Intuitive Teleoperation of an Intelligent Robotic System Using Low-Cost 6-DOF Motion Capture

by

Jonathan Gagné

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Applied Science
in
Systems Design Engineering

Waterloo, Ontario, Canada, 2011

Author's Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

There is currently a wide variety of six degree-of-freedom (6-DOF) motion capture technologies available. However, these systems tend to be very expensive and thus cost prohibitive. A software system was developed to provide 6-DOF motion capture using the Nintendo Wii remote's (wiimote) sensors, an infrared beacon, and a novel hierarchical linear-quaternion Kalman filter. The software is made freely available, and the hardware costs less than one hundred dollars. Using this motion capture software, a robotic control system was developed to teleoperate a 6-DOF robotic manipulator via the operator's natural hand movements.

The teleoperation system requires calibration of the wiimote's infrared cameras to obtain an estimate of the wiimote's 6-DOF pose. However, since the raw images from the wiimote's infrared camera are not available, a novel camera-calibration method was developed to obtain the camera's intrinsic parameters, which are used to obtain a low-accuracy estimate of the 6-DOF pose. By fusing the low-accuracy estimate of 6-DOF pose with accelerometer and gyroscope measurements, an accurate estimation of 6-DOF pose is obtained for teleoperation.

Preliminary testing suggests that the motion capture system has an accuracy of less than a millimetre in position and less than one degree in attitude. Furthermore, whole-system tests demonstrate that the teleoperation system is capable of controlling the end effector of a robotic manipulator to match the pose of the wiimote. Since this system can provide 6-DOF motion capture at a fraction of the cost of traditional methods, it has wide applicability in the field of robotics and as a 6-DOF human input device to control 3D virtual computer environments.

Acknowledgements

First and foremost, I would like to express my gratitude to my supervisors, Prof. Chris Eliasmith and Prof. Jonathan Kofman, for their research guidance, academic advice, and acceptance in allowing me to carve my own research path. Furthermore, they patiently edited this thesis and provided valuable feedback.

I would like to thank my thesis committee readers, Prof. Hamid R. Tizhoosh and Prof. Eihab Abdel-Rahman, for their time and for their feedback. Moreover, I wish to express my appreciation to Prof. Eihab Abdel-Rahman for always making time to discuss nonlinear dynamics with me in his office.

I would like to thank my family for their continuous support. It was in part through their encouragement that I gained the confidence to take on challenges like this thesis and to persevere, no matter how difficult times get.

Finally, I would like express my appreciation for the financial support provided by the University of Waterloo through their President's Scholarship and by NSERC through their Alexander Graham Bell Canada Graduate Scholarship.

Without all your help, this thesis would not be possible.

Dedication

I would like to dedicate this thesis to my parents and grandparents.

Without their support, it would have been impossible to get to where I have.

Table of Contents

List of Figures

Chapter 1 – Introduction

1.1 – Problem Statement, Motivation, and Purpose

There is a need for an intuitive and low-cost six degree of freedom (6-DOF) pose estimation system that can be used to teach robots and interact with 3D software.

1.1.1 – Motivation for Intuitive and Low-Cost 6-DOF Motion Capture System for Teaching Robots

There is huge potential for service robots to perform trivial tasks for humans and thereby increase the efficiency of human activity by freeing up time that would otherwise be spent on these tasks. Before service robots can reach this potential, the intelligence of robots must increase and the cost must decrease. Addressing the intelligence problem is a highly active research area. However, for intelligent robots to become widespread, the robot cost must become affordable by a wide audience. More research is required to allow robots to become more affordable. The primary purpose of this research is to address this cost problem. Specifically, this research focussed on the development of a very low-cost motion capture system, which is used in a programming-by-demonstration framework, to reduce the cost of teaching robots new tasks.

In 2010, the worldwide market value for robotics was $3.2 billion USD for service robots (up 15% from 2009) [1], $5.3 billion USD for industrial robots, $17.5 billion for industrial robotic systems [2], and is expected to continue to increase [1][2]. The potential for service robots is widely recognised, “a robot in every household”; however, it has been a challenge for the industry to produce attractive low-cost-for-value solutions [1]. Before robots can reach their potential of aiding people in every household, they must become low-cost enough that the average household can afnot ford them. A substantial amount of work is required to adequately address this issue. Furthermore, as intelligent robots become able to learn a wider variety of tasks, it will become increasingly time consuming to teach them the numerous tasks required. It thus becomes important to have an easy and low-cost method to teach them.

Having an intuitive teaching method is useful in an industrial setting since it will reduce the robot programming costs resulting from requiring highly skilled labour performing time-consuming

programming. An intuitive teaching method is of even greater importance in the domestic setting because most individuals who could benefit from next generation service robots will not have the time or technical knowledge for low-level programming. For service robots to become widely commercialized, it is crucial that effective teaching solutions are found [3]. Furthermore, as they become intelligent, versatile, and more capable of serving humans, there will be tremendous benefit to having robots that are capable of learning new tasks directly from the people that use them, which will often be the untrained public.

For a robot to perform a task, a robotic manipulator of some kind is often required to modify its environment. The robot must be programmed to use its manipulator to complete each task. One of the most intuitive and time-efficient methods of programming task information is programming-by-demonstration (PbD), which is the process of teaching a robot a new task by having an operator perform the task. For the robot to learn a new task by PbD, the motion and state of the human's manipulator, which is typically the hand, must be known to the robot throughout the period while the task is being demonstrated. Therefore, some form of 6-DOF motion capture of 3D position and 3D orientation (attitude) is necessary. Unfortunately, 6-DOF motion capture technology can often cost thousands or even tens of thousands of dollars. The research described in this thesis addresses this limitation. A low-cost 6-DOF motion capture system using the sensors in the Nintendo® Wii™ remote (colloquially called the *wiimote*) was developed. The developed software enables readily available devices that cost less than $100 to be used. This dramatically reduces the cost of motion capture technology and reduces the cost of teaching robots new tasks via PbD.

While this thesis focuses on the contribution of the developed motion capture system to the field of robotics, its contribution is much broader, since it has general applicability as a human-computer interface for virtual 3D environments.

1.1.2 – Motivation for an Intuitive and Low-Cost 6-DOF Motion Capture System for 3D Computer Software

There currently exists a variety of 6-DOF motion capture technologies available. However, these systems are costly and therefore have not gained widespread use outside of a few domains. The computer mouse is an analogous 2-DOF motion capture device that has become so widespread that it is

used to interface with nearly every personal computer worldwide. As a result, it is indirectly useful to every computer use with few exceptions.

The computer mouse has become widespread primarily because it satisfied the following needs for:

1. Reliable motion capture for use in human-computer interaction
2. Sufficient accuracy
3. Sufficiently low cost that it is affordable by the public

The development of graphical user interfaces (GUI) and high quality 2D computer displays created a need for a 2D human interface device (HID) that satisfies Needs 1-3.The creation of such a 2D HID allowed the further advancement of 2D GUI, and other 2D devices. The computer mouse became prominent since it satisfied the first two needs while being best at satisfying the third. As computer systems and related devices become increasingly capable of processing and displaying 3D spatial information, there is increasing potential for 3D interface devices.

When a 2D object is moved in 2D space projected by a computer monitor, the perspective is unchanged as seen by the human viewer; however, when a 3D object is moved in 3D space, the object's perspective changes with respect to the viewer. Consequently, when manipulating objects in 3D there is a fourth need:

4. Manipulation of perspective

Satisfying the fourth need requires manipulation of 3D orientation (attitude). There are commercial 3D input devices, which are 3-DOF that satisfy the first three needs, but do not completely satisfy the fourth. Moreover, there are commercial 3D input devices that are 6-DOF that satisfy Needs 1, 2, and 4, but do not satisfy Need 3. There is a lack of HIDs that satisfies all four needs. Widespread availability of such a device would be useful to researchers, consumers, and industry in a wide variety of domains either indirectly though the expansion of commonly available 3D GUIs or directly in fields that require direct human-motion-capture. If such a device (or future design iterations) were to become low cost and widespread like that of the computer mouse, it could have a direct impact on the development of 3D technology and indirectly on any system that takes advantage of a 3D GUI.

This thesis has a general and a specific purpose. The general goal is to describe the developed software that interacts with the wiimote to provide a 6-DOF HID. This allows this device to satisfy Needs 1-4 and opens up accessibility to all researchers, industry, and the general public.

The specific purpose of this thesis is to detail a system that interfaces this 6-DOF HID and a robotic manipulator, allowing teleoperation of the robot via human-hand movements.

1.2 – System Overview

The developed motion capture system was specifically designed to augment current human-style teaching systems developed by Wu [8]. The human-style teaching system uses teleoperation from a marker-based stereo-vision system to teach trajectories to a robotic system. The trajectories are provided along with human speech aided task segmentation to enable the robot to learn simple pick-and-place and other simple tasks.

The previous stereo-vision system required that the cameras be rigidly affixed to a stable structure (e.g. the wall) in positions that allow good visibility of the hand of the human operator (teacher), and required calibration when their position was moved or disturbed. This made mobility of the system difficult and time consuming. To solve this problem, the motion capture system was replaced with a portable teleoperation system based on a hybrid-inertial-vision motion-capture system developed in this research.

The teleoperation system is composed of a Kalman-filter based 6-DOF motion capture system using a wiimote and a beacon with four infrared LEDs on it. On the software side, the system is divided into two subsystems, the Wiimote to Robot Interface and the Robot Control Server. The Wiimote to Robot Interface uses Bluetooth to connect to the wiimote to obtain the wiimote's sensor measurements and other state information. The Wiimote to Robot Interface collects the accelerometer measurements, gyroscope measurement, and 2D locations of the LEDs on the wiimote camera and sends these values over a TCP/IP connection to the Robot Control Server. The TCP/IP connection allows teleoperation to be performed at remote sites. Upon receiving the incoming measurement, the Robot Control Server sends the information to the pose filter to determine the location of the wiimote in space.

The pose filter is a hierarchical stochastic filter composed of position filters and an attitude filter, which estimates 3-DOF position and 3-DOF attitude, respectively. The position filters are based on a standard Kalman filter, and the attitude filter is a multiplicative extended Kalman filter, which is a quaternion-based extended Kalman filter. The position filter optimally estimates the 6-DOF pose of the

wiimote at roughly 100 Hz to obtain a 6-DOF trajectory where each pose in the trajectory is the pose that maximizes the a priori probability of all previous sensor values.

A human operator can move the wiimote in a 6-DOF trajectory and the estimated pose of the wiimote is used to teleoperate a robotic manipulator in real time. This effectively serves as the teaching portion of a PbD system.

1.3 – Contributions

This thesis contributes to multiple aspects of the 3D interaction problem. It provides incremental, but critical advancements in several areas by reducing cost and development time, and hence allows more research to be performed in the related areas. The contributions are presented roughly from low-level to high-level, where each higher level requires and is an extension of the previous lower level. The six contributions are:

Transfer of research from guidance, control, and dynamics to the robotics field.

Literature on the multiplicative extended Kalman filter is predominately found in the aerospace and navigation fields, where it is used for optimal estimation of a craft's attitude. Since the applied mathematics of these algorithms can become quite involved, it may be difficult for researchers in other fields to take advantage of the algorithm without a strong control and mathematical background.

Typically, documentation on the multiplicative extended Kalman filtering (MEKF) assumes advanced understanding of Kalman filtering, stochastic modelling, numerical methods of integration, and quaternion calculus. Furthermore, derivations are often unclear, and notation is often inconsistent between authors. To the best of the author's knowledge, there exists no published work that contains a complete treatment of the multiplicative extended Kalman filtering (MEKF) assuming only basic prior knowledge of probability theory and linear systems of ordinary differential equations.

This thesis rigorously details the MEKF and provides relevant background knowledge in a complete, technically accurate manner using consistent notation. Furthermore, derivations for many of the MEKF equations are provided in detail.

Novel pose filter

Optimal estimation of position and attitude are generally addressed independently. However, much of the research that does address them simultaneously is designed for different sensors, is computationally expensive, or uses a less precise algorithm. Due to its effectiveness, the multiplicative extended Kalman filter is the most commonly used attitude estimation algorithm used onboard NASA spacecraft [5] and has considerable potential in other fields. With the use of the multiplicative extended Kalman filter for attitude, three linear Kalman filters for position, and a framework to integrate the individual filters, a novel 6-DOF pose filter was developed.

Using a novel calibration method and the pose filter, the first publicly available software system that is capable of producing 6-DOF motion capture using the wiimote was developed

Gaining precise pose for the wiimote's infrared camera is difficult since the camera calibration parameters are unknown. Moreover, standard calibration techniques are difficult since the user does not have access to the camera's image, but instead, only has access to the 2D relative centroid locations of (up to) the four brightest infrared lights that have an intensity greater than a pre-specified threshold (q.v. Sect. 7.4). A novel calibration technique was developed that merges multiple images taken of a precision-machined and microcontroller-controlled matrix of infrared LEDs to acquired image data that is used to find the camera's intrinsic parameters. Following calibration, the camera can be used to find the pose of beacon with respect to the camera.

Using this calibration method and the novel pose filter, the first publicly available software was developed that is capable of calculating stable 6-DOF pose from the wiimote sensors.

Very low-cost motion capture system

Development of a stable 6-DOF pose estimator is a contribution to the field of motion capture. As discussed in Sect. 2.2.1, stable 6-DOF motion capture can cost thousands and even tens of thousands of dollars. At less than one hundred dollars, this motion capture system can be built, thus greatly reducing system cost and allowing it to find use in many other areas where cost may have previously made use of the technology prohibitive.

Novel teleoperation system

The development of novel a motion capture system is also a contribution to the field of telerobotics. As low-cost robots become increasingly common, cheaper methods of operating them become increasingly important to allow widespread adoption.

A novel system for the teaching portion of PbD

Teaching robots tasks can be a costly and involved process. Intuitive 6-DOF motion capture devices are costly, and to reduce costs, the teach pendent is generally used instead. The teach pendant's trajectories are rarely optimal, velocity information cannot be directly controlled, and it is more difficult and less intuitive to use. However, cost is often the largest constraint, and despite the large limitations of the device's performance, its low cost has allowed it to become the most commonly used device to teach robotic-manipulator systems.

Use of a 6-DOF motion capture device for teleoperation would circumvent the teach pendant's drawbacks; however, the cost of 6-DOF motion capture has prevented widespread adoption. A low-cost 6-DOF teleoperation device would solve this and therefore would be a better alternative to the teach pendant in almost every situation.

1.4 – Thesis Overview

The thesis will detail the entire system and related material as follows. Chapter 2 reviews the literature on robotic programming-by-demonstration, literature on motion capture as it pertains to robotics, and reviews the competing commercially available motion capture systems. Chapter 3 provides the technical details on the hardware used by the developed system.

Chapter 4 reviews background knowledge required by Chapter 5. Specifically, Chapter 4 will described the mathematics involved in attitude kinematics, quaternion algebra and calculus, forward kinematics, inverse kinematics, gyroscope kinematics modelling, and a camera model.

Chapter 5 rigorously details Kalman filtering theory, beginning with the basic linear Kalman filter, then the extended Kalman filter, and finally builds up to the multiplicative extended Kalman filter, which is an extended Kalman filter developed over the $\mathbb{S}^3$ manifold. The Kalman filter equations are explained

in detail because outside of the simplest cases, usage of the Kalman filter requires full understanding of the equations in order to incorporate the system mechanics that is being near-optimally estimated.

Chapter 6 details the design and implementation of the complete teleoperation system. The Wiimote to Robot Interface and the Robot Control Server are developed. Of particular importance, is the development of a novel 6-DOF pose filter, which is composed of three linear Kalman filters, one multiplicative extended Kalman filter and a framework to combine the two. Furthermore, Chapter 6 describes how the wiimote, Wiimote to Robot Interface, Robot Control Server, pose filter subsystem, and robot controllers interact to produce the teleoperation used for teaching by programming-by-demonstration.

Chapter 7 details the calibration of the system, with particular emphasis on the calibration of the wiimote's camera. A novel calibration technique is employed using a custom-made calibration board. Through the calibration, the intrinsic parameters of the camera are obtained, which allow the camera to be used to estimate 6-DOF absolute pose.

Chapter 8 describes three experiments that verify the functionality of the filter, and one subjective experiment demonstrating that the entire system is fully functional. Finally, Chapter 9 provides discussion of the system and concluding remarks.

Appendix A is reserved for mathematical properties used by Chapters 3 and 4, and Appendix B provides derivation of the zeroth- and first-order quaternion integrators, which are used to solve the state-propagation differential used by the multiplicative extended Kalman filter.

Chapter 2 – Literature Review

2.1 – Robotic Programming by Demonstration and Related Work

2.1.1 – Introduction to Robot Programming Methods

Before a robot can complete a task, it must first be programmed to do so. Its program can be created manually by a programmer or it can be generated automatically through human-robot interaction. Manual methods can be costly since they often require considerable time and highly trained personnel. Automatic techniques can often circumvent these obstacles. As surveyed by [7], automatic techniques can be subcategorized into learning systems, instructive systems, and programming by demonstration (PbD).

Robotic *learning systems* (inductive learning systems) use examples and self-exploration to create the robot's program; however, a large number of examples or trials are typically needed [8]. This has been accomplished by hierarchical neural networks [9], chaining simple behaviours together to form larger ones [10], and reinforcement learning as proposed by Smart and Kaelbling [11]. *Instructive systems* create tasks by combining previously known simpler tasks, which have been accomplished using gestures [12][13][14], and by natural language [15]. *PbD*, also known as learning from observation, learning from demonstration, or learning by imitation [16], is performed by having the robot learn to perform the task by watching a human operator perform the task. Since the robotic system in this research must learn from minimal examples, and simple primitive tasks must be learned, not supplied, inductive learning systems and instructive systems literature is not relevant. Hence, only PbD systems are reviewed.

PbD systems require both teaching and learning. The focus of this research is to upgrade a previously built PbD system with a new teaching device and leave the learning method unchanged. PbD related learning methods (see [8] for review) are outside the scope of this research and therefore, only PbD teaching methods are reviewed. Teaching is performed through robotic teleoperation via a human operator.

2.1.2 – Teaching Methods Used in Programming by Demonstration

Robotic Programming by Demonstration (PbD) is the process of programming robots to perform tasks by having a human manually demonstrate the task to the robot. The task can be demonstrated in the robot's reference frame by directly controlling the robot, or it can be demonstrated in the human's reference frame and the robot must infer what the motion should be from its own perspective. Additionally, the task may be demonstrated by physically moving the robot, by teleoperating the robot, or by directly demonstrating the task as the robot watches. Teleoperation is the process of direct manual control of a robot over a distance, which in the narrow sense is the control of a telerobot using one-to-one motion capture from the teleoperator and in the broad sense is the control of the telerobot using any remote control interface by the teleoperator.

Methods of teaching robots tasks by humans can be categorized intro three groups: teaching by guidance (TbG), teaching by passive human demonstration (TbPD), and human-style teaching (HST) [7][17][18]. TbG uses either direct mechanical control or teleoperation using non-one-to-one methods of control in the robot frame. TbPD uses one-to-one motion capture in either the robot frame or operator frame. Finally, HST is an extension to TbPD where the robot additionally interacts actively with the operator as opposed to being a passive learner; however, this extension does not directly relate to teaching the motion trajectories. Since the research focuses on the teaching of task trajectories to robot, the extensions of HST over TbPD are outside the scope of this work.

2.1.3 – Teaching by Guidance

Teach by Guidance (TbG) is a method where the operator controls the robots trajectory and other relevant operations, such as opening and closing a gripper or turning on and off a tool, with a teleoperation method in the broad sense. Since it is a teleoperation-type method, control commands are with respect to the robot frame. It is a simple and low-cost method and thus has gained widespread use in industrial robotics.

Teleoperation in TbG is often performed by a teach pendant, which is a remote control device that allows simple functions such as open gripper, move end effector along x-axis, rotate Joint 5, etc. Since it is the simplest and cheapest teleoperation device, it is arguably the most common device used in industrial robotics. Consequently, its simplicity has major drawbacks, being that trajectories are

rarely optimal and velocity information cannot be directly controlled. Furthermore, it the least intuitive and most constrained, which can make complex task-trajectories difficult or impractical.

To overcome these limitations, more intuitive and effective devices have been created [8], such as a 6-DOF force sensor [19], 6-DOF force mouse [20], force-moment direction sensors, joysticks [22][23][24] and kinaesthetic techniques [25]. Additionally, teaching methods have been used such as hand-gesture-based [26], graphical-based [27] and virtual reality [28][29]. A quantitative evaluation of these teaching methods using a force-moment direction sensor is performed by Choi and Lee [30]. These devices and techniques have allowed TbG trajectories to be smoother, more intuitive and easier to use with respect to the teach pendant; however, they can still be difficult to use and greatly lack the intuitive operation offered by motion capture devices that record natural hand movements.

2.1.4 – Teaching by Passive Human Demonstration

Teaching by Passive Human Demonstration (TbPD) is a method that allows a human operator to teach a task to a robot using natural human movements. This is accomplished by either the operator physically performing the task in the human frame as the robot watches in the robot frame [31][32][33][34][35][36] or via one-to-one teleoperation (narrow sense) in the robot frame [37][38][18]. TbPD can be discriminated from TbG in that TbPD uses one-to-one motion capture of natural movements to teach, whereas TbG uses any other device not capable of one-to-one motion capture.

TbPD can be categorized into two methods: (1) where the human directly performs the task with its own hands or body directly and then the robots watches and infers how it should perform the task, and (2) where the human indirectly performs the task by teleoperating the robot to perform the task itself.

2.1.4.1 – Teaching by Human Passive Demonstration via Direct Methods

Direct methods of TbPD are performed by teaching the robot a task by natural movements, and having the robot passively watch and infer how the task should be performed from its own perspective. This is analogous to an instructor teaching a quiet student a motor skill by demonstrating it. As is discussed at the end of this subsection, direct methods currently have major limitations that require addressing before it can become a robust method. Therefore, this research does not use this method and only

discusses it for completeness. For further details on common characteristics and steps of this method, see [31][39][33][40][8].

For the robot to observe the motion of the human demonstrator, a sensing modality is required. The most commonly used modality is vision [31][33][35][41][42][24][43]. However, sensing and communicating has evolved into multi-modal systems [7][44] that may use tactile and position sensors; force, torque, speech, and inertial measurement devices, virtual reality, and radio-frequency-identification enabled gloves [45][46][47][48][43][36].

The advantage of TbPD is that it is a natural way for a teacher to teach a skill. However, it has the significant drawback in that current state-of-the-art robots have very limited inferencing abilities. Since the robot must use inferencing to determine how it should perform the task from its own perspective [49] with a different physical body and different constraints, this makes task learning via direct methods very challenging [50][51]. Furthermore, before the task can be optimally learned, the motion correspondence between the operator and robot must be solved, which is in itself a challenging problem [52][25][53]. Discussed in the following subsection, one-to-one teleoperation of the robot to perform the task can be used to circumvent these inferencing and correspondence problems.

2.1.4.2 – Teaching by Human Passive Demonstration via Teleoperation Methods

A simple and effective method of teaching a robot tasks is through teleoperation of a robot by using one-to-one motion capture of natural human movements, capturing either the hand [54][55][18] or of multiple points on the body [37][38].

Peters et al.[37] used a full-immersion telemetry suit equipped with magnetic sensors to teleoperate the NASA Robonaut to learn tool-usage skills. Motion capture of the fingers was provided by the data gloves and the position of the arms and head was captured by 6-DOF Polhemus magnetic field sensors [56] attached to the data glove and head. Lieberman [57], and Lieberman and Breazeal [38] teleoperated a humanoid robot “Leonardo” using a mechanical motion-tracking suit to perform push-button tasks. The mechanical suit measured 40 joint angles using potentiometers and gyroscopes. Wu, Kofman, and collaborators [8][54][55] used motion capture from a marker-based vision system to teleoperate a robotic manipulator to perform pick-and-place tasks.

TbPD via teleoperation is more intuitive than TbG, and it is less intuitive than TbPD via direct methods. However, TbPD via teleoperation circumvents the challenging inferencing and correspondence problems of TbPD via direct methods. This dramatically reduces the cognitive requirements of the robot and simplifies the project implementation. Furthermore, trajectories can be consistently made near optimal, and errors in trajectories and task learning can be greatly reduced, which is important when very specific trajectories are required. Lastly, teleoperation methods have a further advantage over direct methods in that the operator does not need to be at the physical location of the robot. The task can be taught from any connected remote site, which is important when the robot is operating in a hazardous environment, or when it is not practical or efficient for the operator to travel to the robot's location. For these advantages, TbPD via teleoperation using one-to-one motion capture is the chosen teaching method used in this research.

The technical details of the motion tracking systems mentioned, along with the competing technologies are discussed in Sect. 2.2. Moreover, the pros and cons of each system are presented to allow the optimal selection of the motion capture technology to be used in teleoperation.

2.2 – Related Commercial Products and Motion Capture Literature

This section reviews the motion capture technologies that provide 6-DOF pose (3-DOF position and 3-DOF attitude). The technologies are described and a review of currently available commercial systems is provided with system costs when available. Cost is included as an important factor in the review of available commercial products in order to demonstrate the dramatic reduction in the cost of the system designed in this research. Finally, in each section, a review of the motion capture technology in the field of robotics is provided.

All prices listed below *do not* include shipping costs, taxes, import duty charges, and cost of installation. Furthermore, they are at the date of publication (2011), and in USD.

2.2.1 – Inertial-Based Tracking with an Inertial Measurement Unit (IMU)

The inertial measurement unit (IMU) is an integrated system of sensors that detects accelerations and angular rates. It is commonly used in large-scale tracking and is an integral unit in inertial navigation

systems (INS) in watercraft, aircraft, spacecraft, and guided missiles. The 6-DOF variant is comprised of a tri-axis accelerometer and a tri-axis gyroscope, and when combined with a tri-axis magnetometer it is referred to as a MARG (Magnetic, Angular Rate, and Gravity) sensor.

The accelerometer detects accelerations with respect to (wrt) an inertial reference frame. Relative position is obtained by accurately aligning the acceleration with gravity, subtracting gravity, and integrating the accelerations twice wrt time. For low-cost systems such as those used in this research, the primary source of error is from subtraction of misaligned gravity vectors. The secondary sources are from discretization of the analog signal, unknown stochastic influence not caused by accelerations, poor calibration, and modeling of linear and nonlinear deterministic influences, which are not completely static. Obtaining relative position through double integration causes errors to grow exponentially and without correction by absolute position sensors, the error is unbounded. The specific accelerometer used in the research is detailed in Sect. 3.2.1.

The gyroscope detects angular rates wrt to itself. Relative attitude is obtained by integrating the angular rate once, which incurs cumulative error. The specific gyroscope used in the research is detailed in Sect. 3.2.2. The absolute attitude is obtained by taking a measurement of a tri-axis accelerometer when it is at rest wrt to the earth, however, only 2-DOF can be determined, since the plane orthogonal to the gravity is not observable. Obtaining stable 3-DOF absolute attitude requires a tri-axis magnetometer be added to the IMU. The magnetometer determines absolute attitude by measuring the earth's magnetic fields. However, it is highly susceptible to fluctuations in magnetic fields, which makes coupling with gyroscopes useful.

IMU/INSs can be grouped in varying grades: marine-grade, aviation/navigation-grade, intermediate-grade, tactical-grade, and automotive/consumer grade [58]. *Marine-grade* INS is used in ships, submarines, and some spacecraft. They have a drift of less than 1.8 km/day; however, they can cost in excess of a million dollars (CDN/USD) and are too large for teleoperation. *Aviation/Navigation-grade* INS drift ~1.5 km/hour and cost approximately 100,000 dollars. *Intermediate-grade* IMU is an order of magnitude less accurate and costs between 20,000-50,000 dollars. *Tactical-grade* IMU is a stand-alone INS for up to a few minutes, and costs between 5,000-20,000 dollars. Finally, *automotive/consumer-grade* IMU is not used on their own for any navigation or tracking, but for a variety of other consumer purposes.

Quality IMUs have the advantage that they can provide one-to-one tracking. They have the disadvantage that they only provide relative position and only full absolute attitude with the coupling of a magnetometer. On their own, they have a prohibitively large disadvantage that they quickly incur exponential error and are only useful for very short periods when measured accelerations are considerably larger than gravity. For these reasons, even the highest-grade IMUs available cannot be used on their own for pose tracking.

Many companies produce such sensor systems. Trivision [65] supplies the Colibri ($1,170) and Colibri Wireless ($1,975), which contain tri-axial accelerometers and gyroscopes. However, these sensors are only considered 3-DOF (attitude only) when not combined with other non-inertial motion capture sensors. Other more precise sensors such as MotionPak ($10,000) by Systron-Donner [66] claim to provide 6-DOF pose detection, although, this motion is not stable and therefore cannot be used for an extended period.

Although individual IMUs cannot be used on their own for motion capture, they can be combined rigidly by a suit, where IMUs record 3-DOF attitude of body landmarks, and the angles between them are recorded. Following calibration of the system such that the distances between the sensors become known, the suit can then be used for motion capture.

Animazoo [67] offers full-body commercial motion capture suits IGS-150 ($20,000), IGS-180 ($45,000), and IGS-190 ($62,000), containing 15, 17, and 19 MARG inertial sensors, respectively. Similarly, Xsens Technologies' [68] MVN (estimated at $50,000-$100,000 as of 2009), is a full-body commercial motion capture suit containing 17 inertial MARG sensors. MVN is primarily an inertial system, however, as of roughly 2010 [69], it can be also be aided by MotionGrid, which is an ultra-wideband RF-based position tracker.

Inertial-based motion capture suits have been used to control the joints of articulated robots. Miller et al. [70], designed a MARG sensor suit to teleoperate NASA's Robonaut humanoid robot. Field et al., [71] used a 16 MARG sensor Xsens Moven (former name of MVN), to track human motion for humanoid motion planning.

2.2.2 – Acoustic-Based Tracking

Acoustic-based trackers estimate position by triangulating ultrasonic sound. Microphones and ultrasonic transmitters are attached to specific locations on the moving body and generally to other fixed locations. Distance is determined by the change in amplitude or phase shift of the pulsed signal. The amplitude method calculates distance using a model of the drop in sound intensity over distance, and the phase-shift method uses the speed of sound and temporal phase-shift resulting from the time-of-flight. Multiplexing the pulses allows tracking of multiple points, and when a minimum of three noncolinear points are known, relative attitude is observable.

Acoustic-based systems require line-of-sight, since occlusion can induce considerable error resulting from decreased amplitude and increased time-of-flight that does not correspond with increased absolute distance between the transmitter and receiver. Furthermore, accuracy is affected by temperature, pressure, humidity, and by interference from other sounds near the frequencies of the pulses. Moreover, electromagnetic interference from electronic devices such as televisions, mobile phones, personal computers, and power lines can interfere with the ultrasonic pulse transmission [72].

Ultrasonic sensors have been used to obtain 3-DOF position in research such as Loke [72]; however, attitude is not obtained. It is possible to attach additional sensors to a rigid body being measured to obtain attitude, thus providing a 6-DOF pose estimation, although, this is problematic because line-of-sight is required, which is difficult to obtain from multiple sensors simultaneously. Range of motion is limited and as soon as line-of-sight is lost, pose accuracy will degrade quickly. Furthermore, it may be difficult for the system to detect that its line-of-sight has been lost.

The Hexamite HX11 [73] system provides modulated acoustic signals that are general used for larger areas, with the systems from $1,800 (for 20 m^2) to $120,000 (for 100,000 m^2). They only natively support 3-DOF position; however, in principle it could be modified with custom software to support 6-DOF with the drawbacks stated above.

The Logitech 3D [74] mouse contains three receivers mounted on the front of the mouse allowing 6-DOF pose. It is also subject to the line-of-sight problem; however, the position of the sensors helps to minimize the line-of-sight problems as long as the mouse points in the vicinity of the transmitters. The basic system retails at $3,600.

2.2.3 – Optical-Based Tracking

Optical tracking uses multiple cameras to locate markers (or features for markerless systems) to triangulate position. Tracking position requires that at least one marker is visible in a minimum of two cameras. Since only position is natively supported, tracking attitude requires that a minimum of three noncolinear markers are visible in at least two cameras.

Optical tracking is accomplished using passive markers, active markers, or markerless systems.

2.2.3.1 – Passive Markers

Any marker that does not emit its own signal is considered a passive marker; however, commercial systems generally use infrared (IR) reflective indicators. The workspace is illuminated with IR light, which is reflected off the indicators and identified by image processing of the high-speed cameras. Markers are indistinguishable from one another and thus require post-processing to discriminate between them so that they can be tracked.

Passive marker systems have the advantage that they can have sub-millimetre accuracy at frames of up to 2 kHz [59]. They have the disadvantage that markers need to be reapplied prior to each use, portability is difficult, line-of-sight is required, occlusions are common, multiple markers are required to determine attitude, and the systems are costly.

Vicon [75] produces the Vicon MX passive motion capture system. Similarly, MotionAnalysis Corp [76] produces a few systems (Osprew, Owl, Raptor Series).

Motion capture from Vicon has been used various robotic research applications. Dsgupta and Nakamua [77] used the Vicon-370 passive marker system to acquire human-gait motion, which provided a basis for driving the locomotion of a humanoid robot. Shon et al. [78] used a Vicon system to teach a humanoid robot via programming by demonstration from the HumanEva datasets [79][79].

The HumanEva datasets are used for benchmarking the state-of-the-art in human motion tracking and were created using passive marker systems. Specifically, the HumanEva I dataset was created from a Vicon system with six 1M-pixel cameras and HumanEva 2 was created from the Vicon MX system with twelve 1.3M-pixel cameras. For a thorough list of the major publications using this technology, see [81].

Additionally, an exhaustive list of the research literature using MotionAnalysis Corp.'s passive motion capture systems can be found at [82].

2.2.3.2 – Active Markers

Active marker systems typically employ IR light emitting diodes (IR-LEDs). The advantages and disadvantages are the same as passive marker systems, with the following exceptions. Since the emitters are often synchronized with the cameras and time modulated, computation and tracking errors are reduced. Consequently, position-update rates are *n* times slower, where *n* is the number of active markers. Furthermore, these systems can be bulky and subjects must wear power packs and secure wires that may impede motion [59].

There are a number of options for this technology. 3rdTech's [83] HiBall 3100 (Price dependent on application) is based on the Wide-Area Tracking research project [84][85][86].

Ascension's [87] ReActor 2 ($85,500.00) is an untethered active optical tracking system for full body motion capture. It requires setup of a motion capture stage of 3m x 3m x 2.4m (Model 332), therefore size can be prohibitive. Northern Digital Inc.'s [88] Optotrak Certus ($57,400) is a high accuracy system (0.1 mm) and designed for research (as claimed by the manufacture [89]). An Optotrak system was used by Nasksuk et al. [90] to capture human motion data, which used modified balance and angular-momentum schemes to produce human motion in a humanoid robot.

2.2.3.3 – Markerless

Markerless system use computer vision techniques to extract relevant features, allowing motion tracking without the use of markers. Current systems do not provide the same level of accuracy as marker systems or require restricted environments. It will take a considerable amount of time for markerless systems to fully mature due to the presence of many unsolved problems, however, this area has considerable potential within the next few decades.

Organic Motion Inc. [92] provides a commercial motion-capture system that operates in a restrictive indoor environment (reflective cloth background) with the use of 14-18 cameras. Organic Motion Stage retails at $88,000.

Zhenning and Kulic [91] designed an 11-DOF upper body markerless motion capture system for a humanoid robot application. Kofman et al [55] used markerless tracking of the human arm to teleoperate a robot manipulator. Markerless system development is an active research area. For a review of related markerless motion capture research see Zhenning and Kulic [91].

2.2.4 – Mechanical-Based Tracking

Mechanical capture systems determine pose by measuring body joint angles and calculating the forward kinematics. An exoskeleton-like rigid structure is affixed to specific locations of an individual's body and electromechanical potentiometers measure the displacement of joint angles.

The advantages of mechanical systems are that they are portable and are not subject to occlusion. The disadvantage is that they can be bulky, restrictive to natural motion due to joint flexibility, and can only detect motion from the joints that are monitored by the exoskeleton. Furthermore, only relative pose can be determined and thus other systems are required for complete absolute pose.

Animazoo Gypsy 6 ($30,000) is a mechanical motion capture suit that detects 17 joint angles, and its successor the Gypsy 7 ($8,000) detects 14 joint angles. Despite both suits containing two gyroscopes used to detect ribcage and head motion, they are primarily mechanical suits and they do not rely on sensor fusion to increase accuracy as a hybrid system would.

Sarcos' [93] Sensuit is a 32- or 35-DOF mechanical motion capture suit that uses hall effect sensors instead of potentiometers to detect joint angles. Ijspeert et al. [95] used a Sensuit to evaluate an adaptive dynamical system-based movement imitation for a humanoid robot. Gray et al. [96] used a Gypse suit to control the Leonardo robot (63-DOF humanoid) in social robotics research where the goal was to aid a robot in understanding the goals and mental state of a human teammate. Ruiz-del-Solar et al. [97] used an Animazoo Gypsy-5 suit to teach a soccer playing humanoid robot how to fall in order to minimize joint/articulation injuries.

Calion et al., [94] used feedback from the motor encoders of a Fujitsu HOAP-2 humanoid robot to record motion used in a PbD system. This differs from the other research in that a human physically moved the robots arms and the robot recorded the path as opposed to a human wearing a suit with encoders that are used to teleoperate the robot. In this strategy, the motion capture and the robot

movement must occur in the same physical location, thus, long distance teleoperation is not possible for this method.

2.2.5 – Magnetic-Based Tracking

Magnetic capture systems determine pose by using receivers to detect magnetic field pulses produced by transmitters attached to relevant points on the body. Tri-axial transmitters emit precise pulses sequentially and the receiver subtracts the earth's magnetic field allowing 6-DOF absolute position, where the earth's magnetic field is estimated while the system is not transmitting.

Magnetic systems can detect pose though many types of materials, thus a line-of-sight is not required. Metallic objects can heavily distort magnetics fields and induce considerable error in AC magnetic systems; however, recent DC systems significantly reduce distortions [59].

Ascension [98] provides many magnetic solutions: SpacePad ($1,495), driveBAY ($3,740), trakSTAR ($3,945), microBIRD ($7,490), medSAFE ($8,990). Also note, Ascension's popular Flock of Birds system has been replaced by the upgraded trakSTAR system. Ascension's 6D Mouse ($795) and Wanda ($2,560) are 6-DOF motion sensors, however, they are not stand-alone system and require a previously purchased system such as Flock of Birds, which adds thousands of additional dollars to their cost.

Similarly, Polhemus [99] provides various magnetic trackers: Patriot ($4,022), Fastrak ($6,350), and Liberty ($10,632). Listed prices for Ascension and Polhemus products are for the basic one-sensor system, and increase with additional 6DOF tracking sensors. Lastly, VR-Space [100] sells Wintracker ($1,790), which is a lower-cost, three sensor system.

Ascension's MotionStar system has found considerable use in robotic research. Molderhauer et al. [101] used MotionStar to capture the trajectories of a human performing tasks such as setting the table, pouring water into a cup, etc. The data are then used by classification methods to identify the subject and the particular task they are performing. Matsunaga and Oshita [102] used MotionStar with a combined support vector machine and state machine to recognise walking motions. Additionally, Ramana et al. [103] used MotionStar and a hidden Markov model to classify four actions: pointing, grasping, rotating, and displacing.

2.2.6 – Hybrid System Tracking

Each type of capture system has weaknesses that can be circumvented by combining multiple types of systems, allowing a more accurate and robust system in dynamic environments. Consequently, this increases the cost of the system when using high-grade components. However, with the recent availability of low-cost sensors, these systems have potential to provide acceptable with the use of stochastic sensor fusion of low-cost, low-grade components. Despite this potential, low-cost commercial systems are not currently available for general use.

Ascension's [98] Hy-BIRD ($22,000) is a hybrid optical-inertial system. It is a helmet mounted tracking system containing an IMU and a laser sensor that detects signals continuously emitted from a surface-mounted laser scanner. Intersense [104]IS-900 is a hybrid acoustic-inertial system. Price ranges depending on implementations, but the basic PC system that comes with one tracker retails at $10,000.

Vlasic et al. [105] developed a hybrid acoustic-inertial motion capture suit. Tri-axial gyroscopes record angular rates, tri-axial accelerometers record acceleration, and piezoelectric transduces and microphones capture distance. The gyroscopes, accelerometers, and acoustic transduces are fixed to multiple points of interest on the human body. Motion was found to be reasonably accurate, however, the frame rate was only 10 Hz and pose estimation is processed off-line from previously stored sensor data, which makes the system not useful for PbD in its current implementation.

Nguyen et al. [106] used a custom-built hybrid fibre-optic inertial system to control a robotic puppet. The system used four fibre-optic sensor tapes that provide bend and twist information [107] for the limbs and inertial sensors to detect orientation of the thorax and pelvis. Ward et al. [108] used a hybrid acoustic-inertial system to provide continuous activity recognition of activities such as sawing, hammering, filing, drilling, grinding, sanding, etc. Two microphones and two tri-axial accelerometers were placed on the wrist and arm. Zhao and Badler [109] used a hybrid optical-magnetic capture system and neural networks to model human motion. The magnetic potion of the system used MotionStar and the optical system used processed video from two Kodak ES310 cameras.

2.2.7 – Very Low-Cost Motion Tracking

This review has demonstrated that there are a large number of systems that provide 6-DOF motion capture. Furthermore, these systems have sufficient accuracy for most robotic applications. However,

current systems suffer from one major obstacle; they are *all very costly* and range between $1,495 to over $100,000. Even the cheapest system is sufficiently cost prohibitive to prevent widespread adoption.

There is a system that overcomes this cost-prohibitive barrier; however, the company supplying it uses it solely for its video game console. The company does not provide software to access the sensors, nor does it provide the advanced algorithms required to optimally fuse the data from the low-cost sensors. Specifically, Nintendo [110] has overcome this cost barrier with an optical-inertial motion capture system, which it uses as its remote control for its Wii console [110]. The sensor system is the Nintendo Wii remote with MotionPlus (wiimote) and the accompanying infrared sensor bar, which combined costs less than $100 thus allowing for a 94% reduction in cost from the second cheapest option and a 99.9% reduction in cost from the most expensive options.

The wiimote system is based on a tri-axial accelerometer, a tri-axial gyroscope, and an infrared camera that detects the location of four infrared points on the sensor bar. Individually, the low-cost sensors are too inaccurate for 6-DOF tracking; however, using quaternion-based extended Kalman filtering, which is primarily used in aerospace research, a stable 6-DOF motion tracking system can be created. This system is sufficiently accurate for use in many 3D interface and robotic application, and is significantly cheaper than the alternatives.

Sony's PlayStation Move [111] is also a hybrid (MARG/Vision) 6-DOF motion capture system for use with its PlayStation 3 video game console; however, a PC interface to the Move's sensors is not yet available as of the date of writing this literature review. The teleoperation system designed in this research can be easily adapted to use the PlayStation Move once an interface is fully developed to access its sensors.

Chapter 3 – System Hardware

This research uses a standard PC, an infrared beacon system, an A465 Articulated Robot System, a Nintendo® Wii™ remote controller (wiimote), and an Arduino microcontroller board. The PC is described in Chapter 7, the beacon system and its construction are described in Chapter 6, and the robot system, wiimote, and Arduino board are detailed in the remainder of Chapter 3.

3.1 – A465 Articulated Robot System

The A465 Articulated Robot System consists of the A465 robot arm and a C500C controller. Both were originally designed and manufactured by CRS Robotics Corporation®, although Thermo CRS Ltd® currently owns the rights and production of the system.

Commands are issued to the C500C controller from a host PC via a straight-through RS-232 connection using the ActiveRobot© Library. Once received by the C500C controller, the commands are interpreted and any motor control to and feedback from the A465 robot is sent through the umbilical cables, creating a closed control loop.

3.1.1 – A465 Robot Arm

The A465 robot arm is a six-joint robotic manipulator as shown in Figure 3.1. It is capable of performing 6-DOF motion on a 2 kg payload with $\pm 0.05\ mm$ repeatability, and has dimensions and a range of motion as shown in Figure 3.2.

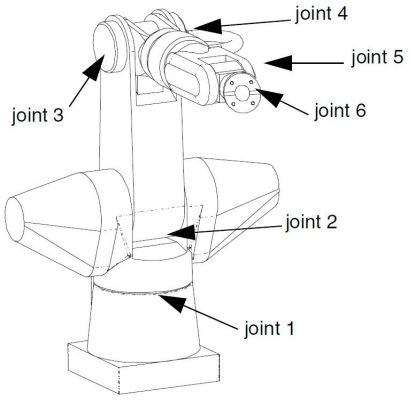


Figure 3.1. The A465 robotic arm with joints labeled (without end effector).
(Image obtained from the A465 Robotic System User Guide)

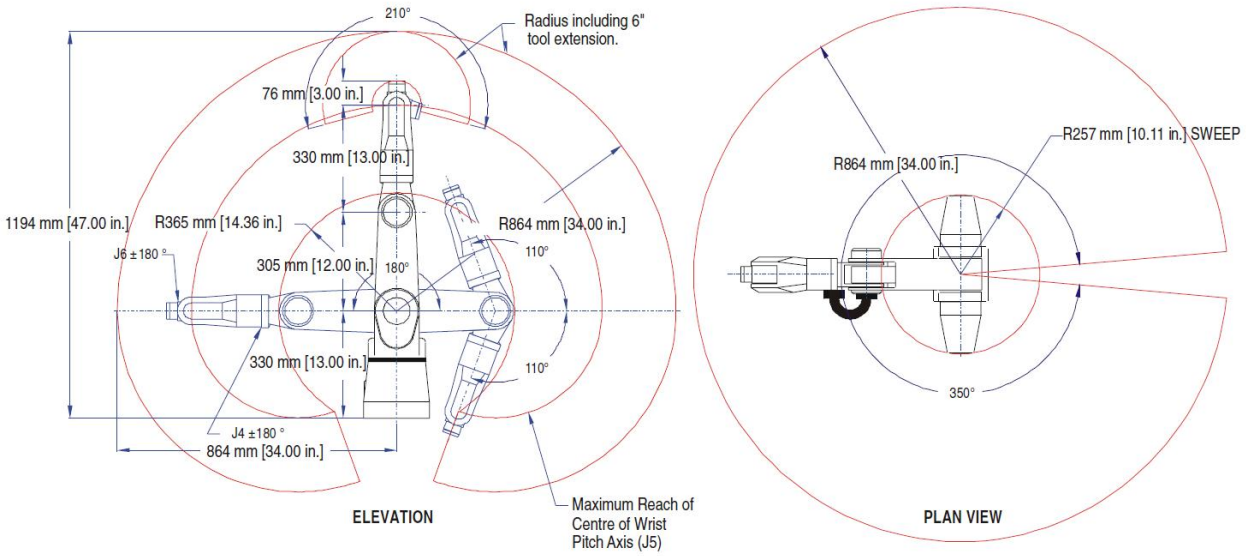


Figure 3.2. The range of motion and dimensions of the A465 robot arm (without end effector). (Image obtained from the A465 Robotic System User Guide)

3.1.2 – C500C Controller

The C500C controller (Figure 3.3 and Figure 3.4) provides power, safety circuits, and motor control for the robot arm. Given an end-effector pose, it calculates the inverse kinematics and drives the joint motors, and incorporates the encoder feedback. Additionally, it detects potentially damaging conditions such as robot runaway, sever collisions, loss of positional feedback, and errors in communication. It contains a 133 MHz i486DX (system processor) and a 60 MHz TMS320C31 DSP (motion control processor) and runs a proprietary real-time multi-tasking operating system (CRS Robot Operating System).

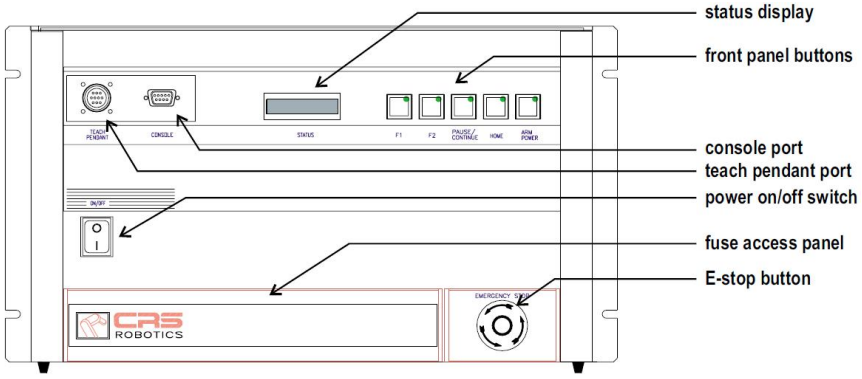


Figure 3.3. Front panel of the C500C controller.
(Image obtained from the A465 Robotic System User Guide)

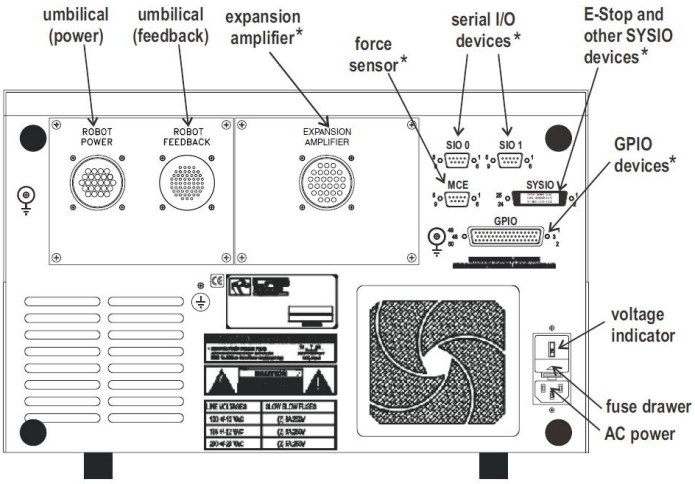


Figure 3.4. Back panel of the C500C controller.
(Image obtained from the A465 Robotic System User Guide)

Robot movement is performed by a motion command to the controller, which specifies what moves and where it moves to, and whether the motion is relative or absolute. The most common of these commands is an end-effector motion to a given pose. Individual motors can be controlled, but they cannot be controlled simultaneously, which makes low-level user control difficult. Besides the optional blending used to smooth trajectory transitions, a trajectory must be completed before another motion command can be initiated. If commands are sent faster than the associated trajectories can be completed, a queue of commands arises and leads to an unacceptable amount of motion latency. To ensure that the motion of the robot arm reasonably matches that of the teleoperator, commands must be sent just prior to the end of the completion of a trajectory.

3.1.3 – ActiveRobot

PC control of the A465 articulated robot system is enabled by a Microsoft Windows® ActiveX application via ActiveRobot. It allows applications to perform diagnostics, homing, calibration, configuration, and motion control of the A465 robot arm.

3.2 – Wiimote

The inertial measurement unit (IMU) and the infrared (IR) camera are the primary hardware used for teleoperation in this research, and provide relative positioning and absolute positioning, respectively. Specifically, the Nintendo® Wii™ remote controller with the Wii Motion Plus™ attachment were used, as they combine to provide a low-cost integrated unit that is widely available. This measurement hardware

will be referenced by its colloquial name, the wiimote, which for this research will refer specifically to the Wii™ remote and the Motion Plus™ attachment.

The Nintendo® Wii™ console uses the wiimote along with an IR LED sensor bar beacon to control the motion of avatars within its games via constrained teleoperation. It is important to note that Nintendo® does not use the wiimote to perform stable 6 degree-of-freedom (DOF) motion, but instead uses cues from the motion detected by the sensors to activate constrained actions. As described in subsequent chapters, a software system that uses the wiimote along with a modified IR LED beacon is developed, which provides stable 6-DOF pose teleoperation (3-DOF in position and 3-DOF attitude).

The wiimote takes user input from twelve buttons, a tri-axial accelerometer, a dual-axial gyroscope, a single-axial gyroscope, and an IR camera, all of which are sampled at 100 Hz. It also provides output to the user by four LEDs, a small rumble motor used to produce vibrations, and 21 mm piezo-electric speaker. Both input and output are transmitted wirelessly to the Wii™ console via a Bluetooth communication interface. Further details of the accelerometer, gyroscopes, camera, and Bluetooth interface are discussed in the following paragraphs. For further details on the other components, see [112].

3.2.1 – Accelerometer

The wiimote contains an ADXL330 integrated circuit (Figure 3.5), which is a low-cost, low power, MEMS tri-axial analog accelerometer made by Analog Devices®. It is designed to sense 3-DOF linear acceleration over $\pm 3g$ and is relatively stable over an operating temperature range of $-25°\text{C}$ to $+75°\text{C}$. It is manufactured for motion and tilt-sensing, but due to its low accuracy, it is not considered navigational grade and not designed for dead-reckoning applications. For detailed specifications, see the ADXL330 datasheet [113].

The wiimote uses the accelerometer over approximately $\pm 5g$ range, and although the ADXL330 provides an analog output, the wiimote discretizes the signal. The wiimote reserves 10 bits of resolution for the x-axis, and 9 bits of resolution for the y- and z-axes, where the x-, y-, and z-axes are shown in Figure 3.6. This is problematic since accelerometer information is sent in discrete units of $0.0943\ m/s^2$, $0.196\ m/s^2$, and $0.189\ m/s^2$, for the x, y, z-axes, respectively. Even without consideration of the original analog accuracy, the digitized resolution prevents the use of dead reckoning beyond a very short period.

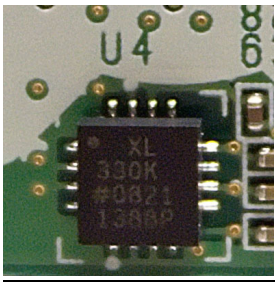


Figure 3.5. The ADXL330 tri-axial accelerometer IC (obtained from [112]).

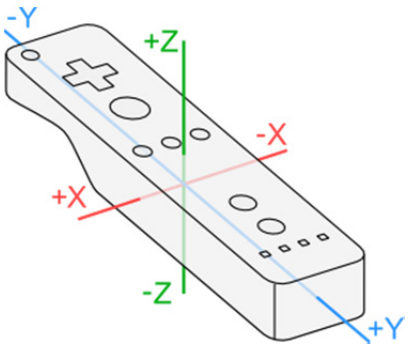


Figure 3.6. The linear axes of the wiimote as sensed by the accelerometer (obtained from [112]).

3.2.2 – Gyroscopes

The wiimote contains two gyroscopes, which together provide 3-DOF angular rate information. The first is an integrated dual-axis gyroscope (IDG-600) based on MEMS technology and produced by InvenSense®. Although the specifications state that it performs auto-calibration, and it also appears that the wiimote transmits 32 bytes of this information via Bluetooth, it is currently unknown how to use this information [114].

The second rate sensor is a single-axis gyroscope (X3500W) based on MEMS technology and produced by Epson Tbyocom®. Its specifications are not available publically, although it is believed to be similar to Epson Tbyocom's® XV-3500CB sensor [114].

The gyroscope measurements are subject to an additive bias, which varies slowly after an initial transience. This must be estimated to obtain reasonable measurements.

The gyroscope's signal is discretized to a unit size of $0.05\ deg/s$. Since this discretization unit size is small, the gyroscope's noise can be empirically analyzed. A simple methodology was used to

determine the properties of the noise. The wiimote was placed on a stable surface, initialized, and the readouts from the gyroscope were taken. The first 130 samples were ignored to remove warm-up transience and to allow a partial stabilization of the bias. Following transience, 4000 samples were collected and analyzed.

Axis 1 and Axis 2 were obtained from the IDG-600 dual-axis gyroscope and Axis 3 is obtained from the X3500W single-axis gyroscope. As shown in Figure 3.7, the noise of all three axes are roughly Gaussian distributed, with a standard deviation of 0.230 deg, 0.211 deg, and 0.429 deg for axes 1, 2, and 3, respectively. Axis 3 has a considerably higher standard deviation than Axes 1 and 2. By plotting the values of each sample (Figure 3.8), it is shown that the X3500W gyroscope went through a high-variance period between samples 1400 – 2000 (6 seconds), which accounts for a reasonable amount of the calculated variance. This was not found to be an isolated incident, and due to this, the sensor has a somewhat variable performance. If this high-variance period is removed, the standard deviation drops to 0.300.

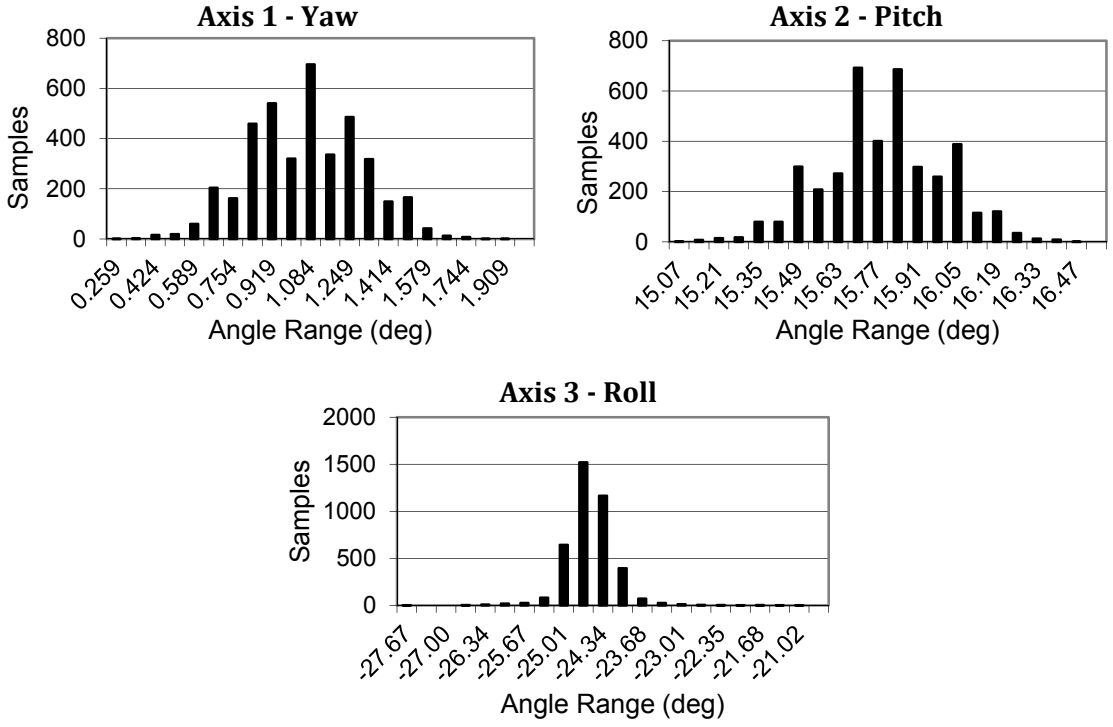


Figure 3.7. The distribution of the noise for each gyroscope axis. Axis 1 and 2 are from the IDG-600 dual-axis gyroscope, and Axis 3 is from the X3500W single-axis gyroscope.

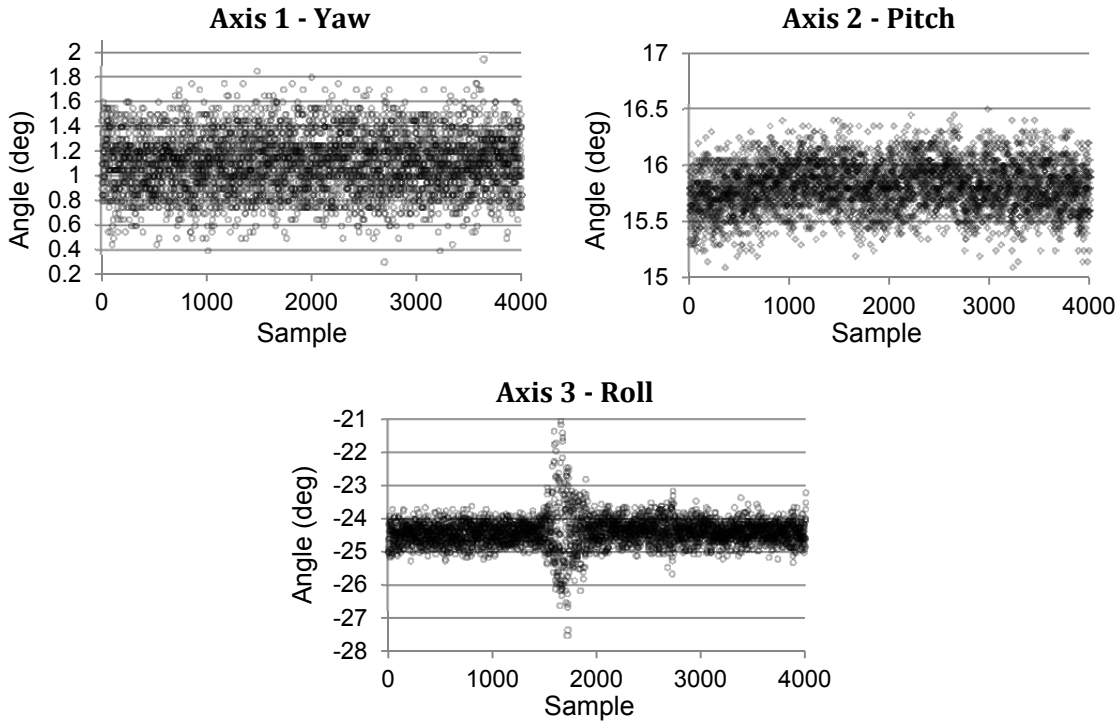


Figure 3.8. Plot of the noise for each gyroscope axis. Axis 1 and 2 are from the IDG-600 dual-axis gyroscope, and Axis 3 is from the X3500W single-axis gyroscope.

3.3 – Arduino Microcontroller

To control the IR calibration board, an Arduino Duemilanove microcontroller (Figure 3.9) using an ATmega328 chip was used. The Arduino Duemilanove has 14 digital input/output pins, 6 analog input/digital output, a 16 MHz clock speed, a USB connection, a power jack, an ICSP header, and a reset button. It has an operating voltage of 5V, a recommended input voltage of 7-12V, and it can provide 50 mA current per I/O Pin. For the particular model using the ATmega328 chip, it has 32 KB of flash memory (2 KB used for bootloader), 2 KB of SRAM, and 1 KB of EEPROM. Arduino boards can be programmed by a stripped-down C language, compiled, and uploaded to the board from a PC via a serial connection.

During calibration, the six analog input pins are set to provide digital output, allowing a total of twenty output pins. The output pins are used to flash on and off twenty sets of LEDs on the calibration board (q.v. Sect. 7.4). Each set of LEDs contains four LEDs connected in series, allowing the Arduino board to control a total of eighty LEDS when no serial communication is being used. When the

calibration board is used for teleoperation, Pin 0 (RX) and Pin 1 (TX) are used for serial communication to the PC, and consequently, only eighteen sets of LEDs (72 LEDS) can be controlled.

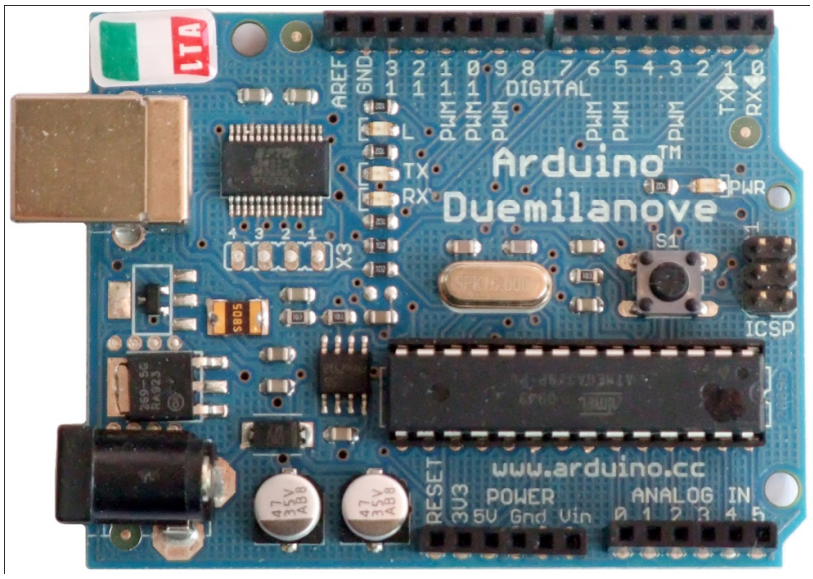


Figure 3.9. Arduino Duemilanove Microcontroller

Chapter 4 – Theoretical Foundations

This chapter provides a review of the background information required in understanding subsequent chapters.

