_pipe = usb_rcvbulkpipe(usbdev, in_endpointAddress);
out_pipe = usb_subbulkpipe(usbdev, out_endpointAddress);
```

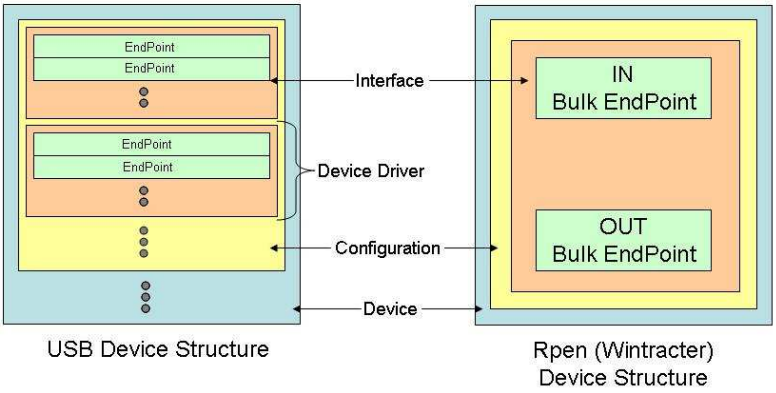


Figure 3.4: Structure of a generic USB device along with the structure of Wintracker, that is used for rpen.

URB (USB request block) is the usual means of communication between the USB core and a USB device driver like the rpen driver. URB has a life cycle. Data communication using URB require some preparation. But, there is a way around this for simple data transfers without explicitly implementing the URB. The USB core provides two functions for this purpose. One is for bulk data transfer and the other is for control data transfer.

The hardware device, Wintracker II, used with the current version of rpen only supports bulk data transfer both for the position data it sends to the host system and for receiving the control commands it gets from the host system. Also, the data length for both the incoming data and outgoing data is 32 bytes. As the Wintracker system is not interrupt based, it uses bulk data transfers, and the data it communicates is simple and concise, we decided to use the bulk data transfer functions below without using the URB concept.

```
int usb_bulk_msg (struct usb_device *usbdev, unsigned int pipe,
void *data, int len, int *actual_length, int timeout);
```

USB Device Driver Structure :

The USB device driver needs to specify one or more USB devices it will support. This can be done using the macro:

```
USB_DVICE (vendorID, productID)
```

There are other versions of this macro, but this version is the most widely used. This information is later used by the hotplug scripts, in user space, to

automatically load the device driver when a device it supports is plugged in to the system.

After specifying the device, the driver has to create the driver structure with at least its name, *probe()* function, *disconnect()* function and the *device_id* table. The driver name has to be unique. The *device_id* table has references to all the USB devices that the driver supports. This table is created with the help of the above *USB_DEVICE* macro. This table is very important as without it the driver is meaningless. The *probe()* function is called by the USB core when it gets a new USB interface which this driver is supposed to handle. The *probe()* function, if successful, claims the device or the interface and initializes the driver to handle the device or interface. It also initializes the driver for use of the device by other applications or kernel components. The *disconnect()* function is called when the interface goes out of the system or the driver is being uninstalled/removed from the system. Thus, the *disconnect()* function needs to dereference/unregister the objects that the *probe()* function has referenced/registered.

Lastly, the USB driver has to register the USB driver structure it created. This is done in the module initialization function called by the *module_init()* function mentioned earlier in section 3.2.1. To register a USB driver, a call to *usb_register()* is made with a pointer to the USB driver structure. Similarly, to unload the driver from the system the USB driver structure has to be unregistered using the *usb_deregister()* function. This function is called from within the function identified by the *module_exit()* macro.

3.2.1.5 Input Device Drivers

Rpen is an input device. It thus needs to register itself as an input device as well as produce input events so that applications in user space can interpret the events correctly and take appropriate actions. Input devices in Linux, starting with version 2.4, are integrated to provide a common interface to user space applications. The Linux input subsystem, primarily the work of Vojtech Pavlik, includes the input device drivers, input event drivers and input core. It sits between the various hardware interfaces (PS/2, serial, USB, etc.) and user space. Due to the input subsystem, any user input from the hardware can be presented to user space in a consistent and device independent way through a range of pre-defined APIs.

As shown in figure 3.5, input events are not part of the input device driver. The input core provides a common set of APIs to the input device drivers to report the events. The device driver must know what kind of events it is getting from the hardware device and how. The device driver then just records these events using the input core APIs ('*/linux/input.h*'). The input event drivers, through '*/dev*' nodes, report these events to the user space applications. In the

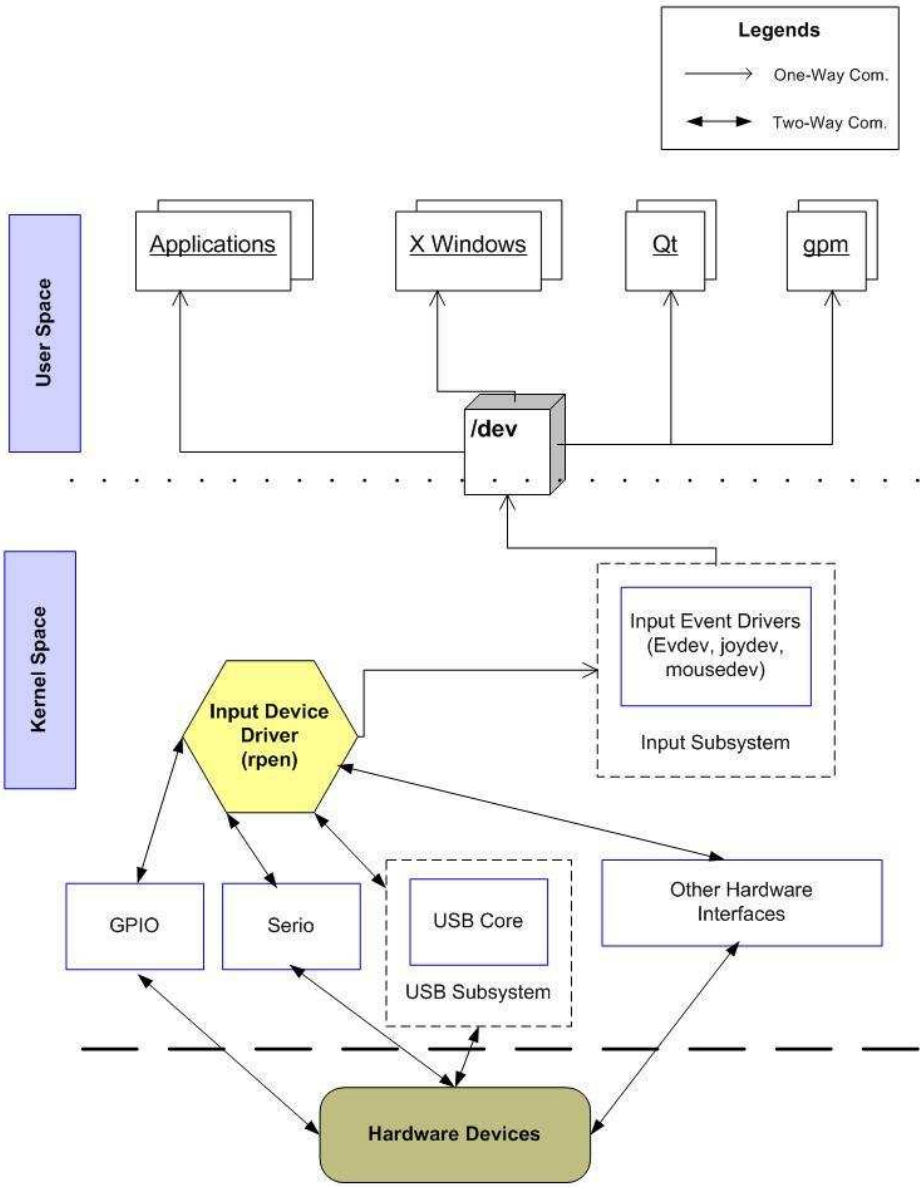


Figure 3.5: Linux input subsystem; adapted from [Ven08].

Table 3.2: Input driver structure used inside the rpen device driver.

Stage	Function/Attribute	Purpose
Initialization	input_allocate_polled_device()	This function allocates an input poll device.
	set_bit()	This function sets the events to be reported. Rpen reports two types of events: EV_KEY and EV_ABS.
	input_set_abs_params()	This function is to set min and max values for the reported postion and orientation.
	poll_interval, poll	At this stage, the poll intervel is set and also the poll function is linked to the poll_device.
Register	input_register_polled_device()	This registers the poll input device.
Report Events	input_report_key()	This function is used to report button click events. Both this and the following function are called from within the poll function.
	input_report_abs()	Through this function absolute position and orientation information is reported.
Unregister	input_unregister_polled_device()	This unregisters the poll input device.

case of rpen, it is using '*evdev*' as its input event driver.

As mentioned earlier, rpen is built on a hardware device which does not support interrupts and for that reason we need to poll the device for sensor information at predefined regular intervals. This mechanism is implemented through a wrapper class of the regular input class defined inside the Linux kernel. This class is called *input_polldev*. It helps to define a simple polled input device. The developer has the facility to provide *poll_interval*, in milliseconds. The regular input_dev structure is a member of the *input_polled_dev* structure. Also, for input device allocation, registering the input device and unregistering the device, the *poll_dev* class has its own wrapper functions. Other than this, the structure of a regular input device driver is similar to the rpen input device driver. The structure of rpen as an input device driver is given in table 3.2.

3.3 Wintracker

The **Wintracker II** developed by VR-Space Inc. [Win09], Taiwan, is the chosen tracking system to work with the current version of rpen. Wintracker is an alternating current (AC) electromagnetic tracking system. The device comes with three sensors/receivers, but for the purpose of rpen we require only one sensor. The sensors provide 6DOF (degrees of freedom) absolute tracking information, containing position information with regards to 3D coordinates (XYZ) as well as the rotation information about these three perpendicular axes (pitch, roll, and yaw). Wintracker comes with a USB interface for high speed data transfer with the host system. With the regular transmitter that comes with Wintracker, the system covers an area up to 75cm. With the Range Extender, not included in the regular version, area coverage can be extended up to 2.75 meters [VR 07]. Figure 3.6 shows the Wintracker system with its Range Extender.

(a) Wintracker II (b) Range extender

Figure 3.6: WintrackerII and range extender [Win09].

Analysis given on position tracking technologies in chapter 2 is the basis of our decision to select an electromagnetic tracking system. Our assumption

was, an off-the-shelf commercial electromagnetic tracking system will provide sufficiently accurate position information for it to be implemented as a pointing device. The assumption also included that much of the metals and electromagnetic distortion effect could be avoided by removing unnecessary metal and electric equipments from any surrounding working areas. Also, we assumed that 'Smoothing' would help to reduce the effects of any other noise and inaccuracy, if required.

VR-Space Inc. is a relatively new company to provide AC electromagnetic tracking systems. There are two major companies that sell electromagnetic tracking systems. For AC electromagnetic tracking systems, Polhemus [Inc09e] is the leading manufacturer and Ascension Technology Corporation [Cor09] is the leader for DC electromagnetic tracking systems. '*Fastrak*' from Polhemus and '*Flock of Birds*' from Ascension are more widely used electromagnetic tracking systems than Wintracker. But, as our current study is a proof-of-concept we opted for a cost-conscious choice. According to Inition [Ini09], a company that provides cutting-edge 3D expertise and technology, Wintracker is comparable to the performance of Fastrak by Polhemus.

3.3.1 Specifications

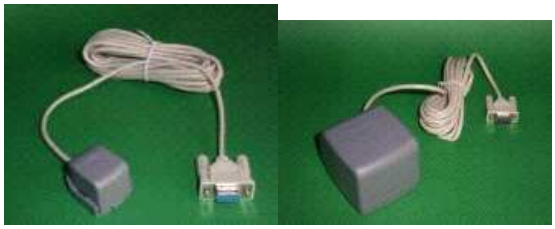
Position coverage: According to the manual of Wintracker II [VR 07], the distance between the transmitter (the base) and the receiver (the sensor) can be at most 75cm. It is also mentioned that at the range 75cm to 1.5m the sensor can be tracked with reduced accuracy. However, in our tests of the system, we have observed that the measurement of the position information beyond 75cm is unreliable. We thus consider invalid, in the rpen device driver, any position value over 74cm. Also, the sign of the position information is often observed showing inaccurate magnitudes and at times fluctuates between '+' and '−'. To get rid of this unwanted complexity, we are only considering the positive position values for rpen. This means rpen can not differentiate between different coordinate sectors of the XYZ space, i.e., sensor position is only measurable in the coordinate sector where x, y, z values are positive.

Angular coverage: Wintracker provides all angular attitudes. The value range for roll, pitch and yaw is $+179°$to $-180°$.

Other specifications:

Static Accuracy (RMS) Position (cm)	0.1524
Static Accuracy (RMS) Orientaion °	0.3
Resolution (RMS) Position (cm)	0.01
Resolution (RMS) Orientaion °	0.01
Latency (ms)	11
Update Rate (per sec)	90 (1)
Interface	USB
Operating Temparature (°C)	10-40
Operating Platforms	Windows and Linux
Sensor Dimensions (cm)	3.75x2.54x2.54
Sensor Cable Length (m)	4.5
Transmitter Dimensions (cm)	5.5x5.5x5.8
Transmitter Cable Length (m)	3.5
Carrier Frequency (Khz)	10.0174
Power Requirements (V)	85 - 264

Figure 3.7 shows the sensor and transmitter of Wintracker II.



(a) Wintracker sensor (b) Wintracker transmitter

Figure 3.7: Wintracker sensor and transmitter [Ini09].

Operating environment: According to the Wintracker manual [VR 07] the presence of large metallic objects will degrade the performance, in other words the accuracy, of the tracking system. In fact, all sizes of metallic objects can degrade the accuracy of the sensor position. [NMF98] has shown that for AC electromagnetic tracking system like Wintracker, the error in the sensor position is proportional to the size of the metal. It is also suggested in the manual that significant presence of metal in walls, floors or ceilings surrounding the Wintracker will degrade its performance as well. As suggested in [NMF98] as well as the manual [VR 07] the only remedial action is to keep metals and other electromagnetic field modifying objects three times further than the distance between the transmitter and the receiver.

3.3.2 Interface

Wintracker sends and receives 32 bytes of data through its USB interface. The format of the data that it sends to the host system is as follows (figure 3.8):

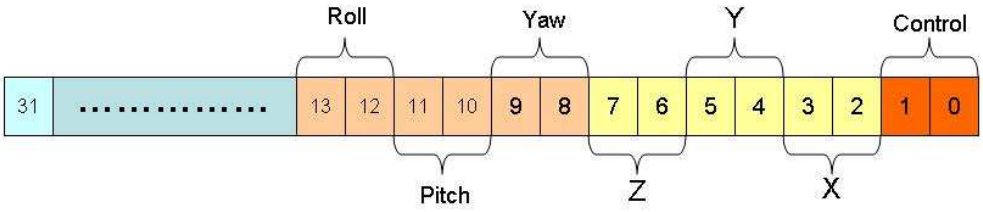


Figure 3.8: Format of Wintracker data sent to host system.

All the position information and the angle information are sent in two bytes, i.e., as short integers. The order of the bytes is first the low order byte and then the higher order byte, following the little endian format. To obtain the actual value of any position or orientation data, it is necessary to perform the following operation in the device driver:

```
__s16  value = data[i+1]*256 + data[i];
here, i is the byte position of low order byte.
```

To send a command to operate the Wintracker system, the host system will have to provide the same length of 32 byte data containing the command. Each byte will contain a character of the command starting with byte 0. '$\backslash 0$' character signifies the end of command and this will be placed in the byte after the last character of the command. Table 3.3 shows the list of commands used by the rpen. Following is a code snippet from the rpen device driver, where we are sending a command to the Wintracker hardware to enable the first sensor and disable the other two sensors:

