NTNU
Norwegian University of
Science and Technology

Tracking of Head Movements for Motion Control

Robert Salai

Master of Science in Engineering Cybernetics
Submission date: June 2009
Supervisor: Geir Mathisen, ITK
Co-supervisor: Sigurd A. Fjerdingen, SINTEF IKT
Aksel A. Transeth, SINTEF IKT

Norwegian University of Science and Technology
Department of Engineering Cybernetics

Problem Description

StatoilHydro har som målsetting å bygge fjernstyrte oljeplattformer. I den anledning jobber SINTEF og NTNU med en robotlab der vi ser på robotløsninger for inspeksjon og vedlikehold av slike installasjoner. Siden operatørene vil befinne seg på land i stedet for på selve platformen er menneske-maskin-grensesnittet mellom operatør og roboter på plattformen svært viktig. En svært relevant problemstilling er hvordan operatøren på en intuitiv måte skal kunne se hva som foregår på plattformen.

Oppgaven tar sikte på å videreutvikle en strategi som grunnlag for å bestemme posisjon og orientering (6 frihetsgrader) til operatørens hode ved bruk av LEDs og NIR/IR-kamera (Wiimote). Dette kan f.eks. benyttes til intuitiv kamerastyring eller direkte robotstyring.

Her er noen resultat fra Dr. Johnny Lee og andre som er morsomme og svært relevante:
http://www.youtube.com/watch?v=Jd3-eiid-Uw
http://www.cs.cmu.edu/~johnny/
http://www.wiili.org/Wiimote
http://www.wiili.org/index.php/Wiimote_driver

Oppgave:
1. Bakgrunnsstoff:
   a. Få en oversikt over eksisterende løsninger for objektfølging vha Wiimote og hvilke muligheter og begrensninger disse har.
   b. Gjør et litteratursøk på posisjons- og orienteringsestimering.
   c. Gjør en undersøkelse av hvilke metoder som finnes for operatørstyrt kamerastyring
2. Velg en eller flere eksisterende løsninger for objektfølging med Wiimote og implementer denne/disse.
3. Vurder løsningen(e) med hensyn på begrensninger, nøyaktighet og robusthet, og evnt. sammenlign løsningene.
4. Eventuell utvidelse hvis det blir tid:
   a. Implementer løsning for styring av pan-tilt-enhet med hodebevegelser.

Assignment given: 12. January 2009
Supervisor: Geir Mathisen, ITK

Summary

The capture of gestures in order to use them as input for intuitive control has been investigated exhaustively in recent years. However, for the most part this has resulted in relatively expensive devices. The contribution of this report is the investigation on the feasibility of the development of a low-cost vision based input device for the tracking of head movements, concerning the use of them for motion control.

The input device relies on the infrared camera, along with the built-in image analysis tools, present on a Nintendo Wii remote for the measurement of the location and orientation of a head-mountable marker. The marker consists of a set of optical feature points which are easily detectable, and organized in a fashion which allows for the determination of its position and orientation in space.

The developed input device was then evaluated in order to determine the operating range, accuracy and robustness, and was shown to be feasible for its intended use. Finally, the implemented device was utilized to control a mechanical output device, being a unit capable of panning and tilting.

Preface

This thesis has been carried out during the final semester for the degree of Master of Technology in Engineering Cybernetics at The Norwegian University of Science and Technology. The work for, and the writing of, this thesis was performed in the period from January to June 2009. The assignment was carried out under the supervision of PhD Aksel Andreas Transeth and MSc Sigurd Aksnes Fjerdingen at SINTEF IKT Applied Cybernetics.

I would like to thank my supervisors for providing such an interesting assignment, allowing me to immerse myself in a field previously unknown to me, and for their input and help throughout this thesis.

Thanks goes to my family for standing by me throughout this semester and the last five years of study. And finally, I would like to thank Kristin for her love, continuous support and encouraging smile.

Robert Salai
Trondheim, June 15, 2009

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

A user needs to have the means of assessing and altering the state of a mechanical or electrical system, in order to efficiently operate it. For this to be possible, an intuitive interface has to exist, to expose the full potential of the system to the operator. Obviously, the complexity of the user interface does depend on the capabilities of the underlying system; however, the reverse is also equally accurate. Furthermore, depending on the degree of complexity of the controllable system, a single interface may not be enough to allow an operator full access to it. Often, full access using a single interface may not be desirable either, and multiple interfaces may be used to expose a different set of functions to serve different kinds of users.

A user interface essentially consists of some sort of input device and some sort of output device. A user may manipulate the system with the means of the input device, and receive feedback through the output device. The most widespread types of input and output devices are the computer mouse and computer keyboard, and the display monitor. Certainly these are generic devices, where buttons are assigned to specific functions depending on the interface provided by the running program. For instance, in some applications the mouse may be used to simply point at graphical objects appearing on the display monitor or while in other it may be used to rotate or displace an object.

When applications try to attain a huge degree of immersion, the necessity for custom, accurate and intuitive input devices arises. Such devices, albeit possible to create, may be quite costly, and thus low-cost alternatives should be explored.

The sensation of immersion is a critical component and highly desired attribute of most virtual reality or augmented reality applications. For a high degree of immersion to be attained, the real and virtual world must be properly aligned with respect to each other. For this illusion to be maintained, accurate information about the user has to be gathered. This may include the position and orientation of the head of the user, which is the issue we will be studying.

A non-invasive method to determine the position, and orientation, of an object is through vision. Thus, the input device explored in this report is comprised of a camera, along with an easily distinguishable object in 3D space.

1.2 Background

The recovery of the position and orientation of a camera from the images it acquires has been a fundamental issue for computer visioning systems, as well as in the photogrammetry community.

This issue is commonly referred to as pose estimation in the computer vision society, while it is called space resection by photogrammetrists. Being a fundamental issue, there are a lot of mathematical approaches to solving the problem.

By analyzing the images acquired by the camera, and extracting certain features, the analyzation of these features may result in the recovery of the pose of the camera. There are approaches for multiple types of features, both for lines and points. Common for the approaches is that the correspondence between the features in the 3D world and the image has to be known. The different approaches to the pose estimation problem are presented in more detail in Chapter 3.

1.3 Contribution

In this report we will attempt to define a low-cost input device which would allow for computer interaction through motions of the operators head, possibly as part of a larger system trying to offer an immersive and intuitive interaction with the environment.

1.3.1 System Description

To perform head tracking two pieces of hardware are necessary, a camera and an optical marker. The camera is used to extract features from the scene. The optical marker provides these feature points, which in turn are used to assess the markers position and orientation in space.

As is stated in the assignment, the camera which will be used is the Nintendo Wii remote. The Wii remote is a wireless, and therefore quite mobile, device, which was initially designed to be the primary input device for the Wii game console. There is also a camera sensor on this device which is capable of some image analysis, making it a good candidate for image feature extraction. Unlike when in use with the game console, where the Wii remote is the non stationary part of the input system and is actively used for pointing and navigation, in this appliance it will be stationary. The Wii remote is further described in Section 4.1.

The optical marker is the non stationary part and it is a more custom contraption. The function of the optical marker is to provide easily detectable points of interest, allowing the system to determine its position and orientation in space. The optical marker is further described in Section 4.2.

1.3.2 Limitations

There are certain problems with this type of input device. For instance, it only works if the marker is in sight of the camera. This limits both the markers region of operation, as well as its orientation compared to the camera. Another issue is that visual measurements may contain significant amounts of noise and be subject to distortions due to use of imperfect lenses. This is especially the case for low-cost devices.

1.4 Report Outline

This thesis consists of seven chapters and is organized as follows:

- **Chapter 1** is an introduction to the assignment, and describes the motivation, introduces the problem, and describes the contribution of this report.
- **Chapter 2** is a short review of some basic mathematical tools.
- **Chapter 3** introduces the pose estimation problem for optical markers and provides an overview of the existing mathematical approaches for

solving it. The chapter also includes an overview of the systems which are similar to the one explored in this report.

- **Chapter 4** describes the hardware and various devices the system is comprised of, and introduces the design of the software implementation.
- **Chapter 5** contains the results of the various test performed on the developed system.
- **Chapter 6** provides a discussion on the hardware and software in use, as well as the obtained results. Moreover, the chapter also includes some proposal for future work and extensions to the developed system.
- **Chapter 7** concludes the report.

The report also contains six appendices:

- **Appendix A** describes how the pan-tilt unit may be interfaced, and gives an overview of the commands used to control it.
- **Appendix B** provides a description on how the Wii remote can interfaced in C# and in MATLAB, and details the functions of the libraries used in the assignment.
- **Appendix C** describes how a software library used for the head tracking is configured and interfaced.
- **Appendix D** details how the library OpenCV may be used for pose estimation for the optical marker used in this report.
- **Appendix E** includes some more detailed results which complement Chapter 5.
- **Appendix F** provides a short overview of the contents on the attached digital media.

Chapter 2

Theoretical Background

Due to the vision based nature of the input device, some basic mathematical concepts about spatial representations will be presented in this chapter.

The following topics will be covered:

- **Position and Orientation**
  The section provides a short introduction to how position and orientation can be expressed in mathematical terms.

- **Homogeneous Coordinates**
  This section introduces the concept of homogeneous coordinates and discusses their usefulness.

- **Projection**
  In this section the concept of projection is presented and described mathematically.

- **Camera Model**
  This section defines a simple camera model, which are used to describe the complete transformation of image acquisition, including position, orientation of the camera, in a unified frame.

2.1 Position and Orientation

The space a rigid body occupies may be uniquely specified by its position, orientation[1] with regards to some global frame of reference.

[1]The space a rigid body occupied does also rely on its scale and form, but here we assume that these properties are contained in description of the rigid body.

In mechanics, the set of independent parameters which describe the possible independent translations and rotations of an object, are often referred to as degrees of freedom (henceforth abbreviated with DOF), and an unrestricted object in three dimensional space has 6 DOF. Three of these are related to the position, or translation, of the object, while the rest describe the rotation of it. The position and orientation of an object is often referred to as its *pose*, and there are multiple ways to represent the pose of an object in a given coordinate system.

The position of a point in 3D space is described by its displacement along the axes of some frame of reference, which can easily be expressed using vector notation. Figure 2.1 illustrates a point at an arbitrary position in 3D space, denoted by the vector $\mathbf{t}$. This arbitrary position is easily described as the translation of the point from the origin of the coordinate system to its current position, which is given by the vector addition of the two positions:

$$\mathbf{t} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} + \begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} \quad \in \mathbb{R}^3$$

where x', y' and z' are the individual translations along their respective axes, as illustrated in Figure 2.1.

A simple change in orientation is illustrated in Figure 2.2, for the two dimensional case. Here there are two coordinate systems, where one is rotated by an angle θ compared to the other. There are a number of ways to parameterize spatial rotations: in matrices, quaternions and Euler angles, to mention a few. All of these representations have advantages and drawbacks.

2.1.1 Rotations

While translations are easily described by vector additions, rotational changes may be carried out by matrix multiplications. A general rotation matrix is defined as any matrix which acts as a rotation in Euclidean space. The general rotation matrix describing rotations in 3D space can be written as

$$\mathbf{R} = \begin{pmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{pmatrix} \in \mathbb{R}^{3\times 3} \tag{2.1}$$

which also has the added properties

- $|\mathbf{R}| = 1$

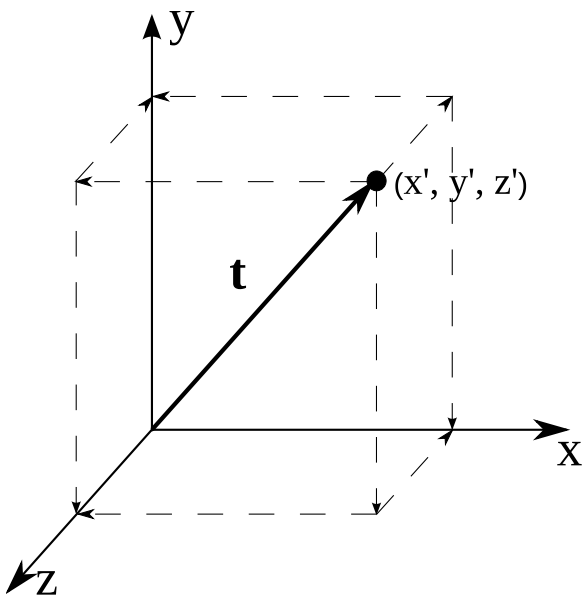


Figure 2.1: Translation in a coordinate frame. The vector $\mathbf{t}$ denotes the translation from the origin of the reference frame to the point with the coordinates (x', y', z').

- $\mathbf{R}\mathbf{R}^T = \mathbf{I}$

where $\mathbf{I}$ is the identity matrix of appropriate size.

There are other ways to represent rotations, in addition to the matrix representation, and all representations have both advantages and drawbacks. For instance, a rotation expressed as a matrix is parameterized by a total of nine parameters, with six constraints between those parameters. However, they are quite efficient for rotational transformations, as only a product between a matrix and a vector has to be calculated. But, the parameters do not immediately have any physical meaning, unlike the parameterization using Euler angles.

Euler Angle Description

Euler's Rotation Theorem states that an arbitrary rotation may be described by only three parameters. The Euler angles define the rotation about the three axes in a Cartesian coordinate frame, which describes the orientation of an object with respect to an initial frame. For instance, the rotation

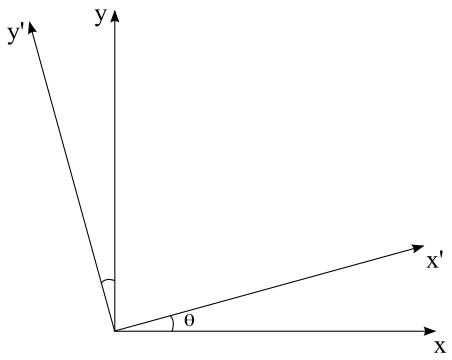


Figure 2.2: A rotation of a reference frame compare to another by θ in two dimensions.

illustrated in Figure 2.2 is a rotation about an axis[2] orthogonal to the x- and y-axis.

Each of the separate three dimensional rotations, about the x, y and z axis, can be described as three simple rotation matrices

$$\begin{aligned}
\mathbf{R}_x(\phi) &= \begin{pmatrix} 1 & 0 & 0 \\ 0 & cos\phi & sin\phi \\ 0 & -sin\phi & cos\phi \end{pmatrix} \\
\mathbf{R}_y(\theta) &= \begin{pmatrix} cos\theta & 0 & -sin\theta \\ 0 & 1 & 0 \\ sin\theta & 0 & cos\theta \end{pmatrix} \\
\mathbf{R}_z(\psi) &= \begin{pmatrix} cos\psi & sin\psi & 0 \\ -sin\psi & cos\psi & 0 \\ 0 & 0 & 1 \end{pmatrix}
\end{aligned} \tag{2.2}$$

where ϕ, θ and ψ define the rotation about the x, y and z axis, respectively.

The complete conversion of a rotation described in yaw, pitch and roll is then expressed by the product of the three simple rotations

$$\begin{aligned}
\mathbf{R}_{x,y,z}(\psi,\theta,\psi) &= \mathbf{R}_z(\phi)\mathbf{R}_y(\theta)\mathbf{R}_x(\psi) \\
&= \begin{pmatrix} c\phi c\theta & s\phi c\psi + c\phi s\theta s\psi & s\phi s\psi - c\phi s\theta c\psi \\ s\phi c\theta & c\phi c\psi - s\phi c\theta s\psi & c\phi s\psi + s\phi s\theta c\psi \\ s\theta & c\theta s\psi & c\theta c\psi \end{pmatrix}
\end{aligned} \tag{2.3}$$

[2]The axis is not illustrated in the figure.

where ϕ, θ and ψ are defined as above, and $c = cos$ and $s = sin$.

While the complete rotation is described with only three parameters, performing rotations is computationally inefficient as multiple evaluations of trigonometric functions are required; but the parameters of this representation match the three degrees of freedoms of 3D orientations, and thus have physical meaning. Another disadvantage of this representation compared to the other mentioned is that it has ambiguity problems, as in certain situations a loss of one degree of freedom may occur. When two of the three rotation axes align, one rotation has no effect. For instance, if the pitch rotation is 90° up or down, the yaw and roll describe the same motion. The effect is referred to as gimbal lock.

Unit Quaternions

Rotations in 3D space may also be parameterized as unit quaternions. A quaternion is a four-tuple which describes a three dimensional vector and the amount of rotation about this vector. This is a far more compact representation than a rotation matrix. In addition, quaternions also avoid the ambiguity issue which may occur with Euler angles, while also being far more computationally efficient at performing point rotations.

For a quaternion

$$\mathbf{q} = \begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix} \tag{2.4}$$

where a defines the scalar part, while b, c and d correspond to the vector part of the quaternion, the associated rotation matrix may be expressed as

$$\mathbf{R} = \begin{pmatrix} a^2 + b^2 - c^2 - d^2 & 2(bc - ad) & 2(bd + ac) \\ 2(bc + ad) & a^2 + c^2 - b^2 - d^2 & 2(cd - ab) \\ 2(bd - ac) & 2(cd + ab) & a^2 + d^2 - b^2 - c^2 \end{pmatrix} \tag{2.5}$$

2.2 Homogeneous Coordinates

Both rotations and translations may be expressed using a single matrix using homogeneous coordinates. Homogeneous coordinates are a useful concept in computer vision because they allow many geometric transformations to be represented uniformly.

By defining the homogeneous coordinates as $(\mathbf{t} \;\; w) = (\frac{\mathbf{t}}{w} \;\; 1)$, where $\mathbf{t}$ is a translation vector and w is a nonzero constant which we can take to

be unity, we can express both the translation and rotation, as well as other geometric transformation such as scaling, shearing and projection, neatly using a 4×4 matrix. If we only consider the unification of the rotation and the translation in this single framework, the matrix is often referred to as the homogeneous transformation matrix[3], and it may be defined as

$$\mathbf{T} = \begin{pmatrix} \mathbf{R} & \mathbf{t} \\ 0 & 1 \end{pmatrix} \in \mathbb{R}^{4\times4} \tag{2.6}$$

where $\mathbf{t} \in \mathbb{R}^3$ is the vector describing the translation, and $\mathbf{R} \in \mathbb{R}^{3\times3}$ is the orthogonal rotation matrix.

The homogeneous coordinates do not directly correspond to real coordinates, but the real coordinates can be calculated by dividing the first three parameters by the forth, in this case defined as w. Thus, a usual approach is to declare $w = 1$, which has also been done in (2.6), to eliminate the need for a division.

2.3 Projection

When the world is perceived through a viewing device, the three-dimensional space is mapped to the image plane of the camera, creating a picture which represents the complexity of the environment in two dimensions. In this process, the objects in the 3D space are projected to the image plane as a snapshot is captured.

Mathematically, projection is defined as the mapping of higher dimensions to lower dimensions. This essentially removes information, as multiple mappings may yield the same lower dimensional result, often leading to the case where the higher dimensional information cannot be recovered. In the sense of computer vision, we are defining projection as the mapping of 3D points to a 2D plane.

There are multiple ways to visualize objects and information. For instance, engineering drawings generally visualize an object in three views; a top down, side and front view. These views make it easy to illustrate features and sizes of objects, making them also popular in modeling software. These views are examples of simple orthographic projections, and can be rendered by projecting all the points on the object along parallel lines which are orthogonal to the projection plane. There are other ways of representing 3D images on a flat plane. For instance, Figure 2.1 is an example on

[3]See Chapter 6.4.6 in [12] for more on the concept of homogeneous transformation matrices.

the projection method called oblique projection, or more specifically, Cavalier projection. Here, two of the axes are perpendicular to each other, while the third is draw in diagonal, making arbitrary angles to the other axes.

The objects we perceive in the real world, either through our eyes or through an imaging device, however, are not projected along parallel lines, but rather converge at the lens of the imaging device. This type of projection is called perspective projection and images formed this way are subject to a change of scale, as well as certain distortions. Unfortunately, perspective projections have the disadvantage of breaking down simple relations between feature points. For instance, parallel lines do no longer appear parallel, and similar objects may differ in scale, complicating the recovery of higher dimensional information.