4.1 – Attitude Kinematics and Parameterization

To perform 6-DOF teleoperation, the 3D position and 3D orientation (attitude) must be calculated. Position kinematics is straightforward to determine, but attitude kinematics is more complex since 3D rotations are nonlinear and noncommutative.

Intuitively, a 3D rotation of a vector can be thought of as a movement along the surface of a 3-sphere. Technically, a 3D rotation is a movement along a $\mathbb{S}^{\mathbf{3}}$ manifold embedded in $\mathbb{R}^{\mathbf{4}}$, which is isomorphic to the special orthogonal group $\mathbf{SO}(\mathbf{3})$.

There are many ways to parameterize a 3D rotation, each with their strengths and weaknesses. The five general representations used in this thesis are the rotation matrix, Euler angles, axis-angle, rotation vector, and the quaternion. Even within the general representations, there are varying specific systems of representation and notation, particularly for Euler angles and quaternions. When developing software to perform rotations, it becomes important to develop all equations around a consistent system, since different sources have different specific versions of representation and typically do not explicitly specify the system use. The remainder of this section provides a review of these representations, and provides a consistent representation and notation system.

4.1.1 – Rotation Matrix

A rotation matrix (direction-cosine matrix) is a 3×3 matrix that represents a transformation from one reference frame to another. It has the advantage that it can be applied directly to vectors to change the vector's reference frame. Formally,

$$^{\mathrm{A}}\mathbf{v} = {}^{\mathrm{A}}_{\mathrm{B}}\mathbf{A}\,{}^{\mathrm{B}}\mathbf{v} \tag{4.1}$$

where $^{\mathrm{A}}\mathbf{v}$ is the vector with respect to frame A, $^{\mathrm{B}}\mathbf{v}$ is the vector with respect to frame B, and ${}^{\mathrm{A}}_{\mathrm{B}}\mathbf{A}$ is the rotation of the reference frame from B to A. For consistency and clarity with subsequent definitions,

reference frames are labelled on the left side, which has the same meaning as the more common right-side labelling.

In rotation matrix form, consecutive rotations can be combined by matrix multiplication. Precisely, a rotation from frame C to frame B, ${}^{\mathrm{B}}_{\mathrm{C}}\mathbf{A}$, followed by a rotation of frame B to frame A, ${}^{\mathrm{A}}_{\mathrm{B}}\mathbf{A}$ is equivalent to a rotation from frame C to frame A, ${}^{\mathrm{A}}_{\mathrm{C}}\mathbf{A}$, and is obtained by

$$ {}^{\mathrm{A}}_{\mathrm{C}}\mathbf{A} = {}^{\mathrm{A}}_{\mathrm{B}}\mathbf{A}\,{}^{\mathrm{B}}_{\mathrm{C}}\mathbf{A} \tag{4.2} $$

Additionally, a rotation from frame B to frame A is the inverse of a rotation from frame A to frame B

$$ {}^{\mathrm{B}}_{\mathrm{A}}\mathbf{A} = \left({}^{\mathrm{A}}_{\mathrm{B}}\mathbf{A}\right)^{-1} \tag{4.3} $$

Since the rotation matrix is of the $\boldsymbol{SO}(\mathbf{3})$, the inverse operation can be conveniently simplified to a matrix transposition

$$ \left({}^{\mathrm{A}}_{\mathrm{B}}\mathbf{A}\right)^{-1} = \left({}^{\mathrm{A}}_{\mathrm{B}}\mathbf{A}\right)^{\mathrm{T}} \tag{4.4} $$

Hence,

$$ {}^{\mathrm{B}}_{\mathrm{A}}\mathbf{A} = \left({}^{\mathrm{A}}_{\mathrm{B}}\mathbf{A}\right)^{\mathrm{T}} \tag{4.5} $$

It is useful to note that a vector rotation is relativistically equivalent to an inverse rotation of the respective reference frame.

Since rotation matrices are orthogonal, the row and column vectors have a vector norm constraint of one. These six norm constraints reduce the nine-parameter matrix to 3-DOF.

The disadvantage to the rotation matrix representation is that there are six redundant parameters. A second disadvantage is that maintaining the matrix's orthogonality constraint can be problematic due to numerical noise during successive integration of rotations. The numerical noise results from the accumulation of errors during floating-point multiplication and leads to matrix transformations that are not purely rotational. Although this problem can be corrected by reorthogonalizing the matrix, this is a computationally expensive nonlinear operation.

4.1.2 – Euler Angles

Leonard Euler discovered that any 3D rotation can be represented by three rotation angles ($\alpha, \beta, \text{and } \gamma$) along orthogonal planes. These rotation angles are commonly referred to as Euler angles.

Despite the fact that the axis of rotation of each Euler angles provides a similar function to a basis, the Euler angles are not true vectors since they generally cannot be combined by vector addition and are noncommutative. Only as the rotations approach zero can they be treated like vectors.

Euler angles as a representation are typically ambiguous, since even without considering the sign of the angles, there are twelve different rotation conventions, six proper and six Tait-Bryan. The rotations can be about the current frame or about the fixed frame and this may cause some confusion. An x-y-z set of rotations about the fixed frame, is equivalent to a z-y-x set of rotations about the current frame. Typically, these conventions are not completely specified, which makes relying on equations from other sources problematic. To avoid ambiguity, these details are explicitly described in this section.

This research uses a Tait-Bryan yaw-pitch-roll system, with a z-y-x set of rotations about the current plane, which is represented formally as

$$\mathbf{A} = \mathbf{A}_{\mathrm{x,roll}}\mathbf{A}_{\mathrm{y,pitch}}\mathbf{A}_{\mathrm{z,yaw}} \tag{4.6}$$

$$\mathbf{A} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta_{roll}) & -\sin(\theta_{roll}) \\ 0 & \sin(\theta_{roll}) & \cos(\theta_{roll}) \end{bmatrix} \begin{bmatrix} \cos(\theta_{\mathrm{pitch}}) & 0 & \sin(\theta_{\mathrm{pitch}}) \\ 0 & 1 & 0 \\ -\sin(\theta_{\mathrm{pitch}}) & 0 & \cos(\theta_{\mathrm{pitch}}) \end{bmatrix} \begin{bmatrix} \cos(\theta_{yaw}) & -\sin(\theta_{yaw}) & 0 \\ \sin(\theta_{yaw}) & \cos(\theta_{yaw}) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{4.7}$$

Attaching reference frames to each rotation, $\mathbf{A}$ can be expressed as:

1. Rotating the xyz frame about the z-axis ($\mathrm{A}_{\mathrm{z,yaw}}$), leads to the XYZ frame.
2. Rotating the XYZ frame about Y-axis ($\mathrm{A}_{\mathrm{y,pitch}}$), leads to the X'Y'Z' frame.
3. Rotating the X'Y'Z' frame about the X'-axis ($\mathrm{A}_{\mathrm{x,roll}}$), leads to the final X''Y''Z'' frame.

The sign of the rotation is determined by the right-hand rule.

Euler angles double cover the 3-sphere when all three angles have a domain from θ to $\theta + 2\pi$. In this research, yaw, pitch, and roll are constrained from $-\pi$ to π, $-\pi/2$ to $\pi/2$, and $-\pi$ to π, respectively, to remove this representational redundancy.

The advantage of this specific representation of Euler angles is that it is a minimal representation and is arguably the easiest representation for humans to visualize. The first disadvantage is that there are discontinuous jumps when yaw and roll pass over the plane at the junction of $-\pi$ and π and for pitch, $-\pi/2$ to $\pi/2$. This is not a problem for specifying attitude, although, it is a problem for the stochastic estimation algorithms typically used in attitude determination. The second disadvantage is that when the second Euler angle is such that the first plane of rotation is aligned with the third, a so-called gimbal-lock singularity occurs, leading to a drop in DOF from three to two. Consequently, rotation that is orthogonal to the second and first or third planes cannot be represented until the system is removed from the singularity.

The Euler angle representation must be converted to another representation before being applied to vectors. Relatively speaking, conversion to a rotation matrix is computationally efficient and conversion to a quaternion is computationally inefficient.

4.1.3 – Axis-Angle

In three dimensions (and only in three dimensions), as stated by Euler's Rotation Theorem [117], any general motion of a rigid body about a fixed point can be characterized by a rotation about an arbitrary axis $\mathbf{k}$ with a magnitude of the rotation being the angle θ. This is referred to as the axis-angle representation. The disadvantage of it is that it is subject to discontinuous jumps when the angle θ passes over the $-\pi$ and π boundary.

The axis-angle representation must first be converted to another representation before being applied to vectors. Conversion to a quaternion is computationally efficient and conversion to a rotation matrix is computationally inefficient.

4.1.4 – Rotation Vector

By multiplying the rotation angle θ by the axis $\mathbf{k}$, the rotation vector $\boldsymbol{\theta}$ representation is obtained. The advantage of the rotation vector is that it is a minimal representation. Like the axis-angle representation, the rotation vector also has the disadvantage of discontinuous jumps when the angle θ passes over the $-\pi$ and π boundary. Additionally, it contain a singularity when $\theta = 0$, which under the presence of numerical noise, can lead to ill behaviour in the neighbourhood of the singularity.

The rotation vector must also be converted to another representation before being applied to vectors. Conversion to a quaternion is computationally efficient and conversion to a rotation matrix is computationally inefficient.

4.1.5 – Quaternions

A quaternion is a hyperimaginary number that forms a four-dimensional normed division algebra over $\mathbb{R}$. It is the smallest representation that is singularity-free and bilinear (Sect. 4.2.1). The quaternion can be described in the form

$$\bar{q} = q_4 + q_1\mathbf{i} + q_2\mathbf{j} + q_3\mathbf{k} \tag{4.8}$$

where

$$\left.\begin{aligned} \mathbf{i}^2 = \mathbf{j}^2 = \mathbf{k}^2 = \mathbf{ijk} = -1 \\ \mathbf{ij} = -\mathbf{ij} = -\mathbf{k} \\ \mathbf{jk} = -\mathbf{kj} = -\mathbf{i} \\ \mathbf{ki} = -\mathbf{ik} = -\mathbf{j} \end{aligned}\right\} \tag{4.9}$$

Rotations in three dimensions are represented by a unit quaternion

$$\bar{q} = \begin{bmatrix} \mathbf{q} \\ q_4 \end{bmatrix} = [q_1 \quad q_2 \quad q_3 \quad q_4]^{\mathrm{T}} \tag{4.10}$$

and

$$\mathbf{q} = \mathbf{k}\sin(\theta/2) \qquad , \qquad q_4 = \cos(\theta/2) \tag{4.11}$$

where $\mathbf{k}$ is the unit vector describing the axis of rotation and θ is the angle of rotation. Both (4.10) and (4.11) demonstrate that transforming the axis-angle representation to the unit quaternion notation is trivial. Additionally, the unit quaternion satisfies the unit norm condition

$$|\bar{q}| = \sqrt{\bar{q}^{\mathrm{T}}\bar{q}} = \sqrt{{q_1}^2 + {q_2}^2 + {q_3}^2 + {q_4}^2} = 1 \tag{4.12}$$

For brevity, through the remainder of the thesis, any use of *quaternion* will refer to the *unit quaternion*.

4.2 – Quaternion Algebra and Calculus

A quaternion is a useful mathematical representation that is frequently used in computational models of rotation. It is the most significant rotation representation used in this research and of particular importance in the development of the multiplicative extended Kalman filter (c.f. 5.5). To take full

advantage of the quaternion, quaternion algebra and calculus are developed in subsections 4.2.1 – 4.2.3. This information is provided in detail, since it is often not present in many textbooks.

4.2.1 – Quaternion Algebra and Properties

Rotations in space using quaternions are performed by quaternion multiplication

$$\begin{aligned}\bar{q} \otimes \bar{p} &= (q_4 + q_1\mathbf{i} + q_2\mathbf{j} + q_3\mathbf{k})(p_4 + p_1\mathbf{i} + p_2\mathbf{j} + p_3\mathbf{k}) \\ &= \begin{bmatrix} q_4p_1 + q_3p_2 - q_2p_3 + q_1p_4 \\ -q_3p_1 + q_4p_2 + q_1p_3 + q_2p_4 \\ q_2p_1 - q_1p_2 + q_4p_3 + q_3p_4 \\ -q_1p_1 - q_2p_2 - q_3p_3 + q_4p_4 \end{bmatrix}\end{aligned} \tag{4.13}$$

There are two notation formats used, the classical Hamiltonian notation [118] where

$$\mathbf{A}(\bar{p})\mathbf{A}(\bar{q}) = \mathbf{A}(\bar{q} \otimes \bar{p}) \tag{4.14}$$

and the natural order notation [115] where

$$\mathbf{A}(\bar{q})\mathbf{A}(\bar{p}) = \mathbf{A}(\bar{q} \otimes \bar{p}) \tag{4.15}$$

which has ordering in the same sequence as rotation matrices and is the proposed standard convention by NASA's Jet Propulsion Laboratory [119]. All use of quaternions in this thesis will follow the natural order notation.

An important property of a quaternion is that is can directly rotate the reference frame of a vector from frame A to frame B using

$${}^{\mathrm{B}}v = {}^{\mathrm{B}}_{\mathrm{A}}\bar{q} \otimes {}^{\mathrm{A}}v \otimes {}^{\mathrm{B}}_{\mathrm{A}}\bar{q}^{-1} \tag{4.16}$$

As is the case for rotations matrices, a series of rotations can be represented in quaternion notation by quaternion multiplication. A rotation ${}^{\mathrm{B}}_{\mathrm{A}}\bar{q}$ from reference frames A to B followed by a rotation ${}^{\mathrm{C}}_{\mathrm{B}}\bar{q}$ from reference frames B to C is equivalent to a single rotation ${}^{\mathrm{C}}_{\mathrm{A}}\bar{q}$ from A to C by

$${}^{\mathrm{C}}_{\mathrm{A}}\bar{q} = {}^{\mathrm{C}}_{\mathrm{B}}\bar{q} \otimes {}^{\mathrm{B}}_{\mathrm{A}}\bar{q} \tag{4.17}$$

These rotations can also be expressed using matrix form by factoring out (4.13) in two useful forms

$$\bar{q} \otimes \bar{p} = \boldsymbol{\mathcal{L}}(\bar{q})\bar{p} \tag{4.18}$$

$$\bar{q} \otimes \bar{p} = \boldsymbol{\mathcal{R}}(\bar{p})\bar{q} \tag{4.19}$$

where

$$\boldsymbol{\mathcal{L}}(\bar{q}) = \begin{bmatrix} q_4 & q_3 & -q_2 & q_1 \\ -q_3 & q_4 & q_1 & q_2 \\ q_2 & -q_1 & q_4 & q_3 \\ -q_1 & -q_2 & -q_3 & q_4 \end{bmatrix} \tag{4.20}$$

which can be further reduced to

$$\boldsymbol{\mathcal{L}}(\bar{q}) = \begin{bmatrix} q_4\mathbf{I}_{3\times 3} - [\mathbf{q}\times] & \mathbf{q} \\ -\mathbf{q}^{\mathrm{T}} & q_4 \end{bmatrix} \tag{4.21}$$

where the skew-symmetric matrix

$$[\mathbf{q}\times] = \begin{bmatrix} 0 & -q_3 & q_2 \\ q_3 & 0 & -q_1 \\ -q_2 & q_1 & 0 \end{bmatrix} \tag{4.22}$$

is the matrix representation of the cross product such that

$$\mathbf{q} \times \mathbf{p} = [\mathbf{q}\times]\mathbf{p} \tag{4.23}$$

Similarly, $\boldsymbol{\mathcal{R}}$ is obtained by

$$\boldsymbol{\mathcal{R}}(\bar{p}) = \begin{bmatrix} p_4 & -p_3 & p_2 & p_1 \\ p_3 & p_4 & -p_1 & p_2 \\ -p_2 & p_1 & p_4 & p_3 \\ -p_1 & -p_2 & -p_3 & p_4 \end{bmatrix} \tag{4.24}$$

$$\boldsymbol{\mathcal{R}}(\bar{p}) = \begin{bmatrix} q_4\mathbf{I}_{3\times 3} + [\mathbf{p}\times] & \mathbf{p} \\ -\mathbf{p}^{\mathrm{T}} & p_4 \end{bmatrix} \tag{4.25}$$

The quaternion analog to the identity matrix is the identity quaternion

$$\bar{q}_{\mathrm{I}} = [0 \quad 0 \quad 0 \quad 1]^{\mathrm{T}} \tag{4.26}$$

which has the property

$$\bar{q}_{\mathrm{I}} \otimes \bar{q} = \bar{q} \otimes \bar{q}_{\mathrm{I}} = \bar{q} \tag{4.27}$$

The inverse rotation of a quaternion is obtained by taking the inverse (complex conjugate) of the quaternion

$$\bar{q}^{-1} = \begin{bmatrix} -\mathbf{q} \\ q_4 \end{bmatrix} \tag{4.28}$$

and has the properties

$$\bar{q} \otimes \bar{q}^{-1} = \bar{q}^{-1} \otimes \bar{q} = \bar{q}_{\mathrm{I}} \tag{4.29}$$

$$(\bar{q} \otimes \bar{p})^{-1} = \bar{p}^{-1} \otimes \bar{q}^{-1} \tag{4.30}$$

For a further list of properties, see Appendix A.

4.2.2 – Quaternion Differentiation

The quaternion time-derivative is the rate of change of a quaternion rotation with respect to time. Attaching the local frame to the object of interest, the rate of change ${}^{\mathrm{L}(t)}_{\mathrm{G}}\dot{\bar{q}}(t)$ of the rotating local frame $\mathrm{L}(t)$ with respect to a fixed global frame G over a window of time by taking the limit of the difference between the quaternions is obtained by

$${}^{\mathrm{L}(t)}_{\mathrm{G}}\dot{\bar{q}}(t) = \lim_{\Delta t \to 0} \frac{1}{\Delta t}\left({}^{\mathrm{L}(t+\Delta t)}_{\mathrm{G}}\bar{q}(t) - {}^{\mathrm{L}(t)}_{\mathrm{G}}\bar{q}(t)\right) \tag{4.31}$$

Expanding the quaternions by splitting the rotations into two successive rotations

$${}^{\mathrm{L}(t)}_{\mathrm{G}}\dot{\bar{q}}(t) = \lim_{\Delta t \to 0} \frac{1}{\Delta t}\left({}^{\mathrm{L}(t+\Delta t)}_{\mathrm{L}(t)}\bar{q}(\mathrm{t}) \otimes {}^{\mathrm{L}(t)}_{\mathrm{G}}\bar{q}(\mathrm{t}) - \bar{q}_{\mathrm{I}} \otimes {}^{\mathrm{L}(t)}_{\mathrm{G}}\bar{q}(t)\right) \tag{4.32}$$

Factoring out ${}^{\mathrm{L}(\mathrm{t})}_{\mathrm{G}}\bar{q}(\mathrm{t})$ and using (4.10) and (4.11)

$${}^{\mathrm{L}(t)}_{\mathrm{G}}\dot{\bar{q}}(t) = \lim_{\Delta t \to 0} \frac{1}{\Delta t}\left(\begin{bmatrix}\mathbf{k} \cdot \sin(\theta/2) \\ \cos(\theta/2)\end{bmatrix} - \begin{bmatrix}\mathbf{0}_{3\times 1} \\ 1\end{bmatrix}\right) \otimes {}^{\mathrm{L}(t)}_{\mathrm{G}}\bar{q}(t) \tag{4.33}$$

Using the small angle approximation (A.25) and reorganizing

$${}^{\mathrm{L}(t)}_{\mathrm{G}}\dot{\bar{q}}(t) \approx \lim_{\Delta t \to 0} \frac{1}{\Delta t}\left(\begin{bmatrix}\mathbf{k} \cdot \theta/2 \\ 1\end{bmatrix} - \begin{bmatrix}\mathbf{0}_{3\times 1} \\ 1\end{bmatrix}\right) \otimes {}^{\mathrm{L}(t)}_{\mathrm{G}}\bar{q}(t) \tag{4.34}$$

$$= \lim_{\Delta t \to 0} \frac{1}{\Delta t}\left(\begin{bmatrix}\boldsymbol{\theta}/2 \\ 1\end{bmatrix} - \begin{bmatrix}\mathbf{0}_{3\times 1} \\ 1\end{bmatrix}\right) \otimes {}^{\mathrm{L}(t)}_{\mathrm{G}}\bar{q}(t) \tag{4.35}$$

$$= \lim_{\Delta t \to 0} \frac{1}{2}\left(\begin{bmatrix}\boldsymbol{\theta}/\Delta t \\ 0\end{bmatrix}\right) \otimes {}^{\mathrm{L}(t)}_{\mathrm{G}}\bar{q}(t) \tag{4.36}$$

$$= \frac{1}{2}\begin{bmatrix}\boldsymbol{\omega} \\ 0\end{bmatrix} \otimes {}^{\mathrm{L}(t)}_{\mathrm{G}}\bar{q}(t) \tag{4.37}$$

Using the property (4.21)

$$ {}^{\mathrm{L}(t)}_{\mathrm{G}}\dot{\bar{q}}(t) = \frac{1}{2}\begin{bmatrix} -[\boldsymbol{\omega}\times] & \boldsymbol{\omega} \\ -\boldsymbol{\omega}^{\mathrm{T}} & 0 \end{bmatrix} {}^{\mathrm{L}(t)}_{\mathrm{G}}\bar{q}(t) \tag{4.38} $$

Using $\boldsymbol{\Omega}$ as defined in (A.19), the quaternion derivative is expressed as

$$ {}^{\mathrm{L}(t)}_{\mathrm{G}}\dot{\bar{q}}(t) = \frac{1}{2}\boldsymbol{\Omega}(\boldsymbol{\omega}) {}^{\mathrm{L}(t)}_{\mathrm{G}}\bar{q}(t) \tag{4.39} $$

In the equation of this section, the vectors $\mathbf{k}$, $\boldsymbol{\theta}$, and $\boldsymbol{\omega}$ are functions of time and are with respect to the local frame $\mathrm{L}(t)$, although, both function-of-time and frame markings are suppressed to improve equation clarity. Additionally, in this work the local frame L will always be a function of time, though this will often be dropped for brevity.

It is also useful to differentiate quaternion equations of the form (4.17), which are solved trivially by applying the product rule

$$ {}^{\mathrm{C}}_{\mathrm{A}}\dot{\bar{q}} = {}^{\mathrm{B}}_{\mathrm{A}}\dot{\bar{q}} \otimes {}^{\mathrm{C}}_{\mathrm{B}}\bar{q} + {}^{\mathrm{B}}_{\mathrm{A}}\bar{q} \otimes {}^{\mathrm{C}}_{\mathrm{B}}\dot{\bar{q}} \tag{4.40} $$

4.2.3 – Solving Quaternion Differential Equations

This section reviews the method to obtain the general solution to the quaternion differential equation of the form (4.39), which is equivalent to solving a system of first-order ordinary differential equations (ODEs). Following [116], the solution to (4.39) is of the general form [120]

$$ {}^{\mathrm{L}}_{\mathrm{G}}\bar{q}(t_k) = \boldsymbol{\Phi}^{\bar{q}}(t_k, t_{k-1}) {}^{\mathrm{L}}_{\mathrm{G}}\bar{q}(t_{k-1}) \tag{4.41} $$

The quaternion transition matrix $\boldsymbol{\Phi}^{\bar{q}}$ can be found by differentiating (4.41), and by substituting in (4.39) and (4.41)

$$ {}^{\mathrm{L}}_{\mathrm{G}}\dot{\bar{q}}(t_k) = \dot{\boldsymbol{\Phi}}^{\bar{q}}(t_k, t_{k-1}) {}^{\mathrm{L}}_{\mathrm{G}}\bar{q}(t_{k-1}) \tag{4.42} $$

$$ \frac{1}{2}\boldsymbol{\Omega}(\boldsymbol{\omega}) {}^{\mathrm{L}}_{\mathrm{G}}\bar{q}(t_k) = \dot{\boldsymbol{\Phi}}^{\bar{q}}(t_k, t_{k-1}) {}^{\mathrm{L}}_{\mathrm{G}}\bar{q}(t_{k-1}) \tag{4.43} $$

$$ \frac{1}{2}\boldsymbol{\Omega}(\boldsymbol{\omega})\, \boldsymbol{\Phi}^{\bar{q}}(t_k, t_{k-1}) {}^{\mathrm{L}}_{\mathrm{G}}\bar{q}(t) = \dot{\boldsymbol{\Phi}}^{\bar{q}}(t_k, t_{k-1}) {}^{\mathrm{L}}_{\mathrm{G}}\bar{q}(t_{k-1}) \tag{4.44} $$

$$ \dot{\boldsymbol{\Phi}}^{\bar{q}}(t_k, t_{k-1}) = \frac{1}{2}\boldsymbol{\Omega}(\boldsymbol{\omega})\, \boldsymbol{\Phi}^{\bar{q}}(t_k, t_{k-1}) \tag{4.45} $$

with initial condition

$$ \boldsymbol{\Phi}^{\bar{q}}(t_{k-1}, t_{k-1}) = \mathbf{I}_{3\times 3} \tag{4.46} $$

Lastly, $\mathbf{\Phi}^{\bar{q}}$ can be obtained by solving (4.45)-(4.46) as a system of first-order ODEs.

The quaternion algebra and calculus provided here is used to develop the multiplicative extended Kalman filter.

4.3 – Robot Kinematics

This section presents an overview of the theory required to calculate the kinematics of the robot. Modelling the kinematic chain is first considered using general forward kinematics. Next, the parameter space of the forward kinematics is reduced using the Denavit-Hartenberg convention, which simplifies the inverse kinematics. Lastly, an overview of computing the inverse kinematics in closed form via kinematics decoupling is presented.

4.3.1 – Forward Kinematics

Forward kinematics is used to calculate the pose of the end effector given the angles of the joints and knowledge about the geometry and motion of the robot. It is required by the Virtual Robot Manipulator (Sect. 6.11) to compute the positions of all links and joints, which are used for safety reasons and to display the Virtual Robot to the screen.

Calculating the forward kinematics requires *rotation and translation* of a vector and can be calculated using

$$^{\mathrm{G}}\mathbf{v} = {}^{\mathrm{G}}_{\mathrm{L}}\mathbf{A}\,{}^{\mathrm{L}}\mathbf{v} + {}^{\mathrm{G}}_{\mathrm{L}}\mathbf{d} \tag{4.47}$$

where ${}^{\mathrm{G}}_{\mathrm{L}}\mathbf{d}$ is the vector translation from the global to the local frame.

Alternatively, the rotation and translation in (4.47) can be expressed as a single homogeneous transformation

$$^{\mathrm{G}}\mathbf{v}^{*} = {}^{\mathrm{G}}_{\mathrm{L}}\mathbf{H}\,{}^{\mathrm{L}}\mathbf{v}^{*} \tag{4.48}$$

where $^{\mathrm{G}}\mathbf{v}^{*}$ and $^{\mathrm{L}}\mathbf{v}^{*}$ are the homogeneous representations of the vectors

$$^{\mathrm{G}}\mathbf{v}^{*} = \begin{bmatrix} {}^{\mathrm{G}}\mathbf{v} \\ 1 \end{bmatrix} \quad and \quad {}^{\mathrm{L}}\mathbf{v}^{*} = \begin{bmatrix} {}^{\mathrm{L}}\mathbf{v} \\ 1 \end{bmatrix} \tag{4.49}$$

and the homogeneous transformation from the local to global frame is

$$ {}^{\mathrm{G}}_{\mathrm{L}}\mathbf{H} = \begin{bmatrix} {}^{\mathrm{G}}_{\mathrm{L}}\mathbf{A} & {}^{\mathrm{G}}_{\mathrm{L}}\mathbf{d} \\ \mathbf{0}_{1\times 3} & 1 \end{bmatrix} \tag{4.50} $$

The homogeneous transformation matrix ${}^{\mathrm{G}}_{\mathrm{L}}\mathbf{H}$ is part of the Special Euclidean Group ($\boldsymbol{SE}(\mathbf{3})$) and thus has the useful property

$$ {}^{\mathrm{G}}_{\mathrm{L}}\mathbf{H^{-1}} = \begin{bmatrix} {}^{\mathrm{G}}_{\mathrm{L}}\mathbf{A}^{\mathrm{T}} & -{}^{\mathrm{G}}_{\mathrm{L}}\mathbf{A}^{\mathrm{T}}\,{}^{\mathrm{G}}_{\mathrm{L}}\mathbf{d} \\ \mathbf{0}_{1\times 3} & 1 \end{bmatrix} \tag{4.51} $$

where $(\cdot)^{\mathrm{T}}$ is the matrix transpose.

Using (4.50) it is possible to calculate the forward kinematics of any rigid robotic arm by performing a transformation for every joint. The transformation of reference frames from the last joint n to the base is obtain by

$$ {}^{0}_{n}\mathbf{H} = \mathbf{T}_1 \dots \mathbf{T}_{n-1}\mathbf{T}_n \tag{4.52} $$

where $\mathbf{T}_i$ is the transformation at joint i for. Although (4.52) can be used to calculate the kinematic chain with a six-link robot such as the one used in this research, the final equation can be large with up to thirty-six parameters. Considering that robot joints are frequently either prismatic or revolute, the Denavit-Hartenberg convention can be used to simplify the equations.

4.3.2 – Denavit-Hartenberg Convention

The Denavit-Hartenberg (DH) convention breaks each transformation matrix into four one-DOF matrices

$$ \mathbf{T}_i = [\mathbf{Rot}_{z,\vartheta_i}][\mathbf{Trans}_{z,d_i}][\mathbf{Trans}_{x,a_i}][\mathbf{Rot}_{x,\alpha_i}] \tag{4.53} $$

$$ = \begin{bmatrix} c_{\vartheta_i} & -s_{\vartheta_i} & 0 & 0 \\ s_{\vartheta_i} & c_{\vartheta_i} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & a_i \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & c_{\alpha_i} & -s_{\alpha_i} & 0 \\ 0 & s_{\alpha_i} & c_{\alpha_i} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.54} $$

$$ \mathbf{T}_i = \begin{bmatrix} c_{\vartheta_i} & -s_{\vartheta_i}c_{\alpha_i} & s_{\vartheta_i}s_{\alpha_i} & c_{\vartheta_i}a_i \\ s_{\vartheta_i} & c_{\vartheta_i}c_{\alpha_i} & -c_{\vartheta_i}s_{\alpha_i} & s_{\vartheta_i}a_i \\ 0 & s_{\alpha_i} & c_{\alpha_i} & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.55} $$

where the DH parameters ϑ_i, a_i, d_i, α_i are the link length, link twist, link offset, and joint angle, respectively. The link offset d_i is the displacement along the z_{i-1}-axis between reference frames o_{i-1} and o_i. The joint angle ϑ_i is the angle of rotation about the z_{i-1}-axis required to align x_{i-1} with x_i. The

link length a_i is the shortest length (common normal) between the z_{i-1}-axis and z_i-axis and is by the DH convention, along the x_i-axis. Finally, the link twist α_i is the angle of rotation about the x_i-axis required to align the z_{i-1}-axis and the z_i-axis. This convention reduces each transformation from six to four parameters; however, in practice each transformation can typically be expressed by one variable and from zero to three constants.

In the DH convention, at least $n + 1$ sets of axes are used. A set of axes i is set up for each of the n joints (prismatic or revolute) and a final set of axes is positioned at the tool or end effector. Furthermore, the axes are positioned and aligned such that the following constraints are held:

1. The axis x_i is orthogonal to the axis z_{i-1}.

2. The axis x_i intersects axis z_{i-1}.

For further details on setting up the reference frames and determining the four DH parameters, refer to [122].

4.3.3 – Inverse Kinematics

Solving the inverse kinematics is required by the Virtual Robot Controller (Sect. 6.10). As the name suggests, the inverse kinematics problem deals with solving the forward kinematics problem in the reverse order. Specifically, given a homogenous transformation matrix of the end effector

$$ {}^0_n\mathbf{T} = \begin{bmatrix} {}^0_n\mathbf{A} & {}^0_n\mathbf{p} \\ \mathbf{0}_{1\times 3} & 1 \end{bmatrix} \in SE(3), \tag{4.56} $$

the general inverse kinematics solves for the joint angles $\theta_1, \dots, \theta_n$, such that

$$ \mathbf{T}_1(\theta_1)\mathbf{T}_2(\theta_2) \dots \mathbf{T}_n(\theta_n) = {}^0_n\mathbf{T}(\theta_1, \dots, \theta_n) \tag{4.57} $$

where $\mathbf{A}$ is the rotation to the end effector and $\mathbf{p}$ is the position of the end effector, and both are with respect to the initial reference frame $\boldsymbol{o}_0$. Before (4.57) can be solved, the homogenous transformation matrix ${}^0_n\mathbf{T}$ must first be obtained. Specifically, ${}^0_n\mathbf{T}$ is the general solution to the forward kinematics problem of Sect. 4.3, and is found using the DH convention.

The general solution of the inverse kinematics problem for a six-link robotic arm, i.e. (4.57) with *n*=6, has not yet been solved analytically [122] due to its highly nonlinear nature, but under certain circumstances, the problem can be made much simpler. Fortunately, many industrial robot arms can be

solved geometrically, because they are typically designed to match the constraints of one of the known solutions.

When the reference frames can be set up such that the frames on the last three joints intersect at a single point (wrist centre), the problem can be solved geometrically via kinematic decoupling. Kinematic decoupling allows the joints after the wrist centre to act as a spherical joint, and breaks the problem into two simpler problems, which are solvable by inverse position kinematics and the inverse orientation kinematics.

Inverse position kinematics determines the position of the wrist centre, and then uses geometry to find the joint angles that are required for the wrist centre to be at that location. Inverse orientation kinematics uses the orientation of the end effector to find the last three joint angles. Solving the inverse kinematics in closed form is useful since there generally exists multiple solutions and rules can be designed to select the appropriate solution. Furthermore, closed form solutions can be substantially less expensive computationally, which is essential when a high update rate is required on limited computational resources, as is often the case. For details on performing inverse kinematics via kinematic decoupling, refer to [122].

4.4 – Gyroscope Kinematics Modelling

The gyroscope is an important sensor in this teleoperation system, and before near-optimal estimation can be made using the gyroscope, a kinematic model of it is necessary.

Gyroscopes measure the angular rate $\boldsymbol{\omega}$, which is defined by the time-derivative of the rotation vector $\boldsymbol{\theta}$. Formally,

$$\boldsymbol{\omega} = \frac{d\boldsymbol{\theta}}{dt} \tag{4.58}$$

Equivalently, the angular rate $\boldsymbol{\omega}$ can be defined as the time-derivative of the Euler angles, since

$$\lim_{|\boldsymbol{\theta}|\to\infty} \alpha = \theta_1, \qquad \lim_{|\boldsymbol{\theta}|\to\infty} \beta = \theta_2, \qquad \lim_{|\boldsymbol{\theta}|\to\infty} \gamma = \theta_3 \tag{4.59}$$

Using this derivation, the three components of $\boldsymbol{\omega}$, will be referred to as *yaw rate, pitch rate,* and *roll rate*.

No sensor can produce perfect measurements since the measured states are always corrupted by unwanted influences, which have properties that to varying degrees are linear, nonlinear, deterministic, and stochastic. For gyroscopes specifically, the greatest portion of measurement error is due to additive noise (stochastic), with a nonzero mean (linear) that drifts over time (nonlinear).

To increase the accuracy of the gyroscopes, a simple yet sufficiently accurate model developed by Farrenkopf [121] is used. This model accounts for the underlying influences that produce a measurement, and is presented formally as

$$\boldsymbol{g}(t) = \boldsymbol{\omega}(t) + \mathbf{b}(t) + \mathbf{n}_{\boldsymbol{\omega}}(t) \tag{4.60}$$

where $\boldsymbol{g}$ is the gyroscope's measurement, $\boldsymbol{\omega}$ is the angular rate, $\mathbf{b}$ is the drift-rate bias, $\mathbf{n}_{\boldsymbol{\omega}}$ is the drift-rate noise, and all four are 3 dimensional column vectors. The drift-rate noise is assumed to be additive, white, and Gaussian, with a mean and autocorrelation defined by

$$\mathbf{E}\{\mathbf{n}_{\boldsymbol{\omega}}(t)\} = \mathbf{0}_{3\times 1} \tag{4.61}$$

$$\mathbf{E}\{\mathbf{n}_{\boldsymbol{\omega}}(t)\mathbf{n}_{\boldsymbol{\omega}}{}^{\mathrm{T}}(t-\tau)\} = \mathbf{Q}_{\boldsymbol{\omega}}(t)\delta(\tau) \tag{4.62}$$

where $\mathbf{0}_{3\times 1}$ is a null vector, τ is a small period of time, $\mathbf{Q}_{\boldsymbol{\omega}}$is the angular-rate noise covariance matrix, $\delta(\tau)$ is the dirac delta function, and $\mathbf{E}\{\cdot\}$ is the expectation operator. Although (4.60) treats the bias $\mathbf{b}$ as a constant, the bias of an actual gyro is a nonlinear function of numerous factors. However, the bias changes sufficiently slowly so that it can be compensated by

$$\dot{\mathbf{b}}(t) = \mathbf{n}_{\mathbf{b}}(t) \tag{4.63}$$

where $\mathbf{n}_{\mathbf{b}}$ is the bias-drift ramp noise, which is assumed to be additive, white, and Gaussian with mean and autocorrelation

$$\mathbf{E}\{\mathbf{n}_{\mathbf{b}}(t)\} = \mathbf{0}_{3\times 1} \tag{4.64}$$

$$\mathbf{E}\{\mathbf{n}_{\mathbf{b}}(t)\mathbf{n}_{\mathbf{b}}{}^{\mathrm{T}}(t-\tau)\} = \mathbf{Q}_{\mathbf{b}}(t)\delta(\tau) \tag{4.65}$$

where $\mathbf{Q}_{\mathbf{b}}$ is the bias noise covariance matrix.

In the wiimote, there is one dual-axis gyro and one single-axis gyro. From experimental observation, it has been determined that the noise of the angular rate's variance on each axis of the dual-axis gyro is similar, and the variance between gyros is different. It is also assumed that the bias-drift ramp noise is the same for both, and that the noise is uncorrelated. Thus, the angular-rate noise covariance and bias noise covariance simplifies to

$$\mathbf{Q}_{\boldsymbol{\omega}} = \begin{bmatrix} \sigma^2_{\boldsymbol{\omega},2} & 0 & 0 \\ 0 & \sigma^2_{\boldsymbol{\omega},2} & 0 \\ 0 & 0 & \sigma^2_{\boldsymbol{\omega},1} \end{bmatrix} \tag{4.66}$$

$$\mathbf{Q}_{\mathbf{b}} = \begin{bmatrix} \sigma^2_{\mathbf{b}} & 0 & 0 \\ 0 & \sigma^2_{\mathbf{b}} & 0 \\ 0 & 0 & \sigma^2_{\mathbf{b}} \end{bmatrix} = \sigma^2_{\mathbf{b}} \cdot \mathbf{I}_{3\times 3} \tag{4.67}$$

where $\sigma^2_{\boldsymbol{\omega},2}$ and $\sigma^2_{\boldsymbol{\omega},1}$ are respectively the variance of the drift-rate noise from the dual-axis gyro and single-axis gyro, and for simplicity, $\sigma^2_{\mathbf{b}}$ is the variance of the drift-rate ramp noise of both gyros. The true bias drift is not similar to additive Gaussian white noise, although, since it varies slowly, a small value for the drift-rate ramp noise is still effective at modelling it.

With equations (4.60)-(4.67), the kinematics of the gyros can be modelled sufficiently well for the current application, and is an integral part of the multiplicative extended Kalman filter in this research (c.f. 5.5).

4.5 – Camera Modelling

In Chapter 7, the wiimote camera calibration is explained. However, before a camera can be calibrated, a specific camera model must be developed. This section develops a mathematical model for the camera.

The pinhole camera model transforms a point in the world $(X_{\mathrm{w}},\ Y_{\mathrm{w}},\ Z_{\mathrm{w}})$ to a point on the camera's image sensor $(x_{\mathrm{im}},\ y_{\mathrm{im}}) = \left(\frac{u}{w},\ \frac{v}{w}\right)$ using

$$\begin{bmatrix} u \\ v \\ w \end{bmatrix} = \underbrace{\overbrace{\begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}}^{\substack{CameraCalibration \\ Matrix}} \times \overbrace{[\mathbf{A} \quad \mathbf{t}]}^{\substack{Extrinsic \\ Params}}}_{ProjectionMatrix} \times \begin{bmatrix} X_{\mathrm{w}} \\ Y_{\mathrm{w}} \\ Z_{\mathrm{w}} \\ 1 \end{bmatrix} \tag{4.68}$$

where f_x and f_y are the focal lengths in pixel coordinates, (c_x, c_y) is the principal point in pixel coordinates, $\mathbf{A}$ is the (3×3) rotation matrix, and $\mathbf{t}$ is the (3×1) translation vector. However, this model does not take into consideration lens distortion.

To incorporate lens distortion in the model, three radial distortion coefficients k_1, k_2, k_3 and two tangential distortion coefficients p_1, p_2 are included. The new model [123] is described by

$$\begin{bmatrix} x_{\mathrm{im}} \\ y_{\mathrm{im}} \end{bmatrix} = \begin{bmatrix} f_x \cdot x_{\mathrm{p}} \\ f_y \cdot y_{\mathrm{p}} \end{bmatrix} + \begin{bmatrix} c_x \\ c_y \end{bmatrix} \tag{4.69}$$

where $(x_{\mathrm{p}}, y_{\mathrm{p}})$ is the point's location if the pinhole camera were perfect, as described by

$$\begin{bmatrix} x_{\mathrm{p}} \\ y_{\mathrm{p}} \end{bmatrix} = (1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \begin{bmatrix} x_{\mathrm{d}} \\ y_{\mathrm{d}} \end{bmatrix} + \begin{bmatrix} 2p_1 x_{\mathrm{d}} y_{\mathrm{d}} + p_2(r^2 + 2x_{\mathrm{d}}^2) \\ 2p_2 x_{\mathrm{d}} y_{\mathrm{d}} + p_1(r^2 + 2y_{\mathrm{d}}^2) \end{bmatrix} \tag{4.70}$$

$$r^2 = {x_{\mathrm{d}}}^2 + {y_{\mathrm{d}}}^2 \tag{4.71}$$

and $(x_{\mathrm{d}}, y_{\mathrm{d}})$ is the point's distorted location, which is obtained by

$$\begin{bmatrix} x_{\mathrm{d}} \\ y_{\mathrm{d}} \end{bmatrix} = \frac{1}{z_{\mathrm{c}}} \begin{bmatrix} x_{\mathrm{c}} \\ y_{\mathrm{c}} \end{bmatrix} \tag{4.72}$$

$$\begin{bmatrix} x_{\mathrm{c}} \\ y_{\mathrm{c}} \\ z_{\mathrm{c}} \end{bmatrix} = \mathbf{A} \begin{bmatrix} X_{\mathrm{w}} \\ Y_{\mathrm{w}} \\ Z_{\mathrm{w}} \end{bmatrix} + \mathbf{t} \tag{4.73}$$

where, $(x_{\mathrm{c}}, y_{\mathrm{c}}, z_{\mathrm{c}})$ is point in the camera frame. For further details of this model used, see [123] and [124].

The intrinsic parameters are f_x, f_y, c_x, c_y, k_1, k_2, k_3, p_1, and p_1, and the extrinsic parameters are contained in the rotation matrix $\mathbf{A}$ (3-DOF) and the translation vector $\mathbf{t}$ (3-DOF).

Chapter 5 – Kalman Filtering

Sensors do not provide perfect measurements and will always contain some level of unwanted random interference (i.e. noise). When greater precision is required than a sensor can deliver using determinist methods, a stochastic model can be implemented to increase the sensor's precision by accounting for unwanted noise. Arguably, the most prominent algorithms used for this task are related to the Kalman filter, which provides optimal or near optimal state estimation under certain conditions.

Unlike most algorithms, Kalman filtering requires an understanding of its derivation, since knowledge of the system modelled is directly incorporated into the algorithm. Many sources describe the (linear) Kalman filter; however, few sources provide precise details of advanced versions such as the multiplicative extended Kalman filter, and those that do often lack clarity. This chapter provides a complete, precise, and clear description of the Kalman Filter, extended Kalman filter, and most importantly, the multiplicative extended Kalman filter. For low-cost sensors such as those in this research, these algorithms are essential and are used extensively to implement the pose filter (Sect. 6.8).

5.1 – Linear Kalman Filtering

The Kalman filter was introduced in 1960 [125], and has since gained considerable prominence in a wide variety of signal processing tasks [126]. The Kalman filter is a linear recursive solution to the observer design problem, which provides an optimal estimation of the system's state by maximizing the a posteriori probability of the previous measurements. Loosely speaking, this means that by taking into account only the previous estimate with its uncertainty and the new measurement of the inertial state with its uncertainty, an updated estimate of the state can be determined that has the highest probability of being correct. It is a prediction-correction type classifier and is optimal in the sense that it minimizes the error in the state covariance [127]. In this section, Kalman filtering theory is developed and then summarized as a prediction phase and a correction phase. This filter is used to develop the position filter (Sect. 5.2).

This Kalman filter is based on two theoretical assumptions:

1. The system's dynamics are linear.
2. The system and measurement noise is additive, white, and Gaussian.

Stating Assumption 2 alternatively, the noise is not correlated in time or system state, and its amplitude follows a zero mean Gaussian distribution. Although it is rare that these strong assumptions stringently hold for real world systems, in practice, conditions are often sufficiently close that the Kalman filter produces good approximations.

5.1.1 – State Representation and Transition Equations

The Kalman filter optimally estimates a model that can be expressed with a system of differential equations of the form [128]

$$\dot{\mathbf{x}}(t) = \mathbf{F}_c\mathbf{x}(t) + \mathbf{B}_c\mathbf{u}(t) + \mathbf{w}_c(t) \tag{5.1}$$

where $\mathbf{x}$ is the state column vector (*n*-dim), an over dot represents the time-derivative, $\mathbf{F}_c$ is the continuous systems matrix (*n*×*n*), $\mathbf{u}$ is a known control vector (*c*-dim), $\mathbf{B}_c$ is a matrix (*n*×*c*) that relates the controls to the change of state, and $\mathbf{w}_c$ is the continuous process noise vector (*n*-dim), which is the noise that affects the state of the system. Furthermore, $\mathbf{w}_c$ is a Gaussian distributed multivariate random variable such that the mean and autocorrelation are

$$\mathbf{E}\{\mathbf{w}_c(t)\} = \mathbf{0}_{n\times 1} \tag{5.2}$$

$$\mathbf{E}\{\mathbf{w}_c(t)\ \mathbf{w}_c^{\mathrm{T}}(t-\tau)\} = \mathbf{Q}_c(t)\delta(\tau) \tag{5.3}$$

where $\mathbf{Q}_c$is the (*n*×*n*) continuous process noise covariance. Although the process noise often does not have a physical interpretation, it is typically used to account for the error in the model of the system dynamics.

The state of any system can only be known through some form of measurements. The Kalman filter assumes that the measurements vector **z** (*m*-dim) is linearly related to the state and subjected to additive white Gaussian noise and hence can be modeled as

$$\mathbf{z}(t) = \mathbf{H}_c\mathbf{x}(t) + \mathbf{v}_c \tag{5.4}$$

where **H** is the continuous measurement matrix (*m*×*n*) that relates the state to the measurements and $\mathbf{v}_c$ is the continuous measurement noise vector (*m*-dim), which is a Gaussian distributed multivariate random variable such that the mean and autocorrelation are

$$\mathbf{E}\{\mathbf{v}_c(t)\} = \mathbf{0}_{m\times 1} \tag{5.5}$$

$$\mathbf{E}\{\mathbf{v}_c(t)\,\mathbf{v}_c^{\mathrm{T}}(t-\tau)\} = \mathbf{R}_c(t)\delta(\tau) \tag{5.6}$$

where $\mathbf{R}_c$ is the continuous measurement noise covariance matrix (*m*×*m*).

In practice, the discrete Kalman filter is almost always used instead of the continuous Kalman filter, thus it is necessary to discretize the continuous Kalman filter by integrating and discretizing (5.1) and by discretizing (5.4). Integrating and discretizing (5.1) leads to solutions of the form

$$\mathbf{x}_k = \mathbf{\Phi}_k\mathbf{x}_{k-1} + \mathbf{D}_k\mathbf{u}_k + \mathbf{w}_k \tag{5.7}$$

where $\mathbf{\Phi}_k$ (*n*×*n*) is the discrete transition matrix (discrete fundamental matrix), and the subscript k is the time-step marking, which is defined specifically as $\mathbf{x}_k = \mathbf{x}(t_k)$. The derivation details of (5.7) are presented in the following paragraph.

Since $\mathbf{F}_c$ is considered time-invariant over the window of integration, the continuous transition matrix $\mathbf{\Phi}_c$ is found by formal integration of (5.1), with a solution of the form

$$\mathbf{\Phi}_c = e^{\mathbf{F}t} \tag{5.8}$$

which can be evaluated using a Taylor-series expansion

$$\mathbf{\Phi}_c = \sum_{i=0}^{\infty} \frac{(\mathbf{F}_c t)^i}{i!} = \mathbf{I}_{n\times n} + \mathbf{F}_c t + \frac{(\mathbf{F}_c t)^2}{2!} + \frac{(\mathbf{F}_c t)^3}{3!} + \cdots \tag{5.9}$$

Since the continuous transition matrix is linear, the discretized transition matrix can be evaluated by setting t to the time step Δt

$$\mathbf{\Phi}_k = \mathbf{\Phi}_c(t = \Delta t) \tag{5.10}$$

The matrix $\mathbf{D}_k$ (*n*×*n*) is found by

$$\mathbf{D}_k = \int_{t_{k-1}}^{t_k} \mathbf{\Phi}(\tau)\mathbf{D}\,d\tau \tag{5.11}$$

Discretization of the measurement dynamics (5.4) is straightforward

$$\mathbf{z}_k = \mathbf{H}_k\mathbf{x}_k + \mathbf{v}_k \tag{5.12}$$

Discretization of the process noise covariance (5.3) leads to

$$\mathbf{Q}_k = \int_{t_{k-1}}^{t_k} \mathbf{\Phi}_c(\tau)\mathbf{Q}_c\mathbf{\Phi}_c^{\mathrm{T}}(\tau)\, d\tau \tag{5.13}$$

and discretization of the measurement noise covariance (5.6) is simply

$$\mathbf{R}_k = \mathbf{E}\{\mathbf{v}_k {\mathbf{v}_{k-1}}^{\mathrm{T}}\} \tag{5.14}$$

The Kalman filter optimally estimates the state of the system described by (5.7) by predicting an estimate of the state and covariance at the next time step, then correcting the estimate with a measurement based on (5.12). As long as the filter is in operation, the prediction-correction cycle repeats while new measurements are taken.

5.1.2 – State and Covariance Prediction

Taking the expectation of (5.7), the *a priori state* $\mathbf{E}\{\mathbf{x}_k^-\} = \hat{\mathbf{x}}_k^-$ at the next time step is estimated by

$$\hat{\mathbf{x}}_k^- = \mathbf{\Phi}_k\hat{\mathbf{x}}_{k-1}^+ + \mathbf{D}_k\mathbf{u}_k \tag{5.15}$$

where a superscript-minus denotes the estimate at the time step immediately preceding a measurement (a priori estimate) and a superscript-plus denotes the estimate at the time step immediately following a measurement (a posteriori estimate). Using (5.15), the estimate of the state is propagated forward.

The predicted error covariance satisfies the Riccati equation of the form

$$\frac{\boldsymbol{d}}{\boldsymbol{dt}}\mathbf{P}_c(t) = \mathbf{F}_c(t)\mathbf{P}_c(t) + \mathbf{P}_c(t)\mathbf{F}_c^{\mathrm{T}}(t) + \mathbf{Q}_c(t) \tag{5.16}$$

which can be integrated formally to obtain an estimate of the *a priori error covariance matrix* (n×n)

$$\mathbf{P}_k^- = \mathbf{\Phi}_k\mathbf{P}_{k-1}^+{\mathbf{\Phi}_k}^{\mathrm{T}} + \mathbf{Q}_k \tag{5.17}$$

5.1.3 – State and Covariance Correction

The purpose of the correction phase (update phase) is to optimally estimate the *a posteriori state* and the *a posteriori error covariance matrix*. The a posteriori state $\hat{\mathbf{x}}_k^+$ is obtained by adding a correction $\widehat{\Delta\mathbf{x}}_k$ to the projected a priori state $\hat{\mathbf{x}}_k^-$, where the correction is determined by multiplying an optimal

gain $\mathbf{K}_k$ (Kalman gain) by the residual $\mathbf{r}_k$ (innovation). Moreover, the residual is the difference between the current measurement $\mathbf{z}_k$ and the estimated projected measurement $\hat{\mathbf{z}}_k$. Specifically,

$$\hat{\mathbf{x}}_k^+ = \hat{\mathbf{x}}_k^- + \mathbf{K}_k \mathbf{r}_k \tag{5.18}$$

where

$$\mathbf{r}_k = [\mathbf{z}_k - \hat{\mathbf{z}}_k] \tag{5.19}$$

$$\hat{\mathbf{z}}_k = \mathbf{H}_k \hat{\mathbf{x}}_k^- \tag{5.20}$$

The state update equation provides a minimum-variance estimate of the a posteriori state and is obtained by combining (5.18)-(5.20)

$$\hat{\mathbf{x}}_k^+ = \hat{\mathbf{x}}_k^- + \mathbf{K}_k(\mathbf{z}_k - \mathbf{H}_k \hat{\mathbf{x}}_k^-) \tag{5.21}$$

where $\mathbf{z}_k$ is the sampled measurement and the Kalman gain (*m*×*n*) is

$$\mathbf{K}_k = \mathbf{P}_k^- \mathbf{H}_k^{\mathrm{T}} \mathbf{S}_k{}^{-1} \tag{5.22}$$

and the residual covariance (*m*×*m*) is

$$\mathbf{S}_k = \mathbf{H}_k \mathbf{P}_k^- \mathbf{H}_k^{\mathrm{T}} + \mathbf{R}_k \tag{5.23}$$

The error covariance is corrected to provide the a posteriori error covariance (*n*×*n*) by

$$\mathbf{P}_k^+ = [\mathbf{I} - \mathbf{K}_k \mathbf{H}_k] \mathbf{P}_k^- [\mathbf{I} - \mathbf{K}_k \mathbf{H}_k]^{\mathrm{T}} + \mathbf{K}_k \mathbf{R}_k \mathbf{K}_k^{\mathrm{T}} \tag{5.24}$$

It is useful to note that when the gain is optimal, the a posteriori error covariance can be simplified by multiplying both sides of the Kalman gain equation (5.22) by $\mathbf{S}_k \mathbf{K}_K^{\mathrm{T}}$

$$\mathbf{K}_k \mathbf{S}_k \mathbf{K}_K^{\mathrm{T}} = \mathbf{P}_k^- \mathbf{H}_k^{\mathrm{T}} \mathbf{S}_k{}^{-1} \mathbf{S}_k \mathbf{K}_K^{\mathrm{T}} \tag{5.25}$$

$$= \mathbf{P}_k^- \mathbf{H}_k^{\mathrm{T}} \mathbf{K}_K^{\mathrm{T}} \tag{5.26}$$

By expanding (5.24) and substituting (5.23)

$$\mathbf{P}_k^+ = \mathbf{P}_k^- - \mathbf{K}_k \mathbf{H}_k \mathbf{P}_k^- - \mathbf{P}_k^- \mathbf{H}_k^{\mathrm{T}} \mathbf{K}_k^{\mathrm{T}} + \mathbf{K}_k \mathbf{S}_k \mathbf{K}_k^{\mathrm{T}} \tag{5.27}$$

and by substituting (5.26) into the last grouping of terms of (5.27)

$$\mathbf{P}_k^+ = \mathbf{P}_k^- - \mathbf{K}_k \mathbf{H}_k \mathbf{P}_k^- - \mathbf{P}_k^- \mathbf{H}_k^{\mathrm{T}} \mathbf{K}_k^{\mathrm{T}} + \mathbf{P}_k^- \mathbf{H}_k^{\mathrm{T}} \mathbf{K}_K^{\mathrm{T}} \tag{5.28}$$

the simplified a posteriori error covariance equation reduces trivially to

$$\mathbf{P}_k^+ = [\mathbf{I} - \mathbf{K}_k \mathbf{H}_k] \mathbf{P}_k^- \tag{5.29}$$

5.2 – Kalman Filter Application: Estimating Linear Trajectories

5.2.1 – Preliminaries

Using the equations derived in Sect. 5.1, a practical filter can be designed to optimally estimate position, velocity, and acceleration of motion in one dimension, based on noisy position and acceleration information from sensors. This is of practical importance since it is used to estimate the position of the wiimote for use in teleoperation, as detailed in Chapter 6.

Although our primary concern is position, velocity and acceleration are also estimated by this filter since having the first and second derivatives of displacement allows optimal estimation of any trajectory that can be expressed by a second-order polynomial equation. Furthermore, with the addition of an appropriate amount of process noise, this filter can be designed to converge to trajectories that are expressed by a high-order polynomial and by a subset of non-polynomial equations. However, there are consequences to adding too much process noise, which are discussed in Sect. 5.3.3.

Since the primary purpose of this filter is to estimate position along an axis, it will be referred to as the *position filter*.

5.2.2 – Position Filter Derivation

A three state filter is selected to estimated position x, velocity $\dot{x}$, and acceleration $\ddot{x}$. Therefore, the state vector is defined as

$$\mathbf{x} \triangleq \begin{bmatrix} x \\ \dot{x} \\ \ddot{x} \end{bmatrix} \tag{5.30}$$

With the use of the wiimote and computational algorithms, position and acceleration measurements are obtained, thus, the measurement vector is defined as

$$\mathbf{z} \triangleq \begin{bmatrix} z_p \\ z_a \end{bmatrix} \tag{5.31}$$

where z_p is the position measurement and z_a is the acceleration measurement.

Assuming constant acceleration $\ddot{x}$, integrating the basic kinematic definitions $\ddot{x} \triangleq d\dot{x}/dt$ and $\dot{x} \triangleq dx/dt$ over the window Δt, the set of equations defining the motion are trivially found

$$x = x_0 + \dot{x}_0 t + 0.5\ddot{x}_0 \mathrm{t}^2 \tag{5.32}$$

$$\dot{x} = \dot{x}_0 + \ddot{x}_0 \mathrm{t} \tag{5.33}$$

$$\ddot{x} = \ddot{x}_0 \tag{5.34}$$

$$\dddot{x} = \sigma_r \tag{5.35}$$

Derived Eqs. (5.32)-(5.35) are of the form found in basic texts on mechanics with one modification; there is a jerk-noise random variable σ_r added to the last equation to account for the inaccurate assumption of a fixed acceleration $\ddot{x}$. Although the error in the acceleration equation $\ddot{x}$ is not truly stochastic, treating it as such allows the Kalman filter to correct acceleration without increasing the complexity of the state space, which is similar to how the bias-drift ramp noise $\sigma_{\mathbf{b}}^2$ was used in Sect. 4.4.

Noticing that Eqs. (5.33)-(5.35) satisfy the form (5.1) the state vector $\mathbf{x}$ can be factored out to obtain the continuous systems matrix $\mathbf{F}_c$, and the continuous process noise vector $\mathbf{w}_c$

$$\mathbf{F}_c = \begin{bmatrix} 0 & 1 & t \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} \quad , \qquad \mathbf{w}_c = \begin{bmatrix} 0 \\ 0 \\ \sigma_r \end{bmatrix} \tag{5.36}$$

Then using (5.7)-(5.10), the discrete state vector $\mathbf{x}_k$, and the discrete transition matrix $\mathbf{\Phi}_k$ are obtained

$$\mathbf{x}_k = \begin{bmatrix} x_k \\ \dot{x}_k \\ \ddot{x}_k \end{bmatrix} \tag{5.37}$$

$$\mathbf{\Phi}_k = \begin{bmatrix} 1 & \Delta t & \Delta t^2/2 \\ 0 & 1 & \Delta t \\ 0 & 0 & 1 \end{bmatrix} \tag{5.38}$$

Note, a shortcut calculation can be made to identify the discrete transition matrix $\mathbf{\Phi}_k$ directly from (5.32)-(5.34) by comparing it with (5.7), which is possible since the additional Eq. (5.32) was calculated.

Using (5.3) and (5.13), and recalling that $\Delta t \triangleq t_k - t_{k-1}$, the discrete process noise covariance $\mathbf{Q}_k$ is calculated

$$\mathbf{Q}_k = \int_{t_{k-1}}^{t_k} \begin{bmatrix} 1 & \tau & \tau^2/2 \\ 0 & 1 & \tau \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & \sigma_r^2 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ \tau & 1 & 0 \\ \tau^2/2 & \tau & 1 \end{bmatrix} d\tau \tag{5.39}$$

$$\mathbf{Q}_k = \sigma_r^2 \begin{bmatrix} \frac{\Delta t^5}{20} & \frac{\Delta t^4}{8} & \frac{\Delta t^3}{6} \\ \frac{\Delta t^4}{8} & \frac{\Delta t^3}{3} & \frac{\Delta t^2}{2} \\ \frac{\Delta t^3}{6} & \frac{\Delta t^2}{2} & \Delta t \end{bmatrix} \tag{5.40}$$

Using (5.12), the discrete measurement vector $\mathbf{z}_k$, discrete measurement matrix $\mathbf{H}_k$, and the discrete measurement noise $\mathbf{v}_k$ are obtained

$$\mathbf{z}_k = \begin{bmatrix} z_{p,k} \\ z_{a,k} \end{bmatrix} \tag{5.41}$$

$$\mathbf{H}_k = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{5.42}$$

$$\mathbf{v}_k = \begin{bmatrix} \sigma_p \\ \sigma_a \end{bmatrix} \tag{5.43}$$

where the random variables σ_p and σ_a are the position-noise standard deviation and the acceleration-noise standard deviation, respectively. It is important to note that the measurement matrix $\mathbf{H}_k$ is derived assuming that the measurements $z_{p,k}$ and $z_{a,k}$ are in compatible units (e.g., *m* and *m/s*^{2}). Furthermore, estimation of state will be in these units (e.g., *m*, *m/s*, and *m/s*^{2}). Lastly, the random variables σ_p and σ_a must also be converted to the appropriate units.

Using (5.43) and (5.14), and assuming no cross-correlation, we obtain the discretized measurement noise covariance

$$\mathbf{R}_k = \begin{bmatrix} \sigma_p^2 & 0 \\ 0 & \sigma_a^2 \end{bmatrix} \tag{5.44}$$

Since there is no known control,

$$\mathbf{u}_k = 0 \tag{5.45}$$

$$\mathbf{D}_k = \mathbf{0}_{1\times 3} \tag{5.46}$$

5.2.3 – State and Covariance Prediction

As stated, Kalman filters operate in a two-phase, prediction-correction cycle. The prediction phase consists of estimating the a priori state $\hat{\mathbf{x}}_k^-$ and a priori covariance matrix $\mathbf{P}_k^-$, which are obtained as follows:

First, update the time step Δt and calculate the transition matrix $\mathbf{\Phi}_k$ using (5.38).

Second, since this system has no control, using (5.45) and (5.46), Eq. (5.15) reduces to

$$\hat{\mathbf{x}}_k^- = \mathbf{\Phi}_k \hat{\mathbf{x}}_{k-1}^+ \tag{5.47}$$

Propagate the state, by calculating the a priori state $\hat{\mathbf{x}}_k^-$ using (5.47), where, the a posteriori state $\hat{\mathbf{x}}_{k-1}^+$ is obtained from the previous correction phase (Sect. 5.2.4), and the initial state $\hat{\mathbf{x}}_0^+$ is determine as described in Sect. 5.3.1.

Third, calculate the discrete process noise covariance $\mathbf{Q}_k$ using (5.40).

Fourth, calculate the a priori covariance $\mathbf{P}_k^-$ using (5.17), where the a posteriori covariance $\mathbf{P}_{k-1}^+$ is obtained from the previous correction phase (Sect. 5.2.4), and the initial covariance $\mathbf{P}_0^+$ is determined as described in Sect. 5.3.1.