```
send_buf_com = kmalloc(myMouse->dataLength, GFP_KERNEL);
send_buf_com[0] = 'S';
send_buf_com[1] = 'A';
send_buf_com[2] = '1';
send_buf_com[3] = '0';
send_buf_com[4] = '0';
send_buf_com[5] = '\0';
retVal = usb_bulk_msg(usbdev, myMouse->out_pipe, send_buf_com,
    myMouse->dataLength, &bytes_written, 10000);
```

Table 3.3: List of Wintracker commands used in the rpen device driver.

Command	Purpose
SP\0	This command requests one data record from the sensor.
SA100\0	This commad enables only the first sensor and disables the other two.
ST0,90,0,0\0	This command modifies the coordinate frame of the transmitter to 0°relative to the sensor's 90°on the Z-axis.

3.4 Filtering and Calibration Techniques

As electromagnetic tracking devices are dependent on electromagnetic fields, they are influenced by metals and objects that are capable of generating or modifying electromagnetic fields. The amount of influence/distortion/noise will depend on the following factors:

- The type of electromagnetic device - altering current (AC) or direct current (DC). As discussed in the chapter 2, AC and DC electromagnetic devices react to the ambient electromagnetic fields and metals differently.
- Distance between these objects and both the transmitter and receiver.
- Size of metal. Here size refers to the area of the metal not the volume, i.e., "What the transmitter 'sees' and what the receiver 'sees' " [NMF98].
- Distance between the transmitter and receiver.

In an office environment the source of noise for the electromagnetic tracking device could be many things, from computer monitors, fluorescent lighting, powered-on electric devices and wiring, metal in tables and chairs, metal in the nearby cubical or walls as well as the metal and hot wires in the floor and ceiling. It should be mentioned that the wires connecting the receiver and the transmitter also create noise. These noises are classified as static noise or dynamic noise. The noise that changes over time is called dynamic noise, whereas, the noise which is constant is termed static noise. Dynamic noise is caused by external electromagnetic fields and static noise is the result of metal interference. Manipulating sampling rate and filtering, dynamic noise can be reduced and this is usually done by the tracking system itself. But the static noise is dependent on the location where the system is being used [Kin00].

Outside the tracking system, calibration techniques are mainly used to compensate for the measurement errors for both the position and orientation. These calibration techniques can broadly be divided into two categories.

1. Function Fit: The idea here is to derive a function which can estimate the distortion. This function then can be used to find errors at any given position within the permissible range. The output of this function is then used to obtain the correct position and orientation.

2. Lookup Table (LUT): A metric of equally spaced error correction factors of the working location is created. Then the position correction is performed by trilinear interpolation of the error correction vectors. Orientation correction can be performed by the interpolation of the quaternion (quaternions are complex numbers on four-dimensional vector space, usually used to represent three-dimensional rotations) errors.

All the calibration methods are required to produce some sort of calibration table or metric. The table is formed from measured positions and true positions. Most of the known techniques use a *'rectilinear grid'* (see figure 3.9) to measure these positions and orientations [Kin00]. In some cases the *'true values'* are taken with the help of a second tracker system, mainly mechanical. [LS97] used a mechanical tracker named Faro Metrecom IND-1.

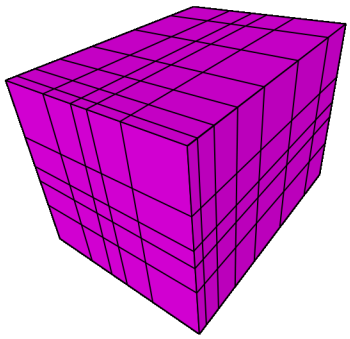


Figure 3.9: Rectilinear grid
(Source: Wikipedia, http://en.wikipedia.org/wiki/File:Rectilinear_grid.svg).

Both of the discussed calibration techniques have achieved relatively high level of error correction quality. [IBH01] claims that with a high order polynomial fit function (eg. 4th order) they have achieved error correction of more than 80% for both position and orientation data. Livingston in his paper [LS97] claims to have reduced the positional errors by 80% using LUT technique.

On the other hand, a function fit technique may not be able to find a good approximation to the functional representation of all the available distortions [LS97][Bac00]. "One might have to sacrifice local accuracy in a part of the working volume in order to obtain better global accuracy"[LS97]. For the LUT technique, the success is very much dependent on the '*granularity*' of the error correction

table [Kin99][IBH01]. A survey of a wide range of calibration techniques used with electromagnetic tracking systems is presented in [Kin00].

There are a few important points that should be noted in these calibration techniques, in the context of rpen. First, it is assumed that the electromagnetic characteristics within the working area will remain unchanged. Second, the importance of forming the truth table with the help of actual values, that eventually forms the function fit or LUT. Lastly, and most importantly, it means the rpen can not be used outside the calibrated work area. Also, this calibration process is not simple and can only be carried out by the person who is knowledgeable in this area. We perceive rpen to be used by regular users not necessarily knowledgeable in such calibration techniques.

Looking at this limitation of the calibration techniques, we opted to use classic noise filtering techniques along with a few other smoothing mechanisms, such as averaging. We also examined the sensor jitter, i.e., when stationary what is the average deviation. Using the Labview measurement tool [Nat09], we examined the power spectral density of the unfiltered data from the tracking device, Wintracker II. We have also employed various filtering techniques on the same dataset to arrive at a specific filter and specific filtering frequency. The test setup and some relevant results are presented in chapter 4.

A classic lowpass filter is used in the current version of rpen. From the unfiltered data coming from Wintracker, it is evident that most of the unexpected data is high frequency noise. This noise is causing erratic jumpy cursor movements on the screen. To reduce the noise level, we employed a Butterworth lowpass filter. What the filter does is, filter out the high frequencies from the input signal and let the low frequency signal pass unaltered. The critical factor is to determine the cut off frequency. This is the frequency beyond which all the frequencies needs to be filtered out. The power spectral density of the unfiltered data helped us to find this cutoff frequency.

After initial experimentation with the Labview measurement tool, we employed the following calculations derived from [SK88] to determine the desired lowpass filter equation to be implemented in the device driver. We supplied the sampling rate of the system and the required cutoff frequency to find out the desired lowpass filter. The only design constant for the lowpass filter is α, where

$$\alpha = \frac{sin(\theta_c/2 - \theta'_c/2)}{sin(\theta_c/2 + \theta'_c/2)} \tag{3.1}$$

For the Butterworth lowpass digital prototype, θ_c represents the prototype's critical or cutoff frequency (ideal cutoff frequency), which is $\pi/2$ [SK88]. θ'_c is the corresponding critical or cutoff frequency for the desired lowpass filter. We can

get θ'_c as follows:

$$\theta'_c = \frac{\omega_c}{f_s} \text{ where, } \omega_c = 2\pi f_c \tag{3.2}$$

Here, f_s = sample rate or sampling frequency, and f_c = cutoff frequency.

To find out desired lowpass filter, we used a second order lowpass prototype as follows:

$$H(z) = \frac{0.293(z+1)^2}{z^2 + 0.173} \text{ where, } z = \frac{z-\alpha}{1-\alpha z} \tag{3.3}$$

The general representation of the solution of equation (3.3) is as follows:

$$\frac{X(z)}{U(z)} = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{1 + a_1 z^{-1} + a_2 z^{-2}} \tag{3.4}$$

where, $X(z)$ is the z-transformation of output sequence and $U(z)$ is the z-transformation of the input sequence. Details on z-transform can be found in [Apl00] and [SK88].

Now, by performing the inverse z-transformation on equation (3.4) we will obtain an equation in terms of the regular input sequence and output sequence. The resultant solution gives us

$$x(k) = b_0 u(k) + b_1 u(k-1) + b_2 u(k-2) - a_1 x(k-1) - a_2 x(k-2) \tag{3.5}$$

where, a_1, a_2, b_0, b_1 and b_2 are all constants and k = current time. Thus, in terms of rpen:

$u(k)$ = current sensor input
$u(k-1)$ = last sensor input
$u(k-2)$ = prior to last sensor input
$x(k)$ = current filter output
$x(k-1)$ = last filter output
$x(k-2)$ = prior to last filter output

Equation (3.5) is implemented as a filter inside the rpen device driver. As mentioned earlier, specific cutoff frequency, its effect on the filter constants and lastly on the sensor data is explained in chapter 4.

3.5 Implementation Design of Rpen

The design issues regarding the Human Computer Interaction (HCI) of rpen was already addressed in chapter 2. There we have addressed why rpen employs the rframe concept, which resembles the touch screen like interaction concept. We have also suggested a stylus based hardware for rpen. Other human factor issues are also raised and addressed in the context of rpen. Here in this section,

we will be presenting the implementation design of rpen, i.e., how the device is implemented, what are the different components of which it is composed and their interactions.

Before going into the details of how rpen is implemented, the design features of rpen as a 3D absolute pointing device are presented below:

1. Sensor movement in 3D space is transformed into screen cursor movements.
2. User interacts with the system through a defined work surface which is referred to as the rframe. The rframe is a representation of the display screen, at an arbitrary position on 3D space.
3. The rframe can be of any size and orientation. Also, it can be configured anywhere on 3D space.
4. The user can define the rframe.
5. The user is able to produce left click and right click events.
6. For ease of operation, rpen has an auto adjust mode. In this mode, if the rpen is not active for certain period of time, it will automatically auto adjust its rframe to a new location defined by the user, the next time that the user wants to start using the rpen.

3.5.1 High Level Design

High level design of the rpen is given in figure 3.10. Wintracker II communicates with the host system through the USB interface. As discussed in section 3.2.1.4, the rpen device driver controls the device through the interface provided by the USB core. The rpen device driver also communicates with the user space applications through sysfs nodes maintained through the USB subsystem. As the rpen is an input device, it reports the input events through the input event driver provided by the input subsystem of Linux. The input events are propagated to the user space applications (X Windows and others) through the *'/dev'* nodes, as specified in section 3.2.1.5.

The user can define the rpen work surface, rframe, with the help of *'rframeDefApp'* application, discussed below. Rframe and other related information is fed to the rpen device driver through the sysfs nodes. This information is required by the device driver to transform the input data of Wintracker to a cursor position on the display. To define the rframe, display information is required. The rframeDefApp gets this information from X Windows.

There are two other daemon processes that work with rpen. Both these processes communicate with rpen through sysfs, and also communicate with X

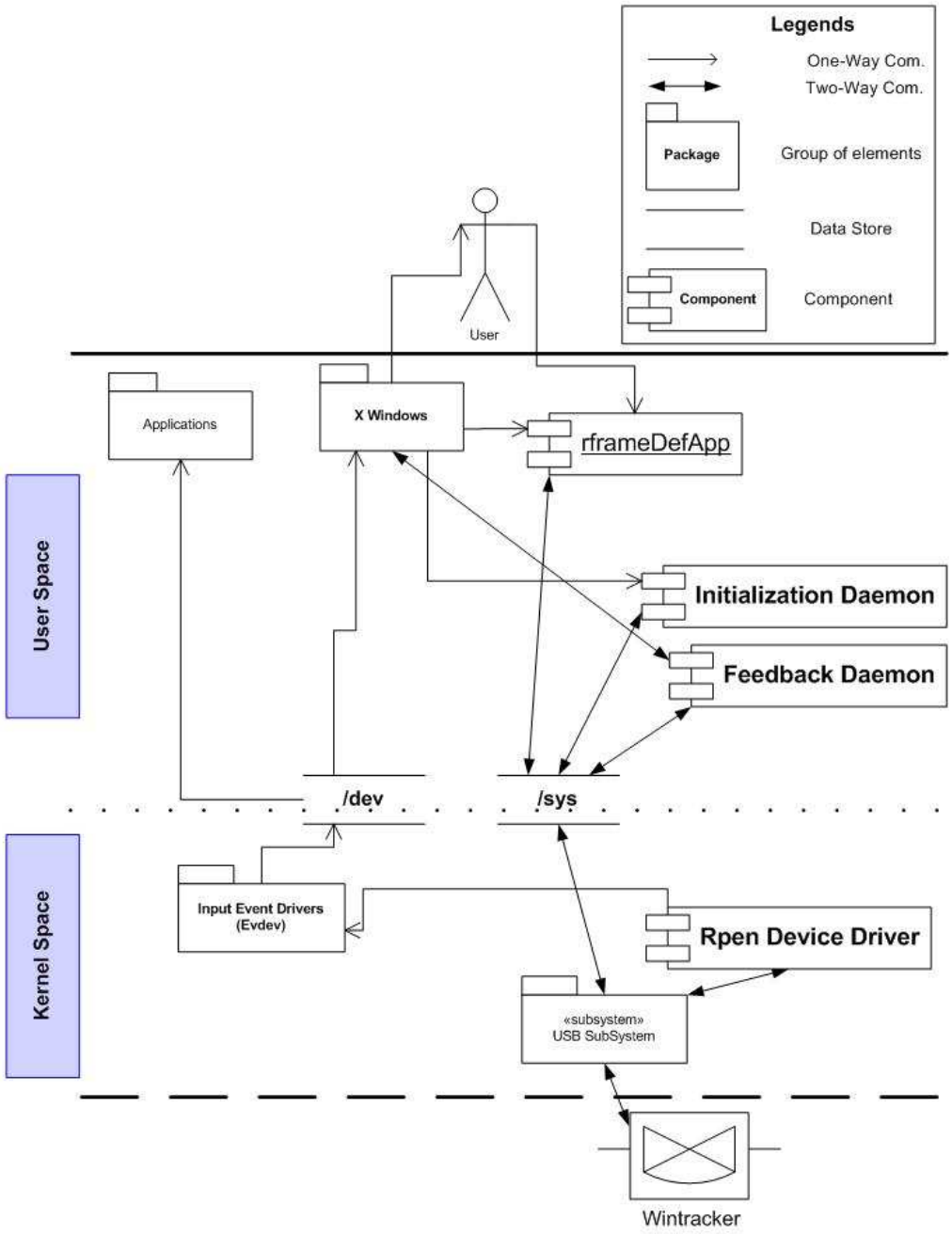


Figure 3.10: High level rpen design.

Windows. An initialization daemon is defined to perform two tasks. One is to update the sysfs nodes with the rframe data being used. The other is to update the rframe automatically when the screen resolution, more specifically, the aspect ratio of the display is changed. Similarly, a feedback daemon is defined to perform two tasks. One is to use the cursor appearance to provide visual feedback of the current sensor position with respect to the rframe. It does this through X Windows. It also produces audio feedback. Another job of the feedback daemon is to facilitate the auto adjust mode.

3.5.2 Rframe

An rframe is a conceptual work surface for the rpen. It can be defined any where in 3D space. It basically represents the display of the monitor. It is rectangular in shape but as it is defined in 3D space, unlike the monitor display, it most closely represents a cube. The rframe can be of any size, only limited by the sensor's capability and effective interaction with the system. If within the sensor's reach, a relatively large rframe (in comparison to the display) could make interaction with the system easier. Whereas, a much smaller rframe, in comparison to the display, will likely make interaction difficult.

The rframe is conceived to be definable any where in 3D space. However, for the ease of operation and implementation in the current version of rpen, we begin with a rframe definable perfectly horizontal to three specific planes, which are the XY-plane, the XZ-plane and the YZ-plane. We refer to these as rframe types. The rframe that is definable anywhere on the space is called the *'generic type'*. The generic rframe is not fully functional in the current version of rpen. However, we will present its concepts, and how it will work in the future.

The rectangular rframe truly represents the display of the host system. Not only do they have the same shape but they also must have the same aspect ratio. This is the reason why in the initialization daemon, the rframe is updated if the display's set aspect ratio does not match that of the active rframe.

All the rframes other than the generic type, work in similar way. The rframe height and width are parallel to the axis of a plane, i.e., a rframe defined on XY-plane has its width along the X-axis and height along the Y-axis. Similarly, for the rframe defined on the XZ-plane, the width is parallel to the X-axis and the height of the rframe is along the Z-axis. For a rframe defined on the YZ-plane the width of the rframe is parallel to the Y-axis and the height parallel to the Z-axis. These restrictions on the rframe have made the calculations inside the transfer function less complex both in terms of logic and time. Simple plus or minus is good enough to determine the 3D position of the sensor with respect to the rframe.

The third dimension of a non-generic rframe represents distance from the

rframe surface. A point can be above the rframe surface or below the surface. A simple minus operation from the rframe's third dimension is good enough to give us this information. This is done this way because of the limitation of the current version of rpen hardware, which can only operate in the positive quadrants. Figure 3.11 provides an example of these types of rframes (other than generic). Here, $minX$, $minY$ and $minZ$ are the minimum distances from X, Y and Z axes, respectively. These values are derived from the three points defining the rframe. The figure is showing how a position data point ($P(x, y, z)$) from the sensor is transformed into a point ($P(x_1, y_1, z_1)$) with respect to the defined rframe.

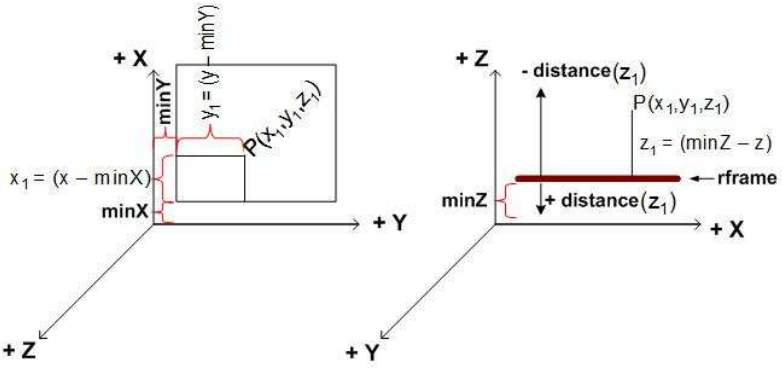


Figure 3.11: An example of a rframe on XY-plane. Rframes on the XZ-plane and on the YZ-plane will be similar.

A generic rframe could be anywhere in 3D space, and have an arbitrary orientation. Determining a point or sensor position with respect to this type of rframe is completely different than for the other types of rframe. As with any other rframe type, the process starts with defining the rframe with three points. The three points should be defined in such a way that they create a right turn or clockwise movement as shown in figure 3.12.

In figure 3.12, the three points P_1, P_2 and P_3 form the vectors $\overrightarrow{P_2P_1}$ and $\overrightarrow{P_2P_3}$. $\overrightarrow{P_2P_3}$ is clockwise from $\overrightarrow{P_2P_1}$ and thus making a right turn at point P_2. This means the cross product of vector $\overrightarrow{P_2P_1}$ and $\overrightarrow{P_2P_3}$ will be positive [CLR01]. Basically the cross product produces a vector which is perpendicular to both vectors used to produce the cross product. This new perpendicular vector is referred to as the normal vector. The normal vector of a plane is used to indicate the front and back side of the plane [WLH07]. The dot product of the normal vector (normalized) with any position vector (vector from the origin to the point) or point on the plane produces a constant value k_n [Fel09][Len04][Dav61]. From this we can determine which side of a plane any given point P_0 (used in the calculation as

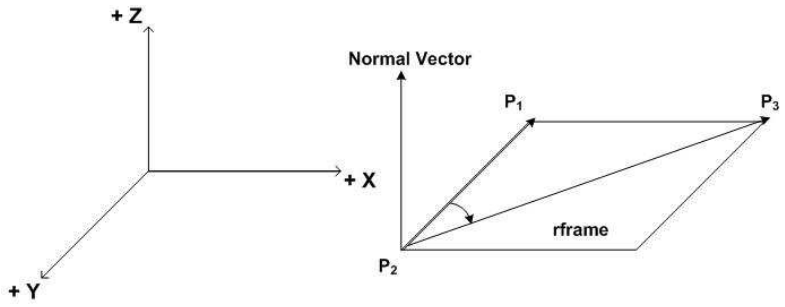


Figure 3.12: An example of a generic rframe, showing the normal vector.

position vector) lies, using the following calculations [Fel09][Len04][VB04]:

$$P_0(x, y.z) \bullet \text{normal vector} < x, y, z > = \text{constant value } k_{P_0} \tag{3.6}$$

if, $k_{P_0} = k_n$ then the point P_0 is in the plane
if, $k_{P_0} > k_n$ then the point P_0 is on the front side
if, $k_{P_0} < k_n$ then the point P_0 is on the back side

Given a point P_0, we need to interpret its position in term of the generic rfame. We do this by dtermining a point P_0' on the plane such that if we place a normal vector (with suitably length) at P_0', it would intersect P_0. The x and y coordinate of P_0' would be used to determine P_0 corresponding screen position, and the length of $\overrightarrow{P_0'P_0}$ would determine the z coordinate. We have used 3D geometric calculations found in [Fel09][Len04][VB04][Ste03] to determine this.

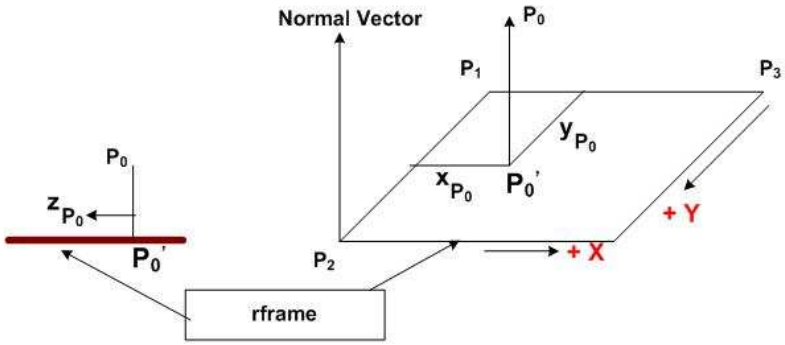


Figure 3.13: Calculating a point on the generic rframe.

The distance between a point and a plane is the length of a perpendicular

line from the plane to the point. The point where this line intersects the plane is the reflected point on the plane, of the given point. As shown in figure 3.13, let it be P_0'. Now the vector $\overrightarrow{P_0'P_0}$ is a parallel vector to the normal vector of the plane. So to calculate the line equation for $P_0'P_0$, we can use the normal vector and P_0. By calculating the intersection point of this line and the plane we can determine the value of P_0'. Once we know P_0', we can determine the $\overrightarrow{P_0'P_0}$. Now, from the perspective of the monitor display, the x, y and z values of point P_0 will be

z_{P_0} = length of $\overrightarrow{P_0P_0'}$

x_{P_0} = distance between the line which defines the height of the rframe and P_0'. In our example from figure 3.13, this would be P_1P_2.

y_{P_0} = distance between the line which defines the width of the rframe and P_0'. In our example from figure 3.13, this would be P_1P_3.

The last step in this process is to determine whether the point P_0' is within the polygon, defined by the rframe, or outside.

As described, the calculation process to find out a sensor position in respect of a generic rframe is complicated and would require more computation time than the other rframe types. The generic rframe is partially implemented in the current version of rpen. However, in some tests we have found out that a 32-bit long integer is not big enough to hold some of the computation results, namely, the constant values (k) described in equation (3.6). A 64-bit long integer is needed for these values. This means that, a fully functional generic rframe implementation has to run on a 64-bit machine with a 64-bit Linux Kernel.

3.5.3 Low Level Design

Here in this section we will describe each individual components of rpen in detail and how they fulfill the design features as specified in section 3.5.1.

3.5.3.1 Rpen Device Driver

Figure 3.14 shows the structure of the rpen device driver. We have already discussed the USB device driver specific operations in section 3.2.1.4 and the input device driver specific operations in section 3.2.1.5. Most of the Wintracker specific operations, like the data processing in 3.3.2 and filtering in 3.4, have also been discussed. Here we will mostly discuss the rpen specific operations and some part of the Wintracker specific operations. We will elaborate on the data and control flow read from the Wintracker, and how this information is translated to input events.

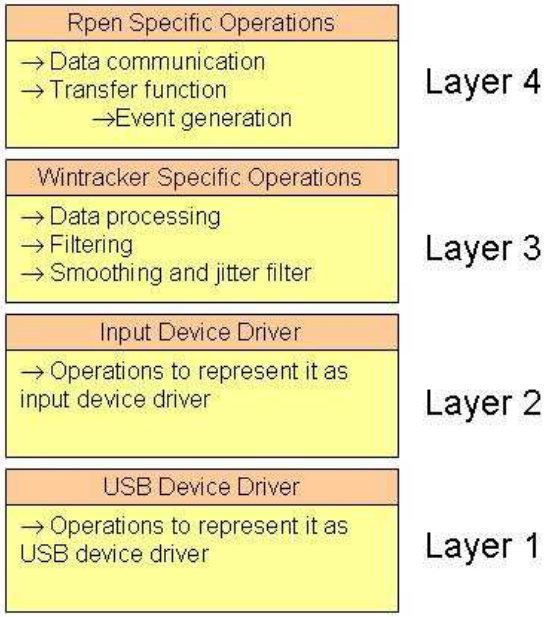


Figure 3.14: Rpen device driver structure.

Data Communication :

Data communication between the rpen device driver and the user space processes are bidirectional. There is some data that the device driver needs from the user space processes to work appropriately, whereas, some data is also required by the user space processes to carry out their functionality and provide user feedback. The set of data that is communicated is as follows:

1. Rframe related information:

   (a) The three points that define the rframe. These values are only read by the device driver and written by all the three user space processes (the device driver reads these values from the sysfs node and the user-space processes write these values as they change the rframe); thus, could be identified as IN only. These values are used by the device driver to calculate the normal vector of the rframe plane. Currently, the normal vector is calculated for all rframe types, but, only used for the generic type rframe.

   (b) Minimum and maximum values of X, Y and Z coordinates inside the rframe. These values are used by the device driver to transform the sensor data to cursor position and for smoothing the sensor data.

These values are also IN only parameters. Rather than being determined from the three points of rframe, these values are fed from the user space processes to save calculation time inside the device driver.

(c) Rframe resolution information is used in the transformation process of the sensor position to the cursor position. As the initialization daemon checks these values to see if the rframe data used by the device driver is up to date, rframe resolution data is not an IN only parameter.

(d) Threshold data (described in section 3.5.3.3) is an IN only parameter. These values are used by the transfer function to generate events appropriately.

2. Display resolution: This is the display resolution set on the host system. This value is used to set maximum acceptable position values of the input device. As the sensor data is mostly fed to the input core as cursor position, the maximum values it can reach are the display width and height. Also the display is only two dimensional and the display width is viewed as the X-axis and the display height is viewed as Y-axis. As such we have set display width as the maximum X value of the rpen input device and height as the maximum Y value.

   As rpen is a 3D input device, we defined the maximum Z value of the device to be the maximum range of the sensor. This value is also passed to the device driver from the user space as part of the display resolution. Like the rframe resolution data, display resolution is also accessed by the initialization daemon for the same reason. This is thus an IN parameter.

3. Multiplication factor: These values are used to transform sensor data to display the cursor position and they are IN only parameters.

4. Number of samples to average: This value is fed by the rframeDefApp application and thus is an IN only parameter. This value is used in the sensor data smoothing process.

5. Flags: There are three flag parameters that are being communicated back and forth between user space and the device driver and are thus IN and OUT parameters. These flags are:

   (a) Transfer flag: This flag indicates to the device driver whether to use or not to use the transfer function. This function transforms sensor data on a rframe to cursor position on the display.

   (b) Update flag: This flag notifies the device driver whether or not the user space processes have updated any of the parameters that it is using.

(c) Rframe orientation flag: This contains the rframe type data (rframe type is explained in section 3.5.2). Rframe orientation flag is a very important flag for the transfer function.

6. Cursor position: This is the reported 3D cursor position to the input event driver. This parameter is OUT only as it is an output of the device driver and used by the user space processes.

7. Click event: This is also an OUT only parameter. This is used to communicate the occurrence of a left button click event. This is used by the feedback daemon to simulate a mouse button click sound.

Transfer Function: The transfer function gets the sensor's 6DOF data and reports to the input event driver the transformed cursor position as well as the button click events. The current version of rpen does not have any button. So the click events are generated from the transfer function through interpretation of the sensor position and orientation, which we refer to as "user gestures."

The transfer function does not always report the transformed cursor position to the input core and the user space processes. When there is no 'active' rframe or the user wants to define a new rframe, the transfer flag is set to signal to the device driver to stop transforming the sensor data, although the data is still filtered. Instead it just reports the filtered sensor data. Also, no click events are reported.

When the transfer function is set to transform the sensor data, it first (**step 1**) interpret, if the right click is initiated or not. Right click can be initiated if the sensor is twisted or rolled to the right on the X-axis more than or equal to 25°. The next step (**step 2**) is to determine the sensor data in respect to the active rframe. As described in section 3.5.2, all the rframe types other than the generic type interpret the sensor data similarly. Section 3.5.2, also describe how it is done for the generic rframe.

The next step (**step 3**) is to interpret the processed point in terms of the display screen. As we have mentioned earlier, the monitor display is two dimensional. The monitor display coordinates are represented as shown in figure 3.15. In this step, we represent the coordinates of the sensor data transformed in terms of the rframe's width and height to the display's x and y coordinates respectively. For example, for the rframe on the XY-plane, the sensor's data coordinates (x, y) from step 2 are transformed in terms of (x, y) coordinates of the monitor. For a rframe on the XZ-plane, the (x, z) coordinates are interpreted as (x, y) coordinates of the display. For a rframe on the YZ-plane, the (y, z) coordinates of the sensor position are transformed in terms of (x, y) of the display. Lastly, for the generic rframe the distance from the line signifying the height of the rframe (the one nearest to the origin, as there will be two such lines) and the line defining

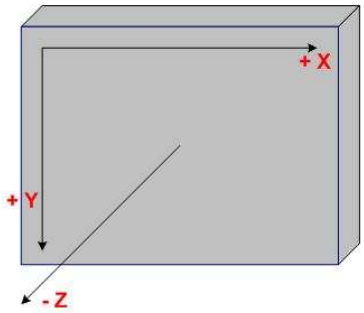


Figure 3.15: Coordinate system for the monitor display.

the width of the rframe (similarly, nearest to the origin), as described in section 3.5.2, is transformed to (x, y) coordinates of the display. The third coordinate is left as it is and reported to the input core and sysfs node.

The fourth step (**step 4**) is to compensate for the difference of the rframe to the display resolution. This transformation is done only on the two position coordinates, in terms of the monitor display, x and y. The calculation is as following:

> Transformed position value = (Position value*Display height)/Rframe height

After this transformation, the (x, y) coordinates are reported to the input core and sysfs node.

The three filtered orientation values (roll, pitch and yaw) do not go through any transformation and are reported directly to the input core. The position data transformation can be explained through the following example. Let an rframe (R) on XY plane be defined with three points $P_1(828, 502, 2350)$, $P_2(828, 3649, 2350)$ and $P_3(4762, 502, 2350)$ and the screen resolution be 1280x1024. Now, a sensor position $P_k(2400, 2600, 2400)$ will be translated (see figure 3.11) to $P'_k(1572, 2098, -50)$ in respect to the rframe (R). Here, -50 signifies that the point is above the rframe (R) surface. Now, transforming (x,y) coordinates of the point P'_k in respect to the display resolution will be $((1572 * 1024)/3147, (2098 * 1024)/3147) = (511, 682)$. So, the input core will get $(511, 682, -50)$ for the sensor data P_k.

To generate the left click and left double click, further processing is done with the z-coordinate of the display and the sensor's roll angle value, i.e., orientation around the X-axis. There are two thresholds on the z-direction that can influence the generated input events. One is below the surface of the rframe, called the surface threshold, and the other is the clutch threshold above the surface of the

rframe. Above the clutch threshold, the sensor position is not reported to the input core; only the z-position is updated in the sysfs for proper visual feedback by the feedback daemon. On the other hand when the sensor position is below the surface threshold, it is interpreted to have initiated a left click. One important aspect of the left button click to remember is that a complete left click is a combination of pressing the button and then releasing it. Similarly in our case, it is going below the surface threshold and then coming back above the threshold.

Due to the sensor's limitation as well as difficulty associated with operating a spatial device, clicking on the same spot twice simultaneously is a difficult operation to achieve. For ease of use, we implemented left double clicking so that it depends on the orientation of the sensor about the X-axis, when it is below the surface threshold. When the sensor is below the surface threshold and is twisted on to the left more than 35°, a left double click is generated.

Due to our choice of sensor (electromagnetic), its accuracy, and its sampling frequency, we found that typically the rpen's screen location at the beginning of a left click did not match its position when it reached the threshold or the bottom of the left click motion. This resulted in clicking in the wrong spot and sometimes initiating a drag operation. This is due to the fact that the path of the rpen was an arc due to rotation around the fingers that held it, or the user's wrist. This prompted us to implement some features which we think will increase ease of use. One such feature is, if the user makes a pronounced downward movement from above the surface threshold of the rframe down below the threshold, then it is interpreted as an intention to left click and the point where this downward movement started is recorded as the click point. But, if the sensor moves more than 20mm in any of the other two directions then this intention is invalidated. While the rpen is in the "intention to left click" state, if the displacement of the sensor is less then 20mm, none of the 6DOF events are reported to the input core. This means for a drag or select motion to start, the sensor has to move more than 20mm.

Another feature is to stop tracking sensor movements if it is tilted more than 8.5°on the right side of the X-axis. This is interpreted as intention of making a right click. Similarly, when below the surface threshold if the sensor is tilted more than 8.0°to the left of the X-axis, the sensor movements are not tracked until it comes back up from the tilt or goes above the threshold. This gesture is interpreted as an intention for a left double click.

Dataflow at Each Data Reading :

The flow of data and control information within the device driver, is shown in figure 3.16.

We have already discussed the steps up to filtering and transforming the

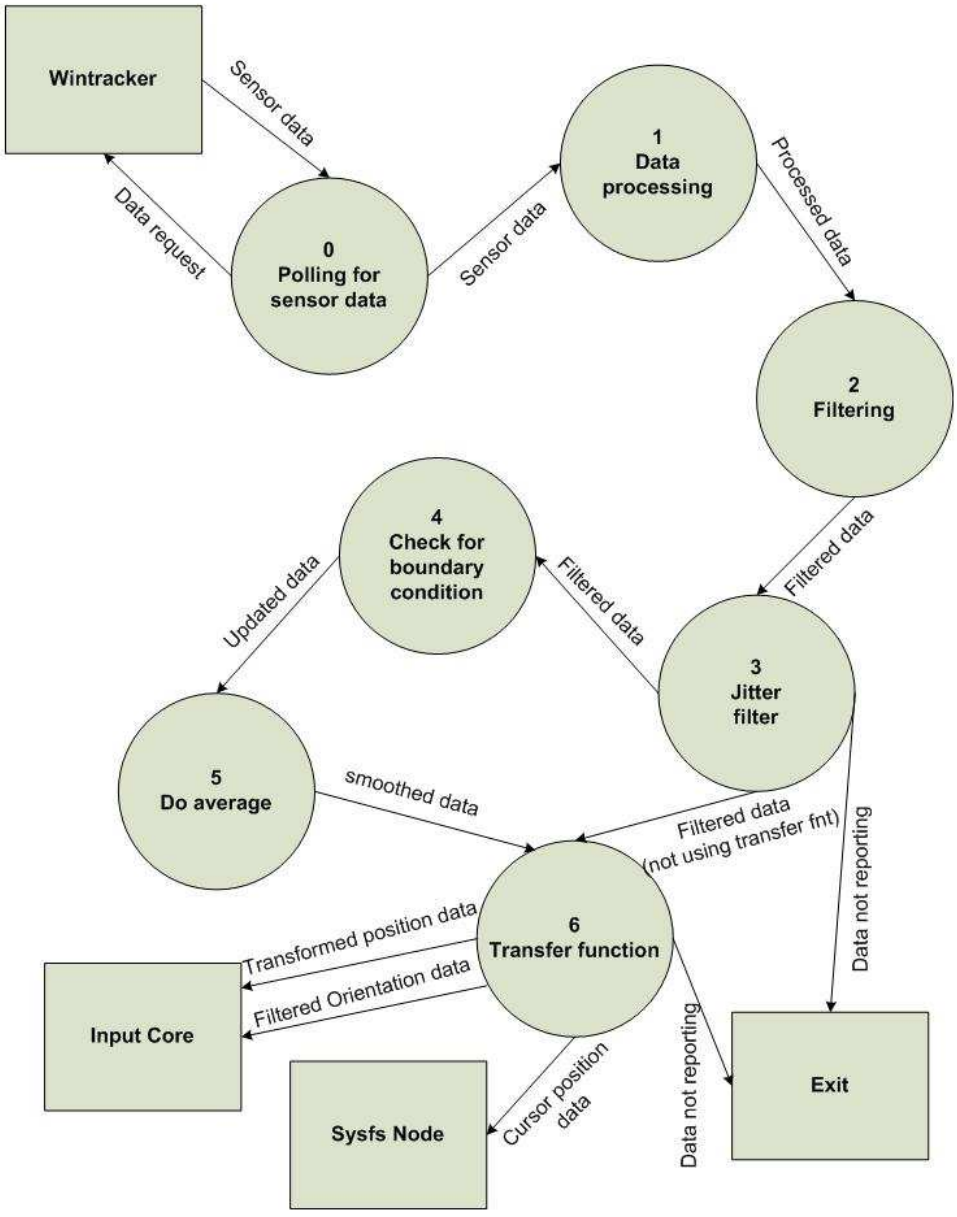


Figure 3.16: Dataflow diagram of the rpen device driver, showing dataflow on each sensor data read.

data. We now discuss the jitter filter. All it does is check if the filtered value of the lowpass filter is within 0.3mm range for position data and 0.99°for the orientation data, in comparison to previous sensor values. If so, it filters this data out; otherwise it lets the lowpass filter value pass through. This jitter filter does not filter out sensor data if it is the same as the previous sensor input. The jitter filter helps to filter out small, jumpy sensor values.

Even after the lowpass and jitter filter, the sensor data was found to be jumpy and noisy. To smooth out the data, we implemented averaging of the previous filter values plus the last output value of the averaging (smoothing) operation. To do an effective averaging, we introduced a boundary check function. The function does not allow the position values to go too far beyond the boundaries set by the rframe and clutch threshold. The sensor position can get at most 2mm outside the set boundary.

The employed smoothing technique is basically a sliding average. It means, it averages a certain number of recent filter values. The number of samples to average is definable by the user, and can be at most 125. This is done for the ease of use of the user. Startup users may like an increased number of averaging points. As users gets used to the system, they typically like as little averaging as possible. It is observed that 25-30 data points to average is good for both smooth but responsive cursor movements.

3.5.3.2 Rframe Definition Application

The rframe definition application *'rframeDefApp'*, is developed to facilitate features related to defining an rframe. Defined rframes are primarily used by the rpen device driver to interpret the sensor movements in accordance with the characteristics of rpen. The user can define multiple rframes using the application, each having a different rframe type, as shown in figure 3.17. But, only one rframe can be active at a time and this active rframe is used by the device driver and the other daemon processes. If there is no active rframe defined, then the sensor data is reported after filtering; averaging and the transfer function are not used. Also, the two daemon processes can not perform any of their assigned tasks.

The rframe definition application uses an XML data structure to store defined rframes, current display information and other data items that are defined using the application. The GUI of the application is also stored in XML format, but in a different file and is rendered through the GnomeUI libraries. The design structure of Rframe Definition Application is given in figure 3.18.

Other than defining new rframes, the user can update, delete and activate rframes, as can be seen in figure 3.17. The user cannot delete an active rframe. A new rframe can be defined through the interface, as shown in figure 3.19. When the user wants to update an existing rframe or define a new rframe, the

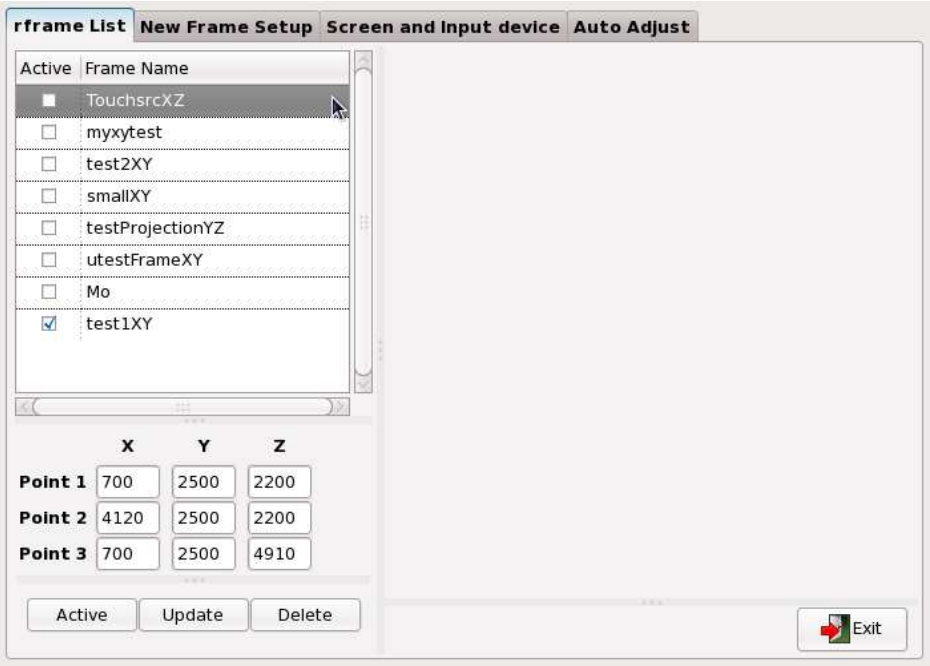


Figure 3.17: Rframe definition application UI - page 1.

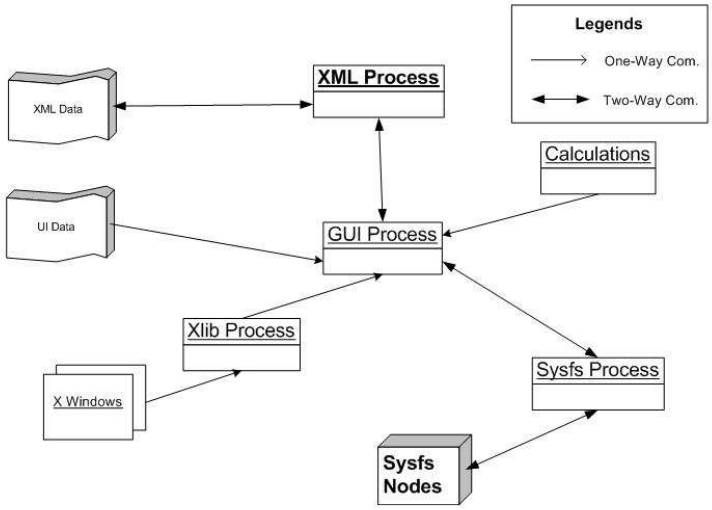


Figure 3.18: Rframe definition application structure.

application sends a signal to the device driver through sysfs node, to send filtered sensor data only, i.e., when using the interface, shown in figure 3.19, the rpen is not using any rframe. When the user is done using this interface, the device driver is commanded again to go back to using the transfer function and the active rframe.

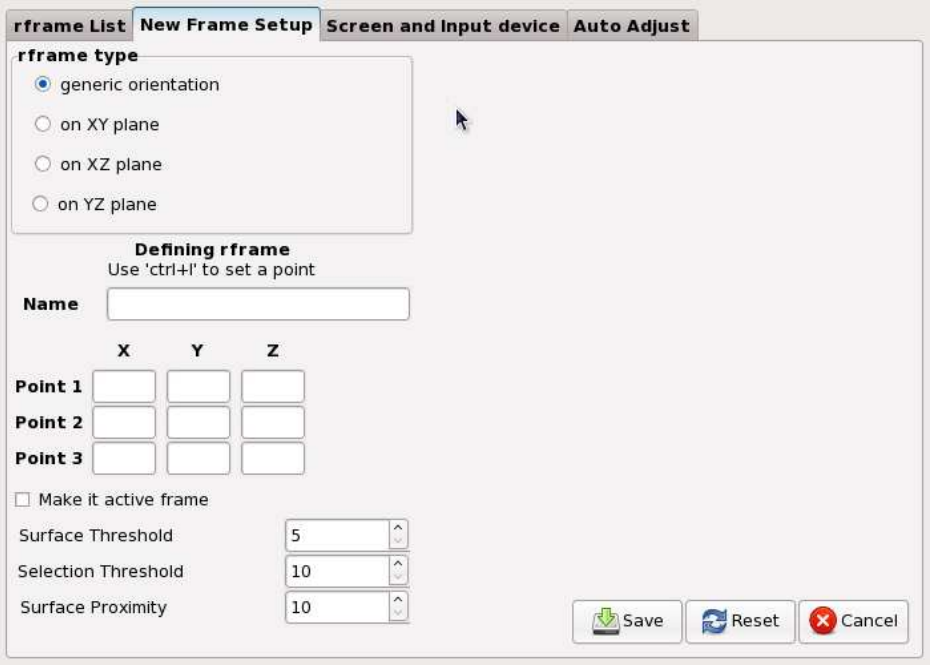


Figure 3.19: Rframe definition application UI - page 2.

When defining a rframe, an user has to provide the rframe type, a unique rframe name, three points that define the rframe, the threshold values and an indication whether or not this rframe should be defined as the active rframe. Here, the rframe name works as the primary key, so, the user must provide a unique name for each rframe. If an rframe is given a duplicate rframe name, then it is not saved by the application. During the rframe defining process, the user may not be able to use the rpen in a meaningful way, other than to define three points. During this process, the user is most likely to use the keyboard to enter commands. To set each of the three points that defines the rframe, the user must use the *'ctrl+l'* key. The points have to be defined in the ways described in section 3.5.2, making a right turn.

It is not easy to define all three points of the rframe without a physical reference. The aspect ratio of the rframe has to match that of the monitor display. The process is easier for the rframes on XY-plane, XZ-plane and YZ-plane, but much harder for the generic rframe. For rframes on specific planes

such as the XY plane, the axis of the width and height is already known. So for these rframes when the user has defined the first two points, it is known on which axis the third point will be and the length of that side can be derived by using the display aspect ratio. The third point is then adjusted following this rule, after the user has given an approximate third point.

The process is more complicated for the generic rframe. It is not known from the first two points whether they define the height or the width of the rframe. So, the approximate third point has to be given by the user to compare the lengths of the two lines (P_1P_2 and P_1P_3 - see figure 3.20) according to the display aspect ratio to find out which one is the width and which one is the height.

All the rframe types use the same technique (as described below) to adjust the third point P_3 (see figure 3.20). As shown in figure 3.20, we know that line P_1P_2 and P_1P_3 are supposed to be perpendicular to each other. We can also predict which line is the width of the rframe and which one is the height. So the lengths of both lines are known. Now, we can adjust the third point following the calculations given below.

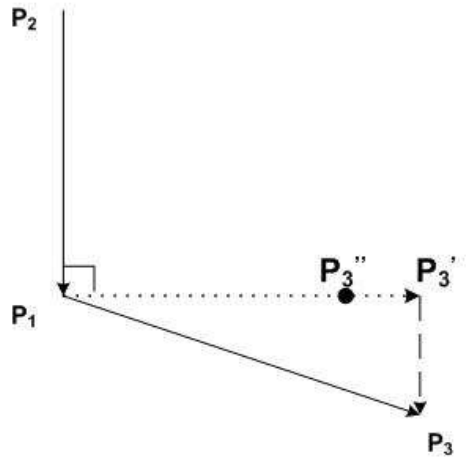


Figure 3.20: Adjustment process of a new rframe.

P_1, P_2 and P_3 are the three points given by the user. Let, P_3'' be the new adjusted point of P_3. In the following calculations we use these points as position

vectors and visa versa. From the figure 3.20 we can see:

$$\overrightarrow{P_3'P_3} = Proj_{\overrightarrow{P_2P_1}}\overrightarrow{P_1P_3} \tag{3.7}$$

$$Proj_{\overrightarrow{P_2P_1}}\overrightarrow{P_1P_3} = (\frac{\overrightarrow{P_2P_1} \bullet \overrightarrow{P_1P_3}}{||\overrightarrow{P_2P_1}||^2})\overrightarrow{P_2P_1} \tag{3.8}$$

$$\overrightarrow{P_1P_3'} = \overrightarrow{P_1P_3} - Proj_{\overrightarrow{P_2P_1}}\overrightarrow{P_1P_3} \tag{3.9}$$

$$P_3'' = P_1 + d\frac{\overrightarrow{P_1P_3'}}{||\overrightarrow{P_1P_3'}||} \tag{3.10}$$

$$\text{as, } \overrightarrow{P_1P_3''} = d\frac{\overrightarrow{P_1P_3'}}{||\overrightarrow{P_1P_3'}||} \tag{3.11}$$

Here, $Proj_{\overrightarrow{P_2P_1}}\overrightarrow{P_1P_3}$ means vector projection of $\overrightarrow{P_1P_3}$ onto $\overrightarrow{P_2P_1}$ and d = the required length of P_1P_3, satisfying the aspect ratio of the set display, which will be adjusted by the above calculation to P_1P_3'' having the same length d. More information about how to deal with similar vector calculations and projections can be found in [Len04][VB04][Ste03].

Other than rframe specific operations, 'rframeDefApp' is used to set the sensor's maximum limit and to set the number of sensor values to be used in the smoothing operation, as shown in figure 3.21. This interface also shows the display resolution and other related information.

The last task the rframe definition application does is to set the auto adjust parameters to be used by the feedback daemon, as shown in figure 3.22. User can switch on/off the auto adjust mode as well as set the time limit for the sensor to be idle to initiate the process of adjusting the rframe automatically. For simplicity, we only allow a generic rframe to be adjusted automatically as one of the three other rframe types, i.e., as a rframe on XY-plane or XZ-plane or YZ-plane.

3.5.3.3 Rpen Daemons

As mentioned in section 3.5.1, there are two daemon processes that works with rpen. One is the *'Initialization Daemon'* and the other is the *'Feedback Daemon'*. The main idea behind the initialization daemon is to communicate the active rframe information to the rpen device driver on system startup. Active rframe information during modification and new active rframe creation is communicated to the rpen device driver by the rframe definition application. Rpen gets the rframe and other related data through sysfs nodes. Sysfs is a RAM based file system. During a system shutdown or reboot sysfs loses the data it contains. So,

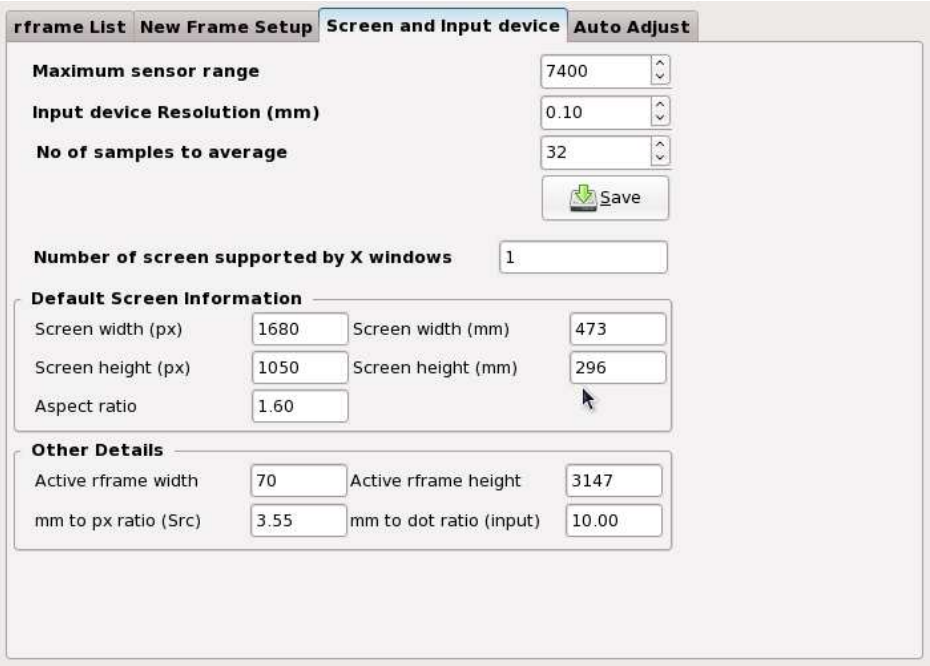


Figure 3.21: Rframe definition application UI - page 3.

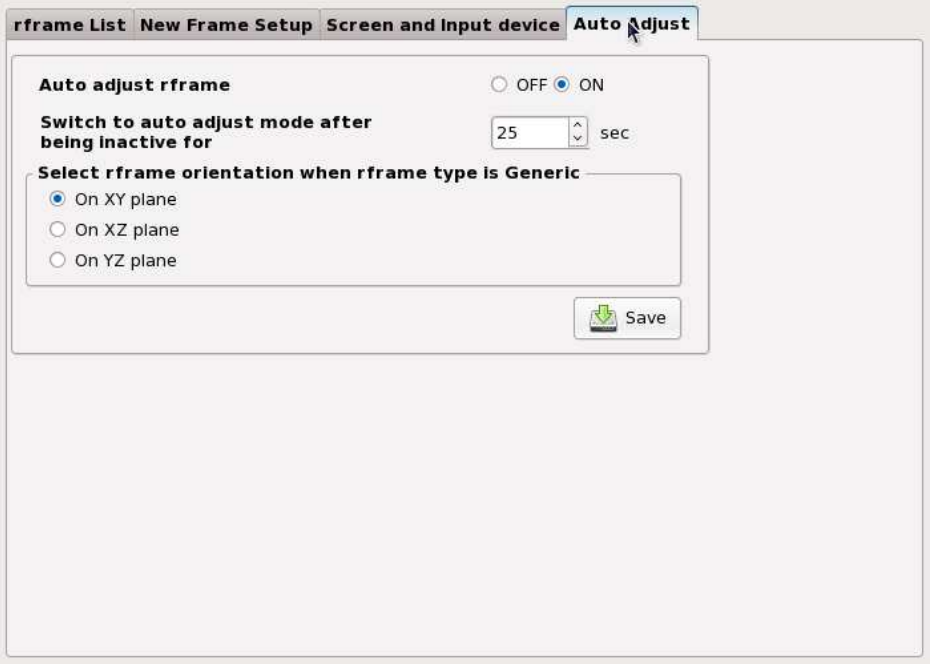


Figure 3.22: Rframe definition application UI - page 4.

after the system startup, these sysfs nodes need to be repopulated in order for the processes dependent on them to work properly. The initialization daemon does this for the rpen device.

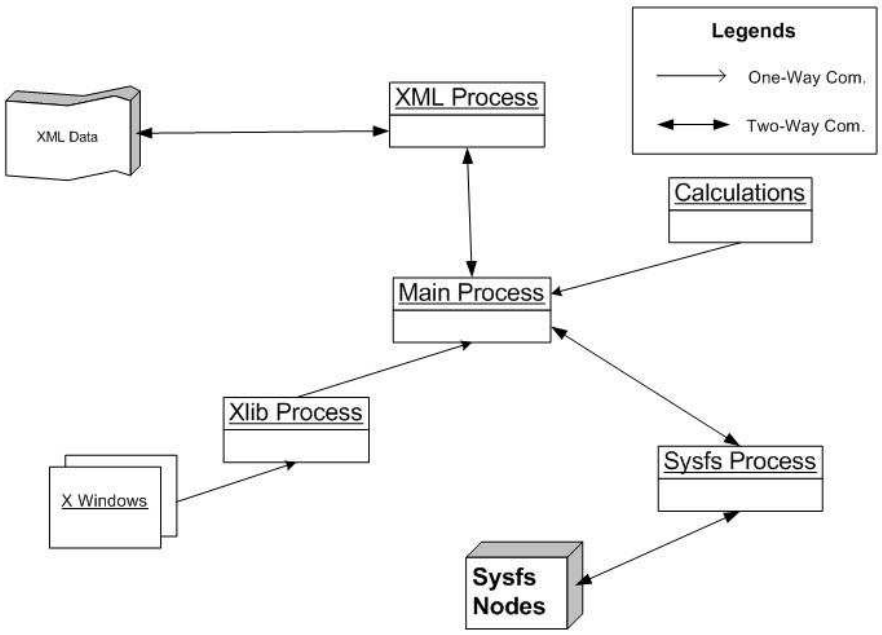


Figure 3.23: Initialization daemon structure.

The other responsibility of this process is to update the rframe according to the set display resolution. As discussed earlier, the display aspect ratio is very important for the rframe to follow. So if the user changes the display aspect ratio while using the rpen, the initialization daemon will adjust the rframe according to the display aspect ratio within at most two and half minutes. The structure of the initialization daemon is given in figure 3.23.

It is difficult for the user to use an rpen with an rframe without a physical reference. The user might unintentionally come down the surface threshold and produce a click. The user might also go too high above the rframe surface to accidentally trigger the clutch mode and wonder why the cursor is not moving. It might also happen that during a drag event the sensor rises above the surface threshold, as hand movements in the air are not always reliable. So, the feedback daemon helps the user to obtain a visual feedback through the cursor appearance. It is done on the basis of z-movement, in terms of the monitor display, of the sensor.

The cursor can take seven shapes, as shown in figure 3.24. Above the clutch threshold it takes a diamond shape. Within the clutch threshold and surface

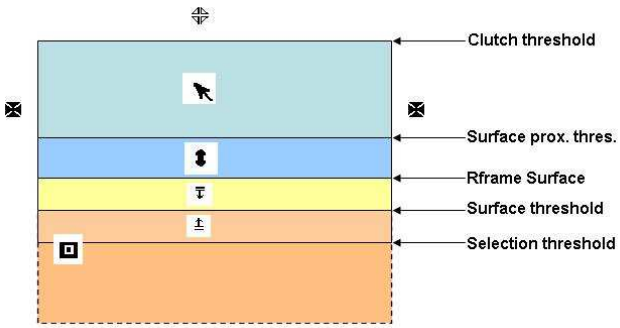


Figure 3.24: Rframe surface layers and corresponding cursor appearance.

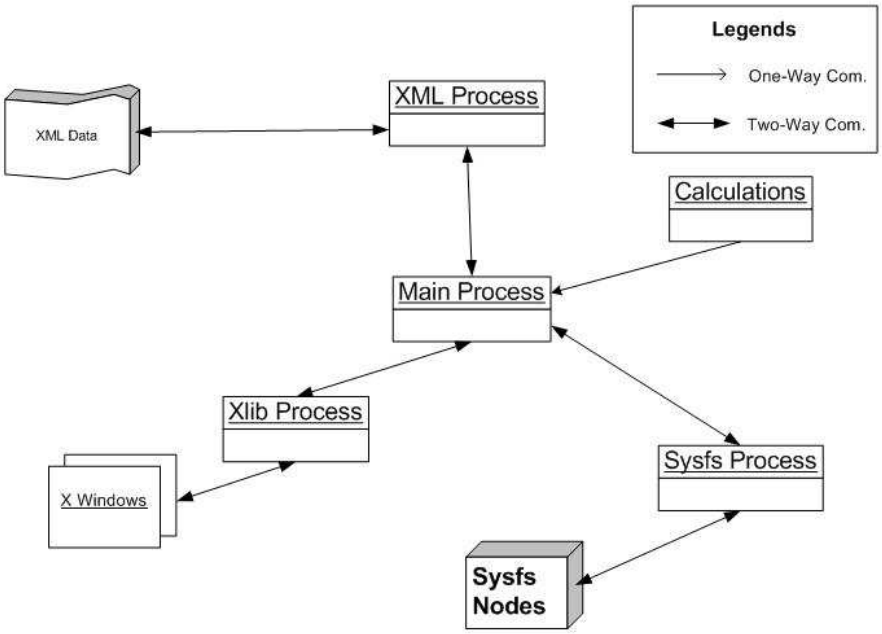


Figure 3.25: Feedback daemon structure.

proximity threshold it is the default left arrow cursor. In between the proximity threshold and surface of the rframe, the cursor appears as a double sided arrow. This layer helps the user to keep the sensor within a good distance from the surface threshold (not too far) so that user can go for a click whenever required without being bothered about accidental clicks. Below the surface and within the surface threshold the cursor appears as a down arrow. It lets the user know that they are about to click. Below the surface threshold it appears as a dotted box, showing it is in click mode. To help the user not to exit the drag motion accidentally, the cursor appears as an upward arrow when the user is coming up from below the surface threshold towards it. The upward arrow will appear within the surface threshold and selection threshold. To signify the sensor is out of the rframe boundary, the cursor appears like a cross.

The feedback daemon also takes care of the auto adjust mode, as discussed earlier. If the auto adjust mode is 'on', the daemon checks if the sensor is idle for more than the set time limit. If so, then the auto adjust flag is switched on and the next time the user picks up the sensor and holds it for at least five seconds at a position, that position is taken as the first point (P_1 as shown in figure 3.20). The other two points are calculated on the basis of this point and the saved width and height of the active rframe. As the other two points are generated from the first point, it is recommended to be defined nearer to the base station (transmitter), which is the origin $(0, 0, 0)$. The structure of the feedback daemon is shown in figure 3.25.

3.6 Usability Testing

Usability of a Human Computer Interaction (HCI) interface can not be defined as plainly as *'user friendly'*. The problem of this interpretation is not that it does not convey the meaning, rather in terms of HCI it is not measurable, thus, too *'vague'* [SP05]. Usability in HCI is sometimes defined by the level of frustration of the user using an interface. A user must be able to accomplish his or her tasks through the interface without any abstraction, difficulty or hardship. The interface must seem to "disappear" to the user, enabling seamless interaction with the system [SP05].

Usability for the interface must be measurable. A measurable set of criteria helps guide the designer to create better, effective and usable interfaces. [RC08] described usability in terms of usefulness, efficiency, effectiveness, learnability, satisfaction, and accessibility.

1. Usefulness: This criterion measures the level of accomplishment of the tasks the user want to do through the interface. This is the most important criterion within the set of criteria of usability. Because, if an interface is

deemed not useful then there is no point in pursuing it.

2. Efficiency: It involves the amount of time required to accomplish the given tasks.

3. Effectiveness: This criterion measures the error rate as well as whether the interface does what it suppose to do.

4. Learnability: It has to do with the amount of training time required by the user to learn the interface and the amount of time to relearn the interface after a while.

5. Satisfaction: It is user's opinion on the interface.

6. Accessibility: This criterion answers the question about whether the interface provides adequate access to complete the tasks that the user needs to perform.

Ben Shneiderman and Catherine Plaisant termed these as admirable goals but they lack practical evaluation [SP05]. They suggested a different set of criteria focusing on efficiency and satisfaction. In their opinion these are easily measurable. Their set of criteria is as follows:

1. Time to learn - the interface

2. Speed of performance - how long it takes to complete a task

3. Rate of errors - by user

4. Retention over time - how well the user remembers how to operate the interface

5. Subjective satisfaction - how much the user liked the interface

3.6.1 Usability Measuring Techniques

To measure usability of an interface, the practitioners and academics have been using many tools and techniques. Depending on the set of circumstances and the interface to be tested, usually a combination of these methods might be used. The goal of doing the test also influences the choice of usability measuring techniques. The product development life cycle also has considerable influence on the choice of method. Some of the techniques have similar characteristics, thus they can be grouped together. Here we present some of the major techniques used for measuring usability.

Ethnographic Research: In this method the test moderator or the analyst observes the users performing their tasks in the actual work environment. As described in [Car03], the observer "shadows" a particular user, and witnesses the situations the participant user goes through during the day. This technique is usually utilized early in the design phase of an interface. The participants of this test are the actual users of the interface.

Participatory Design: It is basically a part of the design process. Participant users become part of the design team. This technique promotes collaboration between the users and the designers in identifying usability issues and their solutions.

Expert Reviews: There are several methods that fall under this broad category. The common thing about these methods is that the participant user is an expert, whose expertise could be in the area of the application or user interface design [SP05]. Expert reviews can be done at any time in the design phase. These reviews can be arranged at short notice and finished within a short time frame. The methods that fall under this category are:

- Heuristic Evaluation
- Guidelines Review
- Consistency Inspection
- Cognitive Walkthrough
- Formal Usability Inspection

Survey: A survey can reach a broad range of users of the product. The results of a large enough survey can be generalized to an entire population. The survey is good in providing statistical data and good for finished products. A well designed questionnaire is crucial for the success of a survey.

Pluralistic Walkthrough: This method systematically examines the usability of an interface from a task-based point of view. Multiple groups made up of users, developers, and designers, independently perform tasks in their own way. Then the groups present their findings to the other groups.

Empirical Method: This method works by determining a hypothesis and a set of objective measures. The user then tests the interface in a controlled environment to prove or disprove the hypothesis. This method can establish cause and effect of a specific problem but can be time consuming.

Formal Design Analysis: This is a model based usability measuring tool. Models like "Goals, Operators, Methods, and Selection Rules" (GOMS) require expert users. This method is difficult to implement and misses some important usability measures.

Interviews and Focus Group Discussions: An interview of individual users can be done just after the usability test. This can be done in the form of questionnaire as well. This technique can reveal the specific concerns about the interface. A focus group can be used to verify these comments. This technique has the potential to go for in-depth user response and assessment of the interface.

Logging Actual Use: This technique uses automated systems and software to record each participants' move and interaction. It produces a huge amount of data to be analyzed.

Usability Testing in Lab Environment: This technique is used to collect experimental data from representative users of the interface, usually in a lab environment. There are two main approaches used for the testing. One is the empirical method discussed earlier and the other is relatively less formal and employs an interactive process to gradually shape the interface. There are many forms of usability testing [SP05], some of which are:

- Paper Mockups uses paper mockup interface or layout of the interface
- Discount Usability Test it uses a quick-and-dirty approach to task analysis, prototype development and interface testing.
- Competitive Usability Test - It tests the new interface with existing similar products from competitors.

3.6.2 Usability Testing Methodology

Testing technique: We have employed a combination of the usability testing techniques described in the last section. We used usability testing in a lab environment which utilizes some part of the empirical method. There was a pre-defined set of tasks prepared for the participant users to perform. It was controlled in the sense that the tasks were designed by the testers and we also supplied the participant users with user instructions on how to perform the tasks. We did not have a hypothesis to prove or disprove for the usability test, but we had some goals and objectives. We knew that the device is still under development and has hardware limitations. So, we tried to make the users aware of these facts and asked them to keep these in mind while evaluating the interface; thus, provide feedback looking beyond those limitations.

To record test results, we used mainly the questionnaire technique and also observation in some respect. After the users had finished the given set of tasks, they were given a set of questions to answer. The users provided the answers themselves, i.e., without any interaction with the test moderator. During the test, users were also observed to check if they were trying to do something that was not instructed, and why they were doing that. The questionnaire and the

instruction manual are included in Appendix A.

Testing goals and objectives: Our goal for the usability test was to find out if rpen does have potential as a pointing device. We also wanted to find out if the participants think rpen could be a viable alternative to any of the currently used pointing devices.

Error or lack of precision is an important factor for this kind of usability testing. Any flaws big or small, influence the user's attitude towards the interface [SP05]. Knowing about the limitations of rpen, we tried to get feedback only on the concept of the rpen, decoupling it from the hardware. We encouraged users to think beyond the demonstrated set of usage, on the scope of rpen's use and applicability.

Participants: For the usability test we chose to have five participants. Rpen was developed as a proof-of-concept, thus, we were not trying to get a diverse and large set of users. It is advocated in [Nie93] that a size of three to six participants provide the best outcome of a usability test. Also, we were looking for specialized opinion within a limited timeframe. We chose participants who had prior knowledge of HCI or had considerable knowledge of similar kinds of devices or technologies. We did not give particular attention to the age factor as well as handedness of the participants. Even then, we obtained a fair distribution of ages among the users although all of our users were right handed. It should be mentioned here that two of the participants used the device at least once before the usability test. The other three participants were seeing and working with the device for the first time.

CHAPTER 4

Results

4.1 Introduction

This chapter presents the significant outcomes of the current study. The first section shows the effects of filtering and smoothing techniques applied on the raw data originating from the Wintracker. The data processing function inside an input device driver is very time sensitive. Enough emphasis has been given by the device driver experts that a device driver, specially an interrupt based one, should only do the bare minimum when called on by the kernel to service the device. As we envision rpen as interrupt based input device it is critical to know how much time is taken by the 'callback function' (function that is called to service the device) at each stage of computation. The second section presents these results along with the effects of roundoff errors, as we could not use floating point operations inside the kernel. Lastly, the observations and user feedback on usability related issues are presented.

4.2 Filtering and Smoothing

This section describes the effects of filtering and smoothing on the raw sensor data. It also presents the data aquisition process.

4.2.1 Test Setup

To understand the nature of noise mixed with the 6DOF sensor data that comes from the Wintracker, we need to analyze the raw sensor data. A user-space program was developed to obtain the raw data. We utilized *'libusb'* in the program to link with the hardware device. Libusb provides user-space mechanisms to communicate with USB hardware. As explained in chapter 3, Wintracker supports only bulk data transfer both for receiving commands as well as sending sensor data. Also, state changes of a sensor only can be known by requesting sensor data from the Wintracker by the host system. We implemented both IN and OUT bulk data transfer for the program using the API/Interface provided by the libusb module. To get sensor data, the program implements an infinite loop

which requests sensor data every millisecond. The same program is also utilized to run the raw data through the tested filters and smoothing techniques.

As the program is used for multiple tasks, it was implemented in stages. These stages are defined with the target to identify the noise and to monitor the changes to the data after applying a technique to filter out the identified noise. The first stage is to obtain the raw sensor data as it comes from the hardware system. The second stage is the lowpass filter. The third stage is the averaging. These stages were not built at once. As we experimented with raw data and different alternate solutions, the other two stages were built on the first stage. These two stages were tuned as we monitored the effects these changes were having on the input data. The test results will be presented in the same way, in the next section.

As the current version of rpen mostly uses the sensor's position data, we have only analyzed the positional data through this program. We generalized the outcome of the noise analysis as, *noise influences the orientation data the same way as it does the position data*. So, the resultant filter and smoothing technique are implemented for both the positional data as well as orientation data, in the same manner.

Data produced at each stage is stored in an individual ASCII text file. All the files have the same data format. The values are integer type, paired as time and value, for each of the three coordinate data. All the columns are tab separated and each line represents a row.

To better understand the noise and filtering effects we collected data by moving the sensor in a predictable manner. We printed a rectangular shape with the resolution of 20x27cm on a regular letter size paper. The size of the rectangle is not important here, it just helped us to make the data predictable. The rectangle was placed on the XY-plane and we moved the sensor on an average 8 complete rotations of the rectangle boundary per minute for one minute or so, each session. This regular and predictable motion of the sensor gave us a signal, roughly forming a sinusoidal wave for the x and y sensor position data. Figure 4.1 shows the ideal pattern for the acquired sensor data for x position information. The pattern for y position will be similar. We expected that the acquired data would not follow the shown ideal pattern due to several reasons. Firstly, the noise in the working volume will create rough edges. Secondly, it is difficult to maintain a constant paced smooth motion for human operaters. But, as the filtering and smoothing techniques were applied on the acquired data, the rough edges on the sine wave started to become smoother.

The raw data are used to produce the power spectral density. A power spectral density has dimensions of power per Hz. This helps to identify the periodicity of the signal. Using this spectrum we tried to identify the filter to be used as well as critical frequencies for those filters. We tested both lowpass

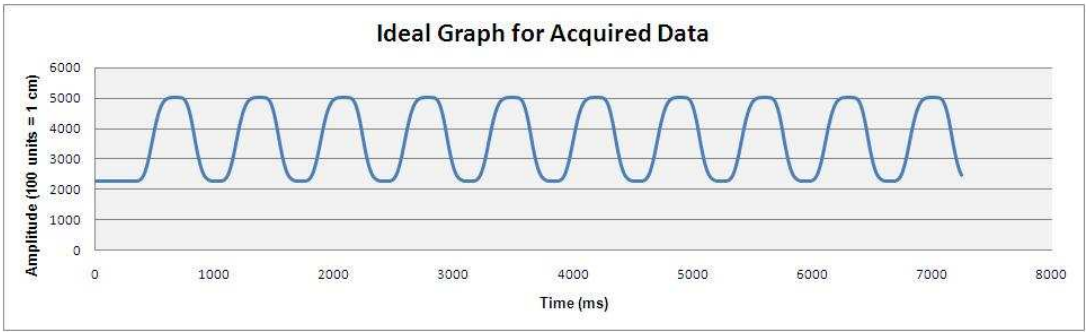


Figure 4.1: Expected raw sensor data pattern for the x position information in ideal scenario.

and bandpass filters in the Labview measurement tools [Nat09] with the critical frequencies identified through the power spectral density. We observed that the lowpass filter was producing the best outcome in reducing noise from the raw sensor data.

The test environment was similar to the one used for the usability test. Attention was given to reduce the presence of metals and other electromagnetic devices from the test work area. But similar to the case in usability testing, there were a considerable amount of metals and electromagnetic devices within the *'undesirable'* distance range.

4.2.2 Results

Here we present the results related to filtering and smoothing techniques applied on the raw sensor data.

4.2.2.1 Power spectral density

The power spectral density (PSD), shown in figure 4.2, is formed using x position data of the sensor. As described in the last section, we were running the data acquisition program on the XY-plane. Thus, most of our activities were either on the X-axis or Y-axis. We have observed that both of the x and y position data produced a similar PSD graph.

As can be seen in the PSD graph (fig: 4.2), 3Hz seems to be a good cutoff frequency. But to make sure that this frequency is good for the working device, we tested the equation it produced in the second layer of the data acquisition program. The process to derive the lowpass filter is already given in section 3.4. To determine the values for the constants we needed to provide the sampling

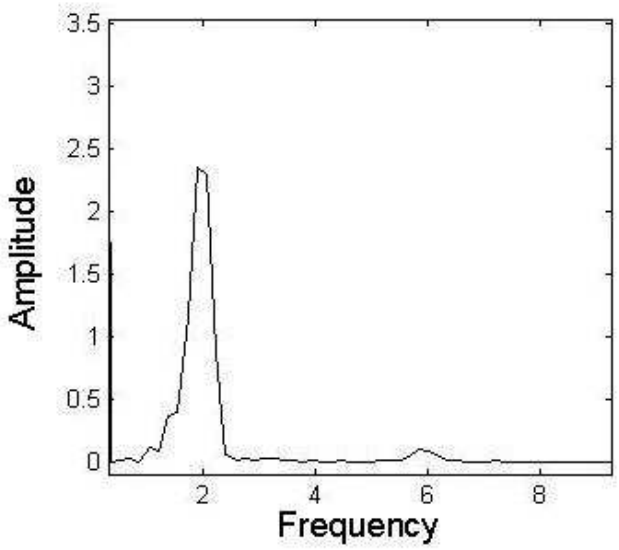


Figure 4.2: Power spectral density of raw sensor data.

rate and cutoff frequency. Even though, the sampling rate or sampling frequency for Wintracker is 90Hz for one sensor, in data acquisition program we polled the device at 1Khz. So, the sampling frequency used to derive the equation is 1000Hz and cutoff frequency 3Hz.

We also tested other cutoff frequencies. Starting from 1Hz to 7Hz including the half points (like-2.5Hz), we have tested them using the above mentioned program. We have seen that as the cutoff frequency increases from 3Hz, the filter output starts to include more of the rough edges seen on the sinusoidal wave, meaning the high frequency noises, making the cursor jump around. On the other hand, as the cutoff frequency goes lower than 3Hz the sine wave gets smoother and flatter; that is, starts to loose some of the actual values. Table 4.1 shows the constant values needed to define the filter for 2Hz, 3Hz and 4Hz cutoff frequencies. In the next section the filter output for these cutoff frequencies are given.

4.2.2.2 Lowpass Filter

In this section we present the output of the lowpass filter using 2Hz, 3Hz and 4Hz cutoff frequency. For test purposes the filter is applied on the x position data only. Y positional data will produce similar results. Figure 4.3 shows that 2Hz cutoff frequency smoothed out the input signal data too much and in some cases reduced the actual input values. With a cutoff frequency of 4Hz the input

Table 4.1: Constant values for the Butterworth lowpass filter at given cutoff frequencies.

Cons./Cuttoff freq.	**2Hz**	**3Hz**	**4Hz**
b_0	35	80	134
b_1	71	161	268
b_2	35	80	134
a_1	18230	17352	16482
a_2	8376	7666	7016

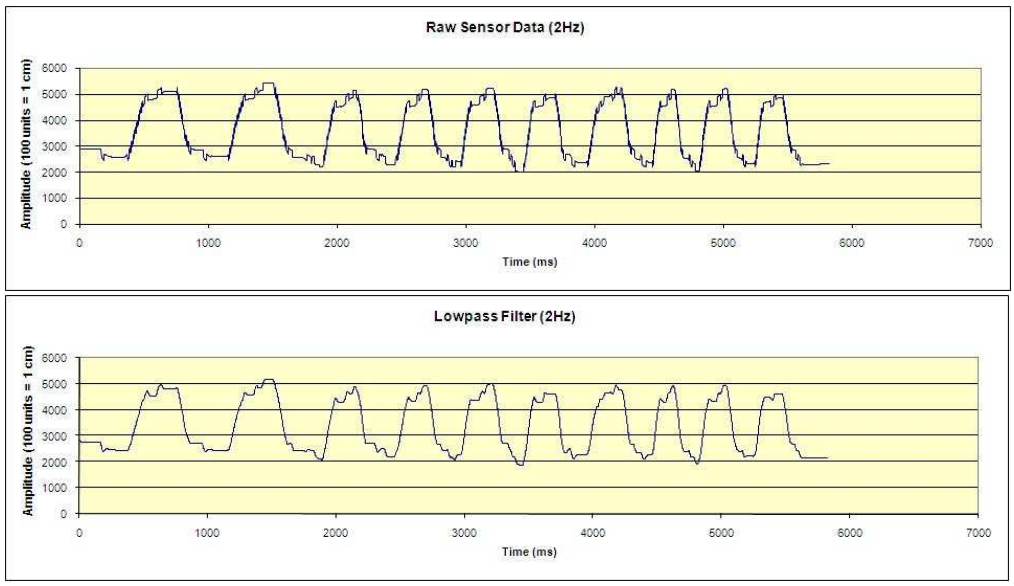


Figure 4.3: Raw sensor data and effect of 2Hz lowpass filter on the data.

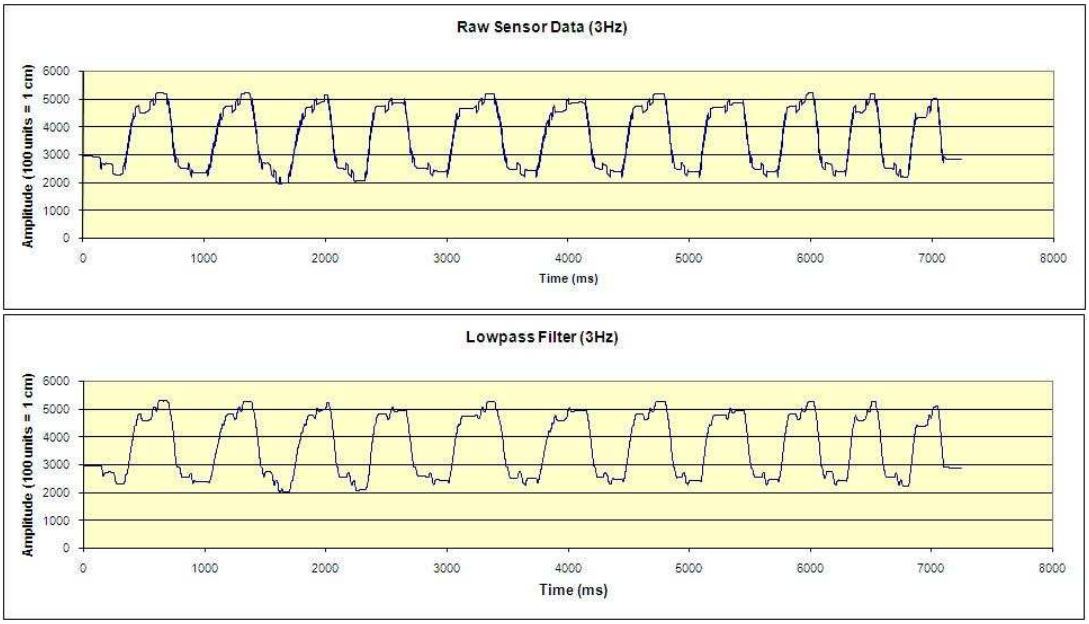


Figure 4.4: Raw sensor data and effect of 3Hz lowpass filter on the data.

signal still contains some noise elements, as shown in figure 4.5, which could be eliminated by lowering the cutoff frequency. Figure 4.4 shows the effects of a 3Hz cutoff frequency lowpass filter, which we found to be the best choice for the input signal.

4.2.2.3 Averaging Filter

Even after performing the filtering, as above, we found that there were sudden and unexpected cursor movements. The jitter filter, as described in section 3.5.3.1, helped to reduce the small jumpy movements. Still, there were some infrequent movements with considerable amount of displacement from the intended position, most often in the opposite direction. We tried to solve the problem with a mechanism, which tracked the direction of the cursor for the last two positions and checked the current movement direction. If the current movement direction is not the same as the last movement then it waits to see the next movement direction. This mechanism does not allow the cursor to move if two consecutive movement directions do not match. But, this mechanism did not work. We found that the unacceptable movement is not a single movement, rather, several movements in that direction.

We also tried many other alternate solutions as well as different versions of averaging before coming to try the currently implemented version of averaging.

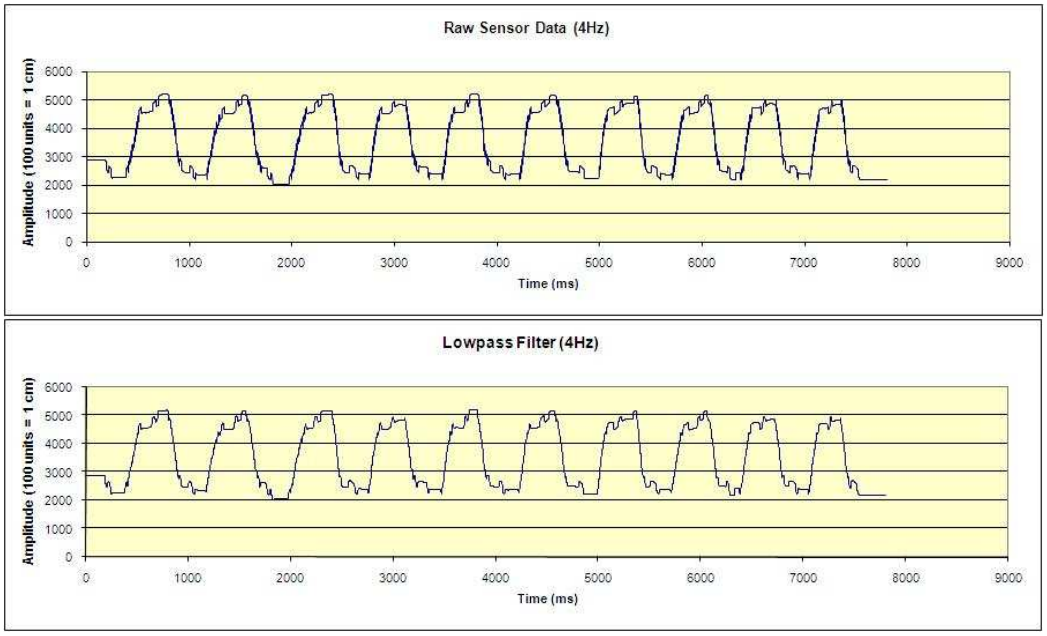


Figure 4.5: Raw sensor data and effect of 4Hz lowpass filter on the data.

As mentioned in section 3.5.3.1, this is a sliding window averaging, meaning the number of samples to average stays constant but the set of values to average changes with each new lowpass filter output value. Say, if the sample to average is 25 then each time it calculates the average on last 24 filter output values and the current filter output, i.e., 25 filter output values, plus the last averaged output. Each time the oldest value from the set of last 24 filter output is dropped and current filter output is add into the set for the next calculation.

We tried with many different values for the number of samples to average. We started off with this value being as low as 3. But, found out that values below 10 do not have any impact on the cursor movement. In fact, this value has to be above 15 to see any kind of changes to the cursor movement. Figure 4.6 shows the effects of the number of samples to average, the value starting at 15 is increased, in steps, to 40. As the number to average increases, the signal or position data starts to become smoother. But, it is observed that as this number is increased, the cursor movement starts to show a time lag, i.e., the display cursor moves for a noticeable time after the sensor movement.

This value will also depend on the environment, i.e., on the amount of electromagnetic distortion in the work space. If the distortion level is low, than the number involved in that average can be small and if the distortion level is high then a higher number will produce a satisfactory result. As mentioned earlier, in our environment we found 25-30 to provide satisfactory results.

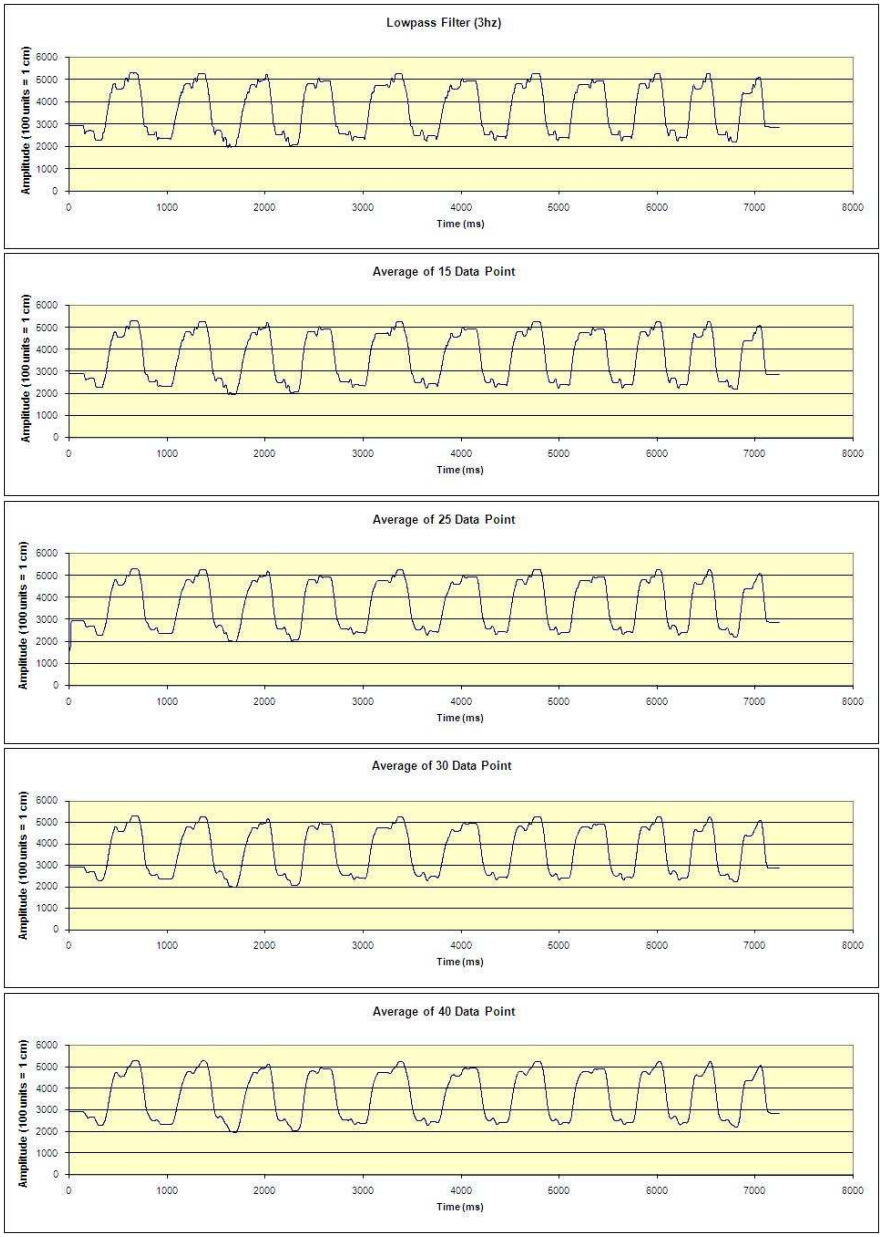


Figure 4.6: The effects of averaging of 15, 25, 30, and 40, 3Hz lowpass filter data points.

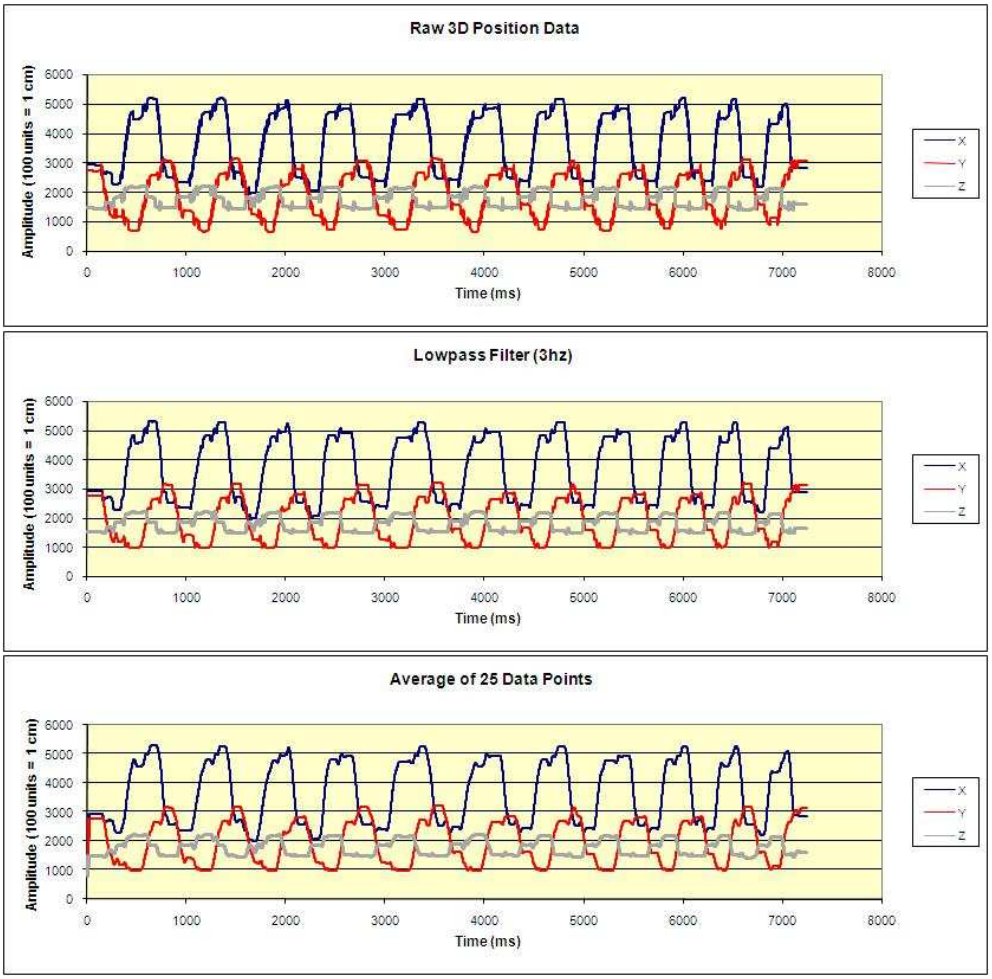


Figure 4.7: The effects of the lowpass filter, jitter filter and averaging on raw sensor data. Rpen implements this scheme to filter out noise and smooth filtered data.

4.2.3 Summary

Figure 4.7 shows the effects of the currently implemented filtering and smoothing techniques on senor's raw 3D positional data. As mentioned in the last section, before arriving at the current scheme we tried many other variations. The majority of the variations included averaging and jitter filters. Currently, we only consider small movements of less than or equal to 0.3mm as jitter. But we also implemented a filtering scheme where we also rejected sensor movements over a certain displacement value. But, as the sampling rate of the hardware is slow, most often the hardware produces big movements. Because of this, the jitter filter having an upper limit, failed to produce satisfactory results.