2.3.1 The Perspective Transform

The equations describing the projection for an arbitrary vector $\mathbf{t}_o = (x_o, y_o, z_o)^T \in \mathbb{R}^3$ onto the image plane described by $z = f$ are

$$\begin{aligned} x_c &= f\frac{x_o}{z_o} \\ y_c &= f\frac{y_o}{z_o} \\ z_c &= f \end{aligned} \tag{2.7}$$

where f is the focal length of the camera, and $\mathbf{t}_i = (x_i,\ y_i,\ z_i)^T \in \mathbb{R}^3$ denote the coordinates of the projected vector $\mathbf{t}_o$ on the image plane. Both vectors, $\mathbf{t}_o$ and $\mathbf{t}_i$, are stated in the same frame of reference. These equations are nonlinear, but can be made linear by stating them in homogeneous coordinates as

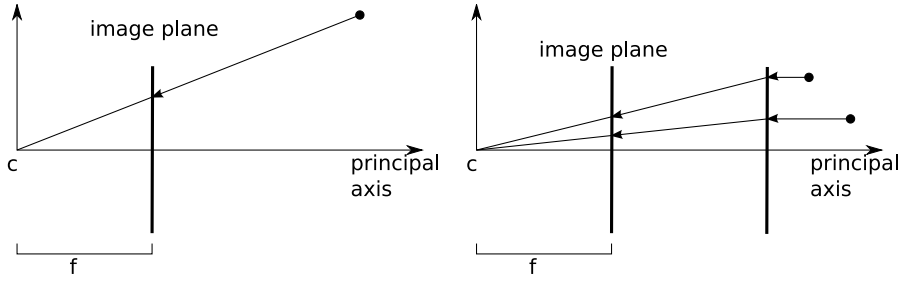
$$\begin{pmatrix} x_i \\ y_i \\ z_i \\ 1 \end{pmatrix} = \begin{pmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & f & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} x_o \\ y_o \\ z_o \\ 1 \end{pmatrix} = \mathbf{P}\mathbf{t}_o \tag{2.8}$$

where $\mathbf{t}_o = (x_o,\ y_o,\ z_o,\ 1)$ is the vector denoting the position of point on an object and $\mathbf{P}$ is the projection matrix defined as

$$\mathbf{P} = \begin{pmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & f & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \tag{2.9}$$

An Approximation to the Perspective Transform

A rigid object, depending on its representation, may be described by an infinite number of feature points, where each point may be projected to the image plane according to (2.8). However, recovering the depths of all the independent feature points is a challenge, especially if the object is sufficiently far away from the camera. In this case, the projection rays are nearly parallel, as the depth differences within the object are negligible compared to the distance to the camera. Thus, the whole object can be assumed to be at the same distance from the camera. This essentially represents an initial orthographic projection to a plane located at the assumed object depth, followed by a perspective projection to the image plane. This is illustrated in Figure 2.3(b) for two points with differing depths compared to the center of projection. This projection is often referred to as weak perspective projection, or scaled orthographic projection. The full perspective projection is provided for comparison in Figure 2.3(a), where a single point is projected onto the image plane.



(a) Full perspective projection (b) Scaled orthographic projection

Figure 2.3: Fig. 2.3(a) illustrates the full perspective projection of a single point being projected to the image plane. Fig. 2.3(b) illustrates the case of weak perspective projection, where two points are first projected to a common plane along parallel lines, followed by a full perspective projection from that plane. **c** indicates the center of projection, while f indicates the length to the image plane. For a camera, this is usually called the focal length.

2.4 Camera Model

A pinhole camera is a very simple camera, comprised of a light-proof box with a small hole through which light enters. Facing the hole is a sheet of photosensitive material, allowing the forming of an image due to the light entering through the hole. The simplest mathematical model which describes the forming of the image is based on an ideal pinhole camera model. This assumes that the focus point, i.e. the pinhole, is infinitesimally small, and the image plane[4] is perfectly flat, and perpendicular to the principal axis of the camera. Figure 2.4 is an illustration of the model, where the principal axis is parallel to the z-axis. Furthermore, the image plane is placed in front of the focus point, which simplifies the notion of the perspective projection, as the projection is not inverted.

If the origin of the image plane frame of reference does not coincide with where the z-axis of the camera reference frame intersects the image plane, a translation to the principal point has to be defined.

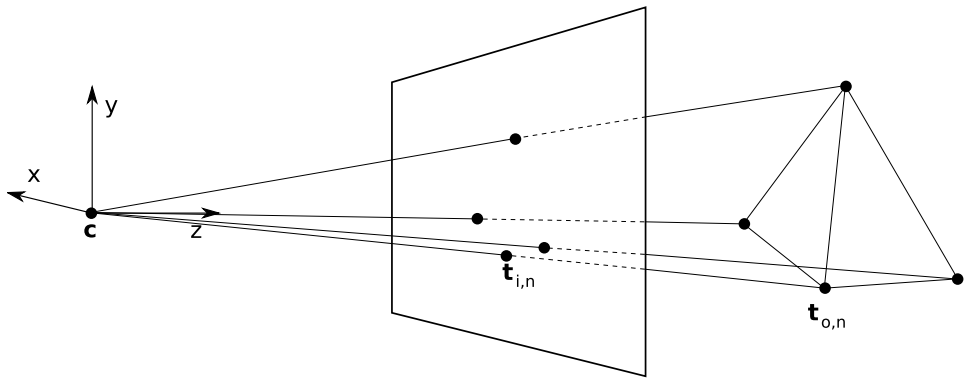


Figure 2.4: The pinhole camera model, where $\mathbf{c}$ denotes the center of projection. $\mathbf{t}_{o,n}$ denotes the n edges of an object, while $\mathbf{t}_{i,n}$ describes the projected position of the n edges in the image plane.

Finally, in order to define the camera in 3D space the position and orientation of the camera compared to a global reference frame has to be defined.

[4] In the physical construction of the pinhole camera, the image plane is represented by the sheet of photosensitive material.

The complete transformation for the camera is hence

$$\begin{aligned}\mathbf{G} &= \mathbf{P}\ \mathbf{T}_c\mathbf{T}_g \\ &= \begin{pmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & f & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & x_c \\ 0 & 1 & 0 & y_c \\ 0 & 0 & 1 & z_c \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} r_{11} & r_{12} & r_{13} & x_g \\ r_{21} & r_{22} & r_{23} & y_g \\ r_{31} & r_{32} & r_{33} & z_g \\ 0 & 0 & 0 & 1 \end{pmatrix}\end{aligned} \tag{2.10}$$

where $\mathbf{P}$ is the projection matrix, $\mathbf{T}_c$ is the transformation matrix which moves the center of projection of the camera to the correct position and $\mathbf{T}_g$ describes the cameras position and orientation in 3D space.

Usually, the parameters which describe the geometric and optical properties of the camera, the intrinsic parameters, and position and orientation, the extrinsic parameters, are grouped together. In this case, $\mathbf{PT}_c$ are the intrinsic parameters, while $\mathbf{T}_g$ are the extrinsic.

Equation (2.10) has all in all nine parameters, where the six DOFs defining the cameras position and orientation in 3D space are the extrinsic. The three intrinsic parameters are the focal length, and the position of principal point in the image plane. This however, assumes that the sensor is known to be Euclidean to a high degree of accuracy. In general, no physical imaging device is ideal, and thus the pinhole camera model does not suffice whenever accuracy is of great importance. Potentially, the camera model should also account for scaling errors, correcting translation errors due to the misalignment of the sensor, correcting sensor skewing errors due to non-orthogonality of the sensor axes, and shearing errors due to unequal scaling along the sensor axes and lens distortions.

Chapter 3

Literature Survey

In this chapter we provide an overview of the various approaches for pose estimation and camera control, as well as the existing systems which are similar to the head tracking system being explored in this report. Furthermore, an overview of the existing software libraries which are used to utilize the specific hardware supplied in this assignment is also provided. The chapter will focus on approaches which are appliable to the provided components, and due to the function of the Wii remote will mainly consider pose estimation techniques using points as features. The Wii remote is discussed in Chapter 4.

The chapter is organized as follows:

- **Perspective n-Point Problem**
  In this section an approach for the pose estimation problem which is applicable to the provided hardware is described. This particular approach is coined the Perspective n-Point problem.

- **Pose Estimation in Research**
  In this section a short overview of the various approaches for estimating the extrinsic parameters of a camera is provided.

- **Software**
  This section presents the various software which may be used to implement the system discussed in this report.

- **Similar Systems**
  This section presents an overview of systems which are similar to the head tracking system explored in this report.

- **Camera Control**
  In this section a short overview of the techniques which have previously been used for camera control is presented.

3.1 Perspective *n*-Point Problem

The perspective n-point (henceforth abbreviated as PnP) problem is the problem of finding the pose of a camera by analyzing the correspondence of n feature points in an image to their positions in the scene. By accurately knowing the geometric distribution of the n points in advance, it is possible to calibrate the exterior parameters of the camera, which define the pose of the camera.

The PnP problem was first formally defined by Fischler and Bolles in [13]. They were aiming to solve the problem of determining the position of a camera by analyzing an image with a set of landmarks with known locations. They also presented analytic solutions to the P2P and P3P problems as well as the coplanar case of the P4P problem, and propose an algorithm for estimating the pose of the camera.

Davis [10] presents more information about the PnP problem, and other general problems regarding pose recovery and perspective projection. Davis also studies the number of solutions possible for the PnP problem, for different values of n, both in case of weak perspective projection and full perspective projection. According to Davis, as presented in Table 3.1, the number of solutions to the PnP problem becomes finite if there are at least three feature points, as long as the points are not in a critical configuration[1].

Table 3.1 shows the number of ambiguous solutions for both the coplanar configuration of points, where all the points are in the same plane, and non-coplanar configurations. Obviously, when working with greater than three feature points, they will always be coplanar. According to Davis, for the case of full perspective projection, four coplanar points are enough to determine a unique pose, while for the non-coplanar configuration, six points are enough. It is also noteworthy that some ambiguous poses are not relevant, or simply present impossible cases, and can safely be discarded. Section 3.2 presents some approaches to the solution of the pose estimation problems.

[1] A critical configuration is a configuration of points where the additional points only provide redundant information. For instance, when all the feature points are in a collinear configuration, there is no additional information in the setup if the number of points exceed two.

Table 3.1: Number of solutions for n feature points for the full and weak perspective projections

Arrangement of the Points	n	WPP	FPP
Coplanar	≤ 2	∞	∞
	3	2	4
	≥ 4	2	1
Non-coplanar	4	1	2
	5	1	2
	≥ 6	1	1

3.2 Pose Estimation in Research

Horaud, Conio and Leboulleux [16] provide an analytic solution to the P4P problem by solving a bi-quadratic polynomial equation in one unknown. They examine the case where the feature points are non-coplanar. They claim that non-coplanar points allow for a more stable solution to the problem compared to the coplanar case, as the solution is not sensitive to the relative orientation of the image plane with respect to the scene containing the points. Also, the computation of the solution is fast, and is therefore suitable for real time applications. They also explore special cases, for instance when the four points are coplanar, or the configuration form three co-linear points, and show that these accidental configurations also provide useful solutions.

DeMenthon and Davis [11] propose a method, called POSIT, for finding the pose of an object from a single image by analyzing four or more non-coplanar feature points. This method exploits the fact that perspective projection can be approximated by scaled orthographic projection for certain situations. This is an iterative method, which does not need a good initial condition. Initially, only a scaling is guessed, but the guess does not need to be accurate. The object pose is calculated through the following iteration steps:

1. Compute an approximate pose based on scaled orthographic projection.

2. Deform the object by shifting the object points from their approximated positions to positions at the same depth but on their lines of sight with regards to the camera.

3. Find the image of these shifted points by a scaled orthographic pro-

jection model.

4. If the scaled orthographic image points are not the same as those found at the previous iteration, go back to Step 1 using these image points instead of the original ones.

5. Else, the exact pose is the last approximate pose

Clearly, being an iterative approach, divergence is an issue. A solution for this is not stated in this paper, but implementations of this algorithm do have upper bounds on the number of allowed iterations. The paper also states that this algorithm is computationally cheap, only requiring around $24n$ arithmetic operations and two square root calculations per iterations, for n feature points. However, they state that the case of four feature points is a critical case, and the estimates may be less accurate than what is desirable. In addition to being fast and not needing an initial pose estimate, the algorithm is easy to implement.

Oberkampf, DeMenthon and Davis extend the POSIT algorithm to handle coplanar points in [27]. They also further explore the ambiguities that arise for the P4P-problem with the points in coplanar configuration. In the case when scaled orthography projection is used for approximating the pose based on four coplanar feature points, there are always two possible solutions. This is because there is no way to distinguish between the actual pose and a mirror about a plane parallel to the image plane. They try to remove unacceptable poses in the iterating process, however a unique solution is never guaranteed. They also study the noise sensitivity of the algorithm, and it is considerably less sensitive to noise in the position estimates compared to the rotational estimates. All in all, the algorithm performs well when more than three points are used, both in favorable and unfavorable configurations.

Lee, Park and Sung [22] describe a stable real-time marker-based camera tracking method for augmented reality systems. They proposed an iterative linear camera match-moving algorithm, where no initial estimates are required. The proposed algorithm works with both coplanar and non-coplanar markers, as long as the points are not co-linear. The algorithm is reported to have good stability and accuracy.

Ansar and Daniilidis [5] developed a fast pose estimation algorithm which is based on depth recovery. They assume a calibrated camera with well known intrinsic parameters is used, and the proposed approach works with either points or lines as tracking features. The emphasis of the paper is on attaining speed and accurate pose estimation with limited numbers of

features points or lines. They guarantee a correct solution in the noiseless case, provided that the feature points are in a configuration where there are no ambiguities. Furthermore, the performance of the point algorithm is superior to other linear algorithms, and comparable to the recent iterative approaches available at that time. In terms of the algorithm using line features, there are no other linear algorithms, and the performance of this algorithm is comparable to previous iterative approaches.

Quan and Lan [30] developed a family of linear, unique solution algorithm for 4-, 5- and n-feature point camera pose estimation. This general algorithm also works for both coplanar and non-coplanar points, and performs almost as well as several other special linear four point algorithms in coplanar configurations. However, it outperforms the special algorithm whenever the points are in quasi-coplanar configuration, allowing the algorithm to perform satisfactory with noisy sets.

Schweighofer and Pinz [32] propose a way to improve on existing iterative pose estimation methods. They explore the ambiguities that arise when estimating camera pose, and propose system to more robustly reject multiple plausible estimates for the case of planar targets.

Lippiello, Siciliano and Villani [24] shows how to use an adaptive extended Kalman filter to estimate position, orientation and the velocities of these for an arbitrary object based on some preselected feature points. The objects are matched to model descriptions created in CAD, and this process is further described in Lippiello and Villani [25]. They describe how to filter and estimate positions and rotations, and their velocities using the extended Kalman filter and the adaptive extended Kalman filter. They also fully explore the prediction capabilities of the Kalman filters, which allows for better tracking. They then compare the performance of the filters, and show that the adaptive filter, when correctly configured, holds a performance advantage over the non-adaptive formulation. The advantage comes at a slight increase in computational cost, making it worth the investment.

Rickard and Davis [31] have recently published a paper where they use the technology present on a Nintendo Wii remote to track an artifact to determine its pose. The artifact has six infrared light emitters attached, as well as a Wii remote which provides an accelerometer. This artifact is observed by two Wii remotes, tracking the IR emitters on the artifact. They use the approach proposed by Kreylos to the tracking of the emitters, which is briefly described in Section 3.4.4. They solve the correspondence problem of the points, observed by the two sensors, by using a guess-and-check approach, which, by their account, results in a correct mapping less than 40% of the time. However, since they try to maintain the discovered

correspondences between points over multiple frames, the system rapidly recognizes incorrect mappings as they become inconsistent when the artifact rotates.

Wimmer et al. [33] use the tracking algorithms implemented in the software library ARToolkit to determine the pose of the Wii remotes camera. ARToolkit can both perform the detection, and tracking, of a square. The detection of the square is not in use here, since the Wii Remote performs this task. With the utilization of four IR light emitters, they create a square, and let ARToolkit determine the pose. Furthermore, they let one of the light emitters blink, to allow it to be distinguished from the rest. This way they resolve the pose ambiguities which arise when attempting to estimate the pose from a square marker. However, as not all LEDs are emitting at all time, multiple frames must be compared, which lowers the sampling rate of tracking system. They claim to achieve a detection ratio of a little more than 10Hz.

3.3 Software

There are a few software libraries in areas such as computer vision, computer graphics, augmented reality and tracking which may perform desirable tasks for the system discussed in this report. In this section we present a short overview of these libraries, and some which may be used to interface the provided hardware.

3.3.1 Computer Vision Software

Computer vision specifies the field which seeks to allow machines to see. Thus, a software library specializing in computer vision usually include algorithms for image processing, object recognition, event detection, motion estimation, pose estimation and more. For a head tracking application such as the one discussed in this report, some of these are relevant issues, and an appropriate library may prove useful. Therefore, a short introduction to a few of them is appropriate.

OpenCV

OpenCV [2] is an open source, cross-platform computer vision library, and initially was an Intel Corporation creation. It was officially launched in 1999, and reached the milestone version 1.0 in 2006. In 2008 it obtained commercial support from Willow Garage, and is still under active develop-

ment. Recently a pre-release to version 1.1 was released. It is mainly written in C, however there are wrappers for other languages available, most notably the EmguCV C# wrapper which is used in this report. The main focus of the library is in image analysis, motion tracking, pose estimation, camera calibration, while also providing several functions for use in robotics and artificial intelligence. The pose estimation implemented in OpenCV is the POSIT algorithm.

ARToolkit

ARToolkit [19] is another open source computer vision library, specifically designed for creating augmented reality applications. In augmented reality applications virtual objects are overlaid on the real world, and to achieve this, ARToolkit provides camera viewport calibration and marker tracking for real-time systems. This library has been used for tracking purposes in [33].

FreeTrack

FreeTrack [1] is a general purpose optical motion tracking library, which is able to track and handle up to six degrees of freedom. FreeTrack is compatible with most web cameras and as such performs image analysis to extract feature points from images. It also supports the use of the Wii remote for feature extraction. This is a Microsoft Windows only library, released under the GNU GPL license. The pose estimation for four points is achieved using an implementation of the POSIT algorithm.

3.3.2 Additional Software

In this section, the software which may be used for communicating with the hardware, or is used by some of the systems described later, is presented.

WiimoteLib

WiimoteLib [28] is a .NET managed library for accessing the Nintendo Wii remote and the extensions from a .NET application. It is easy to use, and has a comprehensive set of functions. The library supports all of the successfully reverse engineered Wii remote features, including the buttons, accelerometer, infrared camera, rumble and LEDs, with the exception of the speaker. It also supports all the Wii remote accessories, as well as the use of multiple Wii remotes. The usage of this library is further described in Appendix B.

WiiLAB

WiiLAB [6] is a MATLAB compatible wrapper to the WiimoteLib library, as well as a set of functions which facilitates the usage of the Wii remote in the MATLAB environment. It has been developed at the University of Notre Dame for educational purposes, allowing the Wii remote to be easily used in teaching. The wrapper provides access and usage of all the functions implemented in WiimoteLib, and thus has great support for the Wii remote and its extensions.

XNA

Microsoft XNA [7] is a toolset that facilitates computer game development for both Microsoft Windows and the Xbox 360. It is primarily targeted at students, hobbyist and independent developers. The library is written in managed C#. The underlying technology is based on Microsoft DirectX, which is another graphics library. XNA also provides an extensive math library, and specifically the support for homogeneous matrices, vectors, quaternions and trigonometric functions is of importance here. The XNA framework is utilized in the virtual reality library Tgex, while the math portion of the library is utilized in the tracking library Tvrx, more closely described in section 3.4.3.

3.4 Similar Systems

In this section, some of the available systems, which are similar to the one explored in this report are examined. These systems either use the same hardware as is provided in this report, or have similar functionality.

3.4.1 WiiDesktopVR

WiiDesktopVR, written by Lee [23], was the first application to perform simple head tracking using the Wii remote. The WiiDesktopVR application includes two demonstrations, showing how a simple head tracking scheme can actually provide quite a lot to visualization applications or games in terms of realism. The first demo is a simple room with targets placed at different depths, and as the observer moves around and his or hers view changes, the room is rendered as if the observer is actually altering the position of the camera. Figure 3.1 shows an image captured from this application. The other demo is essentially a virtual window, as it renders

football stadium, and shows more or less of it, depending on the position of the observer compared to the camera. Figure 3.2 is a screenshot of this demo.