5.2.4 – State and Covariance Correction

In the correction phase, the predicted state and covariance are corrected by finding the a posteriori state $\hat{\mathbf{x}}_k^+$ and a posteriori covariance matrix $\mathbf{P}_k^+$, which are obtained as follows:

First, calculated the residual covariance matrix $\mathbf{S}_k$ using (5.23), where $\mathbf{H}_k$ is obtained by (5.42), and $\mathbf{R}_k$ is obtained by (5.44) using the measurement noise σ_p^2 and σ_a^2 as found by Sect. 5.3.2.

Second, calculate the Kalman gain $\mathbf{K}_k$ using (5.22).

Third, calculate the a posteriori state $\hat{\mathbf{x}}_k^+$ using (5.21), where $\hat{\mathbf{x}}_k^-$ is obtained in the previous prediction phase (Sect. 5.2.3), and $\mathbf{z}_k$ is obtained by (5.41) using the most recent position measurement $z_{p,k}$ and acceleration measurement $z_{a,k}$.

Fourth, calculate the a posteriori covariance $\mathbf{P}_k^+$ using (5.29), where $\mathbf{P}_k^-$ is obtained from the previous prediction phase (Sect. 5.2.3).

Fifth, increase the time step marking, i.e. $k = k + 1$, and repeat the prediction phase (Sect. 5.2.3).

5.3 – Initial Conditions, Noise Values, and Other Considerations

5.3.1 – Initial State and Covariance

Although Eqs. (5.1)-(5.29) formally specify the behaviour and applicability of the Kalman filter, for practical considerations, the setting of initial conditions and noise values requires discussion. Upon the first use of the prediction phase, a guess of the initial a posteriori state $\hat{\mathbf{x}}_0^+$ and a posteriori error covariance $\mathbf{P}_0^+$ must be used. Prior information should be used if known, but typically, no prior information is known. Without prior knowledge, the initial state $\hat{\mathbf{x}}_0^+$ can either be set to zero or a preliminary measurement, and the initial error covariance $\mathbf{P}_0^+$ should be set to infinity.

In practice, the programmer is never completely ignorant about the prior, thus, any high value could be used for the diagonal values, and typically, the off-diagonal values are set to zero. The Kalman filter is very insensitive to initial covariance values and will rapidly converge to the correct covariance. It often makes little difference if the values are set to infinity or unity [128], and thus minimal effort should be spent on obtaining optimal initial covariance values unless optimal state convergence is required in a very small number of measurements.

5.3.2 – Measurement Noise

There are two types of noise vectors that require setting, measurement noise and process noise. The values of the measurement noise vector $\mathbf{v}_k$can often be determined from a simple batch analysis of measurement readout data. Since the noise is assumed to additive white noise, the values of $\mathbf{v}_k$ are the standard deviation of each measurement signal. For many systems, this value is sufficiently stable that it can be treated as constant, but for some systems, the standard deviation is a function of one or more variables. A common example of such a system is in positioning-type systems where the error is a function of the distance between the transmitter and receiver. In these situations, the measurement noise matrix $\mathbf{R}_k(d)$ should be updated at each time step.

While determining $\mathbf{v}_k$, it should be verified that the noise properties resemble additive white Gaussian noise. In practice, the Kalman filter can still provide a good state approximation even if the

noise only somewhat resembles additive white Gaussian noise. When the measurement noise does not follow a Gaussian distribution, e.g. bimodal distributed noise, a more computationally expensive filter such as the particle filter must be employed to maintain high accuracy.

5.3.3 – Process Noise

Determining the process noise $\mathbf{w}_k$ is usually less straightforward and more difficult to measure directly. This is because the process noise is rarely used to model a stochastic noise in the transition dynamics of the system, but rather, is used to represent the inherent error in the mathematical model itself. Moreover, since the error in the model can often be from our lack of perfect knowledge of the physical world, it can be difficult to analyse process noise directly.

If the system dynamics model is perfectly described, then the process noise should be set to zero and the Kalman filter's state estimations will converge. Setting the process noise to zero effectively reduces the Kalman filter to the recursive least squares filter[1].

It is rare in real world applications that the dynamics model is perfect and therefore without process noise, the state estimates will diverge. The process noise counteracts the imperfection in the model and should be set high enough that convergence occurs, although, it should not be set any higher than needed. The higher the process noise the slower the rate of convergence and larger the neighbourhood of convergence, which results in a larger error. In practice, the process noise $\mathbf{w}_k$ is set through experimental methods that minimize the error covariance, while maintaining convergence. This can be done manually through trial and error or can be automated via a computational algorithm.

5.3.4 – Observability

Theoretically, there are quantitative tests that can verify that the chosen states to be filtered are observable. However, often these tests are too complicated to use and are either avoided or forgotten [128]. This topic is rarely discussed pertaining to Kalman filtering, but should be performed whenever it is not completely clear that all states are observable. This is important since a Kalman filter can be built even when the states are not fully observable.

[1] The recursive least-square filter is simply an iterative version of the very widely use batch method known as least squares, which is used to estimate the solution of an overdetermined system.

A far less formal but more practical approach to test for observability is to operate the filter under different scenarios and with different initial state estimates, then use knowledge of the system to determine if the estimates are within the bounds of reasonable expectation. The estimates should converge to within a reasonably sized neighbourhood. If the states are not observable, then the estimates will not improve in accuracy.

When using the practical test, it is important to vary the initial estimates because if the estimates are initially reasonably accurate, the Kalman filter with unobservable states will appear to be functioning correctly, but it is not until inaccurate estimates are used that the filter will reveal that it is not properly functioning. To resolve a malfunctioning Kalman filter with unobservable states, either the chosen states or the measurements must be modified until the states are observable. Outside of checking for observability, this type of practical analysis must be done for all computational algorithms, as it is common for any program to contain runtime errors that were not found during compilation.

5.4 – Extended Kalman Filter

The Kalman filter can only estimate *linear* systems, whereas the extended Kalman filter (EKF) is an adapted version that can estimate *nonlinear* systems. Unlike the Kalman filter, the EKF is not in general an optimal estimator, but typically produces good estimations if the system is not heavily nonlinear[2]. The primary purpose of this section is to develop the foundation required by the multiplicative extended Kalman filter (Sect. 5.5).

5.4.1 – State Representation and Transition Equations

The EKF can estimate the state of system dynamics expressed by the first order nonlinear differential equation

$$\dot{\mathbf{x}} = \boldsymbol{f}_c(\mathbf{x}, \mathbf{u}, \mathbf{w}_c, t) \tag{5.48}$$

where $\boldsymbol{f}_c$ is a nonlinear system function, and the process noise vector $\mathbf{w}_c$ is a multivariate Gaussian random variable described by (5.2)-(5.3). Additionally, the measurement dynamics is described by

$$\mathbf{z}_c = \boldsymbol{h}_c(\mathbf{x}, \mathbf{v}_c, t) \tag{5.49}$$

[2] For heavily nonlinear systems, the unscented Kalman filter often outperforms the extended Kalman filter and therefore should be considered instead for systems of this type.

where $\boldsymbol{h}_c$ is a nonlinear measurement function of state, and the measurement noise vector $\mathbf{w}_c$ is a multivariate Gaussian random variable described by (5.5)-(5.6).

Since system function $\boldsymbol{f}_c$ and the measurement function $\boldsymbol{h}_c$are nonlinear, they require linearization. A first order approximation must be used in the Riccati equations for the system dynamics matrix $\mathbf{F}_k$, the measurement matrix $\mathbf{H}_k$, and two noise linearization matrices $\mathbf{W}_k$ and $\mathbf{V}_k$. Both $\mathbf{F}_k$ and $\mathbf{H}_k$ are obtained by finding the Jacobian matrices of the respective functions $\boldsymbol{f}_c$ and $\boldsymbol{h}_c$ about the operating point $\hat{\mathbf{x}}$

$$\mathbf{F}_k = \left.\frac{\partial \boldsymbol{f}(\mathbf{x})}{\partial \mathbf{x}}\right|_{\mathbf{x}=\hat{\mathbf{x}}_{k-1}^{+}} \tag{5.50}$$

$$\mathbf{H}_k = \left.\frac{\partial \boldsymbol{h}(\mathbf{x})}{\partial \mathbf{x}}\right|_{\mathbf{x}=\hat{\mathbf{x}}_{k}^{-}} \tag{5.51}$$

Additionally, the noise linearization matrices $\mathbf{W}_k$ and $\mathbf{V}_k$ are obtained by

$$\mathbf{W}_k = \left.\frac{\partial \boldsymbol{f}(\mathbf{x})}{\partial \mathbf{w}}\right|_{\mathbf{x}=\hat{\mathbf{x}}_{k-1}^{+}} \tag{5.52}$$

$$\mathbf{V}_k = \left.\frac{\partial \boldsymbol{h}(\mathbf{x})}{\partial \mathbf{v}}\right|_{\mathbf{x}=\hat{\mathbf{x}}_{k}^{-}} \tag{5.53}$$

As in the linear Kalman filter, the continuous transition matrix $\boldsymbol{\Phi}_c$ can be obtained from (5.8)-(5.9) and the discrete transition matrix $\boldsymbol{\Phi}_k$ from (5.10). The $\boldsymbol{\Phi}_k$ will not be used in the state propagation, and as shown by [128], using more than the first two terms of the Taylor series will not improve performance in general. Thus, $\boldsymbol{\Phi}_c$ and $\boldsymbol{\Phi}_k$ can be effectively approximated by

$$\boldsymbol{\Phi}_c \approx \mathbf{I} + \mathbf{F}_k t \tag{5.54}$$

$$\boldsymbol{\Phi}_k \approx \mathbf{I} + \mathbf{F}_k \Delta t \tag{5.55}$$

The discrete process noise covariance $\mathbf{Q}_k$ is obtained by

$$\mathbf{Q}_k = \int_{t_{k-1}}^{t_k} \boldsymbol{\Phi}_c(t_k,\tau)\mathbf{W}_c(t_k,\tau)\mathbf{Q}_c\mathbf{W}_c(t_k,\tau)^{\mathrm{T}}\boldsymbol{\Phi}_c^{\mathrm{T}}(t_k,\tau)\,d\tau \tag{5.56}$$

and the discrete measurement noise covariance is obtained by

$$\mathbf{R}_k = \mathbf{V}_k[\mathbf{E}\{\mathbf{v}_k{\mathbf{v}_{k-1}}^{\mathrm{T}}\}]\mathbf{V}_k^{\mathrm{T}} \tag{5.57}$$

However, the measurement noise is usually assumed to be linearly additive to the state and thus typically $\mathbf{V}_k = \mathbf{I}$. Therefore, (5.57) reduces to the linear Kalman filter's equation for the measurement noise covariance (5.14).

5.4.2 – State and Covariance Prediction

The discrete transition matrix $\mathbf{\Phi}_k$ is an approximation and is only required in the Riccati equations; hence, the performance can be improved by using the actual nonlinear equation to propagate the state to the next time step

$$\hat{\mathbf{x}}_k^- = \mathbf{E}\left\{\int_0^{t_k} \boldsymbol{f}(\mathbf{x}(t_k,\tau),\mathbf{u}(t_k,\tau),\mathbf{w}(t_k,\tau),t_k,\tau)\,d\tau\right\} \tag{5.58}$$

$$\approx \int_0^{t_k} \boldsymbol{f}(\hat{\mathbf{x}}^+(t_k,\tau),\mathbf{u}_k)\,d\tau \tag{5.59}$$

$$= \hat{\mathbf{x}}_{k-1}^+ + \int_{t_{k-1}}^{t_k} \boldsymbol{f}(\hat{\mathbf{x}}^+(\tau),\mathbf{u}_k)\,d\tau \tag{5.60}$$

where $t_k = t_{k-1} + \Delta t$. The solution to (5.60) is not known in closed form but can be found through numerical integration. When the time step Δt is very small and the system not very stiff[3], (5.60) can be reduced using the Euler method (with a step size of Δt) to

$$\hat{\mathbf{x}}_k^- = \hat{\mathbf{x}}_{k-1}^+ + \boldsymbol{f}_c(\hat{\mathbf{x}}_{k-1}^+,\mathbf{u}_k)\Delta t \tag{5.61}$$

When the Euler method (first order Runge-Kutta) becomes unstable or does not provide sufficient accuracy, $\boldsymbol{f}(\hat{\mathbf{x}}_{k-1}^+,\mathbf{u}_k)\Delta t$ must be broken into either smaller steps, or preferably, a more accurate numerical method should be employed. Although there are many numerical integration methods, the fourth order Runge-Kutta method (RK4) is arguably the most widely used due to its effectiveness. Using RK4, the solution to (5.60) is obtain by

$$\hat{\mathbf{x}}_i^- = \hat{\mathbf{x}}_{i-1}^+ + \mathbf{6}(K_1 + 2K_2 + 2K_3 + K_4)\Delta t_j \tag{5.62}$$

and

[3] A system's stiffness is a measure of the behaviour of a system to cause some numerical methods to become unstable.

$$K_1 = \boldsymbol{f}_c(\hat{\mathbf{x}}_{i-1}^+, u_k, t_{i-1}) \tag{5.63}$$

$$K_2 = \boldsymbol{f}_c\left(\hat{\mathbf{x}}_{i-1}^+ + \frac{K_1}{2}, u_k, t_{i-1} + \frac{\Delta t_j}{2}\right) \tag{5.64}$$

$$K_3 = \boldsymbol{f}_c\left(\hat{\mathbf{x}}_{i-1}^+ + \frac{K_2}{2}, u_k, t_{i-1} + \frac{\Delta t_j}{2}\right) \tag{5.65}$$

$$K_4 = \boldsymbol{f}_c\left(\hat{\mathbf{x}}_{i-1}^+ + K_3, u_k, t_{i-1} + \Delta t_j\right) \tag{5.66}$$

where step $i = jk$, the number of steps $j \in \mathbb{Z}^+$, and change in time between steps $\Delta t_j = \frac{\Delta t}{j}$. It should be noted that any approximation-induced error should be accounted for by an appropriate value of process noise.

With the linearized equation (5.51)-(5.57), the equations are of the Riccati form (5.16). Similar to the linear Kalman filter, the a priori error covariance $\mathbf{P}_k^-$ of the extended Kalman filter is obtained by (5.17).

5.4.3 – State and Covariance Correction

The extended Kalman filter corrects the a priori state $\hat{\mathbf{x}}_k^-$ by adding a gain multiplied by a residual. Since this computation does not require linearization, instead of using the approximate measurement matrix to calculate the expected measurement $\hat{\mathbf{z}}_k(5.20)$(7.20), the expectation of the full nonlinear measurement function can be used

$$\hat{\mathbf{z}}_k = \mathbf{E}\{\boldsymbol{h}_c(\mathbf{x}_k, \mathbf{v}_k, t_k)\} = \boldsymbol{h}_c(\hat{\mathbf{x}}_k^-, t_k) \tag{5.67}$$

Using (5.18), (5.19), and (5.67) instead of (5.20), the a posteriori state estimation can be obtained

$$\hat{\mathbf{x}}_k^+ = \hat{\mathbf{x}}_k^- + \mathbf{K}_k\left(\mathbf{z}_k - \boldsymbol{h}_c(\hat{\mathbf{x}}_k^-, t_k)\right) \tag{5.68}$$

where the Kalman gain $\mathbf{K}_k$is obtain from (5.22), the residual covariance $\mathbf{S}_k$ is obtained from (5.23), and the a posteriori error covariance $\mathbf{P}_k^+$ is obtained from (5.29).

5.5 – Multiplicative Extended Kalman Filter

The sole purpose of detailing the EKF is to provide the foundation for the Multiplicative Extended Kalman Filter (MEKF), which is used to estimate attitude. The teleoperation system uses a MEKF to implement the attitude filter, which is an essential component required in the pose filter (q.v. Sect. 6.8) of the wiimote. For an overview of Kalman filtering methods for attitude estimation up until 1981 refer to [129] and for a survey focusing from 1981 to 2007 refer to [130].

5.5.1 – Attitude Representations Used in Filtering

There are many different representations for attitude. These methods have been discussed by Stupnagel [131] and Markley [132], and those directly related to this research are described in Sect. 4.1.

The rotation matrix is a nonsingular representation and since it can be directly applied to vectors, it is a reasonable first candidate for a representation. Not surprisingly, it was used in early strapdown navigation systems as an attitude representation [133][134]. However, with nine parameters specifying 3-DOF motion, it is a highly redundant representation. Furthermore, round-off, quantization, and truncation errors in the propagation of attitude results in failure to satisfy the matrix's orthogonality constraint [125][135][136]. There are methods to reorthogonalize the rotation matrix; however, it has been proven that optimal orthogonalization requires computing a matrix square root [136], which is an expensive operation. For these drawbacks, the rotation matrix has fallen out of widespread use.

Other earlier models used Euler angles [137], and later models used rotation vectors [138]. However, both representations use expensive transcendental functions and have singularities. Moreover, a globally nonsingular three-dimensional parameterization of the rotation group has been proven topologically impossible [139].

The unit quaternion has the lowest dimension of any nonsingular attitude parameterization. Furthermore, it is bilinear in quaternion components and in the angular velocity vector [145]. However, the linear Riccati equations violate the quaternion's nonlinear norm constraint, although, a number of methods have been developed to circumvent this drawback [140][141][142][143][144]. One solution is to incorporate the norm constraint into the 4×4 covariance by projecting it onto a 3×3 matrix. This can be performed without loss of information since the norm constraint results in the 4×4 covariance matrix being rank deficient [125]. Alternatively, a three-component attitude-error vector can be used in

the measurement update equations, while maintaining the use of the quaternion for propagation. The three-component attitude-error vector naturally has a 3×3 covariance matrix that has been shown to be equivalent to the 3×3 projected covariance [125], making both methods equivalent but with the former having a conceptually stronger basis [145]. The use of the quaternion and three-component attitude-error vector method stated here has become known as the Multiplicative Extended Kalman Filter (MEKF) and is the method employed in this thesis.

The remainder of this chapter is mathematically involved due to a complete derivation of the MEKF. It is possible to implement attitude estimation for this exact teleoperation system by skipping to Sect. 5.5.9 and Sect. 5.5.10; however, even a slight change in the system dynamics such as adding or modifying the sensor or modifying the state space will require a complete re-derivation. Therefore, it is of practical importance to include the complete derivation. Furthermore, providing the complete derivation clearly increases the accessibility of this advanced filter to a wider audience, which satisfies one of the contributions of this thesis.

5.5.2 – State Representation

The relevant states for the MEKF are those that pertain to the rotation, which are the relative attitude ${}^{\mathrm{L}}_{\mathrm{G}}\bar{q}_{\mathrm{k}}$ and the gyro bias vector $\mathbf{b}$. Hence, the state vector $\mathbf{x}$ is defined as

$$\mathbf{x}(t) \triangleq \begin{bmatrix} \bar{q}(t) \\ \mathbf{b}(t) \end{bmatrix} \tag{5.69}$$

From (4.39) and (4.63), the dynamics for the state are governed by

$$\dot{\bar{q}}(t) = \frac{1}{2}\boldsymbol{\Omega}(\boldsymbol{\omega})\bar{q}(t) \tag{5.70}$$

$$\dot{\mathbf{b}}(t) = \mathbf{n}_{\mathbf{b}}(t) \tag{5.71}$$

Taking the expectation of (5.70), (5.71) and (4.60) gives

$$\dot{\hat{\bar{q}}}(t) = \mathbf{E}\{\dot{\bar{q}}(t)\} = \frac{1}{2}\boldsymbol{\Omega}\big(\widehat{\boldsymbol{\omega}}(t)\big)\hat{\bar{q}}(t) \tag{5.72}$$

$$\dot{\hat{\mathbf{b}}} = \mathbf{E}\big\{\dot{\mathbf{b}}(t)\big\} = \mathbf{0}_{3\times 1} \tag{5.73}$$

where

$$\hat{\boldsymbol{\omega}}(t) = \mathbf{E}\{\boldsymbol{\omega}(t)\} = \boldsymbol{g}(t) - \mathbf{b}(t) \tag{5.74}$$

5.5.3 – Quaternion State Transition Equations

The quaternion state projection is of the form

$$\overset{\mathrm{L}}{\mathrm{G}}\bar{q}_{\mathrm{k}}^{-} = \boldsymbol{\Phi}_k^{\bar{q}} \overset{\mathrm{L}}{\mathrm{G}}\bar{q}_{\mathrm{k}-1}^{+} \tag{5.75}$$

Note, pre-subscript L and pre-superscript G will be suppressed for brevity, except when it is required for clarity.

An exact closed-form solution of the quaternion transition matrix $\boldsymbol{\Phi}_k^{\bar{q}}$ is currently unavailable in the literature. To simplify the problem, a first-order hold is assumed[4]. The resulting first-order quaternion transition matrix is obtain by

$$\boldsymbol{\Phi}_k^{\bar{q},1} \approx \boldsymbol{\Phi}_k^{\bar{q},0}(\bar{\boldsymbol{\omega}}_k) + \frac{1}{48}\big(\boldsymbol{\Omega}(\dot{\boldsymbol{\omega}}_{k-1})\boldsymbol{\Omega}(\boldsymbol{\omega}_{k-1}) - \boldsymbol{\Omega}(\boldsymbol{\omega}_{k-1})\boldsymbol{\Omega}(\dot{\boldsymbol{\omega}}_{k-1})\big)\Delta t^3 \tag{5.76}$$

where the average angular rate $\bar{\boldsymbol{\omega}}_k$ and the angular acceleration $\dot{\boldsymbol{\omega}}_k$ is

$$\bar{\boldsymbol{\omega}}_k = \frac{\boldsymbol{\omega}_k + \boldsymbol{\omega}_{k-1}}{2} \tag{5.77}$$

$$\dot{\boldsymbol{\omega}}_k \approx \frac{\boldsymbol{\omega}_{k+1} - \boldsymbol{\omega}_k}{\Delta t} \tag{5.78}$$

and the zeroth-order quaternion integrator for the average angular rate is

$$\boldsymbol{\Phi}_k^{\bar{q},0}(\bar{\boldsymbol{\omega}}_k) = \left[\cos\left(\frac{|\bar{\boldsymbol{\omega}}_k|}{2}\Delta t\right)\cdot \mathbf{I}_{4\times4} + \frac{1}{|\boldsymbol{\omega}_k|}\sin\left(\frac{|\bar{\boldsymbol{\omega}}_k|}{2}\Delta t\right)\cdot \boldsymbol{\Omega}(\bar{\boldsymbol{\omega}}_k)\right] \tag{5.79}$$

When $|\bar{\boldsymbol{\omega}}_k|$ is very small, (5.79) will lead to numerical instability by repeated use of (5.75). To allow (5.79) to be stable for small values of $|\bar{\boldsymbol{\omega}}_k|$, the limit of (5.79) it taken as $|\bar{\boldsymbol{\omega}}_k|$ goes to zero

$$\lim_{|\bar{\boldsymbol{\omega}}_k|\to 0} \boldsymbol{\Phi}_k^{\bar{q},0} = \mathbf{I}_{4\times4} + \frac{1}{2}\boldsymbol{\Omega}(\bar{\boldsymbol{\omega}}_k)\Delta t \tag{5.80}$$

Detailed derivations of (5.76), (5.79), and (5.80) are presented in Appendix B.

[4] The first-order hold assumes that a first-order (linear) approximation can be used on the interval between time steps. It can be used to simplify a difference function prior to integration, when the integral of the function is either difficult or does not exist.

5.5.4 – Error State Representation

Traditionally, the error of a state is defined as the actual state value minus the measured (or estimated, projected, etc.) state value. Applying this definition of the error to the quaternion will violate the quaternion unit norm constraint, which will cause the quaternion to become singular. Moreover, the subtraction operation does not have intuitive meaning for a quaternion. Since rotational transformations are performed using multiplication instead of addition, an accurate model of rotation would require the error to be multiplicative as opposed to the traditional subtractive definition. Furthermore, multiplication preserves the quaternion's unit norm constraint, whereas subtraction does not.

The attitude of the true local frame wrt the global frame (rotation from the true local frame to the global frame) ${}^{\mathrm{L}}_{\mathrm{G}}\bar{q}$ is expressed by

$$ {}^{\mathrm{L}}_{\mathrm{G}}\bar{q} = {}^{\mathrm{L}}_{\hat{\mathrm{L}}}\delta\bar{q} \otimes {}^{\hat{\mathrm{L}}}_{\mathrm{G}}\hat{\bar{q}} \tag{5.81} $$

where ${}^{\hat{\mathrm{L}}}_{\mathrm{G}}\hat{\bar{q}}$ is the estimated quaternion attitude of the estimated local frame $\hat{\mathrm{L}}$ wrt the global frame G, and ${}^{\mathrm{L}}_{\hat{\mathrm{L}}}\delta\bar{q}$ is the quaternion error rotation from the estimated local frame $\hat{\mathrm{L}}$ to the true local frame L (attitude of the true local frame wrt estimated local frame). Rearranging in terms of the quaternion error gives

$$ {}^{\mathrm{L}}_{\hat{\mathrm{L}}}\delta\bar{q} = {}^{\mathrm{L}}_{\mathrm{G}}\bar{q} \otimes {}^{\hat{\mathrm{L}}}_{\mathrm{G}}\hat{\bar{q}}^{-1} \tag{5.82} $$

Since the quaternion error $\delta\bar{q}$ is generally very small, the small angle approximation (A.25) is used

$$ \delta\bar{q} = \begin{bmatrix} \delta\mathbf{q} \\ \delta q_4 \end{bmatrix} \tag{5.83} $$

$$ = \begin{bmatrix} \mathbf{k}\sin(\delta\theta/2) \\ \cos(\delta\theta/2) \end{bmatrix} \tag{5.84} $$

$$ \delta\bar{q} \approx \begin{bmatrix} \frac{1}{2}\delta\boldsymbol{\theta} \\ 1 \end{bmatrix} \tag{5.85} $$

Additionally, the gyro error bias is defined as

$$ \Delta\mathbf{b} \triangleq \mathbf{b} - \hat{\mathbf{b}} \tag{5.86} $$

where $\hat{\mathbf{b}}$ is the estimated bias. Using (5.85) and (5.86), the error state is defined as

$$\tilde{\mathbf{x}} \triangleq \begin{bmatrix} \delta\mathbf{q} \\ \Delta\mathbf{b} \end{bmatrix} \tag{5.87}$$

5.5.5 – Continuous Error State Derivation

Following [146], the derivation of the continuous error state begins by taking the derivative (5.81) by using the quaternion chain rule (4.40) and dropping the reference frame sub- and superscripts for brevity

$$\dot{\bar{q}} = \dot{\delta\bar{q}} \otimes \hat{\bar{q}} + \delta\bar{q} \otimes \dot{\hat{\bar{q}}} \tag{5.88}$$

Substituting (4.37) and (5.72) into (5.88), and rearranging

$$\frac{1}{2}\left(\delta\bar{q} \otimes \begin{bmatrix} \boldsymbol{\omega} \\ 0 \end{bmatrix} \otimes \bar{q}\right) = \dot{\delta\bar{q}} \otimes \hat{\bar{q}} + \delta\bar{q} \otimes \frac{1}{2}\left(\delta\bar{q} \otimes \begin{bmatrix} \hat{\boldsymbol{\omega}} \\ 0 \end{bmatrix} \otimes \hat{\bar{q}}\right) \tag{5.89}$$

$$\frac{1}{2}\begin{bmatrix} \boldsymbol{\omega} \\ 0 \end{bmatrix} \otimes \bar{q} = \dot{\delta\bar{q}} \otimes \hat{\bar{q}} + \delta\bar{q} \otimes \left(\frac{1}{2}\begin{bmatrix} \hat{\boldsymbol{\omega}} \\ 0 \end{bmatrix} \otimes \hat{\bar{q}}\right) \tag{5.90}$$

$$\dot{\delta\bar{q}} \otimes \hat{\bar{q}} = \frac{1}{2}\left(\begin{bmatrix} \boldsymbol{\omega} \\ 0 \end{bmatrix} \otimes \bar{q} - \delta\bar{q} \otimes \begin{bmatrix} \hat{\boldsymbol{\omega}} \\ 0 \end{bmatrix} \otimes \hat{\bar{q}}\right) \tag{5.91}$$

$$\dot{\delta\bar{q}} \otimes \hat{\bar{q}} \otimes \hat{\bar{q}}^{-1} = \frac{1}{2}\left(\begin{bmatrix} \boldsymbol{\omega} \\ 0 \end{bmatrix} \otimes \bar{q} - \delta\bar{q} \otimes \begin{bmatrix} \hat{\boldsymbol{\omega}} \\ 0 \end{bmatrix} \otimes \hat{\bar{q}}\right) \otimes \hat{\bar{q}}^{-1} \tag{5.92}$$

$$\dot{\delta\bar{q}} = \frac{1}{2}\left(\begin{bmatrix} \boldsymbol{\omega} \\ 0 \end{bmatrix} \otimes \bar{q} \otimes \hat{\bar{q}}^{-1} - \delta\bar{q} \otimes \begin{bmatrix} \hat{\boldsymbol{\omega}} \\ 0 \end{bmatrix}\right) \tag{5.93}$$

$$\dot{\delta\bar{q}} = \frac{1}{2}\left(\begin{bmatrix} \boldsymbol{\omega} \\ 0 \end{bmatrix} \otimes \delta\bar{q} - \delta\bar{q} \otimes \begin{bmatrix} \hat{\boldsymbol{\omega}} \\ 0 \end{bmatrix}\right) \tag{5.94}$$

Combining (4.60) and (5.74)

$$\boldsymbol{\omega} = \hat{\boldsymbol{\omega}} - \Delta\mathbf{b} - \mathbf{n_b} \tag{5.95}$$

Substituting (5.95) into (5.94)

$$\dot{\delta\bar{q}} = \frac{1}{2}\left(\begin{bmatrix} \hat{\boldsymbol{\omega}} - \Delta\mathbf{b} - \mathbf{n_b} \\ 0 \end{bmatrix} \otimes \delta\bar{q} - \delta\bar{q} \otimes \begin{bmatrix} \hat{\boldsymbol{\omega}} \\ 0 \end{bmatrix}\right) \tag{5.96}$$

$$\dot{\delta\bar{q}} = \frac{1}{2}\left(\begin{bmatrix} \hat{\boldsymbol{\omega}} \\ 0 \end{bmatrix} \otimes \delta\bar{q} - \delta\bar{q} \otimes \begin{bmatrix} \hat{\boldsymbol{\omega}} \\ 0 \end{bmatrix}\right) - \frac{1}{2}\begin{bmatrix} \Delta\mathbf{b} + \mathbf{n_b} \\ 0 \end{bmatrix} \otimes \delta\bar{q} \tag{5.97}$$

$$\dot{\delta\bar{q}} = \frac{1}{2}\left(\begin{bmatrix} -[\hat{\boldsymbol{\omega}}\times] & \hat{\boldsymbol{\omega}} \\ -\hat{\boldsymbol{\omega}}^{\mathrm{T}} & 0 \end{bmatrix}\delta\bar{q} - \begin{bmatrix} [\hat{\boldsymbol{\omega}}\times] & \hat{\boldsymbol{\omega}} \\ -\hat{\boldsymbol{\omega}}^{\mathrm{T}} & 0 \end{bmatrix}\delta\bar{q}\right) - \frac{1}{2}\begin{bmatrix} \Delta\mathbf{b} + \mathbf{n_b} \\ 0 \end{bmatrix} \otimes \delta\bar{q} \tag{5.98}$$

$$\dot{\delta\bar{q}} = \frac{1}{2}\begin{bmatrix} -2[\hat{\boldsymbol{\omega}}\times] & \mathbf{0}_{3\times1} \\ \mathbf{0}_{1\times3} & 0 \end{bmatrix}\delta\bar{q} - \frac{1}{2}\begin{bmatrix} \Delta\mathbf{b} + \mathbf{n_b} \\ 0 \end{bmatrix} \otimes \delta\bar{q} \tag{5.99}$$

Using (4.18) and (4.21)

$$\dot{\delta\bar{q}} = \frac{1}{2}\begin{bmatrix} -2[\hat{\boldsymbol{\omega}}\times] & \mathbf{0}_{3\times1} \\ -\mathbf{0}_{1\times3} & 0 \end{bmatrix}\delta\bar{q} - \frac{1}{2}\begin{bmatrix} -[(\Delta\mathbf{b} + \mathbf{n_b})\times] & (\Delta\mathbf{b} + \mathbf{n_b}) \\ -(\Delta\mathbf{b} + \mathbf{n_b}) & 0 \end{bmatrix}\delta\bar{q} \tag{5.100}$$

$$\dot{\delta\bar{q}} = \frac{1}{2}\begin{bmatrix} -2[\hat{\boldsymbol{\omega}}\times] & \mathbf{0}_{3\times1} \\ -\mathbf{0}_{1\times3} & 0 \end{bmatrix}\delta\bar{q} - \frac{1}{2}\begin{bmatrix} -[(\Delta\mathbf{b} + \mathbf{n_b})\times] & (\Delta\mathbf{b} + \mathbf{n_b}) \\ -(\Delta\mathbf{b} + \mathbf{n_b}) & 0 \end{bmatrix}\begin{bmatrix} \delta\mathbf{q} \\ 1 \end{bmatrix} \tag{5.101}$$

Expanding and collecting second order terms

$$\dot{\delta\bar{q}} = \frac{1}{2}\begin{bmatrix} -2[\hat{\boldsymbol{\omega}}\times] & \mathbf{0}_{3\times1} \\ -\mathbf{0}_{1\times3} & 0 \end{bmatrix}\delta\bar{q} - \frac{1}{2}\begin{bmatrix} \Delta\mathbf{b} + \mathbf{n_b} \\ 0 \end{bmatrix} + \boldsymbol{\mathcal{O}}\left(|\Delta\mathbf{b}||\delta\mathbf{q}|, \left|\mathbf{n}_{\mathbf{b},c}\right||\delta\mathbf{q}|\right) \tag{5.102}$$

Neglecting second order terms in (5.102) and separating the error rate quaternion $\dot{\delta\bar{q}}$ into components

$$\dot{\delta\bar{q}} = \begin{bmatrix} \dot{\delta\mathbf{q}} \\ \dot{\delta q_4} \end{bmatrix} \approx \begin{bmatrix} \frac{1}{2}\dot{\delta\boldsymbol{\theta}} \\ 1 \end{bmatrix} \approx \begin{bmatrix} -\hat{\boldsymbol{\omega}} \times \delta\boldsymbol{\theta} - \frac{1}{2}(\delta\mathbf{b} + \mathbf{n}_{\boldsymbol{\omega}}) \\ 0 \end{bmatrix} \tag{5.103}$$

Therefore,

$$\dot{\delta\boldsymbol{\theta}} \approx -\hat{\boldsymbol{\omega}} \times \delta\boldsymbol{\theta} - \Delta\mathbf{b} - \mathbf{n}_{\boldsymbol{\omega}} \tag{5.104}$$

Taking the time-derivative of (5.86) and combining with (5.71) and (5.73)

$$\dot{\Delta\mathbf{b}} = \dot{\mathbf{b}} - \dot{\hat{\mathbf{b}}} = \mathbf{n_b} \tag{5.105}$$

Combining (5.104) and (5.105), the error-state dynamics is attained by

$$\begin{bmatrix} \dot{\delta\boldsymbol{\theta}} \\ \dot{\Delta\mathbf{b}} \end{bmatrix} = \begin{bmatrix} -[\boldsymbol{\omega}\times] & -\mathbf{I}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \end{bmatrix}\begin{bmatrix} \delta\mathbf{q} \\ \Delta\mathbf{b} \end{bmatrix} + \begin{bmatrix} -\mathbf{I}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{I}_{3\times3} \end{bmatrix}\begin{bmatrix} \mathbf{n}_{\boldsymbol{\omega}} \\ \mathbf{n_b} \end{bmatrix} \tag{5.106}$$

Eq. (5.106) can be expressed in the standard Kalman filter state dynamics form of (5.1), leading to the standard form error-state dynamics

$$\dot{\tilde{\mathbf{x}}} = \tilde{\mathbf{F}}_c\tilde{\mathbf{x}} + \mathbf{W}_c\mathbf{n} \tag{5.107}$$

with

$$\tilde{\mathbf{F}}_c = \begin{bmatrix} -[\boldsymbol{\omega}\times] & -\mathbf{I}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \end{bmatrix} \tag{5.108}$$

$$\mathbf{W}_c = \begin{bmatrix} -\mathbf{I}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{I}_{3\times3} \end{bmatrix} \tag{5.109}$$

$$\dot{\tilde{\mathbf{x}}} = \begin{bmatrix} \dot{\delta\boldsymbol{\theta}} \\ \dot{\Delta\mathbf{b}} \end{bmatrix} \tag{5.110}$$

$$\mathbf{n} = \begin{bmatrix} \mathbf{n}_{\boldsymbol{\omega}} \\ \mathbf{n}_{\mathbf{b}} \end{bmatrix} \tag{5.111}$$

5.5.6 – Discrete Error-State Transition

The discrete error-state transition matrix $\widetilde{\boldsymbol{\Phi}}_k$ propagates the error state to the next time step and is the discretized solution of the error-state dynamics (5.107), which is a first-order system of ODEs. Following [146], the continuous error-state transition has solutions of the form

$$\widetilde{\boldsymbol{\Phi}}_c(t) = \mathrm{e}^{(\mathbf{F}_c t)} \tag{5.112}$$

and with its discrete form found by

$$\widetilde{\boldsymbol{\Phi}}_k = \widetilde{\boldsymbol{\Phi}}_c(\Delta t) \tag{5.113}$$

Taking the Taylor-series expansion of the matrix exponential gives

$$\widetilde{\boldsymbol{\Phi}}_k = \mathbf{I}_{6\times6} + \tilde{\mathbf{F}}_c\Delta t + \frac{1}{2!}\tilde{\mathbf{F}}_c^2\Delta t^2 + \frac{1}{3!}\tilde{\mathbf{F}}_c^3\Delta t^3 + \cdots \tag{5.114}$$

where

$$\left.\begin{aligned} \tilde{\mathbf{F}}_c &= \begin{bmatrix} -[\widehat{\boldsymbol{\omega}}\times] & -\mathbf{I}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \end{bmatrix} \quad , \quad \tilde{\mathbf{F}}_c^2 = \begin{bmatrix} [\widehat{\boldsymbol{\omega}}\times]^2 & [\widehat{\boldsymbol{\omega}}\times] \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \end{bmatrix} \\ \tilde{\mathbf{F}}_c^3 &= \begin{bmatrix} -[\widehat{\boldsymbol{\omega}}\times]^3 & -[\widehat{\boldsymbol{\omega}}\times]^2 \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \end{bmatrix} \quad , \quad \tilde{\mathbf{F}}_c^4 = \begin{bmatrix} [\widehat{\boldsymbol{\omega}}\times]^4 & [\widehat{\boldsymbol{\omega}}\times]^3 \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \end{bmatrix} \end{aligned}\right\} \tag{5.115}$$

Using the notation from [125], the error-state transition matrix can be written as

$$\widetilde{\boldsymbol{\Phi}}_k = \begin{bmatrix} \boldsymbol{\Theta} & \boldsymbol{\Psi} \\ \mathbf{0}_{3\times3} & \mathbf{I}_{3\times3} \end{bmatrix} \tag{5.116}$$

Combining (5.114) with (5.115), and then factoring $\widetilde{\boldsymbol{\Phi}}_k$ in the form (5.116), $\boldsymbol{\Theta}$ is obtained by

$$\boldsymbol{\Theta} = \mathbf{I}_{3\times3} - [\widehat{\boldsymbol{\omega}}\times]\Delta t + \frac{1}{2!}[\widehat{\boldsymbol{\omega}}\times]^2\Delta t^2 - \frac{1}{3!}[\widehat{\boldsymbol{\omega}}\times]^3\Delta t^3 + \cdots \tag{5.117}$$

Substituting $[\widehat{\boldsymbol{\omega}}\times]^x$ with (A.10)-(A.17) and reorganising

$$\boldsymbol{\Theta} = \mathbf{I}_{3\times3} + \left(-\Delta t + \frac{1}{3!}|\widehat{\boldsymbol{\omega}}|^2\Delta t^3 - \cdots\right)[\widehat{\boldsymbol{\omega}}\times] + \left(\frac{1}{2!}\Delta t^2 + \frac{1}{4!}|\widehat{\boldsymbol{\omega}}|^2\Delta t^4 + \cdots\right)[\widehat{\boldsymbol{\omega}}\times]^2 \tag{5.118}$$

$$\begin{aligned}\boldsymbol{\Theta} = \mathbf{I}_{3\times3} &- \frac{1}{|\widehat{\boldsymbol{\omega}}|}\left(|\widehat{\boldsymbol{\omega}}|\Delta t + \frac{1}{3!}|\widehat{\boldsymbol{\omega}}|^3\Delta t^3 - \cdots\right)[\widehat{\boldsymbol{\omega}}\times] \\ &+ \frac{1}{|\widehat{\boldsymbol{\omega}}|^2}\left(1 - \left(1 - \frac{1}{2!}|\widehat{\boldsymbol{\omega}}|^2\Delta t^2 + \frac{1}{4!}|\widehat{\boldsymbol{\omega}}|^4\Delta t^4 + \cdots\right)\right)[\widehat{\boldsymbol{\omega}}\times]^2\end{aligned} \tag{5.119}$$

Noticing that (5.119) contains the Taylor expansion of sine and cosine,

$$\boldsymbol{\Theta} = \mathbf{I}_{3\times3} + \frac{1}{|\widehat{\boldsymbol{\omega}}|}\sin(|\widehat{\boldsymbol{\omega}}|\Delta t)\,[\widehat{\boldsymbol{\omega}}\times] + \frac{1}{|\widehat{\boldsymbol{\omega}}|^2}(1 - \cos(|\widehat{\boldsymbol{\omega}}|\Delta t))[\widehat{\boldsymbol{\omega}}\times]^2 \tag{5.120}$$

Substituting (A.10) into (5.120), $\boldsymbol{\Theta}$ is obtained by

$$\boldsymbol{\Theta} = \cos(|\widehat{\boldsymbol{\omega}}|\Delta t)\,\mathbf{I}_{3\times3} - \sin(|\widehat{\boldsymbol{\omega}}|\Delta t)\left[\frac{\widehat{\boldsymbol{\omega}}}{|\widehat{\boldsymbol{\omega}}|}\times\right] + (1 - \cos(|\widehat{\boldsymbol{\omega}}|\Delta t))\frac{\widehat{\boldsymbol{\omega}}}{|\widehat{\boldsymbol{\omega}}|}\frac{\widehat{\boldsymbol{\omega}}}{|\widehat{\boldsymbol{\omega}}|}^{\mathrm{T}} \tag{5.121}$$

When $|\widehat{\boldsymbol{\omega}}|$ is very small, (5.121) will lead to numerical instability. To provide a stable version that is accurate for very small $|\widehat{\boldsymbol{\omega}}|$, the $\lim_{|\widehat{\boldsymbol{\omega}}|\to0}\boldsymbol{\Theta}$ of (5.121) is taken

$$\lim_{|\widehat{\boldsymbol{\omega}}|\to0}\boldsymbol{\Theta} = \mathbf{I}_{4\times4} - \Delta t[\widehat{\boldsymbol{\omega}}\times] + \frac{\Delta t^2}{2}[\widehat{\boldsymbol{\omega}}\times]^2 \tag{5.122}$$

Similarly, $\boldsymbol{\Psi}$ can be written from the factorization of $\widetilde{\boldsymbol{\Phi}}_k(5.116)(7.116)$ in the form

$$\boldsymbol{\Psi} = -\mathbf{I}_{3\times3}\Delta t - \frac{1}{2!}[\widehat{\boldsymbol{\omega}}\times]\Delta t^2 - \frac{1}{3!}[\widehat{\boldsymbol{\omega}}\times]^2\Delta t^3 + \cdots \tag{5.123}$$

where $\boldsymbol{\Psi}$ is obtained using the same methods used to find $\boldsymbol{\Theta}$. Specifically,

$$\boldsymbol{\Psi} = -\mathbf{I}_{3\times3}\Delta t + \left(\frac{1}{2!}\Delta t^2 + \frac{1}{4!}|\widehat{\boldsymbol{\omega}}|^2\Delta t^4 + \cdots\right)[\widehat{\boldsymbol{\omega}}\times] + \left(-\frac{1}{3!}\Delta t^3 + \frac{1}{5!}|\widehat{\boldsymbol{\omega}}|^2\Delta t^3 - \cdots\right)[\widehat{\boldsymbol{\omega}}\times]^2 \tag{5.124}$$

$$\begin{aligned}\boldsymbol{\Psi} = -\mathbf{I}_{3\times3}\Delta t &+ \frac{1}{|\widehat{\boldsymbol{\omega}}|^2}\left(1 - \left(1 - \frac{1}{2!}|\widehat{\boldsymbol{\omega}}|^2\Delta t^2 + \frac{1}{4!}|\widehat{\boldsymbol{\omega}}|^4\Delta t^4 + \cdots\right)\right)[\widehat{\boldsymbol{\omega}}\times] \\ &+ \left(-|\widehat{\boldsymbol{\omega}}|\Delta t + \left(|\widehat{\boldsymbol{\omega}}|\Delta t - \frac{1}{3!}\Delta t^3 + \frac{1}{5!}|\widehat{\boldsymbol{\omega}}|^2\Delta t^5 - \cdots\right)\right)[\widehat{\boldsymbol{\omega}}\times]^2\end{aligned} \tag{5.125}$$

$$\boldsymbol{\Psi} = -\mathbf{I}_{3\times3}\Delta t + \frac{1}{|\widehat{\boldsymbol{\omega}}|^2}(1 - \cos(|\widehat{\boldsymbol{\omega}}|\Delta t))[\widehat{\boldsymbol{\omega}}\times] + \frac{1}{|\widehat{\boldsymbol{\omega}}|^3}(|\widehat{\boldsymbol{\omega}}|\Delta t - \sin(|\widehat{\boldsymbol{\omega}}|\Delta t))[\widehat{\boldsymbol{\omega}}\times]^2 \tag{5.126}$$

When $|\widehat{\boldsymbol{\omega}}|$ is small, numerical instability can be circumvented by taking $\lim_{|\widehat{\boldsymbol{\omega}}|\to 0}\boldsymbol{\Psi}$ of (5.126)

$$\lim_{|\widehat{\boldsymbol{\omega}}|\to 0}\boldsymbol{\Psi} = -\mathbf{I}_{3\times 3}\Delta t + \lim_{|\widehat{\boldsymbol{\omega}}|\to 0}\frac{\sin(|\widehat{\boldsymbol{\omega}}|\Delta t)\,\Delta t}{2|\widehat{\boldsymbol{\omega}}|}[\widehat{\boldsymbol{\omega}}\times] - \lim_{|\widehat{\boldsymbol{\omega}}|\to 0}\frac{(1-\cos(|\widehat{\boldsymbol{\omega}}|))\,\Delta t}{3|\widehat{\boldsymbol{\omega}}|^2}[\widehat{\boldsymbol{\omega}}\times]^2 \tag{5.127}$$

$$= -\mathbf{I}_{3\times 3}\Delta t + \lim_{|\widehat{\boldsymbol{\omega}}|\to 0}\frac{\cos(|\widehat{\boldsymbol{\omega}}|\Delta t)\,\Delta t^2}{2}[\widehat{\boldsymbol{\omega}}\times] - \lim_{|\widehat{\boldsymbol{\omega}}|\to 0}\frac{\sin(|\widehat{\boldsymbol{\omega}}|)\,\Delta t^2}{6|\widehat{\boldsymbol{\omega}}|}[\widehat{\boldsymbol{\omega}}\times]^2 \tag{5.128}$$

$$= -\mathbf{I}_{3\times 3}\Delta t + \frac{\Delta t^2}{2}[\widehat{\boldsymbol{\omega}}\times] - \lim_{|\widehat{\boldsymbol{\omega}}|\to 0}\frac{\cos(|\widehat{\boldsymbol{\omega}}|)\,\Delta t^3}{6}[\widehat{\boldsymbol{\omega}}\times]^2 \tag{5.129}$$

$$\lim_{|\widehat{\boldsymbol{\omega}}|\to 0}\boldsymbol{\Psi} = -\mathbf{I}_{3\times 3}\Delta t + \frac{\Delta t^2}{2}[\widehat{\boldsymbol{\omega}}\times] - \frac{\Delta t^3}{6}[\widehat{\boldsymbol{\omega}}\times]^2 \tag{5.130}$$

5.5.7 – Process Noise Covariance

The continuous process noise covariance is defined by

$$\mathbf{Q}_c = \mathbf{E}\{\mathbf{n}(t)\,\mathbf{n}^{\mathrm{T}}(t-\tau)\} \tag{5.131}$$

Assuming the noise is white and the noise components are uncorrelated

$$\mathbf{Q}_c = \begin{bmatrix}\mathbf{Q}_{\boldsymbol{\omega}} & \mathbf{0}_{3\times 3}\\ \mathbf{0}_{3\times 3} & \mathbf{Q}_{\mathbf{b}}\end{bmatrix} \tag{5.132}$$

Taking (5.56) and setting $\boldsymbol{\Phi}_c \to \widetilde{\boldsymbol{\Phi}}_c$, the error-state discrete process noise is found by

$$\widetilde{\mathbf{Q}}_k = \int_{t_{k-1}}^{t_k}\widetilde{\boldsymbol{\Phi}}_c(t_k,\tau)\,\mathbf{W}_c(t_k,\tau)\mathbf{Q}_c\mathbf{W}_c(t_k,\tau)^{\mathrm{T}}\widetilde{\boldsymbol{\Phi}}_c^{\mathrm{T}}(t_k,\tau)\,d\tau \tag{5.133}$$

Substituting $\widetilde{\boldsymbol{\Phi}}_c$ for(5.116), $\mathbf{W}_c$ for (5.109), and $\mathbf{Q}_c$ for(5.132)

$$\widetilde{\mathbf{Q}}_k = \int_{t_{k-1}}^{t_k}\begin{bmatrix}\boldsymbol{\Theta} & \boldsymbol{\Psi}\\ \mathbf{0}_{3\times 3} & \mathbf{I}_{3\times 3}\end{bmatrix}\begin{bmatrix}-\mathbf{I}_{3\times 3} & \mathbf{0}_{3\times 3}\\ \mathbf{0}_{3\times 3} & \mathbf{I}_{3\times 3}\end{bmatrix}\begin{bmatrix}\mathbf{Q}_{\boldsymbol{\omega}} & \mathbf{0}_{3\times 3}\\ \mathbf{0}_{3\times 3} & \mathbf{Q}_{\mathbf{b}}\end{bmatrix}\begin{bmatrix}-\mathbf{I}_{3\times 3} & \mathbf{0}_{3\times 3}\\ \mathbf{0}_{3\times 3} & \mathbf{I}_{3\times 3}\end{bmatrix}\begin{bmatrix}\boldsymbol{\Theta}^{\mathrm{T}} & \mathbf{0}_{3\times 3}\\ \boldsymbol{\Psi}^{\mathrm{T}} & \mathbf{I}_{3\times 3}\end{bmatrix}d\tau \tag{5.134}$$

Substituting $\mathbf{Q}_{\mathbf{b}}$ for (4.67) and continuing to simplify gives

$$\widetilde{\mathbf{Q}}_k = \int_{t_{k-1}}^{t_k}\begin{bmatrix}-\boldsymbol{\Theta} & \boldsymbol{\Psi}\\ \mathbf{0}_{3\times 3} & \mathbf{I}_{3\times 3}\end{bmatrix}\begin{bmatrix}\mathbf{Q}_{\boldsymbol{\omega}} & \mathbf{0}_{3\times 3}\\ \mathbf{0}_{3\times 3} & \sigma_{\mathbf{b}}^2\mathbf{I}_{3\times 3}\end{bmatrix}\begin{bmatrix}-\boldsymbol{\Theta}^{\mathrm{T}} & \mathbf{0}_{3\times 3}\\ \boldsymbol{\Psi}^{\mathrm{T}} & \mathbf{I}_{3\times 3}\end{bmatrix}d\tau \tag{5.135}$$

$$\widetilde{\mathbf{Q}}_k = \int_{t_{k-1}}^{t_k} \begin{bmatrix} \mathbf{Q}_{\boldsymbol{\omega}} + \sigma_{\mathbf{b}}^2 \boldsymbol{\Psi}\boldsymbol{\Psi}^{\mathrm{T}} & \sigma_{\mathbf{b}}^2 \boldsymbol{\Psi} \\ \sigma_{\mathbf{b}}^2 \boldsymbol{\Psi}^{\mathrm{T}} & \sigma_{\mathbf{b}}^2 \mathbf{I}_{3\times 3} \end{bmatrix} d\tau \tag{5.136}$$

Through a considerable amount of algebra [125], the discrete process noise is obtained by

$$\widetilde{\mathbf{Q}}_k = \begin{bmatrix} \mathbf{Q}_{11} & \mathbf{Q}_{12} \\ \mathbf{Q}_{12}^{\mathrm{T}} & \mathbf{Q}_{22} \end{bmatrix} \tag{5.137}$$

where

$$\widetilde{\mathbf{Q}}_{11} = \mathbf{Q}_{\boldsymbol{\omega}}\Delta t + \sigma_{\mathbf{b}}^2 \left(\mathbf{I}_{3\times 3} \frac{\Delta t^3}{3} + \frac{\frac{(|\widehat{\boldsymbol{\omega}}|\Delta t)^3}{3} + 2\sin(|\widehat{\boldsymbol{\omega}}|\Delta t) - 2|\widehat{\boldsymbol{\omega}}|\Delta t}{|\widehat{\boldsymbol{\omega}}|^5} [\widehat{\boldsymbol{\omega}}\times]^2 \right) \tag{5.138}$$

$$\widetilde{\mathbf{Q}}_{12} = -\sigma_{\mathbf{b}}^2 \left(\mathbf{I}_{3\times 3} \frac{\Delta t^2}{2} - \frac{|\widehat{\boldsymbol{\omega}}|\Delta t - \sin(|\widehat{\boldsymbol{\omega}}|\Delta t)}{|\widehat{\boldsymbol{\omega}}|^3} [\widehat{\boldsymbol{\omega}}\times] + \frac{\frac{(|\widehat{\boldsymbol{\omega}}|\Delta t)^2}{2} + \cos(|\widehat{\boldsymbol{\omega}}|\Delta t) - 1}{|\widehat{\boldsymbol{\omega}}|^4} [\widehat{\boldsymbol{\omega}}\times]^2 \right) \tag{5.139}$$

$$\widetilde{\mathbf{Q}}_{22} = \sigma_{\mathbf{b}}^2 \Delta t\, \mathbf{I}_{3\times 3} \tag{5.140}$$

Once again, to prevent numerical instability when $|\widehat{\boldsymbol{\omega}}|$ is small, by taking the $\lim_{|\widehat{\boldsymbol{\omega}}|\to 0}$ of (5.138) and (5.139), the submatrices of $\widetilde{\mathbf{Q}}_{11}$ and $\widetilde{\mathbf{Q}}_{12}$ for small $|\widehat{\boldsymbol{\omega}}|$ is obtained by

$$\lim_{|\widehat{\boldsymbol{\omega}}|\to 0} \widetilde{\mathbf{Q}}_{11} = \mathbf{Q}_{\boldsymbol{\omega}}\Delta t + \sigma_{\mathbf{b}}^2 \left(\mathbf{I}_{3\times 3} \frac{\Delta t^3}{3} + \frac{2\Delta t^5}{120} [\widehat{\boldsymbol{\omega}}\times]^2 \right) \tag{5.141}$$

$$\lim_{|\widehat{\boldsymbol{\omega}}|\to 0} \widetilde{\mathbf{Q}}_{12} = -\sigma_{\mathbf{b}}^2 \left(\mathbf{I}_{3\times 3} \frac{\Delta t^2}{2} - \frac{\Delta t^3}{6} [\widehat{\boldsymbol{\omega}}\times] + \frac{\Delta t^4}{24} [\widehat{\boldsymbol{\omega}}\times]^2 \right) \tag{5.142}$$

Note that the limit of (5.140) is not required as it is already stable for small $|\widehat{\boldsymbol{\omega}}|$.

5.5.8 – Measurement Update Equations

In this section, update equations are developed, which use the attitude obtained from the use of homography and the IR camera (see Ch. 5) to correct the projected estimates of the state, error state, and error-state covariance. Specifically, the error-state camera model is developed with the purpose of deriving the error-state measurement matrix $\widetilde{\mathbf{H}}_k$. Technically, these equations are part of the system implementation since they were developed specifically for this application. However, they are included in this theory section since they are necessary for Sect. 5.5.10 to be complete.

The measured state vector $\mathbf{z}_k$ is a function of the measured rotation vector $\boldsymbol{\theta}_k^c$, which is obtained from homography on the camera images. For simplicity, the measurement noise $\mathbf{v}_k$ is assumed to be additive to the state attitude $\bar{q}(\boldsymbol{\theta}_k)$ as demonstrated by

$$\mathbf{z}(\boldsymbol{\theta}_k^c) = [\bar{q}(\boldsymbol{\theta}_k)] + \mathbf{v}_k \tag{5.143}$$

Including the gyro bias $\mathbf{b}_k$ in (5.143), $\mathbf{z}(\boldsymbol{\theta}_k^c)$ can be verified that it is compatible with (5.12)

$$\mathbf{z}(\boldsymbol{\theta}_k^c) = [\mathbf{I}_{4\times4} \quad \mathbf{0}_{4\times3}]\begin{bmatrix}\bar{q}(\boldsymbol{\theta}_k)\\ \mathbf{b}_k\end{bmatrix} + \mathbf{v}_k \tag{5.144}$$

Taking the expectation of (5.144) and simplifying, the estimated projected measurement is obtained by

$$\hat{\mathbf{z}}_k = \mathbf{E}\{\mathbf{z}(\boldsymbol{\theta}_k^c)\} = [\mathbf{I}_{4\times4} \quad \mathbf{0}_{4\times3}]\begin{bmatrix}\hat{\bar{q}}(\boldsymbol{\theta}_k)\\ \hat{\mathbf{b}}_k\end{bmatrix} \tag{5.145}$$

which reduces to

$$\hat{\mathbf{z}}_k = \hat{\bar{q}}_k \tag{5.146}$$

The primary purpose for the development of these equations is to derive the error-state measurement matrix $\widetilde{\mathbf{H}}_k$ for use in the Riccati equation, which is obtained through the derivation of the error-state measurement equations. Formally, the error-state measurement is defined by

$$\tilde{\mathbf{z}}_k \triangleq \mathbf{z}_k - \hat{\mathbf{z}}_k \tag{5.147}$$

Substituting (5.143) and (5.146) into (5.147) gives

$$= \bar{q}_k - \hat{\bar{q}}_k + \mathbf{v}_k \tag{5.148}$$

Using (5.82) and rearranging

$$= \delta\bar{q}_k \otimes \hat{\bar{q}}_k - \hat{\bar{q}}_k + \mathbf{v}_k \tag{5.149}$$

$$= (\delta\bar{q} - \bar{q}_{\mathrm{I}}) \otimes \hat{\bar{q}} + \mathbf{v}_k \tag{5.150}$$

Using (4.19)

$$= \boldsymbol{\mathcal{R}}(\hat{\bar{q}})(\delta\bar{q} - \bar{q}_{\mathrm{I}}) + \mathbf{v}_k \tag{5.151}$$

Expanding and using the small angle approximation for a quaternion (5.85)

$$\approx \boldsymbol{\mathcal{R}}(\hat{\bar{q}})\left(\begin{bmatrix}\frac{1}{2}\delta\boldsymbol{\theta}\\ 1\end{bmatrix} - \begin{bmatrix}\mathbf{0}_{3\mathrm{x}1}\\ 1\end{bmatrix}\right) + \mathbf{v}_k \tag{5.152}$$

$$= \boldsymbol{\mathcal{R}}(\hat{\bar{q}})\left(\begin{bmatrix}\frac{1}{2}\delta\boldsymbol{\theta}\\ 0\end{bmatrix}\right) + \mathbf{v}_k \tag{5.153}$$

Using the $\boldsymbol{\mathcal{R}}(\cdot)$ property (A.4)

$$= [\boldsymbol{\Xi}(\hat{\bar{q}}) \quad \hat{\bar{q}}]\begin{bmatrix}\frac{1}{2}\delta\boldsymbol{\theta}\\ 0\end{bmatrix} + \mathbf{v}_k \tag{5.154}$$

$$= \left[\boldsymbol{\Xi}\left(\tfrac{1}{2}\hat{\bar{q}}\right) \quad \mathbf{0}_{4\text{x}1}\right]\begin{bmatrix}\delta\boldsymbol{\theta}\\ 0\end{bmatrix} + \mathbf{v}_k \tag{5.155}$$

Analogous to (5.144), including the gyro error bias $\Delta\mathbf{b}$ into(5.155) gives

$$\tilde{\mathbf{z}}_k = \left[\boldsymbol{\Xi}\left(\tfrac{1}{2}\hat{\bar{q}}\right) \quad \mathbf{0}_{4\text{x}3}\right]\begin{bmatrix}\delta\boldsymbol{\theta}\\ \Delta\mathbf{b}\end{bmatrix} + \mathbf{v}_k \tag{5.156}$$

Organising (5.156) into the form

$$\tilde{\mathbf{z}}_k = \widetilde{\mathbf{H}}_k\tilde{\mathbf{x}}_k + \mathbf{v}_k \tag{5.157}$$

the error-state measurement matrix is obtained by

$$\widetilde{\mathbf{H}}_k = \left[\boldsymbol{\Xi}\left(\tfrac{1}{2}\hat{\bar{q}}\right) \quad \mathbf{0}_{4\text{x}3}\right] \tag{5.158}$$

5.5.9 – State and Covariance Prediction

Nearly all the equations have been defined up to this point, and the propagation of the state and error state can be computed. The equations in this section rely on equations from Sect. 5.5.10, and values are obtained from the previous iteration whenever a symbol has a subscript marking $\cdot_{k-1}$. For the first iteration, the initial values are determined as described in Sect. 5.3.

Prediction of the a priori state and covariance are obtained in seven steps, noting that $\boldsymbol{g}_k$ is obtained from the readout of the gyros.