4.3 Computation Time and Roundoff Errors

This section describes the computation time required by the rpen device driver at each sensor data cycle. It also describes the roundoff errors made in calculation, for not being able to use the floating point data structure. It also provides the data aquisition process.

4.3.1 Test Setup

This test was conducted to learn about the computation time and roundoff errors inside the rpen device driver. It is mentioned earlier that it is important for an input device driver to complete its operations, on a single data feed from the hardware, within a very short period of time. It is observed that most of the regular input devices, like - mouse, joystick, touchpad, touch screen, etc, do very little inside the device driver, which only interprets the input data format from the hardware and reports it to the input event drivers. But, in our case, we must do the filtering and smoothing on the input data and then using the rframe, interpret the input data to produce input events. Thus, it is important for us to analyze the computation time taken by the rpen device driver on each data read.

As already mentioned in chapter 3, we are required to roundoff some calculations inside the rpen device driver. As the Linux Kernel does not support floating point computations at the kernel level, all the variables used inside the device driver are integer data type. Here we tested the impact of roundoffs inside the device driver.

The data acquisition program used for filtering and smoothing (section 4.2), was also used to test the computation time and examine the roundoff errors. The test environment was similar to the earlier test. The data acquisition program was updated to include the transfer function (described in section 3.5.3.1). This function is same as used inside the rpen device driver. The only difference in

the transfer function is that it does not report input events to the input event drivers; rather, outputs the events in an ASCII text file, simulating the calls to input event APIs.

As we have used the transfer function to test the computation time as well as roundoff errors, we needed to use rframes. We used reference rframes inside the data acquisition program. Calculations involved in all types of rframe other than the generic type, regardless of the size of the rframe, is similar; thus, they will take the same computation time. But, roundoff errors might increase or decrease depending on the size of the rframe. Thus, to evaluate the roundoff errors affected by the size of the rframe, we used three different rframe sizes, with respect to the set display; one rframe smaller than the display resolution, one equal and the other rframe is bigger.

We wanted to investigate the computation time in three stages. As the filtering and smoothing part of the device driver will change with the use of hardware, we separately calculated the time that this segment of the code required. Then we investigated the time taken by the transfer function and lastly, the total time taken with each execution of the input data.

Roundoff is done at two stages of the device driver. One in inside the lowpass filter and the other is during the transformation of the position, in terms of rframe, to display the cursor position. So, we investigated the roundoff errors at two stages: the lowpass filter, and the transfer function. As mentioned above, to check roundoff effects with change of rframe resolution, we examined this using three different scenarios.

The computation time and roundoff error analysis for the rframe types except generic rframe type, will be same. As the generic rframe is partially implemented we will not present its computation time or roundoff error analysis; however, time complexity analysis will be given for the generic rframe. In terms of roundoff error occurrence, the generic rframe type will have the same problems as the other rframe types have plus few extra. We will identify these places where the extra roundoff errors will occur for the generic rframe.

4.3.2 Computation Time and Roundoff Error Results

Here in this section, we present the results of time taken by the device driver on each data read cycle as well as the results of the roundoff effect at each stage.

4.3.2.1 Computation Time

The time complexity of all the functions on each generation of the sensor data is linear, i.e., $\Theta(n)$. This includes the filters, averaging and transfer function. From the equation given in section 3.4, for filtering and the calcualtions done inside

the transfer function, as descibed in section 3.5.2 and 3.5.3.1, we can see that they run in linear time. Even though the averaging involves a for loop, it has a constant number of iterations which can be at most 125. Linear time complexity is true for all types of rframe including generic. The generic rframe does involve many more calculations and decision making (as described in section 3.5.2) than the other types but still takes linear time to compute the events.

We tried to find out the computation time for the filters and smoothing, transfer function and over all, from getting data from the hardware to reporting of the events. A Intel®Core™2 Duo machine with 4GB RAM was used. *'Clock()'* function provided by the C library was used to record the execution time. Events generated through the use of rframe on the XY-plane failed to register any significant time on the machine clock. Thus, it can be generalized that, even though, generic rframe involves more calculations then the other rframe types, total computation time to produce events on the generic rframe will not be significant.

4.3.2.2 Roundoff Errors

There is a significant roundoff effect on filtered data. The reason for this effect is that the roundoff error in many cases propagate to future filter values. Figure 4.8 shows this effect. The graphs are produced using the x position value only. The same raw data is filtered with two version of the 3Hz lowpass filter. One does the roundoff, as done inside the rpen device driver. The other utilizes the double (float) data type to compute filtered values. This version also stores past values in double data type. It should be mentioned here that, roundoff in this case means getting the integer value, i.e., ignoring the decimal value, even if it might be 0.99.

Though, in many cases these two versions of lowpass filter produce similar data points, in some cases they are very different. It is seen that the roundoff version of lowpass filter very closely resembles the raw data, filtering out the noise. Whereas, the filter using floating point, in some cases distorted the input data itself. It can be concluded from this experimentation that doing roundoff is producing better filtering results than the floating point version.

Roundoff effects on transformed data are shown in figure 4.9. The transformation process involves a single division operation and the produced value is not propagated to produce future values. Thus, it is found that the roundoff errors produced in this stage, are not that significant. It is also observed that the size of the rframe does not greatly influence the roundoff errors. Figure 4.9 shows that the x cursor position produced by the floating point version closely follows the cursor position produced by a roundoff version. It is also very similar for all three rframe resolutions.

The generic rframe calculation differs from other rframe types in the transfer

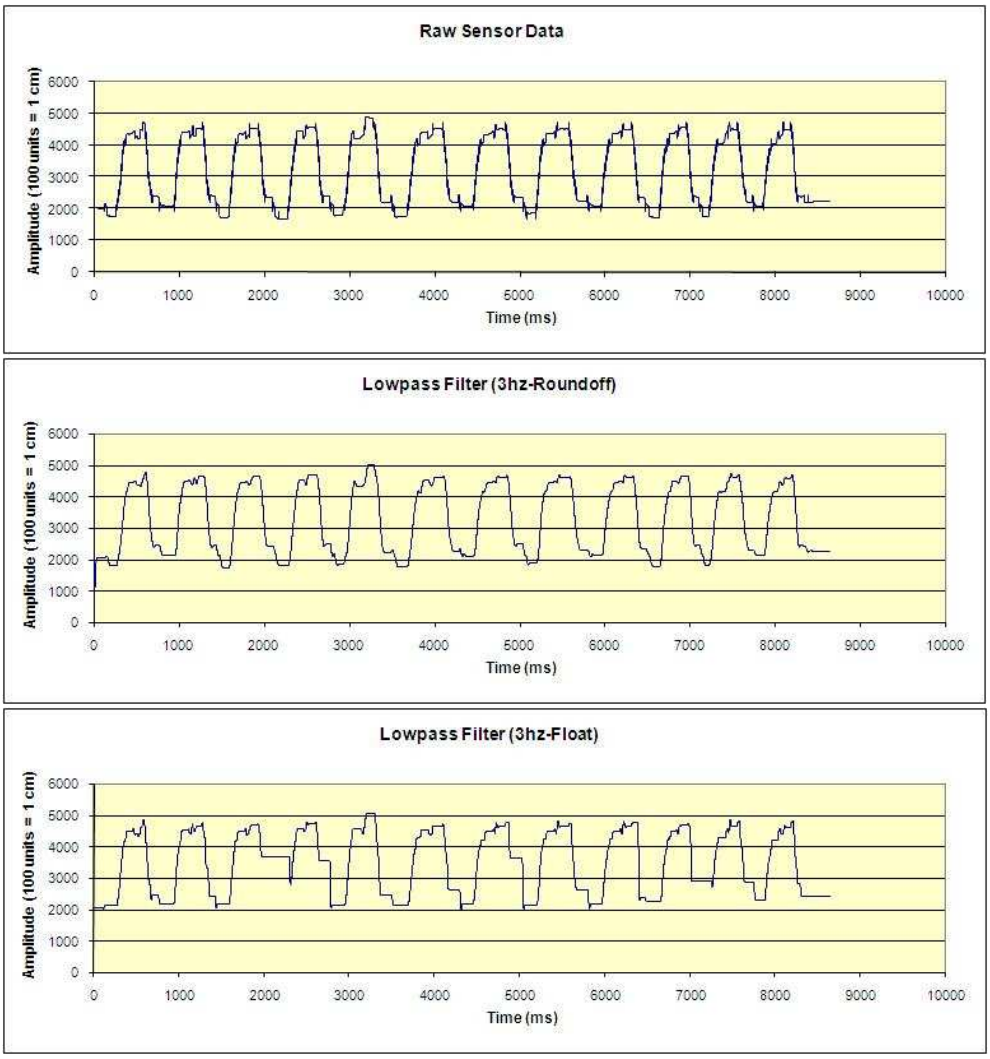


Figure 4.8: The effects of rounding on filtered data.

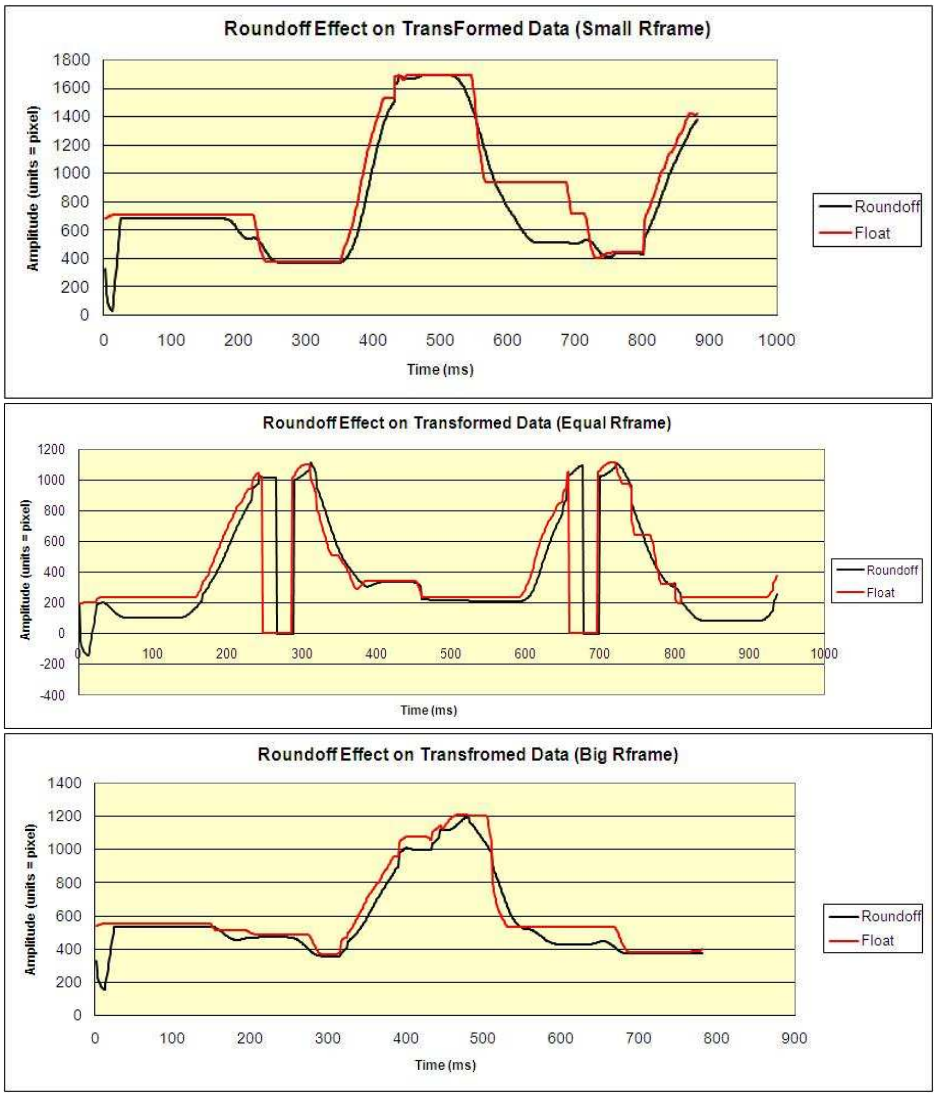


Figure 4.9: Graphs showing roundoff effect on final 3D cursor position data. Also showing roundoff effect depending on rframe resolution. The first one uses a smaller rframe in relative to display resolution. In the second one, a equal sized rframe is used. In the third scenario, a bigger rframe is utilized.

function. The generic rframe utilizes more divisions on top of the one utilized by the other regular rframe types. But, these divisions are not used to produce future cursor positions. The generic rframe also uses a square root function to produce distance. So, if all the divisions used by the generic rframe are added up, it might prove to be significant. As the generic rframe is not fully functional in the current version of rpen we could not test this theory in this study. But, a future study can test the significance of roundoff errors in calculating the cursor position information for the generic rframe.

4.4 Observations on Rpen Use

Before we discuss user feedback from the usability test, we present some of our note-worthy observations here in this section, from the perspective of a designer and developer of rpen. These observations are on the usage of rpen, noted during the development and testing period. Some of these observations are confirmed by the usability test, whereas, some others are disputed, and some observations are not noted by the users. The following are the observations:

1. Rpen is a pointing device; so, it has to stay close to the host system, particularly the monitor. Also, the transmitter and receiver themselves generate some noise. An ergonomic computer environment will also have a revolving chair and height adjustable keyboard tray. Some, if not most, parts of these objects are metal. So, it is very difficult to avoid metal and other electromagnetic distortions surrounding the rpen. As such, it's unlikely a device sensitive to metal like the Wintracker would be practical for use in a general office environment. Likely a device using radio triangulation would fare much better.

2. The wire attached to the sensor is not only a source of distortion but also an obstacle to comfortable and smooth use. For better usage experience, the wire had to be wrapped around the hand.

3. The shape of the sensor is not comfortable to hold in one's hand, especially for an extended period of time. Because of the way rpen works, it is better to hold the sensor attached to a finger or to hold it with two fingers. The shape and size does not permit us to attach it to a finger and holding it with two fingers might be a concern for some users. It is important to keep the sensor up-right, not tilted, while hovering and not going for generating click events.

4. It may be harder for some users, particularly initially, to work with the rframe's threshold concept. A visual feedback is very important is this case.

The current visual feedback via cursor appearance may not be helpful to all users, because of the size of the cursor.

5. Due to electromagnetic distortions, rframe boundaries, most noticeably the thresholds, are not linear or not on a flat plane. At places where the distortion is high the threshold height will not be same as elsewhere.

6. Electromagnetic field distortion makes the cursor movements bumpy at places, i.e., might create unexpected jumps. To reduce this effect new users will want to increase the averaging or smoothing effect. This smooths out the cursor movements but creates time lag, and small adjustments in movement making tougher. But, as the user gains experience with the device, s/he prefers less averaging. The observed range for the number of samples to average was within 20-50 sample points.

7. The time taken to learn to operate the device is relatively short. The above observation can be taken as an evidence of this. In a relatively short session of operation, a new user gains enough confidence to cope with the unexpected jumps. It is also observed that users, in most cases, can successfully retain device operation instructions.

8. The C-D gain (discussed in section 2.2.1) for a relatively smaller rframe, in respect to the display resolution, is noticeable and sometimes gets difficult to operate.

9. Handedness does not seem to be an issue with the rpen. While being right handed, operating the device with the left hand was not much of a challenge and rather preferred at times.

10. Instant transition of operation between one type of rframe to another, is not always smooth. It takes some time for the user to become accustomed to the new working plane.

11. At times, a physical reference of a rframe seems necessary for rpen.

12. Where to place an idle sensor is an issue with some types of rframe. For example, for a rframe on the XY-plane, when the XY-plane is the desktop, the user might choose to place the sensor on the desktop inside the working range, which could be a left click state, or outside the working range.

13. Operating on a large enough rframe while the user is sitting or operating within a confined place, may result in an inability to reach the whole working surface with one hand. These are the places where we confronted the issues discussed in observation 'number 9' earlier. This scenario also validates the point to extend Fitt's law for 3D, discussed in [MI01]. It is argued

that pointing to an object does not only depend on the distance and size of the object but also on the direction of movement.

4.5 Usability Test

This section presents the usability test results. The usability test environment is also described here.

4.5.1 Test Setup

In chapter 3, section 3.6, we have already discussed the goals and objectives of the test. We also discussed the participants and what information they were provided concerning the device as well as the test. Here we will be discussing the test setup and test environment.

The full test was conducted in a single day; participants completing their tasks and answering the questionnaire. Each participant was given roughly half an hour to complete all the tasks. Even though, each participants were allocated an hour for both doing the tasks and providing feedback, users' feedback was not that strongly controlled and they took their time to write the feedback without any interaction or influence of the test moderator. Users were given the questionnaire after they had completed the given tasks.

Figure 4.10: Usability test environment.

The test was conducted in a normal office room environment. Attention was given to remove any unnecessary metal or any other electromagnetic devices. Participants were also advised not to have such objects with them. Even then, there were considerable amount of metals around, which is normal for any office

environment. The chair that the participants used was metal based and most often close to both the transmitter and receiver. The distance between the chair and transmitter was on average 0.5 metre. The sensor/receiver distance varied from more than a meter to around 0.3 metre. There were a few metal frames and screws under the table, which were at times close to the transmitter than the receiver.

Other than these, the computer monitor and keyboard was placed close to the user, meaning close to the transmitter and receiver. The distance between monitor and transmitter was around one metre and the distance between the receiver and the monitor ranged between 1.75 metre to 0.5 metre. There were also a few other metal based accessories and electronic devices that several participants possessed. The room was lit with florescent lighting as well. All in all, there were a considerable amount of metal and other electromagnetic devices around, that did not follow the rules given in [NMF98] [VR 07], to reduce magnetic field distortions.

Figure 4.11: Figure showing placement of different equipment and devices used in the usability test.

The transmitter was placed under a table and the sensor operated over the table, as seen in figure 4.11. There were five pre-defined rframes used for the usability test. All but one, had physical reference. Most of the basic tasks were carried out on a 500x400mm rframe defined on the XY-plane. The default display resolution was 342x271mm. There was one rframe defined on the YZ-plane, which had bigger resolution then the default display screen. One rframe, on the XZ-plane, had the same size as the display screen. Another rframe, defined on the

XY-plane, had a smaller resolution then the default display resolution. The task list and related instructions are given in appendix A.

4.5.2 Results

Accuracy or correctness is an important aspect for any human computer interface. It not only influences the user's anticipation towards the interface [SP05] but also in case of interaction device, like the rpen, might be a source of frustration. Absence of frustration is an important element of usability of an interface [RC08]. A frustrated user is never satisfied with an interface, even if, it might be able to do important and relevant tasks for the user. The feedback for such an interface will always be a little biased, and understandably so.

This was the issue we ran into when we conducted the usability test on rpen. Even though, we prepared the users to anticipate a device which is not precise at times, the feedback is evidence that users cannot fully appreciate the concept we tried to present with imprecise hardware. Coupled with this, there were few legitimate human factor and design issues raised, which we wanted to identify, but was also part of the problem which created dissatisfaction among the participant users. The obvious outcome of this scenario was a unanimous recommendation to change the hardware device. It is understood that in a normal setting it is not possible to eliminate all metals and electromagnetic devices from the surrounding work area. Also the calibration techniques used with electromagnetic devices have its limitations, as discussed earlier. So, an important outcome of the current exercise is that an electromagnetic device is not a good choice for rpen-like general purpose pointing devices, where precision and registration is a big issue.

Even with the limitations, the usability test was able to provide us with some other findings which confirmed some of our observations. All the users found the device very intuitive. It was interesting to see that all the users including the three first time users were able to operate the device almost immediately and completed most of the tasks successfully. As mentioned in section 3.6, most of the tasks in the test were performed on a rframe defined on the XY-plane (representing desktop). It was done so to give the users a frame of reference and to rest the arms on the desktop, if required. But, it was observed that most of the users found the rframes defined on the XZ-plane, parallel to the monitor display and the YZ-plane, wall projection, more intuitive and opinionated to have better potential. Some user also suggested that the rframe on the XZ-plane could be defined close to the monitor, if not for the hardware limitation, and could work as a touch screen.

From the usability test it was evident that time to learn how to operate rpen is very short, which also says it is intuitive. The users could also remember most of the operations after a considerable amount of gap between the usages, which

is known as the retention time. Both of these are important for an interface to be usable. But, there was one feature that the users find a little difficult to remember, and it was not intuitive as well. The feature to tilt the device more than 8.0°on left or right to show intention of left double click and right click, was making users experience of the device uncomfortable at times. Spending time with the device does reduce this occurrence but it was also found that this feature forces the user to hold the sensor in a rigid position, which may not be a natural or comfortable posture to maintain. This does have a potential to create strain injuries for the users. This is a limitation of the current hardware, not the rpen concept. A production version would have a button for right-clicking, and a more accurate and faster sensor would allow the user to perform double-clicks by two dips below the threshold. This would eliminate the need to rotate the device.

We used a physical representation of the predefined rframes used for the usability test. Based on our observations we thought a physical rframe would increase users' satisfaction level. But most of the users commented that they did not even look at the physical rframe while operating the rpen. They were getting their feedback from the display and cursor movements. So, to the users', physical presence of an rframe is not important. This feedback might have to do with the tasks that were assigned to them. Indeed for simple navigation, select, drag and drop and even for text cut and paste, a mouse user never looks at the device or the location of the device, they just operate it looking at the display. But for some other minute operations and 3D object manipulation a physical presence of a surface is important. It is argued in [HPG94] that a user needs a spatial reference while working with a spatial device.

Rframe's placement on the space does not impact the operations of the device. It is found in the test that as long as the user can view the display comfortably, the rframe's position did not bother them. But the size of the rframe is an important factor. Users' confirmed our observation that a rframe smaller than the display is harder to operate on. Especially with an imprecise device, it sometimes gets impossible to do some of the tasks. The movement gain increases as the rframe size gets smaller in respect to display. There needs to be a mechanism to reduce this gain when working with a small rframe. Similarly, the user also expressed that the footprint of the device is too big, comparing it with mouse, a relative pointing device. This also could be solved if the gain for the smaller rframe can be controlled. This might also be solved by using a sensor that is sampled at a much higher rate, allowing the visual feedback on the screen to match the actual movement of the device.

Adequate visual feedback is very important for successful manipulation of the rframe's surface layers. Usability test also came to this same conclusion. The users seemed to have accepted the surface layers but expressed current visual cues

are not noticeable enough; a better visual feedback mechanism is needed. One of the options expressed is to give height level indicator as part of the cursor.

The wire attached to the sensor, was creating discomfort for the users. As well, the wires themselves are also a source of noise. A wireless sensor was preferred by most of the users. They also recommended a shape change for the sensor. Rather then a cube, which is the current shape of the sensor, a stylus shape would be preferred. It also should be mentioned here that, a stylus is better at pointing a smaller object, whereas, it is difficult to point to a miniature object with a cubed shape pointer. A stylus gives a better affordance then a cube at pointing. Also, a pen like stylus is more comfortable to hold for longer period of time then a cube. Of course, we always intended to use a stylus, but unfortunately the Wintracker only provided cube shaped sensors.

As Zhai in [Zha98] found out that fatigue is a huge problem for a free moving spatial device, the rpen users also expressed the same concerns. Prolonged use was difficult for the users, and they needed to rest their hands often. As the sensor has to be held in the air, its use was strenuous. This would be a huge factor for the older users, where their usage time could be very limited. Even if with the rframe on the desktop, users felt stress on the arm muscles and shoulder muscles. So, unlike the mouse, which mostly uses wrist and finger movements, rpen may not have Carpal Tunnel Syndrome, but might create other types of repetitive strain injures. To counter this effect, the rpen has been implemented with a clutch mechanism, as suggested in [HPG94]. This helps the user to put down the sensor, without affecting the cursor movement on the display, to rest his/her arm. It was also observed that operating smaller rframes were less strenuous than the bigger ones.

It should be mentioned here that an accurate, stable, sensor would have allowed us to define an rframe parallel and just above the desktop table, allowing the user to rest their hand. The Wintracker (hardware chosen for current version of the rpen) did not allow us to define such an rframe, hence the user had to hold their arm above the table to access the threshold.

Due to the hardware limitations, the current version of rpen was found not to be sensitive to small movements. This is not solely due to the effect of noise in the hardware; rather it is also due to the mechanisms we adopted to counter the noise. The jitter filter eliminates the very small movements, whereas, the averaging, smooth out all the movements, whether it be noise or cursor movement, over a period of time. Along with the lowpass filter these mechanisms do not allow a specific and precise small movement of the sensor to be reflected as a small cursor movement. Also the slow update rate does play a part in it.

Object manipulation with rpen was found to follow Fitt's law (time required to move the cursor to a target postion is a function of the distance to and size of the target), in terms of the object size. But the distance aspect seemed not

to be so clearly following the law. We observed some users operate sitting beside a rframe on the desktop, having approximately similar sized rframe work surface on the right side and the left side. The users operated on objects on the right side, whether it is far or near, with their right hand and time to reach the object seemed to follow Fitt's law. But, as for the objects on the left side even though it is near to the user, time taken to manipulate the object with right hand is not the same as with objects on the right side. In these cases, where the object was small or far on the left side, users sometimes used their left hand to manipulate these objects. This also led us to conclude that handedness is not a big concern for rpen operation.

Lastly, in the usability test the users expressed a desire to have buttons to generate the click events. There were several causes for this opinion. Firstly, it is difficult for the user, with a spatial device and with its sensor lacking precision, to produce a straight downward movement on the surface of rframe to produce left clicks on the right spot. Not only that, the click is complete when the user comes up over the surface threshold on the same spot. This was a challenge for most of the users. Also, for producing right click and left double click twisting the sensor is not natural and has to be learned and remembered. This twisting was also associated with the tilt problem discussed above. It also means that the users had to remove their attention from the screen to verify that the sensor was tilted from the normal as is required by the click. Having buttons mounted on the device to produce click events will help resolve these issues. Also, it will free up the Z-axis, in terms of display, to do purely 3D interaction and object manipulation.

CHAPTER 5

Conclusions

5.1 Conclusions and Analysis

A new 3D pointing device, capable of performing both 2D and 3D interaction, is presented in this thesis. Developing a 3D interface/device in itself is a difficult riddle to solve. The design task is made more complicated by attempting to make a 3D device operate as a hybrid interface, interacting with both 2D and 3D applications. With an intuitive and natural 3D interaction device, the rpen, our primary goal is to enable 3D tasks to translate to 2D interactions. As this design challenge has been studied rarely before, we adopted the design framework of a 3D spatial input device to analyze the design aspects of the rpen.

The rpen possesses a unique set of interaction techniques. It is a pen-based pointing device, combining the interaction techniques of the 2D mouse and the touch screen. As the rpen is an absolute position device, it resembles a touch screen more closely than the mouse. But unlike the touch screen, the work surface of the rpen can be any arbitrary 3D flat space with any size and orientation. This means the work surface of rpen can be as small as the mouse uses, and also, the rpen can work as both direct and indirect input device. This allows the rpen to harness the advantages of absolute positioning (precision and naturalness).

The rpen concept is independent of any motion tracking system. The only requirement rpen imposes on the hardware is that it should be able to produce absolute position information within a relatively big working volume. High accuracy, high resolution, low latency and robustness are the performance measurements for the motion tracker. The rpen requires a reasonably, moderately high performing hardware within a practical price range. Within these set boundaries, our analysis led us to conclude that an off-the-shelf AC electromagnetic (EM) motion tracker is the most workable choice for the rpen pointing device. Unfortunately the EM motion tracker we tested proved to be noisy and very sensitive to metal objects commonly found in office settings. It thus was found to be a poor choice for the rpen in such a setting.

The trend of input pointing device has been shifting from technology centric design principles to more human centric, ergonomic design systems. The rpen builds on the natural skills of humans to operate a pen as a writing tool, as well as a pointer. Similar to a regular pen, the rpen also operates in 3D space.

Being a 3D spatial device, the rpen needs to address the design challenges of a spatial device. Through hardware design (ergonomic and comfortable pen grip, balanced and light weight pen shaped hardware) and software interface (clutch mechanism, support for all input states and multi-sensory feedback) we seek to solve these challenges. Lastly, through comparative analysis we showed that the rpen's interactive techniques are unique and none of the contemporary input devices, with similar 3D tracking system, use the work surface the way that the rpen does.

We chose to use the Linux platform to implement the software interface of the rpen. Linux as open source software provides ample support for an efficient implementation of the rpen. Due to the choice of hardware, Wintracker II from VR-Space Inc. (an AC electromagnetic device) it was necessary to implement a poll-based USB input device rather than an interrupt-based one, which is usually the norm for rpen-like input pointing devices. To utilize the efficiency of a kernel module as opposed to a user-space process, the rpen device driver was implemented as a kernel module. But in doing so, the opportunity was lost to use floating point operations inside the device driver.

Initially, we were not expecting to spend a substantial amount of time on noise filtering and smoothing. Due to the characteristics of EM devices, a significant amount of time was needed to find an acceptable filter with the correct cutoff frequency. Upon consideration of the environment in which we envisioned the rpen would be operated as a general purpose device by regular users, we excluded the option to employ calibration techniques which are normally used for EM motion trackers. As such, we went for the classical lowpass filter and sliding averaging to smooth-out the jittery sensor data. Through experimentation we came to the conclusion that a combination of lowpass filter with 3Hz cutoff frequency, sliding average using 25-30 filtered data points and a jitter filter of 0.3mm yields the best smoothed output for the sensor data in a normal office environment.

We used a transfer function to translate the sensor's absolute position data in reference to a defined rframe, to a cursor position on the screen. We found that this does not take significant amount of time to produce the output, however, it does incur significant roundoff/truncated error. The filter equations also accumulated considerable amount of roundoff error. Through experiments, we concluded that the roundoff data produce better results than the data produced using floating point. It is also observed that the size of the rframe does not greatly influence the roundoff errors.

Through a user space application, the user is given the opportunity to define any number of rframes although only one rframe is active at a time. Audio-visual feedback about different thresholds and left button click is given through a daemon process. The process also manages the auto adjust mode (adjusting

the rframe location automatically, if the rpen is idle).

A usability test, in a lab setting with five participants, was carried out within a short period of time. Given the limitations of the current version of the rpen, we wanted to find out the positives and negatives of the rpen concept rather than the device as a whole. We also wanted to determine the potential of the rpen concept, in the user's opinion.

The usability test was greatly influenced by the lack of accuracy of the device, due to EM distortions. The users were unsatisfied with the hardware's inaccuracy and lack of robustness. Even then they commented that the rpen was intuitive. Especially, the rframes on the XY-plane (representing a touch screen) and the rframes on the YZ-plane (representing a wall projection) were more intuitive and have better potential than the rframes on XY-plane.

The usability test also confirmed the problem of fatigue when using the device for precision work. It was also felt by the users that adequate feedback about the position in 3D space of the thresholds was needed. It was opinionated that a wireless sensor is necessary as the wires were causing discomfort for the user. Lastly, the users felt the need to have buttons on the sensor to perform a left or right click would be an improvement. However, it is likely that they would not wish this for the left click if the hardware had been faster and more accurate.

The current effort is a proof-of-concept. Even with its short comings due to the motion tracking hardware choice, the rpen is able to create interest and excitement amongst its users. With a more precise and accurate motion tracker and appropriate pen-based hardware design, the rpen is expected to become a viable alternative pointing device, a bridge between the traditional input devices and their futuristic counterparts. Also if the extensions proposed in section 5.3 are to be included in the rpen interface, it will become a very powerful input device. It has the potential to address most of the usability and ergonomic concerns with such spatial devices. For example, Zhai's concern (mentioned in section 2.3) of lack of coordination can be addressed by the distant pointing and C-D gain mechanism. The C-D gain mechanism and the moving rframe relative to the user can also help in reducing user fatigue related issues. All in all, the rpen has great potential to be a highly appreciated 2D and 3D hybrid interface, using the human's natural skills and knowledge to interact with 3D objects.

5.2 Discussions

5.2.1 Contributions

The thesis presents a new concept for a pointing device. Through implementation we have proved that such a device is possible and through usability testing and

literature analysis we have showed such a device does have great potential. Future studies can be undertaken to reveal the full scope and limitations of the device and produce a mature product in the process. Here we list the major contributions of the current research:

1. A new pointing concept is presented. A proof-of-concept device is also developed to show that the concept is realistic.

2. The new device utilizes both 2D and 3D interaction concepts, working in the 3D world, i.e., provides 3D input data. The device is intentionally designed to work as a 2D point device using 3D interactions. It also is capable of working with 3D applications.

3. A new pen-based hardware design is proposed as a 3D pointer. It basically presents the pen-based pointing device in a new way, trying to create a more ergonomic device.

4. A byproduct of the current study is that electromagnetic trackers are not suited to work as a motion tracker for rpen-like generic pointing devices.

5.2.2 Implications and Applications of Rpen

The obvious use of the rpen as is a pointing device. But, the implication of the device is more far reaching when we consider how it can be used. If a link is made with some of the straight forward extensions, listed in section 5.3, to the current version of rpen, then the possibilities are numerous. In the following are listed some of the note-worthy applications of the rpen:

- With the rpen, the user can define a touch screen anywhere, and only be limited by the hardware. The touch screen can be of any size. Also, the touch screen can be in any orientation, horizontal, vertical or titled. A drafting board, drawing board or sketch board can be a touch sensitive surface. The user can paint, draw or write on the virtual touch surface. When a multi-user extension is added to the device, these activities become multi-user enabled. Also, as the rpen's touch screens are not literally a touch screen but just function like one, they do not have the same problem as the regular ones have such as the requirement that the user can physically touch the screen. For instance, the screen could be behind protective glass, or across the room.

- The basic idea of the rpen is software controlled. The two modes (normal and auto adjust) that the current version of rpen has, is configurable with the software tool developed for rpen. So, multiple input modes can be implemented through the software interface and the rpen can switch between these modes seamlessly.

- The work surface that is defined by the rpen is basically a 3D rectangle. So, with appropriate applications, 3D object manipulation is possible.
- Gestures can be interpreted through software applications. So, with the use of appropriate software, the rpen could be used to generate user gestures. Not just the 2D gestures that mice and touch screens are capable of, but more powerful and intuitive 3D gestures.
- The rpen can also work with multiple screens forming a single display. This feature is basically implemented through the operating system not the input deivce itself. But, for the rpen, it can utilize all the screens as a single large touch sensitive display.
- With a C-D gain control mechanism as well as acceleration information derived from the sensor's movement, the device can be used to demonstrate object manipulation following the laws of physics.
- If the distant pointing feature (the term is explained in chapter 2) is added to the rpen, the input device can be used to work with large scale display-systems, even from across a room.
- It can be used as a direct input device like a touch screen or indirect device like a mouse or joystick and the user can easily switch back and forth between the two approaches.
- The rpen can have a continuous pointing traversal path like in the case of mouse as well as discrete pointing capability (sensor motion is not tracked all the time) as implemented for a touch screen.
- The rpen is a very versatile input device, and can be used in multiple input device modes. It can work as direct or indirect input device, it can also accommodate large screen or distance pointing and it can work in a small area like a mouse or a larger area like a touch screen.

5.3 Future Research and Extensions

This research involves a new kind of generic pointing device. Such an effort usually takes a bigger team, looking at all aspects of the device, and requires more time to investigate the full potential, producing a mature device. The current study is a proof-of-concept. We have just laid out the basic underlining design of the device and shown that it works. There is more work to be done. Here we list some future extensions that could be undertaken:

- **Fully implemented generic rframe**: One of the strengths of the rpen is that, the work surface (touch surface) can be defined anywhere in 3D space with any orientation. The current version of the rpen implemented the rframe on the standard three orthogonal planes. The generic rframe, which allows rframes with any orientation or location, is partially implemented here. An extension would be to complete this implementation and determine its advantages and weaknesses (if any).

- **Implementation with a better suited tracking system**: We have determined that the electromagnetic tracker, at its current state of development, is not a suitable motion tracking system for the rpen. Chapter 2 has presented that most of the current research and product development are moving towards hybrid trackers. Initially, we also attempted to develop a custom tracking system for the rpen using RF technology. So, a future study can involve such a tracking system which will enable a much improved usability test of the rpen concept. Also, many other aspects of the input device, including the effects of rframe size as well as the limit of the size of rframe, could be tested thoroughly without the overwhelming adverse influence of the current hardware system.

- **Hardware design**: In the current study we have only alluded to what could be optional hardware designs for the rpen. In fact, in HCI hardware design is a long process which includes prototype design and usability test encompassing all aspects of human factor issues. The issues with buttons, what buttons to add, how and where to add them, etc., also can be addressed through the hardware design process. To make the rpen input device complete, this exercise needs to be done.

- **C-D gain control mechanism**: Unlike traditional input devices, the rpen device uses variable control gain. When the rframe is equal to the display then it has a C-D gain of 1. In the case of small rframes (relative to the display size) the gain increases and in case of bigger rframes (relative to display size) this ratio becomes smaller. We have mentioned what C-D gain means to an input device in chapter 2. By controlling the C-D gain dynamically depending on the acceleration and deceleration of senor movement on the three orthogonal axes, we can make the user feel as if he/she is in full control of the interaction and thereby become more satisfied. This will also work as a solution for smaller rframes. With the combination of C-D gain mechanism, distant pointing and a custom application depicting the physics of motion, we can generate a real world experience for the user.

- **Solution for threshold detection**: Through the usability test, we have seen that users have difficulty in remembering as well as locating the rframe thresholds. The given visual feedback was inadequate. We think it is an

excellent place where we can provide a combination of haptic and other feedback. As a force feedback mechanism might make the rpen device heavy, we believe a combination of audio-tactile and better visual feedback will increase the user's satisfaction level with the interface.

- **User-space device driver**: The current version of rpen utilizes a kernel side device driver. In chapter 3, we have presented the case for a kernel-space vs. user-space device driver. By implementing a full featured user-space rpen device, a comparative study can be done to make the case with practical and tangible data. This may be a better option for the rpen device driver.

- **EM tracker with calibration technique**: In the current version of rpen we have not used any calibration techniques usually associated with EM trackers. A study can be done to observe the impact of calibration on the use of rpen and also if a calibration process be developed to be easily employed by regular users. A calibrated rpen device with the help of a range extender might be able to produce a stable input device with a large work volume (working space).

- **Developing custom application for the device**: The potential of any input device can only fully be appreciated by a user, after they are shown how the device reacts to applications in many different scenarios. The best way to do that is to build a custom made software application or game, which will showcase the strengths of the device.

- **Multi-user and multi-display**: The current motion tracker used with rpen can support three sensors. In fact, most of the 3D motion trackers support multiple sensor input. So, the rpen input device can easily be extended to be a multi-user input device. The rpen pointing device can also work with a multi-screen display as well as multi-display scenarios.

- **Gesture based input**: Gesture-based input is natural and nowadays most of the 3D input devices are using this technique to obtain user input. Gestures are interpreted by applications. Customized applications can be developed to show the range of gestures that can be conveyed using the rpen device. This will also utilize the full range of 6DOF information, given by the 3D motion tracker. Gestures may also be better utilized if multi-sensors are employed for a single user involving multiple finger motion or multiple hand simultaneous movements.

- **Distant pointing**: We have presented many of the input devices utilizing the distant pointing techniques in chapter 2. This technique can also resolve some of the human factor related issues (like fatigue, restriction of hand

reach). The rpen can incorporate this technique to its existing touch-based pointing mechanism.

- **Making rframe relative to user movement**: In the current version of rpen, the rframe is fixed to a given location, once it has been defined. There is a auto adjust mode, but that is also fixed to a location for the duration while the rpen is active. Many 3D interaction researchers have argued, where a pointing device (the rpen) is interacting in reference to an object (in our case the rframe), both of them should be mobile in reference to each other. Thus, a future study can experiment with this feature by recalibrating the rframe to new location (with same size and other parameters, only the points defining the rframe will change) depending on the user's movement. That is, the pointing sensor and the rframe will be in absolute position in relation to each other but will move in respect to user's movement. The feature can dispose of the need for defining new rframes for each orientation and position, only the size needs to be changed.

- **Working on multiple rframes**: Future versions of rpen can be made to allow multiple rframes to be active at the same time, and rpen will switch between them based on proximity. This would allow the user to smoothly switch between using the rpen like a mouse to using it like a touch screen. It would also allow the user to change to a different context (i.e., 3D gestures only, interface for a specific application etc.) by moving to a new rframe.

APPENDIX A

A.1 Usability Questionnaire

1. How easy/hard was it to perform the given tasks using the device? Why?
2. How easy/difficult it is to point to an object (small/medium/big)?
3. How precise and efficient (time taken) is the device in performing the given tasks?
4. How good is the device in retaining a position [clutch]?
5. Are the operations of the device easy to learn and easily remembered?
6. How intuitive do you think the rpen concept is or could be?
7. How would you like the sensors/transmitters to be shaped? Would you prefer a stylus/pen-like sensor?
8. How important is a visual reference, while defining a rframe? What kind of visual reference and feedback do you think would help a user?
9. Is it necessary to have a physical presence of a rframe?
10. What difficulties do you face operating the device (rpen) in the air?
11. What sort of impact does the rframe's placement have on the operation of the device?
    Note: if the rframe is defined where it is difficult to see the screen/display (not getting instant visual feedback), the device will be unsatisfying for the user.
12. How do you find the experience with rpen by itself as well as in comparison to a mouse?
13. What are the merits and weaknesses of the device and its features.
    Note: needs to find out both for the device as a whole and for the concept (without Wintracker) only.
14. What sort of work do you think could be better performed with rpen and what others would be less preferable/suitable?

15. Is the device operation strenuous?

16. Which devices do you think are similar in operations to the rpen? Are the device operations consistent with other similar devices?

17. What ergonomic issues would arise from operating the device?

18. Can you imagine it as a replacement for other types of pointing devices? Which ones and why?

19. What kind of feedback do you think necessary for these kind of devices? Do you think current feedback is adequate?

20. Difference in experience with small to large rframes as well as different screen resolution.

21. What kind of changes do you recommend and why?

22. Does the experience differ for left handedness and right handedness?

23. Any other suggestions or improvement for rpen.

A.2 User Instructions to Perform the Usability Test of 'Rpen'

A.2.1 General Instructions

Rpen works by translating the single sensor movements of the electromagnetic device, Wintracker II, on the rframe to the screen's cursor movement. A display is actually two dimensional. A rframe surface, which can be smaller, bigger or equal to the display size, is a representation of the monitor display. The third dimension of the rframe is used as the height from the surface. If the sensor is above the surface then it is simulated as regular cursor movements, while if the sensor goes below the surface then it is simulated as a left click (left click of a mouse).

In the current implementation of rpen, a rframe can be defined on either the XY, XZ or YZ plane. The XY-plane corresponds to a work surface on a desktop table. For a rframe defined on the XY-plane, x coordinates are translated to screen width and the y coordinates as screen height. The z coordinates are the height measurements from the surface. Similarly, rframe on the XZ plane, representation of a touch screen, have screen width on the x axis and screen height on the z axis; the y axis is the height measurement from the surface. For a rframe on the YZ plane, simulating a wall projection, the y axis is the screen

width, the z axis is the screen height and the x axis is for measuring height from the rframe surface.

Most of the usability test will be done with the rframe defined on the XY-plane. For users' ease of use a physical rframe reference will be given on the desktop to perform most of the usability tasks. As the users in the usability test will not be familiar with rpen and the concept of a rframe, the physical reference will help the users to remember the location of the work surface, i.e., the physical location of the rframe.

The electromagnetic device, Wintracker II, used for the current implementation of rpen has a base station, which is the transmitter and has one sensor or receiver. The base station always remains stationary, set in a specific direction, and the sensor is moved to produce the absolute six (6) degree position information of the sensor relative to the base. Due to some limitations of the hardware device, it is assumed for our implementation that the base will always be placed behind and under the defined rframe. This means the rframe can only be defined within the coordinate sector where the values of x, y, and z is positive.

The sensor has an operational range, which is 0 to 75cm. But the sensor's data become distorted if it is too close to the transmitter as well as too close to the edges. Thus, the rframe has to be defined at least 15cm away from the transmitter and the maximum input position for the rpen is set to 74cm.

The sensor's position data can be distorted by any presence of metal or anything that can create an electromagnetic field. Thus, the usability test will be conducted in a place with a minimum amount of metal and electronic device presence. But, the wires attached to the transmitter and receiver creates some distortions which can not be avoided. Also the update rate of the hardware is very slow. Users will have to keep in mind these limitations of the rpen due to the present choice of hardware. With a better sensor or tracking device the response will be more accurate, fast and smooth. So, the user should not judge the method by the sensor limitation, but instead, focus on the concept, as the sensor is not integral to the method.

A rframe is a cube rather than a surface. The three points that defines the boundary of the rframe work surface is the representation of the display screen. The user can also define a surface threshold which gives the rframe a three dimensional surface representation having a depth. Below this threshold the left click is initiated. There is no limit, rather than the maximum range limitation of the sensor, of the depth below the surface threshold. Above the surface there is a layer signifying the safe traversal region, i.e., here the user is not too close to clicking by accident and also not too far from clicking if required. Above this there is a layer which extends to 60 mm above the surface. Beyond this threshold the sensor is not tracked. This works as a clutch. Other than the clutch threshold, all the parameters are defined by the user from the rframe

Definition Application.

The cursor appearance on the screen provides feedback of the location of the sensor in terms of depth from the surface of the rframe. A left tilted cursor appears when the sensor is above the surface proximity threshold and under the clutch threshold. Above the clutch threshold it takes a diamond shape. Above the surface and within the surface proximity threshold, the cursor has the appearance of a double sided arrow. The cursor appears like a downward arrow when it is under the surface and above surface threshold. The cursor appears like a dotted box, when it is under the surface threshold, i.e., when the left click is initiated. When the sensor is coming back from under the surface threshold the cursor appears like an upward arrow. When the sensor is outside the boundaries of rframe the cursor appears as an iron cross.

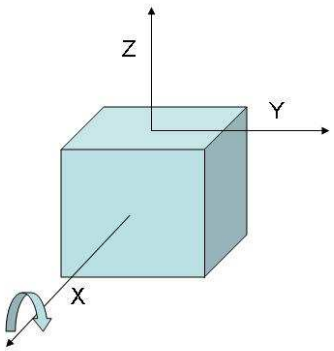


Figure A.1: 3D representation of the sensor, held in hand with desired orientation. Z-axis is for up and down movements. X-axis is for towards and away from the body movements and twist to right or left to produce clicks. Y-axis is for left and right movements

The following are some of the general guideline on how to operate the sensor:

1. The sensor needs to be held in the orientation as show in figure A.1. Whatever the rframe orientation is, the sensor should be held the same way as shown below. The sensor movements are also as described in figure A.1.

2. If the user only wants to traverse i.e., move the cursor, then the sensor has to be held upright as show in figure A.1. A tilt of more than 8°on the x-axis (roll) may result in stall of the cursor movement.

3. Left click will be initiated at the point where the sensor has hit the surface threshold. The user needs to remember that going below the threshold is not the completion of the left click. To complete the left click the user needs

to rise back above the threshold. So, going below the threshold is good for selection but not for a left click. To complete a left click on a button the user needs to go below the surface threshold on that button and then rise back above beyond the threshold on that button.

4. The rpen is a spatial device, i.e., it works in the air without any support for the user's arm. Thus, it will have hand jitter along with other kinds of noise described above. For this reason, a feature has been implemented in the rpen to detect an intention of the user for a left click event. To show the intention of doing a left click the user needs to make a pronounced downward (from the perspective of the rframe surface) movement and go below the threshold. Then, rpen will only consider the downward movement of the sensor and freeze the (x,y) movements, in terms of screen cursor movements. The cursor will stay in the place from where the downward motion was detected. If the user does not reach the surface threshold and makes an upward movement or levels out at a position above the surface threshold then this will not be counted as the intention for a left click. Also if the movement in the (length, width) direction, in terms of work surface, is more than 20mm then the click intention is not considered.

5. Right click is simulated as a right twist of the sensor on the x-axis that is, a roll of more than $+25^{\circ}$. Similarly to left click, the right click is completed when it returns from the $+25^{\circ}$twist. It should be mentioned here that the production version of rpen most likely will come with a button for generating a right click.

6. The user can do a left double click on a selected item by doing a roll on the left side, i.e., opposite to the right click action, of -35°. Unlike the left and right click, the user does not need to come back from the twist to complete the action. It is important to remember that a left double click is only possible when the sensor is below the surface threshold. It is also possible to do a double click by doing two quick left clicks at the same spot. But with the current sensor it is very difficult as the update rate is very slow. With a better hardware it should be possible.

7. For the ease of use for the user while doing a right click and left double click, cursor movements are ignored. Also where the right click is permissible, if the user rotates the sensor more than $+8.5^{\circ}$, it is treated as intention for doing a right click and the cursor remains in that place. Similarly, where the left double click is permissible if the user goes less than -8.0°, it is treated as the intention for doing a left double click and the cursor remains in that place.

8. For drag and drop, the rpen initiates a drag motion after the sensor has

moved in the length or width direction of the rframe 20 mm, after the initialization of the left click (going down to the surface threshold). To end the drag, basically to drop, the user needs to come above the surface threshold.

9. The user has the option to smooth out the cursor movements through the rframe Definition Application. The user can increase or decrease the "number of samples to average" option. The greater the number of samples to average, the smoother the cursor trajectory will be. But there will be a time lag between the actual sensor movement and the cursor movement. On the other hand, the lesser the number of samples to average, the jumpier the cursor becomes but with less time lag. Through experimentations it is found that the preferred value for this parameter is around 20 to 30.

A.2.2 Task Specific Instructions

Task 1 through to task 7 will be performed on a XY-plane with a predefined and a preset rframe and with a physical reference to that rframe.

Task 1: Cursor movement and positioning
This can be done by keeping the sensor above the surface threshold and without tilting it on the x-axis for more than $+8.5°$to $-8.0°$. The main idea of this task is to see the response of the cursor in response to sensor movements and positioning.

Task 2: Clicks - left click, right click and double click
To do a left click, the user needs to follow the general guideline items number 3 and 4. To do the right click, the user has to follow general guideline items number 5, keeping in mind number 7. Similarly, for the left double click the general guideline items number 6 and 7 should be followed.

Task 3: Selecting an object (small/medium/big) on the screen as well as a menu item selection
To select an object or a menu item the user needs to go below the surface threshold at that point as described in general guideline items number 3 and 4. But to click a menu item the user needs to come above the surface threshold at that spot.

There is an alternate way of clicking an object or menu item other than coming up to the threshold and that is to do a left double click. Due to hand jitter and other distortion problems, some users might find this option more suitable.

Task 4: Drag and Drop
To do Drag and Drop the user needs to follow the general guideline item number 8.

Task 5: Navigation through folders and documents
The user is required to open a folder and its sub-folders utilizing the tasks done

above and also open a document within that folder or sub-folder. Then the user needs to close the opened folders and document.

Due to metal distortions and other hardware limitations it may be difficult to operate on small controls such as the close (X) button on the title bar of a folder or document. Manipulating the menu might be a better option in that case, if available. For folders, initiating the context menu and clicking the close option may be easier then closing the (X)-button on the title bar.

Task 6: Navigation through an application and web browser
The user needs to open a web browser and try to left click on the links. It is similar to Task 5.

Task 7: Inserting, deletion or change of text in a document
The task here is to open a text document and do a cut, copy and paste. Also, the user needs to perform text selection for the copy or cut. Text selection can be done by first placing the cursor at the right place, then initiating a left click at the same place and then moving the sensor towards the place up to which the user wants to select. The user has to remember that a slight movement of the sensor upward or downward, in respect to screen, might result in vast area being selected.

For cut, copy and paste, the user must use a combination of left click, double click and right click and selection action.

Task 8: Defining the rframe
An rframe can be defined using the rframe Definition Application as shown in figure A.2.

The user first has to choose the type of rframe. In the current version of rpen the generic rframe, which is the rframe that can be in any orientation and can be on any plane, is partially implemented; thus, it is left out of this usability testing. The user is required to give a unique name to the rframe. The user needs to define the three points which will define the rframe. The user must use ctrl+l to set each point. The user can then make this new rframe the active rframe. As stated above, the user can set the threshold values.

While defining the rframe the actual sensor data is shown in the rframe definition screen, i.e., no rframe is active and no transformation for the sensor data to the screen cursor movement is done. When the rframe definition is done, the device driver returns to using the active rframe and sensor data transformation is done using the active rframe.

Task 9: Working on different orientations of rframe (XZ-touchscreen; YZ-Projection)
The user is to do simple tasks like cursor movements, positioning and clicks on rframe defined on the XZ and the YZ plane. This is basically to show how the rpen can operate on different rframes defined on different planes.

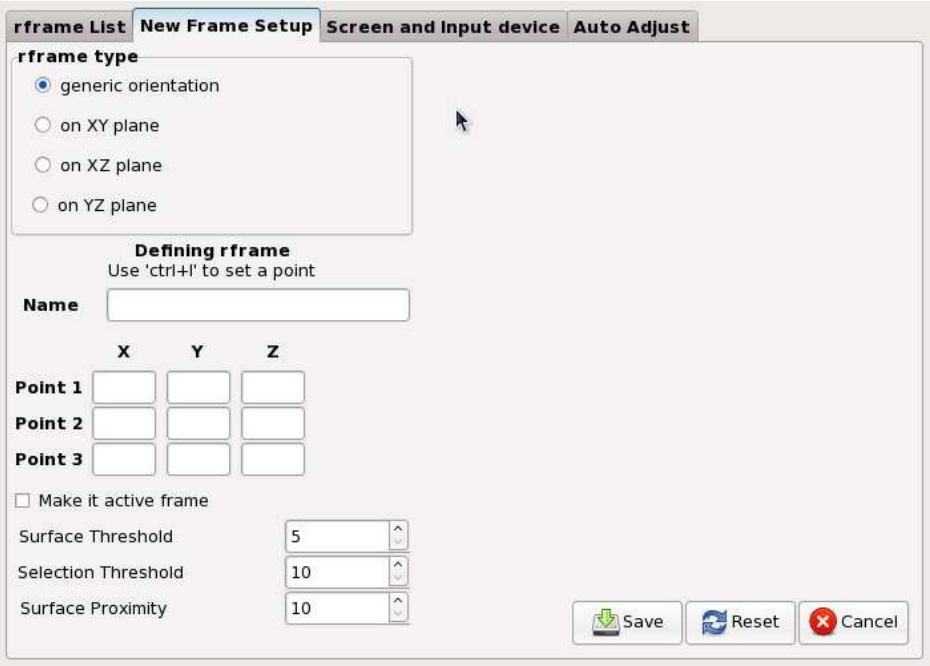


Figure A.2: Rframe definition screen.

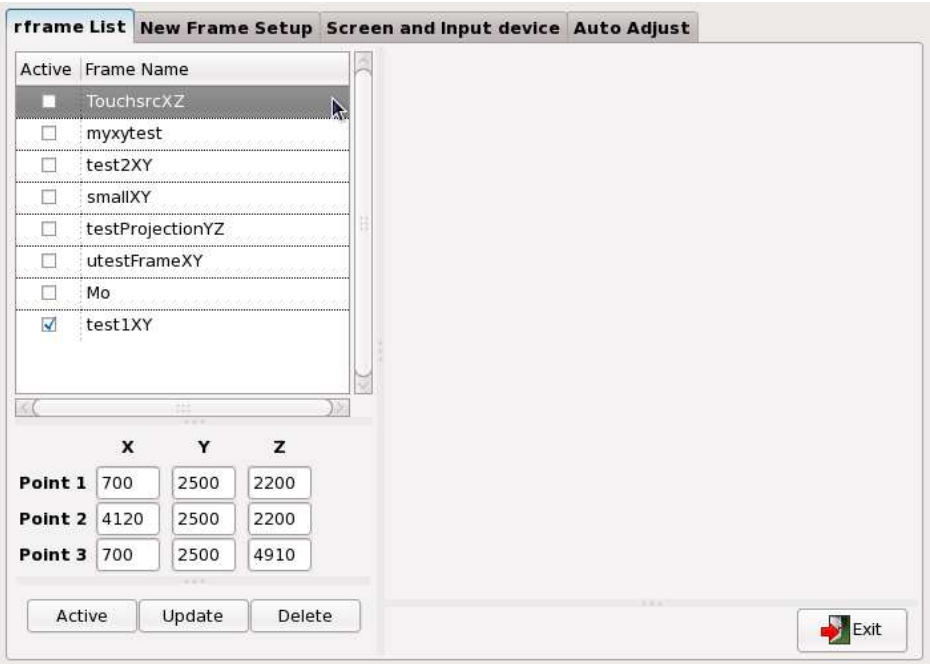


Figure A.3: Rframe activation and updating screen.

As shown in figure A.3, a user can activate a rframe by choosing it from the given list and clicking the 'Active' button. The user can also choose to update and delete a rframe from here. The user can not delete an active rframe.

Task 10: Working in auto adjust mode

Before this task the rpen will be using only the normal mode, which is, that the rframe will never change its defined location or points. In auto adjust mode, if the rpen or sensor is idle, i.e., if the cursor is not moving, for a specified set of seconds then after that time whenever the user picks it up and places the sensor at a place for more than five seconds that will become the initial point of the new rframe and the other points will be calculated from this point using the saved active rframe width and height. The rframe type will remain the same. The user should record the initial point some where near the origin, i.e., (0,0,0), near the base station. The user can switch to auto adjust mode using the rframe Definition Application as shown in figure A.4. After the active rframe is automatically adjusted, the user simply needs to checkout the new rframe by doing some simple tasks like - task 1 and 2. When this task is complete, return back to the normal mode by switching off the auto adjust mode.

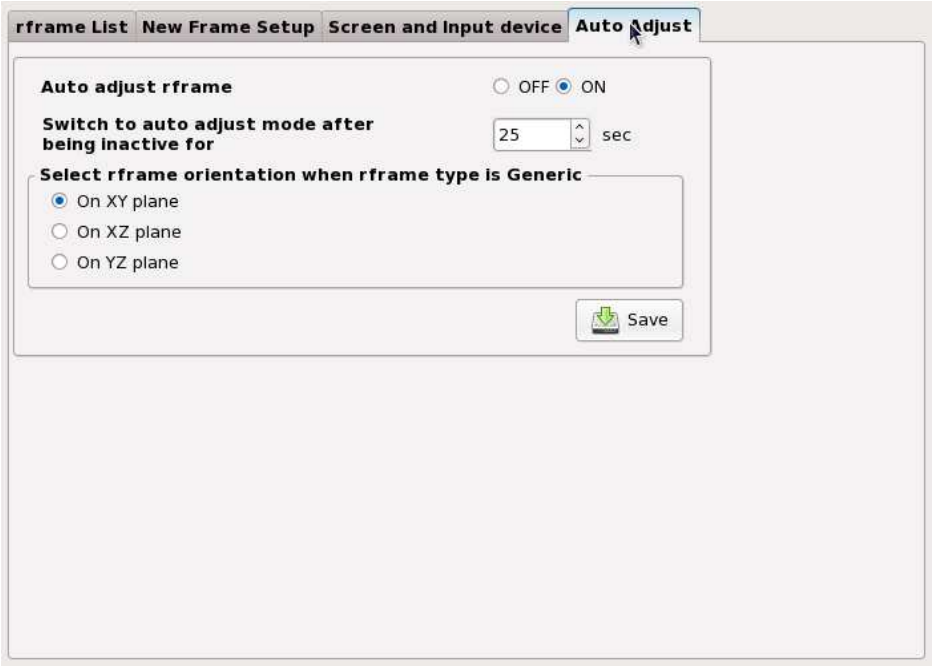


Figure A.4: Switching to auto adjust mode.

Task 11: Changing screen resolution while using the rpen

For this task, the screen resolution is required to be changed. For the usability test, the screen resolution will be changed to a virtual screen with a different

aspect ratio than the currently set one. When the screen resolution is successfully changed, the active rframe will adjust its size accordingly within two and half minutes. The changed rframe resolution can be seen using the rframe Definition Application. The rpen operation will also reflect the change. The user needs to do tasks similar to task 1 and 2 to check out this new rframe. After completing this task, return to the normal screen resolution.

Task 12: Working with the rpen in different combination of screen resolution and rframe size

Tasks 1 to 7 will be done on a XY-plane rframe which will be bigger than the display size. In task 9 the user will be using a XZ-plane rframe which will have the same size as the display screen. For this task the user will be given a rframe to work on XY-plane which is smaller than the display size. The basic idea here is to give the user a feel of how the rpen works, if the rframe size is bigger or smaller or equal to the display size.

APPENDIX B

Here we list all relevant source code developed for the rpen interface. The full rpen device driver source is given. Some of the significant or essential code segment of the user-space processes are given, as well. At the end, the Document Type Definition (DTD) shows the XML data format of rframe related information.

B.1 Rpen Device Driver Source Code

Listing B.1: Rpen Device Driver Source Code (Full)