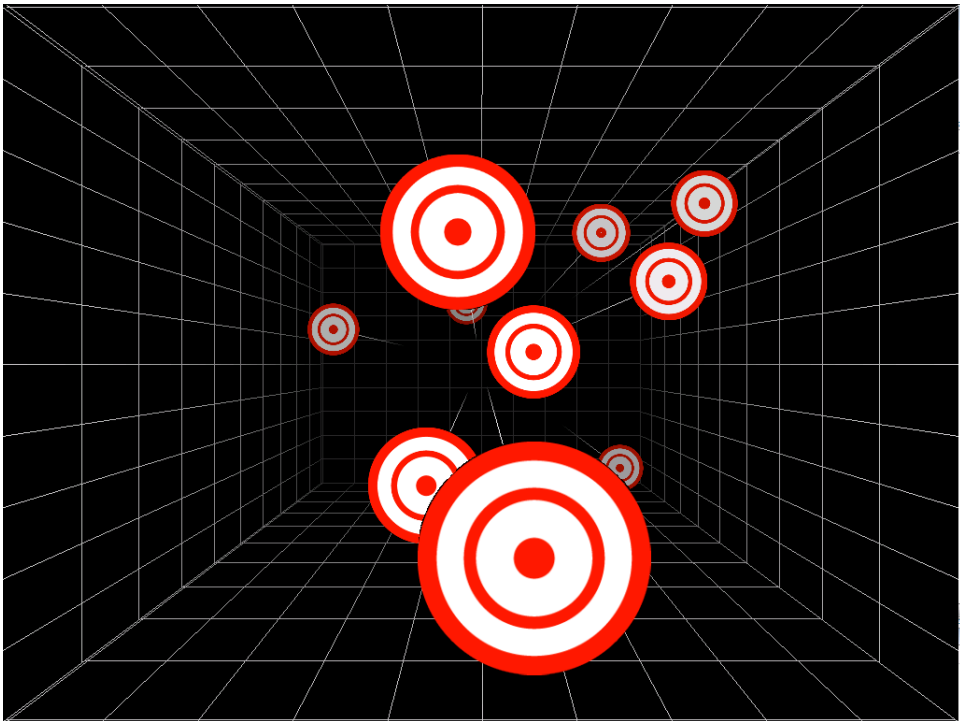


Figure 3.1: Rendering of a virtual room with targets at random positions. With the use of head tracking, the application alters the virtual room and the position of the targets in order to make the virtual room feel real.

In the setup described by Lee, two feature points are used to estimate 3 DOF, being the three translatory motions. The rotations are not calculated[2], or used, and as such some assumptions have been made on the viewing direction.

There are also quite a few ambiguities in this setup, as Table 3.1 clearly suggests. For instance, there is no way to actually know which way is up, and therefore no way to detect if the beacon is upside down. However, for

[2]Although no rotational parameters have been calculated in the WiiDesktopVR applications, with two points alone, the roll may also be calculated. However, one must still assume an "up" direction. Consequently, 4 DOF tracking is possible, albeit not uniquely.

Figure 3.2: Rendering of a football stadium as if it was observed through a window. Using head tracking, the application is able to assess the position of the observer. As the observer moves around, the image changes accordingly, creating an illusion of a real window.

a very limited situation, this may still be enough.

WiiDesktopVR is used in conjunction with WiimoteLib, which is used to communicate with the Wii remote, while Microsoft's DirectX library is used to present the virtual environment. This system is a proof-of-concept, and therefore there is little information on its operating ranges, however, it is clearly limited by the Wii remotes specifications. The IR emitters which are used by Lee are Vishay TSAL6400 diodes, which are further discussed in Section 4.2.

Estimating 3 DOF

For this particular implementation we need to know the distance between the two IR emitters, and the resolution of the camera. Assume that the

distance is defined in millimeters as d, while the horizontal and vertical resolutions are defined as R_h and R_v, respectively. In this case, the 3 DOFs may be approximated as follows. First, the radians each pixel covers have to be calculated. Lee has assumed that the Wii remote has 45° horizontal field of view, and as such the angular span of each pixel can be calculated as

$$\alpha = \frac{\pi}{4}\frac{1}{R_h} \tag{3.1}$$

where α is the angular span per pixel in radians, and R_h is the horizontal resolution of the device.

Assume $p_1 = (x_1, y_1)$, $p_2 = (x_2, y_2) \in \mathbb{R}^2$ denotes the positions of the detected IR emitters in the image plane, while $p_m = (x_m, y_m) \in \mathbb{R}^2$ denotes the position of the midpoint between the emitters. The angle between these points can be found as

$$\beta = \frac{|p_1 - p_2|\alpha}{2} \tag{3.2}$$

where β is angle between these points, and α is given by (3.1). The distance from the camera to the IR emitters is given by

$$z_{ch} = \frac{d}{2tan(\beta)} \tag{3.3}$$

where z_{ch} denotes the distance from the camera to the observers head, and β is given by (3.2).

Now, the remaining distances, the translations horizontally and vertically, can be found as

$$\begin{aligned} x_{ch} &= sin(\alpha_p z_{ch}(x_m - \frac{R_h}{2})) \\ y_{ch} &= sin(\alpha_p z_{ch}(y_m - \frac{R_v}{2})) \end{aligned} \tag{3.4}$$

where z_{ch} is given by (3.3), R_h and R_v are the horizontal and vertical resolution of the camera, and x_{ch} is the horizontal distance from the center of the camera to the observers head, while y_{ch} is the vertical component.

Using these equations to convert the measurements from the input device to be used in the system, allows us to describe certain actions for the detected motions. In the case of WiiDesktopVR, the measurements have been used to define a camera at the estimated position, looking at the origin of the defined reference frame. Using the equations provided above, this origin is situated at the center of projection of the physical camera. However, when using such a device for displaying a virtual environment, it is clearly impractical

requiring it to be placed at the center of the monitor. The offsets required to account for this have not been included in (3.1) - (3.4).

3.4.2 TrackIR

Natural Point [29] is a company that creates optical tracking systems both for the industrial and consumer use. While having a range of products, the system which resembles the system explored in this report the most is TrackIR, both in terms of functionality and cost. This system is primarily aimed at the consumer market, and specifically games, however it has been in use in research as well, i.e. in [21]. Natural Point provides a software development kit, to allow for quick integration with third party applications. This system is currently in its fifth revision.

TrackIR consists of a high frame rate infrared sensitive camera, and a marker which can be mounted on an operators head. The camera has a refresh rate of 120 Hz, a raw sensor resolution of 640×480 pixels and 51.7° field of view. The device reports position at $\frac{1}{150}$ pixel precision, which is a total reporting resolution of 96000×72000. The camera used in TrackIR is illustrated in Figure 3.3.

There are two markers available, one equipped with three reflective spots, which reflect the IR light emitted from LEDs mounted on the camera, while the other is equipped with three IR LEDs mounted in a special configuration.

As this device is made for use in games, its operating range is from 61 cm to 152 cm. Furthermore, Natural Point claims that this system can uniquely determine six degrees of freedom accurately, even with only three feature points.

3.4.3 Tvrx and Tgex

Fleisch [14] has created two software libraries for use in the creation of virtual reality systems. These have been made available at `www.vrhome.de`. One of the libraries, Tgex, manages rendering for stereo vision, while the other, Tvrx, manages six degrees of freedom head tracking for multiple imaging devices. While the library gains support for the Wii remote through WiimoteLib, support for some other imaging devices is also implemented. The pose estimation algorithm used in the tracking library is the POSIT algorithm. More specifically it is the implementation of the POSIT algorithm in OpenCV. OpenCV is employed for some filtering too, in addition to the tracking.

The optical marker is a custom creation consisting of four infrared emit-

Figure 3.3: The TrackIR camera. The image is gathered from [29].

ting diodes. Moreover, the library requires the infrared emitting diodes on the beacon to be in a special configuration. Three of the diodes have be aligned, while the full four have to be in a non coplanar configuration as per the requirement of the POSIT algorithm. Finally for the pose estimation to work properly, the software has to be accurately configured with the geometrical layout of the infrared emitting diodes. The support for multiple Wii remotes allows for tracking of multiple beacons, and thus the software has to be configured with which Wii remote is to be used for head tracking, and which is used for other tasks. The configuration and use of the tracking library is described in Appendix C. The optical marker portrayed in chapter 4.2 is based upon the design proposed by Fleisch.

3.4.4 Oliver Kreylos

Kreylos [20] has a website where he writes about his various endeavors regarding virtual reality. He has also investigated how the Wii remote may perform in a virtual reality context, especially for use in 6 DOF tracking. Here he describes how to design a custom marker, in addition to providing a software solution performing 6 DOF tracking. He also provides the source code for the software, although only for GNU/Linux operating system. The

system consists of a stationary beacon with infrared emitting diodes positioned in the edges of a tetrahedron, and a moving Wii remote as the imaging device. This is to also facilitate the information from the accelerometers on the Wii remote. The system seams promising, and has already seen its use by others, i.e. in [31].

3.5 Camera Control

In virtual reality applications, goggles with small displays instead of lenses are often used for visualization purposes, allowing an operator to move his or her head in any direction without losing sight of the displays. This is a preferable setup, as the operator will be not be restricted by not being able to see the display. Tracking the operators head immediately makes sense in this configuration, as the virtual camera may be made to accurately mimic the viewpoint of the operator. However, when one is restricted to staying in front of a screen, the camera control is not quite as trivial.

In that case, the operator has a very limited area in which to move, while still being able to observe the virtual world. And even if the software, or hardware, has a capability of turning 360°, the operator could not exercise this possibility.

Natural Point showed an example on how this can be solved, and although the example was with a virtual camera, it should be adaptable with hardware systems as well. By using an amplification variable, which is a function of the rotation of the IR marker, to amplify the rotation of the virtual camera, one could dampen unwanted[3] orientation changes, as well as being able to reach rear viewing camera angles while still maintaining sight of the display.

Another way could be to allow the operator to choose when to move the camera, for instance by the push of a button. This would be a "mouse" approach to camera control, as we can essentially pick up the computer mouse when we need to reposition it. The push of a button is here equivalent to picking up the mouse.

The IR marker could also be used as a joystick. By simply tilting it in a direction, the camera could be made to move in that direction. A similar approach is examined by Goh et al. [15], where they explore how the Wii remote can be used to simulate joysticks. They utilize the system and devise procedures for the control of a pan-tilt-zoom camera. They report that the

[3]Unwanted orientation changes are small motions of the head which probably are unintended.

Wii remote adds a lot in terms of mobility, compared to a mouse or a joystick interface, as it already is a wireless device, and, depending on the implementation, can provide a more intuitive control mechanism than the other devices. They both explored the use of the motion sensor, as well as the Wii remotes pointing capabilities for the control mechanisms.

Although there are a lot of examples on camera control of pan-tilt platforms in motion tracking applications, i.e. the system presented by Murray and Basu [26] and the system presented by Daniilidis et al. [9], these however, focus on the pursuit of a detected object. As the Wii remote already performs the object detection, such systems may be realized using a single moving optical marker if the Wii remote is used as the imaging device in those systems. Obviously, doing this with a regular camera has the additional feature of actually being able to capture the moving target in addition to following it. This adds to the value of the camera system in that it keeps the target in view for longer periods of time compared to stationary camera systems. This is not the case with the Wii remote, unless it is used in conjunction with other imaging systems, as the images it captures cannot be accessed.

Chapter 4

System Description

In this chapter the hardware which will be used to create the head tracking system, as well as the device which is going to be controlled using the head tracking system, will be presented.

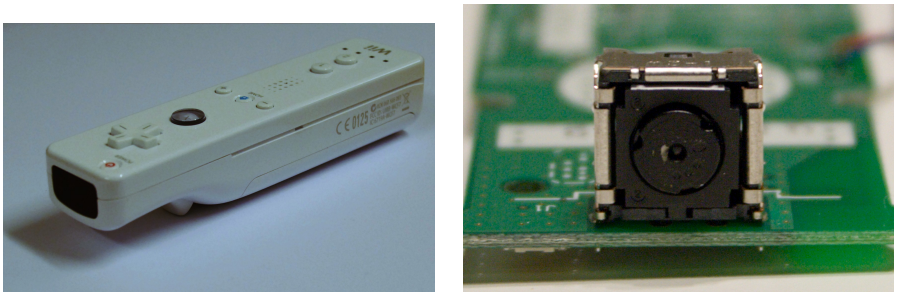
We make a distinction between the input and the output devices of the system. The input device consists of a sensor performing measurements, and a device we are able to interact with. The measurements will be conducted using the Wii remote, which has been provided in the assignment, while the interaction device is a marker with infrared emitting diodes mounted on it. The marker is not provided in the assignment and has to be designed to function properly in conjunction with the chosen software solution, and the provided hardware. The output device, which is the device we will attempt to control, is a pan-tilt unit. This unit has two degrees of freedom, and has been provided in the assignment.

The chapter is organized in the following sections:

- **Nintendo Wii Remote**
  In this section the Wii remote is presented, and its functions are described.
- **Tracking Beacon**
  In this section the configuration of the tracking beacon is presented.
- **Pan-Tilt Unit**
  In this section the device which is going to be controlled is described.
- **System Design**
  This section presents the overall design of the system, and discusses its implementation.

4.1 Nintendo Wii Remote

The sensor which is going to be used for the head tracking system explored in this report is the infrared camera in a Nintendo Wii remote. The Wii remote was initially intended as a gamepad for use with the Nintendo Wii game console, where it allows for interaction with the system. The design of the device bears a resemblance to classical TV remotes, which can be seen in Figure 4.1(a). It has motion sensing capabilities through a three-axis accelerometer, and is equipped with a high framerate monochrome camera which is used to provide pointing and aiming capabilities for the Wii game console. The connection to and communication with the Wii remote is performed wirelessly through Bluetooth.



(a) The Wii remote (b) The Wii remote camera

Figure 4.1: The Nintendo Wii remote and its monochrome camera. The image of the camera is from [3].

The official information about the inner workings of the Wii remote is scarce, and therefore, its actual capabilities are mostly known through reverse engineering and guesswork. However, as this device has caught the attention of a lot of people, a lot of information has been presented online, where [4, 3] are notable sources. The contributors of these websites have discovered how this device operates and made the initial steps of communicating with it, paving the way for the creation of drivers and libraries allowing custom application to utilize the hardware. This has generated extensive interest in research communities as well, and although at first intended for use in conjunction with games, its capabilities have been explored for multiple research projects, i.e. in [23, 20, 14, 31, 21].

For the system explored in this report, the most interesting part of the Wii remote is the infrared imaging sensor. The imaging sensor is paired with hardware capable of performing the image analysis technique called

blob detection. The imaging sensor is created by PixArt Imaging Inc. [18], however the exact product specifications are not published, and therefore only known through actual usage. The device is able to detect up to four blobs, and is able to determine the size and position of them in the image plane. It is believed that the imaging sensor has a resolution of 128×96 pixels, while the Wii remote outputs data at $\frac{1}{8}$ pixel precision, yielding a total a resolution of 1024×768 pixels. This does not appear to be simply upscaled, so the System-on-a-Chip most likely has a filter implemented in hardware as well. Some other undocument properties of the camera are its focal length and principal point. Several reports [14, 20, 23] indicate that the focal length is in the range of approximately 1280 to 1380 units. The focal length $f = 1380$ and the center of projection $\mathbf{c} = (512, 384)$ seemed to work reasonably well in some situations, and is used throughout the work presented in this report.

As previously mentioned, the Wii game console uses this device for navigational purposes, for instance to allow a user to point or aim on the screen. For this to be possible Nintendo has created an additional device, called the Wii Sensor Bar. This device is placed on top of, or below, the screen which is to be used with the game console. The Sensor Bar has two clusters of infrared light emitters which allow the Wii remote to capture their position. Based on this, the game console is able to determine where a user is pointing on the screen. In this setup, the Sensor Bar is stationary part of the input device, while the Wii remote is mobile.

While the Wii remote is typically interfaced wirelessly using Bluetooth, there have been some reports, i.e. [4], suggesting that the imaging sensor is communicating with the rest of the device using I2C. The reports suggest that the imaging sensor runs at 200Hz, which is the maximum speed it has been accessed using I2C. However, when it is interfaced wirelessly, the device operates at 100Hz. To use the I2C bus however, means that the Wii remote has to be disassembled, and the Bluetooth module has to be bypassed, which will not be performed in this work.

4.1.1 Hardware limitations

The implementation of the image analysis in hardware in the Wii remote has both its advantages and drawbacks. A hardware implementation is fast and robust, but not particularly flexible. It does not strain the rest of the system, neither the wireless connection nor the CPU, as there is no need to first transfer a captured image in order to analyze it to find the position of the IR emitters. However, as the image cannot be accessed, there is no

easy way to distinguish the IR sources in software, or even to determine if a particular source is a desirable one. As there is a limit on how many points are detected, in situations where more than four sources are detectable, we have no control over which are actually detected. This means that faulty detection may cause severe issues when trying to accurately determine the pose of the marker.

As mentioned, there is no way to uniquely determine which point is which, as the points are reported rather randomly. This is also a problem, as most pose estimation algorithms require each feature point to be distinguishable from the rest, and uniquely identifiable.

4.1.2 Point Detection

The point detection in the Wii remote is implemented in hardware, and there is no information on how this actually works. However, while performing tests and experiments, some of the features due to the particular implementation of the detection algorithm were observed.

The two cases of an IR source entering and leaving the scene are handled slightly differently. When a source is entering the scene, it is not detected until the entire source is perceived as inside the image. A leaving source however, continues being detected until it has completely left the image. Consequently, an entering source will not be reported at the sensor boundary, and will suddenly appear inside the image. Depending on the intensity of and distance to the source, and therefore the detected size, this may prove an issue. Figure 4.2 and 4.3 illustrate both these situations.

Furthermore, instead of reporting the position of the center of the detected point, it seems that the upper left position of the bounding box enclosing a point is reported. This can easily be observed for a source leaving the scene. For instance, if the point is traversing the scene in a horizontal fashion, the last detected position stays at either 0 or 1016[1] for a while, until the emitter has completely left the scene. Clearly, whenever a point is detected, and a square bounding box is placed around it, the bounding box may never leave the image, nor change its size.

When using the Wii remote as the detector for the optical feature points for pose estimation, a couple more issues have to be addressed. Firstly, any number of detected points may be detected in any order. For four feature

[1] As the upper left position of a bounding box denotes the position of a point, the largest value of pixels which is reported depends on the size of the bounding box. While $(0, 1016)$ are the last positions reported in the horizontal case, the boundaries seem to be at $(0, 760)$ for the vertical case.

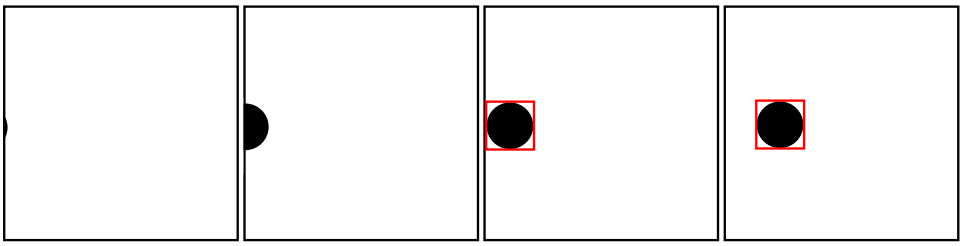


Figure 4.2: IR source entering the scene. The red box surrounding the point indicates whether the point is detected, and where the detected point is positioned.

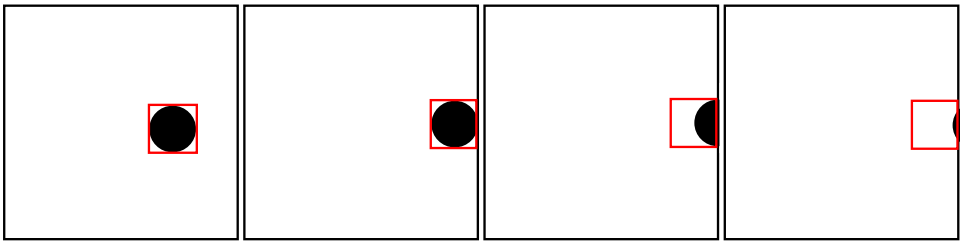


Figure 4.3: IR source entering the scene. The red box surrounding the point indicates whether the point is detected, and where the detected point is positioned.

points this means 24 possible orders the points may be detected in, which in use with an algorithm where the knowledge of the exact correspondences between the points are necessary, poses a problem. Therefore, the detected points have to be sorted in the correct order after detection. The sorting will be described with the introduction of the tracking beacon, in Section 4.2.3.