First, estimate the a priori bias, which is obtained by the discretized solution to (5.73) leading to

$$\hat{\mathbf{b}}_k^- = \hat{\mathbf{b}}_{k-1}^+ \tag{5.159}$$

Second, discretizing (5.74) the true angular rate is estimated with

$$\widehat{\boldsymbol{\omega}}_k^- = \boldsymbol{g}_k - \widehat{\mathbf{b}}_k^- \tag{5.160}$$

Third, Using (5.76)-(5.80) and setting $\boldsymbol{\omega}_k \to \widehat{\boldsymbol{\omega}}_k^-$ and $\boldsymbol{\omega}_{k-1} \to \widehat{\boldsymbol{\omega}}_{k-1}^+$, calculate the first-order quaternion integrator $\boldsymbol{\Phi}_k^{\hat{\bar{q}},1}$ and relating equations by

$$\boldsymbol{\Phi}_k^{\hat{\bar{q}},1} \approx \boldsymbol{\Phi}_k^{\hat{\bar{q}},0}\left(\bar{\boldsymbol{\omega}}_k^{\wedge}\right) + \frac{1}{48}\left(\boldsymbol{\Omega}\left(\dot{\widehat{\boldsymbol{\omega}}}_{k-1}^+\right)\boldsymbol{\Omega}\left(\widehat{\boldsymbol{\omega}}_{k-1}^+\right) - \boldsymbol{\Omega}\left(\widehat{\boldsymbol{\omega}}_{k-1}^+\right)\boldsymbol{\Omega}\left(\dot{\widehat{\boldsymbol{\omega}}}_{k-1}^+\right)\right)\Delta t^3 \tag{5.161}$$

where

$$\bar{\boldsymbol{\omega}}_k^{\wedge} = \frac{\widehat{\boldsymbol{\omega}}_k^- + \widehat{\boldsymbol{\omega}}_{k-1}^+}{2} \tag{5.162}$$

$$\dot{\widehat{\boldsymbol{\omega}}}_{k-1}^+ \approx \frac{\widehat{\boldsymbol{\omega}}_k^- - \widehat{\boldsymbol{\omega}}_{k-1}^+}{\Delta t} \tag{5.163}$$

$$\boldsymbol{\Phi}_k^{\hat{\bar{q}},0}\left(\bar{\boldsymbol{\omega}}_k^{\wedge}\right) = \left[\cos\left(\frac{\left|\bar{\boldsymbol{\omega}}_k^{\wedge}\right|}{2}\Delta t\right)\cdot \mathbf{I}_{4\times 4} + \frac{1}{\left|\bar{\boldsymbol{\omega}}_k^{\wedge}\right|}\sin\left(\frac{\left|\bar{\boldsymbol{\omega}}_k^{\wedge}\right|}{2}\Delta t\right)\cdot \boldsymbol{\Omega}\left(\bar{\boldsymbol{\omega}}_k^{\wedge}\right)\right] \tag{5.164}$$

When $\left|\bar{\boldsymbol{\omega}}_k^{\wedge}\right|$is small, (5.165) is used in place of (5.164)

$$\lim_{|\boldsymbol{\omega}_k|\to 0} \boldsymbol{\Phi}_k^{\hat{\bar{q}},0}\left(\bar{\boldsymbol{\omega}}_k^{\wedge}\right) = \mathbf{I}_{4\times 4} + \frac{1}{2}\boldsymbol{\Omega}\left(\bar{\boldsymbol{\omega}}_k^{\wedge}\right)\Delta t \tag{5.165}$$

Fourth, dropping the frame scripts on (5.75) and setting $\boldsymbol{\Phi}_k^{\bar{q}} \to \boldsymbol{\Phi}_k^{\hat{\bar{q}},1}$, the a priori estimate of the attitude is obtained from

$$\hat{\bar{q}}_k^- = \boldsymbol{\Phi}_k^{\hat{\bar{q}},1}\hat{\bar{q}}_{k-1}^+ \tag{5.166}$$

Fifth, using the (5.116), the error-state transition matrix is obtained by

$$\widetilde{\boldsymbol{\Phi}}_k^{\wedge} = \begin{bmatrix} \boldsymbol{\Theta}_k^{\wedge} & \boldsymbol{\Psi}_k^{\wedge} \\ \mathbf{0}_{3\times 3} & \mathbf{I}_{3\times 3} \end{bmatrix} \tag{5.167}$$

where

$$\boldsymbol{\Theta}_k^{\wedge} = \cos(|\widehat{\boldsymbol{\omega}}_k^-|\Delta t)\,\mathbf{I}_{3\times 3} - \sin(|\widehat{\boldsymbol{\omega}}_k^-|\Delta t)\left[\frac{\widehat{\boldsymbol{\omega}}_k^-}{|\widehat{\boldsymbol{\omega}}_k^-|}\times\right] + (1-\cos(|\widehat{\boldsymbol{\omega}}_k^-|\Delta t))\frac{\widehat{\boldsymbol{\omega}}_k^-}{|\widehat{\boldsymbol{\omega}}_k^-|}\frac{\widehat{\boldsymbol{\omega}}_k^-}{|\widehat{\boldsymbol{\omega}}_k^-|}^{\mathrm{T}} \tag{5.168}$$

$$\boldsymbol{\Psi}_k^{\wedge} = -\mathbf{I}_{3\times 3}\Delta t + \frac{1}{|\widehat{\boldsymbol{\omega}}_k^-|^2}(1-\cos(|\widehat{\boldsymbol{\omega}}_k^-|\Delta t))[\widehat{\boldsymbol{\omega}}_k^-\times] + \frac{1}{|\widehat{\boldsymbol{\omega}}_k^-|^3}(|\widehat{\boldsymbol{\omega}}_k^-|\Delta t - \sin(|\widehat{\boldsymbol{\omega}}_k^-|\Delta t))[\widehat{\boldsymbol{\omega}}_k^-]^2 \tag{5.169}$$

For small values of $|\widehat{\boldsymbol{\omega}}_k^-|$, $\boldsymbol{\Theta}_k^{\wedge}$ is obtained from (5.170) instead of (5.168), and $\boldsymbol{\Psi}_k^{\wedge}$ is obtained from (5.171) instead of (5.169).

$$\lim_{|\widehat{\boldsymbol{\omega}}|\to 0}\boldsymbol{\Theta}_k^{\wedge} = \mathbf{I}_{4\times 4} - \Delta t[\widehat{\boldsymbol{\omega}}_k^-\times] + \frac{\Delta t^2}{2}[\widehat{\boldsymbol{\omega}}_k^-\times]^2 \tag{5.170}$$

$$\lim_{|\widehat{\boldsymbol{\omega}}|\to 0}\boldsymbol{\Psi}_k^{\wedge} = -\mathbf{I}_{3\times 3}\Delta t + \frac{\Delta t^2}{2}[\widehat{\boldsymbol{\omega}}_k^-\times] - \frac{\Delta t^3}{6}[\widehat{\boldsymbol{\omega}}_k^-\times]^2 \tag{5.171}$$

Sixth, using (5.137)-(5.142) the error-state discrete noise covariance $\widetilde{\mathbf{Q}}_k^{\wedge}$ is obtain by replacing $\widehat{\boldsymbol{\omega}} \to \widehat{\boldsymbol{\omega}}_k^-$, as was done in the preceding equations.

Seventh, using (5.17) and applying error-state markings (i.e. setting $\boldsymbol{\Phi}_k \to \widetilde{\boldsymbol{\Phi}}_k^{\wedge}$, $\mathbf{P}_k^- \to \widetilde{\mathbf{P}}_k^-$, $\mathbf{P}_{k-1}^+ \to \widetilde{\mathbf{P}}_{k-1}^+$, and $\mathbf{Q}_k \to \mathbf{Q}_k^{\wedge}$), the a priori error-state covariance is obtained by

$$\widetilde{\mathbf{P}}_k^- = \widetilde{\boldsymbol{\Phi}}_k^{\wedge}\widetilde{\mathbf{P}}_{k-1}^+\widetilde{\boldsymbol{\Phi}}_k^{\wedge^{\mathrm{T}}} + \mathbf{Q}_k^{\wedge} \tag{5.172}$$

This completes the prediction phase of the attitude filter, which predicts the state at the next time step. The next step is to correct the predicted state and covariance estimates.

5.5.10 – State and Covariance Correction

Following the propagation of the state and error-state covariance performed by the methods of Sect. 5.5.9, the state and error-state covariance can be corrected with the use of an attitude measurement $\mathbf{z}_k$ in ten steps.

First, the calculate the residual, which is expressed as

$$\mathbf{r}_k = \mathbf{z}_k - \hat{\mathbf{z}}_k \tag{5.173}$$

Setting $\hat{\bar{q}}_k \to \hat{\bar{q}}_k^-$ and combining (5.173) and (5.146), the residual is obtained by

$$\mathbf{r}_k = \mathbf{z}_k - \hat{\bar{q}}_k^- \tag{5.174}$$

Second, calculate the error-state measurement matrix

$$\widetilde{\mathbf{H}}_k = \left[\boldsymbol{\Xi}\left(\tfrac{1}{2}\hat{\bar{q}}_k^-\right) \quad \mathbf{0}_{4\text{x}3}\right] \tag{5.175}$$

Third, calculate the covariance of the error-state residual (c.f. (5.23))

$$\widetilde{\mathbf{S}}_k = \widetilde{\mathbf{H}}_k \widetilde{\mathbf{P}}_k^- \widetilde{\mathbf{H}}_k^{\mathrm{T}} + \mathbf{R}_k \tag{5.176}$$

where the measurement noise covariance $\mathbf{R}_k$ is obtained by (5.14).

Fourth, calculate the error-state Kalman gain by (c.f. (5.22))

$$\widetilde{\mathbf{K}}_k = \widetilde{\mathbf{P}}_k^- \widetilde{\mathbf{H}}_k^{\mathrm{T}} \widetilde{\mathbf{S}}_k^{-1} \tag{5.177}$$

Fifth, calculate the state correction [146]

$$\Delta\hat{\mathbf{x}}_k^+ = \widetilde{\mathbf{K}}_k \mathbf{r}_k \tag{5.178}$$

Sixth, calculate the correction quaternion $\widehat{\delta\bar{q}}_k^+$**.** Noting that

$$\Delta\hat{\mathbf{x}}_k^+ = \begin{bmatrix} \widehat{\delta\boldsymbol{\theta}}_k^+ \\ \widehat{\Delta\mathbf{b}}_k^+ \end{bmatrix} = \begin{bmatrix} 2 \cdot \widehat{\delta\mathbf{q}}_k^+ \\ \widehat{\Delta\mathbf{b}}_k^+ \end{bmatrix} \tag{5.179}$$

the correction quaternion $\widehat{\delta\bar{q}}_k^+$ is obtained by

$$\widehat{\delta\bar{q}}_k^+ = \begin{bmatrix} \widehat{\delta\mathbf{q}}_k^+ \\ \sqrt{1 - \left|\widehat{\delta\mathbf{q}}_k^+\right|^2} \end{bmatrix} \tag{5.180}$$

except if $\left|\widehat{\delta\mathbf{q}}_k^+\right| > 1$ due to numerical errors, use the normalized correction quaternion

$$\widehat{\delta\bar{q}}_k^+ = \begin{bmatrix} \frac{1}{|\widehat{\delta\mathbf{q}}_k^+|} \cdot \widehat{\delta\mathbf{q}}_k^+ \\ 0 \end{bmatrix} \tag{5.181}$$

Seventh, calculate the a posteriori attitude quaternion

$$\hat{\bar{q}}_k^+ = \widehat{\delta\bar{q}}_k^+ \otimes \hat{\bar{q}}_k^- \tag{5.182}$$

Eight, calculate the a posteriori bias

$$\widehat{\mathbf{b}}_k^+ = \widehat{\mathbf{b}}_k^- + \widehat{\Delta\mathbf{b}}_k^+ \tag{5.183}$$

Ninth**,** calculate an a posteriori estimate of the angular rate

$$\widehat{\boldsymbol{\omega}}_k^+ = \boldsymbol{g}_k - \widehat{\mathbf{b}}_k^+ \tag{5.184}$$

Tenth, calculate the a posteriori error-state covariance (cf. (5.24))

$$\widetilde{\mathbf{P}}_k^+ = \left[\mathbf{I}_{6\times 6} - \widetilde{\mathbf{K}}_k\widetilde{\mathbf{H}}_k\right]\widetilde{\mathbf{P}}_k^-\left[\mathbf{I}_{6\times 6} - \widetilde{\mathbf{K}}_k\widetilde{\mathbf{H}}_k\right]^{\mathrm{T}} + \widetilde{\mathbf{K}}_k\mathbf{R}_k\widetilde{\mathbf{K}}_k^{\mathrm{T}} \tag{5.185}$$

Following this last step, set $k \to k+1$ and repeat the prediction step in Sect. 5.5.9. The correction phase is used in the attitude filter to correct the estimate made by the previous prediction phase.

Using this subsection and Sect. 5.5.9, all the theory required to complete an attitude filter has been presented. Furthermore, the theory in this chapter has provided the foundation to implement the pose filter (q.v. Sect. 6.8).

Chapter 6 – Design and Implementation of Low-Cost 6-DOF Teleoperation

This chapter details the implementation of a low-cost and stable 6-DOF teleoperation device used in the teaching portion of programming by demonstration (PbD). Furthermore, this chapter is limited to aspects of implementing the system, whereas, Chapter 7 details implementing the *calibration* of the system, which is used to obtain specific parameters required by the system.

The following is a brief overview of the chapter. Section 6.1 provides an overview of the complete system, Section 6.2 details the communication between modules, and Section 6.3 details the chosen reference frames for the system, which are important to rigorously specify since the computations of the system are with respect to (wrt) these reference frames. Sections 6.4 through 6.11 detail implementation of specific interfaces, modules, and subsystems.

6.1 – Overview of the System

This section provides an overview of the 6-DOF teleoperation system used to control the A465 Robot Manipulator and the Virtual Robotic Manipulator. At the highest systems-level, the functioning of the system can be described as follows. The human operator moves the wiimote in space. The wiimote to Robot Interface detects the wiimote's sensor values and sends the information to the Robot Control Server, where the sensor values are interpreted and calculations are performed to estimate the pose of the wiimote. Computed pose is sent to a robot controller, where the information is used to operate a robot manipulator. Through these steps, the teaching portion of programming-by-demonstration (i.e. teleoperation) is performed.

At one level deeper, the system can be modularized as shown in Figure 6.1. The human operator physically moves the wiimote in the desired path and the wiimote's IR camera captures the locations of the beacon's four IR LEDs, and the accelerometer and gyroscope data are sampled. The sensor data are sent in an output stream via Bluetooth, the Bluetooth stack, and the human interface device (HID) driver to the WiiYourself! Library [149], which is embedded in the Wiimote to Robot Interface. The Teleoperation module uses WiiYourself! to interpret the input stream and extract the sensor values.

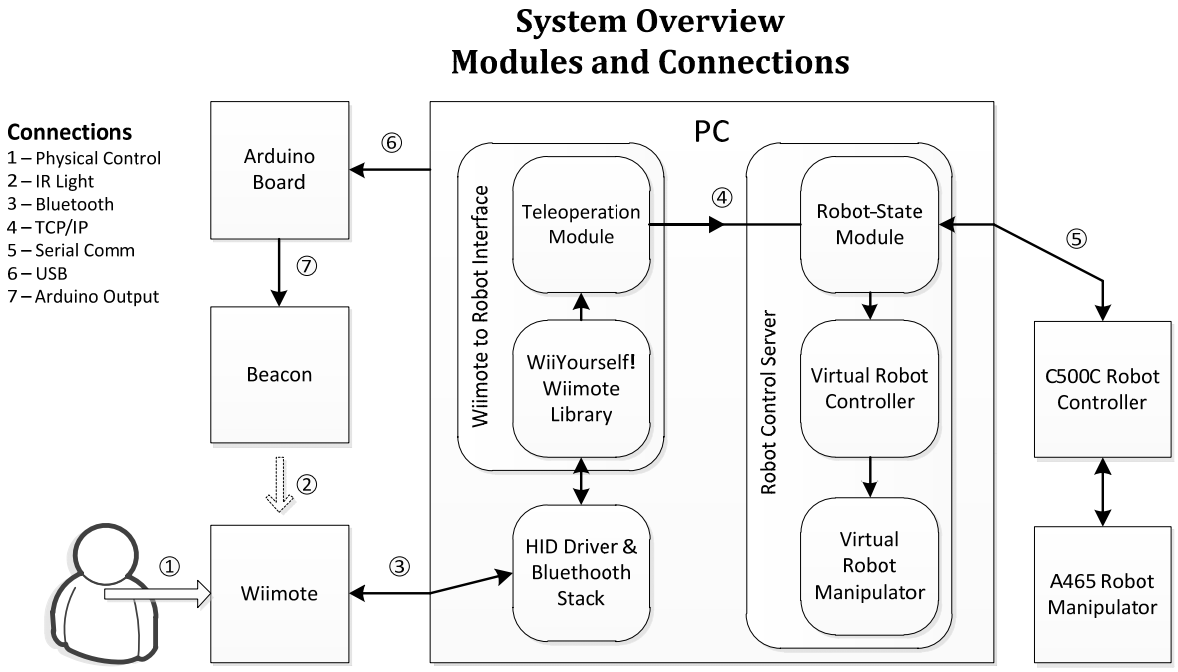


Figure 6.1. Overview of the system including all modules and their respective connections. Bevelled squares depict software, unbevelled squares depict hardware, and arrowheads depict information flow.

The interpreted sensor values are sent via a TCP/IP socket to the Robot Control Server, where the Robot-State Module uses a pose filter to near-optimally estimate the pose of the wiimote. Under normal circumstances, the Robot-State Module sends pose information to the C500C Robot Controller, which calculates the inverse kinematics and sends the appropriate commands to the A465 Robot Manipulator. However, during testing or if trajectories are to be performed while the physical robot is not present, the control stream is redirected to the virtual robotic system, which is a custom built software virtualization of the robotic system. The Virtual Robot Controller replaces the C500C Robot Controller and the Virtual Robot Manipulator is a 3D OpenGL-based simulator replacing the A465 Robot Manipulator. Analogously, the Virtual Robot Controller accepts pose information from the Robot-State Module, calculates the inverse kinematics, and sends the information to the Virtual Robot Manipulator. The Virtual Robot Manipulator uses this information to compute the location of the joints, links, and surface geometry of the virtual robot. OpenGL is use to draw these surfaces on the screen; however, OpenGL is not used to calculate the forward kinematics and surface geometry, which is computed manually. This is advantageous since it lets surface information be known explicitly and thus allows the

possibility of using this spatial data for a future module to handle safety and collision prevention with the external world.

During basic teleoperation, it is not necessary for the beacon to be connected to the PC; however, during IR camera calibration or when using the advanced beacon setup (Sect. 7.4), it is necessary to control the beacon LED in specific patterns and at specific times. An Arduino microcontroller board is used as a PC interfaceable microcontroller to precisely control LED output.

Runtime speed is an important aspect; therefore, the system was created almost entirely in C++ and takes advantage of highly optimized math libraries written in FORTRAN. In total, over 23,000 lines of code were programmed, and it is capable of running in real-time on a low-end modern PC. The remainder of the chapter is dedicated to detailing the system, developed as a major component of this thesis research.

6.2 – Communication and Interactions between Hardware and Software

Section 6.2.1 provides an overview of the types of connections and interactions between and within software and hardware. The connections and interactions are detailed in Sections 6.2.2 to 6.2.5 and Connections 6 and 7 (Figure 6.1) are detailed in Chapter 7.

6.2.1 – Connection Overview

The teleoperation system has hardware and software that require robust communication. Figure 6.1 lists the seven communication connections that are directly controlled by software that was created in this research.

Connection 1 is between the operator and the wiimote and is a direct mechanical control. The operator physically moves the wiimote to the desired path and presses relevant buttons. Connection 2 is an IR connection between the beacon and the wiimote. The four LEDs emit IR light, which is captured by the wiimote's IR camera and used in further processing. Connection 1 and 2 are detailed in Sect. 6.2.2.

Connection 3 is a full-duplex wireless Bluetooth connection between the wiimote and the PC. This data stream is further processed by a Bluetooth stack and a HID driver. Initialization and operation

of the connection is detailed in Sect. 6.2.3. Connection 4 is a simplex TCP/IP socket connection between the Teleoperation Module and the Robot-State Module. Wiimote sensor data are packed into a stream and sent over TCP/IP, where it is received and unpacked by the Robot Control Server to be further processed by the Robot-State Module. Use of the TCP/IP protocol enables teleoperation of the robot with the wiimote at a remote site. TCP/IP transmission details are presented in Sect. 6.2.4. Connection 5 is a full duplex serial connection between the Robot-State Module and the C500C Robot Controller. Communication is handled by the ActiveRobot Library. Further details of this connection are discussed in Sect. 6.2.5.

The last two connections are only required for advanced beacon use and the wiimote calibration board. If only the simple beacon is used, it may be powered by an external 5 V power supply instead. Connection 6 is full duplex USB communication from the PC to the Arduino microcontroller. It is used for uploading code to the controller and for optional LED control by the PC for an advanced beacon use. Further details are discussed in Chapter 7. Connection 7 is a removable hardwired connection from the output pins of the Arduino board to either the beacon or the wiimote-calibration board (discussed in 7.4).

6.2.2 – Human-Wiimote and Beacon-Wiimote Interaction

The system uses the wiimote's sensors to capture 6-DOF natural movements. The human operator interacts with the device by physically manipulating it and performing the required trajectory. Furthermore, wiimote buttons can be easily assigned to do tasks such as open or close grippers.

To have stable 6-DOF motion, the wiimote's IR camera must be pointed in the vicinity of the IR beacon to allow the position of the four beacon IR LEDs to be captured. Inability of the IR camera to capture the LEDs results in high instability of estimated positions. To prevent exponential position error, the system temporarily drops into 3-DOF attitude-only mode, where the position is locked, and only the attitude is tracked. Under general use, the gyroscopes are sufficiently accurate that attitude can remain stable until the LED's are re-identified by the IR camera, at which point the system can resume absolute 6-DOF tracking. The resulting position error will be removed by a smooth correction of position from the locked position to the near-optimally estimated one.

6.2.3 – Wiimote to PC Communication

Before software on the PC can communicate with the wiimote, the device initially requires Bluetooth pairing with the PC. Windows' Bluetooth stack does not natively support the wiimote's Bluetooth connection, thus a proprietary Bluetooth stack is required. For the current implementation, BleuSoliel software was used.

To pair the wiimote with the OS, the wiimote buttons 1 and 2 are pressed simultaneously to activate the wiimote's Bluetooth-device search in order to locate the PC's Bluetooth adapter. During the search process, four blue LEDs on the wiimote will blink. While the LEDs are blinking, BleuSoliel is set to scan for devices to pair with, and once the wiimote is found, it will automatically be paired and a human interface device (HID) driver will be installed to allow communication. If the wiimote device is not found, one of the following problems may have occurred: (1) The wiimote's LEDs were no longer blinking while BleuSoliel was searching (dead battery or search time-out), (2) the PC's Bluetooth adapter is not compatible with the wiimote, (3) the wiimote is out of range, or (4) BleuSoliel or the Bluetooth driver was not installed correctly. The wiimote only requires pairing once. At any future time, a communication connection can be initiated by pressing wiimote buttons 1 and 2 and connecting to the previously paired device in BleuSoliel.

During operation, the wiimote is continuously capturing data from its sensors wrt respective reference frames. The 2D pixel coordinates of the four beacon LEDs (wrt IR-Sensor Frame), the acceleration values (wrt Accelerometer Frame), the angular rate values (wrt Gyroscope Frame), the state of the buttons, and other internal wiimote states are sent to the PC wirelessly via Bluetooth at a rate of up to 100 Hz. Simultaneously, any control commands from the PC to the wiimote are receive by the wiimote and executed by the wiimote hardware (e.g. activate rumble pack, send frequency to wiimote speaker, or change status of the wiimote's onboard blue LEDs).

Following a teleoperation session, it is necessary to do a software disconnect of the wiimote device to terminate the hardware Bluetooth connection. Otherwise, the wiimote will maintain a power-on state until the wiimote's batteries have been depleted.

6.2.4 – Inter-program TCP/IP Socket Communication

The Wii to Robot Interface is responsible for sending wiimote data to the Robot Control Server over a TCP/IP connection. The TCP/IP connection allows for teleoperation of the A465 Robot from any remote site globally that has access to an internet connection. To initiate a TCP/IP connection, the Robot Control Server must first be executed on the local computer attached to the C500C Robot Controller. Second, the IP address of the local computer is entered and the Wiimote to Robot Interface is executed on either a local or a remote computer. A TCP handshake is then initiated and socket communication will be established on port 30005.

When the Wiimote to Robot Interface obtains data from the wiimote, it sends it over the TCP/IP connection. All wiimote data are packed into an endian-independent byte stream and wrapped in several headers. The first header is a simple custom "teleoperation" header used to ensure proper separation of the incoming byte stream on the server. Its format is as follows:

```
| 0xeb90(2bytes) | length(2bytes) | data(length-2bytes) | checksum(2bytes) |
```

The first two bytes (hex) is the teleoperation header identifier, the next two bytes (short int) is the remaining length of the header, the next <*length* – *two*> bytes (unsigned byte stream) is the contained data, and the last two bytes (short int) is the data checksum.

The teleoperation packet is then wrapped in a TCP header to reach the appropriate port and an IP header to reach the appropriate IP address. If travelling across a network, the OS also wraps the IP header in a MAC header. Following transmission, the data are unwrapped until it researches the Robot Control Server, where the teleoperation header is inspected and the byte stream data are extracted or is dropped if the header is malformed. The wiimote sensor data are unpacked from the byte stream data and stored in a queue until the Robot Control Server is able to process it with the Robot-State Module.

6.2.5 – Robot Control Server to C500C Robot Controller Communication

The hardware communication between the Robot Control Server and the C500C Robot Controller is performed through a serial connection. At the software level, it is handled by the ActiveRobot library supplied by CRS. ActiveRobot allows any Microsoft Windows ActiveX application to send command to and receive feedback from the C500C Robot Controller. This allows control of the A465 Robot Manipulator by the Robot Control Server via the C500C Robot Controller.

6.3 – Reference Frames

Sect. 6.3.1 provides an overview of the reference frames used by the system, Sect. 6.3.2 through Sect. 6.3.4 detail the reference frames and describe their uses, and Sect. 6.3.5 discusses the use of reference frames in performing teleoperation.

6.3.1 – Overview

Reference frames play an important role in this research, since all motion is with respect to (wrt) a reference frame. The robot, beacon, and sensors each have multiple reference frames that serve as a basis for measurement vectors and computations. Roughly thirty reference frames are used, although the most prominent sixteen are described in this section. The developed software computes numerous transformations between reference frames and it is therefore important to rigorously define the most important reference frames, which greatly improves the clarity of the algorithms described.

6.3.2 – Sensor Reference Frames

Each sensor measurement is with respect to a different reference frame. Furthermore, the robot and the beacon have their own reference frames. Before sensors can be integrated, fused with Kalman filtering, and then used to teleoperate the robot, they must be transformed into the same frame.

The wiimote has multiple reference frames (Figure 6.2). Wiimote angular rates are detected wrt the *Gyroscope Frame* (Figure 6.2.a), which has axes defined orthogonal to the plane of rotation with the direction defined by the positive angular rate and the right-hand rule. Accelerations are detected in the *Accelerometer Frame* (Figure 6.2.b), where the origin is within the accelerometer, and the positive axes are aligned (equal direction and collinear) with the directions sampled by the accelerometer.

The position obtained from solving the planar homography (q.v. Sect. 6.7) is wrt the *(IR) Camera Frame* (Figure 6.2.c), which has its *x*- and *y*-axes aligned with the IR sensor and its origin located at the bottom right corner of sensor (wrt a user that is pointing the wiimote away). The IR camera's sensor detects 2D dot position wrt the Camera Frame's *x*-*y* plane. Since the Camera Frame's *x*-*y* plane will be often used as a reference for 2D dots on the IR camera's sensor, the Camera Frames's *x*-*y* plane will be explicitly referred to as the *IR-Sensor Frame*.

Before the sensors can be integrated and fused, their reference frames must be converted to a compatible frame. Specifically, the wiimote frames in (a), (b), and (c) in Figure 6.2, are converted to the *User Frame* (Figure 6.2.d), which was chosen since it shares alignment with the A465 Robot Manipulator. Also shown in Figure 6.2.d is the *World Frame*, which is set during pose filter calibration (q.v. Sect. 6.6.2). The World Frame's origin is set to the User Frame's origin, has its *z*-axis aligned with gravity, and its remaining axis constrained by matching its *yaw* with the initial User Frame determined during pose filter calibration. The pose of the User Frame wrt the World Frame is the relative pose sent to the robot during teleoperation.

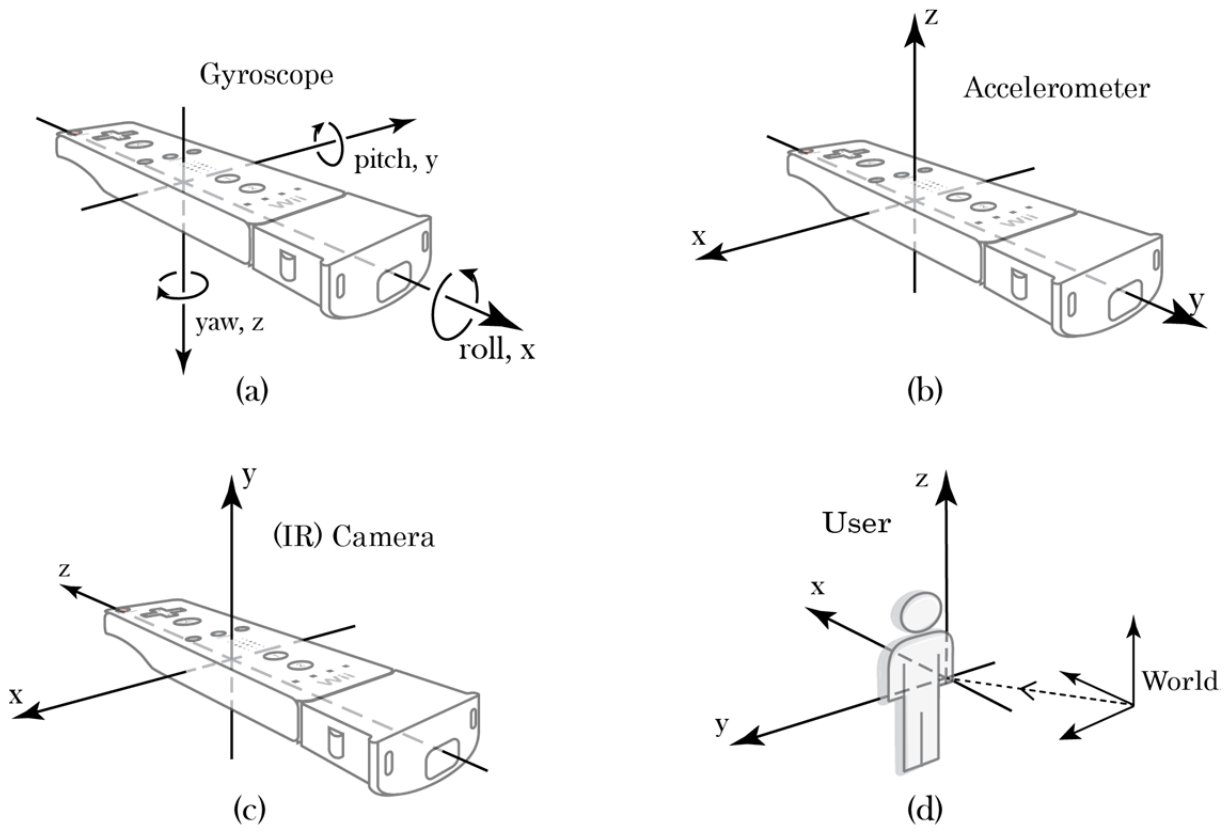


Figure 6.2. Wiimote reference frames: (a) the Gyroscope Frame, (b) the Accelerometer Frame, (c) the (IR) Camera Frame, and (d) the User Frame and the World Frame.

6.3.3 – Beacon Reference Frames

To acquire the position and attitude wrt the beacon, reference frames are set up as shown in Figure 6.3. The User Frame is the same frame specified in Figure 6.2.a. The *Beacon Frame* has the configuration of the Camera Frame with its origin fixed to the front surface of the beacon and with the *y*- and *z*-axes

coplanar to the surface and orthogonal to the front edges. The Beacon Frame is assumed fixed a priori and for this reason is one of the fundamental reference frames for wiimote pose. Even the World Frame is initially set wrt the Beacon Frame.

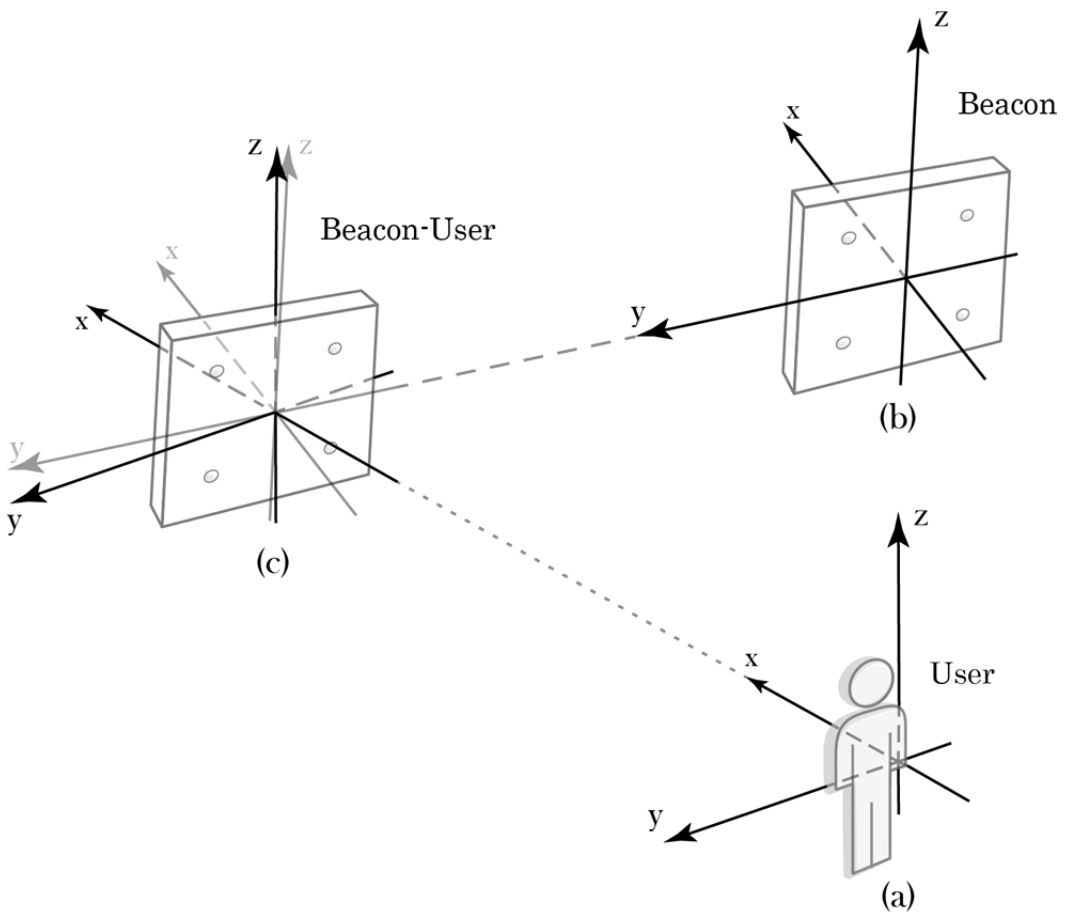


Figure 6.3. Reference frames relating to the extraction of pose from the IR camera: (a) the User Frame, (b) the Beacon Frame, and (c) the Beacon-User Frame.

In the process of calculating pose from IR data, the *Beacon-User Frame* has computational purposes. The Beacon-User Frame has the attitude of the User Frame with the origin of the Beacon Frame. It can be thought of as a rotated copy of the Beacon Frame to match the heading of the User Frame, or as a translated copy of the User Frame to match the origin of the Beacon Frame. The three standard reference frames in Figure 6.3 have three respective OpenCV frames that it uses during computation, which will be referred to as *OpenCV-User Frame, OpenCV-Beacon Frame*, and *OpenCV-Beacon-User Frame*. As shown in Figure 6.4, they are identical except they are rotated along the (–1,–

1,–1) to (+1, +1, +1) axis by 120°. In other words, the *x*-, *y*-, *z*-axes are replaced with the *z*-, *x*-, *y*-axes, respectively.

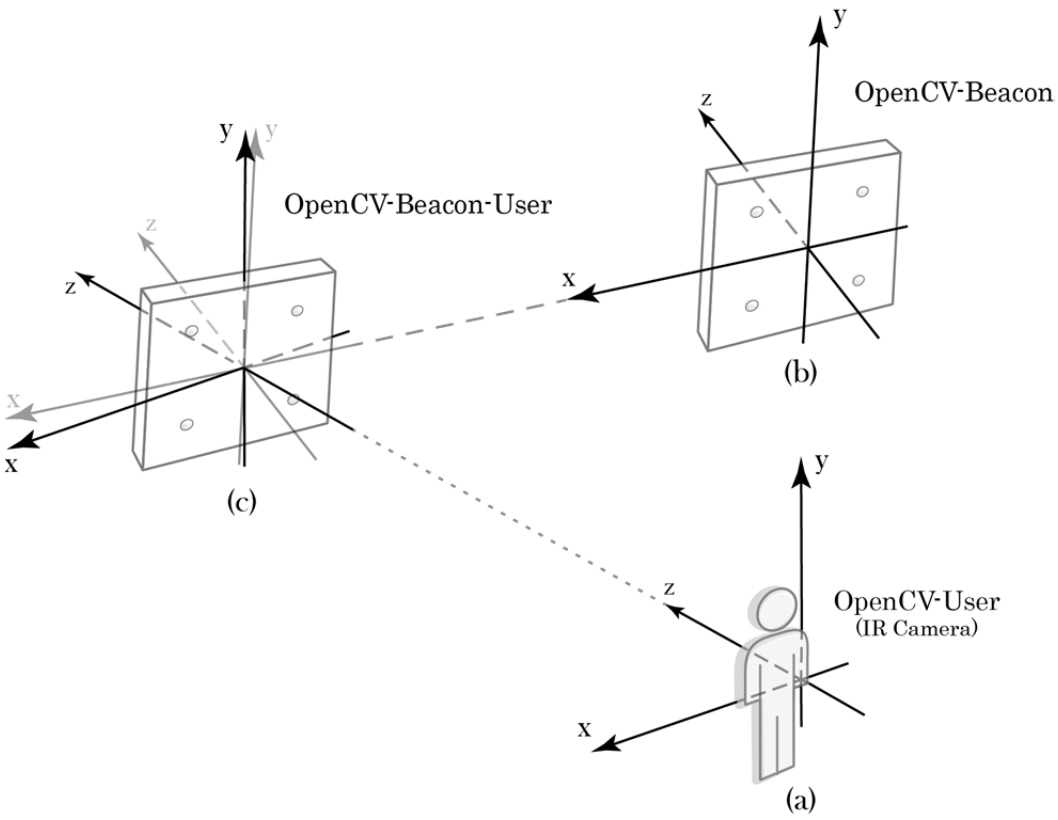


Figure 6.4. Reference frames that are compatible with OpenCV's functions to calculate pose from homography: (a) the OpenCV-User Frame, (b) the OpenCV-Beacon Frame, and (c) the OpenCV-Beacon-User Frame.

The first step in acquiring pose from the IR data is to obtain the LEDs' positions from the IR sensor wrt the IR Camera Frame (same as OpenCV-User Frame) and compare them to their known positions using OpenCV to solve the planar homography (projection transformation of planes). The returned homography is represented as a translation vector (tvecs) and rotation vector (rvecs), which is approximately the pose of the OpenCV-User Frame wrt the OpenCV-Beacon Frame. Changing reference frames, tvecs and rvecs can be interpreted precisely as a transformation from the Beacon-User Frame into the User Frame and as a transformation from the Beacon Frame to the Beacon-User Frame, respectively.

6.3.4 – Robot Reference Frames

The three most important reference frames related to the robot are the Robot Frame, Robot-Ready Frame, and the Tool Frame (Figure 6.5). The *Robot Frame* is attached to the middle of the base of the robot and is the reference frame that all movement of the end effector (and tool) are wrt. This frame is the primary reference the 500C Robot Controller uses when calculating the A465 robot's inverse kinematics. At the end of the robot's end effector is an affixed gripper tool. The *Tool Frame* is positioned in the middle of the grippers. When the robot is turned on, homed, and set to its home position, the robot is said to be in *ready position*. The *Robot-Ready Frame* is permanently fixed to the location of the Tool Frame when the robot is in ready position. In summary, the Robot and the Robot-Ready frames are fixed in space, and the Tool Frame is moveable since it is attached to the end-effector tool. Thus, the Tool Frame overlaps the Robot-Ready Frame when the robot is in ready position.

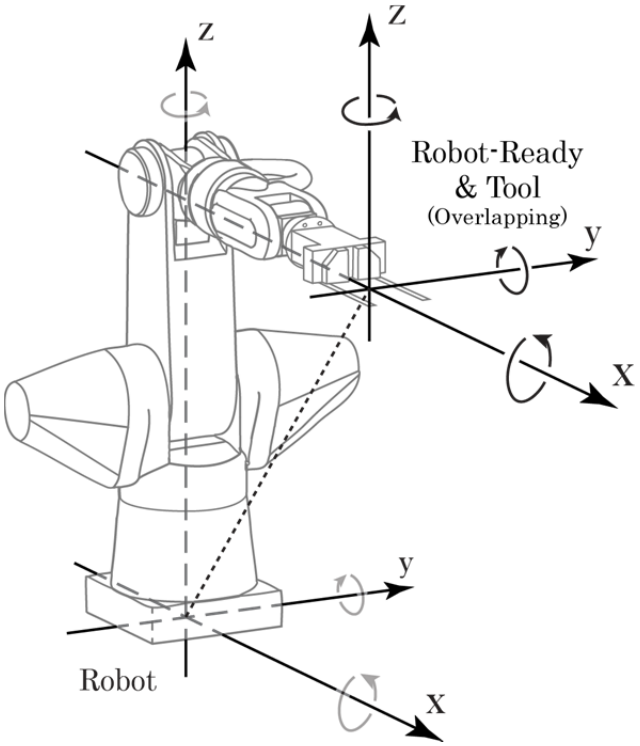


Figure 6.5. Reference frames of the robot: the Robot Frame (attached to robot base), the fixed Robot-Ready Frame (initial tool frame), and the Tool Frame (attached to end-effector tool).

6.3.5 – Using the Reference Frames for Teleoperation

The C500C Robot Controller uses the Robot Frame for reference when moving the Tool Frame, however, teleoperation uses the Robot-Ready Frame. The resultant movements of the robot by movements of the user are shown in Figure 6.6. In Figure 6.6.a, the robot is initially in the robot ready position, but is then moved via teleoperation by the user through the movements in either match-mode teleoperation (Figure 6.6.b) or mirror-mode teleoperation (Figure 6.6.c). The magnitude of the displacement of the Tool Frame wrt the Robot-Ready Frame is identical to the magnitude of the User Frame wrt the World Frame.

In teleoperation *copy mode*, the robot copies the exact motion of the user from respective reference frames without any interpretation (Figure 6.6.b). This mode is useful if teleoperation is performed from the same viewpoint as the robot (e.g., from behind it or with the camera facing the same direction); however, this mode is problematic if the user is facing the robot since the robot will be moving in the opposite direction desired. The preferred teleoperation mode when facing the robot is the mirror mode (Figure 6.6.c). In *mirror mode*, the robot mirrors the motion of the user, which results in the *z*-axis and *pitch* remaining the same, and the *x*-axis, *y*-axis, *yaw*, and *roll* being reversed. The reference frames in Figure 6.6.c are used only for explanatory purposes, since the teleoperation movements are always computed using the reference frames in Figure 6.6.b and when in mirror mode, the appropriate reversals of translation and rotation are applied just prior to sending movement commands to the appropriate robot controller.

For brevity, "Frame" will typically be dropped from the named reference frames when it does not interfere with clarity (e.g., *Beacon Frame* will be referred to as *Beacon*). Furthermore, the reference frames have the first letter capitalized, and physical objects are all lowercase (e.g., *Beacon* refers to the reference Beacon Frame whereas *beacon* refers to the physical object).

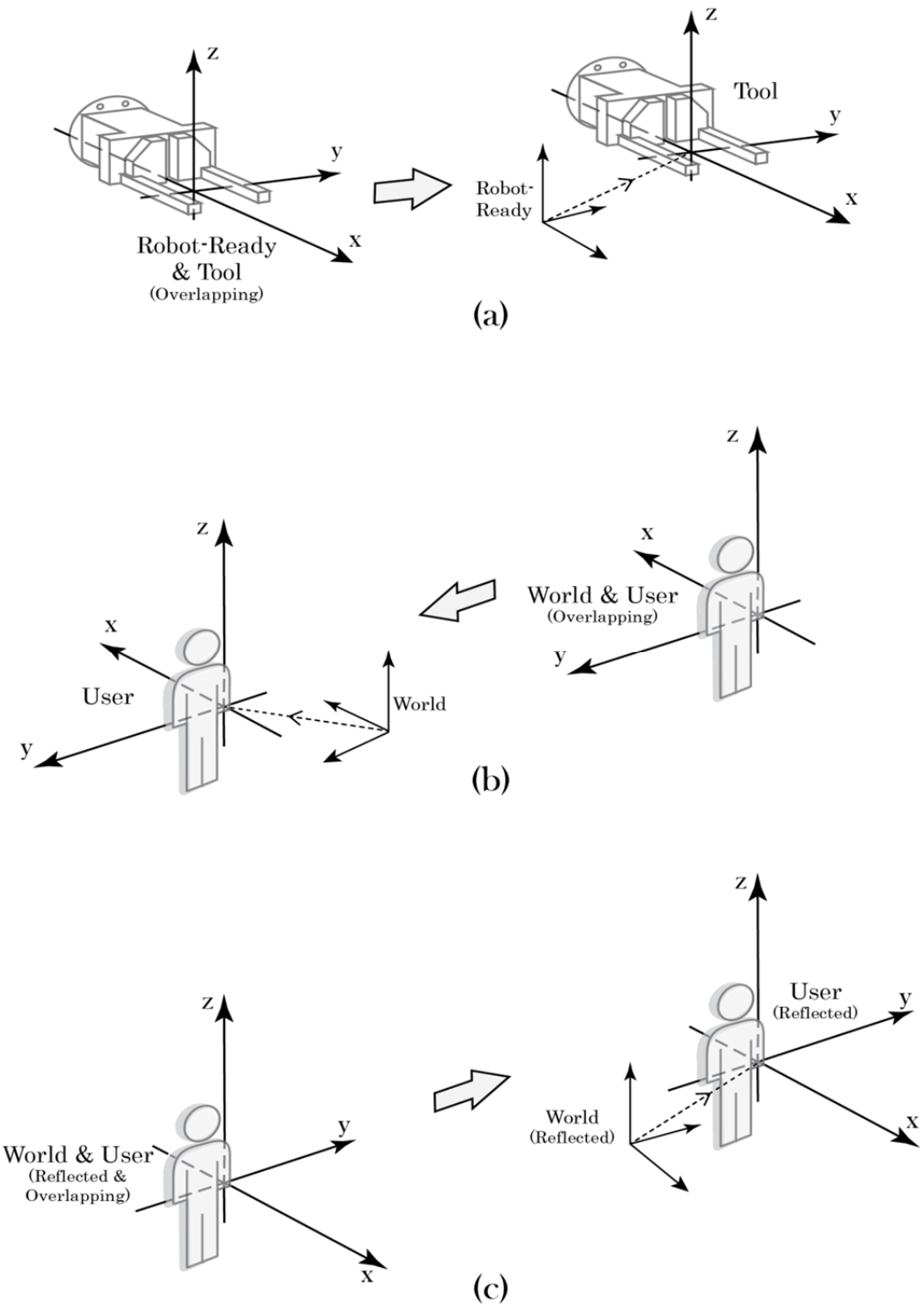


Figure 6.6. Change in pose of reference frames during teleoperation. (a) Movement of robot frames resulting from a change in wiimote position, which resulted from either (b) the movement of the wiimote in 'match mode', or (c) the movement of the wiimote in 'mirror mode'.

6.4 – Wiimote to Robot Interface

Section 6.4.1 provides an overview of the interface, Section 6.4.2 details modification of the WiiYourself! library to increase accelerometer precision, and Section 6.4.3 details implementation of the Teleoperation Module.

6.4.1 – Overview

The Wiimote to Robot Interface is responsible for accessing the wiimote as a HID device, acquiring and processing the incoming byte stream, and sending the sensor values to the Robot Control Server via TCP/IP. Additionally, it used during IR camera calibration to record the (x,y) LED positions reported by the wiimote's IR camera. The wiimote to Robot Interface can be subdivided into the WiiYourself! library (v1.15) and the Teleoperation Module.

6.4.2 – Modification of the WiiYourself! Library

The WiiYourself! library [149] handles the task of connecting and communicating with the wiimote. This library is publicly available and written in C, and was not a contribution of this thesis. However, the source code was modified to increase the precision of the reported accelerometer values. Initially, WiiYourself! reported the *x*, *y*, and *z* accelerometer values with 8-bit precision, although, WiiBrew [153] reported that the wiimote's *x*, *y*, and *z* values are 10-, 9-, and 9-bit, respectively. It was determined that the remaining unaccounted least significant bits were located elsewhere in the raw wiimote bit stream. These bits were identified in the stream and added to the previous most significant byte in a floating point representation, thus decreasing *x*, *y*, and *z* error due to discretization by 4x, 2x, and 2x, respectively.

6.4.3 – Teleoperation Module

The Teleoperation Module uses the WiiYourself! Library to process the wiimote data and then sends them to Robot Control Server. An overview of this process is presented visually in Figure 6.7. As the program is executed, TCP/IP socket communication is established with the Robot Control Server, and the module attempts to connect with the Wiimote. If unable to connect, the program waits until a wiimote is available. Once connected, the program enters the main loop until a termination signal is sent, at which point the wiimote and TCP/IP communication are closed, and the program terminates.

In the main loop, the Teleoperation Module waits for a change in status of the wiimote's state. Upon a status change, a state request is queried for the accelerometers, gyros, IR camera, buttons pressed, and additional wiimote status information such as battery levels and speaker activation. The results are displayed to the screen, encoded in a byte stream, and sent via TCP/IP to the Robot Control Server as detailed in Sect. 6.2.4. During the main loop, if communication to the wiimote is lost, the program is automatically directed to re-establish wiimote communication before resuming the main loop.

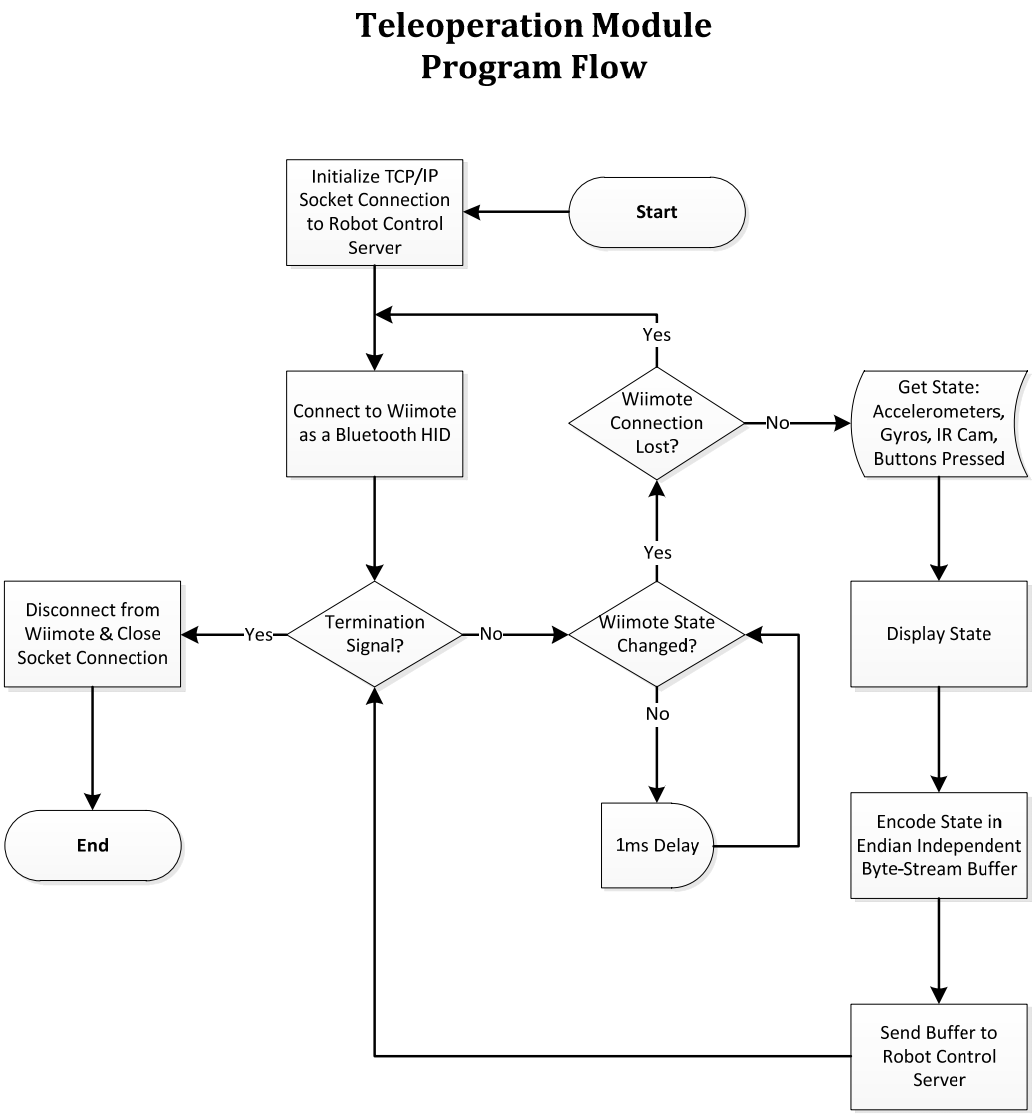


Figure 6.7. Program flow of the Teleoperation Module

6.5 – Robot Control Server

Section 6.5.1 provides an overview of the Robot Control Server, Section 6.5.2 describes the initialization, Section 6.5.3 explains how the system receives wiimote data over TCP/IP, and Section 6.5.4 details the main portion of the program.

6.5.1 – Overview

The Robot Control Server's primary purpose is to process the wiimote's sensor values to accurately estimate the wiimote's 6-DOF pose and to use this pose to control the A465 Robot Manipulator's end effector via the commercial C500C Robot Controller. The Robot Control Server's secondary purpose is to simulate both the commercial controller and the physical robot for testing purposes and when the physical units are unavailable. The respective simulator systems for the C500C Robot Controller and the A465 Robot Manipulator are the Virtual Robot Controller and the Virtual Robot Manipulator, which are shown in Figure 6.1. Furthermore, a detailed overview of the Robot Control Server is visually illustrated by Figure 6.8, where the structure in divided and described by three sections, initialization, TCP/IP server, and the Robot Control Server's main loop.

6.5.2 – Initialization

Initialization is the first task following the execution of the Robot Control Server software, and it can be subdivided into the initialization of the Robot-State System, the pose filter, the TCP/IP asynchronous server, a robot controller, and a robot manipulator.

The Robot-State System maintains the current state of the system. It performs the calculations responsible for changing reference frames and representations for interactions with other modules and devices. Next, the pose filter is calibrated. The pose filter is a subsystem of the Robot-State System and performs all necessary calculations to optimally estimate state from the wiimote sensors.

TCP/IP server begins listening to port 30005 for incoming connections and sets up an I/O interrupt to concurrently receive incoming sensor data from the Wiimote to Robot Interface. Having an asynchronous server allows incoming signals to be processed without interfering with the real-time performance of the system.

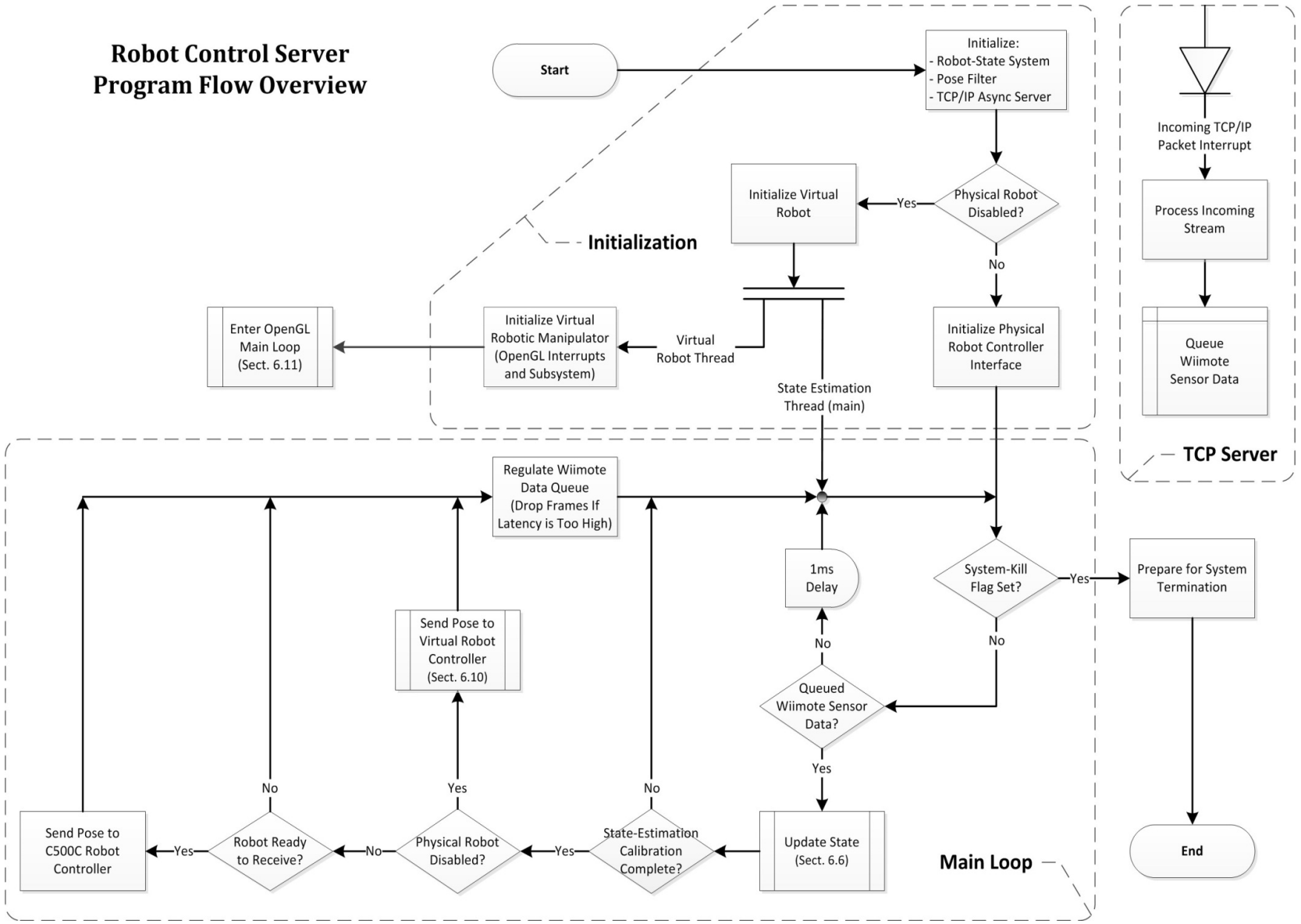


Figure 6.8. Program flow of the Robot Control Server

Prior to program execution, a robot controller flag must be set, which selects the robot that will be used. If the system is connected with the physical robot, C500C Robot Controller communication is initialized. If the physical robot is not present or if testing is required, the Virtual Robot Controller is initialized instead and control is redirected to it. Furthermore, the virtual system initializes the Virtual Robot Manipulator on a parallel thread. The Virtual Robot Manipulator is a 3D virtual robot built within the OpenGL environment (q.v. Sect. 6.9). Following system initialization, the program flow enters the main loop, which is detailed following the description of the TCP/IP interrupt.

6.5.3 – Incoming TCP/IP Interrupt

When the operating system detects an incoming packet stream, an I/O interrupt is executed. The packet is interpreted and unwrapped (q.v. Sect. 6.2.4), and the data stream is processed by the Robot Control Server's asynchronous TCP/IP Server. Specifically, the values of the wiimote's accelerometer (3 floats), gyroscope (3 floats), four IR LED locations (4 floats), and the time elapsed since last refresh are extracted and stored in a WiiData object. This object is pushed to the back of a thread-safe queue (WiiDataQueue) to be processed by the main loop at the next available opportunity.

6.5.4 – Main Loop

The purpose of the main loop is to use the data on the WiiDataQueue to update the robot state (including pose) and to send this pose to a robot controller to teleoperate the robot. At the beginning of each loop iteration, the system-kill flag is checked. If the flag is set to true, the main loop is exited, and the system prepares for termination by cleaning-up resources. Specifically, the socket connection is terminated, the TCP/IP Server is shutdown, and the robot connection prepares for termination. If the physical robot is being used, the robot position is sent to ready mode (starting position) and a command is sent to the C500C Robot Controller to unlink PC control. If the virtual robot is used, the Virtual Robot Manipulator thread is terminated. Following clean-up, the Robot Control Server terminates.

If the system-kill flag's state is false, the main loop queries the WiiDataQueue to determine if any WiiData objects are queued. If no WiiData are queried, the next loop is initiated preceded by a 1 ms delay, which is used to prevent the system from consuming a large amount of resources by continuously checking an empty WiiDataQueue. If there is queued WiiData, the Robot-State System is updated by processing the WiiData and near-optimally estimating the state. The first one hundred wiimote updates are used to calibrate the state-estimation system (q.v. Sect. 6.6.2) and the following updates are used to

estimate state (q.v. Sect. 6.6.3). Next, if calibration is not complete, program flow starts the next loop iteration; if the calibration is complete, pose information is used to update the robot's pose.

When the virtual robot is being used, the pose is sent to the Virtual Robot Controller, where the inverse kinematics is found and the state of the Virtual Robot Manipulator is updated (q.v. Sect. 6.10). When the physical robot is being used, then the C500C Robot Controller is queried if the A465 Robot Manipulator is ready for a new trajectory. If yes, the pose is sent to and used by the C500C Robot Controller to set the end-effector pose. It determines the inverse kinematics and necessary trajectory, and controls the A465 Robot Manipulator to physically perform the trajectory. If the A465 Robot Manipulator is *not* ready, or if the pose was sent to either the Virtual Robot Controller or the C500C Robot Controller, then the WiiDataQueue's length is regulated (refer to Figure 6.8 for program flow clarification). Regulation of the queue insures that occasional slowdowns of the system do not cause the wiimote data to become too out-dated. If the latency of processing the WiiData in the WiiDataQueue does not remain low, WiiData updates are dropped down to two and the elapsed time of the dropped updates are added to the next update queued to minimize interference with pose filter prediction detailed in Figure 6.12. This reduces latency between the wiimote's pose and the robot's pose. Following WiiDataQueue regulation, the main loop begins the next iteration.

6.6 – Updating Robot State

Sect. 6.6.1 provides an overview of how the Robot Control Server updates the robot's state, Sect. 6.6.2 describes the calibration of the pose filter, and finally, Sect. 6.6.3 details the method used to update the state with the pose filter.

6.6.1 – Overview

The robot's state is updated by transforming each of the sensor data's reference frames into a consistent reference frame, processing the IR camera's LED points, and using the pose filter's sensor fusion to optimally combine the sensor data to estimate pose and other relevant robot state information. The representation of this data is changed into the reference frame and representation used by the robot controller to teleoperate the robot manipulator. A detailed visual overview of this process is presented in Figure 6.9 and the remainder of this section discusses it.

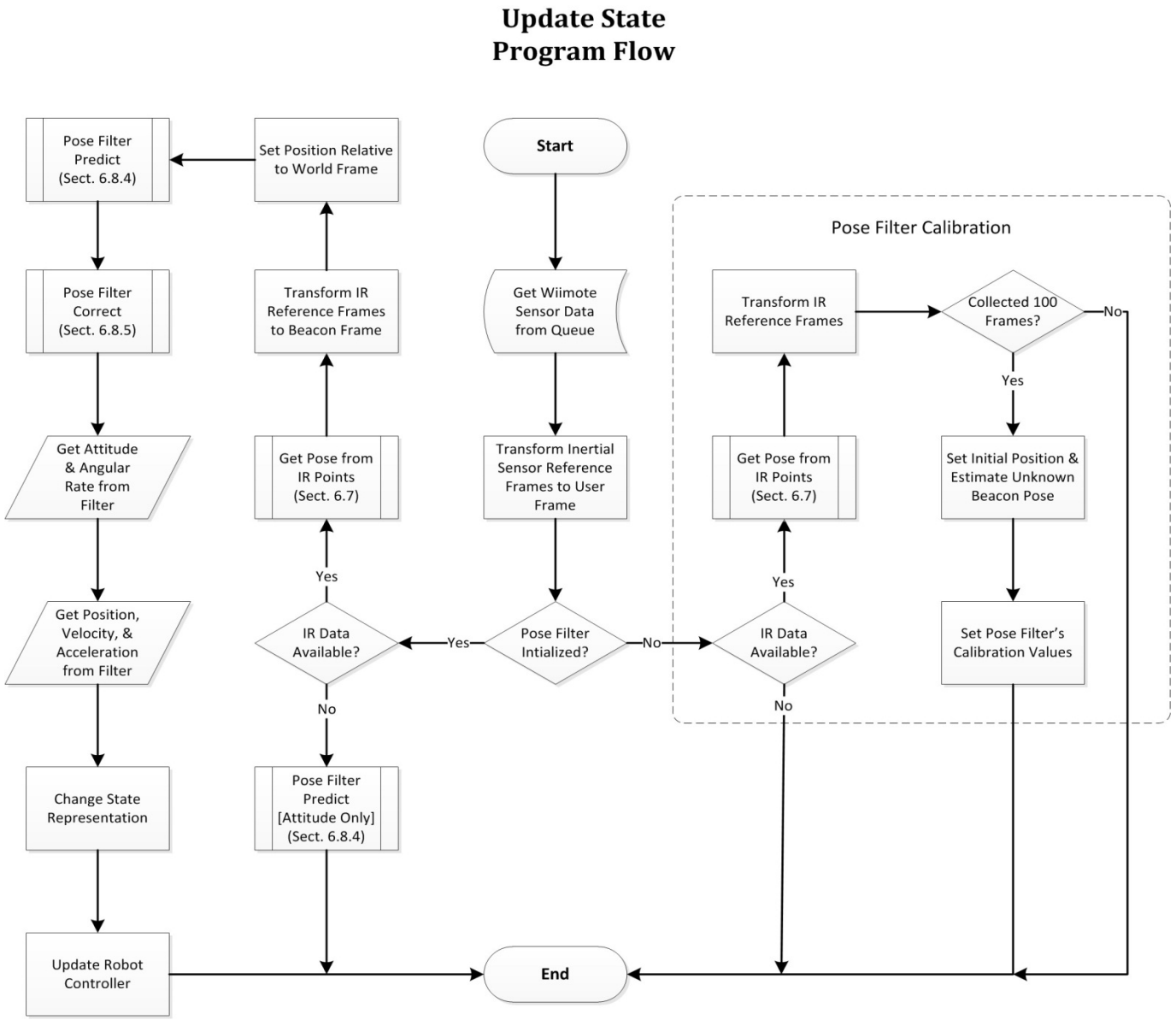


Figure 6.9. Program flow used to update the robot's state

6.6.2 – Pose Filter Calibration

Following connection of the Wiimote to Robot Interface with the Robot Control Server and prior to wiimote tracking, the pose filter is calibrated using wiimote sensor readings while the wiimote is motionless. The first one hundred wiimote updates with a complete set of sensor values are used; however, the first forty gyroscope measurements are dropped to remove measurements acquired during the gyroscope's less accurate transient phase that occurs during warm-up. The wiimote updates are obtained from WiiData objects on the WiiDataQueue (cf. Sect. 6.5.3).