```
/* This driver is for RPEN, a USB input deivce. */

#include <linux/usb.h>
/* required for USB device related functionality */
#include <linux/input-polldev.h>
/* required for input device and polling for input data */
#include <linux/init.h>
/* required for module 'init' and 'exit' */
#include <linux/module.h>
/* required for module functionality */
#include <linux/moduleparam.h>
/* required for module parameter functionality */
#include <linux/kernel.h>
/* required for functions like 'sprintf' or 'printk' */
#include <linux/slab.h>
/* required for memory allocation functionalities like 'kmalloc' */
#include <linux/errno.h>
/* required for reporting errors with specific error no and details */
#include <linux/sysfs.h>
/* required for sysfs related functionality */
#include <asm/uaccess.h>
/* need in order to work with sysfs read/writes .... */
#include <linux/string.h>
/* as used the strlen method */
#include <linux/mutex.h>
/* needed for mutex finctionality inside the probe function */
#include <linux/types.h>
/* needed for data types */

/* ###### Device Driver related Constants and global variables ############## */
#define DRIVER_VERSION        "0.1"
#define DRIVER_AUTHOR         "Mohammad Alam"
#define DRIVER_DESC           "Customized Wintracker"
#define DRIVER_LICENSE        "GPL"

#define DEVICE_VENDOR_ID      0x04b4
#define DEVICE_PRODUCT_ID     0x64df

#define MAX_SCREEN_X          1280
#define MAX_SCREEN_Y          1024

#define ARRAY_SIZE_AVG        125    /* size of the filterOutAgv array */
#define BOUNDARY_THRESHOLD    20     /* 2mm */
#define POSI_CLUTCH_THRESHOLD 600    /* 60mm - above the surface */

#define MIN_SENSOR_DIS_VAL    1500   /* 15 cm without extender */
#define MAX_SENSOR_DIS_VAL    7400   /* 74 cm without extender */
#define MIN_SENSOR_ANG_VAL    -18000 /* -180 degree */
#define MAX_SENSOR_ANG_VAL    18000  /* 180 degree */

#define DIS_JITTER_VALUE_POSI     3     /* 0.3mm */
#define DIS_JITTER_VALUE_NEGI     65532 /* 65535-3=-0.3mm */
#define DIS_PIXEL_JITTER_POSI     0     /* 0.0mm */
#define DIS_PIXEL_JITTER_NEGI     65535 /* 65535-0=0.0mm */
```

```
#define ANG_JITTER_VALUE_POSI      99     /* 0.99 degree */
#define ANG_JITTER_VALUE_NEGI      65436   /* 65535-99=-0.99 degree */

#define UNCLICK_THRESHOLD   200   /* 20 mm -- this distance and beyond (on the surface) will
     negate the intented click */

#define RIGHT_CLICK_ANG      2500     /* +25 deg */
#define LEFT_CLICK_ANG       -2000    /* -15 deg */
#define LEFT_DBLCLICK_ANG    -3500    /* -35 deg */

#define POSI_ROLL_THRESHOLD   850    /* +8.5 deg */
#define NEGI_ROLL_THRESHOLD   -800    /* -8 deg */

static signed long int normalVectorX, normalVectorY, normalVectorZ, kValue, normalVDot;
static int rightBtnClicked, leftBtnDblClicked, clickIntended, leftClickOnly;

/* possible values -1/0/1/2 -- if -1 then first time; if 0 then k-1=2 and k-2=1; if 1 then k
     -1=0 and k-2=2; if 2 then k-1=1 and k-2=0 */
static int k_index, last_k_index, seclast_k_index;
static int avg_track_index, filterOutNo;       /* average related variables */

typedef struct mov {
  int X;
  int Y;
  int Z;
  int A;
  int E;
  int R;
} typeMovement;

typeMovement click_point;

/* ########## Device attribute: global variable manipulated through sysfs */
static int useTransferFlag, updateFlag, rframeOriFlag;
static int minXpos, maxXpos;
static int minYpos, maxYpos;
static int minZpos, maxZpos;
static int surthreshold, selthreshold;
static int rframeWidth, rframeHeight;
static int srcMulFactor, rframeMulFactor;
static int srcFactor, inputFactor, avgSampleNo;
static int pDefPoint1X, pDefPoint1Y, pDefPoint1Z;
static int pDefPoint2X, pDefPoint2Y, pDefPoint2Z;
static int pDefPoint3X, pDefPoint3Y, pDefPoint3Z;
static int x_src_resolution, y_src_resolution, z_resolution;
static int X_sensorPos, Y_sensorPos, Z_sensorPos;
static int leftBtnClicked;

/* ####### Device attribute: functions and group for sysfs data communication ########## */

/* method to write to parameter files */
static ssize_t rpenParam_show(struct device *dev, struct device_attribute *attr,
                      char *buffer)
{
  if (strcmp(attr->attr.name, "flags") == 0)
    return snprintf(buffer, PAGE_SIZE, "%d %d %d\n", useTransferFlag, updateFlag,
        rframeOriFlag);
  if (strcmp(attr->attr.name, "xValRange") == 0)
    return snprintf(buffer, PAGE_SIZE, "%d %d\n", minXpos, maxXpos);
  if (strcmp(attr->attr.name, "yValRange") == 0)
    return snprintf(buffer, PAGE_SIZE, "%d %d\n", minYpos, maxYpos);
  if (strcmp(attr->attr.name, "zValRange") == 0)
    return snprintf(buffer, PAGE_SIZE, "%d %d\n", minZpos, maxZpos);
  if (strcmp(attr->attr.name, "threshold") == 0)
    return snprintf(buffer, PAGE_SIZE, "%d %d\n", surthreshold, selthreshold);
  if (strcmp(attr->attr.name, "rframeSize") == 0)
    return snprintf(buffer, PAGE_SIZE, "%d %d\n", rframeWidth, rframeHeight);
  if (strcmp(attr->attr.name, "multiFactor") == 0)
    return snprintf(buffer, PAGE_SIZE, "%d %d\n", srcMulFactor, rframeMulFactor);
  if (strcmp(attr->attr.name, "resFactor") == 0)
    return snprintf(buffer, PAGE_SIZE, "%d %d %d\n", srcFactor, inputFactor, avgSampleNo);
  if (strcmp(attr->attr.name, "pDefPoint1") == 0)
    return snprintf(buffer, PAGE_SIZE, "%d %d %d\n", pDefPoint1X, pDefPoint1Y, pDefPoint1Z);
  if (strcmp(attr->attr.name, "pDefPoint2") == 0)
    return snprintf(buffer, PAGE_SIZE, "%d %d %d\n", pDefPoint2X, pDefPoint2Y, pDefPoint2Z);
  if (strcmp(attr->attr.name, "pDefPoint3") == 0)
    return snprintf(buffer, PAGE_SIZE, "%d %d %d\n", pDefPoint3X, pDefPoint3Y, pDefPoint3Z);
  if (strcmp(attr->attr.name, "screenRes") == 0)
    return snprintf(buffer, PAGE_SIZE, "%d %d %d\n", x_src_resolution, y_src_resolution,
        z_resolution);
  if (strcmp(attr->attr.name, "sensorPosition") == 0)
    return snprintf(buffer, PAGE_SIZE, "%d %d %d\n", X_sensorPos, Y_sensorPos, Z_sensorPos);
  if (strcmp(attr->attr.name, "clickEvent") == 0)
    return snprintf(buffer, PAGE_SIZE, "%d\n", leftBtnClicked);
}

/* method to read from parameter files wherever updated by the user */
```

```
static ssize_t rpenParam_store(struct device *dev, struct device_attribute *attr,
                        const char *buffer, size_t size)
{
141   if (strcmp(attr->attr.name, "flags") == 0) {
        sscanf(buffer, "%d%d%d", &useTransferFlag, &updateFlag, &rframeOriFlag);
        printk(KERN_ALERT "rpen Module: flag_params TransferFlag: %d :: updateFlag: %d ::
             rframeOriFlag: %d.\n", useTransferFlag, updateFlag, rframeOriFlag);
      }
      if (strcmp(attr->attr.name, "xValRange") == 0) {
        sscanf(buffer, "%d%d", &minXpos, &maxXpos);
        printk(KERN_ALERT "rpen Module: xValRange_params minXpos: %d :: maxXpos: %d.\n", minXpos,
             maxXpos);
      }
      if (strcmp(attr->attr.name, "yValRange") == 0) {
        sscanf(buffer, "%d%d", &minYpos, &maxYpos);
151     printk(KERN_ALERT "rpen Module: yValRange_params minYpos: %d :: maxYpos: %d.\n", minYpos,
             maxYpos);
      }
      if (strcmp(attr->attr.name, "zValRange") == 0) {
        sscanf(buffer, "%d%d", &minZpos, &maxZpos);
        printk(KERN_ALERT "rpen Module: zValRange_params minZpos: %d :: maxZpos: %d.\n", minZpos,
             maxZpos);
      }
      if (strcmp(attr->attr.name, "threshold") == 0) {
        sscanf(buffer, "%d%d", &surthreshold, &selthreshold);
        printk(KERN_ALERT "rpen Module: threshold_params surthreshold: %d :: selthreshold: %d.\n"
             , surthreshold, selthreshold);
      }
161   if (strcmp(attr->attr.name, "rframeSize") == 0) {
        sscanf(buffer, "%d%d", &rframeWidth, &rframeHeight);
        printk(KERN_ALERT "rpen Module: threshold_params rframeWidth: %d :: rframeHeight: %d.\n",
             rframeWidth, rframeHeight);
      }
      if (strcmp(attr->attr.name, "multiFactor") == 0) {
        sscanf(buffer, "%d%d", &srcMulFactor, &rframeMulFactor);
        printk(KERN_ALERT "rpen Module: multiFactor_params srcMulFactor: %d :: rframeMulFactor: %
             d.\n", srcMulFactor, rframeMulFactor);
      }
      if (strcmp(attr->attr.name, "resFactor") == 0) {
        sscanf(buffer, "%d%d%d", &srcFactor, &inputFactor, &avgSampleNo);
171     printk(KERN_ALERT "rpen Module: resFactor_params srcFactor: %d :: inputFactor: %d ::
             avgSampleNo: %d.\n", srcFactor, inputFactor, avgSampleNo);
      }
      if (strcmp(attr->attr.name, "pDefPoint1") == 0) {
        sscanf(buffer, "%d%d%d", &pDefPoint1X, &pDefPoint1Y, &pDefPoint1Z);
        printk(KERN_ALERT "rpen Module: pDefPoint1_params pDefPoint1X: %d :: pDefPoint1Y: %d ::
             pDefPoint1Z: %d.\n", pDefPoint1X, pDefPoint1Y, pDefPoint1Z);
      }
      if (strcmp(attr->attr.name, "pDefPoint2") == 0) {
        sscanf(buffer, "%d%d%d", &pDefPoint2X, &pDefPoint2Y, &pDefPoint2Z);
        printk(KERN_ALERT "rpen Module: pDefPoint2_params pDefPoint2X: %d :: pDefPoint2Y: %d ::
             pDefPoint2Z: %d.\n", pDefPoint2X, pDefPoint2Y, pDefPoint2Z);
      }
181   if (strcmp(attr->attr.name, "pDefPoint3") == 0) {
        sscanf(buffer, "%d%d%d", &pDefPoint3X, &pDefPoint3Y, &pDefPoint3Z);
        printk(KERN_ALERT "rpen Module: pDefPoint3_params pDefPoint3X: %d :: pDefPoint3Y: %d ::
             pDefPoint3Z: %d.\n", pDefPoint3X, pDefPoint3Y, pDefPoint3Z);
      }
      if (strcmp(attr->attr.name, "screenRes") == 0) {
        sscanf(buffer, "%d%d%d", &x_src_resolution, &y_src_resolution, &z_resolution);
        printk(KERN_ALERT "rpen Module: screenRes_params x_src_resolution: %d :: y_src_resolution
             : %d :: z_resolution: %d.\n", x_src_resolution, y_src_resolution, z_resolution);
      }
      if (strcmp(attr->attr.name, "sensorPosition") == 0) {
        sscanf(buffer, "%d%d%d", &X_sensorPos, &Y_sensorPos, &Z_sensorPos);
191     printk(KERN_ALERT "rpen Module: sensorPosition_params X_sensorPos: %d :: Y_sensorPos: %d
             :: Z_sensorPos: %d.\n", X_sensorPos, Y_sensorPos, Z_sensorPos);
      }
      if (strcmp(attr->attr.name, "clickEvent") == 0) {
        sscanf(buffer, "%d", &leftBtnClicked);
        printk(KERN_ALERT "rpen Module: clickEvent_params leftBtnClicked: %d.\n", leftBtnClicked)
             ;
      }

      return strnlen(buffer, size);
}

201 /* set of attributes required to be communicated: each will have its own node is sysfs under
       the device folder */
static DEVICE_ATTR(flags, S_IWUSR | S_IRUGO, rpenParam_show, rpenParam_store);
static DEVICE_ATTR(xValRange, S_IWUSR | S_IRUGO, rpenParam_show, rpenParam_store);
static DEVICE_ATTR(yValRange, S_IWUSR | S_IRUGO, rpenParam_show, rpenParam_store);
static DEVICE_ATTR(zValRange, S_IWUSR | S_IRUGO, rpenParam_show, rpenParam_store);
static DEVICE_ATTR(threshold, S_IWUSR | S_IRUGO, rpenParam_show, rpenParam_store);
static DEVICE_ATTR(rframeSize, S_IWUSR | S_IRUGO, rpenParam_show, rpenParam_store);
static DEVICE_ATTR(multiFactor, S_IWUSR | S_IRUGO, rpenParam_show, rpenParam_store);
static DEVICE_ATTR(resFactor, S_IWUSR | S_IRUGO, rpenParam_show, rpenParam_store);
```