Secondly, when there are more than four sources of infrared light available in the scene; there is no guarantee that the correct emitters will be detected. The issue with the sorting may be subdued and even solved for certain point configurations, but erroneous detections cannot be evaded, except by making sure that there are no unwanted emitters in the camera view. Figure 4.4 illustrates a problematic scenario, with a dominant infrared source present in the camera view. If the Wii remote manages the capture the four desirable dots present in the image at first, they will keep

being detected until any of them are removed from view. Then, if an additional source is present, it will be detected, and stay as one of the preferred detections until it has been removed from view.

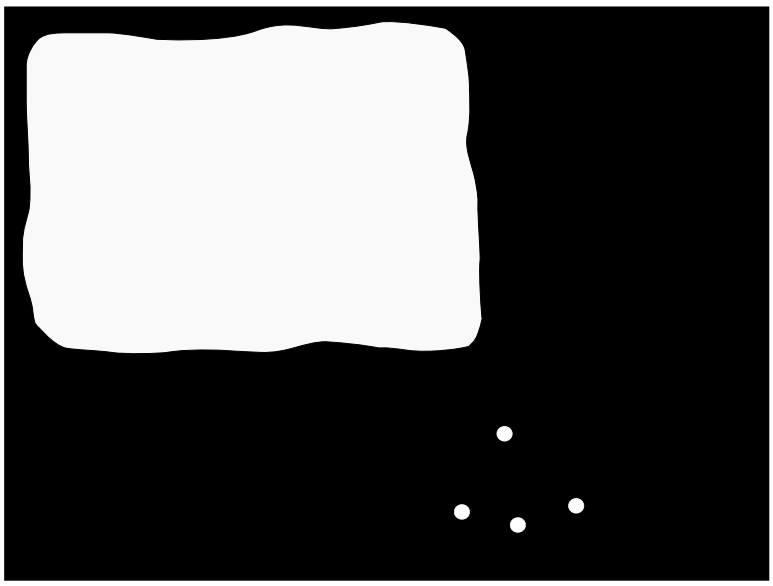


Figure 4.4: Non optimal conditions for point detection. In this case, a dominant light source is present in the image, which may be incorrectly assumed to be a desirable feature point, and thus detected as such.

4.2 Tracking Beacon

The tracking beacon, or marker, is the object which is going to be observed by the Wii remote, and it is the complementary part of the input device proposed in this report. Its purpose is to provide a set of easily distinguishable feature points, detectable to the Wii remote, such that the position and orientation of the marker may be recovered. To allow us to compute the full six degrees of freedom, with as much accuracy as possible, four feature points are going to be used.

Albeit infrared emitting diodes of all types being available, the most common are diodes which emit light at wavelengths of either 850 nm or 940 nm. But according to most report, i.e. in [4, 23], the Wii remote is much more sensitive to the latter, and as such the Vishay TSAL6400 diodes were

chosen for use on the tracking beacon. The TSAL6400 is a high efficiency infrared emitting diode packed in a dome casing. The specification of these LEDs is summarized in Table 4.1, while their expected, relative radiant intensity at different viewing angles is illustrated in Figure 4.5.

According to Figure 4.5, when the diodes are utilized according to their optimal specifications and the viewing angle exceeds 30°, the radiant intensity of the diodes is severely diminished. This leads to the diodes being harder to detect, or possibly becoming non-visible to the infrared camera.

Table 4.1: Vishay TSAL6400 specifications

Parameter	Typical value	Maximum value
Forward voltage	1.35V	1.6V
Forward current	100mA	200mA
Peak wavelength	940nm	-
Radiant intensity	$40\frac{mW}{sr}$	$125\frac{mW}{sr}$
Angle of half intensity	$\pm 25°$	-

4.2.1 Schematics

Due to the fact that four diodes will be used on the beacon, and according to their specification, listed in Table 4.1, we need a voltage source capable of providing at least $1.35\,V \times 4 = 5.4\,V$. To retain the mobility of the beacon, we propose the usage of a battery. By assuming a standard $9\,V$ battery is to be used as the voltage source, the four diodes have been installed in a serial configuration. In addition, a resistor has been connected in serial with the diodes, to limit the current through them. The appropriate size of the resistor can easily be calculated using Ohm's law, and the specifications of the four diodes, and the knowledge of the voltage source which is to be used:

$$\begin{aligned} v &= Ri \\ R &= \frac{9 - (1.35 \times 4)}{0.1}\,\Omega = 36\,\Omega \end{aligned} \tag{4.1}$$

Multiple components are needed to attain a resistance of $36\,\Omega$, however, as we have chosen to use a single resistor, a resistor with resistance of $39\,\Omega$ was used instead. Moreover, a potentiometer was also utilized in order to control the supply voltage to the rest of the circuit, such that the intensity of the diodes could be adjusted. The setup is illustrated in Figure 4.6.

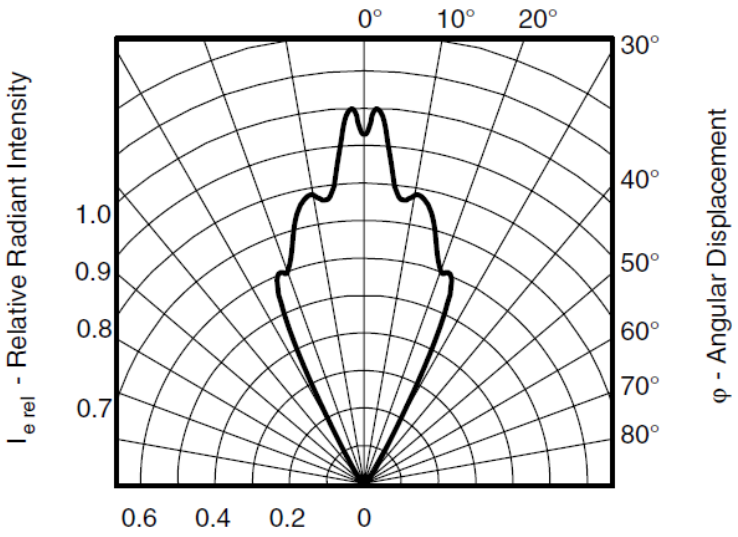


Figure 4.5: The relative radiant intensity of the diode plotted against the angular displacement it is viewed from. When the angular displacement increases above 30°, the radiant intensity decreases significantly. The image is reproduced from the Vishay TSAL6400 product sheet.

4.2.2 Physical Configuration

The physical placement of the diodes has been devised in accordance with the setup proposed by Fleisch [14]. This means that the optical points are in a non-coplanar configuration, while three of the points are nearly co-linear in most cases when viewed from the camera. This setup has been chosen for the purpose of being compatible with the Tvrx library, both in order to test the performance of the implementation of the library, including the pose estimation algorithm, and the fact that it is a proven design. The diode configuration is illustrated in Figure 4.7(a), with the positions of the diode listed in Table 4.2. The positions are stated in a frame of reference with its origin defined in the lower left corner of the board the diodes are installed on. The axes of the reference frame are illustrated in the figure. The implemented marker is shown in Figure 4.7(b).

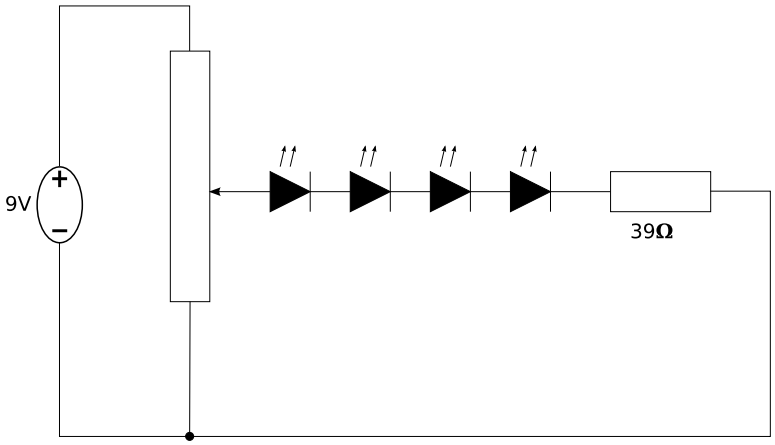


Figure 4.6: Schematic of the tracking beacon. A voltage source which can supply 9 V was used, with a potentiometer with a resistance of up to 500 Ω, allowing changing the supply voltage to the diodes. The current limiting resistor in series with the diodes has the resistance of 39 Ω.

Table 4.2: Diode positions

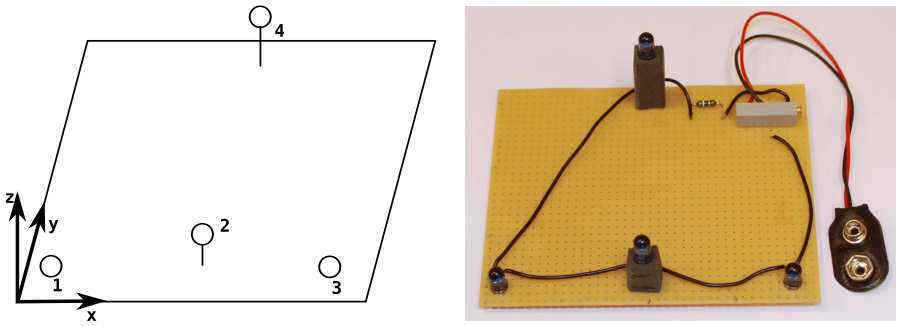
Diode number	x	y	z	Unit
1	5.5	6	4.5	mm
2	49	6	19	mm
3	91.5	6	4.5	mm
4	49	74.5	27	mm

4.2.3 Point Sorting

For the beacon illustrated in Figure 4.7(a), we can exploit the geometry of the points and an approach with some trigonometry and vector calculations is enough to determine the correct order of the points. The following approach will in most cases be sufficient for sorting the points $\mathbf{p}_i$, for $i = 1, ..., 4$, where the emitters are numbered as in Figure 4.7(a).

1. Find the second emitter by, for each detected point, examining the distances from it to the other points. The point which is closest to the other points is assumed to be the second emitter.

2. Find the fourth point by examining the angles between the vectors $\mathbf{t}_{i,2} = \mathbf{p}_i - \mathbf{p}_2, \quad i = 1, 3, 4$. The largest angle will be between $\mathbf{t}_{1,2}$ and



(a) Diode positions

Figure 4.7: The optical marker. The four diodes are placed at the positions designated by the number on the marker illustrated in Figure 4.7(a), with their spatial positions as described in Table 4.2, stated in the reference frame defined as in the figure. Figure 4.7(b) illustrates the implemented marker.

$\mathbf{t}_{3,2}$, and therefore, the fourth point is not one of these.

3. Determine which point is the first and which the third is. This can be done by finding the right normal to the vector $\mathbf{v}_{2,4}$, and then comparing the angles between the remaining vectors. The vector yielding in the smallest angle is the third point, the largest being the first point.

4. Finally, store the points in the correct order.

4.3 Pan-Tilt Unit

The provided device which is to be controlled is a pan-tilt unit (henceforth abbreviated with PTU) produced by Directed Perception Inc. [17], designated model number PTU-D46-17P70T. The unit is computer controllable through communication with a control device, the Directed Perception PTU-D46 Pan-Tilt Controller, which includes a RS485 and a RS232 port for communication, using either a binary or an ASCII-based protocol.

The unit is capable of panning in the range $[-159°,\ 159°]$[2], with a maximum attainable pan velocity of $300°/sec$. The tilt range is $[-47°,\ 31°]$ with a maximum attainable tilt velocity of $60°/sec$. The specifics on the

[2]The actual ranges may vary slightly from the used model, as it recalibrates at initialization.

interaction procedure of the device, including connecting to it and aiming, are covered in Appendix A. The PTU and its dimensions are illustrated in Figure 4.8.

Furthermore, the device may be controlled either by assigning a position, or a velocity in either direction. This allows it to be just as easily integrated with mechanical input devices such as joysticks, as with a software implementation. Instead of controlling the PTU to aim at desired pan or tilt angles in degrees or radians, the total range of the axes are split into a certain number of positions. The angle at which the PTU is aiming at can be calculated based on its current position on the axis and the axis resolution. The axis resolutions are given in seconds of arc per position[3], which corresponds to 0.0002778° per position. For the angle of the pan axis, this conversion can be calculated as

$$\gamma_p = 0.0002778\, R_p P_p \tag{4.2}$$

where γ_p is the pan angle, R_p is the resolution of the pan axis, and P_p is the current pan position. R_p may be queried from the unit, and for this given PTU is 185.1428 positions per degree.

For the angle of the tilt axis, the conversion is as

$$\gamma_t = 0.0002778\, R_t P_t \tag{4.3}$$

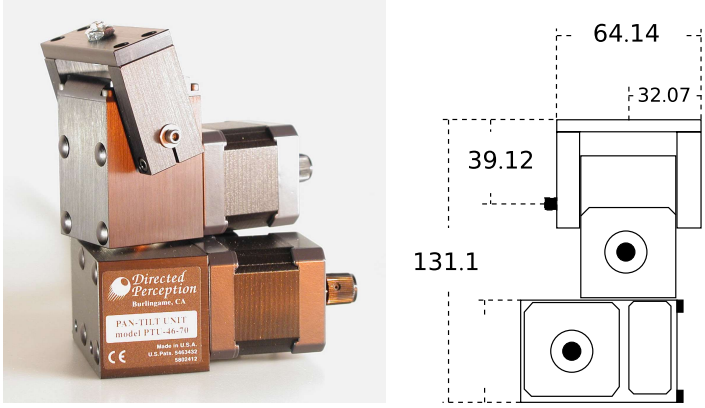
where γ_t is the tilt angle, R_t is the resolution of the tilt axis, and P_t is the current tilt position. R_t is 46.2857 positions per degree for this given unit.

4.4 System Design

The system explored in this report is comprised of a measuring device, the Wii remote, and an interaction device, the tracking beacon. The Wii remote has been presented in Section 4.1, while a possible beacon was presented in Section 4.2. Furthermore, the PTU presented in Section 4.3 is to be controlled in a fashion intuitive to the user interacting with the beacon. The software solution developed in this report attempts to combine the various software and hardware examined in this report to achieve an intuitive way to control a device capable of panning and tilting.

The measurements provided by the Wii remote have to be converted to a signal which can be applied to the PTU. This can be done by attempting to

[3]Minutes and Seconds of arc are units of angular measurement. A minute of arc is equal to $\frac{1}{60}^\circ$, while a second of arc is defined as $\frac{1}{60} arcmin$.



(a) The PTU (b) PTU dimensions

Figure 4.8: The PTU and its dimensions in millimeters. The images are reproduced from the PTU-D46 product sheet.

estimate the full 6 DOFs of the tracking beacon using one of the algorithms described in Section 3.2, or using one of the more complete systems described in Section 3.4, and then converting the estimated beacon pose to a signal which may be used to aim the PTU.

Figure 4.9 illustrates the general program flow of the implemented solution. When the program starts, an initialization of both the software and hardware is performed. The initialization includes the following:

- Detect and connect to the Wii remote.
- Connect to the PTU.
- Reset the PTU, moving it to a resting position[4].

When the initialization is done, the main program loop is entered. First, the program polls for events, and acts accordingly on the results. Only two events are considered in the application; the quit event and the detection of four IR sources. If neither of these have occurred, the program returns to the start and continues to poll for events.

If however, four emitters have been discovered by the Wii remote, these points are first sorted before being used to calculate the pose of the beacon. After this has been performed, a control signal is created and sent to the

[4]The resting position is here when the PTU is at pan angle 0° and tilt angle 0°.

PTU. This procedure is continued until a quit event has happened, which concludes the program.

4.4.1 Implementation Specifics

There are a few key elements needed in order to implement the software solution described above. The first is to communicate with the Wii remote. Although there are a lot of existing libraries which achieves this, the WiimoteLib library introduced in Section 3.3.2 was chosen due to its feature set and ease of use. This library is used to retrieve the measurements performed by the Wii remote, which then are sorted in the correct order using the procedure defined in Section 4.2.3.

Second, a pose estimation algorithm is needed to determine the pose of the beacon based on the sorted feature points. For this task, the POSIT algorithm proposed by DeMenthon and Davis [11] was considered, because it is a proven algorithm and has been implemented in both the OpenCV and the FreeTrack library. However, since the Tvrx library, mentioned in Section 3.4.3, provides a more complete solution for the pose estimation problem with the use of the OpenCV implementation of the algorithm, it was employed for this task. The configuration and usage of the Tvrx library can be found in Appendix C, and the use of the pose estimation algorithm in OpenCV is described in Appendix D.

Last, the serial communication protocol needed to control the PTU needs to be implemented. This includes the initialization of a serial socket with the appropriate settings, and creating a conversion from the pose parameters to the PTU parameters. Appendix A contains more specifics about this.

4.4.2 Camera Control

As mentioned in Section 4.3, the PTU is able of panning in the range $[-159°,\ 159°]$ and tilting in the range $[-47°,\ 31°]$, and the viewing angle at which the diodes are still visible to the camera has been measured to be approximately $50°$ for all viewing directions Thus, the operating area of the PTU is greater than the effective operating area of both the optical marker and of its user, and as such the control mechanism should account for this. Therefore, in order to cover the entire panning and tilting range of the PTU, the following control mechanisms were considered:

1. Aim the PTU using estimated yaw and pitch angles.

2. Aim the PTU using an upscaled estimate of the yaw and pitch angles.

3. Aim the PTU using estimated yaw and pitch angles, and maintain an offset of the achieved angle.

The first procedure is the simplest one, but using it, the whole range of the PTU is not achievable. This was initially implemented in order to investigate if the input device is usable to control the PTU. Using the second procedure however, the complete range is covered, but with the loss of resolution due to the upscaling of the estimated angles. Using the third option, both the original tracking resolution is retained and the whole range of the PTU is covered. However, this procedure needs some user input, in order to distinguish between when to aim the PTU and when not to do this. This procedure was implemented in the manner that whenever an operator keeps a button pushed, the tracking becomes active and the current aiming angles keep being updated until the button is released. This then allows the user to reposition the beacon, in order to further pan or tilt the PTU when reactivating the tracking.

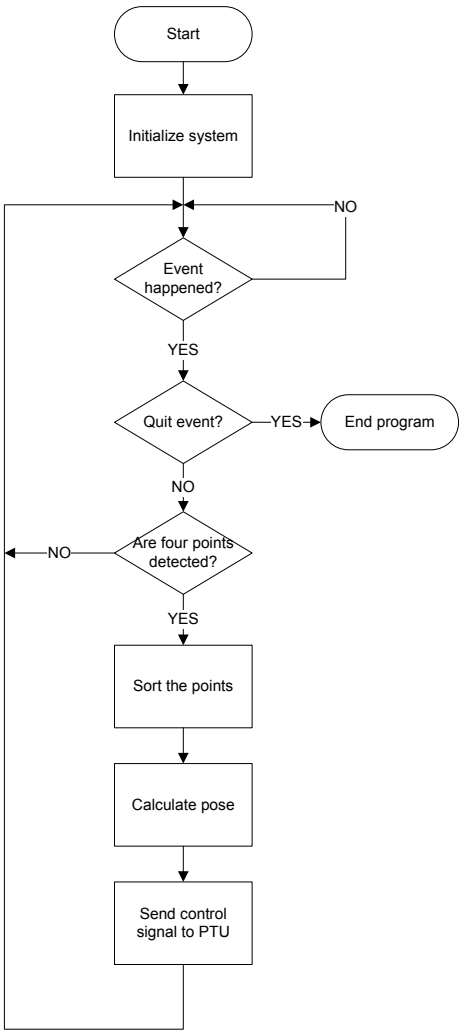


Figure 4.9: A flow chart representing the general program flow using the Wii remote to estimate the beacon pose, and consequently aim the PTU. The notion of an event is here either the discovery of an optical point by the camera, or the push of a button to end the execution of the program.

Chapter 5

System Evaluation

The most sensitive part of the system discussed in this report is the input device. Because the user input is devised through vision measurements, the performance of the system relies heavily on the camera in use, and the pose estimation algorithm. Therefore, in this chapter the operating range of the input device will be determined, in addition to an evaluation of the pose estimation algorithm. The pose estimation algorithm evaluated is the POSIT algorithm developed by DeMenthon and Davis [11]. It has since been implemented in OpenCV [2], which is the implementation being assessed here, and in FreeTrack [1].