The gyroscope and accelerometer values are transformed from their respective reference frames to be wrt the User, and the IR position and attitude values are transformed to be wrt the Beacon. The average values are obtained by calculating the vector mean of each group of values. Furthermore, the average accelerometer value is normalized to unit-vector length and is referred to as the average gravity vector, which identifies the direction of gravity wrt the User.

The average position is used to set the origin of the World. The average gravity vector, the average user attitude, the average user position, and the average gyroscope-rate vectors are sent to the pose filter for calibration.

The main purpose of pose filter calibration is to estimate the Beacon's attitude wrt the World. Therefore, when the attitude of the User is found wrt the Beacon using IR camera data, it can be transformed to be wrt the World. While the wiimote is in use by the user, the gravity vector is *unknown* wrt the User and wrt the Beacon, but it is *known* wrt the World. Thus, knowing the attitude of the User wrt the World allows the gravity vector to be subtracted from the accelerometer measurements, leading to a true estimate of the wiimote acceleration, which can be used to increase the accuracy of position estimation.

The rotation from the Beacon Frame to World Frame ${}^{\mathrm{W}}_{\mathrm{B}}\bar{q}$ is trivially derived using Eqs. (4.3) and (4.2)

$$ {}^{\mathrm{W}}_{\mathrm{B}}\bar{q} = \left({}^{\mathrm{B}}_{\mathrm{W}}\bar{q}\right)^{-1} \tag{6.1} $$

$$ {}^{\mathrm{W}}_{\mathrm{B}}\bar{q} = \left({}^{\mathrm{B}}_{\mathrm{U}}\bar{q} \otimes {}^{\mathrm{U}}_{\mathrm{W}}\bar{q}\right)^{-1} \tag{6.2} $$

where ${}^{\mathrm{B}}_{\mathrm{W}}\bar{q}$, ${}^{\mathrm{B}}_{\mathrm{U}}\bar{q}$, and ${}^{\mathrm{U}}_{\mathrm{W}}\bar{q}$ are the reference frame transformation from World to Beacon, User to Beacon, and World to User, respectively. The World to User ${}^{\mathrm{U}}_{\mathrm{W}}\bar{q}$ is obtained by extracting the *yaw*, *pitch*, and *roll* from the gravity vector wrt the User using the following Eqs. (6.3), (6.4), and (6.5)

$$yaw = 0.0 \tag{6.3}$$

$$pitch = \begin{cases} -\sin^{-1}\left({}^{\mathrm{U}}_{\mathrm{W}}g_x\right), & -1 < {}^{\mathrm{U}}_{\mathrm{W}}g_x < 1 \\ -\pi/_2, & {}^{\mathrm{U}}_{\mathrm{W}}g_x > 1 \\ \pi/_2, & {}^{\mathrm{U}}_{\mathrm{W}}g_x < -1 \end{cases} \tag{6.4}$$

$$roll = \tan 2^{-1}\left({}^{\mathrm{U}}_{\mathrm{W}}g_y, {}^{\mathrm{U}}_{\mathrm{W}}g_z\right) \tag{6.5}$$

where ${}^{\mathrm{U}}g_x$, ${}^{\mathrm{U}}g_y$, and ${}^{\mathrm{U}}g_z$ are the x, y, and z components of the gravity-wrt-User vector, respectively. Note that $\tan 2^{-1}(\dots,\dots)$ is in the common computational-notation form of (y,x) and not the typical scientific-notation form of (x,y) [5]. To prevent double cover of the $\mathbb{S}^3$ manifold, pitch is limited to the interval $\left[-\pi/_2,\ +\pi/_2\right]$ and roll is limited to $[-\pi,\ +\pi]$[6].

The World to User ${}^{\mathrm{U}}_{\mathrm{W}}\bar{q}$ is obtained from a Euler to quaternion transformation using the obtained *yaw*, *pitch*, and *roll* as found by

$${}^{\mathrm{B}}_{\mathrm{U}}\bar{q} = \bar{q}_{yaw} \otimes \bar{q}_{pitch} \otimes \bar{q}_{roll} \tag{6.6}$$

where $\bar{q}_{yaw}$, $\bar{q}_{pitch}$, and $\bar{q}_{roll}$ are obtained by setting the rotation vectors to $\boldsymbol{\theta}_{yaw} = (yaw, 0, 0)$, $\boldsymbol{\theta}_{pitch} = (0, pitch, 0)$, and $\boldsymbol{\theta}_{roll} = (0, 0, roll)$, which can be trivially converted to the respective axis-angle rotations and then finally to the respective quaternions[7].

Lastly, although the attitude filter's bias converges automatically, it can be sped up by using the gyroscope data obtained during calibration. Specifically, the estimate of the attitude filter's gyroscope biases is set to the respective components of the average gyroscope measurements wrt User.

[5] For more information, seehttp://en.wikipedia.org/wiki/Atan2

[6] Although this causes redundant representation at the Euler singularities (Sect. 4.1.2), it has practical value since it helps the previous trajectory to be known more easily in the neighbourhood of the singularity.

[7] This method is computationally inefficient. To greatly increase efficiency, the terms can be fully multiplied out; however, this optimization was not performed in this research since this operation is very infrequently used by the program and would thus have a negligible effect on performance.

6.6.3 – Updating State

Following calibration, each state update begins by obtaining a wiimote-data update from the queue and the inertial sensor values are transformed to be wrt User. Under normal circumstances, IR data are available; however, when they are not, only the inertial data are used in pose filter prediction (q.v. Sect. 6.8). Since the use of the accelerometer incurs error in position extremely quickly without an absolute update (q.v. Sect. 2.2.1), yet the gyroscopes can be used to predict attitude fairly well, the pose filter only predicts attitude when IR data are unavailable.

While IR data are available, they are used to obtain the absolute pose wrt the OpenCV-Beacon-User (q.v. Sect. 6.7) and transformed to be wrt Beacon. The position and attitude (wrt Beacon), the acceleration (wrt Accelerometer), and the angular rates (wrt Gyroscope) are grouped as a measurement data-object to be used by the pose filter for prediction and correction as detailed in Sect. 6.8.4 and Sect. 6.8.5, respectively.

Following optimal estimation of state from the pose filter, the estimated attitude, angular rate, position, velocity, and acceleration are obtained. The attitude is converted into Euler angles, and the complete state is stored for the current time-step (epoch). If the teleoperation is in mirror mode (default mode), the reference frame is flipped as described in Sect. 6.3.5. Finally, the robot controller is updated, which concludes the state updating process.

6.7 – Pose from IR Points

Sect. 6.7.1 provides an overview of pose extraction from IR data. Sect. 6.7.2 details relevant a priori information about the beacon, Sect. 6.7.3 describes the functions performed to obtain pose, and Sect. 6.7.4 details the correspondence between the image IR points and the beacon reference points, which is necessary to solve the homography.

6.7.1 – Overview

Using prior knowledge known about the IR LED locations on the physical beacon and the IR LED positions projected onto the IR camera sensor, the pose of the IR sensor wrt to the beacon can be estimated. This is performed by finding the correspondence between the IR reference points on the beacon and their projected image points onto the IR sensor, followed by solving the unknown homography between the

3D reference points and the 2D projected points. This process is illustrated in Figure 6.10 and is expanded upon in the following subsections.

6.7.2 – A Priori Beacon Information

To solve the homogaphy, a priori information must be known about salient points on the beacon (i.e. the four locations of the LEDs' centroid wrt OpenCV-Beacon).

The IR beacons used in this research have been precisely built using a computer numerical control (CNC) milling machine, which allows the true locations to be known with high precision and accuracy. Specifically, the LEDs are positioned in a coplanar grid with adjacent separation of 2.54 mm (1 inch) wrt the centre of the LEDs. These four LED positions are used as the reference points when solving the homography.

6.7.3 – Obtaining Pose Using OpenCV

Prior to solving the homography, correspondence between 3D reference points (LED positions wrt OpenCV-Beacon) and 2D image points (wrt IR-Sensor) must be determined. Consequently, the points only contain their individual locations and do not have any other identifying features, which interferes with correspondence. To circumvent this obstacle, the locations of the projected reference points onto the IR sensor are approximated. A full perspective projection using the camera matrix, distortion coefficients, estimated pose of the Camera Frame would be ideal; however, a simpler orthogonal projection method was found to be sufficient to allow the correspondence methods to obtain high correspondence accuracy. The projection method and correspondence are described in Sect. 6.7.4.

Following correspondence, the image points are reordered to match the order of the corresponding reference points. Specifically, the order of the reference points are (low *x*, high *y*), (low *x*, low *y*), (high *x*, high *y*), (high *x*, low *y*), and the image points are put into the same order as the corresponding reference points. Furthermore, the image points are scaled to be wrt the image size used during camera calibration (Sect. 7.4), which is *max_width*=1016, and *max_height*=760. The homography is then solved (q.v. Sect. 6.7.5) using the ordered reference points,

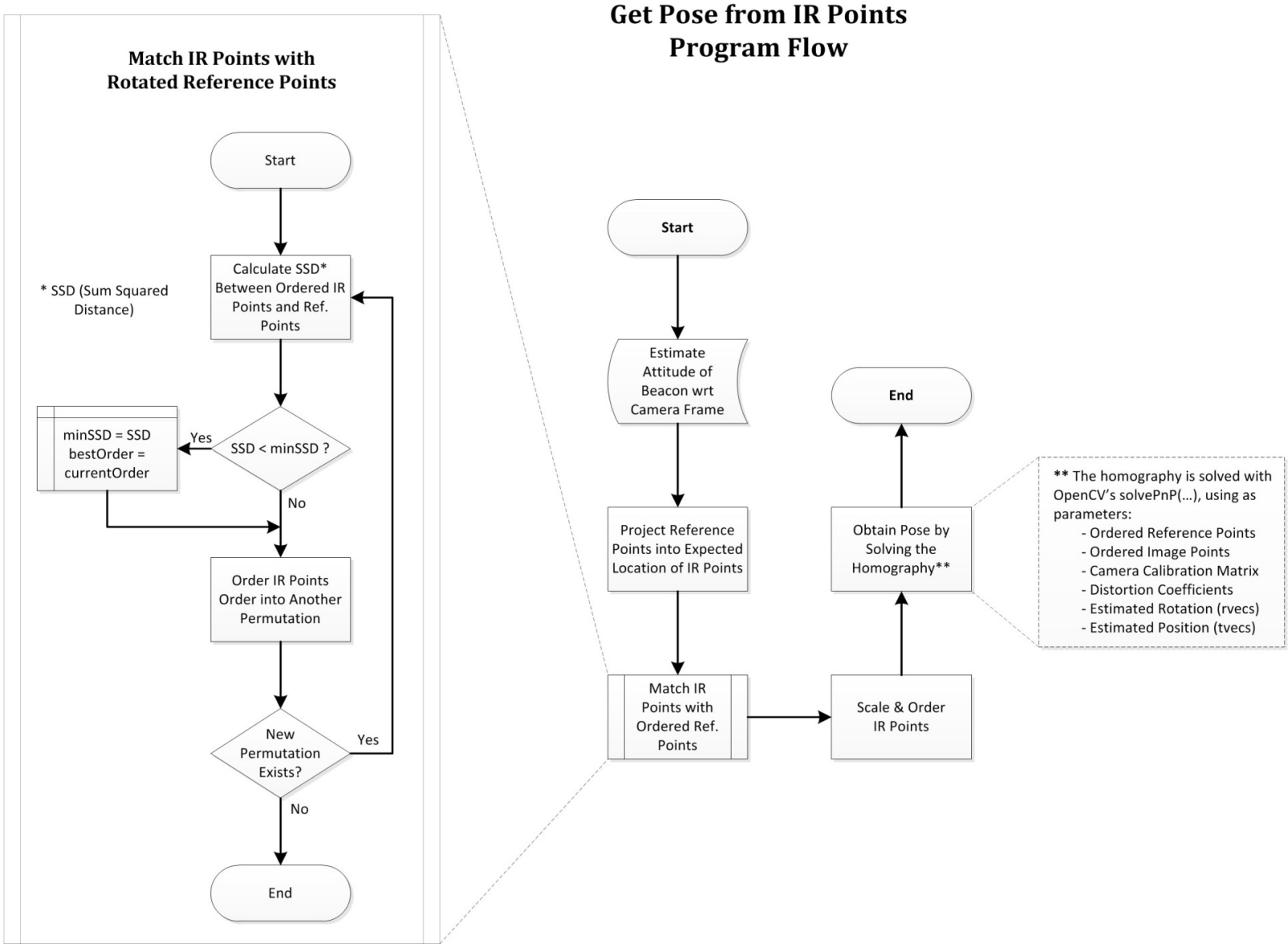


Figure 6.10. Program flow of functions to acquire pose from IR points

the ordered image points, the camera calibration matrix, the distortion coefficients, an approximation of the expected attitude (in rotation vector form), and an approximation of the expected position. The solution of the homography is returned as a rotation vector (rvecs) and a translation vector (tvecs).

6.7.4 – Solving Correspondence between Reference Points and Image Points

To improve correspondence, the (ordered) reference points are transformed by a rotation and a simple orthographic projection. The previous estimate of the User's attitude is obtained from the pose filter and transformed into the Camera Frame wrt the OpenCV-Beacon, which acts as the estimated rotation. The estimated rotation is converted into quaternion form, which is used to transform the reference points about an axis passing through the centre of the four points. Note, the axis is not necessarily orthogonal to the plane the four points are coplanar to. Following rotation, the transformed reference points are then orthographically projected onto the *x*-*y* plane. This whole process can be roughly expressed as a 3D rotation, followed by dropping the *z*-axis, leading to four 2D (approximately) perspective-projected points. The (ordered) projected reference points are used to find correspondence with the unordered image points.

Since there are only four pairs of points requiring correspondence, it is computationally inexpensive to perform a complete comparison between all possible combinations and select the correspondence with the lowest error. Specifically, a function is made to sequentially return the next non-repeating order permutation of the image points in lexicographic order. The image points are compared to the projected reference points for all combinations and the combination that minimizes the error is selected for correspondence.

The error function used is the sum squared error of the Euclidian distance ϵ_{SSM} between the image points and the respective projected reference points, which will be referred to as the sum squared distance (SSD). Using the Euclidian separation, the sum squared error reduces trivially to

$$\begin{aligned} \epsilon_{SSM} &= \sum_{i=1}^{4} \left(\sqrt{\left(x_i^{img} - x_i^{ref}\right)^2 + \left(y_i^{img} - y_i^{ref}\right)^2} \right)^2 \\ &= \sum_{i=1}^{4} \left(\left(x_i^{img} - x_i^{ref}\right)^2 + \left(y_i^{img} - y_i^{ref}\right)^2 \right) \end{aligned} \tag{6.7}$$

where x and y are the components of the point, the superscript img and ref specify image and reference point, respectively, and the subscript i denotes the particular image or reference point. Eq.

(6.7) is used to evaluate each combination. The combination with the lowest SSD is recorded and used to order the image points with the respective reference point prior to solving the homography.

6.7.5 – Solving the Homography Between the Beacon and Projected Image

Once the image points are put in the same order as the corresponding reference points, the planar homography can be solved using the OpenCV function *solvePnP*(…). This nonlinear operation uses an iterative approach to converge to the optimal solution.

The function *solvePnP*(…) takes as parameters: the ordered reference points wrt OpenCV-Beacon, the ordered image points wrt Camera, the camera calibration matrix, the distortion coefficients, the estimated attitude wrt OpenCV-Beacon-User (optional), the estimated position wrt OpenCV-Beacon-User (optional), and a flag specifying whether the estimated attitude and positions are to be used as an initial guess. The parameters are stored in an OpenCV's C++ matrix object (Mat). For an excellent reference on using OpenCV data structures and functions, see [123].

The camera calibration matrix and distortion parameters are found through a novel non-trivial calibration process detailed in Chapter 7. The camera calibration technique should only be used if higher precision is required since it is rather involved. While the camera on each wiimote is the same model, there would be manufacturing tolerances and possible errors that would result in different intrinsic properties of cameras on other wiimotes.c The camera calibration matrix and distortion parameters found by this research could be used as an approximation. However, if greater accuracy is required, the calibration process can be repeated on other wiimotes. The intrinsic parameters are

$$f_x = 1348.6927, \qquad f_y = 1342.3807, \qquad c_x = 496.88116, \qquad c_y = 381.90093$$

$$k_1 = 0.073683679, \qquad k_2 = -0.25457907, \qquad k_3 = -0.23550061$$

$$p_1 = -6.8723600, \qquad p_2 = 0.0019153288$$

where f_x and f_y are the focal lengths in pixel coordinates, (c_x, c_y) is the principal point in pixel coordinates, k_1, k_2, k_3 are the three radial distortion coefficients, and p_1, p_2 are the two tangential distortion coefficients. The details of these parameters are described in Sect. 4.5.

The estimated attitude and the estimated position are optional parameters, which are obtained by the taking the most recent estimate from the pose filter and transforming it to be wrt OpenCV-

Beacon-User. It is useful to use the estimated attitude (stored in rvecs) and position (stored in tvecs) with solvePnP(…) since they decrease the convergence time and greatly decrease the likelihood of obtaining an incorrect solution from the convergence to an incorrect local minimum. Since the beacon LEDs are in a square grid pattern, any 90-degree rotation about the plane the points are coplanar to is a potential solution. Consequently, this creates four potential solutions despite the existence of only one valid solution. By specifying an estimated attitude and position, the initial starting location will likely be within the neighbourhood of convergence of the correct solution, which will make the probability of an incorrect convergence highly improbable under normal operating conditions.

6.8 – Pose Filter

Sect. 6.8.1 provides an overview of pose filtering, Sect. 6.8.2 discusses the high-level implementation of a novel pose filter, Sect. 6.8.3 details initialization of the pose filter, Sect. 6.8.4 details the pose prediction phase, and Sect. 6.8.5 details the pose correction phase.

6.8.1 – Overview

Pose filtering is the process of estimating the most probable state of the position and the attitude given all available prior knowledge. In the context of this research, it is the near-optimal determination of 3-DOF position and 3-DOF attitude of the User wrt the World. Arguably, the most widely used methods for optimally estimating state in general are based on Kalman filtering, which is a highly effective algorithm and is rigorously detailed in Chapter 5. As previously discussed, estimation is performed in Kalman filtering by a two-phase process of prediction and correction. In the *prediction phase*, the estimated state is predicted prior to a measurement reading by propagating the state from the previous time step to the current time step. Next, a new measurement is taken, and then the *correction phase* optimally combines the previously predicted estimated-state with the measurement to obtain a corrected estimated-state.

6.8.2 – Implementation of a Novel Pose Filter

In this research, a novel hierarchical pose filter is developed using multiple Kalman-based filters, which combined, near-optimally estimate 6-DOF pose. Although it is possible to estimate the pose using one large filter, this increases the derivation complexity and increases the processing demands as a result of

the high computational complexity of inverting the $(n \times n)$residual covariance matrix $\mathbf{S}_k$ at every time step, which scales with $O(n^3)$ complexity[8].

To circumvent unfavourable scaling issues of the matrix inversion, the state space is broken up and spread out over four filters since it takes considerably fewer operations to invert the small covariance matrices than one large combined covariance matrix. Consequently, there is a loss in cross-correlations between states in separate filters; however, the states are separated such that the *inter*-cross-correlations are very small and would likely have an insignificant impact on the accuracy of the estimate state.

Position-related states, viz. position, velocity, and acceleration, are handled by three linear Kalman filters (Sect. 5.2), one for each axis (*x, y, z*), and the attitude-related states, viz. attitude and gyro bias, are handled by a MEKF (Sect. 5.5). The pose filter can be viewed as a hierarchical filter that delegates computations to the four subordinate filters, viz. the position filters and the attitude filter. The three specific position filters are referred to as the *x*-filter, *y*-filter, and *z*-filter, where the leading prefix denotes the respective axis wrt the World.

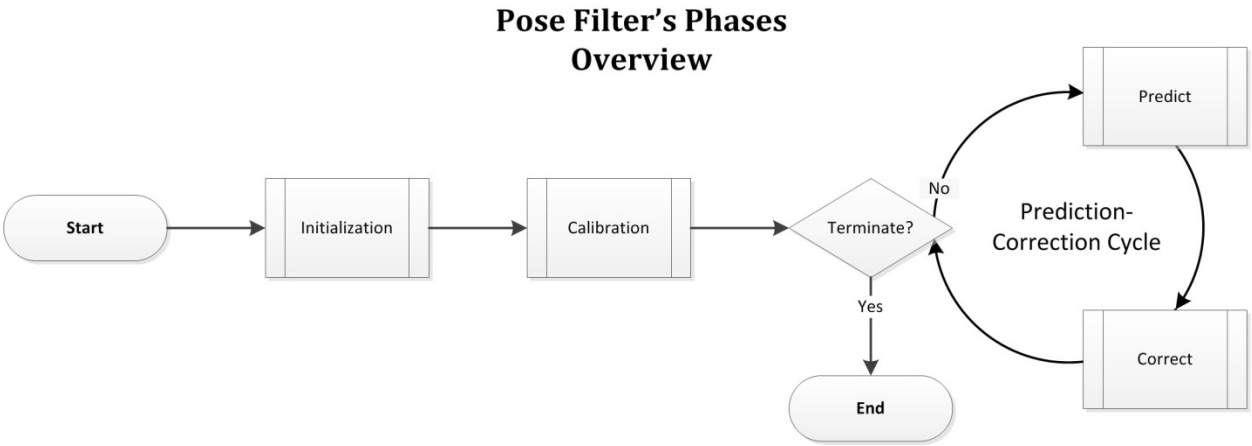


Figure 6.11. High-level overview of the pose filter's phases. Unlike many of the other figures in this chapter, this diagram is not a flow diagram since program flow frequently enters and leaves the pose filter at various times throughout its operation.

[8]Other methods for matrix inversion are asymptotically quicker, although often impractical. The Strassen method [2] has $4.7n^{2.807}$ operations, although, it is less numerically stable and is only useful in practice when $n \geq 110$[3]. Asymptotically faster methods exist, such as the Coppersmith-Winograd algorithm [4] with $O(n^{2.376})$, although, they are not practical until approximately $n > 10{,}000$ [3].

The implementation of the pose filter is explained for the operational stages, which are executed in the order: initialization (Sect. 6.8.3), calibration (Sect. 6.6.2), prediction (Sect. 6.8.4), and correction (Sect. 6.8.5) as illustrated in

Figure 6.11. The calibration was discussed previously in the Update Robot State Section for clarity reasons since many of the operations are handled by the same processes that are used to update the state.

6.8.3– Filter Initialization

Following execution of the Robot Control Server and prior to the connection to the Wiimote to Robot Interface, the pose filter constructor is executed, which begins initialization of the pose filter's four subordinate filters. The position filters' initialization are discussed first, and the details of the attitude filter's initialization follow.

6.8.3.1 – Position Filter Initialization:

Upon initialization of the position filter, the noise variables are set, and all matrices and vectors are set to their initial condition. The position-noise standard deviation was set to $\sigma_p = 0.1$ and the acceleration-noise standard deviation is set to a high value of $\sigma_a = 100$ since there is a considerably large error in subtracting gravity vectors that are even slightly misaligned. Despite the minor influence of the acceleration measurement in the correction phase, it was kept in the measurement update equations by similar systems with more precise sensors, where the influence can be increased by lowering the noise value. Finally, the jerk-noise standard deviation was set to $\sigma_r = 0.45$. Noise selection was done through trial and error, which is a quick, common, and reasonably effective method for selection of these values. However, there is room for improvement in the selection of these values with the use of a more extensive systematic trial and error treatment, or through and automated noise calibration algorithm.

The matrices and vectors are initialized to their respective dimensions, and are set to null, except where otherwise stated. The measurement matrix $\mathbf{H}_k$ is set by Eq. (5.42), the measurement noise covariance $\mathbf{R}_k$ is set by Eq. (5.44), $\mathbf{I}_{3\times3}$ is set to identity (3×3), and the initial covariance matrix $\mathbf{P}_0^+$ is set to demonstrate a large uncertainty about prior knowledge of the state, i.e.,

$$\mathbf{P}_0^+ = 10^3 \cdot \mathbf{I}_{3\times3} \tag{6.8}$$

As stated in Sect. 5.3.1, any large value for the initialization of the covariance will suffice, since the matrix will rapidly converge to the estimated true value of uncertainty.

6.8.3.2 – Attitude Filter Initialization:

Upon attitude filter initialization, the drift-rate noises are set to $\sigma_{\boldsymbol{\omega},1} = \sigma_{\boldsymbol{\omega},2} = 0.2$, the drift-rate ramp noise is set to $\sigma_{\mathbf{b}} = 0.001$, and the camera attitude noise is set to $\sigma_{\mathbf{q}} = 1.0$. Furthermore, the matrices and vectors are initialized to their respective dimensions, and are set to null, except where otherwise stated. The three matrices $\mathbf{I}_{3\times3}$, $\mathbf{I}_{4\times4}$and $\mathbf{I}_{6\times6}$ are set to their respectively sized identities, and the initial attitude estimate is set to

$$\hat{\bar{q}}_0^+ = [0 \quad 0 \quad 0 \quad 1]^{\mathrm{T}} \tag{6.9}$$

the measurement noise covariance is set to

$$\mathbf{R}_k = \sigma_{\mathbf{q}} \cdot \mathbf{I}_{3\times3} \tag{6.10}$$

and the initial error-state covariance is set to

$$\widetilde{\mathbf{P}}_0^+ = 10^3 \cdot \mathbf{I}_{6\times6} \tag{6.11}$$

Following completion of the pose filter's initialization, pose filter calibration is executed as detailed in Sect. 6.6.2. Once the calibration is completed, the system enters the prediction-correction cycle as detailed in the following two subsections.

6.8.4 – Pose Prediction

The pose-prediction phase estimates the state at the current time step immediately preceding a measurement. However, for practical purposes the prediction is performed following a measurement but treated as though it was performed immediately before. This allows the elapsed time since the last measurement correction to be known, which is useful in propagating the state and covariance. The pose filter predicts attitude and position. The attitude prediction is performed by the attitude filter (MEKF implementation) and three position filters (KF implementation).

The process begins by obtaining the current gyro measurement vector from the most recent WiiData object. Since this vector is wrt to Gyro, it requires a transform to be wrt User in order to be in a consistent reference frame with the system. The gyro measurement vector wrt User $\boldsymbol{g}_k$ and the time

step Δt (time since last measurement) are sent to the attitude filter's prediction function where an estimate of the a priori attitude $\hat{\bar{q}}_k^-$, the a priori gyro bias $\widehat{\mathbf{b}}_k^-$, and the a priori error-state covariance $\widetilde{\mathbf{P}}_k^-$ are obtained. The specifics of the attitude filters prediction function are fully detailed by the seven steps is Sect. 5.5.9. If only the attitude is to be estimated due to lack of position-update data (q.v. Sect. 6.6.3), then the prediction phase is complete. However, if the position measurement data are available, then the positions are predicted.

Position prediction is obtained via three position filters. To differentiate the symbols of each axis, an axis-label marking (vis. $^{\mathrm{x}}$, $^{\mathrm{y}}$, or $^{\mathrm{z}}$) is added to the symbol as a pre-superscript, which expresses what axis the symbol is wrt. The a priori state vector wrt the *x*-axis ${}^{\mathrm{x}}\hat{\mathbf{x}}_k^- = [{}^{\mathrm{x}}\hat{x}_k^- \quad {}^{\mathrm{x}}\hat{\dot{x}}_k^- \quad {}^{\mathrm{x}}\hat{\ddot{x}}_k^-]^{\mathrm{T}}$ and the a priori covariance wrt the *x*-axis ${}^{\mathrm{x}}\mathbf{P}_k^-$ is predicted by the *x*-filter using the four steps detailed in Sect. 5.2.3. Furthermore, the *y*-filter and *z*-filter predict ${}^{\mathrm{y}}\hat{\mathbf{x}}_k^-$, ${}^{\mathrm{y}}\mathbf{P}_k^-$ and ${}^{\mathrm{z}}\hat{\mathbf{x}}_k^-$, ${}^{\mathrm{z}}\mathbf{P}_k^-$ using the same method as the *x*-filter. However, it should be noted that each filter has its own variables. Thus, for technical accuracy all symbols in Sect. 5.2 should be marked with the appropriate axis-label marker depending on the filter that is currently performing prediction. For example, when using the *y*-filter, the symbols in Sect. 5.2 should be replaced as follows $\hat{\mathbf{x}}_k^- \rightarrow {}^{\mathrm{y}}\hat{\mathbf{x}}_k^-$, $\mathbf{z}_k \rightarrow {}^{\mathrm{y}}\mathbf{z}_k = [{}^{\mathrm{y}}z_{p,k} \quad {}^{\mathrm{y}}z_{a,k}]^{\mathrm{T}}$, etc. Pose filter prediction is complete following prediction of the three position filters.

The complete a priori state space of the pose filter is defined as

$$ {}^{\mathrm{P}}\hat{\mathbf{x}}_k^- \triangleq \begin{bmatrix} {}^{\mathrm{x}}\hat{\mathbf{x}}_k^- \\ {}^{\mathrm{y}}\hat{\mathbf{x}}_k^- \\ {}^{\mathrm{z}}\hat{\mathbf{x}}_k^- \\ \hat{\bar{q}}_k^- \\ \widehat{\mathbf{b}}_k^- \end{bmatrix} \tag{6.12} $$

and the complete covariance matrix of the pose filter is defined as

$$ {}^{\mathrm{P}}\mathbf{P}_k^- \triangleq \begin{bmatrix} {}^{\mathrm{z}}\mathbf{P}_k^- \\ {}^{\mathrm{z}}\mathbf{P}_k^- \\ {}^{\mathrm{z}}\mathbf{P}_k^- \\ \widetilde{\mathbf{P}}_k^- \end{bmatrix} \tag{6.13} $$

The program flow for the pose filter's prediction phase is illustrated in Figure 6.12.

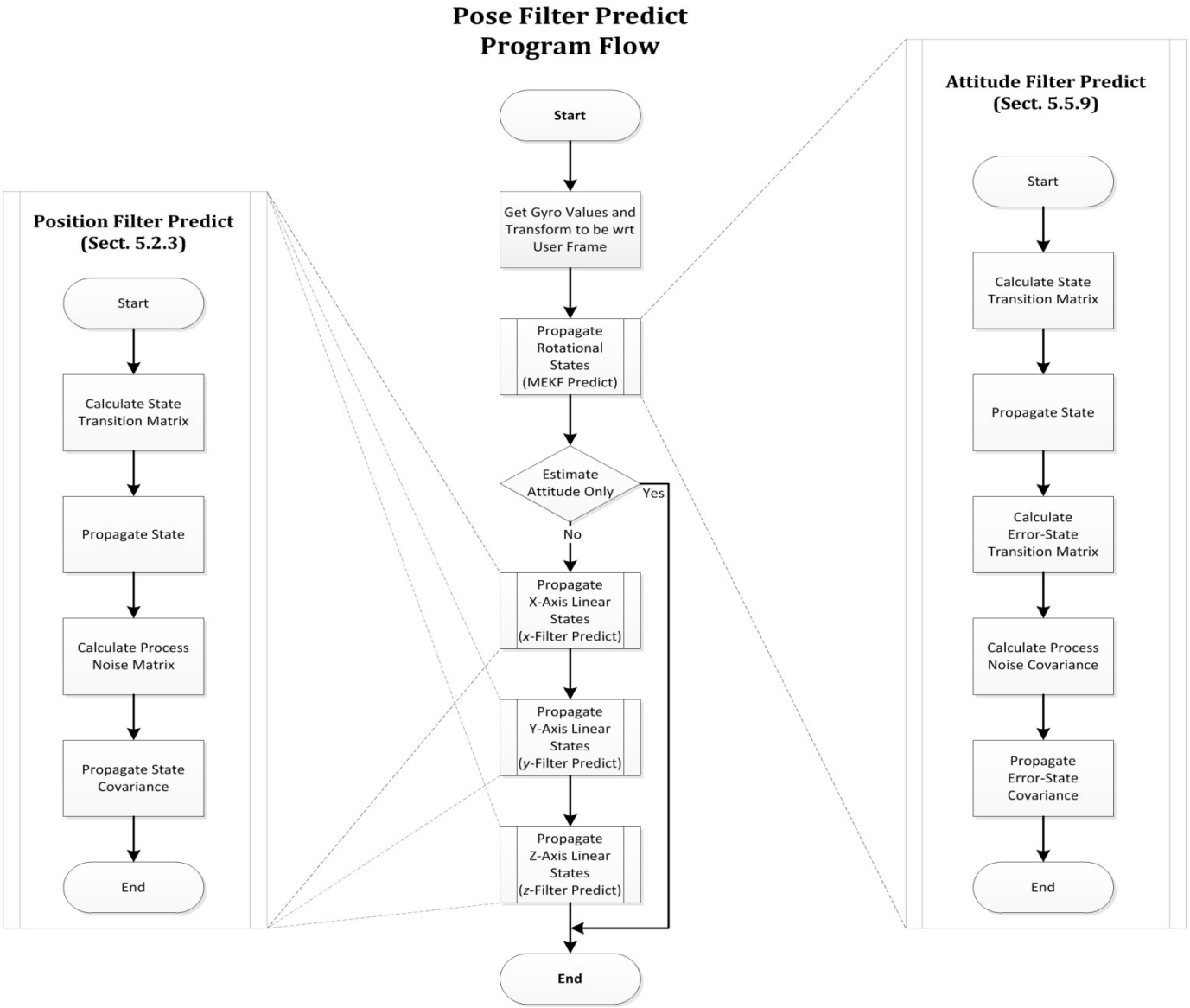


Figure 6.12. Program flow of the state-prediction functions. (Center) Pose filter. (Left) Kalman Filters used for position. (Right) MEKF for attitude prediction

6.8.5 – Pose Correction

The pose correction phase updates the state at the current time step immediately following a measurement by optimally correcting the predicted state. Similar to the prediction phase, the attitude is updated with the attitude filter, and the positions are updated with the position filters.

Upon entering the correction phase, the measured rotation vector (wiimote attitude wrt Beacon) from the solved homography is obtained from the most recent measurement. It is converted to quaternion form, transformed from Beacon to World to obtain the attitude wrt World $\mathbf{z}\left({}^{\mathrm{W}}\boldsymbol{\theta}_k^{\mathrm{c}}\right)$. The attitude filter's measurement vector is set to attitude wrt World, i.e., $\mathbf{z}(\boldsymbol{\theta}_k^{\mathrm{c}}) = \mathbf{z}\left({}^{\mathrm{W}}\boldsymbol{\theta}_k^{\mathrm{c}}\right)$. The a priori rotational states (vis $\hat{\bar{q}}_k^-$ and $\hat{\mathbf{b}}_k^-$) are updated with the a posteriori attitude quaternion $\hat{\bar{q}}_k^+$ and the a posteriori bias $\hat{\mathbf{b}}_k^+$, respectively, by using the attitude filter's correction function. The specifics of the attitude filter's correction phase are detailed in the ten steps in Sect. 5.5.10. Note that the attitude filter's attitude estimate, which describes the wiimote's attitude, is wrt the World, i.e. ${}^{\mathrm{W}}\hat{\bar{q}}_k^-$ and ${}^{\mathrm{W}}\hat{\bar{q}}_k^+$; however, this marking will be dropped for brevity, yet should be assumed.

The position vector from the most recently solved homography is transformed from Beacon-User to World in order to obtain the position measurement vector wrt World ${}^{\mathrm{W}}\mathbf{z}_p = \left[{}^{\mathrm{W}}z_{p,x} \quad {}^{\mathrm{W}}z_{p,y} \quad {}^{\mathrm{W}}z_{p,z}\right]^{\mathrm{T}}$. The estimated a posteriori attitude (wrt World) $\hat{\bar{q}}_k^+$ is taken from the attitude filter and is used to transform the accelerometer measurement vector from User to World to obtain the accelerometer measurement vector wrt World ${}^{\mathrm{W}}\mathbf{z}_{\mathrm{acc}}$. Since the gravity wrt World is known ${}^{\mathrm{W}}\mathbf{g} \triangleq [0 \quad 0 \quad 1]^{\mathrm{T}}$, an estimate of the actual acceleration of the wiimote wrt World (acceleration measurement vector) is obtained by

$$ {}^{\mathrm{W}}\mathbf{z}_{\mathrm{a}} = \left[{}^{\mathrm{W}}z_{a,x} \quad {}^{\mathrm{W}}z_{a,y} \quad {}^{\mathrm{W}}z_{a,z}\right]^{\mathrm{T}} = {}^{\mathrm{W}}\mathbf{z}_{\mathrm{acc}} - {}^{\mathrm{W}}\mathbf{g} \tag{6.14} $$

where ${}^{\mathrm{W}}z_{a,x}$, ${}^{\mathrm{W}}z_{a,y}$, and ${}^{\mathrm{W}}z_{a,z}$ are the acceleration measurement vector's *x*-, *y*-, and *z*-components, respectively.

The linear states are updated using the three position filters. The measurement vectors for each filter are set to

$$ {}^{\mathrm{x}}\mathbf{z}_k = [{}^{\mathrm{x}}z_{p,k} \quad {}^{\mathrm{x}}z_{a,k}]^{\mathrm{T}} = \left[{}^{\mathrm{W}}z_{p,x} \quad {}^{\mathrm{W}}z_{a,x}\right]^{\mathrm{T}} \tag{6.15} $$

$$ {}^{\mathrm{y}}\mathbf{z}_k = [{}^{\mathrm{y}}z_{p,k} \quad {}^{\mathrm{y}}z_{a,k}]^{\mathrm{T}} = \left[{}^{\mathrm{W}}z_{p,y} \quad {}^{\mathrm{W}}z_{a,y}\right]^{\mathrm{T}} \tag{6.16} $$

$$ {}^{z}\mathbf{z}_k = \begin{bmatrix} {}^{z}z_{p,k} & {}^{z}z_{a,k} \end{bmatrix}^{T} = \begin{bmatrix} {}^{W}z_{p,z} & {}^{W}z_{a,z} \end{bmatrix}^{T} \tag{6.17} $$

The position filters' correction functions calculate the a posteriori state vectors ${}^{x}\hat{\mathbf{x}}_k^+$, ${}^{y}\hat{\mathbf{x}}_k^+$, ${}^{z}\hat{\mathbf{x}}_k^+$ and the a posteriori covariance matrices ${}^{x}\mathbf{P}_k^+$, ${}^{y}\mathbf{P}_k^+$, ${}^{z}\mathbf{P}_k^+$ as detailed in the five steps in Sect. 5.2.4.

By this point, the 6-DOF pose of the wiimote wrt World is estimated using the 3-DOF attitude from the attitude filter, and the 1-DOF position from the three position filters. Specifically, the pose filter's a posteriori state space ${}^{P}\hat{\mathbf{x}}_k^+$ and the pose filter's a posteriori covariance matrix ${}^{P}\mathbf{P}_k^+$ are estimated by the pose filter, and defined respectively as

$$ {}^{P}\hat{\mathbf{x}}_k^+ \triangleq \begin{bmatrix} {}^{x}\hat{\mathbf{x}}_k^+ \\ {}^{y}\hat{\mathbf{x}}_k^+ \\ {}^{z}\hat{\mathbf{x}}_k^+ \\ \hat{q}_k^+ \\ \hat{\mathbf{b}}_k^+ \end{bmatrix} \tag{6.18} $$

$$ {}^{P}\mathbf{P}_k^+ \triangleq \begin{bmatrix} {}^{z}\mathbf{P}_k^+ \\ {}^{z}\mathbf{P}_k^+ \\ {}^{z}\mathbf{P}_k^+ \\ \widetilde{\mathbf{P}}_k^+ \end{bmatrix} \tag{6.19} $$

This completes the implementation of the pose filter's correction phase, which is summarized in Figure 6.13.

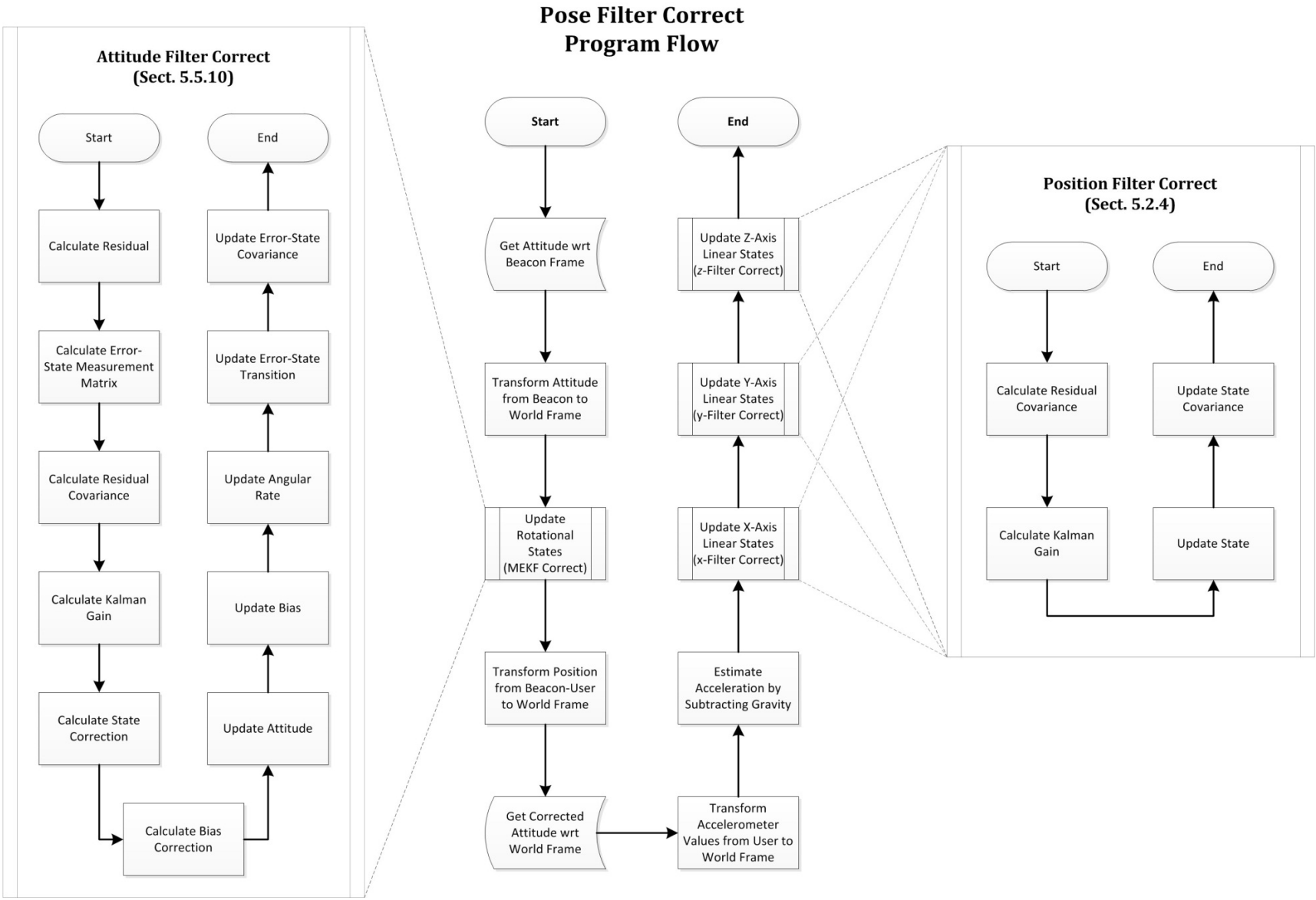


Figure 6.13. Program flow of the state-correction functions. (Center) Pose filter. (Left) Kalman Filters used for position. (Right) MEKF for attitude prediction

6.9 – Kinematic Modelling of the Manipulator

Sect. 6.9.1 provides an overview of kinematic modelling, Sect. 6.9.2 describes the forward kinematics, Sect. 6.9.3 details a model of the manipulator based on the Denavit–Hartenberg convention, and Sect. 6.9.4 provides the derivation of the inverse kinematics.

6.9.1 – Overview and Context

When developing a new motion capture system for a physical robotic manipulator, it is necessary to build a simulator to test the developed system since there is a high risk of damaging the costly robot with software that has not been thoroughly tested. The simulator built in this research consists of the Virtual Robot Controller (Sect. 6.10) and the Virtual Robot Manipulator (Sect. 6.11). The Virtual Robot Controller and the Virtual Robot Manipulator simulate the C5005C controller and the A465 Robot Manipulator, respectively. The simulator provides safety, which is important during testing and useful during regular operation since calculating detailed spatial information of all the links and joints can be used to ensure that the robot does not collide with its environment.

To create the simulator, the kinematics of the A465 Robot Manipulator requires modelling. The Virtual Robot Manipulator computes the forward kinematics using a straightforward model (Sect. 6.9.2). To calculate the inverse kinematics, a complex model is used. The robot is modelled using the Denavit–Hartenberg convention (Sect. 6.9.3) and the inverse kinematics is derived in closed form via kinematics decoupling (Sect. 6.9.4). The solution to the inverse kinematics is later used by the Virtual Robot Controller to control the Virtual Robot Manipulator.

6.9.2 – Forward Kinematics

The forward kinematics is calculated to obtain the position and orientation of each link to render the virtual robot in OpenGL. Instead of letting OpenGL perform the transformations of each link, the Virtual Robot Manipulator system computes the transformations, which allows the position of the links and joints to be known and used for safety purposes.

The transformation of each reference frame at joint i wrt the base reference frame (Robot Frame) must be known. Specifically, ${}^{0}_{1}\mathbf{H}_k$, ${}^{0}_{2}\mathbf{H}_k$, ${}^{0}_{3}\mathbf{H}_k$, ${}^{0}_{4}\mathbf{H}_k$, ${}^{0}_{5}\mathbf{H}_k$, ${}^{0}_{6}\mathbf{H}_k$must be known and are calculated recursively for $1 \leq i \leq 6$ by

$$ {}^{0}_{i}\mathbf{H}_k = {}_{i-1}{}^{0}\mathbf{H}_k \, {}^{i-1}_{i}\mathbf{H}_k \tag{6.20} $$

where

$$ {}^{i-1}_{i}\mathbf{H}_k = \begin{bmatrix} \mathbf{A}_{i,k}(\theta_{i,k}) & \begin{bmatrix} \mathbf{0}_{2\times 1} \\ l_i \end{bmatrix} \\ \mathbf{0}_{1\times 3} & 1 \end{bmatrix} \tag{6.21} $$

and the initial transformation is

$$ {}^{0}_{0}\mathbf{H}_k = \mathbf{I}_{4\times 4} \tag{6.22} $$

The link length l_i for links $1 \leq i \leq 6$ are 33.0 cm, 30.5 cm, 10.0 cm, 23.0 cm, 7.6 cm, and 17.0 cm, respectively. Furthermore, the rotation matrices $\mathbf{A}_{i,k}$ are defined by Eq. (4.7), which simplifies for revolute joints to

$$ \mathbf{A}_{i,k}(\theta_{i,k}) = \begin{cases} \begin{bmatrix} \cos(\theta_{i,k}) & -\sin(\theta_{i,k}) & 0 \\ \sin(\theta_{i,k}) & \cos(\theta_{i,k}) & 0 \\ 0 & 0 & 1 \end{bmatrix}, & i \in [1,4,6] \\ \begin{bmatrix} \cos(\theta_{i,k}) & 0 & \sin(\theta_{i,k}) \\ 0 & 1 & 0 \\ -\sin(\theta_{i,k}) & 0 & \cos(\theta_{i,k}) \end{bmatrix}, & i \in [2,3,5] \end{cases} \tag{6.23} $$

where the joint angles $\theta_{i\in[1,\ldots,6],k}$ are provided by the Virtual Robot Controller.

6.9.3 – Manipulator Modelling using Denavit–Hartenberg Parameters

To simplify the inverse kinematics, the virtual robot is modelled using the Denavit–Hartenberg (DH) convention, manufacture specifications of the A465 Robot Manipulator, and measurements taken from the physical robot in the lab. Furthermore, it is modelled such that the inverse kinematics can be easily determined via kinematic decoupling.

Seven sets of axes are positioned and aligned on the model using the DH convention as shown in Figure 6.14. Moreover, the axes are modelled such that joints 4, 5, and 6 are treated as a spherical joint at position $\mathbf{c}_k$ located at Joint 5, and will be referred to as the wrist centre. This allows the problem to be decomposed into an inverse orientation problem and an inverse position problem.

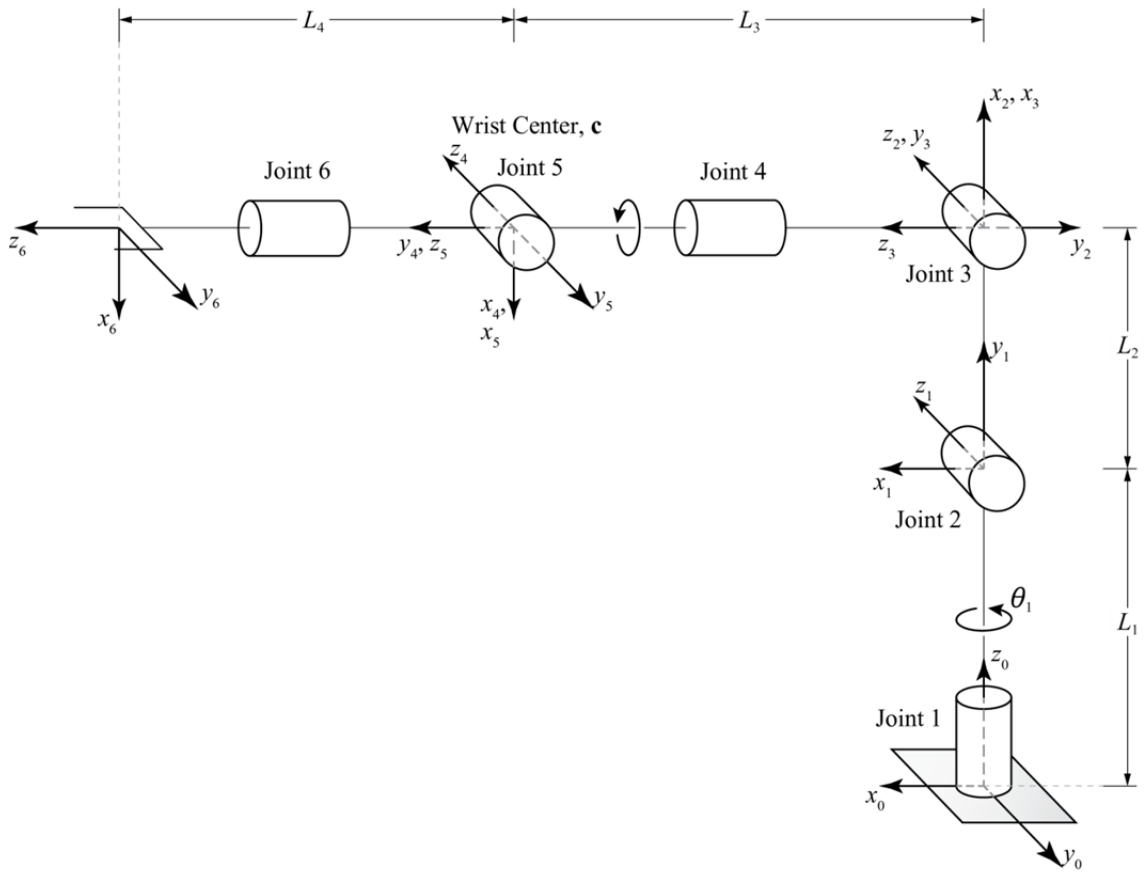


Figure 6.14. Modelling of the robotic manipulator using the Denavit–Hartenberg convention.

L_1 is the length of Link 1 (33.0 cm), L_2 is the length of Link 2 (30.5 cm), L_3 is the length of Link 3 and Link 4 (33.0 cm), and L_4 is the remaining length to the centre of the tool (24.6 cm), which consists of Link 5, Link 6, and a portion of the tool. Each joint i has an angle of rotation $\theta_{i,k}$, with direction specified by the positive z_{i-1}-axis and the right-hand-rule.

Table 6.1. Denavit–Hartenberg parameters for the robotic manipulator for the axes as chosen in Figure 6.14.

Link i	Link Offset d_i	Joint Angle ϑ_i	Link Length a_i	Link Twist α_i
1	L_1	$\theta_{1,k}$	0	$\pi/2$
2	0	$\theta_{2,k} + \pi/2$	L_1	0
3	0	$\theta_{3,k} + \pi/2$	0	$\pi/2$
4	L_3	$\theta_{4,k} + \pi$	0	$\pi/2$
5	0	$\theta_{5,k}$	0	$-\pi/2$
6	L_4	$\theta_{6,k}$	0	0

Using Figure 6.14, the link offset d_i, joint angle ϑ_i, link length a_i, and link twist α_i are obtained as listed in Table 6.1. Note, the zero value of the joint angle is arbitrarily set $\vartheta_i = 0$ when the joints are in the middle of their range of motion, which for joints 2, 3, and 5 is when the adjacent links are fully extended in a straight line.

Using the DH parameters in Table 6.1, the forward and inverse kinematics is calculated.

6.9.4 – Inverse Kinematics

The pose filter provides the pose for teleoperation of the Tool wrt the Robot; however, before the pose can be used by the Virtual Robot Manipulator, the inverse kinematics must be solved by the Virtual Robot Controller. Using the model detailed in Sect. 6.9.3 and visually presented in Figure 6.14, the inverse kinematics can be solved analytically using kinematic decoupling (c.f. Sect. 4.3.3).

Before solving the inverse kinematics problem (Eq. (4.57)) to obtain the six joint angles $\theta_{i\in[1,..,6],k}$, the homogenous transformation ${}^{0}_{6}\mathbf{T}_k$ must be found by solving the decoupled forward kinematics, i.e.

$$ {}^{0}_{6}\mathbf{T}_k = {}^{0}_{3}\mathbf{T}_k \, {}^{3}_{6}\mathbf{T}_k \tag{6.24} $$

where the position decoupled transformation ${}^{0}_{3}\mathbf{T}_k$ and the orientation decoupled transformation ${}^{3}_{6}\mathbf{T}_k$ are obtain by

$$ {}^{0}_{3}\mathbf{T}_k = \mathbf{T}_{1,k}\mathbf{T}_{2,k}\mathbf{T}_{3,k} \tag{6.25} $$

$$ {}^{3}_{6}\mathbf{T}_k = \mathbf{T}_{4,k}\mathbf{T}_{5,k}\mathbf{T}_{6,k} \tag{6.26} $$

Furthermore, each of the homogenous transformation matrices can be decomposed into

$$ {}^{j}_{i}\mathbf{T}_k = \begin{bmatrix} {}^{j}_{i}\mathbf{A}_k & {}^{j}_{i}\mathbf{d}_k \\ \mathbf{0}_{1\times 3} & 1 \end{bmatrix} \tag{6.27} $$

Using Eqs. (4.55), (6.25), and (6.26), the DH parameters in Table 6.1, and simplifying, the decoupled homogenous transformations ${}^{0}_{3}\mathbf{T}_k$ and ${}^{3}_{6}\mathbf{T}_k$ are found, which can be decomposed using (6.27) to obtain the decoupled rotation matrices

$$ {}^{0}_{3}\mathbf{A}_k = \begin{bmatrix} -c_{23}c_1 & s_1 & -s_{23}c_1 \\ -c_{23}s_1 & -c_1 & -s_{23}s_1 \\ -s_{23} & 0 & c_{23} \end{bmatrix} \tag{6.28} $$

$$ {}^{3}_{6}\mathbf{A}_k = \begin{bmatrix} s_4s_6 - c_4c_5c_6 & c_6s_4 + c_4c_5s_6 & c_4s_5 \\ -c_4s_6 - c_5c_6s_4 & c_5s_4s_6 - c_4c_6 & s_4s_5 \\ c_6s_5 & -s_5s_6 & c_5 \end{bmatrix} \tag{6.29} $$

where for clarity c_i, c_{ij}, s_i, and s_{ij} are defined as

$$c_i \triangleq \cos(\theta_{i,k}) \tag{6.30}$$
$$c_{ij} \triangleq \cos(\theta_{i,k} + \theta_{j,k}) \tag{6.31}$$
$$s_i \triangleq \sin(\theta_{i,k}) \tag{6.32}$$
$$s_{ij} \triangleq \sin(\theta_{i,k} + \theta_{j,k}) \tag{6.33}$$

The last three joint angles $\theta_{4,k}, \theta_{5,k}, \theta_{6,k}$ are contained in the orientation decoupled rotation matrix

$${}^{3}_{6}\mathbf{A}_k = [{}^{0}_{3}\mathbf{A}_k]^{\mathrm{T}} \times {}^{0}_{6}\widehat{\mathbf{A}}_k \tag{6.34}$$

where the estimate attitude matrix ${}^{0}_{6}\widehat{\mathbf{A}}_k$is obtained by converting the most recent a posteriori attitude $\hat{\bar{q}}_k^+$ into rotation matrix form. The a posteriori attitude $\hat{\bar{q}}_k^+$ is obtained from the pose filter.

Before extracting the angles, a few definitions must be made. The elements of the estimated attitude matrix are defined as

$${}^{0}_{6}\widehat{\mathbf{A}}_k \triangleq \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \tag{6.35}$$

Additionally, the elements of the spherical joint rotation matrix are defined as

$${}^{3}_{6}\mathbf{A}_k \triangleq \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \tag{6.36}$$

The elements r_{mn} and h_{mn} are updated at each time step k; however, the time step marking is dropped for brevity.

Using (6.34) and substituting in (6.36), (6.28), and (6.35), the following equations are obtained

$$h_{11} = -r_{31}s_{23} - r_{11}c_{23}c_1 - r_{21}c_{23}s_1 \tag{6.37}$$
$$h_{12} = -r_{32}s_{23} - r_{12}c_{23}c_1 - r_{22}c_{23}s_1 \tag{6.38}$$
$$h_{13} = -r_{33}s_{23} - r_{13}c_{23}c_1 - r_{23}c_{23}s_1 \tag{6.39}$$

$$h_{21} = r_{11}s_1 - r_{21}c_1 \tag{6.40}$$
$$h_{22} = r_{12}s_1 - r_{22}c_1 \tag{6.41}$$
$$h_{23} = r_{13}s_1 - r_{23}c_1 \tag{6.42}$$

$$h_{31} = r_{31}c_{23} - r_{21}s_{23}s_1 - r_{11}s_{23}c_1 \tag{6.43}$$
$$h_{32} = r_{32}c_{23} - r_{22}s_{23}s_1 - r_{12}s_{23}c_1 \tag{6.44}$$
$$h_{33} = r_{33}c_{23} - r_{23}s_{23}s_1 - r_{13}s_{23}c_1 \tag{6.45}$$

Setting (6.29) equal to (6.36) provides nine nonlinear equations

$$\begin{bmatrix} s_4s_6 - c_4c_5c_6 & c_6s_4 + c_4c_5s_6 & c_4s_5 \\ -c_4s_6 - c_5c_6s_4 & c_5s_4s_6 - c_4c_6 & s_4s_5 \\ c_6s_5 & -s_5s_6 & c_5 \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \tag{6.46}$$

where h_{mn} are obtained by (6.37) through (6.45). Induction is used on (6.46) to solve the last three joint angles $\theta_{4,k}, \theta_{5,k}, \theta_{6,k}$ given the first three $\theta_{1,k}, \theta_{2,k}, \theta_{3,k}$. For example, using the last column of equations provides

$$c_4s_5 = -r_{13}c_{23}c_1 - r_{23}c_{23}s_1 - r_{33}s_{23} = h_{13} \tag{6.47}$$
$$s_4s_5 = r_{13}s_1 - r_{23}c_1 = h_{23} \tag{6.48}$$
$$c_5 = -r_{13}s_{23}c_1 - r_{23}c_{23}s_1 + r_{33}c_{23} = h_{33} \tag{6.49}$$

which can be used to extract $\theta_{4,k}$ and $\theta_{5,k}$ when the last three joint angles are not in a gimbal-lock singularity. Further details are provided below.

Before the last three joint angles $\theta_{4,k}, \theta_{5,k}, \theta_{6,k}$ can be fully solved for, the first three $\theta_{1,k}, \theta_{2,k}, \theta_{3,k}$ must be obtained. The following paragraphs will be devoted to this task.

The final position of the Tool (wrt the Robot, i.e. $\boldsymbol{o}_0$) is the estimated a posteriori position

$${}^0_6\widehat{\mathbf{p}}_k = \begin{bmatrix} {}^{\mathrm{x}}x_k^+ \\ {}^{\mathrm{y}}x_k^+ \\ {}^{\mathrm{z}}x_k^+ \end{bmatrix} \tag{6.50}$$

where the a posteriori position ${}^{\mathrm{x}}x_k^+, {}^{\mathrm{y}}x_k^+, {}^{\mathrm{z}}x_k^+$ is obtained from the respective position filters.

The position of the wrist centre (wrt the Robot, i.e. $\boldsymbol{o}_0$) has components

$$\mathbf{c}_k \triangleq \begin{bmatrix} c_{x,k} \\ c_{y,k} \\ c_{z,k} \end{bmatrix} \tag{6.51}$$

Since the final position ${}^0_6\widehat{\mathbf{p}}_k$ and attitude ${}^0_6\widehat{\mathbf{A}}_k$ are known, the wrist centre can be obtain from a trivial geometric analysis

$$\mathbf{c}_k = {}^0_6\widehat{\mathbf{p}}_k - L_4 \cdot {}^0_6\widehat{\mathbf{A}}_k \times [0 \quad 0 \quad 1]^{\mathrm{T}} \tag{6.52}$$

which reduces to

$$c_{x,k} = {}^{\mathrm{x}}x_k^+ - L_4r_{13} \tag{6.53}$$
$$c_{y,k} = {}^{\mathrm{y}}x_k^+ - L_4r_{23} \tag{6.54}$$
$$c_{z,k} = {}^{\mathrm{z}}x_k^+ - L_4r_{33} \tag{6.55}$$

By this point, all equations are constrained sufficiently to allow the inverse kinematics to be solved through induction and a geometric analysis.

The first three joint angles $\theta_{1,k}, \theta_{2,k}, \theta_{3,k}$ can be obtained from an analysis of the geometry from the the wrist centre $\mathbf{c}_k$. There are two solutions for the first joint angle

$$\theta_{1\mathrm{a},k} = \mathrm{atan2}\left(c_{y,k}, c_{x,k}\right) \tag{6.56}$$

$$\theta_{1\mathrm{b},k} = \mathrm{atan2}\left(c_{y,k}, c_{x,k}\right) + \pi \tag{6.57}$$

However, (6.57) is outside the range of motion of the A465 Robot Manipulator, thus, only (6.56) is used. Note, the trigonometric arctangent is computed using the computational form $\mathrm{atan2}(y, x)$, as opposed to the mathematical form $\mathrm{atan2}(x, y)$.