```
    static DEVICE_ATTR(pDefPoint1, S_IWUSR | S_IRUGO, rpenParam_show, rpenParam_store);
211 static DEVICE_ATTR(pDefPoint2, S_IWUSR | S_IRUGO, rpenParam_show, rpenParam_store);
    static DEVICE_ATTR(pDefPoint3, S_IWUSR | S_IRUGO, rpenParam_show, rpenParam_store);
    static DEVICE_ATTR(screenRes, S_IWUSR | S_IRUGO, rpenParam_show, rpenParam_store);
    static DEVICE_ATTR(sensorPosition, S_IWUSR | S_IRUGO, rpenParam_show, rpenParam_store);
    static DEVICE_ATTR(clickEvent, S_IWUSR | S_IRUGO, rpenParam_show, rpenParam_store);

    /* Create a group of attributes so that we can create and destory them all at once. */
    static struct device_attribute *rpen_attrs[] = {
      &dev_attr_flags.attr,
      &dev_attr_xValRange.attr,
221   &dev_attr_yValRange.attr,
      &dev_attr_zValRange.attr,
      &dev_attr_threshold.attr,
      &dev_attr_rframeSize.attr,
      &dev_attr_multiFactor.attr,
      &dev_attr_resFactor.attr,
      &dev_attr_pDefPoint1.attr,
      &dev_attr_pDefPoint2.attr,
      &dev_attr_pDefPoint3.attr,
      &dev_attr_screenRes.attr,
231   &dev_attr_sensorPosition.attr,
      &dev_attr_clickEvent.attr,
            NULL   /* need to NULL terminate the list of attributes */
    };

    static struct attribute_group attr_group = {
      .name = "rpenParam",
      .attrs = rpen_attrs,
    };

241 /* ######## Device related structure and initialization ########## */
    typedef struct usb_myMouse {         /* using typedef so that do not has to use 'struct' every
          time an instance is needed */
      char name[128];         /* name of the device */
      char phys[64];          /* physical path of the device */
      signed char *data;       /* URB data */
      signed char *send_buffer;
      int dataLength;         /* URB data length */

      struct usb_device *usbdev;     /* USB device */
      struct usb_interface *interface;  /* usb interface */
251   struct input_polled_dev *inputPollDev;  /* the input device will be a part of this */
      int pollInterval;
      unsigned int in_pipe;      /* input data pipe */
      unsigned int out_pipe;       /* output data pipe */
      struct mutex bulk_io_mutex;   /* used only for bulk out situation in probe */

      typeMovement filterOutput[3];   /* last three filter output will be stored here */
      typeMovement sensorIntput[3];   /* last three sensor input will be stored here */

      typeMovement filterOutAgv[ARRAY_SIZE_AVG];
261   typeMovement lastAvgOutput;
      typeMovement currentAvgOutput;
    } myMouseType;

    /* Initialization of myMouseType data structure */
    void init_myMouseDevice(myMouseType *myMouse)
    {
      int i;

      memset(myMouse, 0, sizeof(myMouseType));
271   myMouse->inputPollDev = NULL;
      myMouse->interface = NULL;
      myMouse->data = NULL;
      myMouse->send_buffer = NULL;

      for (i=0; i<3; i++) {
        myMouse->filterOutput[i].X = 0;
        myMouse->filterOutput[i].Y = 0;
        myMouse->filterOutput[i].Z = 0;
        myMouse->filterOutput[i].A = 0;
281     myMouse->filterOutput[i].E = 0;
        myMouse->filterOutput[i].R = 0;

        myMouse->sensorIntput[i].X = 0;
        myMouse->sensorIntput[i].Y = 0;
        myMouse->sensorIntput[i].Z = 0;
        myMouse->sensorIntput[i].A = 0;
        myMouse->sensorIntput[i].E = 0;
        myMouse->sensorIntput[i].R = 0;
      }
291
      for (i=0; i<ARRAY_SIZE_AVG; i++) {
        myMouse->filterOutAgv[i].X = 0;
        myMouse->filterOutAgv[i].Y = 0;
        myMouse->filterOutAgv[i].Z = 0;
```

```
        myMouse->filterOutAgv[i].A = 0;
        myMouse->filterOutAgv[i].E = 0;
        myMouse->filterOutAgv[i].R = 0;
      }
      myMouse->lastAvgOutput.X = 0;
301   myMouse->lastAvgOutput.Y = 0;
      myMouse->lastAvgOutput.Z = 0;
      myMouse->lastAvgOutput.A = 0;
      myMouse->lastAvgOutput.E = 0;
      myMouse->lastAvgOutput.R = 0;

      myMouse->currentAvgOutput.X = 0;
      myMouse->currentAvgOutput.Y = 0;
      myMouse->currentAvgOutput.Z = 0;
      myMouse->currentAvgOutput.A = 0;
311   myMouse->currentAvgOutput.E = 0;
      myMouse->currentAvgOutput.R = 0;
    }

    /* freeing up the myMouseType structure */
    void free_myMouseData(myMouseType *myMouse)
    {
      if (myMouse->data)
        kfree(myMouse->data);
      if (myMouse->send_buffer)
321     kfree(myMouse->send_buffer);
      if (myMouse->interface)
        myMouse->interface = NULL;
      if (myMouse->inputPollDev)
        input_free_polled_device(myMouse->inputPollDev);
      kfree(myMouse);
    }

    /* function to create a new usb_myMouse device */
    myMouseType *create_myMouse(struct usb_device *udev, struct usb_interface *intf)
331 {
      myMouseType *myMouse;

      /* allocate memory for myMouseType device and initialize */
      if (!(myMouse = kmalloc(sizeof(myMouseType), GFP_KERNEL)))
        return NULL;
      init_myMouseDevice(myMouse);

      /* allocating input device */
      if ((myMouse->inputPollDev = input_allocate_polled_device()) == NULL) {
341     printk(KERN_INFO "myMouse: input_allocate_polled_device failed.\n");
        free_myMouseData(myMouse);
        return NULL;
      }

      /* allocating data required for data input */
      myMouse->dataLength = 32;
      if (!(myMouse->data = kmalloc(myMouse->dataLength, GFP_KERNEL))) {
        free_myMouseData(myMouse);
        printk(KERN_INFO "myMouse: create_myMouse: failed to allocate buffer.\n");
351     return NULL;
      }
      if (!(myMouse->send_buffer = kmalloc(myMouse->dataLength, GFP_KERNEL))) {
        free_myMouseData(myMouse);
        printk(KERN_INFO "myMouse: create_myMouse: failed to allocate buffer.\n");
        return NULL;
      }
      myMouse->send_buffer[0] = 'S';
      myMouse->send_buffer[1] = 'P';
      myMouse->send_buffer[2] = '\0';
361
      /* setting polling interval in msec */
      myMouse->pollInterval = 2;          /* 2 msec */

      /* initializing mutex lock  */
      mutex_init(&myMouse->bulk_io_mutex);

      /* setting myMouse usb device and interface */
      myMouse->usbdev = udev;
      myMouse->interface = intf;
371
      return myMouse;
    }

    /* ######## Defining probe, disconnect, irq_handling and other related callback functions
        ########## */
    static int transformPosition(int posValue)
    {
      int newPosVal;

      /* as floating point not considered output value may not be accurate */
381   if (rframeMulFactor != 0) {
```

```
      newPosVal = (posValue * srcMulFactor)/rframeMulFactor;       /* transforming to screen
          resolution */
    }
    return newPosVal;
  }

  /* this function always provides the ceil of a division */
  int sqroot(int m)
  {
    int i=0;
391 int x1, x2, j;

    while((i*i) <= m*10) {
      i+=1;
    }
    x1=i;

    for(j=0;j<10;j++) {
      x2=m;
      x2/=x1;
401   x2+=x1;
      x2/=2;
      x1=x2;
    }
    return x2;
  }

  static int Magnitude(int X1, int X2, int Y1, int Y2, int Z1, int Z2)
  {
    int VectorX, VectorY, VectorZ;
411
    VectorX = X2 - X1;
    VectorY = Y2 - Y1;
    VectorZ = Z2 - Z1;

    return (int)sqroot(VectorX * VectorX + VectorY * VectorY + VectorZ * VectorZ);
  }

  static void DistancePointLine(int pX, int pY, int pZ, int *xDistance, int *yDistance)
  {
421 int U;
    int IntersecX, IntersecY, IntersecZ;

    U = (((pX - pDefPoint1X) * (pDefPoint2X - pDefPoint1X)) +
    ((pY - pDefPoint1Y) * (pDefPoint2Y - pDefPoint1Y)) +
    (( pZ - pDefPoint1Z) * (pDefPoint2Z - pDefPoint1Z)));

    IntersecX = pDefPoint1X + U*(pDefPoint2X - pDefPoint1X)/(rframeWidth * rframeWidth);
    IntersecY = pDefPoint1Y + U*(pDefPoint2Y - pDefPoint1Y)/(rframeWidth * rframeWidth);
    IntersecZ = pDefPoint1Z + U*(pDefPoint2Z - pDefPoint1Z)/(rframeWidth * rframeWidth);
431
    *yDistance = Magnitude(pX, IntersecX,  pY, IntersecY, pZ, IntersecZ);

    U = (((pX - pDefPoint2X) * (pDefPoint3X - pDefPoint2X)) +
    ((pY - pDefPoint2Y) * (pDefPoint3Y - pDefPoint2Y)) +
    (( pZ - pDefPoint2Z) * (pDefPoint3Z - pDefPoint2Z)));

    IntersecX = pDefPoint2X + U*(pDefPoint3X - pDefPoint2X)/(rframeHeight * rframeHeight);
    IntersecY = pDefPoint2Y + U*(pDefPoint3Y - pDefPoint2Y)/(rframeHeight * rframeHeight);
    IntersecZ = pDefPoint2Z + U*(pDefPoint3Z - pDefPoint2Z)/(rframeHeight * rframeHeight);
441
    *xDistance = rframeHeight - Magnitude(pX, IntersecX,  pY, IntersecY, pZ, IntersecZ);
  }

  static int transferFunction(myMouseType *myMouse, struct input_dev *inputdev, int xPosition,
      int yPosition, int zPosition, int rPos, int ePos, int aPos)
  {
    int tmpP3PX, tmpP3PY, tmpP3PZ;
    int dotValue, dotVnpos;
    int xDistance, yDistance, distanceZ, normalVlen;
    int newXPos, newYPos, newZPos;
451 int transformX, transformY;
    int tmpPXP1, tmpPYP1, tmpPZP1;
    __u16 diffX, diffY;
    int Dvalx, Dvaly, Dvalz, Dvalr;

    if (useTransferFlag == 0) {   /* report the sensor position as it is */
      X_sensorPos = xPosition;
      Y_sensorPos = yPosition;
      Z_sensorPos = zPosition;
      input_report_abs(inputdev, ABS_X, xPosition);
461   input_report_abs(inputdev, ABS_Y, yPosition);
      input_report_abs(inputdev, ABS_Z, zPosition);
      input_report_abs(inputdev, ABS_RX, rPos);
      input_report_abs(inputdev, ABS_RY, ePos);
      input_report_abs(inputdev, ABS_RZ, aPos);
      input_sync(inputdev);
```

```
    } else {
      if (rPos >= RIGHT_CLICK_ANG) {
        if (rightBtnClicked == 0) {
          input_report_key(inputdev, BTN_RIGHT, 1);    /* reporting the click */
          input_sync(inputdev);
          input_report_key(inputdev, BTN_RIGHT, 0);    /* unclicking the right button */
          input_sync(inputdev);
          rightBtnClicked = 1;
        }
      } else {
        if (rightBtnClicked == 1) { /* if right button was clicked then update the flag as
            unclicked */
          rightBtnClicked = 0;
        }
      }

      /* dot product of normal of the plane (rframe) and the point */
      /* dotValue = (normalVectorX*xPosition) + (normalVectorY*yPosition) + (normalVectorZ*
          zPosition); */
      dotValue = (normalVectorX*xPosition) + (normalVectorY*yPosition) + (normalVectorZ*
          zPosition);

      switch (rframeOriFlag) {
      case 0:        /* generic orientation */
        /* to get the position on the surface */
        tmpP3PX = pDefPoint3X - xPosition;
        tmpP3PY = pDefPoint3Y - yPosition;
        tmpP3PZ = pDefPoint3Z - zPosition;
        dotVnpos = (normalVectorX*tmpP3PX) + (normalVectorY*tmpP3PY) + (normalVectorZ*tmpP3PZ);
        newXPos = xPosition + (dotVnpos*normalVectorX/normalVDot);
        newYPos = yPosition + (dotVnpos*normalVectorY/normalVDot);
        newZPos = zPosition + (dotVnpos*normalVectorZ/normalVDot);

        normalVlen = sqroot(normalVDot);
        tmpPXP1 =  xPosition - pDefPoint1X;
        tmpPYP1 =  yPosition - pDefPoint1Y;
        tmpPZP1 =  zPosition - pDefPoint1Z;
        distanceZ = (normalVectorX/normalVlen)*tmpPXP1 + (normalVectorY/normalVlen)*tmpPYP1 + (
            normalVectorZ/normalVlen)*tmpPZP1;

        if (dotValue == kValue) {   /* then the point is on the plane */
          if (leftBtnClicked == 1) {  /* if left button was clicked then unclick it now */
            input_report_key(inputdev, BTN_LEFT, 0);
            leftBtnClicked = 0;
          }

          /* getting the distance value for x and y coordinates */
          DistancePointLine(xPosition, yPosition, zPosition, &xDistance, &yDistance);

          transformX = transformPosition(xDistance);
          transformY = transformPosition(yDistance);

          /* checking for pixel jitter */
          if (!((X_sensorPos == 0) && (Y_sensorPos == 0))) {    /* not (the first time) */
            diffX = transformX - X_sensorPos;
            diffY = transformY - Y_sensorPos;

            if (((diffX <= DIS_PIXEL_JITTER_POSI) || (diffX > DIS_PIXEL_JITTER_NEGI)) && ((
                diffY <= DIS_PIXEL_JITTER_POSI) || (diffY > DIS_PIXEL_JITTER_NEGI))) { /*
                jitter detected */

              printk(KERN_INFO "myMouse_driver: transferFunction: transformed movement of
                  sensor is treated as jitter.\n");
              /* reporting the last movement */
              input_report_abs(inputdev, ABS_X, X_sensorPos);
              input_report_abs(inputdev, ABS_Y, Y_sensorPos);
              input_report_abs(inputdev, ABS_Z, Z_sensorPos);
            } else {
              X_sensorPos = transformX;
              Y_sensorPos = transformY;
              Z_sensorPos = 0;
              input_report_abs(inputdev, ABS_X, transformX);
              input_report_abs(inputdev, ABS_Y, transformY);
              input_report_abs(inputdev, ABS_Z, 0);
            }
          } else {
            X_sensorPos = transformX;
            Y_sensorPos = transformY;
            Z_sensorPos = 0;
            input_report_abs(inputdev, ABS_X, transformX);
            input_report_abs(inputdev, ABS_Y, transformY);
            input_report_abs(inputdev, ABS_Z, 0);
          }

          input_report_abs(inputdev, ABS_RX, rPos);
          input_report_abs(inputdev, ABS_RY, ePos);
          input_report_abs(inputdev, ABS_RZ, aPos);
```

```
            input_sync(inputdev);

          } else if (dotValue < kValue) {

551         if (distanceZ <= selthreshold){
              if (distanceZ <= surthreshold) {     /* if dotValue is with surthreshold then report
                   transformed position */
                if (leftBtnClicked == 1) {     /* if left button was clicked then unclick it now
                     */
                  input_report_key(inputdev, BTN_LEFT, 0);
                  leftBtnClicked = 0;
                }

                /* getting the distance value for x and y coordinates */
                DistancePointLine(newXPos, newYPos, newZPos, &xDistance, &yDistance);

561             transformX = transformPosition(xDistance);
                transformY = transformPosition(yDistance);

                /* checking for pixel jitter -- check if the value is always positive */
                if (!((X_sensorPos == 0) && (Y_sensorPos == 0))) {     /* not (the first time) */
                  diffX = transformX - X_sensorPos;
                  diffY = transformY - Y_sensorPos;

                  if (((diffX <= DIS_PIXEL_JITTER_POSI) || (diffX > DIS_PIXEL_JITTER_NEGI)) && ((
                       diffY <= DIS_PIXEL_JITTER_POSI) || (diffY > DIS_PIXEL_JITTER_NEGI))) {    /*
                        jitter detected */

571                 printk(KERN_INFO "myMouse_driver: transferFunction: transformed movement of
                         sensor is treated as jitter.\n");
                    /* reporting the last movement */
                    input_report_abs(inputdev, ABS_X, X_sensorPos);
                    input_report_abs(inputdev, ABS_Y, Y_sensorPos);
                    input_report_abs(inputdev, ABS_Z, Z_sensorPos);
                  } else {
                    X_sensorPos = transformX;
                    Y_sensorPos = transformY;
                    Z_sensorPos = distanceZ;
                    input_report_abs(inputdev, ABS_X, transformX);
581                 input_report_abs(inputdev, ABS_Y, transformY);
                    input_report_abs(inputdev, ABS_Z, distanceZ);
                  }
                } else {
                  X_sensorPos = transformX;
                  Y_sensorPos = transformY;
                  Z_sensorPos = distanceZ;
                  input_report_abs(inputdev, ABS_X, transformX);
                  input_report_abs(inputdev, ABS_Y, transformY);
                  input_report_abs(inputdev, ABS_Z, distanceZ);
591             }

                input_report_abs(inputdev, ABS_RX, rPos);
                input_report_abs(inputdev, ABS_RY, ePos);
                input_report_abs(inputdev, ABS_RZ, aPos);
                input_sync(inputdev);

              } else if (distanceZ > surthreshold) {  /* its a simulated left button click */
                if (rPos <= LEFT_CLICK_ANG) {
                  if (leftBtnClicked != 1) {
601                 input_report_key(inputdev, BTN_LEFT, 1);
                    leftBtnClicked = 1;
                  }
                } else {
                  if (leftBtnClicked == 1) {  /* if left button was clicked then unclick it now
                       */
                    input_report_key(inputdev, BTN_LEFT, 0);
                    leftBtnClicked = 0;
                  }
                }

611             /* getting the distance value for x and y coordinates */
                DistancePointLine(newXPos, newYPos, newZPos, &xDistance, &yDistance);

                transformX = transformPosition(xDistance);
                transformY = transformPosition(yDistance);

                /* checking for pixel jitter -- check if the value is always positive */
                if (!((X_sensorPos == 0) && (Y_sensorPos == 0))) {     /* not (the first time) */
                  diffX = transformX - X_sensorPos;
                  diffY = transformY - Y_sensorPos;
621
                  if (((diffX <= DIS_PIXEL_JITTER_POSI) || (diffX > DIS_PIXEL_JITTER_NEGI)) && ((
                       diffY <= DIS_PIXEL_JITTER_POSI) || (diffY > DIS_PIXEL_JITTER_NEGI))) {    /*
                        jitter detected */

                    printk(KERN_INFO "myMouse_driver: transferFunction: transformed movement of
                         sensor is treated as jitter.\n");
```

```
                /* reporting the last movement */
                input_report_abs(inputdev, ABS_X, X_sensorPos);
                input_report_abs(inputdev, ABS_Y, Y_sensorPos);
                input_report_abs(inputdev, ABS_Z, Z_sensorPos);
              } else {
                X_sensorPos = transformX;
631             Y_sensorPos = transformY;
                Z_sensorPos = distanceZ;
                input_report_abs(inputdev, ABS_X, transformX);
                input_report_abs(inputdev, ABS_Y, transformY);
                input_report_abs(inputdev, ABS_Z, distanceZ);
              }
            } else {
              X_sensorPos = transformX;
              Y_sensorPos = transformY;
              Z_sensorPos = distanceZ;
641           input_report_abs(inputdev, ABS_X, transformX);
              input_report_abs(inputdev, ABS_Y, transformY);
              input_report_abs(inputdev, ABS_Z, distanceZ);
            }

            input_report_abs(inputdev, ABS_RX, rPos);
            input_report_abs(inputdev, ABS_RY, ePos);
            input_report_abs(inputdev, ABS_RZ, aPos);
            input_sync(inputdev);
          }
651       } else {      /* this is one of the clutches (-) -- distanceZ > selthreshold -- same
              as left click */
            if (rPos <= LEFT_CLICK_ANG) {
              if (leftBtnClicked != 1) {
                input_report_key(inputdev, BTN_LEFT, 1);
                leftBtnClicked = 1;
              }
            } else {
              if (leftBtnClicked == 1) {  /* if left button was clicked then unclick it now */
                input_report_key(inputdev, BTN_LEFT, 0);
                leftBtnClicked = 0;
661           }
            }

            /* getting the distance value for x and y coordinates */
            DistancePointLine(newXPos, newYPos, newZPos, &xDistance, &yDistance);

            transformX = transformPosition(xDistance);
            transformY = transformPosition(yDistance);

            /* checking for pixel jitter -- check if the value is always positive */
671         if (!((X_sensorPos == 0) && (Y_sensorPos == 0))) {    /* not (the first time) */
              diffX = transformX - X_sensorPos;
              diffY = transformY - Y_sensorPos;

              if (((diffX <= DIS_PIXEL_JITTER_POSI) || (diffX > DIS_PIXEL_JITTER_NEGI)) && ((
                  diffY <= DIS_PIXEL_JITTER_POSI) || (diffY > DIS_PIXEL_JITTER_NEGI))) {   /*
                  jitter detected */

                printk(KERN_INFO "myMouse_driver: transferFunction: transformed movement of
                    sensor is treated as jitter.\n");
                /* reporting the last movement */
                input_report_abs(inputdev, ABS_X, X_sensorPos);
                input_report_abs(inputdev, ABS_Y, Y_sensorPos);
681             input_report_abs(inputdev, ABS_Z, Z_sensorPos);
              } else {
                X_sensorPos = transformX;
                Y_sensorPos = transformY;
                Z_sensorPos = distanceZ;
                input_report_abs(inputdev, ABS_X, transformX);
                input_report_abs(inputdev, ABS_Y, transformY);
                input_report_abs(inputdev, ABS_Z, distanceZ);
              }
            } else {
691           X_sensorPos = transformX;
              Y_sensorPos = transformY;
              Z_sensorPos = distanceZ;
              input_report_abs(inputdev, ABS_X, transformX);
              input_report_abs(inputdev, ABS_Y, transformY);
              input_report_abs(inputdev, ABS_Z, distanceZ);
            }

            input_report_abs(inputdev, ABS_RX, rPos);
            input_report_abs(inputdev, ABS_RY, ePos);
701         input_report_abs(inputdev, ABS_RZ, aPos);
            input_sync(inputdev);
          }
        } else  {       /* then dotValue is greater, i.e., the point is above the surface */
          if (distanceZ <= POSI_CLUTCH_THRESHOLD) {
            if (leftBtnClicked == 1) {  /* if left button was clicked then unclick it now */
              input_report_key(inputdev, BTN_LEFT, 0);
```

```
            leftBtnClicked = 0;
          }

          /* getting the distance value for x and y coordinates */
          DistancePointLine(newXPos, newYPos, newZPos, &xDistance, &yDistance);

          transformX = transformPosition(xDistance);
          transformY = transformPosition(yDistance);

          /* checking for pixel jitter -- check if the value is always positive */
          if (!((X_sensorPos == 0) && (Y_sensorPos == 0))) {     /* not (the first time) */
            diffX = transformX - X_sensorPos;
            diffY = transformY - Y_sensorPos;

            if (((diffX <= DIS_PIXEL_JITTER_POSI) || (diffX > DIS_PIXEL_JITTER_NEGI)) && ((
                diffY <= DIS_PIXEL_JITTER_POSI) || (diffY > DIS_PIXEL_JITTER_NEGI))) {   /*
                jitter detected */

              printk(KERN_INFO "myMouse_driver: transferFunction: transformed movement of
                  sensor is treated as jitter.\n");
              /* reporting the last movement */
              input_report_abs(inputdev, ABS_X, X_sensorPos);
              input_report_abs(inputdev, ABS_Y, Y_sensorPos);
              input_report_abs(inputdev, ABS_Z, Z_sensorPos);
            } else {
              X_sensorPos = transformX;
              Y_sensorPos = transformY;
              Z_sensorPos = distanceZ;
              input_report_abs(inputdev, ABS_X, transformX);
              input_report_abs(inputdev, ABS_Y, transformY);
              input_report_abs(inputdev, ABS_Z, distanceZ);
            }
          } else {
            X_sensorPos = transformX;
            Y_sensorPos = transformY;
            Z_sensorPos = distanceZ;
            input_report_abs(inputdev, ABS_X, transformX);
            input_report_abs(inputdev, ABS_Y, transformY);
            input_report_abs(inputdev, ABS_Z, distanceZ);
          }

          input_report_abs(inputdev, ABS_RX, rPos);
          input_report_abs(inputdev, ABS_RY, ePos);
          input_report_abs(inputdev, ABS_RZ, aPos);
          input_sync(inputdev);
        } else {          /* this is one of the clutches (+) */
          /* reporting nothing to the input event subsystem */
          Z_sensorPos = distanceZ;
          return 0;
        }
      }

      break;
    case 1:       /* rframe on xy plane [desktop mode, may be] (here assuming the base and
        the screen have same x-axis, in regards to coordinates) */
      distanceZ = (minZpos - zPosition);

      if (distanceZ == 0) {   /* then the point is on the plane */
        if (leftBtnClicked == 1) {  /* if left button was clicked then unclick it now */
          input_report_key(inputdev, BTN_LEFT, 0);
          leftBtnClicked = 0;
          leftClickOnly = 0;
        }
        else {
          Dvalz = myMouse->currentAvgOutput.Z - myMouse->lastAvgOutput.Z;
          if (Dvalz < 0) {
            Dvalx = myMouse->currentAvgOutput.X - myMouse->lastAvgOutput.X;
            Dvaly = myMouse->currentAvgOutput.Y - myMouse->lastAvgOutput.Y;
            if (Dvalx < 0) {
              Dvalx = (-1)*Dvalx;
            }
            if (Dvaly < 0) {
              Dvaly = (-1)*Dvaly;
            }
            if ((Dvalx < UNCLICK_THRESHOLD) && (Dvaly < UNCLICK_THRESHOLD)) {
              if (clickIntended != 1) {
                click_point.X = myMouse->lastAvgOutput.X;
                click_point.Y = myMouse->lastAvgOutput.Y;
                clickIntended = 1;
              }
              myMouse->lastAvgOutput.Z = zPosition;
              return 0; /* not updating the cursor position */
            } else {
              click_point.X = 0;
              click_point.Y = 0;
              clickIntended = 0;
            }
```

```
791             } else {
                  click_point.X = 0;
                  click_point.Y = 0;
                  clickIntended = 0;
                }
              }

              if (leftBtnDblClicked == 1) {
                leftBtnDblClicked = 0;
              }
801
              if (rPos > POSI_ROLL_THRESHOLD) { /* going for right click */
                return 0;
              }

              transformX = transformPosition(xPosition - minXpos);  /* transferring the coordinate
                   to 0 */
              transformY = transformPosition(yPosition - minYpos);  /* transferring the coordinate
                   to 0 */

              /* checking for pixel jitter */
              if (!((X_sensorPos == 0) && (Y_sensorPos == 0))) {    /* not (the first time) */
811             diffX = transformX - X_sensorPos;
                diffY = transformY - Y_sensorPos;

                if (((diffX <= DIS_PIXEL_JITTER_POSI) || (diffX > DIS_PIXEL_JITTER_NEGI)) && ((
                     diffY <= DIS_PIXEL_JITTER_POSI) || (diffY > DIS_PIXEL_JITTER_NEGI))) {   /*
                     jitter detected */

                  printk(KERN_INFO "myMouse_driver: transferFunction: transformed movement of
                       sensor is treated as jitter.\n");
                  /* reporting the last movement */
                  input_report_abs(inputdev, ABS_X, X_sensorPos);
                  input_report_abs(inputdev, ABS_Y, Y_sensorPos);
                  input_report_abs(inputdev, ABS_Z, Z_sensorPos);
821
                } else {
                  X_sensorPos = transformX;
                  Y_sensorPos = transformY;
                  Z_sensorPos = distanceZ;
                  input_report_abs(inputdev, ABS_X, transformX);
                  input_report_abs(inputdev, ABS_Y, transformY);
                  input_report_abs(inputdev, ABS_Z, distanceZ);
                }
              } else {
831             X_sensorPos = transformX;
                Y_sensorPos = transformY;
                Z_sensorPos = distanceZ;
                input_report_abs(inputdev, ABS_X, transformX);
                input_report_abs(inputdev, ABS_Y, transformY);
                input_report_abs(inputdev, ABS_Z, distanceZ);
              }

              input_report_abs(inputdev, ABS_RX, rPos);
              input_report_abs(inputdev, ABS_RY, ePos);
841           input_report_abs(inputdev, ABS_RZ, aPos);
              input_sync(inputdev);

            } else if (distanceZ > 0) {

              if (distanceZ <= selthreshold){
                if (distanceZ <= surthreshold) {    /* if dotValue is with surthreshold then report
                     transformed position */
                  if (leftBtnClicked == 1) {  /* if left button was clicked then unclick it now */
                    input_report_key(inputdev, BTN_LEFT, 0);
                    leftBtnClicked = 0;
851                 leftClickOnly = 0;
                  }
                  else {
                    Dvalz = myMouse->currentAvgOutput.Z - myMouse->lastAvgOutput.Z;
                    if (Dvalz < 0) {
                      Dvalx = myMouse->currentAvgOutput.X - myMouse->lastAvgOutput.X;
                      Dvaly = myMouse->currentAvgOutput.Y - myMouse->lastAvgOutput.Y;
                      if (Dvalx < 0) {
                        Dvalx = (-1)*Dvalx;
                      }
861                   if (Dvaly < 0) {
                        Dvaly = (-1)*Dvaly;
                      }
                      if ((Dvalx < UNCLICK_THRESHOLD) && (Dvaly < UNCLICK_THRESHOLD)) {
                        if (clickIntended != 1) {
                          click_point.X = myMouse->lastAvgOutput.X;
                          click_point.Y = myMouse->lastAvgOutput.Y;
                          clickIntended = 1;
                        }
                        myMouse->lastAvgOutput.Z = zPosition;
871                     return 0; /* not updating the cursor position */
```

```
            } else {
              click_point.X = 0;
              click_point.Y = 0;
              clickIntended = 0;
            }
          } else {
            click_point.X = 0;
            click_point.Y = 0;
            clickIntended = 0;
          }
        }

        if (leftBtnDblClicked == 1) {
          leftBtnDblClicked = 0;
        }

        if (rPos > POSI_ROLL_THRESHOLD) { /* going for right click */
          return 0;
        }

        transformX = transformPosition(xPosition - minXpos);  /* transferring the
            coordinate to 0 */
        transformY = transformPosition(yPosition - minYpos);  /* transferring the
            coordinate to 0 */

        /* checking for pixel jitter -- check if the value is always positive */
        if (!((X_sensorPos == 0) && (Y_sensorPos == 0))) {    /* not (the first time) */
          diffX = transformX - X_sensorPos;
          diffY = transformY - Y_sensorPos;

          if (((diffX <= DIS_PIXEL_JITTER_POSI) || (diffX > DIS_PIXEL_JITTER_NEGI)) && ((
              diffY <= DIS_PIXEL_JITTER_POSI) || (diffY > DIS_PIXEL_JITTER_NEGI))) {   /*
               jitter detected */

            printk(KERN_INFO "myMouse_driver: transferFunction: transformed movement of
                sensor is treated as jitter.\n");
            /* reporting the last movement */
            input_report_abs(inputdev, ABS_X, X_sensorPos);
            input_report_abs(inputdev, ABS_Y, Y_sensorPos);
            input_report_abs(inputdev, ABS_Z, Z_sensorPos);

          } else {
            X_sensorPos = transformX;
            Y_sensorPos = transformY;
            Z_sensorPos = distanceZ;
            input_report_abs(inputdev, ABS_X, transformX);
            input_report_abs(inputdev, ABS_Y, transformY);
            input_report_abs(inputdev, ABS_Z, distanceZ);
          }
        } else {
          X_sensorPos = transformX;
          Y_sensorPos = transformY;
          Z_sensorPos = distanceZ;
          input_report_abs(inputdev, ABS_X, transformX);
          input_report_abs(inputdev, ABS_Y, transformY);
          input_report_abs(inputdev, ABS_Z, distanceZ);
        }

        input_report_abs(inputdev, ABS_RX, rPos);
        input_report_abs(inputdev, ABS_RY, ePos);
        input_report_abs(inputdev, ABS_RZ, aPos);
        input_sync(inputdev);

      } else if (distanceZ > surthreshold) {  /* its a simulated left button click */
        if (leftBtnClicked != 1) {
          if (clickIntended == 1) {
            printk(KERN_INFO "myMouse_driver: transferFunction: called from click
                intended.\n");
            xPosition = click_point.X;
            yPosition = click_point.Y;
            clickIntended = 0;
          }
          input_report_key(inputdev, BTN_LEFT, 1);
          leftBtnClicked = 1;
          leftClickOnly = 1;
        } else {
          /* taking care of double click */
          if (rPos <= LEFT_DBLCLICK_ANG) {
            if (leftBtnDblClicked == 0) {
              input_report_key(inputdev, BTN_LEFT, 0);
              input_sync(inputdev);
              input_report_key(inputdev, BTN_LEFT, 1);
              input_sync(inputdev);
              input_report_key(inputdev, BTN_LEFT, 0);
              input_sync(inputdev);
              input_report_key(inputdev, BTN_LEFT, 1);
              input_sync(inputdev);
```

```
                input_report_key(inputdev, BTN_LEFT, 0);
                input_sync(inputdev);
                leftBtnDblClicked = 1;
                leftBtnClicked = 0;
              }
            } else {
              if (leftBtnDblClicked == 1) {
                leftBtnDblClicked = 0;
              }
            }

            if (leftClickOnly == 1) {
              Dvalx = myMouse->currentAvgOutput.X - myMouse->lastAvgOutput.X;
              Dvaly = myMouse->currentAvgOutput.Y - myMouse->lastAvgOutput.Y;
              if (Dvalx < 0) {
                Dvalx = (-1)*Dvalx;
              }
              if (Dvaly < 0) {
                Dvaly = (-1)*Dvaly;
              }
              if ((Dvalx < UNCLICK_THRESHOLD) && (Dvaly < UNCLICK_THRESHOLD)) {
                return 0; /* not updating the cursor position */
              } else {
                leftClickOnly = 0;
              }
            }
          }

          if ((rPos < NEGI_ROLL_THRESHOLD) || (rPos > POSI_ROLL_THRESHOLD)) { /* going for
              left double or right click */
            return 0;
          }

          transformX = transformPosition(xPosition - minXpos);  /* transferring the
              coordinate to 0 */
          transformY = transformPosition(yPosition - minYpos);  /* transferring the
              coordinate to 0 */

          /* checking for pixel jitter -- check if the value is always positive */
          if (!((X_sensorPos == 0) && (Y_sensorPos == 0))) {    /* not (the first time) */
            diffX = transformX - X_sensorPos;
            diffY = transformY - Y_sensorPos;

            if (((diffX <= DIS_PIXEL_JITTER_POSI) || (diffX > DIS_PIXEL_JITTER_NEGI)) && ((
                diffY <= DIS_PIXEL_JITTER_POSI) || (diffY > DIS_PIXEL_JITTER_NEGI))) {   /*
                jitter detected */

              printk(KERN_INFO "myMouse_driver: transferFunction: transformed movement of
                  sensor is treated as jitter.\n");
              /* reporting the last movement */
              input_report_abs(inputdev, ABS_X, X_sensorPos);
              input_report_abs(inputdev, ABS_Y, Y_sensorPos);
              input_report_abs(inputdev, ABS_Z, Z_sensorPos);

            } else {
              X_sensorPos = transformX;
              Y_sensorPos = transformY;
              Z_sensorPos = distanceZ;
              input_report_abs(inputdev, ABS_X, transformX);
              input_report_abs(inputdev, ABS_Y, transformY);
              input_report_abs(inputdev, ABS_Z, distanceZ);
            }
          } else {
            X_sensorPos = transformX;
            Y_sensorPos = transformY;
            Z_sensorPos = distanceZ;
            input_report_abs(inputdev, ABS_X, transformX);
            input_report_abs(inputdev, ABS_Y, transformY);
            input_report_abs(inputdev, ABS_Z, distanceZ);
          }

          input_report_abs(inputdev, ABS_RX, rPos);
          input_report_abs(inputdev, ABS_RY, ePos);
          input_report_abs(inputdev, ABS_RZ, aPos);
          input_sync(inputdev);

        }
      } else {      /* this is one of the clutches (-) -- distanceZ > selthreshold -- same
          as left click */
        if (leftBtnClicked != 1) {
          if (clickIntended == 1) {
            printk(KERN_INFO "myMouse_driver: transferFunction: called from click intended
                .\n");
            xPosition = click_point.X;
            yPosition = click_point.Y;
            clickIntended = 0;
          }
```

```
            input_report_key(inputdev, BTN_LEFT, 1);
            leftBtnClicked = 1;
            leftClickOnly = 1;
          } else {
            /* taking care of double click */
            if (rPos <= LEFT_DBLCLICK_ANG) {
              if (leftBtnDblClicked == 0) {
                input_report_key(inputdev, BTN_LEFT, 0);
                input_sync(inputdev);
1041            input_report_key(inputdev, BTN_LEFT, 1);
                input_sync(inputdev);
                input_report_key(inputdev, BTN_LEFT, 0);
                input_sync(inputdev);
                input_report_key(inputdev, BTN_LEFT, 1);
                input_sync(inputdev);
                input_report_key(inputdev, BTN_LEFT, 0);
                input_sync(inputdev);
                leftBtnDblClicked = 1;
                leftBtnClicked = 0;
1051          }
            } else {
              if (leftBtnDblClicked == 1) {
                leftBtnDblClicked = 0;
              }
            }

            if (leftClickOnly == 1) {
              Dvalx = myMouse->currentAvgOutput.X - myMouse->lastAvgOutput.X;
              Dvaly = myMouse->currentAvgOutput.Y - myMouse->lastAvgOutput.Y;
1061          if (Dvalx < 0) {
                Dvalx = (-1)*Dvalx;
              }
              if (Dvaly < 0) {
                Dvaly = (-1)*Dvaly;
              }
              if ((Dvalx < UNCLICK_THRESHOLD) && (Dvaly < UNCLICK_THRESHOLD)) {
                return 0; /* not updating the cursor position */
              } else {
                leftClickOnly = 0;
1071          }
            }
          }

          if ((rPos < NEGI_ROLL_THRESHOLD) || (rPos > POSI_ROLL_THRESHOLD)) { /* going for
               left double or right click */
            return 0;
          }

          transformX = transformPosition(xPosition - minXpos);  /* transferring the
               coordinate to 0 */
          transformY = transformPosition(yPosition - minYpos);  /* transferring the
               coordinate to 0 */
1081
          /* checking for pixel jitter -- check if the value is always positive */
          if (!((X_sensorPos == 0) && (Y_sensorPos == 0))) {     /* not (the first time) */
            diffX = transformX - X_sensorPos;
            diffY = transformY - Y_sensorPos;

            if (((diffX <= DIS_PIXEL_JITTER_POSI) || (diffX > DIS_PIXEL_JITTER_NEGI)) && ((
                 diffY <= DIS_PIXEL_JITTER_POSI) || (diffY > DIS_PIXEL_JITTER_NEGI))) {   /*
                 jitter detected */

              printk(KERN_INFO "myMouse_driver: transferFunction: transformed movement of
                   sensor is treated as jitter.\n");
              /* reporting the last movement */
1091          input_report_abs(inputdev, ABS_X, X_sensorPos);
              input_report_abs(inputdev, ABS_Y, Y_sensorPos);
              input_report_abs(inputdev, ABS_Z, Z_sensorPos);

            } else {
              X_sensorPos = transformX;
              Y_sensorPos = transformY;
              Z_sensorPos = distanceZ;
              input_report_abs(inputdev, ABS_X, transformX);
              input_report_abs(inputdev, ABS_Y, transformY);
1101          input_report_abs(inputdev, ABS_Z, distanceZ);
            }
          } else {
            X_sensorPos = transformX;
            Y_sensorPos = transformY;
            Z_sensorPos = distanceZ;
            input_report_abs(inputdev, ABS_X, transformX);
            input_report_abs(inputdev, ABS_Y, transformY);
            input_report_abs(inputdev, ABS_Z, distanceZ);
          }
1111
          input_report_abs(inputdev, ABS_RX, rPos);
```

```
            input_report_abs(inputdev, ABS_RY, ePos);
            input_report_abs(inputdev, ABS_RZ, aPos);
            input_sync(inputdev);