The chapter is organized into the following sections:

- **Evaluation of the Wii Remote**
  The field of view of the Wii remote will be determined in this section, and consequently the operating the range of the input device.
- **Experiment Equipment**
  In this section the equipment which will be used to perform the experiments is presented.
- **Evaluation of Pose Estimation**
  The experiment procedure and the achieved results of the pose estimation algorithm are presented in this section.

5.1 Evaluation of the Wii Remote

A lot of the specifications of the Wii remote are, as previously mentioned, unknown. However, there are a few properties which can easily be determined to some degree of certainty. Although the properties of the lenses,

the focal length and other intrinsic parameters require more complex measurements, the field of view may be determined. Knowing the field of view also allows us to determine the operation range of the input device.

5.1.1 Field of View

With the use of a single infrared emitter[1] and measuring the last position it is detectable when leaving the camera view, the horizontal and vertical fields of view may be computed. The measurements were conducted by registering the position of the emitter, just as it has left the scene, for upward, downward, left and right directions and a number of distances to the camera. For the set of measurements, the probability density function was computed, and is illustrated in Figure 5.1(a) for the horizontal field of view, and Figure 5.1(b) for the vertical, and Table 5.1 summarizes these results. According to the calculated mean of the measurements, the Wii remote has slightly lower field of view than the 45° horizontal field of view assumed in [23], and significantly higher than the 33° horizontal and 23° vertical field of view assumed in [3]. Furthermore, the pixels are seemingly square, but a slight difference in their dimensions was measured. This may be due to the lens, or even the image analysis algorithm.

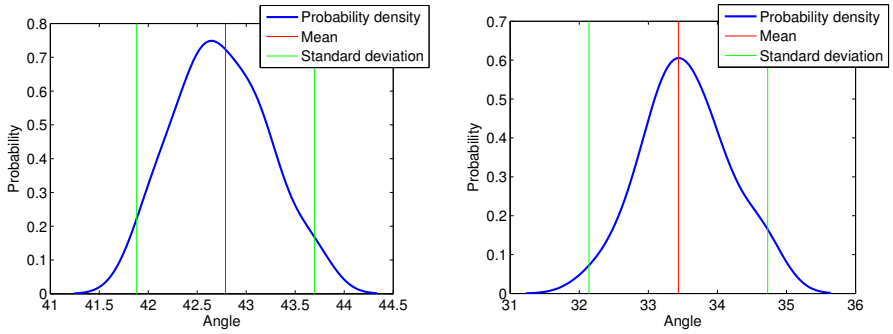
Table 5.1: Measured field of view

Field of view	Mean	σ	deg/pixel
Horizontal	42.7883°	0.9073°	$0.0418°/px$
Vertical	33.4350°	1.2932°	$0.0435°/px$

Additionally, the distance at which the infrared emitters could be detected by the camera was measured. As the 940 nm LEDs are detected with more intensity then equivalent sources with other wavelengths, they are way too bright to be resolved at close distances. For a single emitter, the near limit was measured to be around 60 mm, while it kept being detected at distances greater than 10 m.

For a head mounted beacon, and the camera mounted on the display monitor, we can assume a distance from the beacon to the camera of about 80 cm. With the mean of the measured field of view, this yields an operating area of almost 63 cm horizontally, and 48 cm vertically.

[1] The emitter is the Vishay TSAL6400, with its specification described in chapter 4.2.



(a) Horizontal field of view (b) Vertical field of view

Figure 5.1: Plot of probability density function for horizontal and vertical field of view based on a set of measurements. This is denoted by the blue line, while the red line is the mean of the measurements, and the green line denotes a standard device distance from the mean.

5.2 Experiment Equipment

To be able to reliably review the pose estimation algorithm in a real world setting, a test rack was devised. The test rack has a mount for the Wii remote such that it may stay completely stationary while the experiments are performed. The camera is facing a pan-tilt unit, on which the tracker may be mounted. The PTU is described in more detail in Section 4.3, and it can be controlled to aim at certain angles with good accuracy.

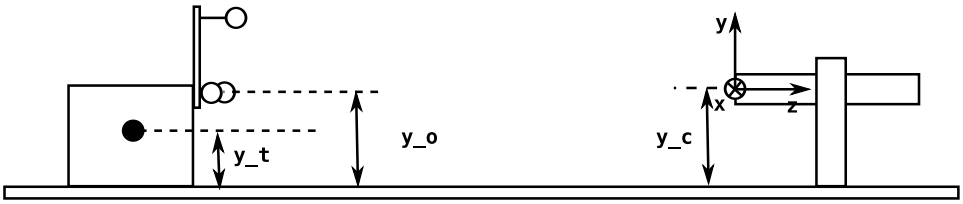
Certainly, using a device only able of panning and tilting to alter the pose of the optical marker, only a few of the degrees of freedom may be examined. However, as changes in yaw, pitch and, to some degree, the translations may be measured, an assessment can still be obtained. It also directly indicates whether the system may be used as the input for the control of the PTU.

5.2.1 Geometric Properties of the Experiment Equipment

In order to determine the pose of the marker while it undergoes geometric transformation due to the motions of the PTU, the actual distances to the marker have to be measured. As the estimated pose, provided by the POSIT algorithm, defines the pose at the feature point defined first in accordance with Figure 4.7(a), the distances to this diode have been measured. Sketches of the setup of the experiment equipment are provided in Figure 5.2(a) and

Figure 5.2(b), with the distance measurements listed in Table 5.2. Here, the global frame of reference is also defined, with the orientation of it as shown in the figure. Furthermore, the origin of the global frame of reference is placed at the center of the lens of the Wii remote, and consequently all distances are described with regards to the this position.

(a) Top view of the experiment equipment



(b) Side view of the experiment equipment

Figure 5.2: Experiment equipment. (a) is a top view of the equipment, and (b) illustrates a side view. The distance measurements here are defined in Table 5.2.

5.2.2 Experiments

The experiments have been devised in order to determine the accuracy and consistency of the chosen algorithm. To determine the accuracy we will look at the case where the pose of the marker is slowly changing. This is done for both the panning and tilting motions using the PTU. In order to evaluate the consistency, the pose of the marker was repeatedly altered in the same fashion as above, but at a higher velocity. This allows us to detect whether there are significant differences in the estimated poses for similar motions. For all the experiments, the velocities of the rotary motions achieved by the PTU are all constant. This is due to a mechanical capability of the PTU.

Table 5.2: Distances on the test equipment

Parameter	Value	Description
z_{op}	15mm	Distance from the significant point to the panning axis in the z direction
z_{co}	821mm	Distance from the camera to the significant point
x_{po}	44mm	Distance from the significant point to the panning axis in the x direction
y_c	172mm	The height of the camera
y_o	132mm	The height of the significant point with the marker at rest
y_t	92mm	The height of the tilting axis

The experiments are summarized as follows:

- Experiment 1: Panning at low, constant velocity.
- Experiment 2: Tilting at low, constant velocity.
- Experiment 3: Repeated panning at constant velocity.
- Experiment 4: Repeated tilting at constant velocity.

The experiment procedures, along with the results, will be presented in Section 5.3. For the experiments 1 and 3, the PTU was instructed to keep the tilt axis at 0°, while panning in the range $[-51.43°, 51.43°]$. For the PTU, these pan angles correspond to the commands `pp-1000` and `pp1000`, where the positions are calculated from (4.2) and the command is defined in Appendix A. The pan angles were chosen such that not all of the IR LEDs were detectable in the limits. As a bonus, the behavior of the various implementations could be witnessed whenever an emitter disappeared and reappeared in the image.

For the experiments 2 and 4, the PTU was instructed to keep the pan axis at 0°, while tilting in the range $[-30.22°, 30.22°]$. The tilt angles correspond to the commands `tp-2350` and `tp2350`, where the positions are calculated from (4.3) and the command is defined in Appendix A. The mentioned tilt angles were chosen to be symmetric and close to the limit of the PTU. In this case, the upper tilt limit is nearly reached.

5.2.3 Expected Pose

Based on the geometric composition of the test rack, with the positions of the mounted devices available, we can calculate the expected pose for the test

cases. For the expected pose, we are concerned with the translation along the axes and the rotation about the axes during the panning and tilting motions. The rotations will here be parameterized using Euler angles, in order to easily compare them. While the yaw and pitch rotations of the system provide the most notable motions and are of most interest, all the parameters where computed and provided as comparisons in the figures of the results for the performed experiments.

5.3 Evaluation of Pose Estimation

In this section, we will evaluate the performance of the pose estimation algorithm when used in conjunction with the Wii remote, in order to discover what is to be expected in terms of accuracy and consistency of the estimate, for the whole system.

The estimation algorithm which is reviewed is the POSIT algorithm, proposed by DeMenthon and Davis [11]. An example of the implementation of the algorithm can be found in OpenCV, which has been used by Fleisch [14] to achieve pose estimation in the library Tvrx. However, the Tvrx library also provides additional functionality, such as filtering of the detected points, as well as the estimated parameters. The parameters of the filters in use may be configured, and the configuration options, further explored in Appendix C, also suggest the possibility of turning the filters completely off. This however, proved not to be the case, since the enabling of this option immediately caused the library to malfunction. Therefore, to also evaluate the performance of the algorithm without the use of a filter, the pose estimation function of the Tvrx library was re-implemented[2] using OpenCV. The Tvrx library was implemented and configured as described in Appendix C, and this implementation will henceforth be referred to as the Tvrx implementation, while the re-implementation of the pose estimation functions of Tvrx will be referred to as the OpenCV implementation, even though both employ the use of the same algorithm. The implementation specifics of the latter are described in Appendix D.

The procedure and results for the experiments are described in Section 5.3.1 through 5.3.4. The results will be presented as Euler angles in degrees for the rotations, while the translations will be presented in millimeters, with the plot figures organized as follows:

- (a) The rotation around x-axis (pitch)

[2] Although the same camera parameters were used, the implementations may slightly vary.

- (b) The rotation around y-axis (yaw)
- (c) The rotation around z-axis (roll)
- (d) The translation along x-axis (sway)
- (e) The translation along y-axis (heave)
- (f) The translation along z-axis (surge)

The rotation and translations mentioned here are of the local reference frame defined in Figure 4.7(a), which is compared to the global reference frame defined in Figure 5.2.

5.3.1 Experiment 1: Low Velocity Panning

The procedure followed for this experiment is described in Table 5.3 for both the Tvrx and OpenCV implementations. The estimates of the poses are illustrated in Figure 5.3. The blue lines in the figure indicate the estimates achieved by Tvrx, the green lines indicate the estimates gathered from the OpenCV implementation, while the red lines denote the expected pose described in Section 5.2.3.

For this experiment, the most prominent motion is in yaw, which is in the range $[-51.43°,\ 51.43°]$, while the pitch, roll and heave should not change during the experiment. However, due to the PTU panning some change in surge and sway should also be observed.

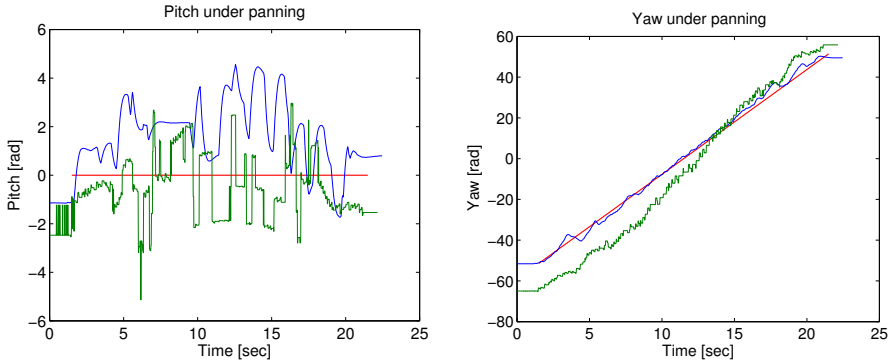
Inspecting the plots in Figure 5.3, we can see that both implementations have some trouble estimating the translatory parameters. For the case of the heave and sway parameters, the form, and the magnitude to some degree, are similar to the expected parameters. Looking at Figure 5.3(f), both implementations have some trouble estimating the distance to the beacon, as well as detecting any change in the parameter. The Tvrx library performs better in this regard, as it does calculate a more correct distance to the beacon. The roll and the pitch angles are poorly estimated, but considering the signal sizes, the error is rather small and was expected. Especially the pitch has large fluctuations around the expected results, while roll suffers of an incline, which should not be present.

The Tvrx library does a good job of estimating the yaw of the beacon. Albeit having some fluctuations, the estimate is fairly close to the actual angle. The OpenCV implementation, on the other hand, provides a more erroneous estimate in this experiment.

Common to all the results shown in Figure 5.3, the Tvrx library yield the better estimates of the two, with the exception of the heave parameter. This clearly indicates that the usage of the POSIT algorithm is better implemented in Tvrx than using the procedure described in Appendix D.

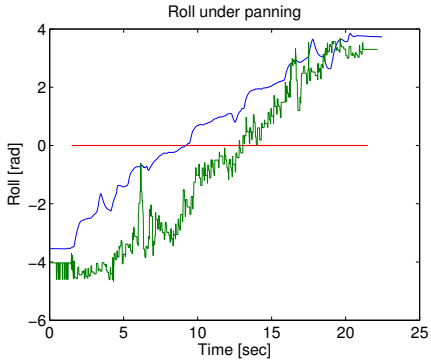
Table 5.3: Experiment 1: Low velocity panning

Step	Description	PTU Command
1	Initialize and reset the system.	'r'
2	Instruct the PTU to move to starting position	'pp-1000'
3	Set the rotation velocity for the PTU	'ps100'
4	Open a new log	-
5	Record the start-up time	-
6	Instruct the PTU to rotate to destination	'pp1000'
7	When PTU is finished panning, conclude the experiment.	-
	This implies closing the log, and resetting the device	'r'

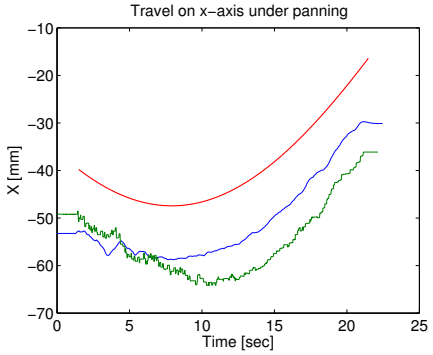


(a) Rotation around the x-axis

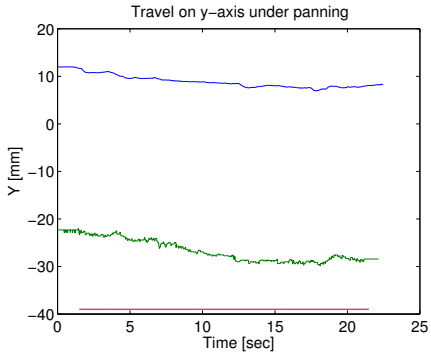
(b) Rotation around the y-axis



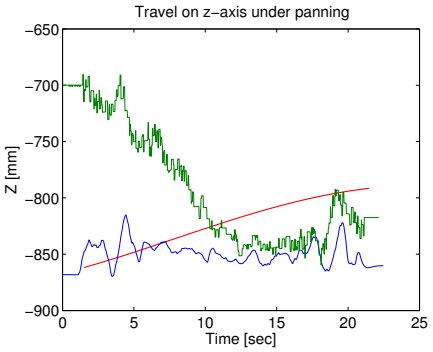
(c) Rotation around the z-axis



(d) Translation along the x-axis



(e) Translation along the y-axis



(f) Translation along the z-axis

Figure 5.3: Estimates of pose parameters based on the OpenCV implementation (green lines) and the Tvrx implementation (blue lines), while panning at low velocity. The red lines are the actual parameters.

5.3.2 Experiment 2: Low Velocity Tilting

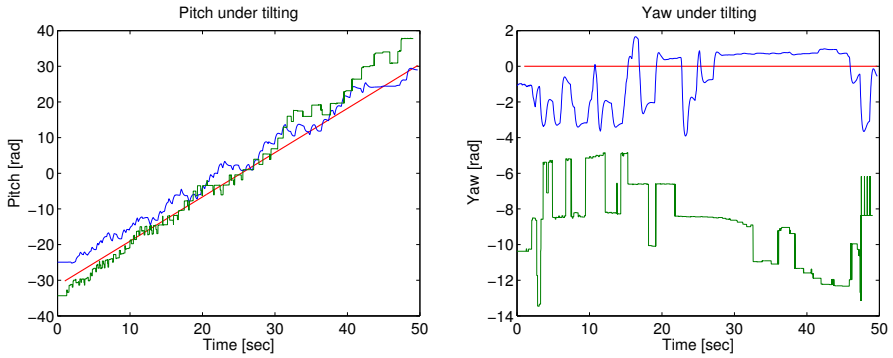
The procedure for performing experiment 2 is described in Table 5.4. In this experiment, the most prominent motion is in pitch, which is in the range $[-30.2°,\ 30.2°]$. Figure 5.4 illustrates the results of this experiment. The blue lines in the figure indicate the estimates achieved by the Tvrx implementation, with the green lines being from the OpenCV implementation and the red lines denoting the expected results.

Considering the plots in Figure 5.4, both implementations have trouble estimating the roll and yaw, and also yield in some error in the translatory parameters. With the exception of heave, the Tvrx implementation provides better estimates, both in terms of detecting the correct change in parameters and the correct magnitude of the signals.

Both implementations provide reasonable estimates of the pitch parameter, but both are less than optimal. The Tvrx implementation seems to be the better, because of the less noisy estimate, but it initially estimates a lower value than the true angle. The OpenCV implementation however estimates a slightly higher incline in the parameter than the true value.

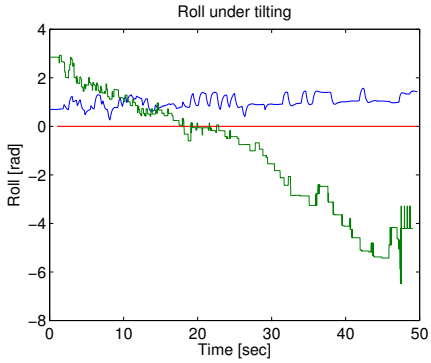
Table 5.4: Experiment 2: Low velocity tilting

Step	Description	PTU Command
1	Initialize and reset the system.	'r'
2	Instruct the PTU to move to starting position	'tp-2350'
3	Set the rotation velocity for the PTU	'ts100'
4	Open a new log	-
5	Record the start-up time	-
6	Instruct the PTU to rotate to destination	'tp2350'
7	When PTU is finished panning, conclude the experiment.	-
	This implies closing the log, and resetting the device	'r'

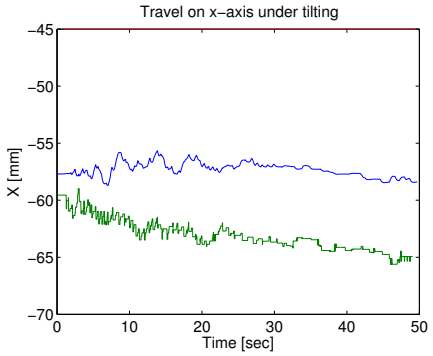


(a) Rotation around the x-axis

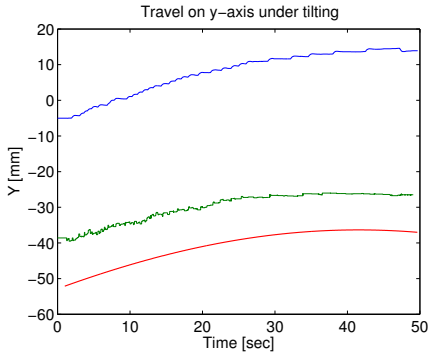
(b) Rotation around the y-axis



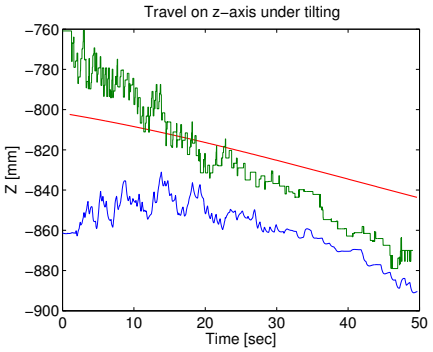
(c) Rotation around the z-axis



(d) Translation along the x-axis



(e) Translation along the y-axis



(f) Translation along the z-axis

Figure 5.4: Estimates of pose parameters based on the OpenCV implementation (green lines) and the Tvrx implementation (blue lines), while tilting at low velocity. The red lines are the actual parameters.