Next, the second joint angle is obtained by

$$\theta_{2,k} = \mathrm{atan2}\left(z_c - L_1, \sqrt{{c_{x,k}}^2 + {c_{y,k}}^2}\right) \tag{6.58}$$

There are two solutions for the third joint angle

$$\theta_{3\mathrm{a},k} = \mathrm{atan2}\left(+\sqrt{1 - {D_k}^2}, D_k\right) \tag{6.59}$$

$$\theta_{3\mathrm{b},k} = \mathrm{atan2}\left(-\sqrt{1 - {D_k}^2}, D_k\right) \tag{6.60}$$

where

$$D_k = \frac{{c_{x,k}}^2 + {c_{y,k}}^2 + \left(c_{z,k} - L_1\right)^2 - L_2^2 - L_3^2}{2L_2L_3} \tag{6.61}$$

Qualitatively, the first solution (6.59) for the third joint angle $\theta_{3\mathrm{a},k}$ geometrically expresses the robot in *arm-up mode*, and the second solution (6.60) $\theta_{3\mathrm{b},k}$ geometrically expresses the robot in *arm-down mode*.

The last three joint angles $\theta_{4,k}, \theta_{5,k}, \theta_{6,k}$ are obtained from the geometrically-decoupled orientation set of equations. Specifically, they are extracted from the use of induction on the nine equations of (6.46). Rearranging (6.49), two solutions for the fifth joint angle are obtained

$$\theta_{5\mathrm{a},k} = \mathrm{atan2}\left(+\sqrt{1 - h_{33}^2},\ h_{33}\right) \tag{6.62}$$

$$\theta_{5\mathrm{b},k} = \mathrm{atan2}\left(-\sqrt{1 - h_{33}^2},\ h_{33}\right) \tag{6.63}$$

The equations used to solve joint angles $\theta_{4,k}$ and $\theta_{6,k}$ depend on the equations used to solve $\theta_{5,k}$. If (6.62) is used, $s_{\theta_{5,k}} > 0$, and if (6.63) is used, $s_{\theta_{5,k}} < 0$. The joint angles $\theta_{4,k}$ and $\theta_{6,k}$ are obtained from induction on last column and bottom row of (6.46), respectively, while the last three joints are not in a singularity state. There are four theoretical solutions for $\theta_{5,k}$. For the first two situations, there is one solution for $\theta_{4,k}$ and $\theta_{6,k}$. Specifically,

when $s_{\theta_{5,k}} > 0$,

$$\theta_{4\mathrm{a},k} = \operatorname{atan2}(h_{13}, h_{23}) \tag{6.64}$$

$$\theta_{6\mathrm{a},k} = \operatorname{atan2}(-h_{31}, h_{32}) \tag{6.65}$$

when $s_{\theta_{5,k}} < 0$,

$$\theta_{4\mathrm{b},k} = \operatorname{atan2}(-h_{13}, -h_{23}) \tag{6.66}$$

$$\theta_{6\mathrm{b},k} = \operatorname{atan2}(h_{31}, -h_{32}) \tag{6.67}$$

The last two theoretical solutions occur when the last three joint angles create gimbal-lock singularities. This occurs when $\theta_{5,k} = 0$ rad and $\theta_{5,k} = \pi$ rad, which occurs when $h_{33} = 1$ and $h_{33} = -1$, respectively. When $h_{33} = 1$, then $\theta_5 = 0$ and (6.46) reduces to

$$\begin{bmatrix} s_4 s_6 - c_4 c_6 & c_6 s_4 + c_4 s_6 & 0 \\ -c_4 s_6 - c_5 s_4 & s_4 s_6 - c_4 c_6 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \tag{6.68}$$

Therefore, when $h_{33} = 1$, the joint angles $\theta_{4,k}$ and $\theta_{6,k}$ are obtained from

$$\theta_{4\mathrm{c},k} + \theta_{6\mathrm{c},k} = \operatorname{atan2}(h_{11}, h_{21}) = \operatorname{atan2}(h_{11}, -h_{12}) \tag{6.69}$$

There are infinitely many solutions for $\theta_{4,k}$ and $\theta_{6,k}$ when using (6.69), therefore to minimize joint rotation, the fourth joint angle at time *k* is set to the joint angle at the previous time step; specifically, $\theta_{4\mathrm{c},k} = \theta_{4\mathrm{c},k-1}$.

Using the same methodology, the joint angles $\theta_{4,k}$ and $\theta_{6,k}$ can be obtained analogically when $h_{33} = -1$ by

$$\theta_{4\mathrm{d},k} - \theta_{6\mathrm{d},k} = \operatorname{atan2}(-h_{11}, -h_{12}) \tag{6.70}$$

Although it is useful to identify all theoretical solutions, they are not all valid in practice. Eq. (6.70) is not used because it is only applicable to a singularity that is outside of the physical range of motion of the A465 Robot Manipulator.

6.10 – Virtual Robot Controller

Sect. 6.10.1 provides an overview of the Robot Controller, and Sect. 6.10.2 details the implementation.

6.10.1 – Overview

The Virtual Robot Controller is responsible for controlling the Virtual Robot Manipulator. Its program flow is straightforward, as illustrated in Figure 6.15. The Virtual Robot Controller obtains the most recent estimate of the pose from the pose filter and updates the end-effector pose with the obtained estimate. Next, the inverse kinematics is solved via the ten steps in Sect. 6.10.2 to obtain the six joint angles. Finally, the Virtual Robot Manipulator is updated with the six joint angles.

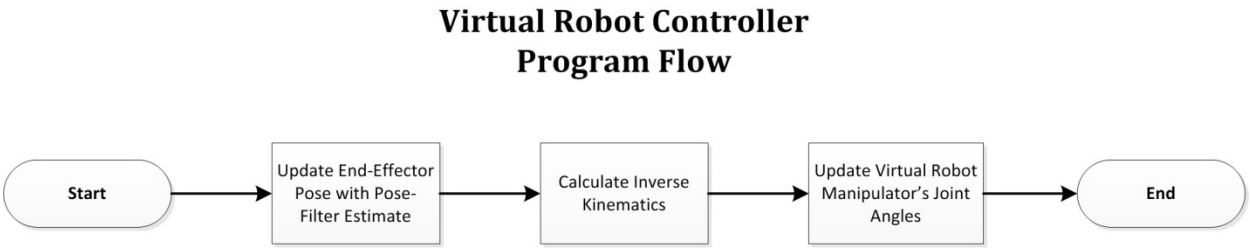


Figure 6.15. Program flow for the Virtual Robot Controller, which is used to operate the Virtual Robot Manipulator

6.10.2 – Implementing the Inverse Kinematics Solution

The inverse kinematics was derived for this system in Sect. 6.9.4. This section uses those equations at the implementation level. The inverse kinematics is solved by the following ten steps:

First, obtain the a posteriori attitude $\hat{\bar{q}}_k^+$ and the a posteriori position ${}_6^0\hat{\mathbf{p}}_k = [{}^{\mathrm{x}}\hat{x}_k^+ \quad {}^{\mathrm{y}}\hat{x}_k^+ \quad {}^{\mathrm{z}}\hat{x}_k^+]^{\mathrm{T}}$, from the pose filter.

Second, obtain the estimated attitude matrix ${}_6^0\hat{\mathbf{A}}_k$ by converting the most recent a posteriori attitude $\hat{\bar{q}}_k^+$ into rotation matrix form.

Third, calculate the wrist centre position $\mathbf{c}_k$ from (6.51), (6.53), (6.54), and (6.55).

Fourth, calculate the relevant h_{mn} values from (6.37)-(6.45), where r_{mn} are the elements of the estimated attitude matrix ${}_6^0\widehat{\mathbf{A}}_k$ as defined in (6.35).

Fifth, calculate the first joint angle $\theta_{1,k} = \theta_{1\mathrm{a},k}$ using (6.56).

Sixth, calculate the second joint angle $\theta_{2,k}$ using (6.58).

Seventh, calculate the third joint angle $\theta_{3,k}$ using (6.59) when the robot is in arm-up mode, and using (6.60) when the robot is in arm-down mode, where D is obtained by (6.61).

Eight, check if $|h_{33} - 1| < \epsilon$, which indicates that the last three angles $\theta_{4,k}, \theta_{5,k}, \theta_{6,k}$ are in the neighbourhood of a singularity. Use a small value for the error, e.g. $\epsilon = 0.0000001$. If the inequality is true, set $\theta_{4,k} = \theta_{4,k-1}$, $\theta_{5,k} = 0$, calculate $\theta_{6,k} = \theta_{6\mathrm{c},k}$ using (6.69), and end the inverse kinematics computations. If the inequality is false, continue to the ninth step.

Ninth, calculate the remaining two solutions for the last three joint angles $\theta_{4,k}, \theta_{5,k}, \theta_{6,k}$. Specifically, calculate $\theta_{5\mathrm{a},k}$, $\theta_{4\mathrm{a},k}$, and $\theta_{6\mathrm{a},k}$ using (6.62), (6.64), and (6.65), respectively, and calculate $\theta_{5\mathrm{b},k}$, $\theta_{4\mathrm{b},k}$, and $\theta_{6\mathrm{b},k}$ using (6.63), (6.66), and (6.67), respectively.

Tenth, test *solution a* ($\theta_{5\mathrm{a},k}$, $\theta_{4\mathrm{a},k}$, $\theta_{6\mathrm{a},k}$) and *solution b* ($\theta_{5\mathrm{b},k}$, $\theta_{4\mathrm{b},k}$, $\theta_{6\mathrm{b},k}$) against the A465 Robot Manipulator's range-of-motion constraints, and select the valid solution (if it exists) for $\theta_{5,k}$, $\theta_{4,k}$, and $\theta_{6,k}$ that are closest to the previous angles $\theta_{5,k-1}$, $\theta_{4,k-1}$, and $\theta_{6,k-1}$. This completes the inverse kinematics computations.

When the pose cannot be reached even without range-of-motion considerations, the angles are set to be invalid and handled appropriated by the Virtual Robot Manipulator.

6.11 – Virtual Robot Manipulator

Sect. 6.11.1 provides an overview of the Virtual Robot Manipulator, which is composed of a spatial model and a graphical model. Sect. 6.11.2 details the spatial model, and Sect. 6.11.3 details the graphical model.

6.11.1 – Overview

The Virtual Robot Manipulator provides a virtual representation of the A465 Robot Manipulator for testing, safety, and visualization purposes. It is initialized on a separate thread by the Robot Control Sever. The Virtual Robot Manipulator is modularized into a spatial model component and a graphical component. The spatial model stores the spatial information of the links and joints, contains information of the vertices and surface faces, and performs the forward kinematics calculations. The graphical model is an OpenGL-based visual representation of the spatial model. Screen captures of the Virtual Robot Manipulator are shown in the results subsection of the experimental testing of the complete system (Sect. 8.5.2).

6.11.2 – Implementation of the Spatial Model

The spatial model contains the spatial representation of the virtual robot and its forward kinematics functions. This model is primarily composed of links and joints.

Links represent the rigid structures of the virtual robot and store all spatial information of the 3D structure in space. They are initialized with a length and a width, and based on these dimensions, the spatial model creates the required vertices, surfaces, and the direction of the surface normals. There are six robot links, with length as documented in the A465 Robot Manipulator's specifications, and one initial link that represents the surface the robot is attached to.

Joints represent the movable segments that connect adjacent links, and are controlled by the Virtual Robot Controller by specifying a joint angle. They are initialized with an angle of rotation and a range of motion, which specifies the angle between adjacent links and the constraints on this angle as documented in the A465 Robot Manipulator's specification, respectively.

Each time the Virtual Robot Controller calculates the inverse kinematics, the solutions to the six joint angles $\theta_{i\in[1,\ldots,6]}$ are sent to the six respective joints on the virtual robot. At the next virtual robot update, the graphical model causes the spatial model to computer the forward kinematics using the six joint angles and the procedure detailed in Sect. 6.9.2. Specifically, using (6.20) the six homogenous transformation matrices ${}^{0}_{i\in[1,\ldots,6]}\mathbf{H}_{k}$ are computed and used to transform the vertices and surface normals of the spatial model from their initial position to the current position.

6.11.3 – Implementation of the Visualization System

The visualization system is a visual representation of the spatial model displayed on the computer monitor with the use of OpenGL. The graphical model consists of the main OpenGL thread and four software interrupts, vis. Draw Screen Interrupt, Virtual Robot Update Interrupt, Keyboard Interrupt(s), and the Window Resize Interrupt.

The main OpenGL thread initializes Glut (OpenGL variant), the display mode, the new window, the rendering system and the software interrupts. Following initialization, the thread enters the Glut main loop, which diverts thread control to the Glut subsystems until a system termination signal is sent.

The Draw Screen Interrupt performs the drawing of the graphical model to the video card. Specifically, it calls a function that prepares OpenGL, sets the reference frame, and sets up the lighting system. Next, the function creates the 3D geometry in OpenGL of each link and joint that is specified by the spatial model's vertices and normal planes. The 3D geometry is rendered to the video buffer. A virtual model is drawn only when all the joint angles are solvable by the Virtual Robot Controller. However, when the joint angles are not solvable due to the pose being out of theoretical range, only the final link is drawn. If the joint angles are solvable theoretically but out of range practically (due to being outside the specified range of motion of the joint), then the full robot is drawn, but all joints out of range are painted red. Once the 3D geometry is rendered to the video buffer, the video card's active buffer is swapped to display the new visual representation of the virtual robot.

The Virtual Robot Update Interrupt sets the timing system for redrawing the graphical model. During initialization of the graphical model, an initial timer is set for the first of this interrupt. Once called, it signals the spatial model to calculate the forward kinematics as specified in Sect. 6.11.2. Next, a Draw Screen Interrupt is called, which performed the tasks as specified in the previous paragraph. Finally, a timer is set for the next Virtual Robot Update. In the current implementation, the timer is set to 50 ms; however, this can be adjusted to suit the system resources of the system.

The Keyboard Interrupts are used for system termination and to control the camera angle that the robot is viewed from. Specifically, "Esc" terminates the program and "Space" toggles full screen/windowed mode. Character keys "a", "d", "s", "w", "r", and "f" linearly transformation the camera along the $+x$, $-x$,$+y$, $-y$,$+z$, and $-z$-axes, respectively. Arrow keys "left", "right", "up", and "down" rotate the camera $-yaw$, $+yaw$, $-pitch$, and $+pitch$, respectively. Finally, the "home" key resets the camera to its initial pose.

Visualization System (OpenGL) Initialization, Interrupts, and Subsystem Program Flow

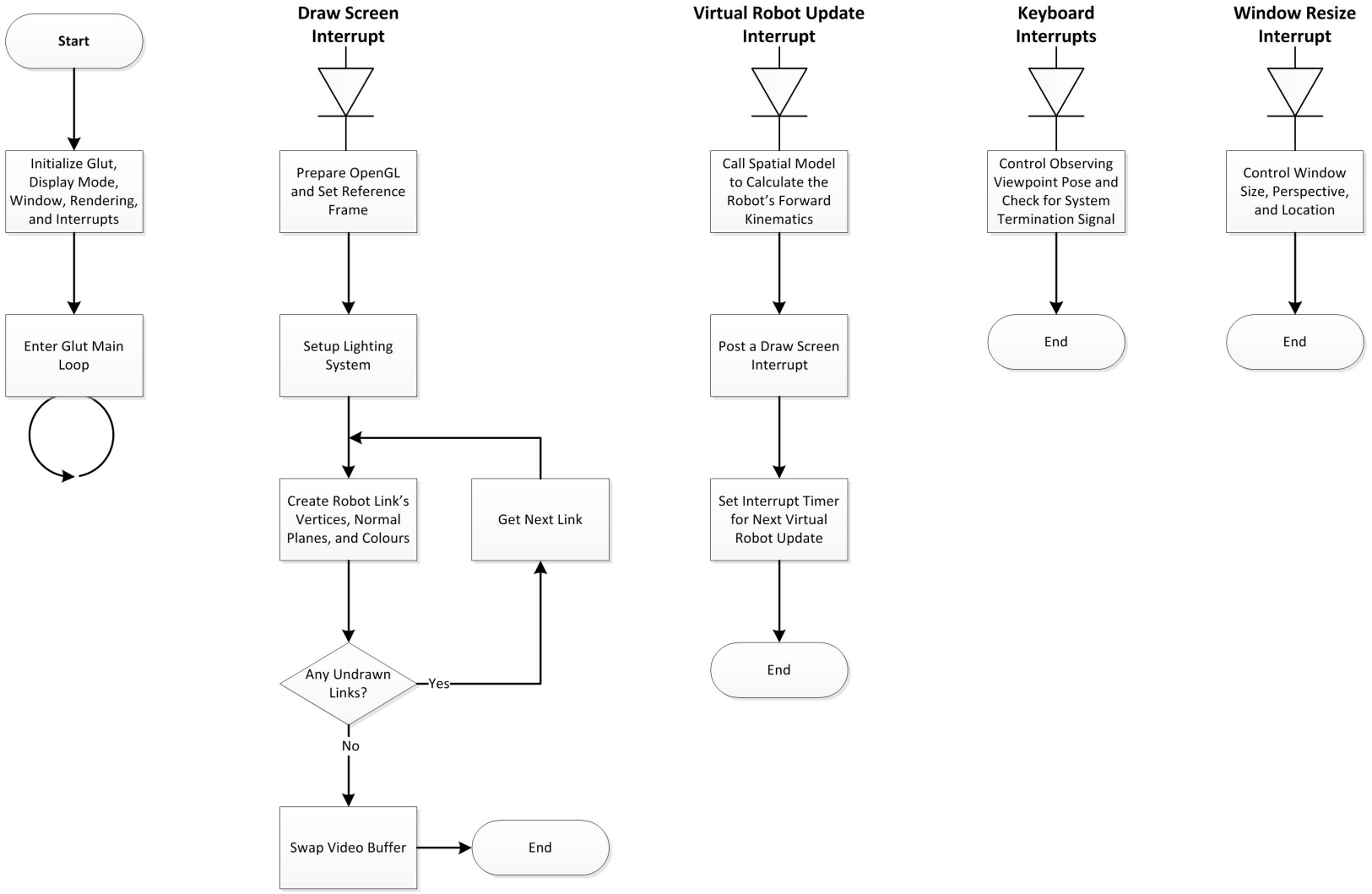


Figure 6.16. Program flow of the graphical model

The Window Resize Interrupt handles changes when the user modifies the window with the mouse. Specifically, it handles windows size, perspective, and location.

The main loop and the interrupts of the graphical model are illustrated in Figure 6.16.

Chapter 7 – Calibration of the Wiimote IR Camera and Accelerometers

Sect. 7.1 provides an overview of the calibration process, Sect. 7.2 details the fabrication of a calibration board used to calibrate the wiimote's IR camera, Sect. 7.3 details the communication between the PC, the Arduino microcontroller, and the calibration board, Sect. 7.4 details the methodology used to calibrate the board, and finally, Sect. 7.5 details the calibration of the wiimote's accelerometers.

7.1 – Overview

Before 6-DOF absolute pose can be obtained, the wiimote camera requires calibration. Unfortunately, the raw image from the wiimote's camera is unavailable. Instead, the centroids of up to the four brightest IR objects that are brighter than a predefined threshold are provided. To simulate the salient aspects of a raw image, a novel method is used, where a camera calibration board is fabricated to sequentially illuminate the LEDs to create a virtual image. Using multiple virtual images, the camera can be calibrated using traditional methods.

The wiimote's triaxial accelerometer is considerably inaccurate. To increase the accuracy, the wiimote is calibrated by estimating six correction parameters, vis. three gains and three biases.

7.2 – Fabrication of the LED Camera Calibration Board

The LED camera calibration board is a $30.48\ \text{cm} \times 30.48\ \text{cm} \times 0.95\ \text{cm}$ ($12" \times 12" \times 3/8"$) rigid plastic board containing eighty IR LEDs in an evenly spaced grid. The LEDs are wired serially in groups of four to create twenty circuits, which are each attached to ground and an output pin on the Arduino microcontroller board. The output pins are controlled by software uploaded onto the Arduino board that can optionally receive commands from the PC via a USB serial connection.

Eighty Panasonic LN162S IR LEDs are used, which are chosen to allow wide-angled viewing of the calibration board by the wiimote IR camera from a wavelength that the IR camera is highly sensitive to. Specifically, LN162S is a $3\ \text{mm}$ (diameter) IR LED that emits monochromatic light at $\lambda_{\text{P}} = 950\ \text{nm}$ (typical), with a power output $P_{\text{O}} = 3.5\ \text{mW}$ (typical) spread over a wide angle (half-power angle

$\theta_{1/2} = \pm 80°$). It has a max rated continuous forward current $I_F = 50\ \text{mA}$ and a variable nonlinear forward voltage drop $V_F = 1.2\ \text{V}$ (typical). Furthermore, the dimensions of the LED are illustrated in Figure 7.1. For further specifications, see the product datasheets [154].

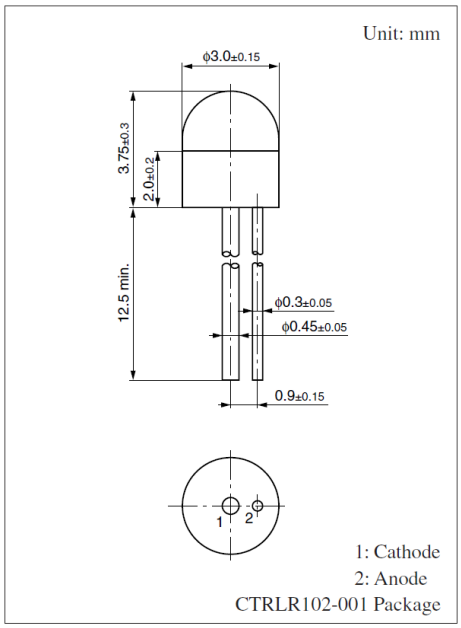


Figure 7.1. Dimension specifications of the Panasonic LN162S IR LED.
(Diagram obtained from the Panasonic LN162S Datasheet [154])

Fabrication is completed in seven steps:

Step 1. A precision CNC milling machine is used to drill one hundred slots for the LEDs; however, only eighty are used since the particular Arduino board used only has twenty output pins and only enough amperage to power four serially-connected LEDs per pin. Each slot is made up of a counterbore hole and a through-hole. The counterbore hole is milled 2 mm deep with a diameter to provide a press fit to the 3 mm diameter LEDs. The through-hole has a 2 mm diameter and is milled in the counterbore hole in a position offset slightly left of the centre to allow room for the LED's two leads shown in Figure 7.1. Each slot is precision positioned in a square grid with an adjacent LED separation of $2.54\ \text{cm}$ (1") from the centre of the counterbore holes.

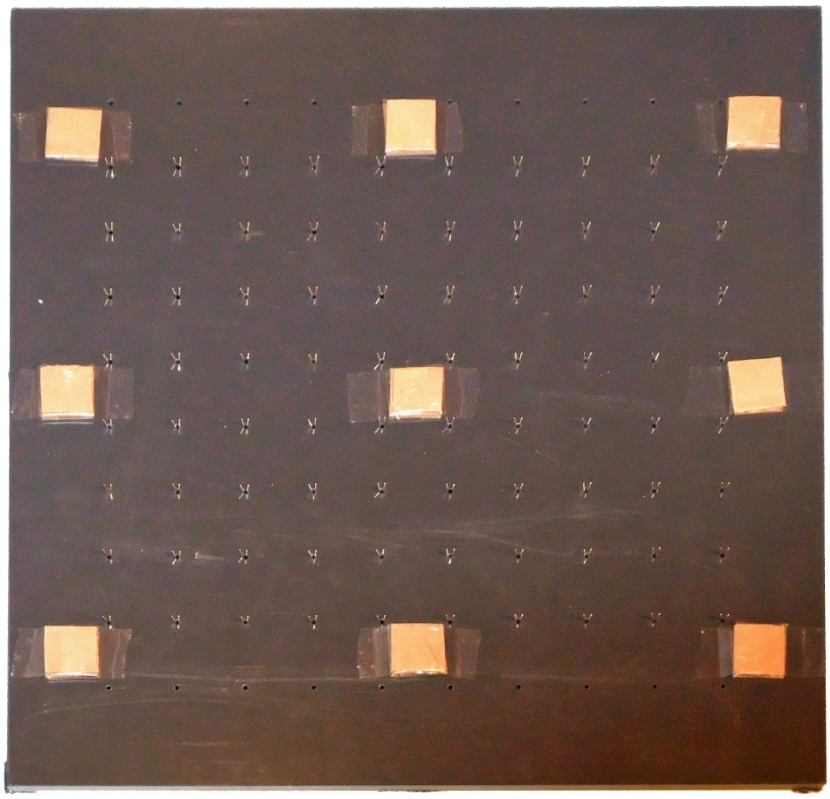


Figure 7.2. Calibration board with 80 LEDs inserted and 9 spacers attached.

Figure 7.3. Close-up of the LED leads coated with liquid electrical tape.

Step 2. Eighty LEDs' leads are inserted through the board's front surface into the slots. The leads are aligned horizontally with the anode on the left side to fit easily through the through-hole, and the top of the LED are pushed fully into the press-fit counterbore hole. The leads are approximately aligned with where the perfboards will be positioned, and nine cardboard spacers are taped to the back surface of the board. The back surface of the calibration board is illustrated up to this point in fabrication in Figure 7.2.

Step 3. To prevent the leads from touching, which would cause an LED(s) to malfunction and could potentially damage the Arduino board by drawing too much amperage from the output pins, an electrical insulator (Brush-On Electrical Tape) is painted onto the leads as shown in Figure 7.3.

Step 4. Four perfboards with copper pads are positioned on top of the spacers such that the leads pass through the appropriate holes in the perfboards as shown in Figure 7.4. This requires that the ends of the 160 leads be pre-aligned with the centre of the respective perfboards' holes to within a tolerance of $\pm 0.3\ \mathrm{mm}$ prior to positioning the perfboards.

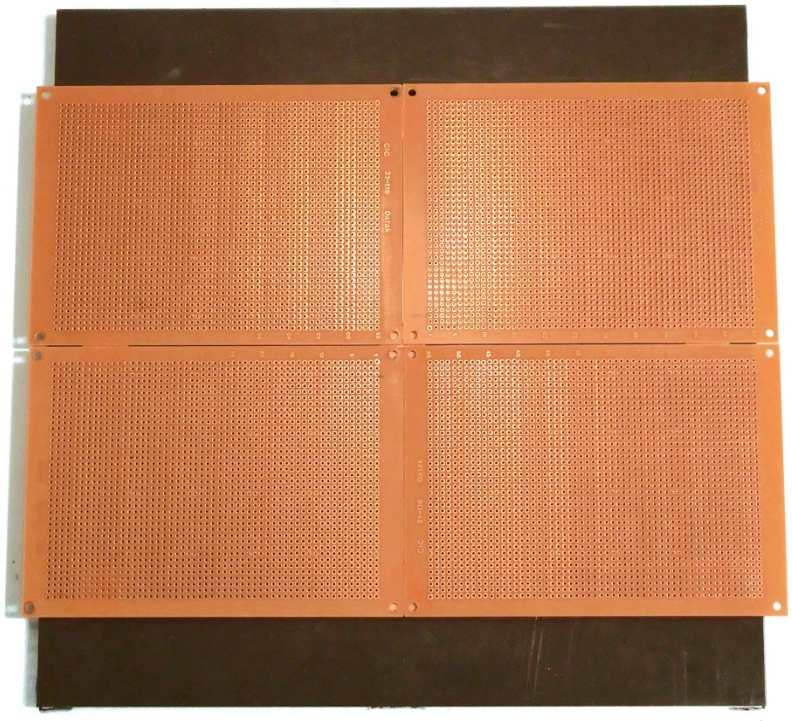


Figure 7.4. Four perfboards with copper pads are positioned on top of the spaces and aligned such that the LED leads are inserted through the appropriate holes in the perfboards.

Step 5. The groups of four LEDs are soldered in series creating twenty circuits as shown in Figure 7.5. A close-up view of this is shown in Figure 7.6.

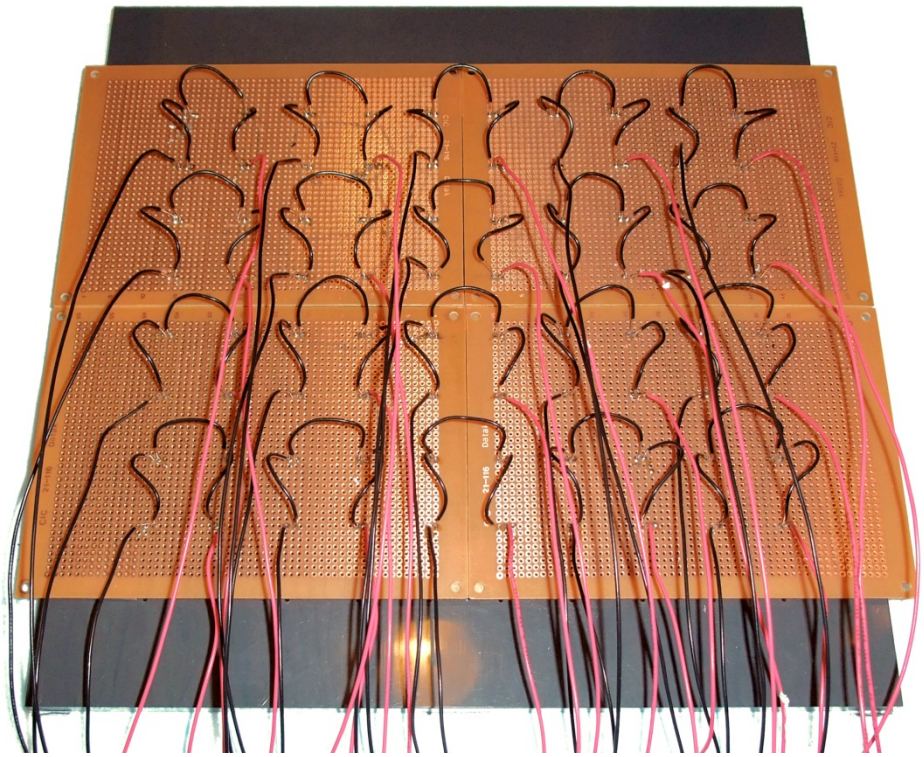


Figure 7.5. LED leads are soldered in series into groups of 4, creating 20 circuits.

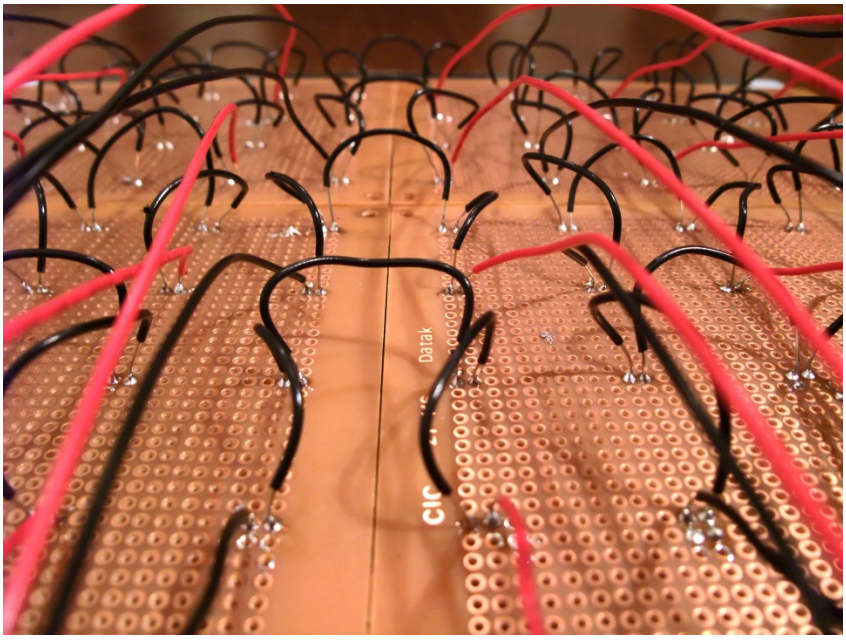


Figure 7.6. Close-up of the circuits.

Step 6. The anode of the first LED in each series is connected to an Arduino output pin and the cathode of the last LED in each series is connected to the Arduino's ground pin as shown in Figure 7.7. E.g., the cathode of the last LED in Circuit 0 is connected to Pin 0 and Circuit 13 is connected to Pin 13. Analog pins 0-5 are set to digital mode and are referred to as digital pin 14-19. Under normal circumstances, it is important to put a resister in series with the LEDs to protect both the LEDs and the Arduino board; however, the internal resistance of the Arduino circuits at the pins ranged from 40.5 Ω to 42.0 Ω (avg. 40.95 Ω) [9], which is a resistance higher than required to protect a circuit containing four LN162S IR LEDs.

Note that the circuits attached to Pin 0 and Pin 1 must be disconnected (Figure 7.8) if the PC is used to control the Arduino board since these pins are also used for serial communication by the microcontroller. Keeping these two circuits connected interferes with data transmission during operation; however, they may be connected if control is not required by the PC during operation.

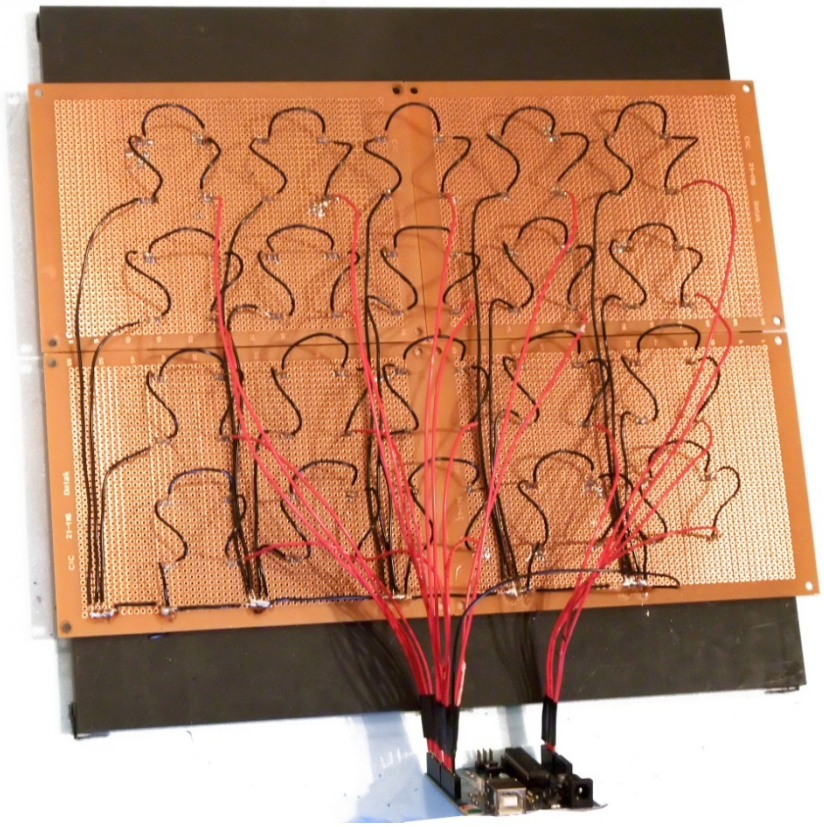


Figure 7.7. Calibration board with grounds soldered and connected to the Arduino microcontroller.

[9] The internal resistance R_{int} was calculated using $V = I \times (R_{ext} + R_{int})$, where V is the measured voltage drop, I is the current, and R_{ext} is the measured resistance of a "safety" resister, which was used to limit excessive current. The test was done multiple times on different pins and with multiple safety resistors ($239\ \Omega, 328\ \Omega,$ and $669\ \Omega$).

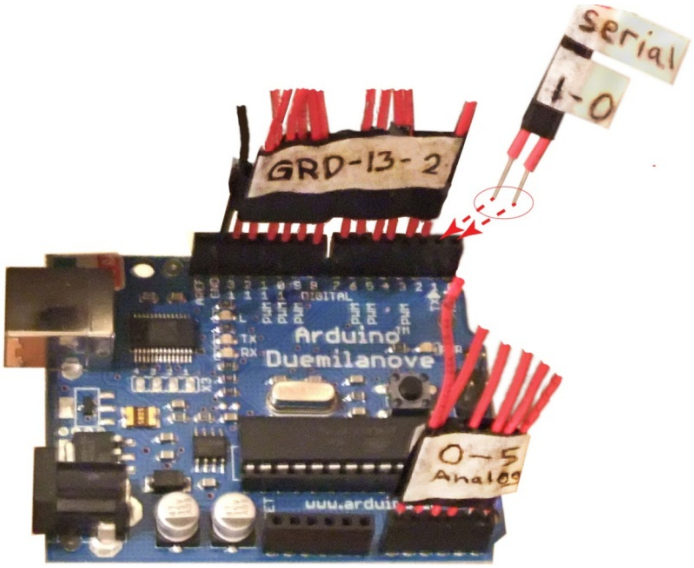


Figure 7.8. The connection from the LED circuits to the Arduino microcontroller.

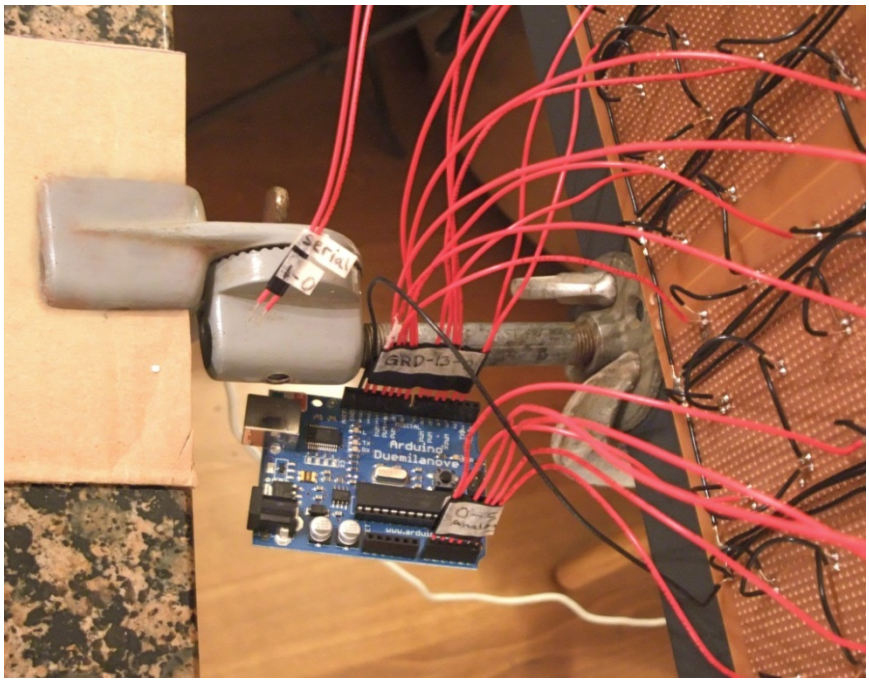


Figure 7.9. Calibration board mounted and connected to the computer (Back View).

Step 7. The board is attached to a surface with an adjustable metal bracket/clamp as shown in Figure 7.9, and the Arduino board is connected to the PC using a USB A to B cable.

The front view of the completed calibration board is show in Figure 7.10.

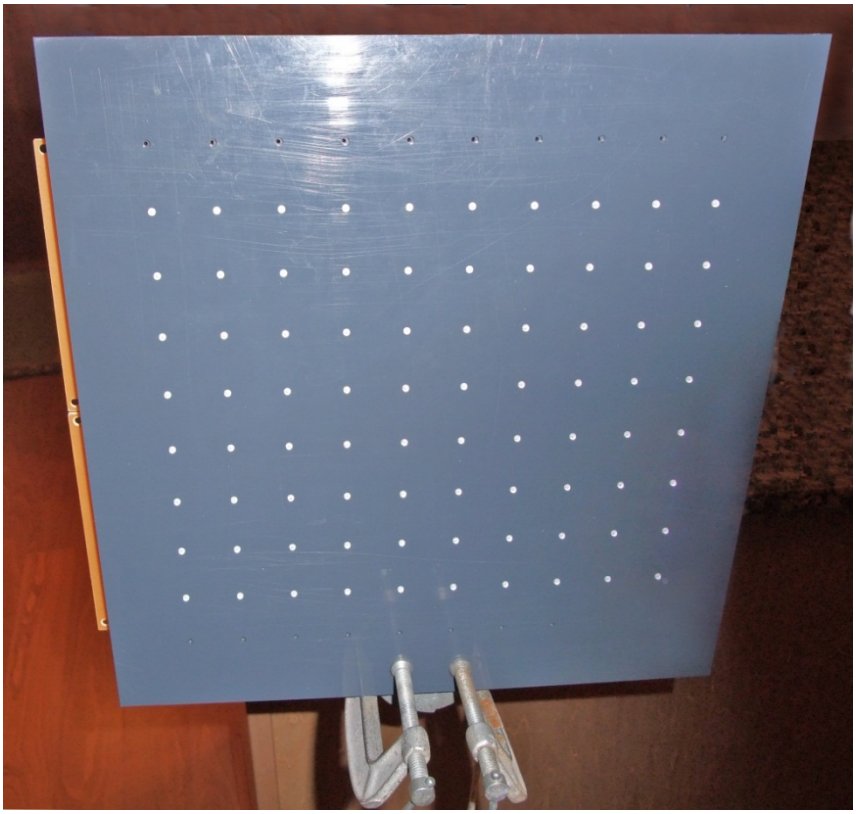


Figure 7.10. Calibration board completed and mounted (Front View).

7.3 – Communication

7.3.1 – PC to Arduino Microcontroller Board USB Communication

The PC and Arduino microcontroller board communicate to each other via a serial connection, which for the Arduino Duemilanove is physically implemented through a USB (A-B) connection. The PC's OS recognizes Arduino's communication as a virtual com port, therefore, before any communications can be established, the com port of the PC's respective USB port must be identified. In Windows, it will be COM*x* and can be found by examining the Windows Device Manager or in Linux, it likely is identified as /dev/ttyUSB*x*, where *x* is the com port number.

The primary purpose of the serial connection is to upload the code to the Arduino board. Using the Arduino IDE, sketches (Arduino programs) are written, compiled, and uploaded from the PC to the Arduino board via the serial com channel. Upon detecting an incoming sketch, the Arduino board resets itself, activates the on-board bootloader, the bootloader writes the incoming program to memory, and the bootloader executes a jump to the address of the program's first instruction, which causes the program to be executed on the Arduino board. As long as no further serial communications are required between the operating Arduino board and the PC, the USB cable can be disconnected. However, the USB provides operating power to the Arduino board, thus if it is disconnected, an external DC power supply (6 to 20 V) must be connected to the *Gnd* and *Vin* pins on the board to provide power.

The secondary purpose of the serial connection is to provide real-time communication between the PC and the operating Arduino program. This is required to build a "smart" beacon. The conventional beacon only contains four LEDs, which are always on during operation and fixed in location. In contrast, the smart beacon is a virtual implementation of the conventional beacon on the calibration board that moves to stay in the field of view of the wiimote's IR camera. Initially, all LEDs on the calibration board are turned off except for four LED in a 1" by 1" square, which acts as a conventional beacon. When the wiimote is moved such that the LEDs are nearly out of focus of the IR camera, the four active LEDs are quickly shut off and four inactive LEDs that would be in a better field of view of the camera are activated. The PC then accounts for the new location of the active LEDs by transforming the location of the reference LED points wrt the Beacon Frame (for homography calculations) to the new location of the active LEDs. This allows absolute pose to be obtained from a greater field of view.

The commands to activate and deactivate the LEDs on the calibration board are performed by compiled C code on the PC that communicates over a 9600 baud serial connection (one stop bit, no parity) to the Arduino board, where the microcontroller turns on (sets voltage high) and off (sets voltage low) the appropriate pins controlling the calibration board's LED circuits (q.v. Sect. 7.3.2). To allow the PC to operate the smart beacon via serial communication, LED circuits 0 and 1 must be disconnected from pins 0 and 1 as illustrated in Figure 7.8. Leaving circuits 0 and 1 attached will interfere with communication, since the microcontroller uses pins 0 and 1 for serial communication.

7.3.2 – Arduino Microcontroller to Calibration Board Connections

The circuits of the calibration board are wired to the ground pin and the twenty output pins of the Arduino board. The software uploaded to the Arduino board directly controls the output pins and can be set to a high state ($+5$ V) or a low state (0 V). Setting to a high state activates the four LEDs in the respective circuit and setting the pin to a low state deactivates the four LEDs in the respective circuit.

7.4 – Calibration of the Wiimote IR Camera

7.4.1 – Overview

Before the wiimote's IR camera can be used to obtain absolute pose information, the camera's intrinsic parameters must be identified. These include the focal lengths, the principal point, and preferably the distortion parameters. It is common practice in camera calibration methods to take multiple images of a very flat checkerboard, and the intersection of the squares are identified and used as feature points by the calibration process. However, the raw images from the wiimote's camera are not available. Rather, the images are processed by an on-board processor to obtain the centroids of up to the brightest four points that are brighter than a predetermined threshold. The (x, y) locations of these points are the output of the wiimote.

While holding the calibration board and the wiimote stable, a microcontroller is used to sequentially flash the LEDs on and off. By merging multiple images of this process, a virtual image of the LEDs is created. Furthermore, by repeating this multiple times, many virtual images are created, which can be used with the known position measurements of the calibration board's LEDs to calibrate the camera. Using this novel method to calibrate the wiimote's camera, the intrinsic parameters are obtained, which allows the pose of the wiimote to be known in space wrt a set of four illuminated LEDs during teleoperation.

7.4.2 – Calibration Board Setup

The calibration board was set up prior to calibration, as detailed in this subsection.

Step 1. To begin, the calibration board was mounted on a stable surface using the adjustable bracket as illustrated in Figure 7.9 and Figure 7.10. This allowed the calibration board to be held rigidly in adjustable poses.

Step 2. The calibration board's circuits were inserted into the appropriate pins of the Arduino board as discussed in Step 6 of Sect. 7.2.

Step 3. The Arduino board was connected to the PC with a USB connection.

Step 4. With the calibration board in place, the wiimote was placed on a stable surface such that the applicable LEDs were within the field of view of the camera.

The wiimote has a limited field of view and a low resolution, which prevent the entire calibration board from being visible by the camera at various wide-angles and at a distance close enough to identify all eighty LEDs accurately. Therefore, only 48 LEDs (8 x 6) were used, which was sufficiently low that all LEDs can be identified in each frame, yet high enough to allow accurate calibration.

Step 5. A sketch was written, compiled, and uploaded to the Arduino board. The sketch activated and deactivated the groups of four LEDs in a rotating sequence, which was performed using twelve output pins. Specifically, the sketch began by setting all pins to a low state except for the first pin in the sequence, which was set to high. Following a 250 ms delay, the first pin was set to a low state. After a 30 ms delay, the next pin in the series was set to high and the process was repeated for the remaining eleven pins, after which, the cycle was repeated at the first pin. This resulted in each circuit (four LEDs) being activated one at a time for 250 ms, with a 30 ms transition in between when no LEDs were activated. When executed, this 30 ms transition provided ample time for the on and off transiences of the LEDs with remaining time to distinguish from the next set of LED positions in the IR camera's output data.

Step 6. The wiimote was connected to the PC via Bluetooth. If the wiimote device has not previously been paired to the PC then this must be completed first. The procedure is discussed in Sect. 6.2.3.

7.4.3 – Build Data Frames

The camera locations (x, y) for the four-eight LEDs were recorded for all twenty-three data frames, as detailed in this subsection. A data frame refers to the collection of all LED locations at each camera measurement for at least one complete cycle of the forty-eight LEDs. As mentioned, twenty-three of the data frames were used.

Step 7. The camera was tested to ensure that all forty-eight LEDs were visible. This was accomplished by running the Wiimote to Robot Interface in test mode, which caused the LED locations to be

outputted to the screen and prevented a connection to Robot Control Server. As the Arduino board activated each circuit, it was verified that the four active LEDs were present on the screen output of the Wiimote to Robot Interface. If not all LEDs were visible, the wiimote or the calibration board was adjusted until they were. Upon verifying that all active LEDs were visible, the Wiimote to Robot Interface was terminated.

Step 8. The LED locations were recorded. Specifically, the Wiimote to Robot Interface was run in record mode, which caused the program to record the forty-eight LED locations to the hard drive. Each time this step was performed, one new output file was created.

Step 9. When less than twenty-three files had been created, a new pose of the calibration board was set by adjusting the calibration board's mounting bracket, and then the process continued at Step 7. Once twenty-three files were created, the process continued at Step 10.

7.4.4 – Build Unordered Image Frames by Processing Data Frames

The raw-output wiimote-measurements in the data frames are not useful until they are processed into ordered image frames used by camera calibration. This subsection details the intermediate process of creating unordered image frames out of data frames, which are later used to build ordered image frames.

Step 10. The first data-frame file was opened and one complete cycle of camera's measurements for the forty-eight LEDs were identified.

Step 11. Since multiple measurements were taken during the 250 ms period while a set of four LEDs were active, there were multiple recorded values grouped together for the each set of LEDs. The estimated location of an LED was found by taking the average recorded value of the LED in the group (excluding the first and last recording since their locations can be heavily corrupted by camera sensor noise while the LEDs are in the process of activating and deactivating). Averaging improves accuracy by smoothing out the fluctuations of reported LED locations caused by noise in the camera sensor.

Step 12. The average LED locations for each data frame were stored and are referred to as unordered image frames. There is one image frame for each data frame.

Step 13. Step 10 was repeated until all data frames had been transformed into unordered image frames. Once all twenty-three unordered image frames were created, the process continues at Step 14.

Image frames are not images in the raster sense, but rather, are a collection of forty-eight LED data points, which are used for calibration. Furthermore, in some sense they are pseudo images, because they are a collection of multiple overlapping images to create a virtual image, which simulates a processed version of what would have been recorded if raw unprocessed images were obtained directly from the IR camera. Unordered image frames are equivalent to the data typically obtained by finding the corners of a checkerboard prior to image correspondence, which is a common practice in the calibration of standard cameras. Data points from four of the twenty-three image frames are illustrated in Figure 7.11.

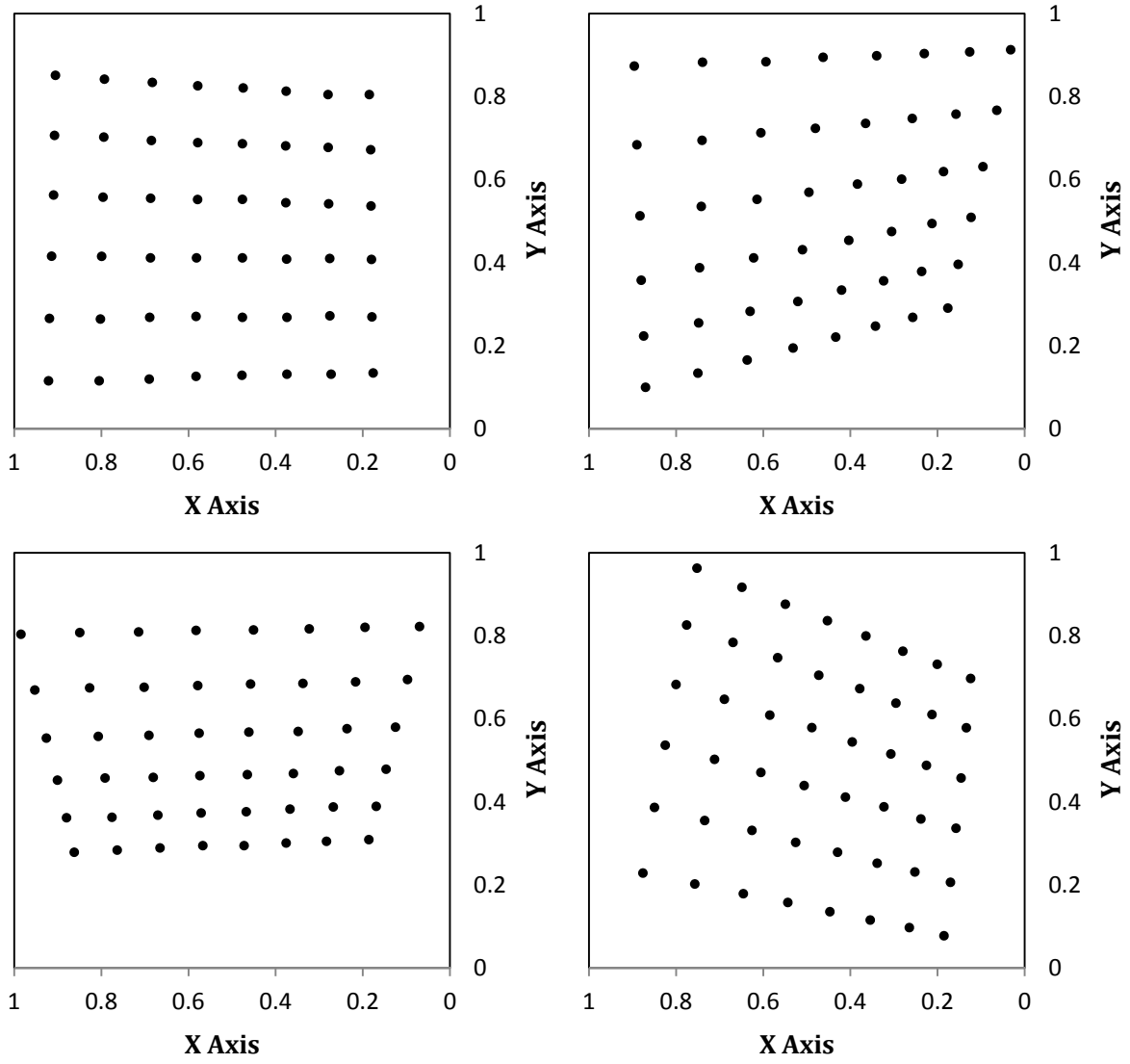


Figure 7.11. Four of the twenty-three calibration images

7.4.5 – Order Image-Frame Points by Finding the Correspondence

For the image data to be used in calibration, it is required that the points in each frame be placed in the same order. This is accomplished through correspondence, and then reordering based on the identified correspondence.

Step 14. Correspondence between respective points in the twenty-three unordered images was found. This is a non-trivial task to automate, and was thus done manually. Since it is difficult to identify the LED point correspondence by manually observing raw (x, y) point data, software was built to output the location of each point to the screen. Each point was then manually selected with the mouse in a set order for each unordered image to determine correspondence.

Step 15. The point data for each unordered image was automatically ordered based on the identified correspondence by the developed software to obtain twenty-three *ordered* image frames.

7.4.6 – Calibrate the camera

The ordered image frames and respective reference points were prepared and used for calibration, which enabled the intrinsic camera parameters to be obtained.

Step 16. The image-points calibration vector was created.

a) Each image point was stored in an OpenCV *Point2f(x, y)* object.

b) The forty-eight Point2f objects for each image frame were inserted into an image vector.

c) The twenty-three image vectors were inserted into an image-points calibration vector. The image-points calibration vector is of the C++ form: *vector<vector<Point2f>>*.

Step 17. The reference-points calibration vector was created.

a) Twenty-three OpenCV *Point3f(x, y, z)* reference points were added to a reference vector, where each point is the known location of the LED point in the real world (in cm) wrt a reference frame that has its origin at the centre of the bottom-right (facing the matrix) LED. Furthermore, the horizontal *x*-axis is in the direction of and collinear to the bottom row of LEDs, and the vertical *y*-axis is in the direction of and collinear to the far-right column of LEDs. The points are specifically defined as, $Point3f\,(2.54i, 2.54j, 0.0),\ \ i \in \{0,1,\dots,5\},\ j \in \{0,1,\dots,6\}$

b) The reference vector was added to the reference-points calibration vector twenty-three times, since OpenCV requires a separate reference vector for each image vector. The reference-points calibration vector is of the C++ form: *vector<vector<Point3f>>.*

Step 18. The camera was calibrated using OpenCV's *calibrateCamera(…)* function using the image-points calibration vector and the reference-points calibration vector. Since this is a nonlinear operation, to improve the estimate of the calibration, the *calibrateCamera(…)* function was run multiple times while holding a few parameters constant each time. First the extrinsic parameters $\mathbf{A}$, $\mathbf{t}$ were estimated, then the extrinsic parameters and the principal point c_x, c_y, followed by the previous parameter and the focal length f_x, f_y, and finally, all the previous parameters $\mathbf{A}$,$\mathbf{t}$, c_x, c_y, f_x, f_y with the distortion parameters k_1, k_2, k_3, p_1, p_2.

The intrinsic parameters were obtained by camera calibration, and are presented here to eight significant figures:

$$f_x = 1348.6927, \qquad f_y = 1342.3807, \qquad c_x = 496.88116, \qquad c_y = 381.90093$$

$$k_1 = 0.073683679, \qquad k_2 = -0.25457907, \qquad k_3 = -0.23550061$$

$$p_1 = -6.8723600, \qquad p_2 = 0.0019153288$$

These obtained parameters are used to solve the homography to obtain the pose of the beacon wrt the wiimote during teleoperation as discussed in Sect. 6.7.5.

7.5 – Accelerometer Calibration

Using accelerometers that are precise and highly accurate can greatly improve performance. Unfortunately, the ADXL330 triaxial accelerometer used in the wiimote is neither precise nor accurate. At rest, the vector sum of the three axes' measurements should be equal to one; however, for the ADXL330 as implemented in the wiimote, this is far from the case. These values may be off by up to 10%. This is a significant error, and therefore calibration is useful.

The wiimote was calibrated using the method detailed in [157]. This calibration model uses an iterative method to find a gain G_i and a bias B_i for each axis i that minimizes the error between the

model and the gravity vector. The six unknowns can be calculated while the accelerometer is at rest using six different tilt angles and the gravity vector. Knowledge of the true tilt angles is not required.

Measurements of six tilt angles were obtained by placing the wiimote on each of its six sides (a level surface is not required) and 500 measurements were taken per side. The average of the 500 measurements was used as a measured tilt angle. A Python implementation of Won and Golnaraghi's method [157] was programmed and the six tilt angles were used to compute the gain G_i and bias B_i for each axis. Finally, the accelerometer was calibrated by using the three computed gains and biases to adjust the accelerometer measurements obtained during teleoperation by the Wiimote to Robot Interface.

Chapter 8 – Experimental System Verification and Testing

8.1 – Summary

Testing of the system was performed to verify that the design was capable of teleoperation using 6-DOF motion capture for use in robotics applications, or more generally, as a 6-DOF computer input-device. The system was tested in four phases. In the first and second phases, the position filter and attitude filter, respectively, were quantitatively tested in isolation from the rest of the system with simulated data. Third, the combined pose filter was quantitatively tested (not isolated) using real data from the wiimote. Finally, the whole system was qualitatively tested to determine if the pose of the wiimote can be correctly interpreted by the motion capture system and used to control a robot manipulator via teleoperation.

The four phases of testing are detailed in the following four sections, and are each subdivided into methodology, results, and discussion subsections. Furthermore, the position filter and attitude filter were tested with simulated data and thus contain an additional subsection detailing the model used to produce the simulated data.

8.2 – Position-Filter Testing with Simulated Data

The position filter was tested in isolation to verify that it can successfully converge to provide optimal estimation of position, velocity, and acceleration.

8.2.1 – Simulated-Data Model

This subsection defines the motion model used to generate the simulated dynamics and the motion model's initial conditions.

8.2.1.1 – Motion Model

To test the position filter, simulated-state data were used, with a state space as defined by

$$\mathbf{x}_s(t) \triangleq \begin{bmatrix} x(t) \\ \dot{x}(t) \\ \ddot{x}(t) \end{bmatrix} \tag{8.1}$$

where x is the position, $\dot{x}$ is the velocity, and $\ddot{x}$ is the acceleration at time t. Assuming a constant acceleration, a system of equations that models the dynamics of motion can be obtained by

$$x(t) = x_0 + \dot{x}_0 t + 0.5\ddot{x}_0 \mathrm{t}^2 \tag{8.2}$$

$$\dot{x}(t) = \dot{x}_0 + \ddot{x}_0 \mathrm{t} \tag{8.3}$$

$$\ddot{x}(t) = \ddot{x}_0 \tag{8.4}$$

where x_0 is the initial position, $\dot{x}_0$ is the initial velocity, and $\ddot{x}_0$ is the initial velocity.

It is unrealistic to sample the position filter's position and acceleration measurements directly from Eq. (8.2) and (8.4) since the wiimote's sensors corrupt the true state with random noise. To obtain more realistic measurements, the measurement matrix $\mathbf{z}_k$ is obtained at each time step k by adding normally-distributed white noise to the simulated state data. Formally,

$$\mathbf{z}_k = \begin{bmatrix} x(t_k) \\ \ddot{x}(t_k) \end{bmatrix} + \begin{bmatrix} \mathcal{N}(0, \sigma_p, t_k) \\ \mathcal{N}(0, \sigma_a, t_k) \end{bmatrix} \tag{8.5}$$

where $\mathcal{N}(0, \sigma_p, t_k)$ is the normally-distributed position-noise with a zero mean and a standard deviation of σ_p, and $\mathcal{N}(0, \sigma_a, t_k)$ is the normally-distributed acceleration-noise with zero mean and a standard deviation of σ_a, both of which are produced by a normally-distributed pseudo-random number generator.

8.2.1.2 – Initial Conditions

The simulated-state initial values were arbitrarily set to reasonable values of

$$x_0 = 10.0\ cm,\ \ \dot{x}_0 = 2.0\ \ cm/s,\ \text{and}\ \ \ddot{x}_0 = -0.6\ \ cm/s^2$$

Furthermore, the simulated state data were discretely generated for $0 \leq t < 8\ s$ at a time step interval $\Delta t = 0.01$.

To obtain reasonably realistic convergence rates, the noise random variables were set to values that were found to produce good results in teleoperation of real-world data as discussed in Sect. 6.8.3.1. Specifically, the position-noise standard deviation was set to $\sigma_p = 0.1$, the acceleration-noise standard

deviation was set to a high value of $\sigma_a = 100$, and finally the jerk-noise standard deviation was set to $\sigma_r = 0.45$.

8.2.2 – Methodology

The position filter was implemented as described in Sect. 5.2.3 and Sect. 5.2.4, and was run for the interval $0 \leq t < 8\ s$. At each discrete time interval Δt, the simulated (true-) state $\mathbf{x}_\mathrm{s}$ was calculated using Eq. (8.2), (8.3), (8.4), and a measurement $\mathbf{z}_k$ was taken using Eq. (8.5). Furthermore, the initial conditions were set as detailed in Sect. 8.2.1.2.

The position filter was passed through one iteration of prediction and correction, and the time t_k, simulated true-state $\mathbf{x}_\mathrm{s}$, estimated a posteriori state $\hat{\mathbf{x}}_k^+$, and the a posteriori covariance $\mathbf{P}_k^+$ were recorded. The recording of these values was repeated at each iteration.