          }
        } else  {       /* then dotValue is greater, i.e., the point is above the surface */
          if (((-1)*distanceZ) <= POSI_CLUTCH_THRESHOLD)  {
            if (leftBtnClicked == 1) {  /* if left button was clicked then unclick it now */
1121          input_report_key(inputdev, BTN_LEFT, 0);
              leftBtnClicked = 0;
              leftClickOnly = 0;
            }
            else {
              Dvalz = myMouse->currentAvgOutput.Z - myMouse->lastAvgOutput.Z;
              if (Dvalz < 0) {
                Dvalx = myMouse->currentAvgOutput.X - myMouse->lastAvgOutput.X;
                Dvaly = myMouse->currentAvgOutput.Y - myMouse->lastAvgOutput.Y;
                if (Dvalx < 0) {
1131              Dvalx = (-1)*Dvalx;
                }
                if (Dvaly < 0) {
                  Dvaly = (-1)*Dvaly;
                }
                if ((Dvalx < UNCLICK_THRESHOLD) && (Dvaly < UNCLICK_THRESHOLD)) {
                  if (clickIntended != 1) {
                    click_point.X = myMouse->lastAvgOutput.X;
                    click_point.Y = myMouse->lastAvgOutput.Y;
                    clickIntended = 1;
1141              }
                  myMouse->lastAvgOutput.Z = zPosition;
                  return 0; /* not updating the cursor position */
                } else {
                  click_point.X = 0;
                  click_point.Y = 0;
                  clickIntended = 0;
                }
              } else {
                click_point.X = 0;
1151            click_point.Y = 0;
                clickIntended = 0;
              }
            }

            if (leftBtnDblClicked == 1) {
              leftBtnDblClicked = 0;
            }

            if (rPos > POSI_ROLL_THRESHOLD) { /* going for right click */
1161          return 0;
            }

            transformX = transformPosition(xPosition - minXpos);  /* transferring the
                coordinate to 0 */
            transformY = transformPosition(yPosition - minYpos);  /* transferring the
                coordinate to 0 */

            /* checking for pixel jitter -- check if the value is always positive */
            if (!((X_sensorPos == 0) && (Y_sensorPos == 0))) {    /* not (the first time) */
              diffX = transformX - X_sensorPos;
              diffY = transformY - Y_sensorPos;
1171
              if (((diffX <= DIS_PIXEL_JITTER_POSI) || (diffX > DIS_PIXEL_JITTER_NEGI)) && ((
                  diffY <= DIS_PIXEL_JITTER_POSI) || (diffY > DIS_PIXEL_JITTER_NEGI))) {   /*
                  jitter detected */

                printk(KERN_INFO "myMouse_driver: transferFunction: transformed movement of
                    sensor is treated as jitter.\n");
                /* reporting the last movement */
                input_report_abs(inputdev, ABS_X, X_sensorPos);
                input_report_abs(inputdev, ABS_Y, Y_sensorPos);
                input_report_abs(inputdev, ABS_Z, Z_sensorPos);

              } else {
1181            X_sensorPos = transformX;
                Y_sensorPos = transformY;
                Z_sensorPos = distanceZ;
                input_report_abs(inputdev, ABS_X, transformX);
                input_report_abs(inputdev, ABS_Y, transformY);
                input_report_abs(inputdev, ABS_Z, distanceZ);
              }
            } else {
              X_sensorPos = transformX;
              Y_sensorPos = transformY;
1191          Z_sensorPos = distanceZ;
              input_report_abs(inputdev, ABS_X, transformX);
              input_report_abs(inputdev, ABS_Y, transformY);
              input_report_abs(inputdev, ABS_Z, distanceZ);
```

```
            }

            input_report_abs(inputdev, ABS_RX, rPos);
            input_report_abs(inputdev, ABS_RY, ePos);
            input_report_abs(inputdev, ABS_RZ, aPos);
            input_sync(inputdev);

          } else {        /* this is one of the clutches (+) */
            /* reporting nothing to the input event subsystem */
            Z_sensorPos = distanceZ;
            return 0;
          }
        }

        myMouse->lastAvgOutput.X = xPosition;
        myMouse->lastAvgOutput.Y = yPosition;
        myMouse->lastAvgOutput.Z = zPosition;
        myMouse->lastAvgOutput.A = aPos;
        myMouse->lastAvgOutput.E = ePos;
        myMouse->lastAvgOutput.R = rPos;

        break;
      case 2:       /* rframe on xz plane [touchscreen mode, usual] (here assuming the base and
           the screen have same x-axis, in regards to coordinates) */
        distanceZ = (minYpos - yPosition);

        if (distanceZ == 0) {   /* then the point is on the plane */

          if (leftBtnClicked == 1) {  /* if left button was clicked then unclick it now */
            input_report_key(inputdev, BTN_LEFT, 0);
            leftBtnClicked = 0;
            leftClickOnly = 0;
          }
          else {
            Dvaly = myMouse->currentAvgOutput.Y - myMouse->lastAvgOutput.Y;
            if (Dvaly < 0) {
              Dvalx = myMouse->currentAvgOutput.X - myMouse->lastAvgOutput.X;
              Dvalz = myMouse->currentAvgOutput.Z - myMouse->lastAvgOutput.Z;
              if (Dvalx < 0) {
                Dvalx = (-1)*Dvalx;
              }
              if (Dvalz < 0) {
                Dvalz = (-1)*Dvalz;
              }
              if ((Dvalx < UNCLICK_THRESHOLD) && (Dvalz < UNCLICK_THRESHOLD)) {
                if (clickIntended != 1) {
                  click_point.X = myMouse->lastAvgOutput.X;
                  click_point.Z = myMouse->lastAvgOutput.Z;
                  clickIntended = 1;
                }
                myMouse->lastAvgOutput.Y = yPosition;
                return 0; /* not updating the cursor position */
              } else {
                click_point.X = 0;
                click_point.Z = 0;
                clickIntended = 0;
              }
            } else {
              click_point.X = 0;
              click_point.Z = 0;
              clickIntended = 0;
            }
          }

          if (leftBtnDblClicked == 1) {
            leftBtnDblClicked = 0;
          }

          transformX = transformPosition(xPosition - minXpos);  /* transferring the coordinate
               to 0 */
          transformY = transformPosition(zPosition - minZpos);  /* doing the swap: as z value
               is the height */

          /* as the base is assumed to be always behind and under the rframe -- following swap
               is necessary */
          /* this has to be done as screens 0,0 starts at the upper left corner and for rframe
               (with the above assumtion) it is lower left corner */
          transformY = (y_src_resolution - transformY);   /* this is making y=0 to y=screen
               height visa versa and in between */


          /* checking for pixel jitter -- check if the value is always positive */
          if (!((X_sensorPos == 0) && (Y_sensorPos == 0))) {    /* not (the first time) */
            diffX = transformX - X_sensorPos;
            diffY = transformY - Y_sensorPos;
```

```
            if (((diffX <= DIS_PIXEL_JITTER_POSI) || (diffX > DIS_PIXEL_JITTER_NEGI)) && ((
                diffY <= DIS_PIXEL_JITTER_POSI) || (diffY > DIS_PIXEL_JITTER_NEGI))) {   /*
                jitter detected */

              printk(KERN_INFO "myMouse_driver: transferFunction: transformed movement of
                  sensor is treated as jitter.\n");
              /* reporting the last movement */
              input_report_abs(inputdev, ABS_X, X_sensorPos);
              input_report_abs(inputdev, ABS_Y, Y_sensorPos);
              input_report_abs(inputdev, ABS_Z, Z_sensorPos);
            } else {
              X_sensorPos = transformX;
              Y_sensorPos = transformY;
              Z_sensorPos = distanceZ;
              input_report_abs(inputdev, ABS_X, transformX);
              input_report_abs(inputdev, ABS_Y, transformY);
              input_report_abs(inputdev, ABS_Z, distanceZ);
            }
          } else {
            X_sensorPos = transformX;
            Y_sensorPos = transformY;
            Z_sensorPos = distanceZ;
            input_report_abs(inputdev, ABS_X, transformX);
            input_report_abs(inputdev, ABS_Y, transformY);
            input_report_abs(inputdev, ABS_Z, distanceZ);
          }

          input_report_abs(inputdev, ABS_RX, rPos);
          input_report_abs(inputdev, ABS_RY, ePos);
          input_report_abs(inputdev, ABS_RZ, aPos);
          input_sync(inputdev);

        } else if (distanceZ > 0) {

          if (distanceZ <= selthreshold){
            if (distanceZ <= surthreshold) {     /* if dotValue is with surthreshold then report
                transformed position */

              if (leftBtnClicked == 1) {  /* if left button was clicked then unclick it now */
                input_report_key(inputdev, BTN_LEFT, 0);
                leftBtnClicked = 0;
                leftClickOnly = 0;
              }
              else {
                Dvaly = myMouse->currentAvgOutput.Y - myMouse->lastAvgOutput.Y;
                if (Dvaly < 0) {
                  Dvalx = myMouse->currentAvgOutput.X - myMouse->lastAvgOutput.X;
                  Dvalz = myMouse->currentAvgOutput.Z - myMouse->lastAvgOutput.Z;
                  if (Dvalx < 0) {
                    Dvalx = (-1)*Dvalx;
                  }
                  if (Dvalz < 0) {
                    Dvalz = (-1)*Dvalz;
                  }
                  if ((Dvalx < UNCLICK_THRESHOLD) && (Dvalz < UNCLICK_THRESHOLD)) {
                    if (clickIntended != 1) {
                      click_point.X = myMouse->lastAvgOutput.X;
                      click_point.Z = myMouse->lastAvgOutput.Z;
                      clickIntended = 1;
                    }
                    myMouse->lastAvgOutput.Y = yPosition;
                    return 0; /* not updating the cursor position */
                  } else {
                    click_point.X = 0;
                    click_point.Z = 0;
                    clickIntended = 0;
                  }
                } else {
                  click_point.X = 0;
                  click_point.Z = 0;
                  clickIntended = 0;
                }
              }

              if (leftBtnDblClicked == 1) {
                leftBtnDblClicked = 0;
              }

              transformX = transformPosition(xPosition - minXpos);  /* transferring the
                  coordinate to 0 */
              transformY = transformPosition(zPosition - minZpos);  /* doing the swap: as z
                  value is the height */

              /* as the base is assumed to be always behind and under the rframe -- following
                  swap is necessary */
              /* this has to be done as screens 0,0 starts at the upper left corner and for
                  rframe (with the above assumtion) it is lower left corner */
```

```
    transformY = (y_src_resolution - transformY);   /* this is making y=0 to y=screen
          height visa versa and in between */

    /* checking for pixel jitter -- check if the value is always positive */
    if (!((X_sensorPos == 0) && (Y_sensorPos == 0))) {    /* not (the first time) */
      diffX = transformX - X_sensorPos;
      diffY = transformY - Y_sensorPos;

1361  if (((diffX <= DIS_PIXEL_JITTER_POSI) || (diffX > DIS_PIXEL_JITTER_NEGI)) && ((
          diffY <= DIS_PIXEL_JITTER_POSI) || (diffY > DIS_PIXEL_JITTER_NEGI))) {   /*
           jitter detected */

        printk(KERN_INFO "myMouse_driver: transferFunction: transformed movement of
            sensor is treated as jitter.\n");
        /* reporting the last movement */
        input_report_abs(inputdev, ABS_X, X_sensorPos);
        input_report_abs(inputdev, ABS_Y, Y_sensorPos);
        input_report_abs(inputdev, ABS_Z, Z_sensorPos);
      } else {
        X_sensorPos = transformX;
        Y_sensorPos = transformY;
1371    Z_sensorPos = distanceZ;
        input_report_abs(inputdev, ABS_X, transformX);
        input_report_abs(inputdev, ABS_Y, transformY);
        input_report_abs(inputdev, ABS_Z, distanceZ);
      }
    } else {
      X_sensorPos = transformX;
      Y_sensorPos = transformY;
      Z_sensorPos = distanceZ;
      input_report_abs(inputdev, ABS_X, transformX);
1381  input_report_abs(inputdev, ABS_Y, transformY);
      input_report_abs(inputdev, ABS_Z, distanceZ);
    }

    input_report_abs(inputdev, ABS_RX, rPos);
    input_report_abs(inputdev, ABS_RY, ePos);
    input_report_abs(inputdev, ABS_RZ, aPos);
    input_sync(inputdev);

  } else if (distanceZ > surthreshold) {  /* its a simulated left button click */
1391
    if (leftBtnClicked != 1) {
      if (clickIntended == 1) {
        printk(KERN_INFO "myMouse_driver: transferFunction: called from click
            intended.\n");
        xPosition = click_point.X;
        zPosition = click_point.Z;
        clickIntended = 0;
      }
      input_report_key(inputdev, BTN_LEFT, 1);
      leftBtnClicked = 1;
1401  leftClickOnly = 1;
    } else {
      /* taking care of double click */
      if (rPos <= LEFT_DBLCLICK_ANG) {
        if (leftBtnDblClicked == 0) {
          input_report_key(inputdev, BTN_LEFT, 0);
          input_sync(inputdev);
          input_report_key(inputdev, BTN_LEFT, 1);
          input_sync(inputdev);
          input_report_key(inputdev, BTN_LEFT, 0);
1411      input_sync(inputdev);
          input_report_key(inputdev, BTN_LEFT, 1);
          input_sync(inputdev);
          input_report_key(inputdev, BTN_LEFT, 0);
          input_sync(inputdev);
          leftBtnDblClicked = 1;
          leftBtnClicked = 0;
        }
      } else {
        if (leftBtnDblClicked == 1) {
1421      leftBtnDblClicked = 0;
        }
      }

      if (leftClickOnly == 1) {
        Dvalx = myMouse->currentAvgOutput.X - myMouse->lastAvgOutput.X;
        Dvalz = myMouse->currentAvgOutput.Z - myMouse->lastAvgOutput.Z;
        if (Dvalx < 0) {
          Dvalx = (-1)*Dvalx;
        }
1431    if (Dvalz < 0) {
          Dvalz = (-1)*Dvalz;
        }
        if ((Dvalx < UNCLICK_THRESHOLD) && (Dvalz < UNCLICK_THRESHOLD)) {
          return 0; /* not updating the cursor position */
```

```
                } else {
                  leftClickOnly = 0;
                }
              }
            }

            transformX = transformPosition(xPosition - minXpos);  /* transferring the
                coordinate to 0 */
            transformY = transformPosition(zPosition - minZpos);  /* doing the swap: as z
                value is the height */

            /* as the base is assumed to be always behind and under the rframe -- following
                swap is necessary */
            /* this has to be done as screens 0,0 starts at the upper left corner and for
                rframe (with the above assumtion) it is lower left corner */
            transformY = (y_src_resolution - transformY);   /* this is making y=0 to y=screen
                 height visa versa and in between */

            /* checking for pixel jitter -- check if the value is always positive */
            if (!((X_sensorPos == 0) && (Y_sensorPos == 0))) {    /* not (the first time) */
              diffX = transformX - X_sensorPos;
              diffY = transformY - Y_sensorPos;

              if (((diffX <= DIS_PIXEL_JITTER_POSI) || (diffX > DIS_PIXEL_JITTER_NEGI)) && ((
                  diffY <= DIS_PIXEL_JITTER_POSI) || (diffY > DIS_PIXEL_JITTER_NEGI))) {   /*
                   jitter detected */

                printk(KERN_INFO "myMouse_driver: transferFunction: transformed movement of
                    sensor is treated as jitter.\n");
                /* reporting the last movement */
                input_report_abs(inputdev, ABS_X, X_sensorPos);
                input_report_abs(inputdev, ABS_Y, Y_sensorPos);
                input_report_abs(inputdev, ABS_Z, Z_sensorPos);
              } else {
                X_sensorPos = transformX;
                Y_sensorPos = transformY;
                Z_sensorPos = distanceZ;
                input_report_abs(inputdev, ABS_X, transformX);
                input_report_abs(inputdev, ABS_Y, transformY);
                input_report_abs(inputdev, ABS_Z, distanceZ);
              }
            } else {
              X_sensorPos = transformX;
              Y_sensorPos = transformY;
              Z_sensorPos = distanceZ;
              input_report_abs(inputdev, ABS_X, transformX);
              input_report_abs(inputdev, ABS_Y, transformY);
              input_report_abs(inputdev, ABS_Z, distanceZ);
            }

            input_report_abs(inputdev, ABS_RX, rPos);
            input_report_abs(inputdev, ABS_RY, ePos);
            input_report_abs(inputdev, ABS_RZ, aPos);
            input_sync(inputdev);

          }
        } else {      /* this is one of the clutches (-) -- distanceZ > selthreshold -- same
            as left click */

          if (leftBtnClicked != 1) {
            if (clickIntended == 1) {
              printk(KERN_INFO "myMouse_driver: transferFunction: called from click intended
                  .\n");
              xPosition = click_point.X;
              zPosition = click_point.Z;
              clickIntended = 0;
            }
            input_report_key(inputdev, BTN_LEFT, 1);
            leftBtnClicked = 1;
            leftClickOnly = 1;
          } else {
            /* taking care of double click */
            if (rPos <= LEFT_DBLCLICK_ANG) {
              if (leftBtnDblClicked == 0) {
                input_report_key(inputdev, BTN_LEFT, 0);
                input_sync(inputdev);
                input_report_key(inputdev, BTN_LEFT, 1);
                input_sync(inputdev);
                input_report_key(inputdev, BTN_LEFT, 0);
                input_sync(inputdev);
                input_report_key(inputdev, BTN_LEFT, 1);
                input_sync(inputdev);
                input_report_key(inputdev, BTN_LEFT, 0);
                input_sync(inputdev);
                leftBtnDblClicked = 1;
                leftBtnClicked = 0;
              }
```

```
            } else {
              if (leftBtnDblClicked == 1) {
                leftBtnDblClicked = 0;
              }
            }

            if (leftClickOnly == 1) {
              Dvalx = myMouse->currentAvgOutput.X - myMouse->lastAvgOutput.X;
1521          Dvalz = myMouse->currentAvgOutput.Z - myMouse->lastAvgOutput.Z;
              if (Dvalx < 0) {
                Dvalx = (-1)*Dvalx;
              }
              if (Dvalz < 0) {
                Dvalz = (-1)*Dvalz;
              }
              if ((Dvalx < UNCLICK_THRESHOLD) && (Dvalz < UNCLICK_THRESHOLD)) {
                return 0; /* not updating the cursor position */
              } else {
1531            leftClickOnly = 0;
              }
            }
          }

          transformX = transformPosition(xPosition - minXpos);  /* transferring the
               coordinate to 0 */
          transformY = transformPosition(zPosition - minZpos);  /* doing the swap: as z value
                is the height */

          /* as the base is assumed to be always behind and under the rframe -- following
               swap is necessary */
          /* this has to be done as screens 0,0 starts at the upper left corner and for
               rframe (with the above assumtion) it is lower left corner */
1541      transformY = (y_src_resolution - transformY);   /* this is making y=0 to y=screen
               height visa versa and in between */

          /* checking for pixel jitter -- check if the value is always positive */
          if (!((X_sensorPos == 0) && (Y_sensorPos == 0))) {    /* not (the first time) */
            diffX = transformX - X_sensorPos;
            diffY = transformY - Y_sensorPos;

            if (((diffX <= DIS_PIXEL_JITTER_POSI) || (diffX > DIS_PIXEL_JITTER_NEGI)) && ((
                 diffY <= DIS_PIXEL_JITTER_POSI) || (diffY > DIS_PIXEL_JITTER_NEGI))) {   /*
                 jitter detected */

              printk(KERN_INFO "myMouse_driver: transferFunction: transformed movement of
                   sensor is treated as jitter.\n");
1551          /* reporting the last movement */
              input_report_abs(inputdev, ABS_X, X_sensorPos);
              input_report_abs(inputdev, ABS_Y, Y_sensorPos);
              input_report_abs(inputdev, ABS_Z, Z_sensorPos);
            } else {
              X_sensorPos = transformX;
              Y_sensorPos = transformY;
              Z_sensorPos = distanceZ;
              input_report_abs(inputdev, ABS_X, transformX);
              input_report_abs(inputdev, ABS_Y, transformY);
1561          input_report_abs(inputdev, ABS_Z, distanceZ);
            }
          } else {
            X_sensorPos = transformX;
            Y_sensorPos = transformY;
            Z_sensorPos = distanceZ;
            input_report_abs(inputdev, ABS_X, transformX);
            input_report_abs(inputdev, ABS_Y, transformY);
            input_report_abs(inputdev, ABS_Z, distanceZ);
          }
1571
          input_report_abs(inputdev, ABS_RX, rPos);
          input_report_abs(inputdev, ABS_RY, ePos);
          input_report_abs(inputdev, ABS_RZ, aPos);
          input_sync(inputdev);
        }

      } else  {       /* then dotValue is greater, i.e., the point is above the surface */
        if (((-1)*distanceZ) <= POSI_CLUTCH_THRESHOLD)  {

1581      if (leftBtnClicked == 1) {  /* if left button was clicked then unclick it now */
            input_report_key(inputdev, BTN_LEFT, 0);
            leftBtnClicked = 0;
            leftClickOnly = 0;
          }
          else {
            Dvaly = myMouse->currentAvgOutput.Y - myMouse->lastAvgOutput.Y;
            if (Dvaly < 0) {
              Dvalx = myMouse->currentAvgOutput.X - myMouse->lastAvgOutput.X;
              Dvalz = myMouse->currentAvgOutput.Z - myMouse->lastAvgOutput.Z;
1591          if (Dvalx < 0) {
```

```
                Dvalx = (-1)*Dvalx;
              }
              if (Dvalz < 0) {
                Dvalz = (-1)*Dvalz;
              }
              if ((Dvalx < UNCLICK_THRESHOLD) && (Dvalz < UNCLICK_THRESHOLD)) {
                if (clickIntended != 1) {
                  click_point.X = myMouse->lastAvgOutput.X;
                  click_point.Z = myMouse->lastAvgOutput.Z;
1601              clickIntended = 1;
                }
                myMouse->lastAvgOutput.Y = yPosition;
                return 0; /* not updating the cursor position */
              } else {
                click_point.X = 0;
                click_point.Z = 0;
                clickIntended = 0;
              }
            } else {
1611          click_point.X = 0;
              click_point.Z = 0;
              clickIntended = 0;
            }
          }

          if (leftBtnDblClicked == 1) {
            leftBtnDblClicked = 0;
          }

1621      transformX = transformPosition(xPosition - minXpos);  /* transferring the
               coordinate to 0 */
          transformY = transformPosition(zPosition - minZpos);  /* doing the swap: as z value
                is the height */

          /* as the base is assumed to be always behind and under the rframe -- following
               swap is necessary */
          /* this has to be done as screens 0,0 starts at the upper left corner and for
               rframe (with the above assumtion) it is lower left corner */
          transformY = (y_src_resolution - transformY);   /* this is making y=0 to y=screen
               height visa versa and in between */

          /* checking for pixel jitter -- check if the value is always positive */
          if (!((X_sensorPos == 0) && (Y_sensorPos == 0))) {    /* not (the first time) */
            diffX = transformX - X_sensorPos;
1631        diffY = transformY - Y_sensorPos;

            if (((diffX <= DIS_PIXEL_JITTER_POSI) || (diffX > DIS_PIXEL_JITTER_NEGI)) && ((
                 diffY <= DIS_PIXEL_JITTER_POSI) || (diffY > DIS_PIXEL_JITTER_NEGI))) {   /*
                 jitter detected */

              printk(KERN_INFO "myMouse_driver: transferFunction: transformed movement of
                   sensor is treated as jitter.\n");
              /* reporting the last movement */
              input_report_abs(inputdev, ABS_X, X_sensorPos);
              input_report_abs(inputdev, ABS_Y, Y_sensorPos);
              input_report_abs(inputdev, ABS_Z, Z_sensorPos);
            } else {
1641          X_sensorPos = transformX;
              Y_sensorPos = transformY;
              Z_sensorPos = distanceZ;
              input_report_abs(inputdev, ABS_X, transformX);
              input_report_abs(inputdev, ABS_Y, transformY);
              input_report_abs(inputdev, ABS_Z, distanceZ);
            }
          } else {
            X_sensorPos = transformX;
            Y_sensorPos = transformY;
1651        Z_sensorPos = distanceZ;
            input_report_abs(inputdev, ABS_X, transformX);
            input_report_abs(inputdev, ABS_Y, transformY);
            input_report_abs(inputdev, ABS_Z, distanceZ);
          }

          input_report_abs(inputdev, ABS_RX, rPos);
          input_report_abs(inputdev, ABS_RY, ePos);
          input_report_abs(inputdev, ABS_RZ, aPos);
          input_sync(inputdev);
1661
        } else {      /* this is one of the clutches (+) */
          /* reporting nothing to the input event subsystem */
          Z_sensorPos = distanceZ;
          return 0;
        }
      }

      myMouse->lastAvgOutput.X = xPosition;
      myMouse->lastAvgOutput.Y = yPosition;
```

```
1671        myMouse->lastAvgOutput.Z = zPosition;
            myMouse->lastAvgOutput.A = aPos;
            myMouse->lastAvgOutput.E = ePos;
            myMouse->lastAvgOutput.R = rPos;

            break;
          case 3:       /* rframe on yz plane [projection mode, may be] (here assuming the base and
                the screen have same x-axis, in regards to coordinates) */
            distanceZ = (minXpos - xPosition);

            if (distanceZ == 0) {   /* then the point is on the plane */
1681
              if (leftBtnClicked == 1) {  /* if left button was clicked then unclick it now */
                input_report_key(inputdev, BTN_LEFT, 0);
                leftBtnClicked = 0;
                leftClickOnly = 0;
              }
              else {
                Dvalx = myMouse->currentAvgOutput.X - myMouse->lastAvgOutput.X;
                if (Dvalx < 0) {
                  Dvaly = myMouse->currentAvgOutput.Y - myMouse->lastAvgOutput.Y;
1691              Dvalz = myMouse->currentAvgOutput.Z - myMouse->lastAvgOutput.Z;
                  if (Dvaly < 0) {
                    Dvaly = (-1)*Dvaly;
                  }
                  if (Dvalz < 0) {
                    Dvalz = (-1)*Dvalz;
                  }
                  if ((Dvaly < UNCLICK_THRESHOLD) && (Dvalz < UNCLICK_THRESHOLD)) {
                    if (clickIntended != 1) {
                      click_point.Y = myMouse->lastAvgOutput.Y;
1701                  click_point.Z = myMouse->lastAvgOutput.Z;
                      clickIntended = 1;
                    }
                    myMouse->lastAvgOutput.X = xPosition;
                    return 0; /* not updating the cursor position */
                  } else {
                    click_point.Y = 0;
                    click_point.Z = 0;
                    clickIntended = 0;
                  }
1711              } else {
                    click_point.Y = 0;
                    click_point.Z = 0;
                    clickIntended = 0;
                  }
              }

              if (leftBtnDblClicked == 1) {
                leftBtnDblClicked = 0;
              }
1721

              transformX = transformPosition(yPosition - minYpos);
              transformY = transformPosition(zPosition - minZpos);  /* doing the swap: as z value
                   is the height */

              /* as the base is assumed to be always behind and under the rframe -- following swap
                   is necessary */
              /* this has to be done as screens 0,0 starts at the upper left corner and for rframe
                   (with the above assumtion) it is lower left corner */
              transformX = (x_src_resolution - transformX);   /* this is making x=0 to y=screen
                   width visa versa and in between */
              transformY = (y_src_resolution - transformY);   /* this is making y=0 to y=screen
                   height visa versa and in between */

1731          /* checking for pixel jitter -- check if the value is always positive */
              if (!((X_sensorPos == 0) && (Y_sensorPos == 0))) {    /* not (the first time) */
                diffX = transformX - X_sensorPos;
                diffY = transformY - Y_sensorPos;

                if (((diffX <= DIS_PIXEL_JITTER_POSI) || (diffX > DIS_PIXEL_JITTER_NEGI)) && ((
                     diffY <= DIS_PIXEL_JITTER_POSI) || (diffY > DIS_PIXEL_JITTER_NEGI))) {   /*
                     jitter detected */

                  printk(KERN_INFO "myMouse_driver: transferFunction: transformed movement of
                       sensor is treated as jitter.\n");
                  /* reporting the last movement */
                  input_report_abs(inputdev, ABS_X, X_sensorPos);
1741              input_report_abs(inputdev, ABS_Y, Y_sensorPos);
                  input_report_abs(inputdev, ABS_Z, Z_sensorPos);
                } else {
                  X_sensorPos = transformX;
                  Y_sensorPos = transformY;
                  Z_sensorPos = distanceZ;
                  input_report_abs(inputdev, ABS_X, transformX);
                  input_report_abs(inputdev, ABS_Y, transformY);
```

```
            input_report_abs(inputdev, ABS_Z, distanceZ);
          }
1751      } else {
            X_sensorPos = transformX;
            Y_sensorPos = transformY;
            Z_sensorPos = distanceZ;
            input_report_abs(inputdev, ABS_X, transformX);
            input_report_abs(inputdev, ABS_Y, transformY);
            input_report_abs(inputdev, ABS_Z, distanceZ);
          }

          input_report_abs(inputdev, ABS_RX, rPos);
1761      input_report_abs(inputdev, ABS_RY, ePos);
          input_report_abs(inputdev, ABS_RZ, aPos);
          input_sync(inputdev);

        } else if (distanceZ > 0) {

          if (distanceZ <= selthreshold){
            if (distanceZ <= surthreshold) {     /* if dotValue is with surthreshold then report
                transformed position */

              if (leftBtnClicked == 1) {  /* if left button was clicked then unclick it now */
1771            input_report_key(inputdev, BTN_LEFT, 0);
                leftBtnClicked = 0;
                leftClickOnly = 0;
              }
              else {
                Dvalx = myMouse->currentAvgOutput.X - myMouse->lastAvgOutput.X;
                if (Dvalx < 0) {
                  Dvaly = myMouse->currentAvgOutput.Y - myMouse->lastAvgOutput.Y;
                  Dvalz = myMouse->currentAvgOutput.Z - myMouse->lastAvgOutput.Z;
                  if (Dvaly < 0) {
1781                Dvaly = (-1)*Dvaly;
                  }
                  if (Dvalz < 0) {
                    Dvalz = (-1)*Dvalz;
                  }
                  if ((Dvaly < UNCLICK_THRESHOLD) && (Dvalz < UNCLICK_THRESHOLD)) {
                    if (clickIntended != 1) {
                      click_point.Y = myMouse->lastAvgOutput.Y;
                      click_point.Z = myMouse->lastAvgOutput.Z;
                      clickIntended = 1;
1791                }
                    myMouse->lastAvgOutput.X = xPosition;
                    return 0; /* not updating the cursor position */
                  } else {
                    click_point.Y = 0;
                    click_point.Z = 0;
                    clickIntended = 0;
                  }
                } else {
                  click_point.Y = 0;
1801              click_point.Z = 0;
                  clickIntended = 0;
                }
              }

              if (leftBtnDblClicked == 1) {
                leftBtnDblClicked = 0;
              }

              transformX = transformPosition(yPosition - minYpos);
1811          transformY = transformPosition(zPosition - minZpos);  /* doing the swap: as z
                  value is the height */

              /* as the base is assumed to be always behind and under the rframe -- following
                  swap is necessary */
              /* this has to be done as screens 0,0 starts at the upper left corner and for
                  rframe (with the above assumtion) it is lower left corner */
              transformX = (x_src_resolution - transformX);    /* this is making x=0 to y=screen
                    width visa versa and in between */
              transformY = (y_src_resolution - transformY);    /* this is making y=0 to y=screen
                    height visa versa and in between */

              /* checking for pixel jitter -- check if the value is always positive */
              if (!((X_sensorPos == 0) && (Y_sensorPos == 0))) {     /* not (the first time) */
                diffX = transformX - X_sensorPos;
1821            diffY = transformY - Y_sensorPos;

                if (((diffX <= DIS_PIXEL_JITTER_POSI) || (diffX > DIS_PIXEL_JITTER_NEGI)) && ((
                    diffY <= DIS_PIXEL_JITTER_POSI) || (diffY > DIS_PIXEL_JITTER_NEGI))) {    /*
                      jitter detected */

                  printk(KERN_INFO "myMouse_driver: transferFunction: transformed movement of
                      sensor is treated as jitter.\n");
                  /* reporting the last movement */
```

```
                input_report_abs(inputdev, ABS_X, X_sensorPos);
                input_report_abs(inputdev, ABS_Y, Y_sensorPos);
                input_report_abs(inputdev, ABS_Z, Z_sensorPos);
              } else {
1831            X_sensorPos = transformX;
                Y_sensorPos = transformY;
                Z_sensorPos = distanceZ;
                input_report_abs(inputdev, ABS_X, transformX);
                input_report_abs(inputdev, ABS_Y, transformY);
                input_report_abs(inputdev, ABS_Z, distanceZ);
              }
            } else {
              X_sensorPos = transformX;
              Y_sensorPos = transformY;
1841          Z_sensorPos = distanceZ;
              input_report_abs(inputdev, ABS_X, transformX);
              input_report_abs(inputdev, ABS_Y, transformY);
              input_report_abs(inputdev, ABS_Z, distanceZ);
            }

            input_report_abs(inputdev, ABS_RX, rPos);
            input_report_abs(inputdev, ABS_RY, ePos);
            input_report_abs(inputdev, ABS_RZ, aPos);
            input_sync(inputdev);
1851
          } else if (distanceZ > surthreshold) {  /* its a simulated left button click */

            if (leftBtnClicked != 1) {
              if (clickIntended == 1) {
                printk(KERN_INFO "myMouse_driver: transferFunction: called from click
                    intended.\n");
                yPosition = click_point.Y;
                zPosition = click_point.Z;
                clickIntended = 0;
              }
1861          input_report_key(inputdev, BTN_LEFT, 1);
              leftBtnClicked = 1;
              leftClickOnly = 1;
            } else {
              /* taking care of double click */
              if (rPos <= LEFT_DBLCLICK_ANG) {
                if (leftBtnDblClicked == 0) {
                  input_report_key(inputdev, BTN_LEFT, 0);
                  input_sync(inputdev);
                  input_report_key(inputdev, BTN_LEFT, 1);
1871              input_sync(inputdev);
                  input_report_key(inputdev, BTN_LEFT, 0);
                  input_sync(inputdev);
                  input_report_key(inputdev, BTN_LEFT, 1);
                  input_sync(inputdev);
                  input_report_key(inputdev, BTN_LEFT, 0);
                  input_sync(inputdev);
                  leftBtnDblClicked = 1;
                  leftBtnClicked = 0;
                }
1881            } else {
                if (leftBtnDblClicked == 1) {
                  leftBtnDblClicked = 0;
                }
              }

              if (leftClickOnly == 1) {
                Dvaly = myMouse->currentAvgOutput.Y - myMouse->lastAvgOutput.Y;
                Dvalz = myMouse->currentAvgOutput.Z - myMouse->lastAvgOutput.Z;
                if (Dvaly < 0) {
1891              Dvaly = (-1)*Dvaly;
                }
                if (Dvalz < 0) {
                  Dvalz = (-1)*Dvalz;
                }
                if ((Dvaly < UNCLICK_THRESHOLD) && (Dvalz < UNCLICK_THRESHOLD)) {
                  return 0; /* not updating the cursor position */
                } else {
                  leftClickOnly = 0;
                }
1901          }
            }

            transformX = transformPosition(yPosition - minYpos);
            transformY = transformPosition(zPosition - minZpos);  /* doing the swap: as z
                 value is the height */

            /* as the base is assumed to be always behind and under the rframe -- following
                 swap is necessary */
            /* this has to be done as screens 0,0 starts at the upper left corner and for
                 rframe (with the above assumtion) it is lower left corner */
```

```
        transformX = (x_src_resolution - transformX);   /* this is making x=0 to y=screen
              width visa versa and in between */
        transformY = (y_src_resolution - transformY);   /* this is making y=0 to y=screen
              height visa versa and in between */

        /* checking for pixel jitter -- check if the value is always positive */
        if (!((X_sensorPos == 0) && (Y_sensorPos == 0))) {    /* not (the first time) */
          diffX = transformX - X_sensorPos;
          diffY = transformY - Y_sensorPos;

          if (((diffX <= DIS_PIXEL_JITTER_POSI) || (diffX > DIS_PIXEL_JITTER_NEGI)) && ((
               diffY <= DIS_PIXEL_JITTER_POSI) || (diffY > DIS_PIXEL_JITTER_NEGI))) {   /*
                jitter detected */

            printk(KERN_INFO "myMouse_driver: transferFunction: transformed movement of
                 sensor is treated as jitter.\n");
            /* reporting the last movement */
            input_report_abs(inputdev, ABS_X, X_sensorPos);
            input_report_abs(inputdev, ABS_Y, Y_sensorPos);
            input_report_abs(inputdev, ABS_Z, Z_sensorPos);
          } else {
            X_sensorPos = transformX;
            Y_sensorPos = transformY;
            Z_sensorPos = distanceZ;
            input_report_abs(inputdev, ABS_X, transformX);
            input_report_abs(inputdev, ABS_Y, transformY);
            input_report_abs(inputdev, ABS_Z, distanceZ);
          }
        } else {
          X_sensorPos = transformX;
          Y_sensorPos = transformY;
          Z_sensorPos = distanceZ;
          input_report_abs(inputdev, ABS_X, transformX);
          input_report_abs(inputdev, ABS_Y, transformY);
          input_report_abs(inputdev, ABS_Z, distanceZ);
        }

        input_report_abs(inputdev, ABS_RX, rPos);
        input_report_abs(inputdev, ABS_RY, ePos);
        input_report_abs(inputdev, ABS_RZ, aPos);
        input_sync(inputdev);
      }

    } else {      /* this is one of the clutches (-) -- distanceZ > selthreshold -- same
         as left click */

      if (leftBtnClicked != 1) {
        if (clickIntended == 1) {
          printk(KERN_INFO "myMouse_driver: transferFunction: called from click intended
               .\n");
          yPosition = click_point.Y;
          zPosition = click_point.Z;
          clickIntended = 0;
        }
        input_report_key(inputdev, BTN_LEFT, 1);
        leftBtnClicked = 1;
        leftClickOnly = 1;
      } else {
        /* taking care of double click */
        if (rPos <= LEFT_DBLCLICK_ANG) {
          if (leftBtnDblClicked == 0) {
            input_report_key(inputdev, BTN_LEFT, 0);
            input_sync(inputdev);
            input_report_key(inputdev, BTN_LEFT, 1);
            input_sync(inputdev);
            input_report_key(inputdev, BTN_LEFT, 0);
            input_sync(inputdev);
            input_report_key(inputdev, BTN_LEFT, 1);
            input_sync(inputdev);
            input_report_key(inputdev, BTN_LEFT, 0);
            input_sync(inputdev);
            leftBtnDblClicked = 1;
            leftBtnClicked = 0;
          }
        } else {
          if (leftBtnDblClicked == 1) {
            leftBtnDblClicked = 0;
          }
        }

        if (leftClickOnly == 1) {
          Dvaly = myMouse->currentAvgOutput.Y - myMouse->lastAvgOutput.Y;
          Dvalz = myMouse->currentAvgOutput.Z - myMouse->lastAvgOutput.Z;
          if (Dvaly < 0) {
            Dvaly = (-1)*Dvaly;
          }
          if (Dvalz < 0) {
```

```
                Dvalz = (-1)*Dvalz;
              }
1991          if ((Dvaly < UNCLICK_THRESHOLD) && (Dvalz < UNCLICK_THRESHOLD)) {
                return 0; /* not updating the cursor position */
              } else {
                leftClickOnly = 0;
              }
            }
          }

          transformX = transformPosition(yPosition - minYpos);
          transformY = transformPosition(zPosition - minZpos);  /* doing the swap: as z value
              is the height */
2001
          /* as the base is assumed to be always behind and under the rframe -- following
              swap is necessary */
          /* this has to be done as screens 0,0 starts at the upper left corner and for
              rframe (with the above assumtion) it is lower left corner */
          transformX = (x_src_resolution - transformX);   /* this is making x=0 to y=screen
              width visa versa and in between */
          transformY = (y_src_resolution - transformY);   /* this is making y=0 to y=screen
              height visa versa and in between */

          /* checking for pixel jitter -- check if the value is always positive */
          if (!((X_sensorPos == 0) && (Y_sensorPos == 0))) {    /* not (the first time) */
            diffX = transformX - X_sensorPos;
            diffY = transformY - Y_sensorPos;
2011
            if (((diffX <= DIS_PIXEL_JITTER_POSI) || (diffX > DIS_PIXEL_JITTER_NEGI)) && ((
                diffY <= DIS_PIXEL_JITTER_POSI) || (diffY > DIS_PIXEL_JITTER_NEGI))) {   /*
                jitter detected */

              printk(KERN_INFO "myMouse_driver: transferFunction: transformed movement of
                  sensor is treated as jitter.\n");
              /* reporting the last movement */
              input_report_abs(inputdev, ABS_X, X_sensorPos);
              input_report_abs(inputdev, ABS_Y, Y_sensorPos);
              input_report_abs(inputdev, ABS_Z, Z_sensorPos);
            } else {
              X_sensorPos = transformX;
2021          Y_sensorPos = transformY;
              Z_sensorPos = distanceZ;
              input_report_abs(inputdev, ABS_X, transformX);
              input_report_abs(inputdev, ABS_Y, transformY);
              input_report_abs(inputdev, ABS_Z, distanceZ);
            }
          } else {
            X_sensorPos = transformX;
            Y_sensorPos = transformY;
            Z_sensorPos = distanceZ;
2031        input_report_abs(inputdev, ABS_X, transformX);
            input_report_abs(inputdev, ABS_Y, transformY);
            input_report_abs(inputdev, ABS_Z, distanceZ);
          }

          input_report_abs(inputdev, ABS_RX, rPos);
          input_report_abs(inputdev, ABS_RY, ePos);
          input_report_abs(inputdev, ABS_RZ, aPos);
          input_sync(inputdev);
        }
2041
      } else  {       /* then dotValue is greater, i.e., the point is above the surface */
        if (((-1)*distanceZ) <= POSI_CLUTCH_THRESHOLD)  {

          if (leftBtnClicked == 1) {  /* if left button was clicked then unclick it now */
            input_report_key(inputdev, BTN_LEFT, 0);
            leftBtnClicked = 0;
            leftClickOnly = 0;
          }
          else {
2051        Dvalx = myMouse->currentAvgOutput.X - myMouse->lastAvgOutput.X;
            if (Dvalx < 0) {
              Dvaly = myMouse->currentAvgOutput.Y - myMouse->lastAvgOutput.Y;
              Dvalz = myMouse->currentAvgOutput.Z - myMouse->lastAvgOutput.Z;
              if (Dvaly < 0) {
                Dvaly = (-1)*Dvaly;
              }
              if (Dvalz < 0) {
                Dvalz = (-1)*Dvalz;
              }
2061          if ((Dvaly < UNCLICK_THRESHOLD) && (Dvalz < UNCLICK_THRESHOLD)) {
                if (clickIntended != 1) {
                  click_point.Y = myMouse->lastAvgOutput.Y;
                  click_point.Z = myMouse->lastAvgOutput.Z;
                  clickIntended = 1;
                }
                myMouse->lastAvgOutput.X = xPosition;
```

```
                return 0; /* not updating the cursor position */
              } else {
                click_point.Y = 0;
2071            click_point.Z = 0;
                clickIntended = 0;
              }
            } else {
              click_point.Y = 0;
              click_point.Z = 0;
              clickIntended = 0;
            }
          }

2081      if (leftBtnDblClicked == 1) {
            leftBtnDblClicked = 0;
          }

          transformX = transformPosition(yPosition - minYpos);
          transformY = transformPosition(zPosition - minZpos);  /* doing the swap: as z value
                is the height */

          /* as the base is assumed to be always behind and under the rframe -- following
               swap is necessary */
          /* this has to be done as screens 0,0 starts at the upper left corner and for
               rframe (with the above assumtion) it is lower left corner */
          transformX = (x_src_resolution - transformX);   /* this is making x=0 to y=screen
               width visa versa and in between */
2091      transformY = (y_src_resolution - transformY);   /* this is making y=0 to y=screen
               height visa versa and in between */

          /* checking for pixel jitter -- check if the value is always positive */
          if (!((X_sensorPos == 0) && (Y_sensorPos == 0))) {    /* not (the first time) */
            diffX = transformX - X_sensorPos;
            diffY = transformY - Y_sensorPos;

            if (((diffX <= DIS_PIXEL_JITTER_POSI) || (diffX > DIS_PIXEL_JITTER_NEGI)) && ((
                 diffY <= DIS_PIXEL_JITTER_POSI) || (diffY > DIS_PIXEL_JITTER_NEGI))) {   /*
                 jitter detected */

              printk(KERN_INFO "myMouse_driver: transferFunction: transformed movement of
                   sensor is treated as jitter.\n");
2101          /* reporting the last movement */
              input_report_abs(inputdev, ABS_X, X_sensorPos);
              input_report_abs(inputdev, ABS_Y, Y_sensorPos);
              input_report_abs(inputdev, ABS_Z, Z_sensorPos);
            } else {
              X_sensorPos = transformX;
              Y_sensorPos = transformY;
              Z_sensorPos = distanceZ;
              input_report_abs(inputdev, ABS_X, transformX);
              input_report_abs(inputdev, ABS_Y, transformY);
2111          input_report_abs(inputdev, ABS_Z, distanceZ);
            }
          } else {
            X_sensorPos = transformX;
            Y_sensorPos = transformY;
            Z_sensorPos = distanceZ;
            input_report_abs(inputdev, ABS_X, transformX);
            input_report_abs(inputdev, ABS_Y, transformY);
            input_report_abs(inputdev, ABS_Z, distanceZ);
          }
2121
          input_report_abs(inputdev, ABS_RX, rPos);
          input_report_abs(inputdev, ABS_RY, ePos);
          input_report_abs(inputdev, ABS_RZ, aPos);
          input_sync(inputdev);