5.3.3 Experiment 3: Repeated Panning

The procedure for the repeated panning experiment is explained in Table 5.5, while the results for the forward passes are presented in Figure 5.5. For this experiment the number of passes has been set to 2000, which consequently yields 1000 passes from pan angle $-51.43°$ to angle $51.43°$, and the same amount in the reverse direction. Because of the number of results, only the mean and standard deviation of the forward panning motion have been include in Figure 5.5. However, the full plots of both forward and reverse passes have been included in Appendix E.

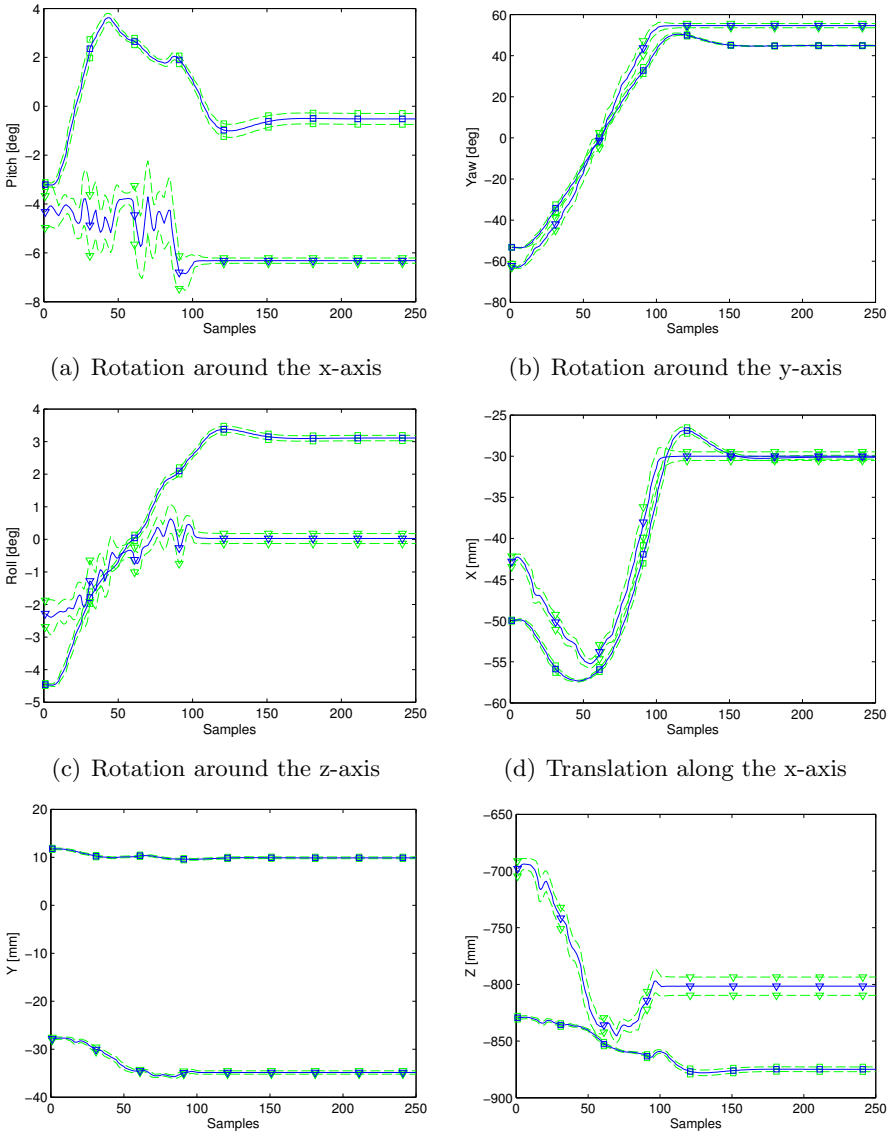
The lines marked with a square indicate the estimates from the Tvrx library, while the lines marked with a triangle denote the parameters estimated by the OpenCV implementation. The blue lines indicate the calculated mean, and the dashed green lines indicate a standard deviation distance to the mean.

By inspecting the results, we can see that the PTU needs around two seconds[3] to complete the whole panning motion. For the OpenCV implementation, the parameters come to rest at this time, but the effects of the Kalman filter employed by the Tvrx implementation can be witnessed. Due to the panning velocity the Tvrx implementation has an overshoot and a whole second is needed before it comes to rest. In terms of consistency, both implementations perform well for the parameters which have distinctive motions. But the Tvrx implementation is also very consistent for the remaining parameters as well. This can be seen by the low variability of the estimates as evident by the plotted standard deviation.

[3] A complete pass is performed in five seconds, and 250 samples are taken.

Table 5.5: Experiment 3: Repeated panning

Step	Description	PTU Command
1	Initialize and reset the system.	'r'
2	Initialize run counter to 0	-
3	Set the rotation velocity for the PTU	'ps1000'
4	Instruct the PTU to move to starting position	'pp-1000'
5	Wait 5 seconds	-
6	Record run number	-
7	Open a new enumerated log	-
8	Record the current time as the start-up time	-
9	Instruct the PTU to rotate to destination	'pp1000'
10	Wait 5 seconds	-
11	Increase run number	-
12	Record run number	-
13	Open a new enumerated log	-
14	Record the current time as the start-up time	-
15	If run number has not reached the run limit, go to step 4	-
	else, conclude the experiment	-
	This implies closing the log, and resetting the device	'r'



(a) Rotation around the x-axis

(b) Rotation around the y-axis

(c) Rotation around the z-axis

(d) Translation along the x-axis

(e) Translation along the y-axis

(f) Translation along the z-axis

Figure 5.5: Estimates of pose parameters based on the OpenCV implementation (solid line marked with triangles), and the Tvrx implementation (solid line marked with squares) for repeated panning motions. The dashed lines above and below both of the mean plots indicate a standard deviation distance from the mean.

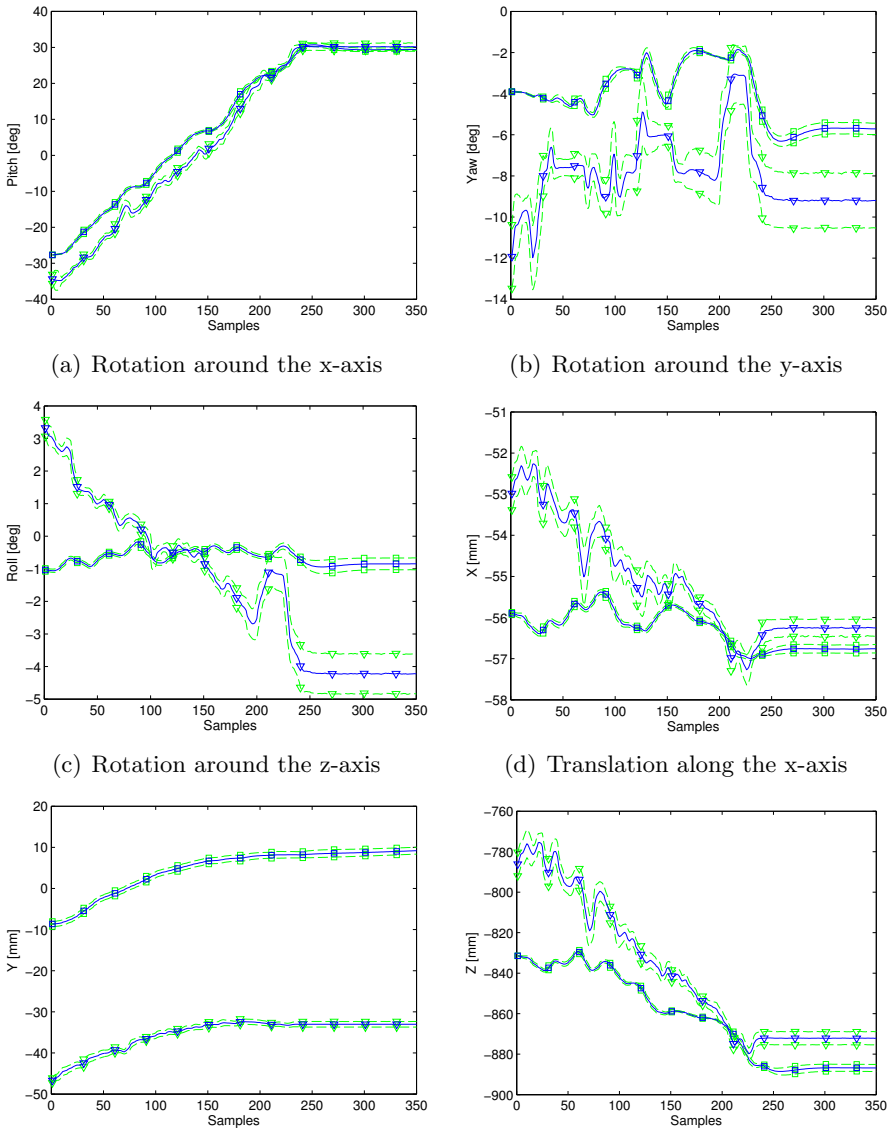
5.3.4 Experiment 4: Repeated Tilting

The procedure for the repeated tilting experiment is explained in Table 5.6, while the results for the forward passes are presented in Figure 5.6. Similar to the repeated panning experiment, 2000 passes were conducted in this instance as well. The PTU was instructed to start at $-30.2°$ and tilt to angle $30.2°$ at first, before returning. Only the upward pass is illustrated in Figure 5.6, and also here the lines marked with squares correspond to the Tvrx estimates, while the triangle marked lines correspond to the OpenCV estimates. The complete plots of all the passes are included in Appendix E.

By inspecting the results, much of the same conclusions can be drawn. However, for the case of the Tvrx implementation the overshoot from the last experiment is not observable in this instance. This may be due to the slower velocity of the tilting compared to the panning. The PTU requires around five seconds completing the entire tilting motion. The Tvrx implementation generally provides more consistent estimates compared to the OpenCV implementation, for all the parameters. However, there are quite a lot of fluctuations present in the estimate of the pitch, even in the case of the mean. Evidently, both implementations have trouble capturing the motion of the marker in this case. This may be due to the design of the beacon, as the positions of the diodes compared to each other change less than in the case of panning.

Table 5.6: Experiment 4: Repeated tilting

Step	Description	PTU Command
1	Initialize and reset the system.	'r'
2	Initialize run counter to 0	-
3	Set the rotation velocity for the PTU	'ts1000'
4	Instruct the PTU to move to starting position	'tp-2350'
5	Wait 5 seconds	-
6	Record run number	-
7	Open a new enumerated log	-
8	Record the current time as the start-up time	-
9	Instruct the PTU to rotate to destination	'tp2350'
10	Wait 5 seconds	-
11	Increase run number	-
12	Record run number	-
13	Open a new enumerated log	-
14	Record the current time as the start-up time	-
15	If run number has not reached the run limit,	-
	go to step 4	
	else, conclude the experiment	-
	This implies closing the log, and resetting the device	'r'



(a) Rotation around the x-axis

(b) Rotation around the y-axis

(c) Rotation around the z-axis

(d) Translation along the x-axis

(e) Translation along the y-axis

(f) Translation along the z-axis

Figure 5.6: Estimates of pose parameters based on the OpenCV implementation (solid line marked with triangles), and the Tvrx implementation (solid line marked with squares) for repeated tilting motions. The dashed lines above and below both of the mean plots indicate a standard deviation distance from the mean.

Chapter 6

Discussion

The chapter contains some closing remarks and is organized in the following sections:

- **Nintendo Wii Remote**
  Remarks about the imaging hardware used in the assignment are presented here.
- **Optical Marker**
  The optical marker is discussed in this section.
- **Pan-Tilt Unit**
  The pan-tilt unit which acted as the output devices is discussed in this section.
- **Experimental Setup**
  Remarks about the experimental test bench used in the assignment are presented here. The achieved results are also debated in this section.
- **Future Work**
  In this section some extensions to the assignment and future work are discussed.

6.1 Nintendo Wii Remote

The Wii remote is an impressive device, which has proven to be very useful in much more diverse areas than for which it was originally intended. The device has been successfully utilized in both research and educational projects, where its area of application has been extended due to the vision

and motion sensing capabilities of the device. Some of these areas are virtual and augmented reality applications, where the Wii remote is used to interact with the virtual world or track objects in space, and motion control, where its motion sensing capabilities are used to control a robotic arm or a vehicle. Some of the success of the device is due to the fact that it is versatile, and may be used in a variety of applications, while still being a low-cost device.

In the system explored in this report, the mobility and motion sensing capabilities of the Wii remote were not utilized. The only part of the device actually necessary for the system function is the infrared camera. In an application where the device is to remain stationary, there is less need for a wireless communication protocol and a compact, mobile voltage source. However, in contrast to the system explored in this report, reversing the roles of the tracker and the camera is possible, and can be equally fitting for some applications. This is actually how the system proposed by Kreylos [20] is initially configured, but it is applied by Rickard and Davis in [31] in a manner similar to the system explored in this report.

6.2 Optical Marker

The marker with the optical feature points has been configured according to the marker proposed by Fleisch [14], but this is not an optimal design. In general, when using optical markers as feature points, collinear configuration of points should be avoided. Such a configuration effectively makes one point insignificant, as its position can be described as a distance between the edge points. The optical points in this setup do not completely align for all observation angles, but only when the y-axes are aligned[1]. But seeing as the algorithm which is used for pose estimation is not defined for the coplanar configuration of points, the specific geometric setup of the diodes can impact its performance. However, the nearly collinear points do provide very useful information when the point correspondence problem is to be solved. The special setup is easily distinguished allowing the feature points to be sorted in correct order.

6.3 Pan-Tilt Unit

The PTU used for both testing purposes and as an output device proved to be easily controllable. Particularly, the use of human-readable commands

[1] Assuming the coordinate frames are defined as in Figure 5.2(b) and 5.2(a).

adds to its ease of use. When used in conjunction with the input device devised in this report, the issue of reducing information about the pose of the input device to the two degrees of freedom pan-tilt unit had to be managed. This can be done in a variety of ways, and some procedures were considered. The assigning the yaw and pitch of the marker to the pan and tilt angles of the unit proved to be the most functional approach. The pitch and yaw angles were then utilized in different ways in order to account for the increased range of operation of the PTU compared to the beacon in use. Although the implemented procedures served their purpose in this assignment, in terms of proving that camera control with the developed input device is possible, additional control mechanisms should be investigated.

6.4 Experimental Setup

With the use of the experimental setup illustrated in Figure 5.2, the performance of the pose estimation algorithm could be assessed in a real world environment. This also indicated how well the entire system would work, especially with the PTU as the targeted output.

However, because only two degrees of freedom are actuated in the experimental setup, some restrictions are imposed on the types of experiments which are possible to perform in order to evaluate the system. With the actuation provided by the PTU, the primary action on the beacon is the change of orientation, while the translation changes occur due to the beacon not being mounted at the center of the pan and tilt axes. Incidentally, the yaw and pitch angles of the beacon can be actuated by panning and tilting, respectively. In addition, the roll could have been examined by rotating the z-axis of the beacon into the x-axis of the global reference frame, and then actuating the tilt angles of the PTU. This however, was not examined in this report, as the targeted output was the PTU making the yaw and pitch of the input of more interest.

Even though only two degrees of freedom are actuated in the experimental setup, this depicts the intended area of operation of the implemented system, namely head tracking for computer operation. For the case of a computer user sitting in front of display monitor, the most prominent motions would be regarding the changes in head orientation. But for some applications in a similar setup, the translations of the head are more useful. This is certainly the case for virtual reality application proposed by Lee [23], and also for applications providing immersion through stereo vision. A software library for solving the latter case has been developed by Fleisch

[14], and was briefly discussed in the report, in Section 3.4.3. This is useful for the stereo vision implementations where a single display monitor is used to provide slightly different images to the eyes of the operator. In such systems the sense of depth is due to the difference in the images, but for a continuously moving user, every single projection has to be recalculated for the current head position in order to maintain this sensation.

6.4.1 Experimental Results

The results of the experiments conducted in order to examine the accuracy and consistency of the pose estimation, presented in Chapter 5, indicate that the system is usable for its intended task. None of the implementations were accurate enough in terms of estimating the absolute position, but some of this error can be blamed on mounting errors on the experiment rig and on the horizontal and vertical alignment errors in the mounting of the camera, and also on the use of an under parameterized camera model. Even though the mount of the camera keeps the device in the same position for the duration of the experiments, the mount does not guarantee that it is completely level horizontally. Since the true position of the beacon compared to the camera is not recoverable by exact means, a misalignment of the camera in this setup would yield in data which is not comparable to the calculated and expected pose. A solution to this issue may be to initially detect a resting pose, followed by a registration of the offsets in the 6 DOFs compared to the expected pose, and finally using these offsets to align the reference frames.

Both implementations do track the relative change in the 6 DOFs reasonably well. In particular, the Tvrx implementation proved to be the better one, having the better accuracy, less noise and generally low variability in the estimated values. In this assignment, the particular angles of importance are the yaw and pitch, which were used to control the PTU. The results indicate that these angles were estimated the best for the performed experiments.

However, it is clear that the POSIT algorithm would perform better if both more feature points and more precise measurements of them were available, since the four feature points used to estimate the pose are the lowest possible number of points for use with this algorithm. On the other hand, the algorithm is very robust, as the estimates proved to be highly consistent for similar poses.

6.5 Future Work

An obvious issue with using the Wii remote as an imaging device is the fact that its properties are poorly documented. Thus, throughout the report, the effects of errors in the camera properties, such as lens distortion and image sensor defects, have not been studied. However, due to the nature of the implementation of the camera, with the integration of image analysis tools, a conventional calibration process is not easily carried out. In order to perform a calibration of the camera, a known pattern with easily distinguishable features has to be observed. This would yield in a large number of correspondences, allowing for the determination of the camera calibration matrix. With the increased accuracy, this system could potentially be used for more complex tasks, such as robot control.

The impact on the pose estimation accuracy due to the use of several filter designs should also be investigated. One of the implementations considered in this report used a Kalman filter for the smoothing of the estimated values, and this implementation was shown to be superior to the other, but more attention should be directed to the tuning of it. This could easily increase the accuracy of the implemented system. Furthermore, there are numerous approaches to the pose estimation problem, and thus, alternatives to the POSIT algorithm should be considered.

Although some techniques for camera control on pan-tilt platforms were discussed in Section 3.5, only a few of the approaches were implemented and tried out. And even if they did work satisfactory as a proof of concept, they should be further tuned and more methods should be examined as well.

Even though the created beacon can be mounted on the head of an operator by some means, little care was taken to this when its design was devised. Because of the required non-coplanarity of the feature points, the current design would be somewhat intrusive even if implemented on to goggles. A coplanar marker would possibly be easier to mount on goggles without it being disturbing to the user. This would however warrant significant changes in the software, by the change of both the pose estimation algorithm and the algorithm used to determine the point correspondences.

The usage of the input system on more suitable output devices should be studied as well. Even though the estimation of the full 6 DOFs of the beacon was performed, only the yaw and pitch angles were needed for the control of the PTU. There are several application which are able to employ the rest of the estimated values. Such an application could be the replacement of the mouse as the input device to a computer, and pointing by head orientation. However, if the beacon was not head mounted, this could also be used as a

3D mouse. By further coupling the input of the 3D mouse with a motion planning approach such as inverse kinematics could yield in intuitive control mechanisms for robot control. As previously mentioned, the head tracking is also interesting in stereo vision applications where it can aid the system to produce better image projections. There clearly are a lot of applications which could benefit from the system explored in this report

Chapter 7

Conclusion

In this report a vision based input device for motion control using head tracking was explored. The imaging device used was the infrared camera on a Wii remote, and it was combined with a trackable optical marker. By employing an algorithm for pose estimation, these devices made it possible to robustly determine the position and orientation of the marker in 3D space.

The results presented in the report indicate that the system is not entirely accurate, and thus, the absolute position and orientation of the marker is not perfectly estimated. This means that the system is not entirely suited for control of devices where perfect spatial information is required, without further tuning. However, the input device proved to be accurate enough for the control of a less complex system, namely a pan-tilt platform. Moreover, the system is very robust in terms of yielding consistent estimates for similar poses. This is a desired property, because faulty estimates are essentially undetectable to systems which rely on visual measurements. So, even though a very simple and highly optimistic mathematical model was used to describe the camera, the system could indeed be used for motion control through head tracking.