The position-filter error vector $\boldsymbol{\varepsilon}_k^\mathbf{x}$ between the estimated state $\hat{\mathbf{x}}_k^+$ and the simulated true-state $\mathbf{x}_\mathrm{s}$was obtained at each time step k using

$$\boldsymbol{\varepsilon}_k^\mathbf{x} = [\mathbf{x}_\mathrm{s}(t_k)] - [\hat{\mathbf{x}}_k^+] \tag{8.6}$$

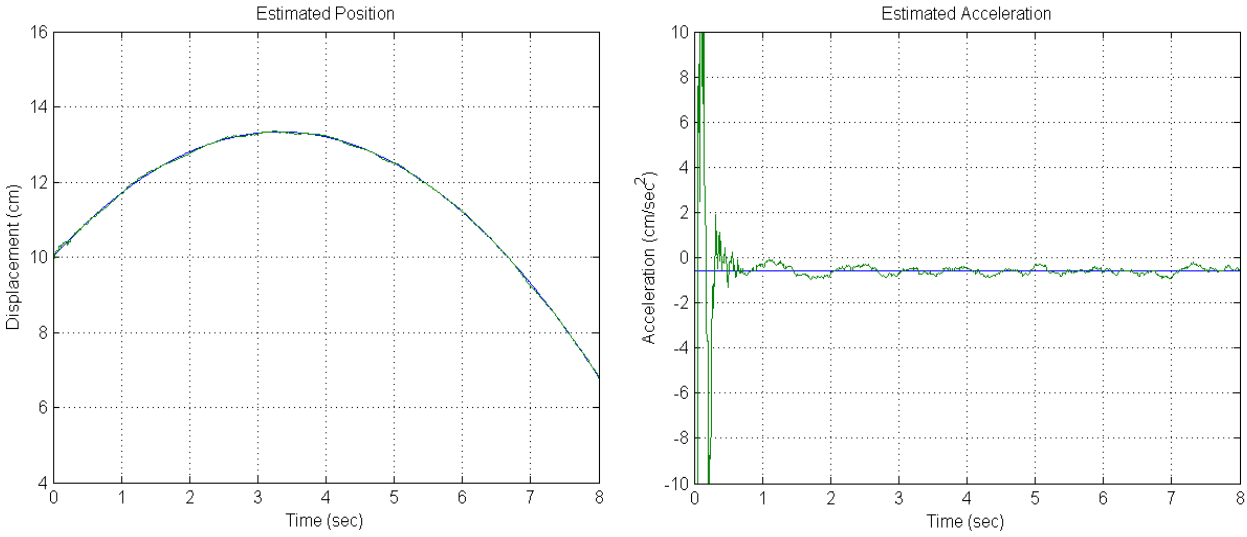
Finally, the theoretical one-σ error bounds were calculated by

$$\boldsymbol{\epsilon}_k^\mathbf{x} = \left[\pm\sqrt{\mathrm{P}_k^+(1,1)} \quad \pm\sqrt{\mathrm{P}_k^+(2,2)} \quad \pm\sqrt{\mathrm{P}_k^+(3,3)}\right]^\mathrm{T} \tag{8.7}$$

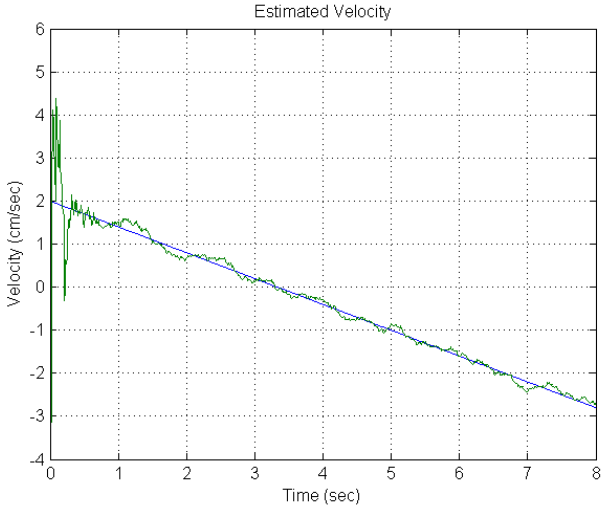
where $\mathbf{P}_k^+(i,j)$ is the element of the a posteriori covariance at row i and column j. The error bounds are the pseudo boundaries that at least one-standard-deviation (~68.3%) of error should be within. For the filter to be functioning correctly, the error boundaries should converge and bound the majority of the error.

8.2.3 – Results

The position filter's estimated state $\hat{\mathbf{x}}_k^+$ (green) and the simulated true-state $\mathbf{x}_\mathrm{s}$(blue) are plotted in Figure 8.1 to allow a direct comparison. The position-filter error vector $\boldsymbol{\varepsilon}_k^\mathbf{x}$ (blue) and its theoretical one-σ error bounds $\boldsymbol{\epsilon}_k^\mathbf{x}$ (green and red) are plotted in Figure 8.2. At steady state, the error bounds had converged to $\boldsymbol{\epsilon}_k^\mathbf{x} = [0.0262\ cm \quad 0.115\ cm/s \quad 0.336\ cm/s^2]^\mathrm{T}$.

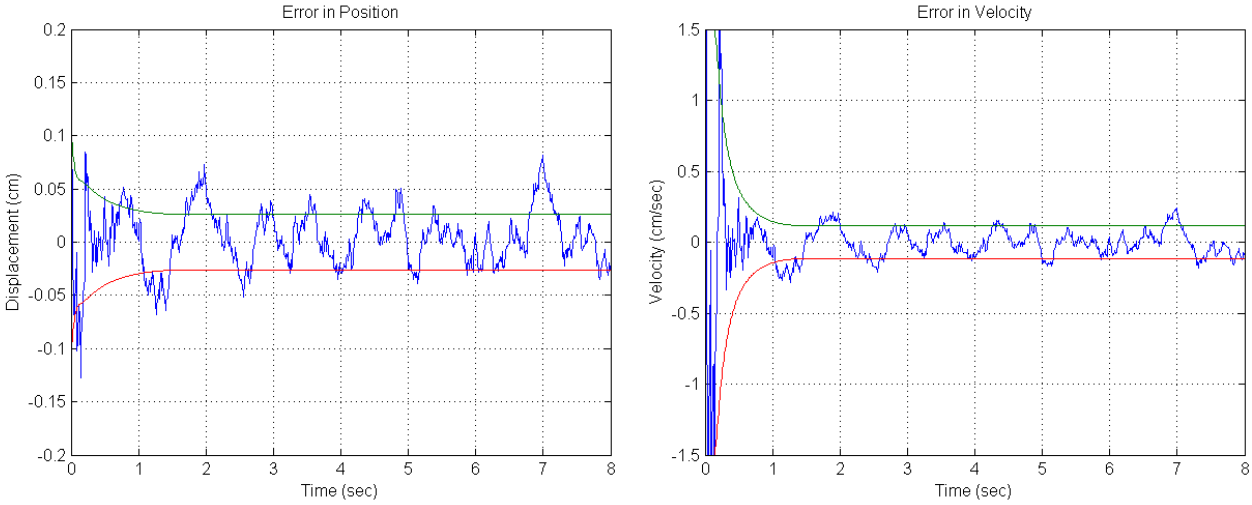


(a) (b)

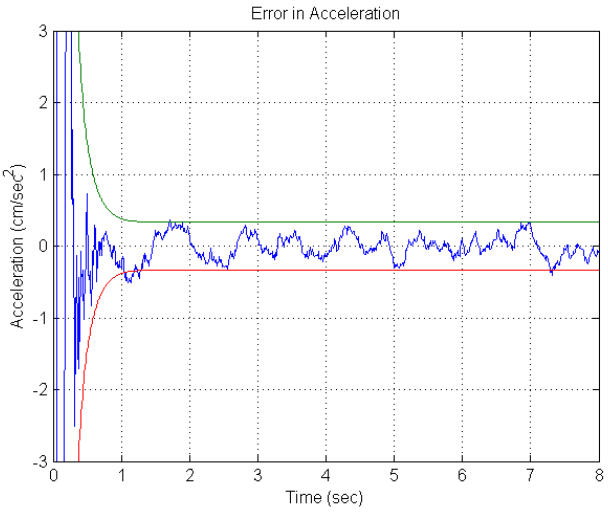


(c)

Figure 8.1. Estimated state by position filter (green) and simulated true value of state (blue). (a) Estimated position, (b) estimated velocity, and (c) estimated acceleration.



(a) (b)



(c)

Figure 8.2. Error in state estimation (blue) and the theoretical one-σ error bounds (green & red) by the position filter. (a) Error in position, (b) error in velocity, and (c) error in acceleration.

8.2.4 – Discussion

The primary purpose of the position filter was to accurately estimate position; however, velocity and acceleration are also estimated to increase the predictive capability of the filter. The nearly overlapping green and blue curves in Figure 8.1.a demonstrate that the position filter's estimated position (green) is extremely close to the true position (blue). Examining Figure 8.1.b and Figure 8.1.c, after a brief transient period, the estimated velocity and acceleration were also close to the true velocity and acceleration. Figure 8.1 suggests that the position filter was functioning correctly; however, a further error analysis was required.

A detailed error analysis is presented in Figure 8.2. The blue line is the error in the estimated state, and green and red curves are the positive and negative theoretical error bounds, respectively. A properly designed and functioning Kalman filter that is estimating observable states will have theoretical error bounds that converge until reaching a steady-state neighborhood. Furthermore, 68.3% of the error in estimated state should be within the error bounds.

Observing Figure 8.2.a-c, the theoretical error bounds converge and the majority of the state error is within the error bounds, demonstrating that the position filter was functioning properly. Additionally, despite a very high error in the accelerometer measurements, the error bounds converge to produce a high degree of accuracy. Specifically, shortly after one second, the error bounds converge to $\pm 0.0262\ cm$ for position, $\pm 0.115\ cm/s$ for velocity, and $\pm 0.336\ cm/s^2$ for acceleration.

8.3 – Attitude-Filter Testing with Simulated Data

The attitude filter was tested in isolation with simulated data to verify that it can successfully converge to provide optimal estimation of the attitude and the three gyroscope biases.

The attitude filter implementation uses radians to describe the attitude when in rotation vector form and rad/s for the angular rate. However, to improve clarity, all values in this section are converted to degrees, but it should be made explicitly clear that in the implemented system all values were in radians.

8.3.1 – Simulated-Data Model

8.3.1.1 – Defining the Model

Before simulated data can be used as measurements, the data must be modeled such that they are sufficiently close to real-world data. The evolution of the simulated true-attitude $\bar{q}_k$ at time step k can be expressed as

$$\bar{q}_k = \bar{q}(\boldsymbol{\theta_k}) \otimes \bar{q}_{k-1} \tag{8.8}$$

where $\bar{q}(\boldsymbol{\theta_k})$ is the change in attitude between time steps, and $\boldsymbol{\theta_k}$ is the rotation vector, which is obtained by

$$\boldsymbol{\theta_k} = \boldsymbol{\omega}_k \Delta t \tag{8.9}$$

where $\boldsymbol{\omega}_k$ is the angular rate that is assumed to be constant over the time interval $\Delta t = t_k - t_{k-1}$.

The wiimote's gyroscope measures angular rates, but do so inaccurately by adding an initially unknown drifting bias and random noise to the measurement. Based on the gyro model described in Sect. 4.4 and formally expressed in Eq. (4.60), simulated gyro measurements can be produced using

$$\boldsymbol{g}_{s,k} = \boldsymbol{\omega}_k + \mathbf{b}_k + \begin{bmatrix} \mathcal{N}(0, \sigma_{\boldsymbol{\omega},2}, t_k) \\ \mathcal{N}(0, \sigma_{\boldsymbol{\omega},2}, t_k) \\ \mathcal{N}(0, \sigma_{\boldsymbol{\omega},1}, t_k) \end{bmatrix} \tag{8.10}$$

where $\mathbf{b}_k$ is the drifting gyro bias vector, and $\mathcal{N}(0, \sigma_{\boldsymbol{\omega}}, t_k)$ is normally-distributed gyro-noise with a zero mean and a standard deviation of $\sigma_{\boldsymbol{\omega}}$, which is produced by a normally-distributed pseudo-random number generator. Furthermore, to model the gyroscope more accurately, the bias is allowed to drift as described by

$$\mathbf{b}_k = \mathbf{b}_{k-1} + \mathbf{b}_\delta \Delta t \tag{8.11}$$

where $\mathbf{b}_\delta$ is the bias drift rate.

The simulated camera-attitude-measurement is produced with a simple model

$$\bar{q}_k^{\mathrm{c}} = \delta\bar{q}(\boldsymbol{\theta}_{\mathcal{N}}) \otimes \bar{q}_k \tag{8.12}$$

where $\delta\bar{q}(\boldsymbol{\omega}_{\mathcal{N}})$ is error quaternion and $\boldsymbol{\theta}_{\mathcal{N}}$ is the rotation-vector error, which is obtained by

$$\boldsymbol{\theta}_{\mathcal{N}} = \begin{bmatrix} \mathcal{N}(0, \sigma_{\mathbf{q}}, t_k) \\ \mathcal{N}(0, \sigma_{\mathbf{q}}, t_k) \\ \mathcal{N}(0, \sigma_{\mathbf{q}}, t_k) \end{bmatrix} \tag{8.13}$$

8.3.1.2 – Initial Conditions

The initial conditions were arbitrarily set to reasonable yet challenging values. Specifically, the time step interval was set to $\Delta t = 0.01$, and the angular rate was set to

$$\boldsymbol{\omega}_k = [4.0 \quad 2.0 \quad 1.0]\, deg/s$$

with an initial attitude in rotation vector form of

$$\boldsymbol{\theta}_0 = [5.0 \quad 5.0 \quad 5.0] deg$$

The initial bias vector was set to

$$\mathbf{b}_0 = [20.0 \quad -6.0 \quad 10.0] deg$$

and had a bias drift rate of

$$\mathbf{b}_\delta = [0.005 \quad 0.005 \quad 0.005]\, deg/s$$

To obtain reasonably realistic convergence rates, the noise random variables were set to values equal to and greater than the values that were found to produce good results in teleoperation of real world data as discussed in Sect. 6.8.3.2. Specifically, the drift-rate noises were set to $\sigma_{\boldsymbol{\omega},1} = \sigma_{\boldsymbol{\omega},2} = 0.25$, the drift-rate ramp noise was set to $\sigma_{\mathbf{b}} = 0.01$, and the camera attitude noise was set to $\sigma_{\mathbf{q}} = 1.0$. Using a greater amount of noise than is required will generally produce less accurate results. Therefore, the results obtained from this demonstration should be seen as an upper bound on the error of the given simulated data.

8.3.2 – Methodology

The attitude filter was implemented as described in Sect. 5.5.9 and Sect. 5.5.10, and the attitude filter was tested with simulated wiimote data over the interval $0 \leq t < 8\ s$. At each time interval Δt, the simulated true-attitude $\bar{q}_k$ was calculated using Eq. (8.8) and (8.9), the gyro bias $\mathbf{b}_k$ drifted using (8.11), a gyro measurement $\boldsymbol{g}_{s,k}$ was taken with (8.10), and finally, a camera-attitude-measurement $\bar{q}_k^c$ was taken with (8.12) and (8.13).

The attitude filter was passed through one iteration of prediction and correction, and the time t_k, simulated true-attitude $\bar{q}_k$, simulated true-attitude in rotation vector form $\boldsymbol{\theta}_k$, simulated true-bias $\mathbf{b}_k$, estimated a posterioriattitude$\hat{\bar{q}}_k^+$, the estimated a posteriori attitude in rotation vector form $\widehat{\boldsymbol{\theta}}_k^+$, estimated a posteriori bias $\widehat{\mathbf{b}}_k^+$, and a posteriori error-state covariance $\widetilde{\mathbf{P}}_k^+$ were recorded. This process of recording values was repeated at each consecutive iteration.

The attitude-error vector $\boldsymbol{\varepsilon}_k^{\boldsymbol{\theta}}$ between estimated attitude $\widehat{\boldsymbol{\theta}}_k^+$ and the simulated true-attitude $\boldsymbol{\theta}_k$ was calculated at each time step using

$$\boldsymbol{\varepsilon}_k^{\boldsymbol{\theta}} = [\boldsymbol{\theta}_k] - \left[\widehat{\boldsymbol{\theta}}_k^+\right] \tag{8.14}$$

Equivalently, the bias-error vector $\boldsymbol{\varepsilon}_k^{\mathbf{b}}$ between estimated bias $\widehat{\mathbf{b}}_k^+$ and the simulated true-bias $\mathbf{b}_k$ wascalculated at each time step using

$$\boldsymbol{\varepsilon}_k^{\mathbf{b}} = [\mathbf{b}_k] - \left[\widehat{\mathbf{b}}_k^+\right] \tag{8.15}$$

Finally, the theoretical one-σ error bounds for the attitude and bias were calculated using

$$\boldsymbol{\epsilon}_k^{\boldsymbol{\theta}} = \left[\pm\sqrt{\mathrm{P}_k^+(1,1)} \quad \pm\sqrt{\mathrm{P}_k^+(2,2)} \quad \pm\sqrt{\mathrm{P}_k^+(3,3)}\right]^{\mathrm{T}} \tag{8.16}$$

$$\boldsymbol{\epsilon}_k^{\mathbf{b}} = \left[\pm\sqrt{\mathrm{P}_k^+(4,4)} \quad \pm\sqrt{\mathrm{P}_k^+(5,5)} \quad \pm\sqrt{\mathrm{P}_k^+(6,6)}\right]^{\mathrm{T}} \tag{8.17}$$

where $\mathrm{P}_k^+(i,j)$ is the element of the a posteriori covariance at row i and column j. The error bounds are the pseudo boundaries that at least one standard deviation (~68.3%) of error should be within.

For the filter to be functioning correctly, the error boundaries should converge and bound the majority of attitude error.

8.3.3 – Results

The attitude filter was passed through one iteration of prediction and correction, and the time t_k, simulated true-attitude $\bar{q}_k$, simulated true-attitude in rotation vector form $\boldsymbol{\theta}_k$, simulated true-bias $\mathbf{b}_k$, estimated a posterioriattitude$\hat{\bar{q}}_k^+$, estimated a posteriori attitude in rotation vector form $\widehat{\boldsymbol{\theta}}_k^+$, estimated a posteriori bias $\widehat{\mathbf{b}}_k^+$, and a posteriori error-state covariance $\widetilde{\mathbf{P}}_k^+$ were recorded.

The attitude filter's estimated attitude $\hat{\bar{q}}_k^+$ (green) and the simulated true-attitude $\bar{q}_k$(blue) are plotted in Figure 8.3 to allow a direct comparison. However, the attitude filter's error-model is in

rotation vector form. Moreover, rotation vector form is easier to visualize by humans. Therefore, although the attitude filter computes the attitude in quaternion form, it was converted to rotation vector form to improve clarity and to allow the error to be directly compared with theoretical error.

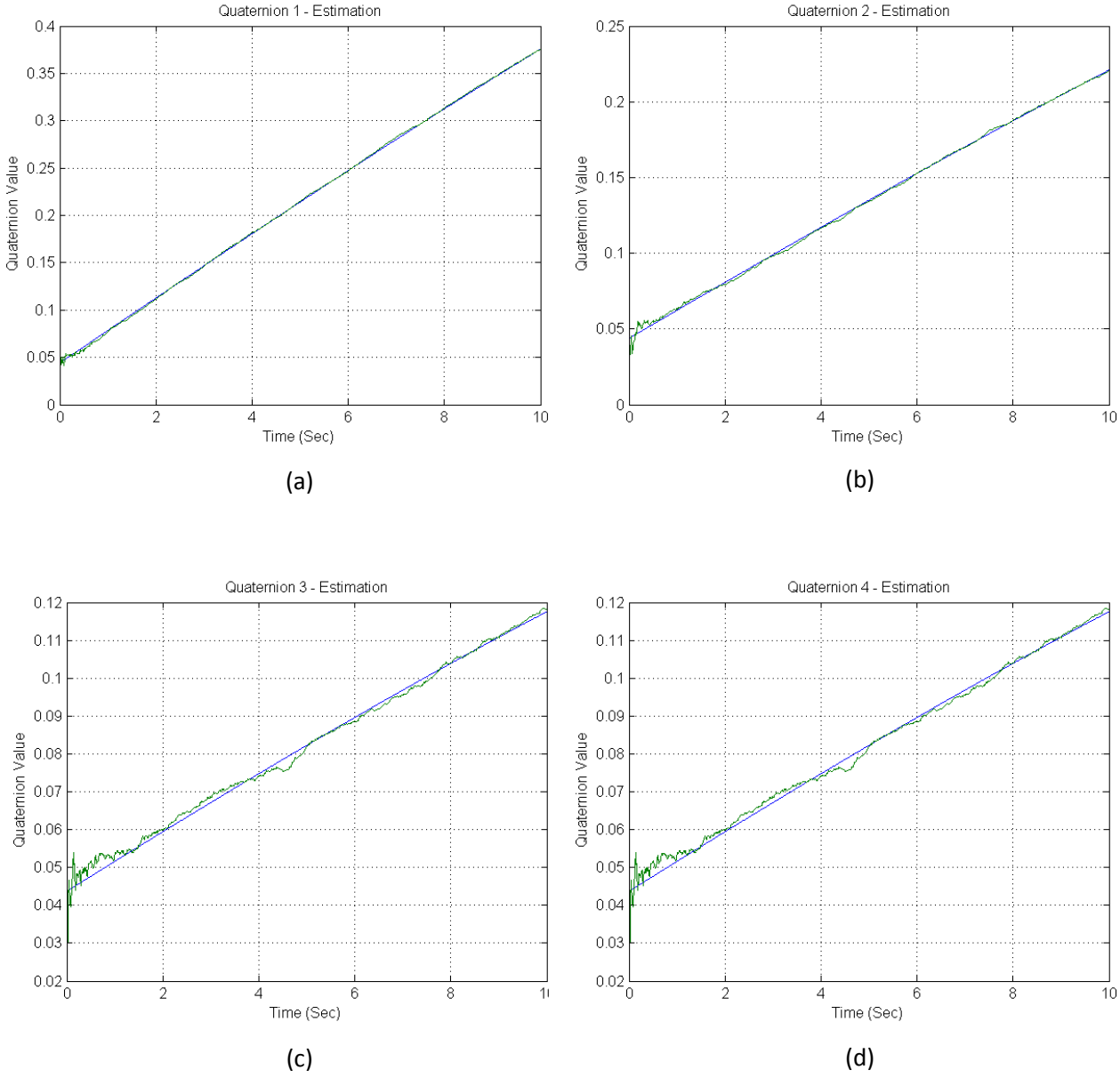


Figure 8.3. Estimated quaternion state by attitude filter (green) and true value of state (blue). Estimated quaternion component (a) q_1i, (b) q_2j, (c) q_3k, (d) q_4.

The estimated attitude in rotation vector form $\widehat{\boldsymbol{\theta}}_k^+$ (green) and the simulated true-attitude in rotation vector form $\boldsymbol{\theta}_k$ (blue) is plotted in Figure 8.4. To compare the error with the theoretical error bounds, the attitude-error vector $\boldsymbol{\varepsilon}_k^{\boldsymbol{\theta}}$ (blue) and the theoretical error bounds $\boldsymbol{\epsilon}_k^{\boldsymbol{\theta}}$ (green and red) are plotted in Figure 8.5. The theoretical error bounds converge to $\boldsymbol{\epsilon}_k^{\boldsymbol{\theta}} = [0.237\ deg \quad 0.237\ deg \quad 0.237\ deg]^{\mathrm{T}}$.

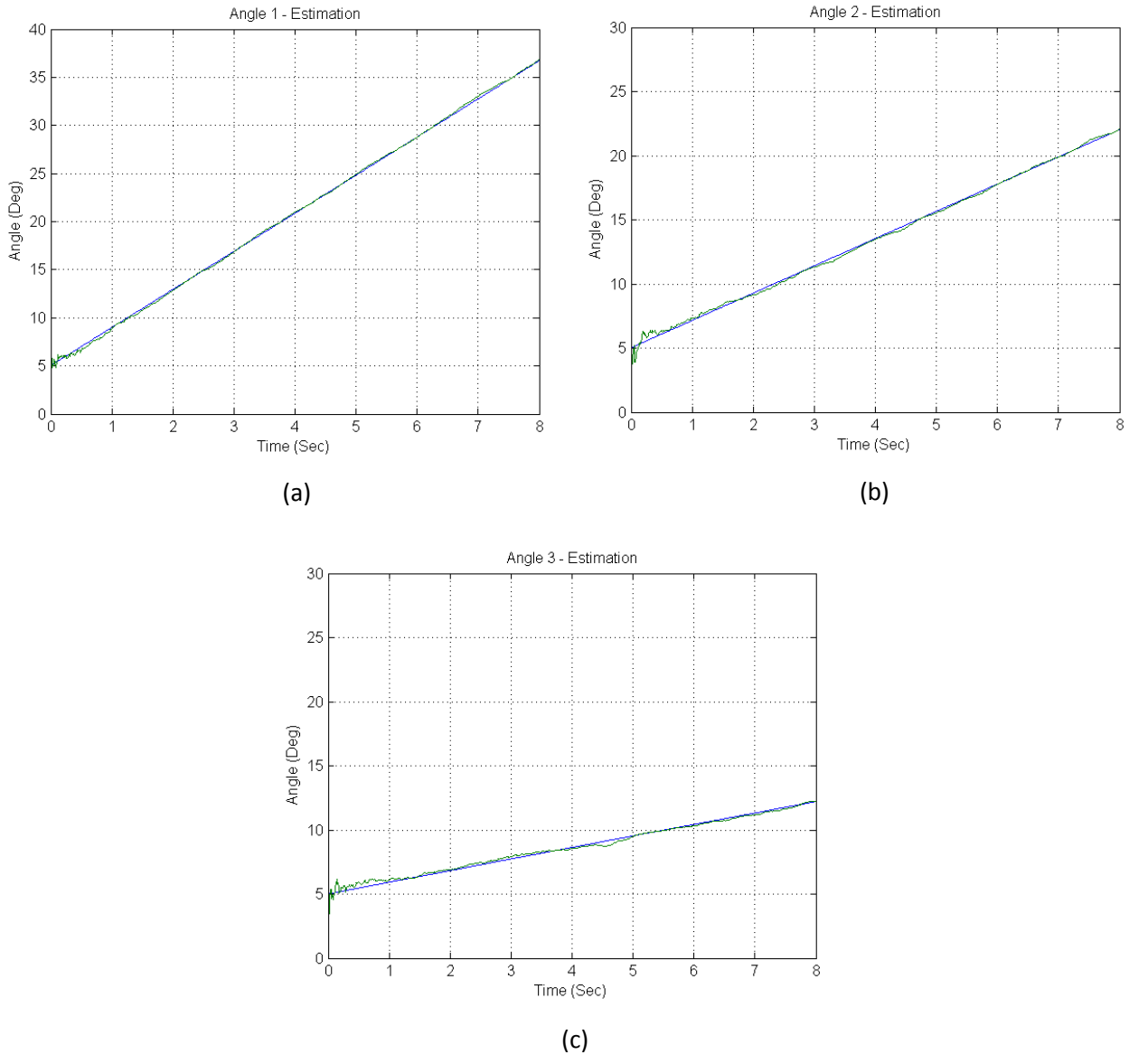
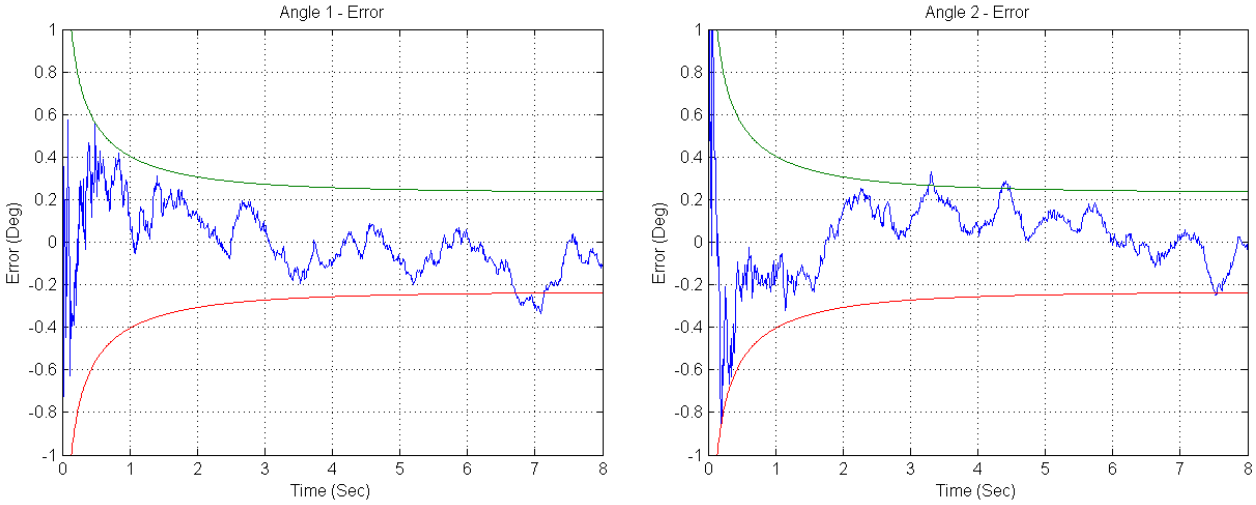
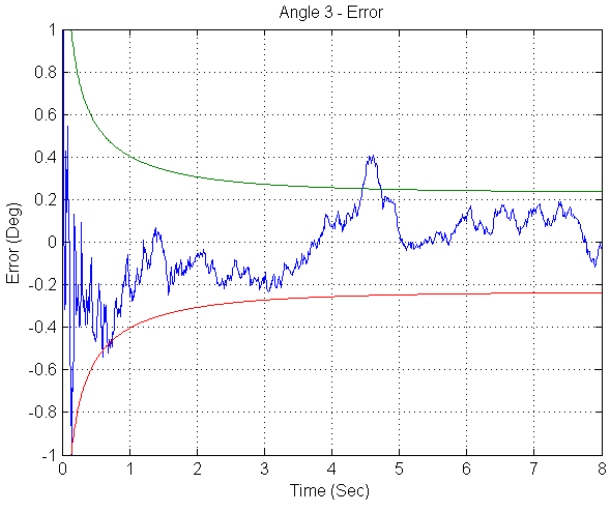


Figure 8.4. Estimated rotation vector by attitude filter (green) and true values of angles (blue). (a) Estimated Angle 1, (b) estimated Angle 2, and (c) estimated Angle 3.



(a) (b)

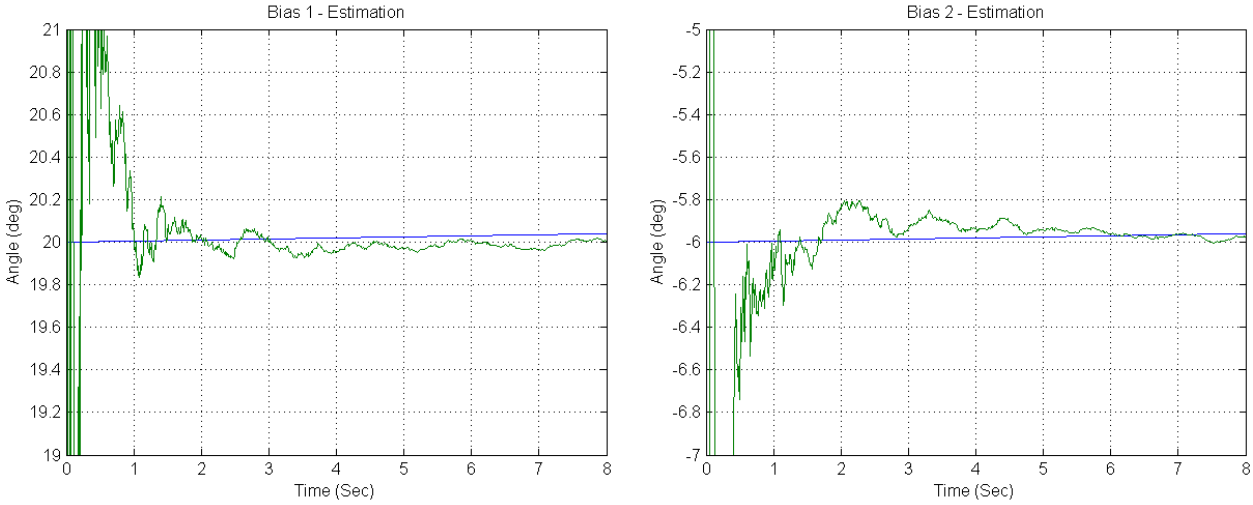


(c)

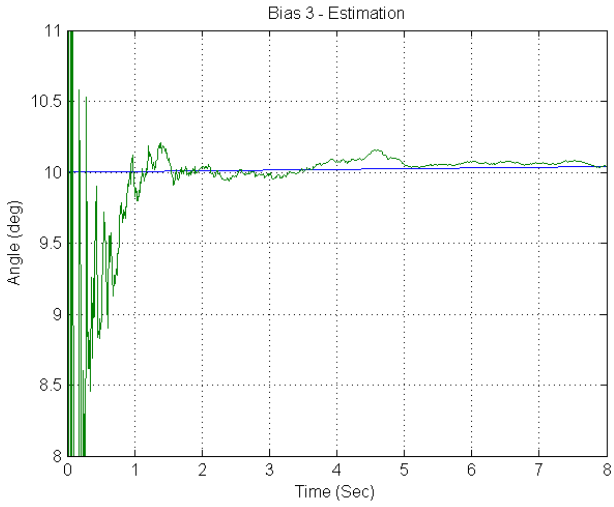
Figure 8.5. Error in rotation vector (blue) and the one-σ theoretical bounds of error (green & red) by the attitude filter. (a) Error in Angle 1, (b) error in Angle 2, and (c) error in Angle 3.

The estimated a posteriori bias $\widehat{\mathbf{b}}_k^+$ (green) and the simulated true-bias $\mathbf{b}_k$(blue) are plotted in Figure 8.6. Similarly, the bias-error vector $\boldsymbol{\varepsilon}_k^{\mathbf{b}}$(blue) and the theoretical error bounds $\boldsymbol{\epsilon}_k^{\mathbf{b}}$ (green and red) are plotted in Figure 8.7. The theoretical error bounds converge to

$\boldsymbol{\epsilon}_k^{\mathbf{b}} = [0.0997\ deg \quad 0.0997\ deg \quad 0.0997\ deg]^{\mathrm{T}}$.

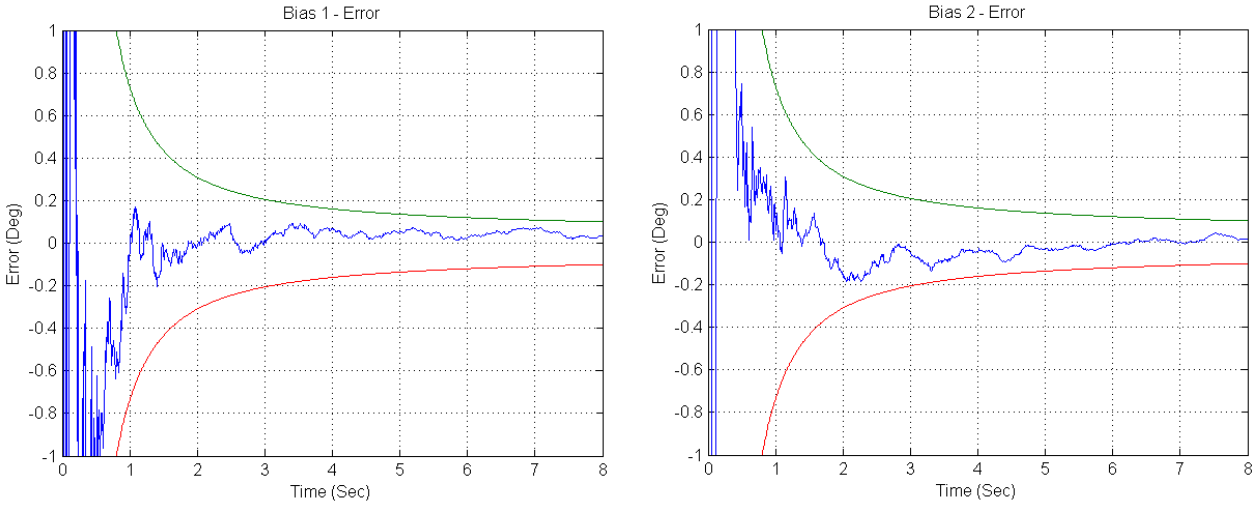


(a) (b)

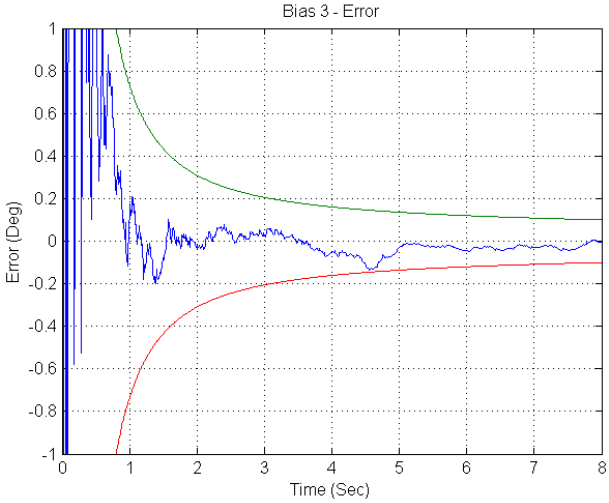


(c)

Figure 8.6. Estimated gyro biases by attitude filter (green) and true values of biases (blue). (a) Estimated Bias 1, (b) estimated Bias 2, and (c) estimated Bias 3.



(a) (b)



(c)

Figure 8.7. Error in gyro biases by attitude filter (green) and true values of biases (blue). (a) Error in Bias 1, (b) error in Bias 2, and (c) error in Bias 3.

8.3.4 – Discussion

The primary purpose of the attitude filter is to accurately estimate attitude; however, the gyro bias vector is also estimated to increase the predictive capability of the filter. The gyro biases can often be greater than the true angular rate when used in many motion capture situations, thus accurate estimates of the biases are needed to extract an accurate sample of the gyro rates. Concerning bias estimation, this experiment is significantly more challenging for the attitude filter than would be required in a real-world scenario using the complete teleoperation system. This is because during pre-estimation calibration of the pose filter, the wiimote is not moved (i.e. true angular rate is zero), thus gyro measurements are directly sampling the gyro biases and the gyro noise only. This allows a quick and accurate estimation of the gyro biases, which are available to the complete system, but not available in this simulation experiment. For this reason and since the noise values are higher than required on real data, this simulation experiment is sufficiently challenging to demonstrate proper functioning.

The nearly overlapping estimated quaternion attitude (green) and true quaternion attitude (blue) curves in Figure 8.3 demonstrate that the filter can effectively estimate attitude. Since it is difficult to appreciate the accuracy by observing quaternion values, the estimated (green) and true (blue) attitude in rotation vector form are plotted in degrees in Figure 8.4. Similarly, the estimated rotation-vector angles of attitude greatly overlap the true values. Figure 8.3 and Figure 8.4 suggest that the attitude filter is functioning correctly; however, further error analysis was required.

A detailed error analysis of the estimate rotation-vector attitude is presented in Figure 8.5. The blue line is the error in the estimated rotation angle, and green and red are the positive and negative theoretical error bounds, respectively. Observing Figure 8.5.a-c, the theoretical error bounds converge and the majority of the state error is within the error bounds, which demonstrates that the attitude filter is functioning properly. Furthermore, the error quickly converges to a value predominately within $\pm 0.237\ deg$.

8.4 – Pose-Filter Testing with Real Data

The pose filter was fully integrated into the system and was quantitatively tested with real data from the wiimote as sent by the Wiimote to Robot Interface and received by the Robot Control Server.

8.4.1 – Methodology

To test the pose filter in a realistic scenario, the complete teleoperation system was built as described in Chapter 6 and calibrated as detailed in Chapter 7. By building the complete system, it allowed for real-world data from the wiimote to be used as input for the pose filter, and the output from the pose filter was used to verify that it was functioning correctly.

To obtain the pose filter's state estimates and confidence in those estimates, the pose filter was modified to record to the hard drive the a posteriori state ${}^{\mathrm{P}}\hat{\mathbf{x}}_k^+$ and the a posteriori covariance ${}^{\mathrm{P}}\mathbf{P}_k^+$ at each time step k. The estimates at each time step k are the optimal estimation of the wiimote's state and covariance at a respective time t_k. Specifically, time t_k is the time stamp taken when the Wiimote to Robot Interface first obtains the measurements from the wiimote's sensors. At each point when the pose filter records the a posteriori state ${}^{\mathrm{P}}\hat{\mathbf{x}}_k^+$ and the a posteriori covariance ${}^{\mathrm{P}}\mathbf{P}_k^+$, the value of the time step t_k is also recorded. Relevant portions of this recorded pose filter data were plotted as discussed below.

The experiment was set up by providing power to an IR beacon and connecting the wiimote to the PC over Bluetooth. The beacon and wiimote were positioned on a stable surface, and the wiimote was pointed towards the beacon such that the four IR LEDs were in view of the wiimote camera. The Robot Control Server was executed and then the Wiimote to Robot Interface was executed in that order. The Wiimote to Robot Interface automatically connected to the wiimote, the Robot Control Server, and used the first one-hundred wiimote measurements (1.088 s) to calibrate the pose filter (q.v. Sect. 6.6.2). Following calibration, a human operator picked up the wiimote and proceeded to draw a roughly circular loop in the air, after which, it was placed back on the stable surface it was originally resting on. During this time, the time t_k, the a posteriori state ${}^{\mathrm{P}}\hat{\mathbf{x}}_k^+$, and the a posteriori covariance ${}^{\mathrm{P}}\mathbf{P}_k^+$ were recorded at each time step.

While drawing the loop, the wiimote was directed in the vicinity of the beacon, except for two short periods; the first period was initiated at roughly $t_k = 1$ s and the second was initiated at roughly

$t_k = 3$ s (by definition, the first measurement following calibration is set to $t_k \triangleq 0$). The exact starting and ending periods of the interval when the beacon's four LEDs were not completely in the field of view of the wiimote camera was obtained from the recorded data. The lack of state update during these brief periods should result in an increase in the theoretical error bounds of the attitude, keep constant the theoretical error bounds of the bias, and keep constant the theoretical error bounds of the position. The position error bounds should not change because by design, the position is not predicted when there is no position update information available, for reasons previously discussed.

After $t_k = 4.414$ s, the Wiimote to Robot Interface and Robot Control Server were shut down and no further data were recorded.

Since the pose filter's constituent filters (position filter and attitude filter) were demonstrated to be functioning individually, only aspects relating directly to pose estimation were plotted, except where noted. To demonstrate the path moved in space, the trajectory was plotted in three dimensions. Furthermore, the theoretical one-σ error bounds of the estimated a posteriori position $\hat{x}_k^+$, estimated a posteriori attitude $\widehat{\boldsymbol{\theta}}_k^+$, and estimated a posteriori bias $\widehat{\mathbf{b}}_k^+$ were plotted. Only one of each position, attitude, and bias were plotted, since the other three curves are identical.

8.4.2 – Results

The pose filter's estimated trajectory $[{}^{\mathrm{x}}\hat{\mathbf{x}}^+(t_k) \quad {}^{\mathrm{y}}\hat{\mathbf{x}}^+(t_k) \quad {}^{\mathrm{z}}\hat{\mathbf{x}}^+(t_k)]^{\mathrm{T}}_{t_k \in [0, 4.414]}$ of the roughly circular path drawn by the wiimote is illustrated in Figure 8.8. Furthermore, the converging theoretical error bounds of position ${}^{\mathrm{P}}\boldsymbol{\epsilon}_k^{\mathbf{x}}$, attitude ${}^{\mathrm{P}}\boldsymbol{\epsilon}_k^{\boldsymbol{\theta}}$, and bias ${}^{\mathrm{P}}\boldsymbol{\epsilon}_k^{\mathbf{b}}$ are obtained by taking the square root of the diagonal elements of the respective covariance matrix and are plotted in Figure 8.9. Finally, the exact time periods while not all of the four IR beacon LEDs were in the wiimote camera's field of view were determined to be between 1.053 to 1.200 s (first period) and 3.095 to 3.645 s (second period) to a tolerance within ± 0.01 seconds.

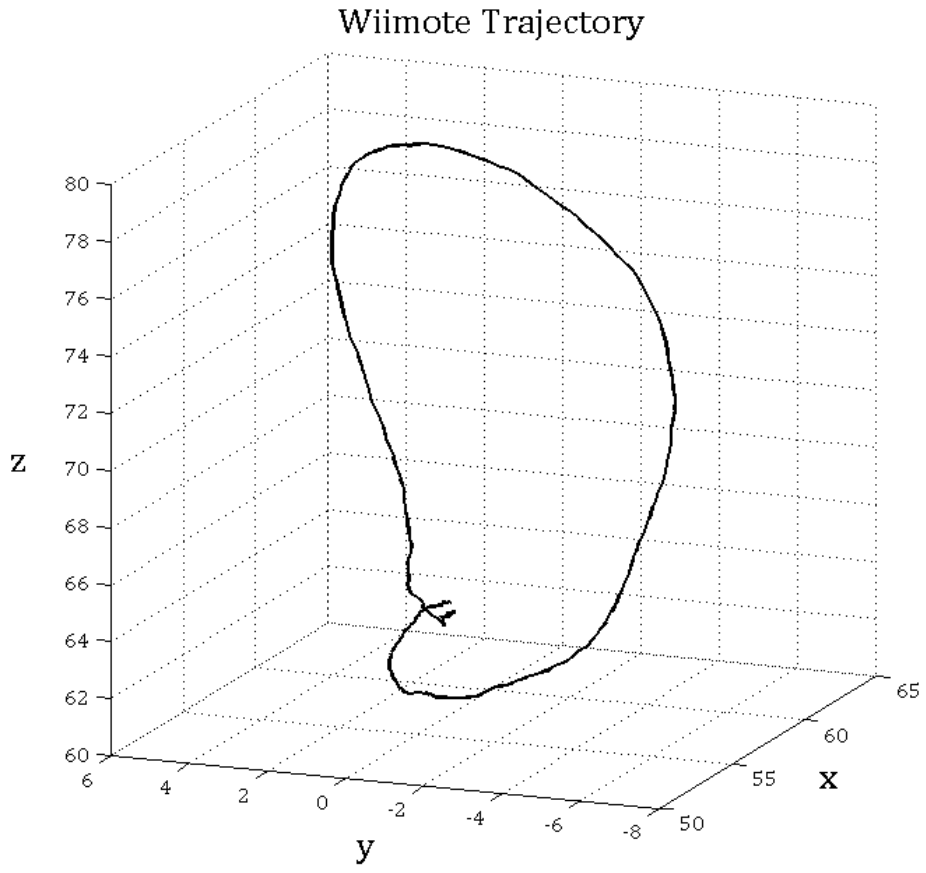


Figure 8.8. Estimated wiimote trajectory of a hand drawn loop in the air (in cm).

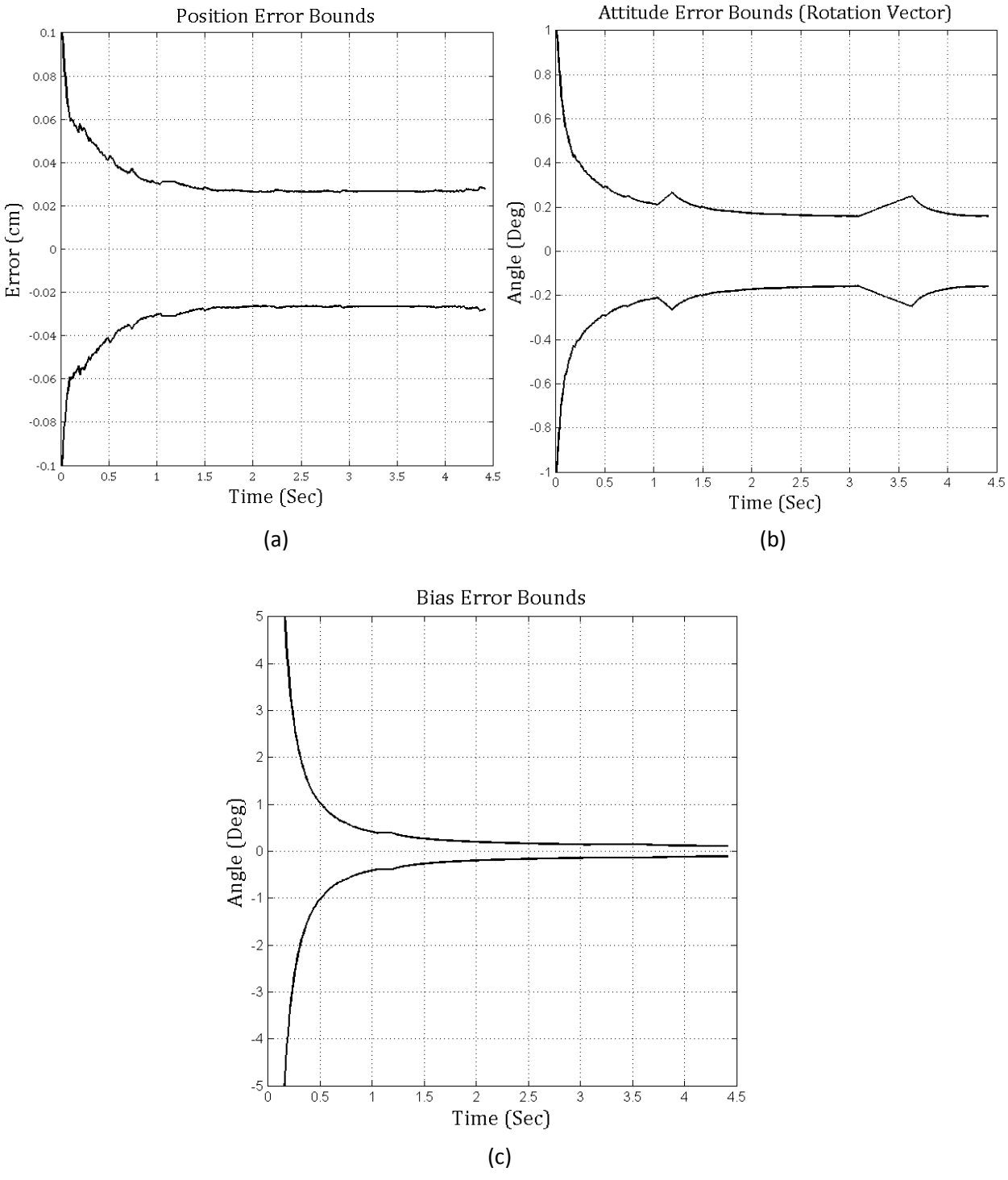


Figure 8.9. Theoretical one-σ error bounds for the pose and the estimated bias. (a) Error in position, (b) error in attitude, and (c) error in bias.

8.4.3 – Discussion

The primary purpose of this experiment was to verify that the pose filter functions correctly on real-world data. Specifically, the primary purpose was to verify that the estimated 3-DOF position and estimated 3-DOF attitude were sufficiently accurate for the applications discussed in Chapter 1.

In this experiment, the wiimote was moved in a roughly circular trajectory. The pose filter's estimated path of this trajectory is illustrated in Figure 8.8. Unfortunately, the true trajectory is unknown. However, a rough analysis on the state accuracy can still be performed for the following reasons. First, a Kalman-derived filter performs optimally on real data when the selected values for the noise variable are relatively close or even slightly greater than the true noise values. Second, when the noise values are properly selected, the majority of the error in estimated state is within the theoretical error bounds. Third, the noise values selected for the pose filter were partially based on the variance of the sensors' noise, but were greatly modified to improve performance. Specifically, they were selected through systematic trial by running the system many times and selecting the noise values that appeared to maximize pose accuracy. Therefore, these three statements suggest that error of the estimated state by the pose filter is in the vicinity of the theoretical error. This claim is further supported by the demonstration in the previous two experiments that the error in the estimated state by the pose filter's constituent filters (pose and attitude filter) was predominately within the theoretical boundary on simulated data.

As shown in Figure 8.9, the theoretical error in pose is quite low. The theoretical position error-bounds converge to roughly $\pm 0.028\ cm$ (Figure 8.9.a) per axis and the theoretical attitude error-bounds converge to roughly $\pm 0.16\ deg$ (Figure 8.9.b) per axis. As demonstrated in Figure 8.7 and for reasons previously discussed, the true error in the pose filter gyro bias estimates is likely significantly less than the theoretical error bounds in Figure 8.9.b. It should also be noted that the calibration of the pose filter provides a priori knowledge of the true state, and since the state covariance has not been adjusted to account for this, the true error rate will converge more quickly than the theoretical rate, particularly for the estimated gyro biases.

Finally, as expected, during the periods when the beacon is out of view of the camera (1.053 to 1.200 and 3.095 to 3.645 s), the position and bias error bounds remain constant, and the attitude error bounds increase. Once the wiimote camera regains the view of the beacon, at 1.200 s and 1.3645 s, the

theoretical error bounds continue to converge, which is what would be expected of a properly functioning filter.

This experiment on the pose filter demonstrates that the pose filter is functioning correctly, and suggests that the estimated state is sufficiently accurate.

8.5 – Full-System Testing with Real Data

The final phase of testing was performed to provide evidence that the complete system functions as stated. In this *qualitative* testing, the wiimote was moved into various poses and the system controlled a robot manipulator via teleoperation to match the pose of the wiimote. This experiment does not quantitatively test the accuracy of the system; it was performed solely to provide further evidence that the complete system is fully functional.

8.5.1 – Methodology

To test the full system in a realistic scenario, the complete teleoperation system was built as described in Chapter 6 and calibrated as detailed in Chapter 7. This experiment was designed to verify that the wiimote can be used to teleoperate a robotic manipulator and was tested by placing the wiimote in varies poses and verifying that a robot manipulator successfully moved to the appropriate pose.

The experiment was set up by providing power to an IR beacon and connecting the wiimote to the PC over Bluetooth. The beacon and wiimote were positioned on a stable surface, and the wiimote was pointed towards the beacon such that the four IR LEDs were in view of the wiimote camera. The Robot Control Server was executed and then the Wiimote to Robot Interface was executed in that order. The Wiimote to Robot Interface automatically connected to the wiimote, the Robot Control Server, and used the first one hundred wiimote measurements to calibrate the pose filter (q.v. Sect. 6.6.2).

Following calibration, a human operator picked up the wiimote and qualitatively tested it under multiple pose trajectories of various types to determine if the system appeared to properly teleoperate a robot manipulator using the wiimote. Observations and a subjective analysis were recorded. Afterwards, a human operator proceeded to place the wiimote in eight qualitatively different poses where the beacon's four LEDs are in view of the wiimote camera. The eight poses were chosen such that they were reachable by the robot manipulators. The poses were: top left, top right, centre, bottom

left, bottom right, roll, forward, and backward. Wrt the Robot Frame, *top* is high z, *bottom* is low z, *right* is high y, *left* is low y, *forward* is high x, *backward* is low x, *centre* is at home position, and *roll* is rotation about the x-axis (as illustrated in Figure 8.10).

It was also necessary to test the system with poses that are not attainable by the robot manipulators. Specifically, two poses were chosen that are solvable by the inverse kinematics but outside of a joint's range of motion, and one pose was chosen to demonstrate when the pose is not solvable by the inverse kinematics.

The Virtual Robot Manipulator was chosen to demonstrate the poses, because it tests more of the system (i.e. Virtual Robot Controller and Virtual Robot Manipulator); however, the system could be set to bypass the virtual system and instead be directly handled by the C500C Robot Controller. It is already known that the C500C Robot Controller and A465 Robot Manipulators are fully functioning, thus, the virtual system was used instead.

As each of the eleven poses were performed on the wiimote, a screenshot of the Virtual Robot Manipulator performing the corresponding poses was captured.

8.5.2 – Results

Qualitatively, the robot manipulator's teleoperated pose mimics the wiimote pose sufficiently accurately that the author was not able to distinguish a difference using the human eye alone. Furthermore, from visual inspection, the robot's pose trajectory followed the wiimote's pose trajectory quite quickly, with one exception. Due to the high level of error in the accelerometer measurements, there was a small lag in teleoperated positions when the human operator caused sudden and relatively large changes in acceleration. However, this latency in position was small and fully within acceptable limits of applications presented in this work. Additionally, the attitude latency was negligible due to the relatively high accuracy of the gyroscopes.

The screen shots of the eight reachable poses are shown in Figure 8.10, and the three unreachable poses are shown in Figure 8.11.

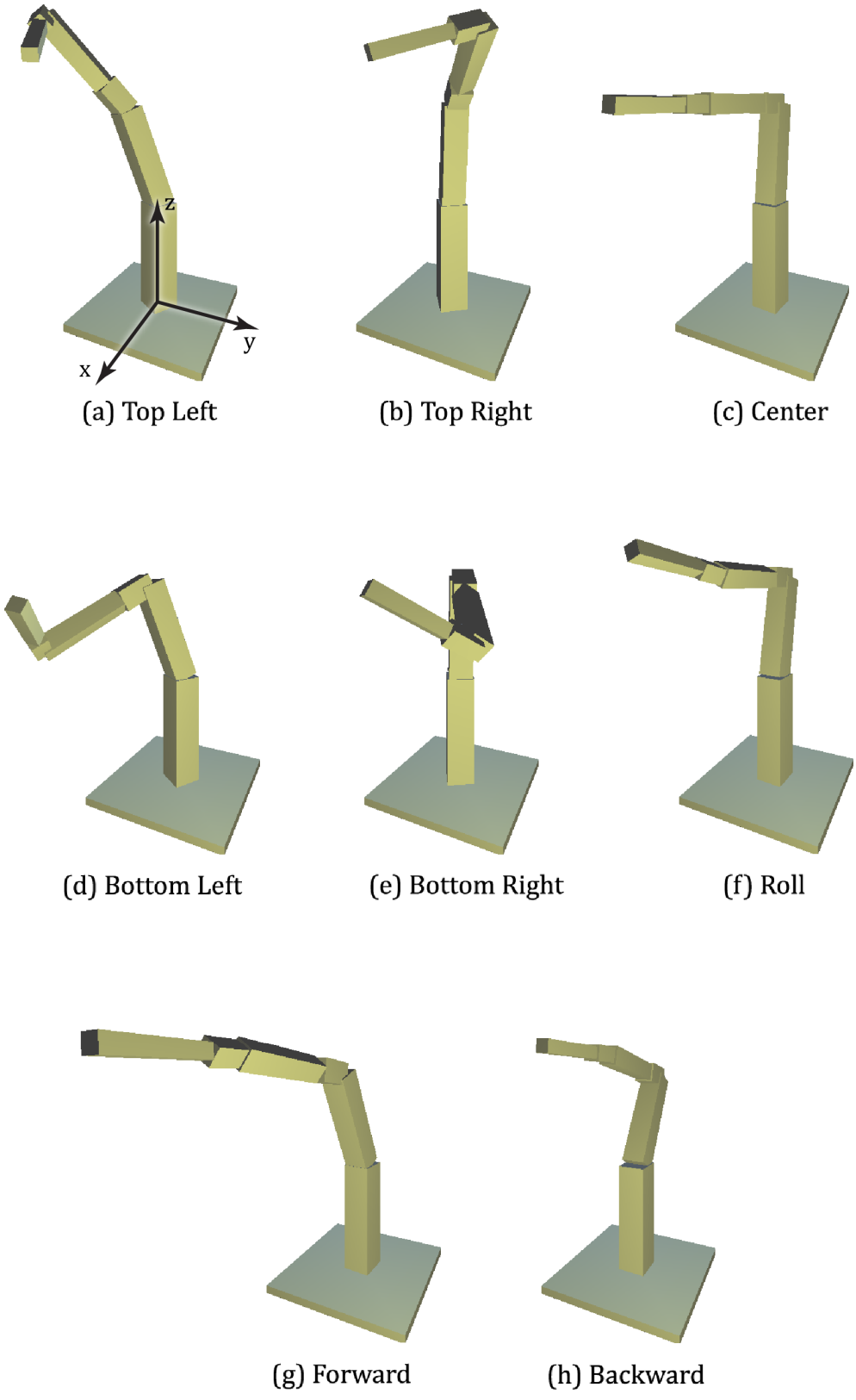


Figure 8.10. Screenshots for eight reachable poses by the Virtual Robot Manipulator.

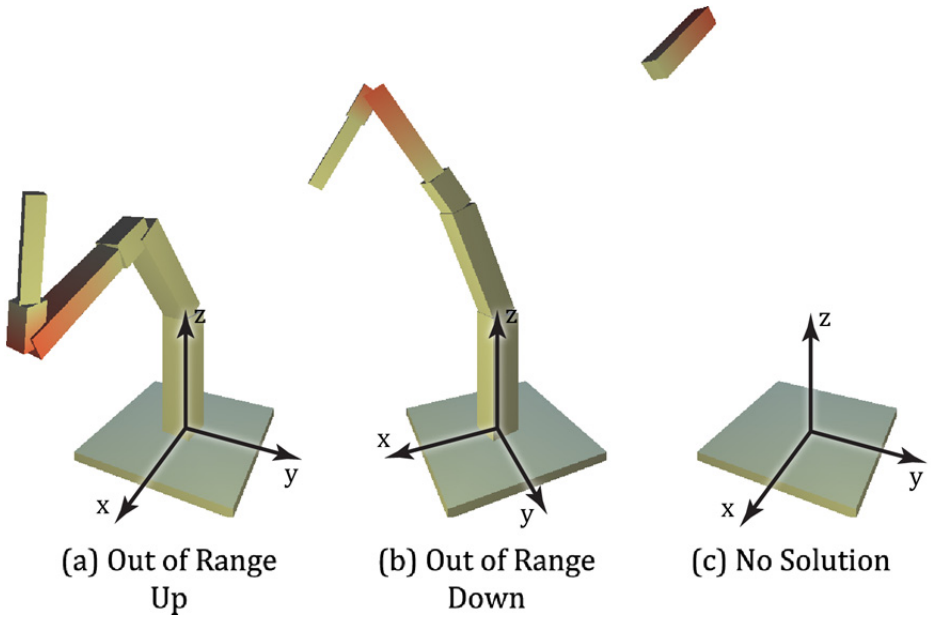


Figure 8.11. Screenshots for three unreachable poses by the Virtual Robot Manipulator.

8.5.3 – Discussion

The purpose of this experiment was a high-level subjective test to suggest that the whole system functions correctly. As shown in Figure 8.10, the system is capable of teleoperating the Virtual Robot Manipulator to various wiimote poses. Furthermore, Figure 8.11 demonstrates that the Virtual Robot Manipulator will visually indicate when a pose is not reachable by the A465 Robot Manipulator. Figure 8.11.a and Figure 8.11.b are poses that are solvable by the inverse kinematics, but not solvable when the motion constraints of the joint are considered, and Figure 8.11.c demonstrates a pose that is completely outside of the A465 Robot Manipulator's reachable domain, even without consideration of the A465 Robot Manipulator's constraints on joint motion.

It should be noted that the Virtual Robot Controller receives commands that are equivalent to those that are sent to the C500C Robot Controller. Since it is known that the C500C Robot Controller is fully functional, the experimental poses in Figure 8.10 could have been equivalently performed on the A465 Robot Manipulator. However, because of limitations on the range of motion of the joints of the A465 Robot Manipulator, the experimental poses in Figure 8.11 cannot be performed by the A465 Robot Manipulator.

The subjective tests performed in this experiment along with the quantitative results from the previous three experiments suggest that the system is fully functioning, and is sufficiently accurate for the applications discussed in Chapter 1.

Comparing the accuracy of the system to other commercially available systems is reserved for future work (q.v. Sect. 9.3).

8.6 – Limitations of the System

The tested system has been shown to be an effective and fully functioning motion capture and teleoperation system. However, the limitations of the system should also be discussed, which were observed during testing.

The largest limitation of the system is that the wiimote camera has a limited field of view. Consequently, if the attitude wrt the beacon varies too greatly, the wiimote will lose sight of the beacon, and the system will temporarily drop from 6-DOF motion to 3-DOF. However, there are many possible solutions to this problem. One solution would be to use multiple beacons, although, this method also has drawbacks, since a pre-estimation discovery mode would be required to identify the absolute pose of the beacon. A better solution would be to use the calibration board as a "smart" beacon, where the PC would use the Arduino microcontroller to activate the LEDs that are closest to the camera's centre of view, as described in Sect. 7.3.1. This would increase the variety of poses that could be used for teleoperation by using a virtual beacon that could move to any known location on the calibration board. This solution is substantially easier to implement than the first solution. A third solution would be to attach one or more beacons to the end of the wiimote and use a second wiimote to identify the affixed beacon(s). Both the second and third solutions would address the camera's limited field of view limitation sufficiently well.

A second limitation of the system is that the wiimote needs to be kept within roughly a meter of the beacon. This is a consequence of the resolution of the wiimote camera and the small size of the LEDs used on the beacon (3 *mm*). This limitation is easily remedied by using bright LEDs that are larger (5 *mm* or larger) and sufficiently separated, which would enable the remote to be used at several meters away.

The final limitation of the system is that there is a small latency in position when sudden and large changes in acceleration are produced on the wiimote. As previously mentioned, the latency is relatively small (negligible for smooth movements), and would only be an issue in applications where many quick movements are required. This limitation could be remedied by using better sensors. However, despite this minor limitation, this characteristic has more benefits than drawbacks, since high-acceleration motions such as jittering of a hand are often not wanted. This characteristic effectively provides a smoothing function, which allows the trajectories to be more fluid and closer to the human operator's desired trajectory. Smoothing of high accelerations is commonly employed in teleoperation systems, and is essential for teleoperation work that requires fine movements, such as those performed in the health industry.

Since the third limitation provides a greater benefit than its relatively minor drawback, only the first two limitations are addressed in the Future Work Section of the thesis (Sect. 9.3).