        } else {    /* this is one of the clutches (+) */
          /* reporting nothing to the input event subsystem */
          Z_sensorPos = distanceZ;
          return 0;
2131    }
      }

      myMouse->lastAvgOutput.X = xPosition;
      myMouse->lastAvgOutput.Y = yPosition;
      myMouse->lastAvgOutput.Z = zPosition;
      myMouse->lastAvgOutput.A = aPos;
      myMouse->lastAvgOutput.E = ePos;
      myMouse->lastAvgOutput.R = rPos;

2141  break;
    }
  }

  return 0;
}
```

```
static int boundary_check(myMouseType *myMouse)
{
  int valX, valY, valZ;
2151  int flag;

  valX = myMouse->filterOutput[k_index].X;
  valY = myMouse->filterOutput[k_index].Y;
  valZ = myMouse->filterOutput[k_index].Z;

  flag = 0;

  switch (rframeOriFlag) {
  case 0:     /* generic */
2161    /* update this later with proper code */
    break;
  case 1:     /* xy */
    if ((valX >= (minXpos - BOUNDARY_THRESHOLD)) && (valX <= (maxXpos + BOUNDARY_THRESHOLD))
        && (valY >= (minYpos - BOUNDARY_THRESHOLD)) && (valY <= (maxYpos + BOUNDARY_THRESHOLD
        )) && (valZ <= minZpos + POSI_CLUTCH_THRESHOLD + BOUNDARY_THRESHOLD)) {

      flag = 1;   /* inside rframe */
    } else {
      /* need to change the X or Y value to border value and reinitialize the sensor and
          filter arrays to this value */

      if (valX < (minXpos - BOUNDARY_THRESHOLD)) {
2171    valX = minXpos - BOUNDARY_THRESHOLD;
      } else if (valX > (maxXpos + BOUNDARY_THRESHOLD)) {
        valX = maxXpos + BOUNDARY_THRESHOLD;
      }

      if (valY < (minYpos - BOUNDARY_THRESHOLD)) {
        valY = minYpos - BOUNDARY_THRESHOLD;
      } else if (valY > (maxYpos + BOUNDARY_THRESHOLD)) {
        valY = maxYpos + BOUNDARY_THRESHOLD;
      }
2181
      if (valZ > minZpos + POSI_CLUTCH_THRESHOLD + BOUNDARY_THRESHOLD) {
        valZ = minZpos + POSI_CLUTCH_THRESHOLD + BOUNDARY_THRESHOLD;
      }

      flag = 0;   /* outside rframe */
    }
    break;
  case 2:     /* xz */
    if ((valX >= (minXpos - BOUNDARY_THRESHOLD)) && (valX <= (maxXpos + BOUNDARY_THRESHOLD))
        && (valZ >= (minZpos - BOUNDARY_THRESHOLD)) && (valZ <= (maxZpos + BOUNDARY_THRESHOLD
        )) && (valY <= minYpos + POSI_CLUTCH_THRESHOLD + BOUNDARY_THRESHOLD)) {
2191
      flag = 1;   /* inside rframe */
    } else {
      /* need to change the X or Z value to border value and reinitialize the sensor and
          filter arrays to this value */

      if (valX < (minXpos - BOUNDARY_THRESHOLD)) {
        valX = minXpos - BOUNDARY_THRESHOLD;
      } else if (valX > (maxXpos + BOUNDARY_THRESHOLD)) {
        valX = maxXpos + BOUNDARY_THRESHOLD;
      }
2201
      if (valZ < (minZpos - BOUNDARY_THRESHOLD)) {
        valZ = minZpos - BOUNDARY_THRESHOLD;
      } else if (valZ > (maxZpos + BOUNDARY_THRESHOLD)) {
        valZ = maxZpos  + BOUNDARY_THRESHOLD;
      }

      if (valY > minYpos + POSI_CLUTCH_THRESHOLD + BOUNDARY_THRESHOLD) {
        valY = minYpos + POSI_CLUTCH_THRESHOLD + BOUNDARY_THRESHOLD;
      }
2211
      flag = 0;   /* outside rframe */
    }
    break;
  case 3:     /* yz */
    if ((valY >= (minYpos - BOUNDARY_THRESHOLD)) && (valY <= (maxYpos + BOUNDARY_THRESHOLD))
        && (valZ >= (minZpos - BOUNDARY_THRESHOLD)) && (valZ <= (maxZpos + BOUNDARY_THRESHOLD
        )) && (valX <= minXpos + POSI_CLUTCH_THRESHOLD + BOUNDARY_THRESHOLD)) {

      flag = 1;   /* inside rframe */
    } else {
      /* need to change the Y or Z value to border value and reinitialize the sensor and
          filter arrays to this value */
2221
      if (valY < (minYpos - BOUNDARY_THRESHOLD)) {
        valY = minYpos - BOUNDARY_THRESHOLD;
      } else if (valY > (maxYpos + BOUNDARY_THRESHOLD)) {
```

```
            valY = maxYpos + BOUNDARY_THRESHOLD;
          }

          if (valZ < (minZpos - BOUNDARY_THRESHOLD)) {
            valZ = minZpos - BOUNDARY_THRESHOLD;
          } else if (valZ > (maxZpos + BOUNDARY_THRESHOLD)) {
2231        valZ = maxZpos  + BOUNDARY_THRESHOLD;
          }

          if (valX > minXpos + POSI_CLUTCH_THRESHOLD + BOUNDARY_THRESHOLD) {
            valX = minXpos + POSI_CLUTCH_THRESHOLD + BOUNDARY_THRESHOLD;
          }

          flag = 0;    /* outside rframe */
        }
        break;
2241    }

      myMouse->filterOutput[k_index].X = valX;
      myMouse->filterOutput[k_index].Y = valY;
      myMouse->filterOutput[k_index].Z = valZ;

      return flag;
    }

    static int submit_input_events(myMouseType *myMouse, struct input_dev *inputdev, char *data)
2251 {
      int V1X, V1Y, V1Z, i;
      int V2X, V2Y, V2Z;
      /* __u16 curX, curY, curZ; */
      __s16 curX, curY, curZ;
      __s16 curA, curE, curR, retVal;
      __u16 diffX, diffY, diffZ, diffA, diffE, diffR;
      typeMovement sum_avg;

      if (updateFlag == 1) {       /* if there has been a change in the rframe setup */
2261    input_set_abs_params(inputdev, ABS_X, 0, x_src_resolution, 0, 0);
        input_set_abs_params(inputdev, ABS_Y, 0, y_src_resolution, 0, 0);
        input_set_abs_params(inputdev, ABS_Z, (-1)*z_resolution, z_resolution, 0, 0);   /* the z-
            value will be reported as it is */

        /* re-initiating the averaging sequence */
        avg_track_index = -1;
        filterOutNo = -1;

        /* the following portion could be done on the application side */
        /* getting the normal of the plane plus the k value */
2271    /* normal (vector) = V1 x V2 : V1 = (p1 - p2) and V2 = (p3 - p2) assuming all the points
            are given clockwise */
        V1X = pDefPoint1X - pDefPoint2X;
        V1Y = pDefPoint1Y - pDefPoint2Y;
        V1Z = pDefPoint1Z - pDefPoint2Z;

        V2X = pDefPoint3X - pDefPoint2X;
        V2Y = pDefPoint3Y - pDefPoint2Y;
        V2Z = pDefPoint3Z - pDefPoint2Z;

        /* the way the points are */
2281    normalVectorX = (V1Y*V2Z) - (V1Z*V2Y);
        normalVectorY = (V1Z*V2X) - (V1X*V2Z);
        normalVectorZ = (V1X*V2Y) - (V1Y*V2X);

        /* k = normal * p1 */
        kValue = (normalVectorX*pDefPoint1X) + (normalVectorY*pDefPoint1Y) + (normalVectorZ*
            pDefPoint1Z);
        normalVDot = (normalVectorX*normalVectorX) + (normalVectorY*normalVectorY) + (
            normalVectorZ*normalVectorZ);

        updateFlag = 0;
      }
2291
      if (data[0] == 68) {
        /* printk(KERN_INFO "myMouse_driver: submit_input_events: received data from sensor
            number = %d.\n", data[1]); */
        curX = data[3];
        curX = curX * 256 + data[2];
        curY = data[5];
        curY = curY * 256 + data[4];
        curZ = data[7];
        curZ = curZ * 256 + data[6];
        curA = data[9];
2301    curA = curA * 256 + data[8];
        curE = data[11];
        curE = curE * 256 + data[10];
        curR = data[13];
        curR = curR * 256 + data[12];
```

```
        /* make the curX, curY, curZ values positive if they are negetive */
        if (curX < 0) {
          curX = (-1)*curX;
        }
2311    if (curY < 0) {
          curY = (-1)*curY;
        }
        if (curZ < 0) {
          curZ = (-1)*curZ;
        }
      } else {
        printk(KERN_INFO "myMouse_driver: submit_input_events: received data with data[0] = %d.\n
            ", data[0]);
        return -1;
      }
2321
      if ((curX < MIN_SENSOR_DIS_VAL) && (curY < MIN_SENSOR_DIS_VAL) && (curZ <
          MIN_SENSOR_DIS_VAL)) {
        printk(KERN_INFO "myMouse_driver: submit_input_events: recorded movement of sensor is
            less than min acceptable range.\n");
        return 0;  /* not reporting */
      } else if ((curX > MAX_SENSOR_DIS_VAL) || (curY > MAX_SENSOR_DIS_VAL) || (curZ >
          MAX_SENSOR_DIS_VAL)) {
        printk(KERN_INFO "myMouse_driver: submit_input_events: recorded movement of sensor is
            more than max acceptable range.\n");
        return 0;  /* not reporting */
      }

      if (k_index == -1) {
2331    for (i=0; i<3; i++) {
          myMouse->filterOutput[i].X = curX;
          myMouse->filterOutput[i].Y = curY;
          myMouse->filterOutput[i].Z = curZ;
          myMouse->filterOutput[i].A = curA;
          myMouse->filterOutput[i].E = curE;
          myMouse->filterOutput[i].R = curR;

          myMouse->sensorIntput[i].X = curX;
          myMouse->sensorIntput[i].Y = curY;
2341      myMouse->sensorIntput[i].Z = curZ;
          myMouse->sensorIntput[i].A = curA;
          myMouse->sensorIntput[i].E = curE;
          myMouse->sensorIntput[i].R = curR;
        }
      }

      /* possible values -1/0/1/2 -- if -1 then first time; if 0 then k-1=2 and k-2=1; if 1 then
          k-1=0 and k-2=2; if 2 then k-1=1 and k-2=0 */
      if (k_index == -1) {       /* if for the first time - k-1 and k-2 values will be zero */
        last_k_index = -1;
2351    seclast_k_index = -1;
        k_index = 0;
        myMouse->sensorIntput[k_index].X = curX;
        myMouse->sensorIntput[k_index].Y = curY;
        myMouse->sensorIntput[k_index].Z = curZ;
        myMouse->sensorIntput[k_index].A = curA;
        myMouse->sensorIntput[k_index].E = curE;
        myMouse->sensorIntput[k_index].R = curR;

        myMouse->filterOutput[k_index].X = (80*myMouse->sensorIntput[k_index].X + 161*myMouse->
            sensorIntput[2].X + 80*myMouse->sensorIntput[1].X + 17352*myMouse->filterOutput[2].X
            - 7666*myMouse->filterOutput[1].X)/10000;
2361    myMouse->filterOutput[k_index].Y = (80*myMouse->sensorIntput[k_index].Y + 161*myMouse->
            sensorIntput[2].Y + 80*myMouse->sensorIntput[1].Y + 17352*myMouse->filterOutput[2].Y
            - 7666*myMouse->filterOutput[1].Y)/10000;
        myMouse->filterOutput[k_index].Z = (80*myMouse->sensorIntput[k_index].Z + 161*myMouse->
            sensorIntput[2].Z + 80*myMouse->sensorIntput[1].Z + 17352*myMouse->filterOutput[2].Z
            - 7666*myMouse->filterOutput[1].Z)/10000;
        myMouse->filterOutput[k_index].A = (80*myMouse->sensorIntput[k_index].A + 161*myMouse->
            sensorIntput[2].A + 80*myMouse->sensorIntput[1].A + 17352*myMouse->filterOutput[2].A
            - 7666*myMouse->filterOutput[1].A)/10000;
        myMouse->filterOutput[k_index].E = (80*myMouse->sensorIntput[k_index].E + 161*myMouse->
            sensorIntput[2].E + 80*myMouse->sensorIntput[1].E + 17352*myMouse->filterOutput[2].E
            - 7666*myMouse->filterOutput[1].E)/10000;
        myMouse->filterOutput[k_index].R = (80*myMouse->sensorIntput[k_index].R + 161*myMouse->
            sensorIntput[2].R + 80*myMouse->sensorIntput[1].R + 17352*myMouse->filterOutput[2].R
            - 7666*myMouse->filterOutput[1].R)/10000;

      } else if (k_index == 0) {
        if (last_k_index != -1) {
          seclast_k_index = last_k_index;
        }
2371    last_k_index = 0;
        k_index = 1;
        myMouse->sensorIntput[k_index].X = curX;
        myMouse->sensorIntput[k_index].Y = curY;
        myMouse->sensorIntput[k_index].Z = curZ;
```

```
        myMouse->sensorIntput[k_index].A = curA;
        myMouse->sensorIntput[k_index].E = curE;
        myMouse->sensorIntput[k_index].R = curR;

        myMouse->filterOutput[k_index].X = (80*myMouse->sensorIntput[k_index].X + 161*myMouse->
            sensorIntput[0].X + 80*myMouse->sensorIntput[2].X + 17352*myMouse->filterOutput[0].X
            - 7666*myMouse->filterOutput[2].X)/10000;
2381    myMouse->filterOutput[k_index].Y = (80*myMouse->sensorIntput[k_index].Y + 161*myMouse->
            sensorIntput[0].Y + 80*myMouse->sensorIntput[2].Y + 17352*myMouse->filterOutput[0].Y
            - 7666*myMouse->filterOutput[2].Y)/10000;
        myMouse->filterOutput[k_index].Z = (80*myMouse->sensorIntput[k_index].Z + 161*myMouse->
            sensorIntput[0].Z + 80*myMouse->sensorIntput[2].Z + 17352*myMouse->filterOutput[0].Z
            - 7666*myMouse->filterOutput[2].Z)/10000;
        myMouse->filterOutput[k_index].A = (80*myMouse->sensorIntput[k_index].A + 161*myMouse->
            sensorIntput[0].A + 80*myMouse->sensorIntput[2].A + 17352*myMouse->filterOutput[0].A
            - 7666*myMouse->filterOutput[2].A)/10000;
        myMouse->filterOutput[k_index].E = (80*myMouse->sensorIntput[k_index].E + 161*myMouse->
            sensorIntput[0].E + 80*myMouse->sensorIntput[2].E + 17352*myMouse->filterOutput[0].E
            - 7666*myMouse->filterOutput[2].E)/10000;
        myMouse->filterOutput[k_index].R = (80*myMouse->sensorIntput[k_index].R + 161*myMouse->
            sensorIntput[0].R + 80*myMouse->sensorIntput[2].R + 17352*myMouse->filterOutput[0].R
            - 7666*myMouse->filterOutput[2].R)/10000;

    } else if (k_index == 1) {
        seclast_k_index = 0;
        last_k_index = 1;
        k_index = 2;
2391    myMouse->sensorIntput[k_index].X = curX;
        myMouse->sensorIntput[k_index].Y = curY;
        myMouse->sensorIntput[k_index].Z = curZ;
        myMouse->sensorIntput[k_index].A = curA;
        myMouse->sensorIntput[k_index].E = curE;
        myMouse->sensorIntput[k_index].R = curR;

        myMouse->filterOutput[k_index].X = (80*myMouse->sensorIntput[k_index].X + 161*myMouse->
            sensorIntput[1].X + 80*myMouse->sensorIntput[0].X + 17352*myMouse->filterOutput[1].X
            - 7666*myMouse->filterOutput[0].X)/10000;
        myMouse->filterOutput[k_index].Y = (80*myMouse->sensorIntput[k_index].Y + 161*myMouse->
            sensorIntput[1].Y + 80*myMouse->sensorIntput[0].Y + 17352*myMouse->filterOutput[1].Y
            - 7666*myMouse->filterOutput[0].Y)/10000;
        myMouse->filterOutput[k_index].Z = (80*myMouse->sensorIntput[k_index].Z + 161*myMouse->
            sensorIntput[1].Z + 80*myMouse->sensorIntput[0].Z + 17352*myMouse->filterOutput[1].Z
            - 7666*myMouse->filterOutput[0].Z)/10000;
2401    myMouse->filterOutput[k_index].A = (80*myMouse->sensorIntput[k_index].A + 161*myMouse->
            sensorIntput[1].A + 80*myMouse->sensorIntput[0].A + 17352*myMouse->filterOutput[1].A
            - 7666*myMouse->filterOutput[0].A)/10000;
        myMouse->filterOutput[k_index].E = (80*myMouse->sensorIntput[k_index].E + 161*myMouse->
            sensorIntput[1].E + 80*myMouse->sensorIntput[0].E + 17352*myMouse->filterOutput[1].E
            - 7666*myMouse->filterOutput[0].E)/10000;
        myMouse->filterOutput[k_index].R = (80*myMouse->sensorIntput[k_index].R + 161*myMouse->
            sensorIntput[1].R + 80*myMouse->sensorIntput[0].R + 17352*myMouse->filterOutput[1].R
            - 7666*myMouse->filterOutput[0].R)/10000;

    } else if (k_index == 2) {
        seclast_k_index = 1;
        last_k_index = 2;
        k_index = 0;
        myMouse->sensorIntput[k_index].X = curX;
        myMouse->sensorIntput[k_index].Y = curY;
2411    myMouse->sensorIntput[k_index].Z = curZ;
        myMouse->sensorIntput[k_index].A = curA;
        myMouse->sensorIntput[k_index].E = curE;
        myMouse->sensorIntput[k_index].R = curR;

        myMouse->filterOutput[k_index].X = (80*myMouse->sensorIntput[k_index].X + 161*myMouse->
            sensorIntput[2].X + 80*myMouse->sensorIntput[1].X + 17352*myMouse->filterOutput[2].X
            - 7666*myMouse->filterOutput[1].X)/10000;
        myMouse->filterOutput[k_index].Y = (80*myMouse->sensorIntput[k_index].Y + 161*myMouse->
            sensorIntput[2].Y + 80*myMouse->sensorIntput[1].Y + 17352*myMouse->filterOutput[2].Y
            - 7666*myMouse->filterOutput[1].Y)/10000;
        myMouse->filterOutput[k_index].Z = (80*myMouse->sensorIntput[k_index].Z + 161*myMouse->
            sensorIntput[2].Z + 80*myMouse->sensorIntput[1].Z + 17352*myMouse->filterOutput[2].Z
            - 7666*myMouse->filterOutput[1].Z)/10000;
        myMouse->filterOutput[k_index].A = (80*myMouse->sensorIntput[k_index].A + 161*myMouse->
            sensorIntput[2].A + 80*myMouse->sensorIntput[1].A + 17352*myMouse->filterOutput[2].A
            - 7666*myMouse->filterOutput[1].A)/10000;
        myMouse->filterOutput[k_index].E = (80*myMouse->sensorIntput[k_index].E + 161*myMouse->
            sensorIntput[2].E + 80*myMouse->sensorIntput[1].E + 17352*myMouse->filterOutput[2].E
            - 7666*myMouse->filterOutput[1].E)/10000;
2421    myMouse->filterOutput[k_index].R = (80*myMouse->sensorIntput[k_index].R + 161*myMouse->
            sensorIntput[2].R + 80*myMouse->sensorIntput[1].R + 17352*myMouse->filterOutput[2].R
            - 7666*myMouse->filterOutput[1].R)/10000;
    }

    if (last_k_index != -1) {
      if ((myMouse->filterOutput[k_index].X == myMouse->filterOutput[last_k_index].X) && (
          myMouse->filterOutput[k_index].Y == myMouse->filterOutput[last_k_index].Y) && (
```

```
            myMouse->filterOutput[k_index].Z == myMouse->filterOutput[last_k_index].Z) && (
            myMouse->filterOutput[k_index].A == myMouse->filterOutput[last_k_index].A) && (
            myMouse->filterOutput[k_index].E == myMouse->filterOutput[last_k_index].E) && (
            myMouse->filterOutput[k_index].R == myMouse->filterOutput[last_k_index].R))   /*
            checking if the sensor has moved or not */
        {
          printk(KERN_INFO "myMouse_driver: submit_input_events: no movement of sensor.\n");
        } else {
          diffX = myMouse->filterOutput[k_index].X - myMouse->filterOutput[last_k_index].X;
          diffY = myMouse->filterOutput[k_index].Y - myMouse->filterOutput[last_k_index].Y;
2431      diffZ = myMouse->filterOutput[k_index].Z - myMouse->filterOutput[last_k_index].Z;
          diffA = myMouse->filterOutput[k_index].A - myMouse->filterOutput[last_k_index].A;
          diffE = myMouse->filterOutput[k_index].E - myMouse->filterOutput[last_k_index].E;
          diffR = myMouse->filterOutput[k_index].R - myMouse->filterOutput[last_k_index].R;

          if (((diffX <= DIS_JITTER_VALUE_POSI) || (diffX > DIS_JITTER_VALUE_NEGI)) && ((diffY <=
               DIS_JITTER_VALUE_POSI) || (diffY > DIS_JITTER_VALUE_NEGI)) && ((diffZ <=
              DIS_JITTER_VALUE_POSI) || (diffZ > DIS_JITTER_VALUE_NEGI)) && ((diffA <=
              ANG_JITTER_VALUE_POSI) || (diffA > ANG_JITTER_VALUE_NEGI)) && ((diffE <=
              ANG_JITTER_VALUE_POSI) || (diffE > ANG_JITTER_VALUE_NEGI)) && ((diffR <=
              ANG_JITTER_VALUE_POSI) || (diffR > ANG_JITTER_VALUE_NEGI))) {   /* jitter detected
              */

            printk(KERN_INFO "myMouse_driver: submit_input_events: recorded movement of sensor is
                 treated as jitter.\n");
            return 0;  /* not reporting */
          }
2441    }
      }

      if (useTransferFlag == 1) {

        retVal = boundary_check(myMouse);

        if (avg_track_index == (ARRAY_SIZE_AVG-1)) {
          avg_track_index = 0;
        } else {
2451      avg_track_index++;
        }
        myMouse->filterOutAgv[avg_track_index].X = myMouse->filterOutput[k_index].X;
        myMouse->filterOutAgv[avg_track_index].Y = myMouse->filterOutput[k_index].Y;
        myMouse->filterOutAgv[avg_track_index].Z = myMouse->filterOutput[k_index].Z;
        myMouse->filterOutAgv[avg_track_index].A = myMouse->filterOutput[k_index].A;
        myMouse->filterOutAgv[avg_track_index].E = myMouse->filterOutput[k_index].E;
        myMouse->filterOutAgv[avg_track_index].R = myMouse->filterOutput[k_index].R;

        if (filterOutNo == -1) {
2461      filterOutNo++;

          myMouse->currentAvgOutput.X = myMouse->filterOutput[k_index].X;
          myMouse->currentAvgOutput.Y = myMouse->filterOutput[k_index].Y;
          myMouse->currentAvgOutput.Z = myMouse->filterOutput[k_index].Z;
          myMouse->currentAvgOutput.A = myMouse->filterOutput[k_index].A;
          myMouse->currentAvgOutput.E = myMouse->filterOutput[k_index].E;
          myMouse->currentAvgOutput.R = myMouse->filterOutput[k_index].R;
        } else {
          sum_avg.X = 0;
2471      sum_avg.Y = 0;
          sum_avg.Z = 0;
          sum_avg.A = 0;
          sum_avg.E = 0;
          sum_avg.R = 0;

          /* if (filterOutNo < (ARRAY_SIZE_AVG-1)) { */
          if (filterOutNo < (avgSampleNo - 1)) {
            filterOutNo++;
            for (i=0; i<(filterOutNo+1); i++) {
2481          sum_avg.X = sum_avg.X + myMouse->filterOutAgv[i].X;
              sum_avg.Y = sum_avg.Y + myMouse->filterOutAgv[i].Y;
              sum_avg.Z = sum_avg.Z + myMouse->filterOutAgv[i].Z;
              sum_avg.A = sum_avg.A + myMouse->filterOutAgv[i].A;
              sum_avg.E = sum_avg.E + myMouse->filterOutAgv[i].E;
              sum_avg.R = sum_avg.R + myMouse->filterOutAgv[i].R;
            }

            sum_avg.X = (myMouse->lastAvgOutput.X + sum_avg.X)/(filterOutNo+2);
            sum_avg.Y = (myMouse->lastAvgOutput.Y + sum_avg.Y)/(filterOutNo+2);
2491        sum_avg.Z = (myMouse->lastAvgOutput.Z + sum_avg.Z)/(filterOutNo+2);
            sum_avg.A = (myMouse->lastAvgOutput.A + sum_avg.A)/(filterOutNo+2);
            sum_avg.E = (myMouse->lastAvgOutput.E + sum_avg.E)/(filterOutNo+2);
            sum_avg.R = (myMouse->lastAvgOutput.R + sum_avg.R)/(filterOutNo+2);
          } else {
            if ((avgSampleNo - 1) == avg_track_index) {
              for (i=0; i<avgSampleNo; i++) {
                sum_avg.X = sum_avg.X + myMouse->filterOutAgv[i].X;
                sum_avg.Y = sum_avg.Y + myMouse->filterOutAgv[i].Y;
                sum_avg.Z = sum_avg.Z + myMouse->filterOutAgv[i].Z;
```

```
2501              sum_avg.A = sum_avg.A + myMouse->filterOutAgv[i].A;
                  sum_avg.E = sum_avg.E + myMouse->filterOutAgv[i].E;
                  sum_avg.R = sum_avg.R + myMouse->filterOutAgv[i].R;
                }
              } else if ((avgSampleNo - 1) < avg_track_index) {
                for (i=(avg_track_index - (avgSampleNo - 1)); i<(avg_track_index+1); i++) {
                  sum_avg.X = sum_avg.X + myMouse->filterOutAgv[i].X;
                  sum_avg.Y = sum_avg.Y + myMouse->filterOutAgv[i].Y;
                  sum_avg.Z = sum_avg.Z + myMouse->filterOutAgv[i].Z;
                  sum_avg.A = sum_avg.A + myMouse->filterOutAgv[i].A;
2511              sum_avg.E = sum_avg.E + myMouse->filterOutAgv[i].E;
                  sum_avg.R = sum_avg.R + myMouse->filterOutAgv[i].R;
                }
              } else if ((avgSampleNo - 1) > avg_track_index) {
                for (i=0; i<(avg_track_index+1); i++) {
                  sum_avg.X = sum_avg.X + myMouse->filterOutAgv[i].X;
                  sum_avg.Y = sum_avg.Y + myMouse->filterOutAgv[i].Y;
                  sum_avg.Z = sum_avg.Z + myMouse->filterOutAgv[i].Z;
                  sum_avg.A = sum_avg.A + myMouse->filterOutAgv[i].A;
                  sum_avg.E = sum_avg.E + myMouse->filterOutAgv[i].E;
2521              sum_avg.R = sum_avg.R + myMouse->filterOutAgv[i].R;
                }

                for (i=(ARRAY_SIZE_AVG - (avgSampleNo - (avg_track_index+1))); i<ARRAY_SIZE_AVG; i
                    ++) {
                  sum_avg.X = sum_avg.X + myMouse->filterOutAgv[i].X;
                  sum_avg.Y = sum_avg.Y + myMouse->filterOutAgv[i].Y;
                  sum_avg.Z = sum_avg.Z + myMouse->filterOutAgv[i].Z;
                  sum_avg.A = sum_avg.A + myMouse->filterOutAgv[i].A;
                  sum_avg.E = sum_avg.E + myMouse->filterOutAgv[i].E;
                  sum_avg.R = sum_avg.R + myMouse->filterOutAgv[i].R;
2531            }
              }

              sum_avg.X = (myMouse->lastAvgOutput.X + sum_avg.X)/(avgSampleNo+1);
              sum_avg.Y = (myMouse->lastAvgOutput.Y + sum_avg.Y)/(avgSampleNo+1);
              sum_avg.Z = (myMouse->lastAvgOutput.Z + sum_avg.Z)/(avgSampleNo+1);
              sum_avg.A = (myMouse->lastAvgOutput.A + sum_avg.A)/(avgSampleNo+1);
              sum_avg.E = (myMouse->lastAvgOutput.E + sum_avg.E)/(avgSampleNo+1);
              sum_avg.R = (myMouse->lastAvgOutput.R + sum_avg.R)/(avgSampleNo+1);
            }
2541
            myMouse->currentAvgOutput.X = sum_avg.X;
            myMouse->currentAvgOutput.Y = sum_avg.Y;
            myMouse->currentAvgOutput.Z = sum_avg.Z;
            myMouse->currentAvgOutput.A = sum_avg.A;
            myMouse->currentAvgOutput.E = sum_avg.E;
            myMouse->currentAvgOutput.R = sum_avg.R;
          }

          retVal = transferFunction(myMouse, inputdev, myMouse->currentAvgOutput.X, myMouse->
              currentAvgOutput.Y, myMouse->currentAvgOutput.Z, myMouse->currentAvgOutput.R, myMouse
              ->currentAvgOutput.E, myMouse->currentAvgOutput.A);
2551      } else {
          retVal = transferFunction(myMouse, inputdev, myMouse->filterOutput[k_index].X, myMouse->
              filterOutput[k_index].Y, myMouse->filterOutput[k_index].Z, myMouse->filterOutput[
              k_index].R, myMouse->filterOutput[k_index].E, myMouse->filterOutput[k_index].A);
        }

        return 0;
      }

      static void myMouse_readData(struct input_polled_dev *polled_dev)
      {
        int retVal, bytes_io, val;
2561    unsigned char *send_buf_com;       /* data buffer to be send to the device */
        myMouseType *mouse = polled_dev->private;
        struct input_dev *inputdev = polled_dev->input;

        mutex_lock(&mouse->bulk_io_mutex);
        if (!mouse->interface) {    /* disconnect was called */
          retVal = -ENODEV;
          goto exit;
        }

2571    if (updateFlag == 1) {
          send_buf_com = kmalloc(mouse->dataLength, GFP_KERNEL);
          switch (rframeOriFlag) {
          case 0:   /* generic */
            send_buf_com[0] = 'S';
            send_buf_com[1] = 'T';
            send_buf_com[2] = '0';
            send_buf_com[3] = ',';
            send_buf_com[4] = '0';
            send_buf_com[5] = ',';
2581        send_buf_com[6] = '0';
            send_buf_com[7] = ',';
```

```
            send_buf_com[8] = '0';
            send_buf_com[9] = '\0';
            break;
          case 1:   /* XY */
            send_buf_com[0] = 'S';
            send_buf_com[1] = 'T';
            send_buf_com[2] = '0';
            send_buf_com[3] = ',';
2591        send_buf_com[4] = '90';
            send_buf_com[5] = ',';
            send_buf_com[6] = '0';
            send_buf_com[7] = ',';
            send_buf_com[8] = '0';
            send_buf_com[9] = '\0';
            break;
          case 2:   /* XZ */
            send_buf_com[0] = 'S';
            send_buf_com[1] = 'T';
2601        send_buf_com[2] = '0';
            send_buf_com[3] = ',';
            send_buf_com[4] = '90';
            send_buf_com[5] = ',';
            send_buf_com[6] = '90';
            send_buf_com[7] = ',';
            send_buf_com[8] = '0';
            send_buf_com[9] = '\0';
            break;
          case 3:   /* YZ */
2611        send_buf_com[0] = 'S';
            send_buf_com[1] = 'T';
            send_buf_com[2] = '0';
            send_buf_com[3] = ',';
            send_buf_com[4] = '0';
            send_buf_com[5] = ',';
            send_buf_com[6] = '90';
            send_buf_com[7] = ',';
            send_buf_com[8] = '0';
            send_buf_com[9] = '\0';
2621        break;
          default:  /* generic */
            send_buf_com[0] = 'S';
            send_buf_com[1] = 'T';
            send_buf_com[2] = '0';
            send_buf_com[3] = ',';
            send_buf_com[4] = '0';
            send_buf_com[5] = ',';
            send_buf_com[6] = '0';
            send_buf_com[7] = ',';
2631        send_buf_com[8] = '0';
            send_buf_com[9] = '\0';
            break;
          }

          retVal = usb_bulk_msg(mouse->usbdev, mouse->out_pipe, send_buf_com, mouse->dataLength, &
               bytes_io, 10000);
          if (retVal != 0) {
            free_myMouseData(mouse);
            kfree(send_buf_com);
            mutex_unlock(&mouse->bulk_io_mutex);
2641        printk(KERN_INFO "myMouse_driver: myMouse_readData: failed to send command for making
                 sensor0 A=90 being the zero to wintracker.\n");
            goto exit;
          }
          kfree(send_buf_com);
        }

        retVal = usb_bulk_msg(mouse->usbdev, mouse->out_pipe, mouse->send_buffer, mouse->dataLength
             , &bytes_io, 500); /* timeout time is 500 msec */
        if (retVal != 0) {
          mutex_unlock(&mouse->bulk_io_mutex);
          printk(KERN_INFO "myMouse_driver: usb_myMouse_probe: failed to send command for single
               output to wintracker with retVal: %d.\n", retVal);
2651    }

        retVal = usb_bulk_msg(mouse->usbdev, mouse->in_pipe, mouse->data, mouse->dataLength, &
             bytes_io, 500); /* timeout time is 500 msec */
        if (!retVal) {
          val = submit_input_events(mouse, inputdev, mouse->data);
        }
      exit:
        mutex_unlock(&mouse->bulk_io_mutex);
        if (retVal)
          printk(KERN_INFO "myMouse_driver: myMouse_readData: myMouse_readData failed with result %
               d.\n", retVal);
2661  }

      static void init_device_data(void)
```

```
    {
      useTransferFlag = 0;       /* not to use the transfer function */
      updateFlag = 0;        /* no update */
      minXpos = 0;
      maxXpos = MAX_SCREEN_X;
      minYpos = 0;
      maxYpos = MAX_SCREEN_Y;
2671  minZpos = 0;
      maxZpos = MAX_SCREEN_Y;
      surthreshold = 1;
      selthreshold = 10;
      srcMulFactor = 1;
      rframeMulFactor = 1;
      srcFactor = 1;
      inputFactor = 1;
      x_src_resolution = MAX_SCREEN_X;
      y_src_resolution = MAX_SCREEN_Y;
2681  z_resolution = MAX_SENSOR_DIS_VAL;
      leftBtnClicked = 0;      /* set to not clicked */
      rightBtnClicked = 0;       /* set to not clicked */
      leftBtnDblClicked = 0;       /* set to not clicked */
      X_sensorPos = 0;
      Y_sensorPos = 0;
      Z_sensorPos = 0;
      k_index = -1;
      last_k_index = -1;
      seclast_k_index = -1;
2691  clickIntended = 0;
      leftClickOnly = 0;
      avg_track_index = -1;
      filterOutNo = -1;
      avgSampleNo = 25;
    }

    static void init_myMouse_input(struct input_polled_dev *inputPollDev, struct usb_device *udev
        , myMouseType *myMouse)
    {
      struct input_dev *inputdev;
2701
      inputdev = inputPollDev->input;

      /* setting the types of events the device will support*/
      set_bit(EV_KEY, inputdev->evbit);
      set_bit(EV_ABS, inputdev->evbit);      /* X, Y, Z, roll, pitch and yaw as abs */

      /* following are the keys/buttons the device will support */
      set_bit(BTN_LEFT, inputdev->keybit);
      set_bit(BTN_RIGHT, inputdev->keybit);
2711
      /* setting the absolute events */
      input_set_abs_params(inputdev, ABS_X, 0, x_src_resolution, 0, 0);
      input_set_abs_params(inputdev, ABS_Y, 0, y_src_resolution, 0, 0);
      input_set_abs_params(inputdev, ABS_Z, (-1)*z_resolution, z_resolution, 0, 0);
      input_set_abs_params(inputdev, ABS_RX, MIN_SENSOR_ANG_VAL, MAX_SENSOR_ANG_VAL, 0, 0);    /*
          need to set this value */
      input_set_abs_params(inputdev, ABS_RY, MIN_SENSOR_ANG_VAL, MAX_SENSOR_ANG_VAL, 0, 0);    /*
          need to set this value */
      input_set_abs_params(inputdev, ABS_RZ, MIN_SENSOR_ANG_VAL, MAX_SENSOR_ANG_VAL, 0, 0);    /*
          need to set this value */

      /* setting device related information */
2721  usb_make_path(udev, myMouse->phys, sizeof(myMouse->phys));
      strlcat(myMouse->phys, "/input0", sizeof(myMouse->phys));        /* making the input device
          path */

      inputdev->name = myMouse->name;    /* it could be hard coded */
      inputdev->phys = myMouse->phys;
      inputdev->id.bustype = BUS_USB;
      inputdev->id.vendor = udev->descriptor.idVendor;
      inputdev->id.product = udev->descriptor.idProduct;
      inputdev->id.version = udev->descriptor.bcdDevice;

2731  inputPollDev->poll_interval = myMouse->pollInterval;  /* how offen the device to be polled
          */
    }

    static int usb_myMouse_probe(struct usb_interface *intf, const struct usb_device_id *dev_id)
    {
      struct usb_host_interface *iface_desc;
      struct usb_endpoint_descriptor *in_endpoint, *out_endpoint;
      myMouseType *myMouse;
      int errNo;
      char *tempBuffer;
2741  unsigned char *send_buf_com;       /* data buffer to be send to the device */
      int retVal, bytes_written;

      struct usb_device *usbdev = interface_to_usbdev(intf);
```

```
    iface_desc = intf->cur_altsetting;    /* getting the currently active interface setting */

    if (iface_desc->desc.bNumEndpoints < 2) { /* there should be atleast two endpoints one IN
        and other is OUT */
      printk(KERN_INFO "myMouse_driver: usb_myMouse_probe: endpoint number is less than 2.\n");
      return -ENODEV;
2751  }

    /* getting the device endpoints */
    in_endpoint = &iface_desc->endpoint[0].desc;
    out_endpoint = &iface_desc->endpoint[1].desc;

    /* checking the status of the endpoint */
    if (((in_endpoint->bmAttributes & USB_ENDPOINT_XFERTYPE_MASK) != USB_ENDPOINT_XFER_BULK) &&
        ((out_endpoint->bmAttributes & USB_ENDPOINT_XFERTYPE_MASK) != USB_ENDPOINT_XFER_BULK))
        {
      printk(KERN_INFO "myMouse_driver: usb_myMouse_probe: the endpoints are not of bulk type.\
          n");
      return -ENODEV;
2761  }
    if (((in_endpoint->bEndpointAddress & USB_ENDPOINT_DIR_MASK) != USB_DIR_IN) || ((
        out_endpoint->bEndpointAddress & USB_ENDPOINT_DIR_MASK) != USB_DIR_OUT)) {
      printk(KERN_INFO "myMouse_driver: usb_myMouse_probe: couldn't find proper endpoints.\n");
      return -ENODEV;
    }

    /* creating myMouseType device */
    if (!(myMouse = create_myMouse(usbdev, intf))) {
      printk(KERN_INFO "myMouse_driver: usb_myMouse_probe: failed to create myMouse type device
          .\n");
      return -ENOMEM;
2771  }

    /* setting device name */
    if (!(tempBuffer = kmalloc(63, GFP_KERNEL))) {
      free_myMouseData(myMouse);
      printk(KERN_INFO "myMouse_driver: usb_myMouse_probe: failed to allocate tempBuffer.\n");
      return -ENOMEM;
    }
    if (usbdev->descriptor.iManufacturer && usb_string(usbdev, usbdev->descriptor.iManufacturer
        , tempBuffer, 63) > 0)
      strcat(myMouse->name, tempBuffer);
2781  if (usbdev->descriptor.iProduct && usb_string(usbdev, usbdev->descriptor.iProduct,
        tempBuffer, 63) > 0)
      sprintf(myMouse->name, "%s %s", myMouse->name, tempBuffer);
    if (!strlen(myMouse->name))
      sprintf(myMouse->name, "Customized USB Pointer Device %04x:%04x", le16_to_cpu(usbdev->
          descriptor.idVendor), le16_to_cpu(usbdev->descriptor.idProduct));
    kfree(tempBuffer);

    /* Initializing device data */
    init_device_data();

    /* Initializing input device */
2791  init_myMouse_input(myMouse->inputPollDev, usbdev, myMouse);

    /* create endpoint pipe */
    myMouse->in_pipe = usb_rcvbulkpipe(usbdev, in_endpoint->bEndpointAddress);
    myMouse->out_pipe = usb_sndbulkpipe(usbdev, out_endpoint->bEndpointAddress);

    /* initialize the poll_dev handler */
    myMouse->inputPollDev->private = myMouse;
    myMouse->inputPollDev->poll = myMouse_readData;      /* need to define this function */

2801  /* following command is to disable last to receiver: enabling only the first one */
    send_buf_com = kmalloc(myMouse->dataLength, GFP_KERNEL);
    send_buf_com[0] = 'S';
    send_buf_com[1] = 'A';
    send_buf_com[2] = '1';
    send_buf_com[3] = '0';
    send_buf_com[4] = '0';
    send_buf_com[5] = '\0';
    mutex_lock(&myMouse->bulk_io_mutex);
    retVal = usb_bulk_msg(usbdev, myMouse->out_pipe, send_buf_com, myMouse->dataLength, &
        bytes_written, 10000); /* timeout time is 10 sec */
2811  if (retVal != 0) {
      free_myMouseData(myMouse);
      kfree(send_buf_com);
      mutex_unlock(&myMouse->bulk_io_mutex);
      printk(KERN_INFO "myMouse_driver: usb_myMouse_probe: failed to send command for only
          enabling receiver 0 to wintracker.\n");
      return retVal;
    }

    mutex_unlock(&myMouse->bulk_io_mutex);
    kfree(send_buf_com);
```

```
2821
      /* registering input device */
      errNo = input_register_polled_device(myMouse->inputPollDev);
      if (errNo) {
        printk(KERN_INFO "myMouse: input device register failed, errNo: %d.\n", errNo);
        free_myMouseData(myMouse);
        usb_set_intfdata(intf, NULL);
        return errNo;
      }
      else {
2831    printk(KERN_INFO "myMouse: Detected device: %s.\n", myMouse->name);
        usb_set_intfdata(intf, myMouse);  /* saving input device structure information */

        retVal = sysfs_create_group(&usbdev->dev.kobj, &attr_group);
        if (retVal) {
          printk(KERN_INFO "myMouse: could not create sysfs nodes.\n");
        }

        return 0;
      }
2841 }

     static void usb_myMouse_disconnect(struct usb_interface *intf)
     {
       myMouseType *mouse = usb_get_intfdata(intf);
       struct usb_device *udev = mouse->usbdev;

       usb_set_intfdata(intf, NULL); /* setting interface to NULL value */
       if (mouse) {
         input_unregister_polled_device(mouse->inputPollDev);
2851     free_myMouseData(mouse);

         sysfs_remove_group(&udev->dev.kobj, &attr_group);
       }
     }

     /* ############## Module structure and initialization ############## */

     /* Module parameters */
     static int maxXRes = MAX_SCREEN_X;
2861 module_param(maxXRes, uint, 0644);
     MODULE_PARM_DESC(xres, "Maximum horizontal resolution.");

     static int maxYRes = MAX_SCREEN_Y;
     module_param(maxYRes, uint, 0644);
     MODULE_PARM_DESC(yres, "Maximum vertical resolution.");

     /* Following devices will work with the driver */
     static struct usb_device_id usb_myMouse_id_table [] = {
       { USB_DEVICE(DEVICE_VENDOR_ID, DEVICE_PRODUCT_ID) },  /* USB_DEVICE - macro defined in
           linux/usb.h file */
2871   { }              /* Terminating entry */
     };
     MODULE_DEVICE_TABLE (usb, usb_myMouse_id_table);     /* binding the module with the device */

     static struct usb_driver usb_myMouse_driver = {
       .name = "myMouse_usb",
       .probe = usb_myMouse_probe,
       .disconnect = usb_myMouse_disconnect,
       .id_table = usb_myMouse_id_table,
     };
2881
     /* module initialization - called when the module is installed */
     static int __init usb_myMouse_init(void)
     {
       int retVal;
       retVal = usb_register(&usb_myMouse_driver); /* registering the driver with the usb
           subsystem */
       if (retVal) {
         printk(KERN_INFO "myMouse: usb_register failed. Error number %d", retVal);
       }