Appendix A

Pan-Tilt Unit

The pan-tilt device is produced by Directed Perception Inc., and consist of a pan-tilt unit, designated model number PTU-46-17P70T, and a controller, designated model number PTU-D46. We will only be interacting with the controller, and letting it interpret our commands and making the unit carry them out. The commands to control the unit have all been gathered from the user manual of the unit.

A.1 Serial Communication

The communication input to the controller a female DB-9 connector, accepting RS-232 connections with the settings in Table A.1 set.

Table A.1: Pan-Tilt unit settings

Setting	Value
Baudrate	9600
Start bit	1
Stop bit	1
Data bits	8
Parity	None
Handshaking	None

After a connection has been made, the communication is performed in human readable ASCII, and the commands to control the PTU are summarized in Table A.2.

Table A.2: Pan-Tilt unit control commands

Command	Description
PP	Query for current pan position
PP<number>	Set pan position to number
PN	Query for minimum allowable pan position
PX	Query for maximum allowable pan position
PS<number>	Set pan speed to number
TP	Query for current tilt position
TP<number>	Set tilt position to number
TN	Query for minimum allowable tilt position
TX	Query for maximum allowable tilt position
TS<number>	Set tilt speed to number
PR	Query for pan resolution
TR	Query for tilt resolution
C	Query for current control mode
CI	Set independent mode, where both velocity and position control is possible
CV	Set velocity mode

Table A.3: Queried Pan-Tilt unit properties

Property	Value	Angle
Maximum pan position	3048	156.7668°
Minimum pan position	−3048	−156.7668°
Maximum tilt position	2390	30.7310°
Minimum tilt position	−3608	−46.3923°

A.2 Serial Communication in C#

For the serial communication in C# the .NET class `SerialPort` was used, and Microsoft Developer Network [8] has the documentation on all the workings of this class. Listing A.1 shows how a communication port was opened using this class, while Listing A.2 and A.3 show how reading and writing is performed, respectively. The port is configured as the stated settings in Table A.1. The reading example is performed in a separate thread, and the read string is simply printed to the console. The writing example, on the other hand, is performed in the parent thread.

Listing A.1: Creation of the port

```
SerialPort serialPort = new SerialPort();
serialPort.PortName = "COM1";
// Following settings are from PanTilt Unit Manual
serialPort.BaudRate = 9600;
serialPort.Parity = Parity.None;
serialPort.StopBits = StopBits.One;
serialPort.DataBits = 8;
serialPort.Handshake = Handshake.None;

serialPort.WriteTimeout = 1000;
serialPort.ReadTimeout = 1000;

serialPort.Open();
if (!serialPort.IsOpen)
  quit = true;
```

Listing A.2: Reading from the port

```
Thread readThread = new Thread(ReadThreadFunction);

...

// Read function
public void ReadThreadFunction()
{
  while (!quit) {
    try {
string msg = serialPort.ReadLine();
Console.WriteLine(msg);
    } catch (TimeoutException) { }
  }
}
```

Listing A.3: Writing to the port

```
public void Write(string msg)
{
  serialPort.WriteLine(msg);
}
```

A.3 Serial Communication in MATLAB

In MATLAB, a serial port can be created using the `serial` function with all the settings from Table A.1 passed as arguments, while opening is done with `fopen`. This is shown in Listing A.4, along with the closing and deletion of the serial object. Writing is done using `fprintf`, while `fscanf` may be used to receive data. Reading and consequently writing to a port is shown in Listing A.5.

Listing A.4: Creation of the port

```
s = serial('COM1', 'BaudRate', 9600, 'DataBits', 8, ...
'FlowControl', 'none', 'Parity', 'none');

fopen(s)

...

fclose(s)
delete(s)
```

Listing A.5: Reading from and writing to the port

```
str = fscanf(s,'\%s');

fprintf(s, str);
```

Appendix B

Using the Wii Remote

B.1 Connecting to the Wii Remote

First, to connect the Wii remote to a computer, a compatible Bluetooth adapter is needed. Both a Cambridge Silicon Radio chip using the Toshiba Bluetooth stack and a D-Link DBT-122 adapter has proven compatible for this purpose, however, these are not the only ones which are compatible with the Wii remote. The connection process is similar to when connecting to a cellular phone, or any wireless, Bluetooth compatible, interface. The procedure for connecting to the Wii remote is as follows:

1. Start the Bluetooth software, and have it search for a device.

2. Enable Discovery mode on the Wii remote by pressing the buttons 1 and 2 simultaneously until the LEDs start flashing.

3. The Wii remote should now show up as a discovered device named **Nintendo RVL-CNT-01**.

4. Make a connection to the device. There is no need to use a security code or PIN number when connecting to the device.

In particular, it seems that the Wii remote is pickier with regards to the Bluetooth communication stacks, rather than to the particular adapters in use. In this regard, the Toshiba stack seems like the best out there, but as it is not readily available, it is advisable to check [4, 3, 23] before settling on a particular adapter.

B.2 Using the Wii Remote in C# with WiimoteLib

B.2.1 Accessing the Wii Remote

Here, we will look at what is needed code-wise to use the library WiimoteLib to access the Wii remote. A brief description of this library has been made in section 3.3.2. To properly sets up one Wii Remote allowing us to read the sensor, we need to setup an event to handle the state changes on the device. Listing B.1 how to create the `Wiimote` object, while Listing B.2 shows how to handle the events generated by the Wii remote.

Listing B.1: Creating an instance of the Wii Remote

```
using WiimoteLib;

...

  // Create a Wiimote object
  Wiimote wm = new Wiimote();

  // Create a WiimoteState object
  WiimoteState wms;

  // Setup event handler
  wm.WiimoteChanged += wm_WiimoteChanged;

  // Connect to the Wii Remote
  wm.Connect();

  // Set the desired report type
  wm.SetReportType(WiimoteLib.InputReport.IRAccel, true);
```

Listing B.2: Handling the events

```
void wm_WiimoteChanged(object sender,
    WiimoteChangedEventArgs args)
{
  // Store the state
  wms = args.WiimoteState;
}
```

B.2.2 Gathering Information from the Wii Remote

If the event handler is setup as previous mentioned, then whenever the Wii Remote reports a change, such as a button press, acceleration or other, we

catch it and store its state. The position and the size of the infrared sources are of particular importance and these are stored in the class `WiimoteState`. The members of `WiimoteState`, and their members are shown in Figure B.1.

`IRState` holds all the information about the infrared sensor, and what it has found. The variables `Midpoint` and `RawMidpoints` hold the midpoints of the first two points in the `IRSensors` array.

`IRSensors` is an array holding the states of the four infrared sources as they are detected by the sensor. The variable `Found` holds whether the point has been detected or not. Both `RawPosition` and `Position` hold its position, where the first is the unprocessed data as it is reported from the Wii remote, and its range is in $[0, 1023]$. The latter holds the normalized position of the detected point, in the range $[0.0, 1.0]$. `Size` describes how big the found object is. This value is in the range $[0, 15]$. Listing B.3 shows an example of retrieving the positions of the first detected IR LED, and storing them in local variables.

Listing B.3: Retrieving the positions of the IR points

```
float x = wms.IRState.IRSensors[0].Position.x;
float y = wms.IRState.IRSensors[0].Position.y;
```

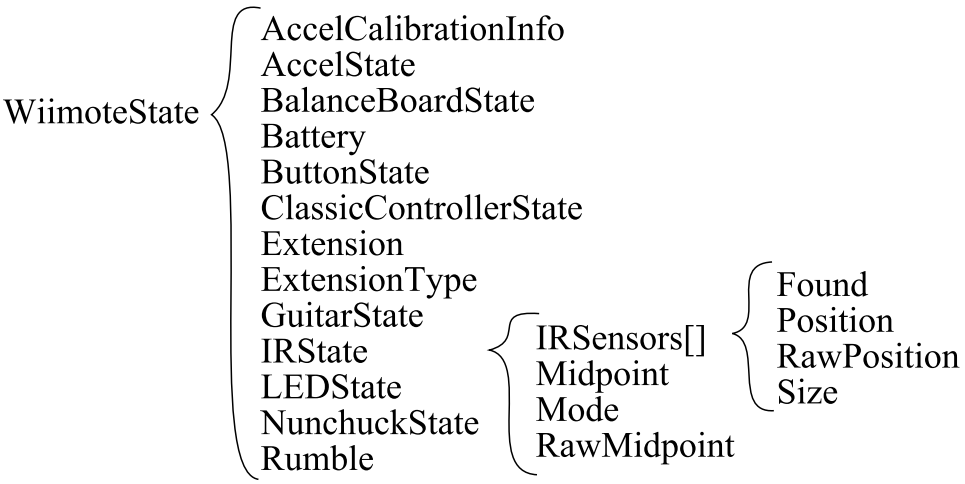


Figure B.1: The WiimoteState class structure expanded.

B.3 Using the Wii Remote in MATLAB with WiiLAB

WiiLAB [6] is a set of MATLAB functions which allow the use of the Wii remote in MATLAB. The actual communication with the device is done using WiimoteLib, but some work has been done to make it possible to access this library in unmanaged code such as MATLAB.

B.3.1 Using the Wii remote

A lot of MATLAB functions have been made to ease the use of the Wii remote. Table B.1 shows the most common functions. This list has been retrieved from [6].

Table B.1: Useful WiiLAB functions

Class	Function Name	Description
Connection	`initializeWiimote()`	Connects to and initializes the Wii remote
	`isWiimoteConnected`	Checks if any devices are connected
	`disconnectWiimote`	Disconnects and shuts down the Wii remote
Wiimote State	`isButtonPressed(button)`	Checks if `button` is pressed
	`getWiimoteAccel()`	Retrieves acceleration data from the device
	`getAccelData(seconds)`	
	`getWiimoteIR()`	Retrieves the IR values from the Wii remote
Feedback	`setWiimoteLEDs(led1, led2,led3,led4)`	Sets the LEDs to the desired values
	`setWiimoteRumble(on)`	Turns on rumble

In addition, the structure `wiimote` can be accessed if it has been declared as a `global`. The structure holds all the current states of the Wii remote, as of its last update. This allows for more custom usage of the device, which may not be possible through the use of the available functions.

Appendix C

Tvrx

C.1 Interfacing the Library

The library has a very simple interface, making it quite easy to use. All the other libraries are handled through Tvrx, including OpenCV, which is used for the pose estimation and filtering, as well as WiimoteLib, which is used to connect to the Wii remote. As the Wii remote is event driven, and the library creates an event handler for this event, the sensor readings will at all times be updated.

The library creates a singleton of type `TrackerManager`, which contains all the relevant transforms for the pose estimation. First the system needs to be initialized, using a call to the objects initialize method, as well as starting the pose estimation.

Listing C.1: Main methods in the library

```
// initialize
TrackerManager.Instance.Initialize();
TrackerManager.Instance.StartTracking();

...

// update
TrackerManager.Instance.Update();
...

// stop
TrackerManager.Instance.StopTracking();
```

After initialization, whenever a need to calculate and update the pose arises, a call to the update method has to be made. This call, based on

the currently detected points, also recalculates all the transform which were defined in the configuration file. The configuration file is treated more closely in Appendix C.2. The tracking algorithm can be stopped at any time by calling the `StopTracking` method, and this has to be called for a gracefully exit of the program too. All these method calls are summarized in Listing C.1.

As previously mentioned, whenever the update method is called, the library recalculates the pose of the marker. The transform may be accessed using the method `GetDeviceTransform`. This method returns a matrix of type `Microsoft.Xna.Framework.Matrix`, which is a 4×4 transformation matrix in homogeneous coordinates. The same can be done for the configured proxy. Listing C.2 shows both these calls. The parameter of `GetDeviceTransform`, and also for `GetProxyTransform`, is noteworthy here; the parameter has to match the *id* the device has been assigned in the previously mentioned configuration file. For the device transform, this setting is set as an attribute to `<TrackedDevice>`, while for the proxy transform, it is set in `<TrackerProxy>`.

Listing C.2: Get transform matrices

```
Matrix dtransform = TrackerManager.Instance.
    GetDeviceTransform(1);

Matrix ptransform = TrackerManager.Instance.
    GetProxyTransform(1);
```

C.2 Tvrx Configuration

The Tvrx library has quite a few configuration options, and even though most of the options are described in [14], some more options had to be altered in order to get the correct rotation and translation transformations. All the options are defined using `XML` in the configuration file `Config/tracker.xml`. And as such, the configuration file has to start with the standard `XML` description, before the environment is defined. Listing C.3 shows how this is done, and also how to close off the environment.

Listing C.3: Start and end of the configuration file

```
<?xml version="1.0" encoding="utf-8" ?>
<Tvrx>
```

```
...

</Tvrx>
```

The camera, with its make, may now be declared as a `TrackerCam` object. Here we are able to rearrange the axes, as well as state the camera's position and orientation. In our system we assume that the camera is placed in the center of the frame of reference, with no rotation, and thus is defined as in Listing C.4

Listing C.4: The TrackerCam object

```
<TrackerCam
  id="0" cameramodel="Wiimote"
  translation="0,0,0" rotation="0,0,0" scale="0.001"
  xAxis="x" yAxis="y" zAxis="z">
</TrackerCam>
```

Next, the geometrical setup of the custom marker has to define in the object `MarkerBody`. For the marker illustrated in Figure 4.7(a), the definition of the `XML` object is as in Listing C.5.

Listing C.5: MarkerBody object

```
<MarkerBody>
  <WiiMarkerBody id="5" name="Head Beacon"
                 nearClip="20" farClip="1500"
                 translation="0,0,0" rotation="0,0,0">
    <point3d value="5.5, 6, 4.5"/>
    <point3d value="49, 6, 19"/>
    <point3d value="91.5, 6, 4.5"/>
    <point3d value="49, 74.5, 27"/>
  </WiiMarkerBody>
</MarkerBody>
```

As this library supports tracking of multiple markers using multiple cameras, the markers have to be matched to their respective cameras. This is done in the `TrackedDevice` object. In addition, the filters which are to be used with these particular devices are defined here. Listing C.6 shows how our device is configured. Here, the `TrackerCamId` with $id = 0$ has been matched with the `MarkerBodyId` with $id = 5$, and a filter with the settings in $id = 4$ will be used for both rotation and the translation. Additionally, the library has been instructed to reverse the translation along the $x-$ and $y-$axes, and offset the rotation for yaw and roll by π. Additional offsets may be defined in `<LocalTranslation>` and `<LocalRotation>`.

Listing C.6: TrackedDevice object

```
<TrackedDevice id="1" type="WiiMote" rotation="True"
    translation="True">
  <MarkerBodyId>5</MarkerBodyId>
  <TrackerCamId>0</TrackerCamId>
  <ReverseTranslation>True,True,False</ReverseTranslation>
  <ReverseRotation>False,True,False</ReverseRotation>
  <LocalTranslation>0,0,0</LocalTranslation>
  <LocalRotation>0,3.1416,3.1416</LocalRotation>
  <WorldTranslation>0,0,0</WorldTranslation>
  <WorldRotation>0,0,0 </WorldRotation>
  <RotationFilterId>4</RotationFilterId>
  <TranslationFilterId>4</TranslationFilterId>
</TrackedDevice>
```

The filters with their parameters may be defined in the `Datafilter` object. The five filters shown in Listing C.7 have been defined by the author of the library, and the filter with $id = 4$ have been used in conjunction with the Wii remote.

Listing C.7: Filter definitions

```
<Datafilter>
  <PassThrough class="PassThroughFilter" id="0"/>
  <!--A value is hz of measure updates. for 60fps use
      0.01666 and for 30fps use 0.0333-->
  <Kalman class="KalmanFilter" id="1" A="1, 0.0333, 0, 1"
          measurement_noise_cov="0.05"
          process_noise_cov_1="0.0001"
          process_noise_cov_2="0.0001"/>
  <Kalman class="KalmanFilter" id="2" A="1, 0.0333, 0, 1"
          measurement_noise_cov="0.1"
          process_noise_cov_1="0.00001"
          process_noise_cov_2="0.00001"/>
  <Kalman class="KalmanFilter" id="3" A="1, 0.0333, 0, 1"
          measurement_noise_cov="0.3"
          process_noise_cov_1="0.0001"
          process_noise_cov_2="0.0001"/>
  <Kalman class="KalmanFilter" id="4" A="1, 0.005, 0, 1"
          measurement_noise_cov="1.0"
          process_noise_cov_1="0.0000001"
          process_noise_cov_2="0.0000001"/>
</Datafilter>
```

Furthermore, the library needs a definition of a proxy, which may be used to hold the complete calculated device rotation and translation. However,

in this system, the proxy would have the same pose as the marker. Listing C.8 defines our proxy.

Listing C.8: Proxy definition

```
<TrackerProxy id="0" type="HEADTRACKER" activity="Camera">
  <TranslationDeviceIds>1</TranslationDeviceIds>
  <RotationDeviceIds>1</RotationDeviceIds>
</TrackerProxy>
```

Appendix D

OpenCV

This chapter describes how the POSIT algorithm in OpenCV may be used to estimate pose, and the code needed to achieve that objective. Actually, the description is made for the C# wrapper library EmguCV, but the procedure is valid for the C/C++ OpenCV library if care is taken to the language and library differences.

The procedure covers the initialization and finalization of the estimation algorithm, as well as the update steps which yield the pose estimates.

D.1 Initialization

The creation of the POSIT object requires only two parameters; an array with the feature points which define the geometric setup of the optical marker, and the number of the feature points.

The n-feature points have to be stored in an $n \times 3$ `float` array with the positions given in any given units. When this is done, the method to create a POSIT object may be invoked, allowing it to return a pointer to the object. Listing D.1 shows an example on this for our particular marker.

Listing D.1: POSIT Initialization

```
float[,] points = {{5.5f, 6.0f, 4.5f},
                   {49.0f, 6.0f, 16.5f},
                   {91.5f, 6.0f, 4.5f},
                   {49.0f, 74.5f, 27.0f}};

IntPtr positObject = CvInvoke.cvCreatePOSITObject(
    testpoints, points.GetLength(0));
```

D.2 Update

For any given POSIT object, a pose may be estimated for a number of image points. However, the image points have to be compared to their respective optical feature point. So, any given image point has to be the projected representation of its respective physical feature point. In Listing D.2, we have assumed that the correct mappings are known, and the image point coordinates have been correctly stored in `imagePoints`. We also have to make sure that the image points are defined in a coordinate system where the origin is in the center of the image.

Furthermore, the termination criteria, the focal length of the camera and the variables in which to store the estimated rotation and translation are needed for the update step. As per DeMenthon and Davis in [11] the termination criteria is to determine when a pose is accepted, or when it has failed. The success criterion is the amount of change between the estimated poses in consecutive iterations, while the fail criterion is defined as an upper limit for the number of iterations. A complete update step is shown in Listing D.2.

Listing D.2: POSIT pose estimation

```
// declare variables
float[] rotationMatrix = new float[9];
float[] translationVector = new float[3];

// store the detected image points in correct order
float[,] imagePoints = {{u1, v1},
                        {u2, v2},
                        {u3, v3},
                        {u4, v4}};

// shift points around the principal point (u_p, v_p)
for (int i = 0; i < imagePoints.GetLength(0); ++i) {
  points[i, 0] -= u_p;
  points[i, 1] -= v_p;
}

MCvTermCriteria criteria;
criteria.type = TERMCRIT.CV_TERMCRIT_EPS | TERMCRIT.
    CV_TERMCRIT_ITER;
criteria.epsilon = 0.00001f;
criteria.max_iter = 100;

double focalLength = 1380;
```

```
CvInvoke.cvPOSIT(positObject, points, focalLength,
    criteria, rotationMatrix, translationVector);
```

D.3 Finalization

When finishing up the program, we need to tell OpenCV to release the allocated memory, which it has allocated for the use in the pose estimation for a POSIT object. This is easily done as shown in Listing D.3.