Chapter 9 – Conclusion and Future Work

9.1 – Summary of Thesis

There is a need for a 6-DOF motion capture system for robotics and more generally for human-computer interaction that is intuitive, low-cost, and sufficiently accurate. However, no such system is currently available commercially. The general goal of this work was to address this problem by developing a motion capture system to function as a general-purpose motion capture system for human-computer interaction. However, the majority of the thesis is devoted to a more specific goal, to provide a solution to this problem that focuses on an application within a programming-by-demonstration system, where a motion capture system was developed to teleoperate a robotic manipulator.

A novel teleoperation system was developed to replace a previous teleoperation system used for the teaching portion of a programming-by-demonstration system. The teleoperation system is composed of a Kalman-filter based 6-DOF motion capture system using a wiimote and a beacon with four infrared LEDs on it. On the software side, the system is divided into two subsystems, the Wiimote to Robot Interface and the Robot Control Server. The Wiimote to Robot Interface uses Bluetooth to connect to the wiimote to obtain the wiimote's sensor measurements and other state information. The Wiimote to Robot Interface collects the accelerometer measurements, gyroscope measurements, and 2D locations of the LEDs on the wiimote camera and sends these values over a TCP/IP connection to the Robot Control Server. The TCP/IP connection allows teleoperation to be performed at remote sites. Upon receiving the incoming measurements, the Robot Control Server sends the information to the pose filter to determine the location of the wiimote in space.

The pose filter is a hierarchical stochastic filter composed of position filters and an attitude filter, which estimates 3-DOF position and 3-DOF attitude, respectively. The position filters are based on a standard Kalman filter, and the attitude filter is a multiplicative extended Kalman filter, which is a quaternion-based extended Kalman filter. The position filter optimally estimates the 6-DOF pose of the wiimote at roughly 100 *Hz* to obtain a 6-DOF trajectory where each pose in the trajectory is the pose that maximizes the a priori probability of all previous sensor values.

The pose filter uses measurements from the wiimote sensors to optimally estimate pose. The gyroscope provides angular rate information, which aids in attitude determination. The accelerometers

provide acceleration information, which aids in position determination; however, since determination of true accelerations is highly inaccurate, the accelerometers aid only minimally in the estimation of position. The camera provides absolute position and attitude information, which are used in the correction phase of the pose filter. However, the homography between the wiimote and the beacon must be solved at each time step before the absolute position and attitude information is available. Additionally, the camera's intrinsic parameters are required, which were previously unknown.

To obtain the camera's intrinsic parameters, a novel calibration technique was developed. A calibration board was precision fabricated with eighty infrared LEDs embedded into the board. Using a microcontroller, the LEDs were flashed on and off in sequence while the wiimote camera took multiple 2D measurements of the location of the LEDs on the camera sensor. The measurements were then merged to create a pseudo image of the calibration board. Taking multiple pseudo images of the calibration at various poses, the camera was calibrated to obtain its intrinsic parameters. These intrinsic parameters were used to solve the homography to obtain absolute estimates of the pose, which was combined with the gyroscope and accelerometer values by the pose filter to optimally estimate pose at each time step.

After implementing the complete system, a human operator can move the wiimote in a 6-DOF trajectory and the estimated pose of the wiimote was used to teleoperate a virtual robotic manipulator in real time. Through testing of this system, it was determined that the pose filter was fully functioning and that the complete teleoperation system was capable of controlling a robotic manipulator using one-to-one movements with the wiimote.

9.2 – Summary of Contributions

This section provides a summary of the contributions of this thesis that were originally stated in Sect. 1.3. This research primarily focused on the development of a robotic teleoperation system based on a 6-DOF motion capture system; however, the system and portions of it can be beneficial in other domains. Thus, this work is a contribution in six areas.

Transfer of research from guidance, control, and dynamics to the robotics field.

Literature on the multiplicative extended Kalman filter (MEKF) is predominately found in the aerospace and navigation fields, where it is used for optimal estimate of a craft's attitude. Since the applied

mathematics of these algorithms can become quite involved, it may be difficult for researchers in other fields to take advantage of the algorithm without a strong control and mathematical background. To the best of the author's knowledge, there exists no published work that contains a complete treatment of the multiplicative extended Kalman filtering (MEKF) assuming only basic prior knowledge of probability theory and linear systems of ordinary differential equations.

This thesis rigorously details the MEKF and provides relevant background knowledge in a complete, and technically accurate manner using consistent notation. Furthermore, derivations for many of the MEKF equations are provided in detail. This is important since the system's equations of motion are built directly into the filter. Providing these detailed derivations allow other researchers to modify the pose filter to directly match the hardware they are using and the system they wish to predict.

A novel pose filter

Optimal estimation of position and attitude are generally addressed independently. However, much of the research that does address them simultaneously are designed for different sensors, are computationally expensive, or use a less precise algorithm. This research develops a novel 6-DOF pose filter based on linear Kalman filters and a multiplicative extended Kalman filter, which effectively estimates pose of a wiimote using low-grade sensors.

Using a novel calibration method and the pose filter, the first publicly available software system that is capability of producing 6-DOF motion capture using the wiimote was developed

Gaining precise pose for the wiimote's infrared camera is difficult since the camera calibration parameters are unknown. Moreover, standard calibration techniques are difficult since the user does not have access to the camera's image but rather, the 2D relative centroid locations of (up to) the four brightest infrared lights that are greater than a pre-specified threshold. A novel calibration technique was developed that merges multiple camera sensor measurements taken of a precision-machined and microcontroller-controlled matrix of infrared LEDs to acquired pseudo image data that is used to find the camera's intrinsic parameters. Following calibration, the camera can be used to find the pose of beacon with respect to the camera.

Using this calibration method and the novel pose filter, the first publicly available software was developed that is capable of calculating stable 6-DOF pose from the wiimote sensors.

A very low-cost motion capture system

Development of a stable 6-DOF pose estimator is a contribution to the field of motion capture. As discussed in Sect. 2.2, stable 6-DOF motion capture can be thousands and even tens of thousands of dollars. At less than one hundred dollars, this motion capture system can be built, thus greatly reducing system cost and allowing it to find use in many other areas where cost may have previously made use of the technology prohibitive. This system can also be used as a human interface device to provide 6-DOF navigation of a 3D virtual environment.

A novel teleoperation system

The development of novel motion capture system is also a contribution to the field of telerobotics. As low-cost robots become increasingly common, cheaper methods of operating them become increasingly important to allow widespread adoption.

A novel system for the teaching portion of PbD

Teaching robots tasks can be a costly and involved process. Intuitive 6-DOF motion capture devices are costly, and to reduce costs, the teach pendant is generally used instead. Consequently, the teach pendant's trajectories are rarely optimal, velocity information cannot be directly controlled, and it is more difficult and less intuitive to use. However, cost is often the largest constraint, and despite the large limitations of the device's performance, its low cost has allowed it to become the most common device used to teach robotic manipulator systems.

Use of a 6-DOF motion capture device for teleoperation can circumvent the teach pendant's drawbacks; however, the cost of 6-DOF motion capture has prevented widespread adoption. A low-cost 6-DOF teleoperation device would solve this and would thus be a better alternative to the teach pendant in almost every situation.

These six contributions demonstrate the wide applicability of the system developed in this research.

9.3 – Future Work

To further improve this system and address the limitation discussed in Sect. 8.6, various modifications are proposed. To overcome the wiimote camera's limited field of view limitation (first limitation), the

smart beacon should be implemented as described in Sect. 7.3.1. Furthermore, the limitation that the wiimote must be used within roughly a meter of the beacon (second limitation) can be addressed by using larger LEDs on the calibration board.

Additionally, it would be beneficial to perform higher precision testing on the accuracy of the teleoperation system with real data. Since the A465 Robot Manipulator has a repeatability of $\pm 0.05\ mm$ and a resolution between from $0.0\ mm$ to $0.150\ mm$ depending on the axial and radial distance from the tool flange surface and centre, it can be used fairly effectively to test the accuracy of the wiimote. To test the accuracy of the teleoperation system, the wiimote should be attached to the end effector of the A465 Robot Manipulator such that the wiimote camera sensor is approximately overlapping the robot's Tool Frame. Next, the A465 Robot Manipulator would be directed to perform a predefined trajectory or set of poses. Finally, the known trajectory or poses of the robot's Tool Frame can be directly compared to the poses estimated by the teleoperation system. Although, the accuracy of the teleoperation system was found to be sufficient for the current application, more precise testing such as the experiment described here would further clarify the applicability of the current system to other applications.

Bibliography

[1] International Federation of Robotics (IFR). (2011). World Robotics – Service Robots 2011.

[2] International Federation of Robotics (IFR). (2011). World Robotics – Industrial Robots 2011.

[3] Lockerd A. and Breazeal C. (2004). "Tutelage and socially guided robot learning", IEEE/RSJ Int. Conf. on Intelligent Robots and Systems, pt 4, v 4, p 3475-80.

[4] Wu, X. and Kofman, J. (2008). "Human-Inspired Robot Task Learning from Human Teaching", *proc. of IEEE Int. Conf. on Robotics and Automation (ICRA 2008)*, Pasadena, CA, USA, p 3334-39.

[5] Crassidis, J.L., Markley, F.L., and Cheng, Y., "A Survey of Nonlinear Attitude Estimation Methods," AIAA Journal of Guidance, Control, and Dynamics, Vol. 30, No. 1, Jan.-Feb. 2007, pp. 12-28.

[6] Lockerd A. and Breazeal C.(2004), "Tutelage and socially guided robot learning", *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, pt 4, v 4, p 3475-80.

[7] Biggs, G., MacDonald, B. (2003). A Survey of Robot Programming Systems. *Proceedings of the Australasian Conference on Robotics and Automation*, *CSIRO*, **1**.

[8] Wu, X. (2009). Human-Inspired Robot Task Teaching and Learning. Doctoral dissertation, Department of Systems Design Engineering, University of Waterloo.

[9] Billard and S. Schaal. Robust learning of arm trajectories through human demonstration. In Intelligent Robots and Systems, 2001. Proceedings. 2001 IEEE/RSJ International Conference on, volume 2, pages 734{739, Nov 2001.

[10] Juyang Weng and Yilu Zhang. Action chaining by a developmental robot with a value system. In Development and Learning, 2002. Proceedings. The 2nd International Conference on, pages 53{60, 2002.

[11] W.D. Smart and L. Pack Kaelbling. E_ective reinforcement learning for mobile robots. In Proceedings of the IEEE Intl. Conf. on Robotics and Automation (ICRA '02), volume 4, pages 3404{3410, May 2002.

[12] R.M. Voyles and P.K. Khosla. Gesture-based programming: a preliminary demonstration. In Proceedings of the IEEE Intl. Conf. on Robotics and Automation (ICRA '99), volume 1, pages 708{713, May 1999.

[13] J.J. Steil, G. Heidemann, J. Jockusch, R. Rae, N. Jungclaus, and H. Ritter. Guiding attention for grasping tasks by gestural instruction: the gravis-robot architecture. In Intelligent Robots and Systems, 2001. Proceedings. 2001 IEEE/RSJ International Conference on, volume 3, pages 1570{1577, Nov 2001.

[14] M. Strobel, J. Illmann, B. Kluge, and F. Marrone. Using spatial context knowledge in gesture recognition for commanding a domestic service robot. In Robot and Human Interactive Communication, 2002. Proceedings. 11th IEEE International Workshop on, pages 468{473, 2002.

[15] S. Lauria, G. Bugmann, T. Kyriacou, and E. Klein. Mobile robot programming using natural language. Robotics & Autonomous Systems, 38(3{4):171{181, March 2002.

[16] Schaal, S (1999). "Is imitation learning the route to humanoid robots?", *Trends in Cognitive Sciences*, v 3, n 6, p 233-242.

[17] Thomaz A.L. and Breazeal C. (2008), "Experiments in socially guided exploration: lessons learned in building robots that learn with and without human teachers", *Connection Science*, v 20, n 2-3, p91-110

[18] Wu, X. and Kofman, J. (2008). “Human-Inspired Robot Task Learning from Human Teaching”, *proc. of IEEE Int. Conf. on Robotics and Automation (ICRA 2008)*, Pasadena, CA, USA, p 3334-39.

[19] Muto S. and Shimokura K. (1994), “Teaching and control of robot contour-tracking using contact point detection”, *in Proc. 1994 IEEE Int. Conf. on Robotics and Automation*, v 1, pt 1, p 674-81.

[20] Choi M. H. and Kim S. J. (2000), “Force/moment direction sensor and its use for human-robot interface in robot teaching”, *Proc. of the IEEE Int. Conf. on Systems, Man and Cybernetics*, v 3, p 2222-7.

[21] Choi M. H. and Kim S.J. (2001), "A Force/Moment Direction Sensor and Its Application in Intuitive Robot Teaching Task", *Trans. on Control, Automation, and Systems Engineering*, v 3, n 4, p 236-241.

[22] Kristensen S., Horstmann S., Klandt J., Lohnert F., and Stopp A. (2001),”Human-friendly interaction for learning and cooperation”, *Proc. of IEEE Int. Conf. on Robotics and Automation*, v 3, p 2590-2595.

[23] Dixon K.R. (2004), Inferring User Intent for Learning by Observation, doctoral dissertation, Department of Electrical & Computer Engineering, Carnegie Mellon University.

[24] Akanyeti O., Nehmzow U. and Billings S.A. (2008), “Robot training using system identification”, *Robotics and Autonomous Systems*, v 56, n 12, p 1027-41

[25] Calinon S, Guenter F. and Billard A. (2007), “On learning, representing, and generalizing a task in a humanoid robot”, *IEEE Trans. Syst, Man, Cybernetics, Part B*, v 37, n 2, p 286-98

[26] Voyles R.M., Morrow J.D., and Khosla P.K. (1997), “Towards gesture-based programming: shape from motion primordial learning of sensorimotor primitives”, *Robotics and Autonomous Systems*, v 22, n 3-4, p 361-75.

[27] Uechi M., Ogata H., Nakamura Y., and Mizukawa M. (1999), “A component architecture for customizing robot-teaching systems”, *in Proc. 1999 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, v 3, pt 3, p 1625-32.

[28] Kunii Y. and Hashimoto H. (1997), “Tele-teaching by human demonstration in virtual environment for robotic network system”, *in Proc. 1997 IEEE Int. Conf. on Robotics and Automation*, v 1, pt 1, p 405-10.

[29] Aleotti J. and Caselli S. (2006), “Robust trajectory learning and approximation for robot programing by demonstration”, *Robotics and Autonomous Systems*, v 54, n 5, p 409-13

[30] Choi M. H. and Lee W. W. (2003), “Quantitative evaluation of an intuitive teaching method for industrial robot using a force / moment direction sensor”, *Int. J. of Control, Automation and Systems*, v 1, n 3, p 395-400.

[31] Kuniyoshi Y., Inaba M., and Inoue H. (1994), “Learning by watching: extracting reusable task knowledge from visual observation of human performance”, *IEEE Trans. on Robotics and Automation*, v 10, n 6, p 799-822.

[32] Dillmann R., Rogalla O., Ehrenmann M., Zöllner R., and Bordegoni M. (1999), “Learning Robot Behaviour and Skills Based on Human Demonstration and Advice: the machine learning paradigm”, *Proc. of the 9th Int. Symposium of Robotics Research*, Utah, USA, p 229–38 .

[33] Dillmann R. (2004), “Teaching and learning of robot tasks via observation of human performance”, *J. of Robotics and Autonomous Systems*, v 47, n 2-3, p 109-116.

[34] Ekvall S. and Kragic D. (2006), “Learning task models from multiple human demonstrations”, *15th IEEE Int. Symposium on Robot and Human Interactive Communication*, p 358-363.

[35] Chella A., Dindo H. and Infantino I. (2007), “Imitation learning and anchoring through conceptual spaces”, *Applied Artificial Intelligence*, v 21, n 4-5, p 89-105

[36] Rusu R.B., Gerkey B. and Beetz M. (2008), “Robots in the kitchen: exploiting ubiquitous sensing and actuation”, *Robotics and Autonomous Systems*, v 56, n 10, p 844-56.

[37] Peters R.A. II, Campbell C.L., Bluethmann W.J., and Huber E. (2003), “Robonaut task learning through teleoperation”, *IEEE Int. Conf. on Robotics and Automation*, pt 2, v 2, p 2806-11.

[38] Lieberman J. and Breazeal C. (2004), “Improvements on action parsing and action interpolation for learning through demonstration”, *in Proc. 2004 4th IEEE/RAS Int. Conf. on Humanoid Robots*, v 1, pt 1, p 342-65.

[39] Chen J. and McCarragher B. (1998), “Robot programming by demonstration-selecting optimal event paths”, *in Proc. 1998 IEEE Int. Conf. on Robotics and Automation*, v 1, pt 1, p 518-23.

[40] Ekvall S. and Kragic D. (2008), “Robot learning from demonstration: A task-level planning approach”, *Int. Journal of Advanced Robotic Systems*, v 5, n 3, p 223-234.

[41] Chella A., Dindo H. and Infantino I. (2006), “Learning high-level tasks through imitation”, *2006 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, p 3648-54

[42] Nakaoka S., Nakazawa A., Kanehiro F., Kaneko K., Morisawa M., Hirukawa H., and Ikeuchi K. (2007), “Learning from observation paradigm: leg task models for enabling a biped humanoid robot to imitate human dances”, *Int. Journal of Robotics Research*, n 8, p 829-44

[43] Asfour T., Azad P., Gyarfas F. and Dillmann R. (2008), “Imitation learning of dual-arm manipulation tasks in humanoid robots”, *Int. Journal of Humanoid Robotics*, v 5, n 2, p 183-202

[44] Iba S. (2004), Interactive Multi-Modal Robot Programming, doctoral dissertation, tech. report CMURI-TR-04-50, Robotics Institute, Carnegie Mellon University.

[45] Ikezawa K., Konishi Y., and Ishigaki H. (1997), “Development of a teaching system for an industrial robot using stereo vision”, *in Proc. of the SPIE*, v 3203, p 23-32.

[46] Ehrenmann M., Rogalla O., Zöllner R., and Dillmann R. (2001), “Teaching service robots complex tasks: programming by demonstration for workshop and household environments”, *Proc. of the IEEE Int. Conf. on Field and Service Robotics*, Finnland.

[47] Ehrenmann M., Zollner R., Rogalla O., and Dillmann, R. (2002), “Programming service tasks in household environments by human demonstration”, *Proc. of 11th IEEE Int. Workshop on Robot and Human Interactive Communication*, p 460-7.

[48] Chen J. and Zelinsky A. (2003), “Programming by demonstration: coping with suboptimal teaching actions”, *Int. J. of Robotics Research*, v 22, n 5, p 299-319.

[49] Breazeal C., Berlin M., Brooks A., Gray J. and Thomaz A.L. (2006), “Using perspective taking to learn from ambiguous demonstrations”, *Robot and Autonomous Systems*, v 54, n 5, p 385-93

[50] Knoop S., Pardowitz M., and Dillmann R. (2007), “Automatic robot programming from learned abstract task knowledge”, *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, p 1651-7.

[51] Knoop S., Pardowitz M., and Dillmann R. (2008), “From abstract task knowledge to executable robot programs”, *J. of Intelligent and Robotic Systems*, v 52, n 3-4, p 343-362.

[52] Calinon S. and Billard A. (2007), “Active teaching in robot programming by demonstration”, *16th IEEE Int. Conf. on Robot and Human Interactive Communication*, p 702-7

[53] Otero N., Saunders J., Dautenhahn K. and Nehaniv C. (2008), “Teaching robot companions: the role of scaffolding and event structuring”, *Connection Science*, v 20, n 2-3, p 111-34.

[54] Kofman J., Wu X., Luu T., and Verma S. (2005), "Teleoperation of A Robot Manipulator Using A Vision-Based Human-Robot Interface", *IEEE Trans. on Industrial Electronics*, v 52, n 5, p 1206-19.

[55] Kofman, J., Verma, S. and Wu, X. (2007). “Robot-Manipulator Teleoperation by Markerless Vision-Based Hand-Arm Tracking”. *Int. J. of Optomechatronics*, n 1(3), p 331-357.

[56] Kieg, J. C. “Motion Tracking: Polhemus Technology”, Virtual Reality Systems, vol. 1, No 1, pp. 32-46, March, 1993.

[57] Lieberman J. (2004), Teaching a Robot Manipulation Skills Through Demonstration, Masters of Science thesis, Dept. of Mechanical Engineering, MIT.

[58] Groves, D. (2007). Principles of GNSS, Inertial, and Multisensor Integrated Navigation Systems. Artech House Publishers.

[59] Field, M., Pan, Z., Stirling, D., and Naghdy, F., (2011). Human motion capture sensors and analysis in robotics. *Industrial Robotics*. Vol. 38. No. 2. pp. 163-171.

[60] Sigal, L., Balan, Al., and Black, M. (2009), “HumanEva: synchronized video and motion capture dataset and baseline algorithm for evaluation of articulated human motion”, *International Journal of Computer Vision*, Vol. 87, Nos 1-2, pp. 4-27.

[61] Li, W., Zhang, Z., and Liua, Z. (2008), “Expandable data-driven graphical modeling of human actions based on salient postures”, *IEEE Transactions on Circuits and systems for Video Technology,* Vol. 18, No. 11, pp. 1499-1510.

[62] Rosenhahn, B., Schmaltz, C., Brox, T., Weickert, J., Cremers, D., and Seidel, H.P. (2008), “Markerless motion capture of man-machine interaction”, *IEEE Conference on Computer vision and Patern Recognition, (CVPR 2008)*, Anchorage, AK, June 23-28, pp. 1-8.

[63] Moeslund, T.B., Hilton, A., and Krger, V. (2006), “A survey of advances in vision-based human motion capture and analysis”, *Computer Vision and Image Understanding,* Vol. 104, No. 2, pp. 90-126.

[64] Hu, W., Tan, T., Wang, L., and Maybank, S. (2004), “A survey on visual surveillance of object motion and behaviours, *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews,* Vol. 34, No. 3, pp.334-352.

[65] http://www.trivisio.com/. Last Accessed: August, 2011

[66] http://www.systron.com/. Last Accessed: August, 2011

[67] http://www.animazoo.com/. Last Accessed: August, 2011

[68] http://www.xsens.com/. Last Accessed: August, 2011

[69] http://www.xsens.com/en/news/entertainment-news/xsens-releases-uwb-rf-based-position-aiding-technology-to-enable-occlusion-free-real-time-driftless-multi-person-motion-capture-in-large-volumes . Last Accessed: August, 2011

[70] Miller, N.; Jenkins, O.C.; Kallmann, M.; Mataric, M.J. (2004) Motion capture from inertial sensing for untethered humanoid teleoperation. *4th IEEE/RAS International Conference on Humanoid Robots*, vol.2, no., pp. 547-565 Vol. 2, 10-12 Nov.

[71] Field, M.; Stirling, D.; Naghdy, F.; Pan, Z. (2008) Motion Segmentation for Humanoid control planning. *ARAA Australasian Conference on Robotics and Automation, 2008. ACRA '08.*, 3-5 December.

[72] Loke, Y.L., Gopalai, A.A, Khoo, B.h., Senanayake, S.M.N.A. (2009) Smart system for archery using ultrasound sensors. Advanced Intelligent Mechatronics, AIM 2009. IEEE/ASME International Conference on Digital Object Identifier. Pp. 1160-1164.

[73] http://www.hexamite.com/index.html. Last Accessed: August, 2011

[74] http://www.logitech.com/. Last Accessed: August, 2011

[75] http://www.vicon.com/. Last Accessed: August, 2011

[76] http://www.motionanalysis.com/. Last Accessed: August, 2011

[77] Dasgupta, A. and Nakamura, Y. (1999), "Making feasible walking motion of humanoid robots from human motion capture data", Proceedings of 1999 IEEE International Conference on Robotics and Automation, Vol. 2, pp. 1044-9.

[78] Shon, A.P., Grochow, K., Rao, R.P.N. (2005) Robotic imitation from human motion capture using Gaussian processes. *Proceedings of 2005 5th IEEE-RAS International Conference on Humanoid Robots*, v 2005, p 129-134

[79] L. Sigal and M. J. Black. HumanEva: Synchronized Video and Motion Capture Dataset for Evaluation of Articulated Human Motion, *Techniacl Report CS-06-08*, Brown University, 2006.

[80] L. Sigal, A. Balan and M. J. Black. HumanEva: Synchronized Video and Motion Capture Dataset and Baseline Algorithm for Evaluation of Articulated Human Motion, In *International Journal of Computer Vision*, Vol. 87 (1-2), 2010.

[81] http://vision.cs.brown.edu/humaneva/publications.html. Last Accessed: August, 2011

[82] http://www.motionanalysis.com/html/movement/publications.html. Last Accessed: August, 2011

[83] http://www.3rdtech.com/. Last Accessed: August, 2011

[84] Bishop, Gary. 1984. "The Self-Tracker: A Smart Optical Sensor on Silicon," UNC Chapel Hill Computer Science Dissertation TR84-002

[85] Greg Welch, Gary Bishop, Leandra Vicci, Stephen Brumback, Kurtis Keller, D'nardo Colucci. 1999. "The HiBall Tracker: High-Performance Wide-Area Tracking for Virtual and Augmented Environments," Proceedngs of the ACM Symposium on Virtual Reality Software and Technology 1999 (VRST 99), University College London, December 20-22, 1999.

[86] Greg Welch, B. Danette Allen, Adrian Ilie, and Gary Bishop, "Measurement Sample Time Optimization for Human Motion Tracking/Capture Systems," Proceedings of Trends and Issues in Tracking for Virtual Environments, Workshop at the IEEE Virtual Reality 2007 Conference (Charlotte, NC USA) (Gabriel Zachmann, ed.), Shaker, March 11 2007.

[87] http://www.ascension-tech.com/. Last Accessed: August, 2011

[88] http://www.ndigital.com/. Last Accessed: August, 2011

[89] http://www.ndigital.com/lifesciences/certus-motioncapturesystem.php. Last Accessed: August, 2011

[90] Nasksuk, N., Lee, G., Rietdyk, S. (2005) Whole-body human-to-humanoid motion transfer. *Proceedings of 2005 5th IEEE-RAS International Conference on Humanoid Robots*, v 2005, p 104-109

[91] Zhenning, L., Kulic, D. (2010). Particle Filter based motion tracking. Int. Conf. on Control Automation Robotics & Vision, ICARCV 2010, p 555-560

[92] http://www.organicmotion.com/. Last Accessed: August, 2011

[93] http://www.sarcos.com/. Last Accessed: August, 2011

[94] Calinon, S., Guenter, F., and Billard, A. (2007) On learning, representing and generalizing a task in a humanoid robot. *IEEE Transactions on Systems, Man and Cybernetics: Part B*, Vol. 37, No. 2, pp. 286-298.

[95] Ijspeert, A.J., Nakanishi, J. and Schaal, S. (2002), "Movement imitation with nonlinear dynamical systems in humanoid robots", Proceedings of the IEEE International Conference on Robotics and Automation, ICRA02, Washington, DC.

[96] Gray, J., Breazeal, C., Berlin, M., Brooks, A. and Lieberman, J. (2005), "Action parsing and goal inference using self as simulator", IEEE InternationalWorkshop on Robot and Human Interactive Communication, Nashville, TN, (ROMAN), 13-15. August, pp. 202-9.

[97] Ruiz-del-Solar, J., Palma-Amestoy, R., Marchant, R., Parra-Tsunekawa, I. and Zegers, P. (2009), "Learning to fall: designing low damage fall sequences for humanoid soccer robots, robotics and autonomous systems", *Robotics and Autonomous Systems*, v 57, n 8, p 796-807, 31 July 2009

[98] http://www.ascension-tech.com/index.php. Last Accessed: August, 2011

[99] http://www.polhemus.com/. Last Accessed: August, 2011

[100] http://www.vr-space.com/. Last Accessed: August, 2011

[101] Moldenhauer, J., Boesnach, I., Beth, T., Wank, V. and Bos, K. (2005), "Analysis of human motion for humanoid robots", Proceedings of the 2005 IEEE International Conference on Robotics and Automation. ICRA 2005, Barcelona, Spain, 18-22 April, pp. 311-6

[102] Matsunaga, T. and Oshita, M. (2007), "Recognition of walking motion using support vector machine", Proceedings of the ISICE2007, pp. 337-42.

[103] Ramana, P.K.R., Grest, D. and Volker, K. (2007), "Human action recognition in table-top scenarios: an HMM-based analysis to optimize the performance", Proceedings of 12th International Conference on Computer Analysis of Images and Patterns, CAIP 2007, Vienna, Austria, August 27-29, Lecture Notes in Computer Science, pp. 101-8.

[104] http://www.intersense.com/. Last Accessed: August, 2011

[105] Vlasic, D., Adelsberger, R., Vannucci, G., Barnwell, J., Gross, M., Matusik, W., Popović J., Practical motion capture in everyday surroundings. ACM Transactions on Graphics, 26(3), 2007, pp. 1-35

[106] Nguyen, K.D., Chen, I.-M., Yeo, S.H. and Duh, B.-L. (2007), "Motion control of a robotic puppet through a hybrid motion capture device", IEEE International Conference on Automation Science and Engineering, Rome, 22-25 September, pp. 753-8.

[107] Danisch, L. A., "Spatially continuous six-degrees-of-freedom position and orientation sensor", *Proc. SPIE Vol. 3541, Fiber Optic and Laser Sensors and Applications*, pp. 48-56, 1999.

[108] Ward, J.A., Lukowicz, P., Troster, G., and Starner, T. (2006) Activity recognition of assembly tasks using body-worn microphones and accelerometers. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol.28, no.10, pp.1553-1567

[109] Zhao, L. and Badler, N.I. (2005), "Acquiring and validating motion qualities from live limb gestures", Graphical Models, Vol. 67 No. 1, pp. 1-16.

[110] http://www.nintendo.com/. Last Accessed: August, 2011

[111] http://us.playstation.com/ps3/playstation-move/. Last Accessed: December, 2011

[112] http://wiibrew.org/wiki/wiimote. Last Accessed: August, 2011

[113] http://www.analog.com/static/imported-files/data_sheets/ADXL330.pdf. Last Accessed: August, 2011

[114] http://wiibrew.org/wiki/wiimote/Extension_Controllers#Wii_Motion_Plus. Last Accessed: August, 2011

[115] Shuster, M.D. "A Survey of Attitude representations", Journal of the Astronautical Sciences, vol. 41, no. 4, pp. 429-517, October-December 1993.

[116] Trawny, N. and Roumeliotis, S.I. (2005). *Indirect Kalman Filter for 3D Attitude Estimation*. Technical Report 2005-002, Rev. 57, University of Minnesota, Department of Computer Science and Engineering, MARS Lab.

[117] Novi Commentarii academiae scientiarum Petropolitanae 20, 1776, pp. 189–207 (E478)

[118] Hamilton, W.R., (1866) Elements of Quaternions, Longmans, Green, and Co., London.

[119] Breckenridge, W.G. "Quaternions – Proposed Standard Conventions," JPL, Tech. Rep. INTEROFFICE MEMORANDUM IOM343-79-1199, 1999.

[120] Maybeck, P.S., Stochastic Models, Estimation and Control. New York: Academic Press, 1979, vol. 1.

[121] Farrenkopf, R.L., "Analytic Steady-State Accuracy Solutions for Two Common Spacecraft Attitude Estimators," Journal of Guidance and Control, Vol. 1, 1978. Pp 282-284

[122] Spong, M.W., Hutchinson, S., Vidyasager, M. (2005) Robot Modeling and Control. Wiley

[123] Bradski, G, and Kaehler, A. "Learning OpenCV". O'Reilly, 2008.

[124] Emanuele Trucco, Alessandro Verri, "Introductory Techniques for 3-D Computer Vision", Prentice Hall, 1998.

[125] Kalman, R.E., "A new Approach to Linear Filtering and Prediction Problems," *Transactions of the ASME, Seies D, Journal of Basic Engineering*, Vol. 82, pp. 35-45, 1960.

[126] G. Bradski and A. Kaebler. *Learning OpenCV*. O'Reilly Media. 2008

[127] G. Welch and G. Bishop. *An introduction to the Kalman filter*. available at http://www.cs.unc.edu/welch/kalman/kalmanIntro.html. 2004

[128] Fundamentals of Kalman Filtering – A practical approach, P. Zarchan & H. Musoff

[129] Lefferts, E.J., Markley, F.L., Shuster, M.D. Kalman Filtering for Spacecraft Attitude Estimation. AIAA 20th Aerospace Sciences Meeting. Jan 11-14, 1982.

[130] Crassidis, J.L., Markley, F.L., and Cheng, Y., "A Survey of Nonlinear Attitude Estimation Methods," AIAA Journal of Guidance, Control, and Dynamics, Vol. 30, No. 1, Jan.-Feb. 2007, pp. 12-28.

[131] Stuelpnagel, J., "On the Parameterization of the Three-Dimensional Rotation Group", SIAM Review, Vol. 6, Oct. 1964, pp. 422-430

[132] Markley, F.L., “Parameterization of the Attitude”, in Spacecraft Attitude Determination and Control, J.R. Werts, ed., D. Reidel, Dorrecht, Holland, 1978

[133] Pauling, D.C., Jackson, D.b., and Brown, C.D., “SPARS Algorithms and Simulation Results”, ibid., pp. 293-317.

[134] Edwards, A., “The State of Strapdown Inertial Guidance and Navigation”, Navigation, Vol. 18, Winter 1971-72, pp. 286-401

[135] Wilcox, J.C., “A New Algorithm for Strapped-Down Inertial Navigation”, IEEE Transactions on Aerospace and Electronic Systems, Vol. AES-3, 1967 pp. 796-802.

[136] Giardina, C.R., Bronson, R., and Wallen, L., “An Optimal Normalization Scheme”, IEEE Transactions on Aerospace and Electronic Systems, Vol. AES-11, July 1975, pp. 443-446.

[137] Farrell, J. L., “Attitude Determination by Kalman Filtering,” *Automatica*, Vol. 6, pp. 419-430, 1970

[138] Pittelkau, M.E., “Spacecraft Attitude Determination Using the Bortz Equation,” *AAS Paper 01-310, AAS/AIAA Astrodynamics Specialist Conference*, Quebec City, Quebec, August 2001.

[139] Stuelpnagel, J., “On the Parameterization of the Three Dimensional Rotation Group”, *SIAM Review*, Vol. 6, No. 4, 422-430, 1964

[140] BarItzhack, I.Y., and Oshman, Y.,”Attitude Determination from Vector Observations: Quaternion Estimation,“ *IEEE Transactions on Aerospace, and Electron Systems*, Vol. AES-21, No. 1, pp. 128-135, 1985

[141] Lefferets, E. J., Markley, F.L., and Shuster, M. D., “Kalman Filtering for Spacecraft Attitude Estimation,“ *Journal of Guidance, Control, and Dynamics*, Vol. 5, No. 5, pp. 417-239, 1982

[142] Bar-Itzhack, I.Y. Deutschmann, J., and Markley, F.L., “Quaternion Normalization in Additive EKF for Spacecraft Attide Determination,” *AIAA Paper 91-2706, AIAA Guidance, Navigationand Control Conference*, New Orleans, LA, August 1991

[143] Deutschmann, J., Markley, F.L. and Bar-Itzhack, I., “Quaternion Normalization in Spacecraft Attidue Determination,” *Flight Mechanics/Estimation Theory Symposium, NASA Conference Publication 3186*, Goddard Space Flight Center, Greenbelt, MD, May 1992

[144] Shuster, M.D., “The Quaternoin in the Kalman Filter,” *Astrodynamics 1993, Advances in the Astronautical Science*, Vol. 85, pp. 25-37, 1993

[145] Markley, F.L. 2003. “Attitude Error Representation for Kalman Filtering”, *Journal of Guidance, Control and Dynamics*, Vol. 26. No. 2, pp. 311-317

[146] Trawny, N. and Roumeliotis, S.I. (2005). *Indirect Kalman Filter for 3D Attitude Estimation*. Technical Report 2005-002, Rev. 57, University of Minnesota, Department of Computer Science and Engineering, MARS Lab.

[147] Maybeck, P.S., *Stochastic Models, Estimation and Control*. New York: Academic Press, 1979, Vol. 1.

[148] Wertz, J. R. Ed., Spacecraft Attidude Determination and Control. Dordrecht; Boston: Kluwer Academic, 1979

[149] gl.tter. (2007-2010) WiiYourself! Native C++ Wiimote Library v1.15. gl.tter.org. [Online]. Available: http://gl.tter.org. Last Accessed: August, 2011

[150] Strassen, V., (1969) Gaussian elimination is not optimal. Numer. Math. 13. Pages 354-356.

[151] Coppersmith, D. & Winograd, S. (1990), Matrix multiplication via arithmetic progressions, Journal of Symbolic Computation 9 (3): 251–280

[152] Higham, N. (1990). Exploiting Fast Matrix Multiplication Within the Level 3 Blas. ACM Transactions on Mathematical Software, Vol. 16, No. 4, Pages 352-368.

[153] WiiBrew. *Wiimote*. http://wiibrew.org/wiki/Wiimote. Last Accessed: Nov. 2011.

[154] Panasonic LN162S IR LED Datasheet. http://www.datasheetcatalog.org/datasheet/panasonic/SHC00002AED.pdf. Last Accessed: August, 2011

[155] Bradski, G, and Kaehler, A. “Learning OpenCV”. O'Reilly, 2008.

[156] Emanuele Trucco, Alessandro Verri, "Introductory Techniques for 3-D Computer Vision", Prentice Hall, 1998.

[157] Won, S.P., Golnaraghi, F. (2010) A Triaxial Accelerometer Calibration Method Using a Mathematical Model. IEEE Transactions of Instrumentation and Measurements, Vol. 59, No. 8.

[158] Trawny, N. and Roumeliotis, S.I. (2005). *Indirect Kalman Filter for 3D Attitude Estimation*. Technical Report 2005-002, Rev. 57, University of Minnesota, Department of Computer Science and Engineering, MARS Lab.

[159] Wertz, J. R. Ed., Spacecraft Attitude Determination and Control. Dordrecht; Boston: Kluwer Academic, 1979

Appendix A – Mathematics Properties

A.1 – Quaternion Multiplication Identities

This section presents relevant mathematical properties. For a more complete list, see [158].

It is useful to note that the matrix $\boldsymbol{\mathcal{L}}(\bar{q})$ from (4.18) can be expressed in the form

$$\boldsymbol{\mathcal{L}}(\bar{q}) = [\boldsymbol{\Psi}(\bar{q}) \quad \bar{q}] \tag{A.1}$$

where

$$\boldsymbol{\Psi}(\bar{q}) = \begin{bmatrix} q_4\mathbf{I}_{3\times 3} - [\mathbf{q}\times] \\ -\mathbf{q}^{\mathrm{T}} \end{bmatrix} \tag{A.2}$$

with the property

$$\boldsymbol{\Psi}^{\mathrm{T}}\boldsymbol{\Psi} = \mathbf{I}_{3\times 3} \tag{A.3}$$

Analogously, the matrix $\boldsymbol{\mathcal{R}}(\bar{p})$ from (4.19) can be expressed equivalently

$$\boldsymbol{\mathcal{R}}(\bar{p}) = [\boldsymbol{\Xi}(\bar{p}) \quad \bar{p}] \tag{A.4}$$

where

$$\boldsymbol{\Xi}(\bar{p}) = \begin{bmatrix} p_4\mathbf{I}_{3\times 3} + [\mathbf{p}\times] \\ -\mathbf{p}^{\mathrm{T}} \end{bmatrix} \tag{A.5}$$

with the property

$$\boldsymbol{\Xi}^{\mathrm{T}}\boldsymbol{\Xi} = \mathbf{I}_{3\times 3} \tag{A.6}$$

A.2 – Properties of the Cross Product Matrix $[\boldsymbol{\omega}\times]$

Anti-Commutativity

$$[\boldsymbol{\omega}\times] = -[\boldsymbol{\omega}\times]^{\mathrm{T}} \tag{A.7}$$

$$[\boldsymbol{\alpha}\times]\boldsymbol{\beta} = -[\boldsymbol{\beta}\times]\boldsymbol{\alpha} \tag{A.8}$$

$$\boldsymbol{\alpha}^{\mathrm{T}}[\boldsymbol{\beta}\times] = -\boldsymbol{\beta}^{\mathrm{T}}[\boldsymbol{\alpha}\times] \tag{A.9}$$

For a complete list, see [158].

A.3 – Powers of the Cross Product Matrix $[\boldsymbol{\omega}\times]$

$$[\boldsymbol{\omega}\times]^2 = \boldsymbol{\omega}\boldsymbol{\omega}^{\mathrm{T}} - |\boldsymbol{\omega}|^2 \cdot \mathbf{I}_{3\times 3} \tag{A.10}$$

$$\begin{aligned}[\boldsymbol{\omega}\times]^3 &= \left(\boldsymbol{\omega}\boldsymbol{\omega}^{\mathrm{T}} - |\boldsymbol{\omega}|^2 \cdot \mathbf{I}_{3\times 3}\right)[\boldsymbol{\omega}\times] \\ &= \boldsymbol{\omega}\boldsymbol{\omega}^{\mathrm{T}}[\boldsymbol{\omega}\times] - |\boldsymbol{\omega}|^2 \cdot [\boldsymbol{\omega}\times] \\ &= \boldsymbol{\omega}(-[\boldsymbol{\omega}\times]\boldsymbol{\omega}) - |\boldsymbol{\omega}|^2 \cdot [\boldsymbol{\omega}\times] \\ &= -\boldsymbol{\omega}\left(\underbrace{\boldsymbol{\omega}\times\boldsymbol{\omega}}_{0}\right) - |\boldsymbol{\omega}|^2 \cdot [\boldsymbol{\omega}\times]\end{aligned} \tag{A.11}$$

$$[\boldsymbol{\omega}\times]^3 = -|\boldsymbol{\omega}|^2 \cdot [\boldsymbol{\omega}\times] \tag{A.12}$$

$$\begin{aligned}[\boldsymbol{\omega}\times]^4 &= [\boldsymbol{\omega}\times]^3[\boldsymbol{\omega}\times] \\ &= -|\boldsymbol{\omega}|^2 \cdot [\boldsymbol{\omega}\times]^2\end{aligned} \tag{A.13}$$

$$\begin{aligned}[\boldsymbol{\omega}\times]^5 &= -|\boldsymbol{\omega}|^3 \cdot [\boldsymbol{\omega}\times]^2 \\ &= -|\boldsymbol{\omega}|^2 \cdot [\boldsymbol{\omega}\times]\left(\boldsymbol{\omega}\boldsymbol{\omega}^{\mathrm{T}} - |\boldsymbol{\omega}|^2 \cdot \mathbf{I}_{3\times 3}\right) \\ &= +|\boldsymbol{\omega}|^4 \cdot [\boldsymbol{\omega}\times]\end{aligned} \tag{A.14}$$

$$[\boldsymbol{\omega}\times]^6 = +|\boldsymbol{\omega}|^4 \cdot [\boldsymbol{\omega}\times]^2 \tag{A.15}$$

$$[\boldsymbol{\omega}\times]^7 = -|\boldsymbol{\omega}|^6 \cdot [\boldsymbol{\omega}\times] \tag{A.16}$$

$$[\boldsymbol{\omega}\times]^8 = -|\boldsymbol{\omega}|^6 \cdot [\boldsymbol{\omega}\times]^{\mathbf{2}} \tag{A.17}$$

A.4 – Properties of the Ω Matrix

The Ω matrix is a useful matrix in quaternion algebra and calculus and is defined as

$$\boldsymbol{\Omega}(\boldsymbol{\omega}) = \begin{bmatrix} 0 & \omega_z & -\omega_y & \omega_x \\ -\omega_z & 0 & \omega_x & \omega_y \\ \omega_y & -\omega_x & 0 & \omega_z \\ -\omega_x & -\omega_y & -\omega_z & 0 \end{bmatrix} \tag{A.18}$$

which can be reduced to and used to calculate the powers of $\boldsymbol{\Omega}$ with use of (A.10)-(A.17)

$$\boldsymbol{\Omega}(\boldsymbol{\omega}) = \begin{bmatrix} -[\boldsymbol{\omega}\times] & \boldsymbol{\omega} \\ -\boldsymbol{\omega}^{\mathrm{T}} & 0 \end{bmatrix} \tag{A.19}$$

$$\boldsymbol{\Omega}(\boldsymbol{\omega})^2 = \begin{bmatrix} [\boldsymbol{\omega}\times]^2 - \boldsymbol{\omega}\boldsymbol{\omega}^{\mathrm{T}} & -[\boldsymbol{\omega}\times]\boldsymbol{\omega} \\ \boldsymbol{\omega}^{\mathrm{T}}[\boldsymbol{\omega}\times] & -\boldsymbol{\omega}^{\mathrm{T}}\boldsymbol{\omega} \end{bmatrix} \tag{A.20}$$
$$= -|\boldsymbol{\omega}|^2 \cdot \mathbf{I}_{4\times 4}$$

$$\boldsymbol{\Omega}(\boldsymbol{\omega})^3 = -|\boldsymbol{\omega}|^2 \cdot \boldsymbol{\Omega}(\boldsymbol{\omega}) \tag{A.21}$$

$$\boldsymbol{\Omega}(\boldsymbol{\omega})^4 = |\boldsymbol{\omega}|^4 \cdot \mathbf{I}_{4\times 4} \tag{A.22}$$

$$\boldsymbol{\Omega}(\boldsymbol{\omega})^5 = |\boldsymbol{\omega}|^4 \cdot \boldsymbol{\Omega}(\boldsymbol{\omega}) \tag{A.23}$$

$$\boldsymbol{\Omega}(\boldsymbol{\omega})^6 = -|\boldsymbol{\omega}|^6 \cdot \mathbf{I}_{4\times 4} \tag{A.24}$$

A.5 – Small-Angle Approximation

When angles are very small, the transcendental trigonometric functions can be reasonably approximated by taking the first-order approximation of the Taylor-series expansion leading to

$$\sin(x) \approx x \quad \cos(x) \approx 1 \tag{A.25}$$

Note that for small angles (e.g. the error angles $\delta\boldsymbol{\theta}$ of Sect. 5.5.4**),** this is a reasonably accurate approximation.

Appendix B – Mathematical Derivations

B.1 – Zeroth-Order Integrator

The zeroth-order quaternion integrator is used to solve propagation equations of the form

$$\bar{q}_{\mathrm{k}} = \boldsymbol{\Phi}_k^{\bar{q}} \bar{q}_{\mathrm{k-1}} \tag{B.1}$$

Since the general solution to (B.1) does not currently exist in the literature, the problem needs to be simplified. One method of solving this is to use the zeroth-order hold, which is the assumption that a zeroth-order (linear) approximation can be used on the interval between time steps. It can be used to simplify difference functions prior to integration, when the integral of the function is either difficult or does not exist.

Using the zeroth-order hold, following [146][147], $\boldsymbol{\Phi}_k^{\bar{q},0}$ can be obtained by

$$\boldsymbol{\Phi}_k^{\bar{q},0} = \boldsymbol{\Phi}^{\bar{q},0}(\mathrm{t}_k, \mathrm{t}_{k-1}) = \boldsymbol{\Phi}^{\bar{q},0}(\Delta t) = \mathrm{e}^{\left(\frac{1}{2}\boldsymbol{\Omega}(\boldsymbol{\omega})\Delta t\right)} \tag{B.2}$$

taking the Taylor expansion of (B.2)

$$\boldsymbol{\Phi}_k^{\bar{q},0} = \sum_{i=0}^{\infty} \frac{\left(\frac{1}{2}\boldsymbol{\Omega}(\boldsymbol{\omega}_k)\Delta t\right)^i}{i!} \tag{B.3}$$

$$= \mathbf{I}_{4\times4} + \frac{1}{2}\boldsymbol{\Omega}(\boldsymbol{\omega}_k)\Delta t + \frac{1}{2!}\left(\frac{1}{2}\boldsymbol{\Omega}(\boldsymbol{\omega}_k)\Delta\mathrm{t}\right)^2 + \frac{1}{3!}\left(\frac{1}{2}\boldsymbol{\Omega}(\boldsymbol{\omega}_k)\Delta t\right)^3 + \cdots \tag{B.4}$$

With the properties in Sect. A.4, Eq. (B.4) can be expanded to

$$\begin{aligned}\boldsymbol{\Phi}_k^{\bar{q},0} = \mathbf{I}_{4\times4} &+ \frac{1}{2}\boldsymbol{\Omega}(\boldsymbol{\omega}_k)\Delta t + \frac{1}{2!}\left(\frac{1}{2}\Delta\mathrm{t}\right)^2 |\boldsymbol{\omega}_k|^2 \cdot \mathbf{I}_{4\times4} \\ &- \frac{1}{3!}\left(\frac{1}{2}\Delta\mathrm{t}\right)^3 |\boldsymbol{\omega}_k|^2 \cdot \boldsymbol{\Omega}(\boldsymbol{\omega}_k) + \frac{1}{4!}\left(\frac{1}{2}\Delta\mathrm{t}\right)^4 |\boldsymbol{\omega}_k|^4 \cdot \mathbf{I}_{4\times4} \\ &+ \frac{1}{5!}\left(\frac{1}{2}\Delta\mathrm{t}\right)^5 |\boldsymbol{\omega}_k|^4 \cdot \boldsymbol{\Omega}(\boldsymbol{\omega}_k) - \frac{1}{6!}\left(\frac{1}{2}\Delta\mathrm{t}\right)^6 |\boldsymbol{\omega}_k|^6 \cdot \mathbf{I}_{4\times4} - \cdots\end{aligned} \tag{B.5}$$

By regrouping $\mathbf{I}_{4\times4}$ and $\boldsymbol{\Omega}(\boldsymbol{\omega}_k)$ matrices, and including the $|\boldsymbol{\omega}_k|$ term into the preceding brackets

$$\boldsymbol{\Phi}_k^{\bar{q},0} = \left[\left(1 - \frac{1}{2!}\left(\frac{1}{2}|\boldsymbol{\omega}_k|\Delta\mathrm{t}\right)^2 + \frac{1}{4!}\left(\frac{1}{2}|\boldsymbol{\omega}_k|\Delta\mathrm{t}\right)^4 - \frac{1}{6!}\left(\frac{1}{2}|\boldsymbol{\omega}_k|\Delta\mathrm{t}\right)^6 + \cdots\right) \cdot \mathbf{I}_{4\times4}\right. \tag{B.6}$$

$$+\frac{1}{|\boldsymbol{\omega}|}\left(\frac{1}{2}|\boldsymbol{\omega}_k|\Delta t-\frac{1}{3!}\left(\frac{1}{2}|\boldsymbol{\omega}_k|\Delta t\right)^3+\frac{1}{5!}\left(\frac{1}{2}|\boldsymbol{\omega}_k|\Delta t\right)^5-\cdots\right)\cdot\boldsymbol{\Omega}(\boldsymbol{\omega}_k)\Bigg]$$

By noticing that (B.6) contains the Taylor expansion of the cosine and sine functions, the zeroth-order quaternion transition matrix $\boldsymbol{\Phi}_k$ can be reduced to

$$\boldsymbol{\Phi}_k^{\bar{q},0}=\left[\cos\left(\frac{|\boldsymbol{\omega}_k|}{2}\Delta t\right)\cdot\mathbf{I}_{4\times4}+\frac{1}{|\boldsymbol{\omega}_k|}\sin\left(\frac{|\boldsymbol{\omega}_k|}{2}\Delta t\right)\cdot\boldsymbol{\Omega}(\boldsymbol{\omega})\right] \tag{B.7}$$

When $|\boldsymbol{\omega}|$ is very small, (B.7) will lead to numerical instability by repeated integration using (B.1). To correct this, the limit of (B.7) as $|\boldsymbol{\omega}|$ goes to zero is taken

$$\lim_{|\boldsymbol{\omega}_k|\to0}\boldsymbol{\Phi}_k^{\bar{q},0}=\lim_{|\boldsymbol{\omega}_k|\to0}\left[\cos\left(\frac{|\boldsymbol{\omega}_k|}{2}\Delta t\right)\cdot\mathbf{I}_{4\times4}+\frac{1}{|\boldsymbol{\omega}_k|}\sin\left(\frac{|\boldsymbol{\omega}_k|}{2}\Delta t\right)\cdot\boldsymbol{\Omega}(\boldsymbol{\omega}_k)\right] \tag{B.8}$$

Applying L'Hopital's Rule

$$\lim_{|\boldsymbol{\omega}_k|\to0}\boldsymbol{\Phi}_k^{\bar{q},0}=\mathbf{I}_{4\times4}+\lim_{|\boldsymbol{\omega}_k|\to0}\left[\cos\left(\frac{|\boldsymbol{\omega}_k|}{2}\Delta t\right)\left(\frac{\Delta t}{2}\right)\cdot\boldsymbol{\Omega}(\boldsymbol{\omega}_k)\right] \tag{B.9}$$

$$\lim_{|\boldsymbol{\omega}_k|\to0}\boldsymbol{\Phi}_k^{\bar{q},0}=\mathbf{I}_{4\times4}+\frac{1}{2}\boldsymbol{\Omega}(\boldsymbol{\omega}_k)\Delta t \tag{B.10}$$

Instead of using the matrix integration form (B.1), it is sometimes useful to use the quaternion integration form

$${}^{\mathrm{L}}_{\mathrm{G}}\bar{q}_{\mathrm{k}}^{-}=\bar{\Phi}_k^{\bar{q},0}\otimes{}^{\mathrm{L}}_{\mathrm{G}}\bar{q}_{\mathrm{k}-1}^{+} \tag{B.11}$$

Expanding (B.7), and setting $q_i=\frac{1}{|\boldsymbol{\omega}_k|}\sin\left(\frac{|\boldsymbol{\omega}_k|}{2}\Delta t\right)$ and $q_4=\cos\left(\frac{|\boldsymbol{\omega}_k|}{2}\Delta t\right)$ for clarity

$$\boldsymbol{\Phi}_k^{\bar{q},0}=\begin{bmatrix}q_4 & q_i\omega_z & -q_i\omega_y & q_i\omega_x\\ -q_i\omega_z & q_4 & q_i\omega_x & q_i\omega_y\\ q_i\omega_y & -q_i\omega_x & q_4 & q_i\omega_z\\ -q_i\omega_x & -q_i\omega_y & -q_i\omega_z & q_4\end{bmatrix}=\mathcal{L}(\bar{\Phi}_k) \tag{B.12}$$

Finally, by using (4.20), (B.12), and by factoring out the terms, the zeroth-order quaternion, the transition quaternion $\bar{\Phi}_k^{\bar{q},0}$ used in (B.11) is obtained by

$$\bar{\Phi}_k^{\bar{q},0}=\begin{bmatrix}\frac{\boldsymbol{\omega}_k}{|\boldsymbol{\omega}_k|}\cdot\sin\left(\frac{|\boldsymbol{\omega}_k|}{2}\Delta t\right)\\ \cos\left(\frac{|\boldsymbol{\omega}_k|}{2}\Delta t\right)\end{bmatrix} \tag{B.13}$$

B.2 – First-Order Integrator

The first-order quaternion integrator is used to solve propagation equations of the form (B.1). It is a first-order approximation method, which is a more accurate expansion of the zeroth-order integrator and is derived by elaborating on the method of [159]. The first-order hold assumes that a first-order (linear) approximation can be used on the interval between time steps, and is used in-place of the zeroth-order hold to provide greater accuracy.

Defining the angular acceleration $\dot{\boldsymbol{\omega}}_k$, which is constant with the first-order hold assumption,

$$\dot{\boldsymbol{\omega}}_k \approx \frac{\boldsymbol{\omega}_{k+1} - \boldsymbol{\omega}_k}{\Delta t} \tag{B.14}$$

It is also useful to note the time derivative of $\boldsymbol{\Omega}(\boldsymbol{\omega}_k)$ is

$$\dot{\boldsymbol{\Omega}}(\boldsymbol{\omega}_k) = \boldsymbol{\Omega}(\dot{\boldsymbol{\omega}}_k) \tag{B.15}$$

Additionally, since $\dot{\boldsymbol{\omega}}_k$is constant

$$\left.\begin{aligned} \ddot{\boldsymbol{\omega}}_k = \dddot{\boldsymbol{\omega}}_k = \cdots = \mathbf{0}_{3\times 1} \\ \boldsymbol{\Omega}(\ddot{\boldsymbol{\omega}}_k) = \boldsymbol{\Omega}(\dddot{\boldsymbol{\omega}}_k) = \cdots = \mathbf{0}_{4\times 4} \end{aligned}\right\} \tag{B.16}$$

The average angular velocity during a time step is defined as

$$\bar{\boldsymbol{\omega}}_k = \frac{\boldsymbol{\omega}_k + \boldsymbol{\omega}_{k-1}}{2} \tag{B.17}$$

The average $\boldsymbol{\Omega}(\bar{\boldsymbol{\omega}}_k)$ matrix can be defined as

$$\boldsymbol{\Omega}(\bar{\boldsymbol{\omega}}_k) = \frac{1}{\Delta t}\int_{t_{k-1}}^{t_k} \boldsymbol{\omega}\,(\tau)\,\mathrm{d}\tau = \boldsymbol{\Omega}(\boldsymbol{\omega}_{k-1}) + \frac{1}{2}\boldsymbol{\Omega}(\dot{\boldsymbol{\omega}}_{k-1})\Delta t \tag{B.18}$$

$$\Leftrightarrow \boldsymbol{\Omega}(\boldsymbol{\omega}_{k-1}) = \boldsymbol{\Omega}(\bar{\boldsymbol{\omega}}_k) - \frac{1}{2}\boldsymbol{\Omega}(\dot{\boldsymbol{\omega}}_{k-1})\Delta t \tag{B.19}$$

where $t_k = t_{k-1} + \Delta t$.

To begin, $\bar{q}_{\mathrm{k}}^{-}$ is expanded by a Taylor series about time t_{k-1}

$$\bar{q}_k = \bar{q}_{k-1} + \dot{\bar{q}}_{k-1}\Delta t + \frac{1}{2!}\ddot{\bar{q}}_{k-1}\Delta t^2 + \frac{1}{3!}\dddot{\bar{q}}_{k-1}\Delta t^3 + \cdots \tag{B.20}$$

where repeated use of (4.39) and (B.14)-(B.16) leads to

$$\left.\begin{aligned}
\dot{\bar{q}}_{k-1} &= \frac{1}{2}\boldsymbol{\Omega}(\boldsymbol{\omega}_{k-1})\bar{q}_{k-1} \\
\ddot{\bar{q}}_{k-1} &= \frac{1}{2}\boldsymbol{\Omega}(\dot{\boldsymbol{\omega}}_{k-1})\bar{q}_{k-1} + \frac{1}{4}\boldsymbol{\Omega}(\boldsymbol{\omega}_{k-1})\bar{q}_{k-1} \\
\dddot{\bar{q}}_{k-1} &= \frac{1}{2}\boldsymbol{\Omega}(\dot{\boldsymbol{\omega}}_{k-1})\,\boldsymbol{\Omega}(\boldsymbol{\omega}_{k-1})\bar{q}_{k-1} + \frac{1}{4}\boldsymbol{\Omega}(\boldsymbol{\omega}_{k-1})\boldsymbol{\Omega}(\dot{\boldsymbol{\omega}}_{k-1})\bar{q}_{k-1} + \frac{1}{8}\boldsymbol{\Omega}(\boldsymbol{\omega}_{k-1})^3\bar{q}_{k-1} \\
\vdots \qquad & \qquad\qquad \vdots
\end{aligned}\right\} \tag{B.21}$$

Using (B.20), (B.21) and regrouping terms

$$\begin{aligned}
\bar{q}_k = &\left(\mathbf{I}_{4\times4} + \frac{1}{2}\boldsymbol{\Omega}(\boldsymbol{\omega}_{k-1})\Delta t + \frac{1}{2!}\left(\frac{1}{2}\boldsymbol{\Omega}(\boldsymbol{\omega}_{k-1})\Delta\mathrm{t}\right)^2 + \frac{1}{3!}\left(\frac{1}{2}\boldsymbol{\Omega}(\boldsymbol{\omega}_{k-1})\Delta t\right)^3 + \cdots\right)\bar{q}^{+}_{k-1} \\
&+ \frac{1}{4}\Delta t^2\,\boldsymbol{\Omega}(\dot{\boldsymbol{\omega}}_{k-1})\bar{q}^{+}_{k-1} + \left(\frac{1}{12}\boldsymbol{\Omega}(\dot{\boldsymbol{\omega}}_{k-1})\boldsymbol{\Omega}(\boldsymbol{\omega}_{k-1}) - \frac{1}{24}\boldsymbol{\Omega}(\boldsymbol{\omega}_{k-1})\boldsymbol{\Omega}(\dot{\boldsymbol{\omega}}_{k-1})\right)\Delta t^3\bar{q}^{+}_{k-1} + \cdots
\end{aligned} \tag{B.22}$$

With use of (B.19), Eq. (B.22) can be rewritten as

$$\begin{aligned}
\bar{q}_k = &\left(\mathbf{I}_{4\times4} + \tfrac{1}{2}\boldsymbol{\Omega}(\bar{\boldsymbol{\omega}}_k)\Delta t + \tfrac{1}{2!}\left(\tfrac{1}{2}\boldsymbol{\Omega}(\bar{\boldsymbol{\omega}}_k)\Delta\mathrm{t}\right)^2 + \tfrac{1}{3!}\left(\tfrac{1}{2}\boldsymbol{\Omega}(\bar{\boldsymbol{\omega}}_k)\Delta t\right)^3 + \cdots\right. \\
&\left. + \tfrac{1}{48}\left(\boldsymbol{\Omega}(\dot{\boldsymbol{\omega}}_{k-1})\boldsymbol{\Omega}(\boldsymbol{\omega}_{k-1}) - \boldsymbol{\Omega}(\boldsymbol{\omega}_{k-1})\boldsymbol{\Omega}(\dot{\boldsymbol{\omega}}_{k-1})\right)\Delta t^3 + \cdots\right)\bar{q}^{+}_{k-1}
\end{aligned} \tag{B.23}$$

Noticing that the first line is the Taylor series expansion of $\frac{1}{2}\boldsymbol{\Omega}(\bar{\boldsymbol{\omega}}_k)\Delta t$, (B.23) is reduced to

$$\bar{q}_k = \left(\mathrm{e}^{\left(\frac{1}{2}\boldsymbol{\Omega}(\bar{\boldsymbol{\omega}}_k)\Delta t\right)} + \frac{1}{48}\left(\boldsymbol{\Omega}(\dot{\boldsymbol{\omega}}_{k-1})\boldsymbol{\Omega}(\boldsymbol{\omega}_{k-1}) - \boldsymbol{\Omega}(\boldsymbol{\omega}_{k-1})\boldsymbol{\Omega}(\dot{\boldsymbol{\omega}}_{k-1})\right)\Delta t^3 + \cdots\right)\bar{q}^{+}_{k-1} \tag{B.24}$$

Since the integrator is of the form

$$\bar{q}_k = \boldsymbol{\Phi}_k^{\bar{q},1}\bar{q}^{+}_{k-1} \tag{B.25}$$

by neglecting higher order terms, the first-order quaternion integrator $\boldsymbol{\Phi}_k^{\bar{q},1}$ is obtained by

$$\boldsymbol{\Phi}_k^{\bar{q},1} \approx \mathrm{e}^{\left(\frac{1}{2}\boldsymbol{\Omega}(\bar{\boldsymbol{\omega}}_k)\Delta t\right)} + \frac{1}{48}\left(\boldsymbol{\Omega}(\dot{\boldsymbol{\omega}}_{k-1})\boldsymbol{\Omega}(\boldsymbol{\omega}_{k-1}) - \boldsymbol{\Omega}(\boldsymbol{\omega}_{k-1})\boldsymbol{\Omega}(\dot{\boldsymbol{\omega}}_{k-1})\right)\Delta t^3 \tag{B.26}$$

which can be presented in a compact form with the use of (B.2)

$$\boldsymbol{\Phi}_k^{\bar{q},1} \approx \boldsymbol{\Phi}_k^{\bar{q},0}(\boldsymbol{\omega}_k = \bar{\boldsymbol{\omega}}_k) + \frac{1}{48}\left(\boldsymbol{\Omega}(\dot{\boldsymbol{\omega}}_{k-1})\boldsymbol{\Omega}(\boldsymbol{\omega}_{k-1}) - \boldsymbol{\Omega}(\boldsymbol{\omega}_{k-1})\boldsymbol{\Omega}(\dot{\boldsymbol{\omega}}_{k-1})\right)\Delta t^3 \tag{B.27}$$