2891   /* info(DRIVER_VERSION " : " DRIVER_DESC); */
       return retVal;
     }

     /* module exit - called when the module is removed */
     static void __exit usb_myMouse_exit(void)
     {
       usb_deregister(&usb_myMouse_driver);  /* deregistering the driver with the usb subsystem */
     }

2901 /* Module properties */
     MODULE_AUTHOR(DRIVER_AUTHOR);
     MODULE_DESCRIPTION(DRIVER_DESC);
     MODULE_LICENSE(DRIVER_LICENSE);
     MODULE_VERSION(DRIVER_VERSION);
```

```
module_init(usb_myMouse_init);
module_exit(usb_myMouse_exit);
```

B.2 Initialization Daemon Source Code

Listing B.2: Code segment of Initialization Daemon

```
int main(void)
{
  rframeType active_rframe;
  scrInfo saved_SrcData, X_SrcData;
  typeCalValues calVal;
  rpenParam parameterlist;
  type3Dpoint point1, point3;
  int retVal, aspectRChangeFlg;
  char msg[500];

      daemonize();
  openlog(DAEMON_NAME, LOG_PID, LOG_DAEMON);
  syslog(LOG_INFO, "Starting Daemon.");
  closelog();
  log_message(LOG_FILENAME, "rframe_Init: Daemon initialization done");

  retVal = getSysfsNodePath(&rpenParam_filePath);
  if (retVal != 0) {
    log_message(LOG_FILENAME, "rframe_Init: sysfs file path not found");
  }

      /* The infinite Loop */
      while (1) {
    /* 1st check if there is a valid XML file present */
        retVal = isValidXMLExists();
    if (retVal == 0) {      /* valid */
      retVal = getActiveFrameTemplate(&active_rframe);
      if (retVal == 0) {    /* has active template */
        retVal = isScreenExists();
        if (retVal > 0) {   /* screen data exists */
          retVal = getScreenData(&saved_SrcData);
          retVal = getDefaultScreenInfo(&X_SrcData);
          if (retVal == 0) {    /* successfully got Xwindows screen data */
            if ((saved_SrcData.defScrWidth == X_SrcData.defScrWidth) && (saved_SrcData.
                defScrHeight == X_SrcData.defScrHeight)) {   /* startup situation - write all
                 data in sysfs */
              retVal = isSysfsNodeExists(&rpenParam_filePath);
              if (retVal == 0) {  /* sysfs node exists */
                log_message(LOG_FILENAME, "rframe_Init: Got sysfs");
                /* check the sysfs node data is up to date */
                retVal = readrframeSizeParam(&parameterlist, &rpenParam_filePath);
                retVal = readScreenResParam(&parameterlist, &rpenParam_filePath);
                if (retVal == -1) {
                  log_message(LOG_FILENAME, "rframe_Init: Something is wrong getting sysfs
                      data!!");
                }
                if (!((active_rframe.rframeWidth == parameterlist.rfWidth) && (saved_SrcData.
                    defScrWidth == parameterlist.xRes))) {
                  /* not up to date - write sysfs data */
                  retVal = writeAllSysfsData(&active_rframe, &saved_SrcData);
                  if (retVal == 0) {
                    log_message(LOG_FILENAME, "rframe_Init: DEBUG: successfully wrote in
                        sysfs");
                    /* leave and wait */
                  } else {
                    log_message(LOG_FILENAME, "rframe_Init: ERROR: couldn't write in sysfs");
                    /* leave and wait */
                  }
                } else {
                  /* everything is OK */
                  log_message(LOG_FILENAME, "rframe_Init: DEBUG: everthing OK");
                  /* leave and wait */
                }
              } else {
                log_message(LOG_FILENAME, "rframe_Init: DEBUG: sysfs node does not exist");
                snprintf(msg, 500, "Error: main.c: main: sysfs path: \"%s\"",
                    rpenParam_filePath);
                log_message(LOG_FILENAME, msg);

                retVal = getSysfsNodePath(&rpenParam_filePath);
                if (retVal != 0) {
                  log_message(LOG_FILENAME, "rframe_Init: sysfs file path not found");
                }
                /* leave and wait */
              }
            } else {  /* screen resolution changed */
              /* recalculate related data */
              retVal = calculateAspectRat(X_SrcData.defScrWidth, X_SrcData.defScrHeight, &
                  calVal);
              X_SrcData.aspectRatio = calVal.aspectRatio;
```

```
retVal = calculateSrcRes(X_SrcData.defScrWidth, X_SrcData.defScrWidthMM, &
    calVal);
X_SrcData.srcMMtoPX = calVal.srcDotsPerMM;
X_SrcData.selected = 1;
if (retVal == 0) {
  if (calVal.aspectRatio != saved_SrcData.aspectRatio) {
    aspectRChangeFlg = 1;
    log_message(LOG_FILENAME, "ERROR: aspect ratio of the new set screen does
        not match the previous one");
    /* aspect ratio changed - restricted for the time being */
    point1.X = active_rframe.X1;
    point1.Y = active_rframe.Y1;
    point1.Z = active_rframe.Z1;
    point3.X = active_rframe.X3;
    point3.Y = active_rframe.Y3;
    point3.Z = active_rframe.Z3;

    /*  in case of generic calculate height */
    if (active_rframe.rframeType == 0) {
      retVal = calFrameHeightByWidth(active_rframe.rframeWidth, X_SrcData.
          aspectRatio, &calVal);
      active_rframe.rframeHeight = calVal.rframeHeight;
      retVal = calPointByDis(&point1, &point3, active_rframe.rframeHeight);
    /*  in case of XY-plane calculate width */
    } else if (active_rframe.rframeType == 1) {
      retVal = calFrameWidthByHeight(active_rframe.rframeHeight, X_SrcData.
          aspectRatio, &calVal);
      active_rframe.rframeWidth = calVal.rframeWidth;
      retVal = calPointByDis(&point1, &point3, active_rframe.rframeWidth);
    /*  in case of XZ-plane calculate height */
    } else if (active_rframe.rframeType == 2) {
      retVal = calFrameHeightByWidth(active_rframe.rframeWidth, X_SrcData.
          aspectRatio, &calVal);
      active_rframe.rframeHeight = calVal.rframeHeight;
      retVal = calPointByDis(&point1, &point3, active_rframe.rframeHeight);
    /*  in case of YZ-plane calculate width */
    } else if (active_rframe.rframeType == 3) {
      retVal = calFrameWidthByHeight(active_rframe.rframeHeight, X_SrcData.
          aspectRatio, &calVal);
      active_rframe.rframeWidth = calVal.rframeWidth;
      retVal = calPointByDis(&point1, &point3, active_rframe.rframeWidth);
    }

    active_rframe.X3 = point3.X;
    active_rframe.Y3 = point3.Y;
    active_rframe.Z3 = point3.Z;

    retVal = updateFrameTemplate(active_rframe.rfName, &active_rframe);
    if (retVal == 0) {
      log_message(LOG_FILENAME, "rframe_Init: DEBUG: successfully saved rframe
          data in XML");
    } else {
      log_message(LOG_FILENAME, "rframe_Init: DEBUG: could not save rframe data
           in XML");
    }
  } else {
    /* screen res changed with same aspect ratio */
    aspectRChangeFlg = 0;
  }
  retVal = updateScreenData(&X_SrcData);
  if (retVal == 0) {
    log_message(LOG_FILENAME, "rframe_Init: DEBUG: successfully saved screen
        data in XML");
  } else {
    log_message(LOG_FILENAME, "rframe_Init: DEBUG: could not save screen data
        in XML");
  }
  retVal = updateMultiFactorData(X_SrcData.defScrHeight, active_rframe.
      rframeHeight);
  if (retVal == 0) {
    log_message(LOG_FILENAME, "rframe_Init: DEBUG: successfully saved
        multiFactor data in XML");
  } else {
    log_message(LOG_FILENAME, "rframe_Init: DEBUG: could not save multiFactor
        data in XML");
  }
}

/* check if sysfs node exists */
retVal = isSysfsNodeExists(&rpenParam_filePath);
if (retVal == 0) {  /* sysfs node exists */
  if (aspectRChangeFlg == 0) {    /* only write screen related info */
    retVal = writeSrcSysfsData(&active_rframe, &X_SrcData);
    if (retVal == 0) {
      log_message(LOG_FILENAME, "rframe_Init: DEBUG: successfully wrote in
          sysfs");
      /* leave and wait */
```

```
                    } else {
                      log_message(LOG_FILENAME, "rframe_Init: ERROR: couldn't write in sysfs");
                      /* leave and wait */
                    }
151                 } else {        /* write all info */
                    retVal = writeAllSysfsData(&active_rframe, &X_SrcData);
                    if (retVal == 0) {
                      log_message(LOG_FILENAME, "rframe_Init: DEBUG: successfully wrote in
                          sysfs");
                      /* leave and wait */
                    } else {
                      log_message(LOG_FILENAME, "rframe_Init: ERROR: couldn't write in sysfs");
                      /* leave and wait */
                    }
                  }
161               } else {
                    log_message(LOG_FILENAME, "rframe_Init: DEBUG: sysfs node does not exist");
                    /* leave and wait */
                    retVal = getSysfsNodePath(&rpenParam_filePath);
                    if (retVal != 0) {
                      log_message(LOG_FILENAME, "rframe_Init: sysfs file path not found");
                    }
                  }
                }
              }
171         } else {
              log_message(LOG_FILENAME, "rframe_Init: DEBUG: no screen info");
              /* leave and wait */
            }
          } else {
            log_message(LOG_FILENAME, "rframe_Init: DEBUG: no active rframe");
            /* leave and wait */
          }
        } else {
          log_message(LOG_FILENAME, "rframe_Init: DEBUG: no valid XML file");
181       /* leave and wait */
        }

        openlog(DAEMON_NAME, LOG_PID, LOG_DAEMON);
        syslog(LOG_INFO, "Now pausing for 120 second.");
        closelog();
        log_message(LOG_FILENAME, "rframe_Init: Now pausing for 120 second");
            sleep(30);     /* wait 2 min */
          }

191     log_message(LOG_FILENAME, "rframe_Init: End of Daemon operation");
        openlog(DAEMON_NAME, LOG_PID, LOG_DAEMON);
        syslog(LOG_INFO, "Ending Daemon.");
        closelog();
        exit(EXIT_SUCCESS);
    }
```

B.3 Feedback Daemon Source Code

Listing B.3: Code segment of Feedback Daemon

```
int main(void)
{
  struct timespec interval, remainder;
  typeMovement last_movement, current_position;
  rframeType active_rframe;
  int retVal, counter, scrout_sound_flag, leftBtnClick_flag, leftClick_sound_flag;
  long timeDiff_sec;
  gboolean autoAdjMode, cursor_ChStatus;
        Cursor outbd_cursor, clutch_cursor, apro_surThes_cursor, apro_selThes_cursor,
            left_click_cursor, surProx_cursor;
  char msg[500];

  interval.tv_sec = 0;
  interval.tv_nsec = 500000000;      /* 500 msec */

        daemonize();
  openlog(DAEMON_NAME, LOG_PID, LOG_DAEMON);
  syslog(LOG_INFO, "Starting Daemon.");
  closelog();
  log_message(LOG_FILENAME, "rframe_FBAutoA: Daemon initialization done");

  /* daemon specific initialization */
  last_movement.X = 0;
  last_movement.Y = 0;
  last_movement.Z = 0;
  last_movement.event_time = 0;
  scrout_sound_flag = 0;
  leftClick_sound_flag = 0;
  autoAdj_Status = 0;        /* OFF */
  autoAdjMode = FALSE;
  cursor_ChStatus = FALSE;
  display = XOpenDisplay(NULL);
  root = RootWindow(display, DefaultScreen(display));
  outbd_cursor = XCreateFontCursor(display, XC_iron_cross);
  default_cursor = XCreateFontCursor(display, XC_left_ptr);
  surProx_cursor = XCreateFontCursor(display, XC_double_arrow);
  clutch_cursor = XCreateFontCursor(display, XC_diamond_cross);
  apro_surThes_cursor = XCreateFontCursor(display, XC_based_arrow_down);
  apro_selThes_cursor = XCreateFontCursor(display, XC_dotbox);
  left_click_cursor = XCreateFontCursor(display, XC_based_arrow_up);

  sprintf(msg, "My Process ID: %d.\n", getpid());
  log_message(LOG_FILENAME, msg);

  retVal = getSysfsNodePath(&rpenParam_filePath);      /* getting the sysfs file path */
  retVal = getSetXMLData();
  snprintf(msg, 500, "Threshold values: surthreshold: %d and selthreshold: %d", surthreshold,
      selthreshold);
  log_message(LOG_FILENAME, msg);

  counter = 0;

        /* The infinite Loop */
        while (1) {

    retVal = isSysfsNodeExists(&rpenParam_filePath);
    if (retVal != 0) {
      retVal = getSysfsNodePath(&rpenParam_filePath);      /* getting the sysfs file path */
    }

    retVal = readSensorPosParam(&current_position, &rpenParam_filePath);
    snprintf(msg, 500, "retVal value of readSensorPosParam: %d", retVal);
    log_message(LOG_FILENAME, msg);
    retVal = readClickEventParam(&leftBtnClick_flag, &rpenParam_filePath);
    snprintf(msg, 500, "retVal value of readClickEventParam: %d and leftBtnClick_flag value:
        %d", retVal, leftBtnClick_flag);
    log_message(LOG_FILENAME, msg);

    if ((last_movement.X != current_position.X) || (last_movement.Y != current_position.Y) ||
        (last_movement.Z != current_position.Z)) {
    /* there was a movement */
      if (last_movement.event_time != 0) {
        timeDiff_sec = (long)(labs(current_position.event_time - last_movement.event_time));
        snprintf(msg, 500, "Time Difference: %ld", timeDiff_sec);
        log_message(LOG_FILENAME, msg);
      } else {
        snprintf(msg, 500, "last_movement.event_time: %ld", (long) last_movement.event_time);
        log_message(LOG_FILENAME, msg);
      }
```

```
snprintf(msg, 500, "value of autoAdj_Status: %d", autoAdj_Status);
log_message(LOG_FILENAME, msg);

if (autoAdj_Status == 1) {    /* if the auto adjust mode is ON */
  if ((autoAdjMode == FALSE) && (timeDiff_sec > inActive_limit)) {
    autoAdjMode = TRUE;
    /* set the transfer function not to be used by the device driver */
    /* to show actual sensor position */
    retVal = writeFlagParam(0, -1, -1, &rpenParam_filePath);
    if (retVal == -1) {
      log_message(LOG_FILENAME, "Could not save flag info in sysfs node");
        }
  } else if ((autoAdjMode == TRUE) && ((timeDiff_sec > 5) && (timeDiff_sec <
      inActive_limit))) {
    /* get the current rframe orientation and adjust the new rframe orientation
       accordingly */
    retVal = getActiveFrameTemplate(&active_rframe);
    if (retVal == -1) {
      log_message(LOG_FILENAME, "Could not get active rframe data from XML");
        }
        if (active_rframe.rframeType == 0) {      /* if the rframe is a generic one
            then */
          active_rframe.rframeType = genericOri;
        }

        log_message(LOG_FILENAME, "Going to re-define the active rframe");

    active_rframe.X1 = last_movement.X;
    active_rframe.Y1 = last_movement.Y;
    active_rframe.Z1 = last_movement.Z;

    /* calculate the other rframe points */
    if (active_rframe.rframeType == 1) {       /* if the rframe type = ON XY plane */
          active_rframe.X2 = active_rframe.X1;
      active_rframe.Y2 = (active_rframe.Y1 + active_rframe.rframeHeight);
      active_rframe.Z2 = active_rframe.Z1;
          active_rframe.X3 = (active_rframe.X1 + active_rframe.rframeWidth);
      active_rframe.Y3 = active_rframe.Y1;
      active_rframe.Z3 = active_rframe.Z1;
        } else if (active_rframe.rframeType == 2) {     /* if the rframe type = ON XZ
            plane */
          active_rframe.X2 = (active_rframe.X1 + active_rframe.rframeWidth);
      active_rframe.Y2 = active_rframe.Y1;
      active_rframe.Z2 = active_rframe.Z1;
      active_rframe.X3 = active_rframe.X1;
      active_rframe.Y3 = active_rframe.Y1;
      active_rframe.Z3 = (active_rframe.Z1 + active_rframe.rframeHeight);
        } else if (active_rframe.rframeType == 3) {     /* if the rframe type = ON YZ
            plane */
          active_rframe.X2 = active_rframe.X1;
      active_rframe.Y2 = active_rframe.Y1;
      active_rframe.Z2 = (active_rframe.Z1 + active_rframe.rframeHeight);
          active_rframe.X3 = active_rframe.X1;
      active_rframe.Y3 = (active_rframe.Y1 + active_rframe.rframeWidth);
      active_rframe.Z3 = active_rframe.Z1;
        }
    /* the new rframe have the same width and height as the old one */
    /* save (update) the new rframe in XML */
    retVal = updateFrameTemplate(active_rframe.rfName, &active_rframe);
    if (retVal == -1) {
      log_message(LOG_FILENAME, "Could not update updated rframe data in XML");
        }
    /* save in sysfs */
    retVal = writeAllSysfsData(&active_rframe, &saved_src_Info);
    if (retVal == -1) {
      log_message(LOG_FILENAME, "Could not save sysfs date");
        }
    autoAdjMode = FALSE;
  }
}

if (autoAdjMode == FALSE) {

  snprintf(msg, 500, "position values: x: %d, y: %d and z: %d", current_position.X,
      current_position.Y, current_position.Z);
  log_message(LOG_FILENAME, msg);

  /* do the processing for changing the cursor and making sound if needed */
  /* check the current_movement values with xres and yres in saved_src_Info */
  /* if current values are greater then change cursor appearance and make sound */
  /* use gdk_window_set_cursor () for setting the cursor appearance */
  /* need to find a way to produce the sound */
  if ((current_position.X > saved_src_Info.defScrWidth) || (current_position.Y >
      saved_src_Info.defScrHeight) || (current_position.X < 0) || (current_position.Y <
       0)) {  /* out of screen */
    log_message(LOG_FILENAME, "out of screen is called");
    if (scrout_sound_flag != 1) {
```

```
            cursor_ChStatus = TRUE;
            scrout_sound_flag = 1;
            gnome_sound_play(OUTSRC_SND_FILE);
            log_message(LOG_FILENAME, "played the out of screen sound");
            XDefineCursor(display, root, outbd_cursor);
            XFlush(display);
          }
        } else if ((current_position.Z < 0) && (abs(current_position.Z) >
            POSI_CLUTCH_THRESHOLD)) { /* above the surface (+) clutch */
          log_message(LOG_FILENAME, "positive clutch is called");
          if (scrout_sound_flag != 1) {
            cursor_ChStatus = TRUE;
            scrout_sound_flag = 1;
            gnome_sound_play(OUTSRC_SND_FILE);
            log_message(LOG_FILENAME, "played the out of screen sound");
            XDefineCursor(display, root, clutch_cursor);
            XFlush(display);
          }

        } else if ((current_position.Z < 0) && (abs(current_position.Z) <= surfaceProximity))
            { /* just above the surface */
          log_message(LOG_FILENAME, "surface poximity is called");
          cursor_ChStatus = TRUE;
          XDefineCursor(display, root, surProx_cursor);
          XFlush(display);
          if (scrout_sound_flag == 1) {
            scrout_sound_flag = 0;
          }
        } else if ((current_position.Z >= 0) && (current_position.Z <= surthreshold)) { /*
            within surface and surface threshold */
          log_message(LOG_FILENAME, "approaching surface threshold is called");
          cursor_ChStatus = TRUE;
          XDefineCursor(display, root, apro_surThes_cursor);
          XFlush(display);
          if (scrout_sound_flag == 1) {
            scrout_sound_flag = 0;
          }
        } else if ((current_position.Z > 0) && (current_position.Z > surthreshold) && (
            current_position.Z <= selthreshold) && ((last_movement.Z - current_position.Z) >
            0)) { /* between surface threshold and selThreshold + coming back */
          log_message(LOG_FILENAME, "approaching selection threshold is called");
          cursor_ChStatus = TRUE;
          XDefineCursor(display, root, apro_selThes_cursor);
          XFlush(display);
          if (scrout_sound_flag == 1) {
            scrout_sound_flag = 0;
          }
        } else if ((current_position.Z > 0) && (current_position.Z > surthreshold)) { /*
            beyond surface threshold */
          log_message(LOG_FILENAME, "inside click-event range is called");
          cursor_ChStatus = TRUE;
          XDefineCursor(display, root, left_click_cursor);
          XFlush(display);
          if (scrout_sound_flag == 1) {
            scrout_sound_flag = 0;
          }
        } else if (cursor_ChStatus == TRUE) {
          log_message(LOG_FILENAME, "converting cursor to default");
          cursor_ChStatus = FALSE;
          XDefineCursor(display, root, default_cursor);
          XFlush(display);
          if (scrout_sound_flag == 1) {
            scrout_sound_flag = 0;
          }
        }

        if (leftBtnClick_flag == 1) {
          if (leftClick_sound_flag != 1) {
            leftClick_sound_flag = 1;
            gnome_sound_play(CLICK_SND_FILE);
            log_message(LOG_FILENAME, "played the left click sound");
          }
        } else {
          if (leftClick_sound_flag == 1) {
            leftClick_sound_flag = 0;
            gnome_sound_play(UNCLICK_SND_FILE);
            log_message(LOG_FILENAME, "played the left unclick sound");
          }
        }

      } else {
        log_message(LOG_FILENAME, "autoAdjMode value was TRUE");
      }

      snprintf(msg, 500, "value of current_movement.event_time: %ld", (long) current_position
          .event_time);
      log_message(LOG_FILENAME, msg);
```

```
        /* change last movement data */
        last_movement.X = current_position.X;
        last_movement.Y = current_position.Y;
        last_movement.Z = current_position.Z;
        last_movement.event_time = current_position.event_time;
      }

244   if (nanosleep(&interval, &remainder) == -1) {
        if (errno == EINTR) {
          log_message(LOG_FILENAME, "nanosleep interrupted by EINTR");
        } else {
          log_message(LOG_FILENAME, "nanosleep interrupted by EINTR");
          snprintf(msg, 500, "nanosleep interrupted with errno: %d", errno);
          log_message(LOG_FILENAME, msg);
        }
      }

254   counter++;
      if (counter == 100) {
        retVal = getSetXMLData();
        snprintf(msg, 500, "Threshold values: surthreshold: %d and selthreshold: %d",
            surthreshold, selthreshold);
        log_message(LOG_FILENAME, msg);
        counter = 0;
      }
          }

    XFreeCursor(display, outbd_cursor);
264         XFreeCursor(display, clutch_cursor);
            XFreeCursor(display, apro_surThes_cursor);
            XFreeCursor(display, apro_selThes_cursor);
            XFreeCursor(display, left_click_cursor);
            XFreeCursor(display, surProx_cursor);
            XFreeCursor(display, default_cursor);

            XCloseDisplay(display);

    log_message(LOG_FILENAME, "rframe_FBAutoA: End of Daemon operation");
274 openlog(DAEMON_NAME, LOG_PID, LOG_DAEMON);
    syslog(LOG_INFO, "Ending Daemon.");
    closelog();
    exit(EXIT_SUCCESS);
  }
```

B.4 Rframe Definition Application Source Code

Listing B.4: Code segment of Rframe Definition Application

```
/* #### Calculation functions ######### */

/* checking if two lines are perpendicular */
int isPerpendicular(type3Dpoint *point1, type3Dpoint *point2, type3Dpoint *point3) {
  type3Dpoint resVector1, resVector2;
  int vecDotValue;

  resVector1.X = point1->X - point2->X;
      resVector1.Y = point1->Y - point2->Y;
      resVector1.Z = point1->Z - point2->Z;

  resVector2.X = point3->X - point1->X;
      resVector2.Y = point3->Y - point1->Y;
      resVector2.Z = point3->Z - point1->Z;

  vecDotValue = resVector1.X*resVector2.X + resVector1.Y*resVector2.Y + resVector1.Z*
      resVector2.Z;

  if (vecDotValue == 0) {   /* the two vectors are perpendicular */
    return 0;
  } else {
    return -1;
  }
}

/* re-adjusts point3 according to point1, point2 and the rframe type */
int calPointByDisPlusLine(type3Dpoint *point1, type3Dpoint *point2, type3Dpoint *point3,
    type3Dpoint *retPoint, int distance) {
  type3Dpoint tmpVectorP2P1, tmpVectorP1P3, tmpVecP1Pprim3;
  type3Dpoint projVP2P1OnP1P3;
  int vecDotValue, lengthSqrP2P1, lengthP1Pprim3, retValue;

  tmpVectorP2P1.X = point1->X - point2->X;
      tmpVectorP2P1.Y = point1->Y - point2->Y;
      tmpVectorP2P1.Z = point1->Z - point2->Z;

  tmpVectorP1P3.X = point3->X - point1->X;
      tmpVectorP1P3.Y = point3->Y - point1->Y;
      tmpVectorP1P3.Z = point3->Z - point1->Z;

  vecDotValue = tmpVectorP2P1.X*tmpVectorP1P3.X + tmpVectorP2P1.Y*tmpVectorP1P3.Y +
      tmpVectorP2P1.Z*tmpVectorP1P3.Z;
  lengthSqrP2P1 = (tmpVectorP2P1.X*tmpVectorP2P1.X) + (tmpVectorP2P1.Y*tmpVectorP2P1.Y) + (
      tmpVectorP2P1.Z*tmpVectorP2P1.Z);

  projVP2P1OnP1P3.X = (vecDotValue/lengthSqrP2P1)*tmpVectorP2P1.X;
  projVP2P1OnP1P3.Y = (vecDotValue/lengthSqrP2P1)*tmpVectorP2P1.Y;
  projVP2P1OnP1P3.Z = (vecDotValue/lengthSqrP2P1)*tmpVectorP2P1.Z;

  tmpVecP1Pprim3.X = tmpVectorP1P3.X - projVP2P1OnP1P3.X;
  tmpVecP1Pprim3.Y = tmpVectorP1P3.Y - projVP2P1OnP1P3.Y;
  tmpVecP1Pprim3.Z = tmpVectorP1P3.Z - projVP2P1OnP1P3.Z;

  lengthP1Pprim3 = sqrt(tmpVecP1Pprim3.X*tmpVecP1Pprim3.X + tmpVecP1Pprim3.Y*tmpVecP1Pprim3.Y
      + tmpVecP1Pprim3.Z*tmpVecP1Pprim3.Z);

  /* new adjusted point3 */
  retPoint->X = point1->X + distance*(tmpVecP1Pprim3.X/lengthP1Pprim3);
  retPoint->Y = point1->Y + distance*(tmpVecP1Pprim3.Y/lengthP1Pprim3);
  retPoint->Z = point1->Z + distance*(tmpVecP1Pprim3.Z/lengthP1Pprim3);

  retValue = isPerpendicular(point1, point2, retPoint);
  if (retValue == 0)
    return 0;
  else
    return -1;
}
```

B.5 XML Data Format to Store Rframe Data

Listing B.5: XML Document Type Definition (DTD)

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE rframeTemplates [
 <!ELEMENT rframeTemplates (Templates , Screens , InputRes , SensorRange , AvgSampleNo ,
     MultiFactor , AutoAdjustMode)>
<!ELEMENT Templates (Template)*>
<!ELEMENT Template (rfType , rfName , Active , Points , SurfaceThres , SelectThres ,
     SurProximity , rfWidth , rfHeight)>
<!ELEMENT rfType (#PCDATA)>
<!ELEMENT rfName (#PCDATA)>
<!ELEMENT Active (#PCDATA)>
<!ELEMENT Points (Point)+>
<!ELEMENT Point (X , Y , Z)>
<!ELEMENT X (#PCDATA)>
<!ELEMENT Y (#PCDATA)>
<!ELEMENT Z (#PCDATA)>
<!ELEMENT SurfaceThres (#PCDATA)>
<!ELEMENT SelectThres (#PCDATA)>
<!ELEMENT SurProximity (#PCDATA)>
<!ELEMENT rfWidth (#PCDATA)>
<!ELEMENT rfHeight (#PCDATA)>
<!ELEMENT Screens (Screen)*>
<!ELEMENT Screen (ScreenWidthPX , ScreenHeightPX , ScreenWidthMM , ScreenHeightMM ,
     AspectRatio , SrcMMtoPX , Selected)>
<!ELEMENT ScreenWidthPX (#PCDATA)>
<!ELEMENT ScreenHeightPX (#PCDATA)>
<!ELEMENT ScreenWidthMM (#PCDATA)>
<!ELEMENT ScreenHeightMM (#PCDATA)>
<!ELEMENT AspectRatio (#PCDATA)>
<!ELEMENT SrcMMtoPX (#PCDATA)>
<!ELEMENT Selected (#PCDATA)>
<!ELEMENT InputRes (#PCDATA)>
<!ELEMENT SensorRange (#PCDATA)>
<!ELEMENT AvgSampleNo (#PCDATA)>
<!ELEMENT MultiFactor (srcMultiFactor , rfMultiFactor)>
<!ELEMENT srcMultiFactor (#PCDATA)>
<!ELEMENT rfMultiFactor (#PCDATA)>
<!ELEMENT AutoAdjustMode (AutoActive , InActiveMin , GenericRfOri)>
<!ELEMENT AutoActive (#PCDATA)>
<!ELEMENT InActiveMin (#PCDATA)>
<!ELEMENT GenericRfOri (#PCDATA)>
]>
```

References

[Apl00] J. Dwight Aplevich. *The Essentials of Linear State-Space Systems*. John Wiley & Sons, New York, NY, USA, 2000.

[Bac00] Eric R. Bachmann. *Inertial and Magnetic Tracking of Limb Segment Orientation for Inserting Humans into Synthetic Environments*. PhD thesis, Naval Postgraduate School, Monterey, CA, USA, December 2000.

[BS90] Teresa W. Bleser and John Sibert. "Toto: a tool for selecting interaction techniques." In *UIST '90: Proceedings of the 3rd annual ACM SIGGRAPH symposium on User interface software and technology*, pp. 135–142, New York, NY, USA, 1990. ACM.

[Car03] John M. Carroll, editor. *HCI Models, Theories, and Frameworks: Towards a Mutidisciplinary Science*. Morgan Kaufmann Publishers, San Francisco, CA, USA, 2003.

[CLR01] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Riverst, and Clifford Stein. *Introduction to Algorithms*. MIT Press, Cambridge, MA, USA, 2nd edition, 2001.

[Con09] Sixense's TrueMotion Controller. http://www.ecnmag.com/CES-2009-Best-of-011609.aspx, 22 September 2009.

[Cor09] Ascension Technology Corporation. Burlington, Vermont, USA. http://www.ascension-tech.com, 8 August 2009.

[CRK05] Jonathan Corbet, Alessandro Rubini, and Greg Kroah-Hartman. *Linux Device Drivers*. O'Reilly Media, Inc., Sebastopol, CA, USA, 3rd edition, February 2005.

[Dav61] Harry F. Davis. *Introduction to Vector Analysis*. Allyn and Bacon, Inc., Boston, MA, USA, 1961.

[DTR07] Nguyen Thong Dang, Monica Tavanti, Ivan Rankin, and Matthew Cooper. "A comparison of different input devices for a 3D environment." In *ECCE '07: Proceedings of the 14th European conference on Cognitive ergonomics*, pp. 153–160, New York, NY, USA, 2007. ACM.

[Fel09] Mark Feldman. "The Win95 Game Programmer's Encyclopedia." Basic 3D Math: http://www.geocities.com/SiliconValley/2151/math3d.html, 24 September 2009.

[Fer91] Frank J. Ferrin. "Survey of Helmet Tracking Technologies." In *SPIE Proceedings, Vol. 1456: Large-Screen Projection, Avionic, and Helmet-Mounted Displays*, pp. 86–94, 1991.

[FH93] S. Sidney Fels and Geoffrey E. Hinton. "Glove-Talk: A neural network interface between a data-glove and a speech synthesizer." *IEEE Transactions on Neural Networks*, **3**(6):2–8, Jan 1993.

[FS06] Shinichi Fukushige and Hiromasa Suzuki. "Interactive 3D pointing device using mirror reflections." In *GRAPHITE '06: Proceedings of the 4th international conference on Computer graphics and interactive techniques in Australasia and Southeast Asia*, pp. 231–235, New York, NY, USA, 2006. ACM.

[Han97] Chris Hand. "A Survey of 3D Interaction Techniques." *Computer Graphics Forum*, **16**(5):269–281, 1997.

[HBF05] Johann B Hummel, Michael R. Bax, Michael L. Figl, Yan Kang, Calvin Maurer Jr., Wolfgang W. Birkfellner, Helmar Bergmann, and Ramin Shahidi. "Design and application of an assessment protocol for electromagnetic tracking systems." *Med. Phys.*, **32**(7):2371–9, July 2005.

[Her76] Christopher F. Herot. "Graphical input through machine recognition of sketches." *SIGGRAPH Comput. Graph.*, **10**(2):97–102, 1976.

[HHN85] Edwin L. Hutchins, James D. Hollan, and Donald A. Norman. "Direct Manipulation Interfaces." *Human-Computer Interaction*, **1**:311–338, 1985.

[Hin08] Ken Hinckley. "Input technologies and techniques." In *The human-computer interaction handbook: fundamentals, evolving technologies and emerging applications*, pp. 151–168. Lawrence Erlbaum Associates Inc., NJ, USA, 2nd edition, 2008.

[HPG94] Ken Hinckley, Randy Pausch, John C. Goble, and Neal F. Kassell. "A survey of design issues in spatial input." In *UIST '94: Proceedings of the 7th annual ACM symposium on User interface software and technology*, pp. 213–222, New York, NY, USA, 1994. ACM.

[IBH01] Milan Ikits, J. Dean Brederson, Charles D. Hansen, and John M. Hollerbach. "An Improved Calibration Framework for Electromagnetic Tracking Devices." In *VR '01: Proceedings of the Virtual Reality 2001 Conference (VR'01)*, pp. 63–70, Washington, DC, USA, 2001. IEEE Computer Society.

[Inc09a] InterSense Inc. Billerica, MA, USA. http://www.intersense.com, 5 September 2009.

[Inc09b] Northern Digital Inc. Waterloo, Ontario, Canada. http://www.ndigital.com/index.php, 7 September 2009.

[Inc09c] PhaseSpace Inc. San Leandro, CA, USA. http://www.phasespace.com/index.html, 5 September 2009.

[Inc09d] Sensable Technologies Inc. http://www.sensable.com/index.htm, 6 September 2009.

[Inc09e] Polhemus Incorporated. Colchester, Vermont, USA. http://www.polhemus.com, 8 August 2009.

[Ini09] Inition. London, UK. http://www.inition.co.uk, 11 August 2009.

[Jac96] Robert J. K. Jacob. "Human-computer interaction: input devices." *ACM Comput. Surv.*, **28**(1):177–179, 1996.

[JLM93] Robert J.K. Jacob, John J. Leggett, Brad A. Myers, and Randy Pausch. "Interaction Styles and Input/Output Devices." *Behaviour and Information Technology*, **12**(2):69–79, March-April 1993.

[JS92] Robert J. K. Jacob and Linda E. Sibert. "The perceptual structure of multidimensional input device selection." In *CHI '92: Proceedings of the SIGCHI conference on Human factors in computing systems*, pp. 211–218, New York, NY, USA, 1992. ACM.

[JSM94] Robert J. K. Jacob, Linda E. Sibert, Daniel C. McFarlane, and Jr. M. Preston Mullen. "Integrality and separability of input devices." *ACM Trans. Comput.-Hum. Interact.*, **1**(1):3–26, 1994.

[Kin99] Volodymyr Kindratenko. "Calibration of electromagnetic tracking devices." *Virtual Reality: Research, Development, and Applications*, **4**(2):139–150, 1999.

[Kin00] Volodymyr Kindratenko. "A survey of electromagnetic position tracker calibration techniques." *Virtual Reality: Research, Development, and Applications*, **5**(3):169–182, 2000.

[KWR06] Ernst Kruijff, Gerold Wesche, Kai Riege, Gernot Goebbels, Martijn Kunstman, and Dieter Schmalstieg. "Tactylus, a pen-input device exploring audiotactile sensory binding." In *VRST '06: Proceedings of the ACM symposium on Virtual reality software and technology*, pp. 312–315, New York, NY, USA, 2006. ACM.

[Len04] Eric Lengyel. *Mathematics for 3D Game Programming and Computer Graphics, Second Edition*. Charles River Media, Inc., Rockland, MA, USA, 2004.

[Lov05] Robert Love. *Linux Kernel Development*. Novell Press, Indianapolis, Indiana, USA, 2nd edition, June 2005.

[LS97] Mark A. Livingston and Andrei State. "Magnetic Tracker Calibration for Improved Augmented Reality Registration." *Presence: Teleoperators and Virtual Environments*, **6**(5):532–546, 1997.

[Ltd09] Animazoo UK Ltd. http://www.animazoo.com/default.aspx, 5 September 2009.

[MAB92] Kenneth Meyer, Hugh L. Applewhite, and Frank A. Biocca. "A survey of position trackers." *Presence: Teleoperators and Virtual Environments*, **1**(2):173–200, 1992.

[MCR90] Jock Mackinlay, Stuart K. Card, and George G. Robertson. "A semantic analysis of the design space of input devices." *Humam-Computer Interaction*, **5**(2):145–190, 1990.

[MI01] Atsuo Murata and Hirokazu Iwase. "Extending Fitts' Law to a three-dimensional pointing task." *Human Movement Science*, **20**:791–805, 2001.

[MIT09] SixthSense MIT Media Lab. http://www.media.mit.edu/research, 8 September 2009.

[MKS01] I. Scott MacKenzie, Tatu Kauppinen, and Miika Silfverberg. "Accuracy measures for evaluating computer pointing devices." In *CHI '01: Proceedings of the SIGCHI conference on Human factors in computing systems*, pp. 9–16, New York, NY, USA, 2001. ACM.

[Mot09] BurstGlove Motion4U. http://www.3dmotion4u.com, 8 September 2009.

[MPC04] F. Martinot, P. Plénacoste, and C. Chaillou. "The DigiTracker, a Three Degrees of Freedom Pointing Device." In *10th Eurographics Symposium on Virtual Environments*, pp. 99–104, June 2004.

[Nat09] LabVIEW National Instruments Corporation. TX, USA. http://www.ni.com, 17 September 2009.

[Nie93] Jakob Nielsen. *Usability Engineering*. Academic Press, New York, USA, 1993.

[NMF98] Mark A. Nixon, Bruce C. McCallum, W. Richard Fright, and N. Brent Price. "The Effects of Metals and Interfering Fields on Electromagnetic Trackers." *Presence: Teleoperators and Virtual Environments*, **7**(2):204–218, 1998.

[Obl09] g-speak Oblong Industries. http://oblong.com, 8 September 2009.

[Pro09a] Microsoft Project Natal. http://en.wikipedia.org/wiki/Project_Natal, 7 September 2009.

[Pro09b] Xbox 360 Project Natal. Microsoft, http://www.xbox.com/en-US/live/projectnatal, 7 September 2009.

[PV89] T.S. Perry and J. Voelcker. "Of mice and menus: designing the user-friendly interface." *IEEE Spectrum*, **26**(9):46–51, Sep 1989.

[RBG00] Jannick P. Rolland, Yohan Baillot, and Alexei A. Goon. *Fundamentals of Wearable Computers and Augmented Reality*. Lawrence Erlbaum, NJ, USA, 2000.

[RC08] Jeffrey Rubin and Dana Chisnell. *Handbook of Usability Testing: How to Plan, Design, and Conduct Effective Tests*. Wiley Publishing, Inc., Indianapolis, Indiana, USA, 2nd edition, May 2008.

[Sen09] Project Natal Sensor. http://xbox360.ign.com/articles/101/1016309p1.html, 22 September 2009.

[SHC96] Andrei State, Gentaro Hirota, David T. Chen, William F. Garrett, and Mark A. Livingston. "Superior augmented reality registration by integrating landmark tracking and magnetic tracking." In *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pp. 429–438, New York, NY, USA, 1996. ACM.

[Shn83] Ben Shneiderman. "Direct Manipulation: A Step Beyond Programming Languages." *Computer*, **16**(8):57–69, 1983.

[Six09] TrueMotion Sixense Entertainment. http://www.sixense.com, 8 September 2009.

[SK88] Robert D. Strum and Donald E. Kirk. *First principles of discrete systems and digital signal processing*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1988.

[SK04] Oliver Stefani and Ioannis Karaseitanidis. "Designing 3D input devices for immersive environments." In *2004 IEEE International Conference on Systems, Man and Cybernetics*, volume 7, pp. 6280–6285, Oct. 2004.

[Sko96] Paul F. Skopowski. *"Immersive Articulation of the Human Upper Body in a Virtual Environment."*. Master's thesis, Naval Postgraduate School, Monterey, California, USA, December 1996.

[SP05] Ben Shneiderman and Catherine Plaisant. *Designing the User Interface : Strategies for Effective Human-Computer Interaction*. Addison-Wesley, Boston, MA, USA, 4th edition, 2005.

[SPS92] Andrew Sears, Catherine Plaisant, and Ben Shneiderman. "A new era for high precision touchscreens." In *Advances in human-computer interaction (vol. 3)*, pp. 1–33. Ablex Publishing Corp., Norwood, NJ, USA, 1992.

[Ste03] James Stewart. *Calculus: Early Transcendentals*. Thomson Learning Inc., Belmont, CA, USA, 5th edition, 2003.

[SWK04] Ehud Sharlin, Benjamin Watson, Yoshifumi Kitamura, Fumio Kishino, and Yuichi Itoh. "On tangible user interfaces, humans and spatiality." *Personal Ubiquitous Comput.*, **8**(5):338–346, 2004.

[Sys09] Vicon Motion Systems. Colorado, USA. http://www.vicon.com/index.html, 5 September 2009.

[VB04] James M. Van Verth and Lars M. Bishop. *Essential Mathematics for Games and Interactive Applications: A Programmer's Guide*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2004.

[VB05] Daniel Vogel and Ravin Balakrishnan. "Distant freehand pointing and clicking on very large, high resolution displays." In *UIST '05: Proceedings of the 18th annual ACM symposium on User interface software and technology*, pp. 33–42, New York, NY, USA, 2005. ACM.

[Ven08] Sreekrishnan Venkateswaran. *Essential Linux Device Drivers*. Prentice Hall Press, Upper Saddle River, NJ, USA, 2008.

[VR 07] VR Space Inc., Taoyuan, Taiwan. *Wintracker: User's Manual*, Jan 2007.

[WBV99] Greg Welch, Gary Bishop, Leandra Vicci, Stephen Brumback, Kurtis Keller, and D'nardo Colucci. "The HiBall Tracker: high-performance wide-area tracking for virtual and augmented environments." In *VRST '99: Proceedings of the ACM symposium on Virtual reality software and technology*, pp. 1–10, New York, NY, USA, 1999. ACM.

[Wel09] Gregory F. Welch. "History: The use of the kalman filter for human motion tracking in virtual reality." *Presence: Teleoper. Virtual Environ.*, **18**(1):72–91, 2009.

[Wii09] Nintendo Wii. http://www.nintendo.com/wii/what/controllers, 22 September 2009.

[Win09] VR-Space Inc. Wintracker II. http://www.vr-space.com, 5 August 2009.

[WLH07] Richard S. Wright, Benjamin Lipchak, and Nicholas Haemel. *Opengl®superbible: comprehensive tutorial and reference*. Addison-Wesley Professional, fourth edition, June 2007.

[Won07] Alexander Wong. *"Low-Cost Visual/Inertial Hybrid Motion Capture System for Wireless 3D Controllers."*. Master's thesis, University of Waterloo, Waterloo, Ontario, Canada, 2007.

[Zha95] Shumin Zhai. *Human Performance in Six Degree of Freedom Input Control*. PhD thesis, University of Toronto, ON, Canada, 1995.

[Zha98] Shumin Zhai. "User performance in relation to 3D input device design." *SIGGRAPH Computer Graphics*, **32**(4):50–54, 1998.

[ZMB96] Shumin Zhai, Paul Milgram, and William Buxton. "The influence of muscle groups on performance of multiple degree-of-freedom input." In *CHI '96: Proceedings of the SIGCHI conference on Human factors in computing systems*, pp. 308–315, New York, NY, USA, 1996. ACM.

[ZMI99] Shumin Zhai, Carlos Morimoto, and Steven Ihde. "Manual and gaze input cascaded (MAGIC) pointing." In *CHI '99: Proceedings of the SIGCHI conference on Human factors in computing systems*, pp. 246–253, New York, NY, USA, 1999. ACM.