Listing D.3: POSIT finalization

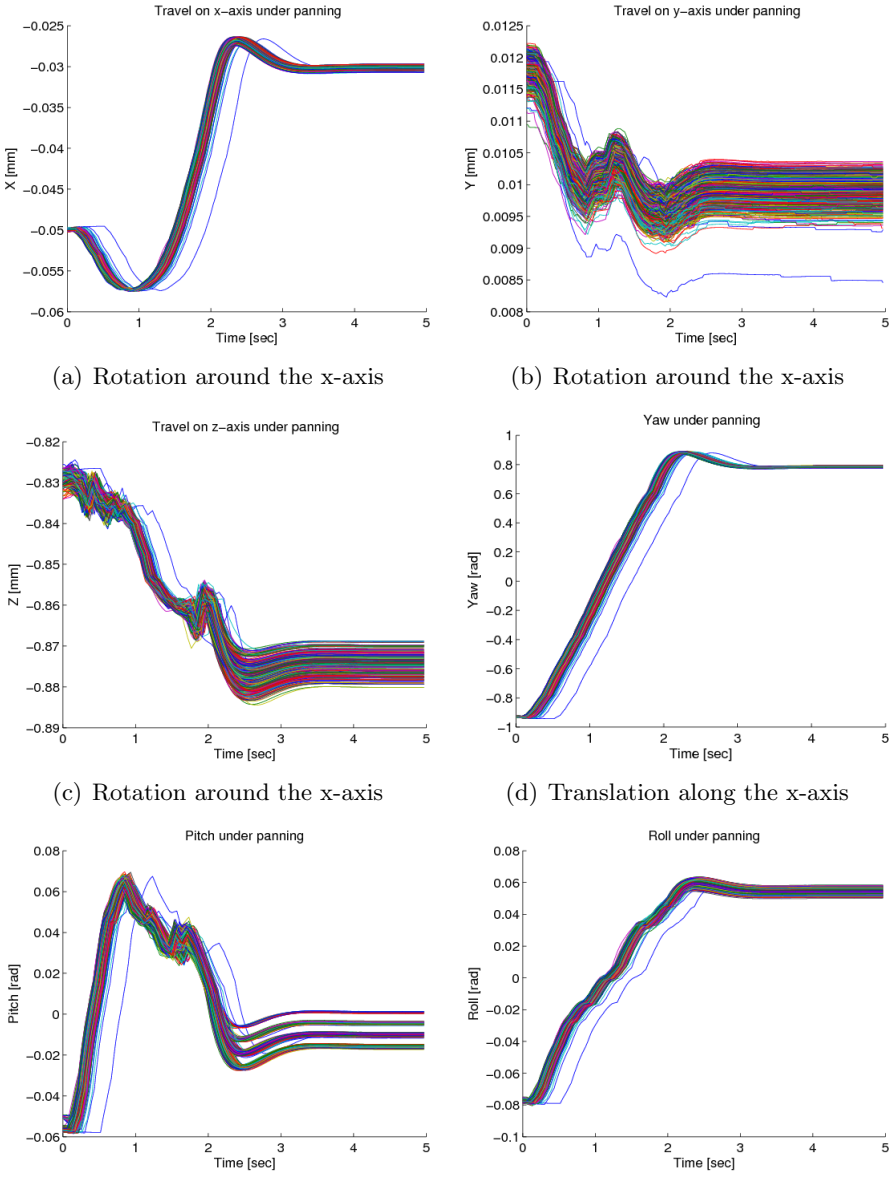
```
CvInvoke.cvReleasePOSITObject(ref positObject);
```

Appendix E

Additional Plots

In this chapter some of the additional results which were referred to in the report are organized. This includes the full plots of Experiment 3 and 4, including the pose estimates during the reverse motions.

E.1 Estimated Values For Forward Panning Motion



(a) Rotation around the x-axis

(b) Rotation around the x-axis

(c) Rotation around the x-axis

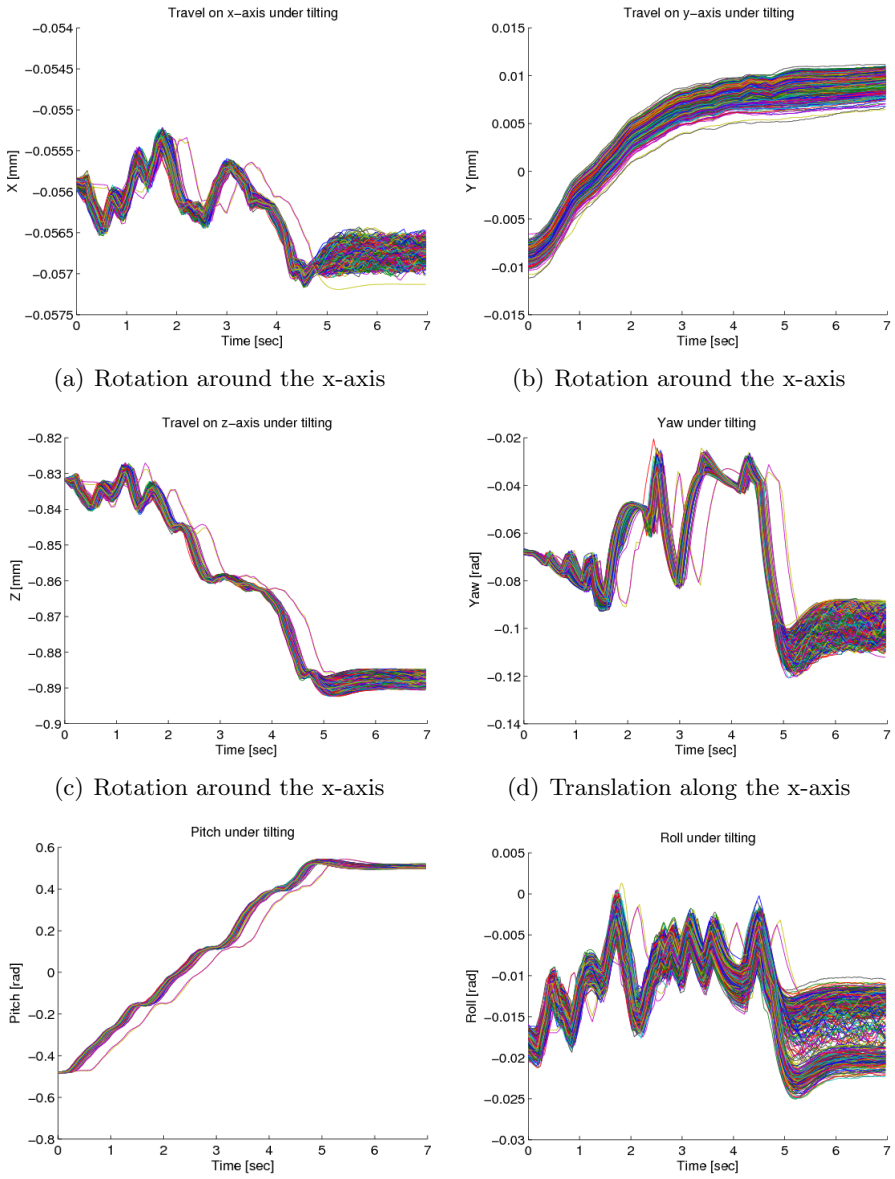
(d) Translation along the x-axis

(e) Translation along the y-axis

(f) Translation along the z-axis

Figure E.1: Estimates of pose parameters while panning repeatedly.

E.2 Estimated Values For Forward Tilting Motion



(a) Rotation around the x-axis

(b) Rotation around the x-axis

(c) Rotation around the x-axis

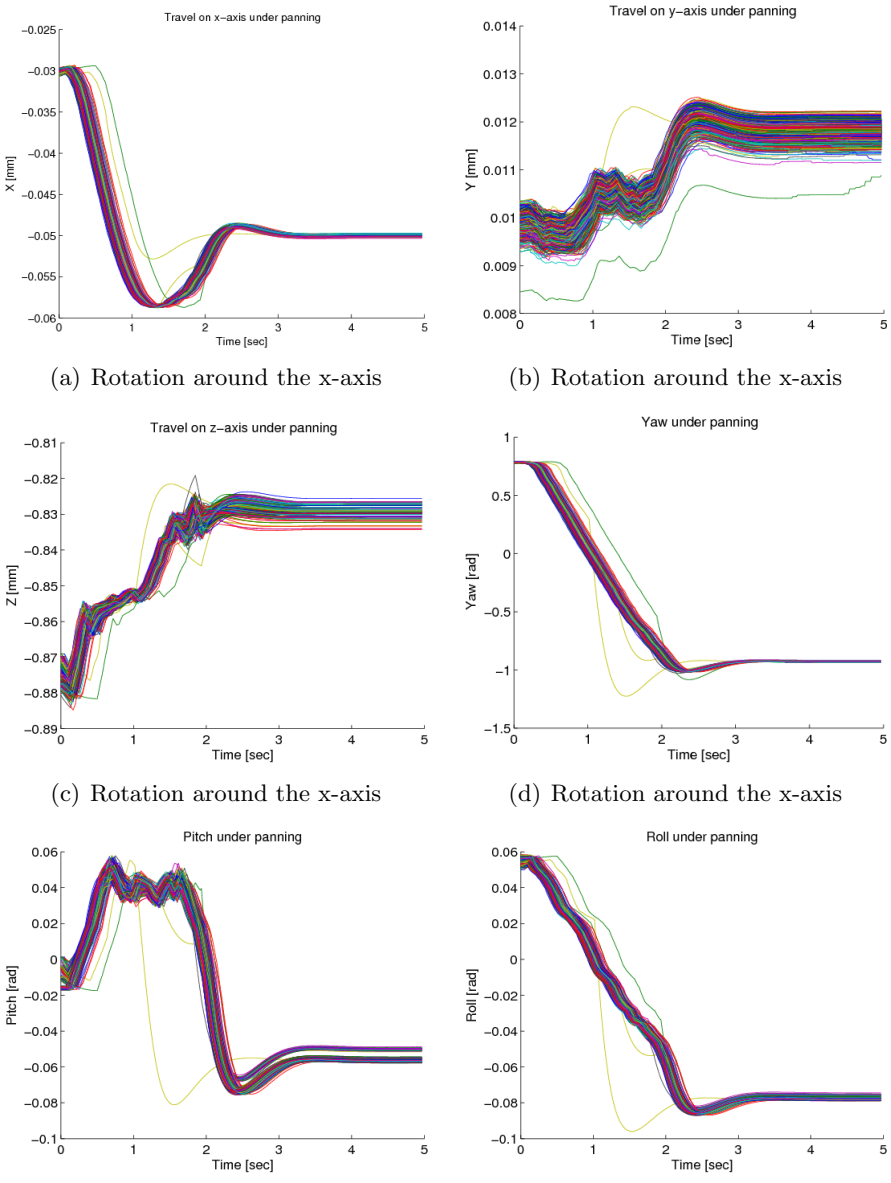
(d) Translation along the x-axis

(e) Translation along the y-axis

(f) Translation along the z-axis

Figure E.2: Estimates of pose parameters while tilting repeatedly.

E.3 Estimated Values For Reverse Panning Motion



(a) Rotation around the x-axis

(b) Rotation around the x-axis

(c) Rotation around the x-axis

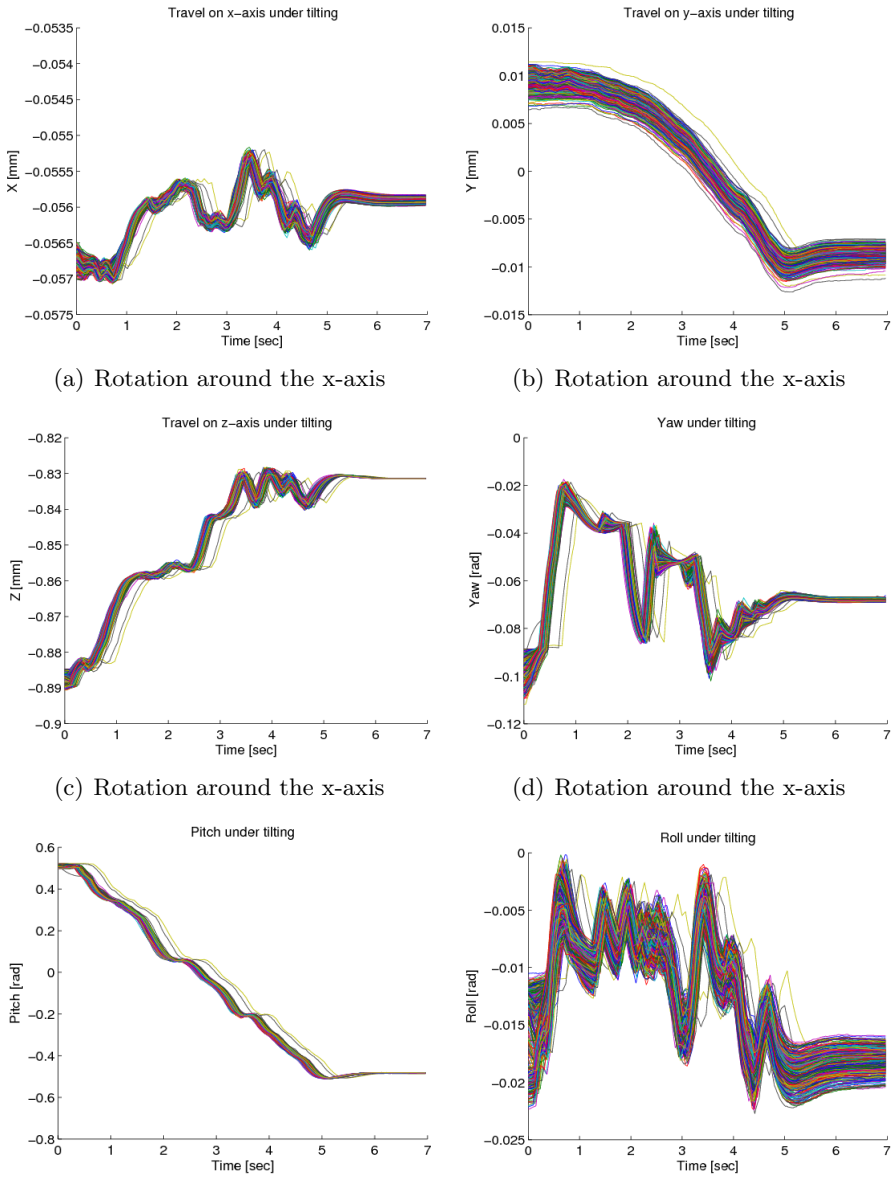
(d) Rotation around the x-axis

(e) Rotation around the x-axis

(f) Rotation around the x-axis

Figure E.3: Estimates of pose parameters while panning at low velocity.

E.4 Estimated Balues For Reverse Tilting Motion



(a) Rotation around the x-axis

(b) Rotation around the x-axis

(c) Rotation around the x-axis

(d) Rotation around the x-axis

(e) Rotation around the x-axis

(f) Rotation around the x-axis

Figure E.4: Estimates of pose parameters while tilting at low velocity.

Appendix F

Contents of the Attached CD

The directory structure of the attached CD is as follows:

- `/software/`
  All the software is included in this directory.
  - `/ext_lib/`
    The external libraries used in this assignment are placed in this directory.
  - `/vs/`
    The Visual Studio project including all the code is place here.
  - `/matlab/`
    The MATLAB specific code is included here.
- `/report/`
  This report in digital form is placed in this directory.

In addition, the README file included on the CD should be examined for more information.

Bibliography

[1] FreeTrack, `http://www.free-track.net`. [Online; Last Accessed March 2009].

[2] OpenCV, `http://opencv.willowgarage.com`. [Online; Last Accessed May 2009].

[3] WiiBrew wiki, `http://www.wiibrew.org/wiki/Wiimote`. [Online; Accessed January-May 2009].

[4] WiiLi wiki, `http://www.wiili.org/index.php/Wiimote`. [Online; Last Accessed January 2009].

[5] Adnan Ansar and Kostas Daniilidis. Linear pose estimation from points or lines. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25:282–296, 2002.

[6] Jordan Brindza, Jessica Szweda, and Aaron Striegel. WiiLAB TWiki, `http://netscale.cse.nd.edu/twiki/bin/view/Edu/WiiMote`. [Online; Last Accessed May 2009].

[7] Microsft Corporation. XNA Creators Club, `http://creators.xna.com`. [Online; Accessed January-May 2009].

[8] Microsoft Corporation. Microsoft Developer Network, `http://msdn.microsoft.com`. [Online; Accessed January-May 2009].

[9] K. Daniilidis, C. Krauss, M. Hansen, and G. Sommer. Real-time tracking of moving objects with an active camera. *Real-Time Imaging*, 4(1):3–20, 1998.

[10] E. R. Davies. *Machine Vision: Theory, Algorithms, Practicalities*. Elsevier, 2005.

[11] Daniel F. DeMenthon and Larry S. Davis. Model-based object pose in 25 lines of code. *International Journal of Computer Vision*, 15:123–141, 1995.

[12] Olav Egeland and Jan Tommy Gravdahl. *Modeling and Simulation for Automatic Control*. NTNU, 2nd. edition, 2002.

[13] Martin A. Fischler and Robert C. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, 24(6):381–395, June 1981.

[14] Timo Fleisch. VR homepage, `http://www.vrhome.de`, 2009. [Online; Last Accessed May 2009].

[15] A. H. W. Goh, Y. S. Yong, C. H. Chan, S. J. Then, L. P. Chu, S. W. Chau, and H. W. Hon. Interactive ptz camera control system using wii remote and infrared sensor bar. *Proceedings of World Academy of Science, Engineering and Technology*, 36:127–132, 2008.

[16] Radu P. Horaud, Bernard Conio, Olivier Leboulleux, and Bernard Lacolle. An analytic solution for the perspective 4-point problem. *Computer Vision, Graphics and Image Processing*, 47:33–44, 1989.

[17] Directed Perception Inc. Directed Perception - advanced pan tilt tracking gimbal, `http://www.dperception.com`. [Online; Last Accessed May 2009].

[18] PixArt Imaging Inc. Pixart Imaging Inc. homepage, `http://www.pixart.com.tw`. [Online; Last Accessed February 2009].

[19] Hirokazu Kato. ARToolkit, `http://www.hitl.washington.edu/artoolkit/`, 1999. [Online; Last Accessed April 2009].

[20] Oliver Kreylos. Oliver Kreylos Research and Development homepage - Wiimote Hacking, `http://idav.ucdavis.edu/~okreylos/ResDev/Wiimote/index.html`, 2009. [Online; Last Accessed April 2009].

[21] Joseph J. Laviola. Bringing VR and spatial 3D interaction to the masses through video games. *IEEE Comput. Graph. Appl.*, 28(5):10–15, 2008.

[22] Bum-Jong Lee, Jong Seung Park, and Mee Young Sung. Vision-based real-time camera matchmoving with a known marker. In *ICEC*, pages 193–204, 2006.

[23] Johnny Chung Lee. Johnny Chung Lee's homepage, `http://www.johnnylee.net/projects/wii/`, 2009. [Online; Accessed January-March 2009].

[24] Vincenzo Lippiello, Bruno Siciliano, and Luigi Villani. Adaptive extended kalman filtering for visual motion estimation of 3d objects. *Control Engineering Practice*, 15(1):123 – 134, 2007.

[25] Vincenzo Lippiello and Luigi Villani. Managing redundant visual measurements for accurate pose tracking. *Robotica*, 21(5):511–519, 2003.

[26] D. Murray and A. Basu. Motion tracking with an active camera. *IEEE Trans. Pattern Anal. Mach. Intell.*, 16(5):449–459, 1994.

[27] Denis Oberkampf, Daniel F. DeMenthon, and Larry S. Davis. Iterative pose estimation using coplanar feature points. *Comput. Vis. Image Underst.*, 63(3):495–511, 1996.

[28] Brian Peek. WiimoteLib, `http://www.wiimotelib.org`, 2007. [Online; Accessed January-May 2009].

[29] Natural Point. Natural Point, `http://www.naturalpoint.com`, 2009. [Online; Last Accessed March 2009].

[30] Long Quan and Zhongdan Lan. Linear n-point camera pose determination. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21:774–780, 1999.

[31] Ian F. Rickard and James E. Davis. Self-calibrating optical object tracking using wii remotes. *Sensors, Cameras, and Systems for Indistrial/Scientific Applications*, 2009.

[32] Gerald Schweighofer. Robust pose estimation from a planar target. *IEEE Trans. Pattern Anal. Mach. Intell.*, 28(12):2024–2030, December 2006.

[33] Raphael Wimmer, Sebastian Boring, and Johannes Müller. Tracking the wiimote in 3d using artoolkit. In *Proceedings of the Second Workshop on Mobile and Embedded Interactive Systems (MEIS'08)*, September 2008.