Multi-Level Information Fusion for Environment aware Robotic Navigation

---

Submitted as a requirement for the degree of
Master of Engineering

Michael Shiel

BEng (Hons), GCRC

Robotics and Aerospace Systems

School of Electrical Engineering and Computer Science

Science and Engineering Faculty

August 13, 2013

Abstract

This thesis aims to develop the hardware and software framework for an integrated navigation system using a variety of sensors with inertial navigation as the basis. However, any real-world sensor is susceptible to a variety of errors, such as GPS signals being obstructed and the resultant loss of positional information, or the drift and temperature sensitivity inherent in MEMS inertial sensors. A dynamic fusion algorithm will be used to develop a system with a high level of resistance to the typical problems that affect standard navigation systems.

Current robotic systems are complex combinations of multiple independent systems. This increasing complexity leads to higher costs, longer development times, and difficult usage. This thesis proposes the usage of commodity hardware in the form of an Android smartphone as the basis of an inertial navigation system. Combined with this low-cost hardware, a novel filtering methodology is adopted, allowing the usage of low-end sensors while achieving a level of performance rivalling that of sensors up to an order of magnitude more costly.

Specifically, this thesis details the implementation of an inertial navigation system (INS) on a Samsung Nexus S Android powered smartphone using a cascaded kalman filter architecture. This filtering architecture is combined with a Microsoft Kinect 3D scanning sensor. Using the Kinect in tandem with the phone's built in GPS sensor, the system is capable of

maintaining a reliable position estimation throughout GPS outages. This is demonstrated by the system's ability to transition between indoor and outdoor environments (e.g. through a building, under a bridge or tunnel) without losing track of it's position.

Contents

List of Figures **xi**

1 Introduction **1**

1.1 Motivation 1

1.2 Research Objective 2

1.2.1 System Implementation 2

1.3 Thesis Structure 3

2 Background Information **5**

2.1 Literature Review 6

2.1.1 Biological Methods 6

2.1.2 Redundant Sensor Configurations 7

2.1.3 Vision Based Methods 8

2.1.4 Infrared Measurements 11

2.1.5 Electrostatic Measurements 12

2.1.6 Horizon Detection 13

2.2 Notation 15

2.3 Sensors 15

2.3.1 Accelerometer 16

2.3.2 Gyroscope 18
2.3.3 Magnetometer 19
2.3.4 GPS 20
2.3.5 Kinect 22
2.3.6 Thermopiles 26
2.3.7 Inertial Sensor Modelling 26
2.4 Systems 30
2.4.1 IMU 30
2.4.2 AHRS 30
2.4.3 INS 31
2.4.4 INS/GNSS 32
2.4.5 GNSS 32
2.5 Rotation Representations 34
2.5.1 Euler Angles 34
2.5.2 Rotation Matrix 35
2.5.3 Euler-Axis 36
2.5.4 Quaternions 36
2.6 Reference Frames 37
2.6.1 Body Frame 37
2.6.2 Inertial Frame 38
2.6.3 ECEF Frame 39
2.6.4 Tangent Plane 39
2.7 Filtering Algorithms 39
2.7.1 Kalman Filtering 39
2.7.2 Extended Kalman Filter 42
2.7.3 Unscented Kalman Filter 42

2.8 Depth Imaging 43
2.8.1 Point Clouds 43
2.8.2 Ranging Techniques 44
2.8.3 Registration 46

3 Algorithm Development **49**
3.1 Baseline INS 50
3.1.1 Performance Comparison 56
3.2 INS Structure 59
3.2.1 Least-Squares Integration 59
3.2.2 Cascaded Integration 60
3.2.3 Centralised Integration 62
3.2.4 Federated Integration 63
3.2.5 No-Reset 64
3.2.6 Fusion-Reset 65
3.2.7 Zero-Reset 66
3.2.8 Cascaded 67
3.2.9 Summary 67
3.2.10 Reference Navigation Processor 72
3.3 On-line Confidence Estimation 73
3.3.1 GPS Confidence Estimation 74
3.3.2 Kinect Confidence Estimation 76
3.4 Integration Kalman Filter 83
3.5 Summary 87

4 System Implementation **89**
4.1 Hardware 89

4.1.1 Decision To Use A Smartphone . . . 89
4.1.2 Commercial IMUs . . . 90
4.1.3 iOS vs Android . . . 93
4.1.4 Nexus S . . . 96
4.1.5 Kinect . . . 100
4.1.6 PhoneDrone Board . . . 101
4.1.7 Kinect Data Platform . . . 103
4.1.8 Commercial INS . . . 103
4.1.9 Platforms . . . 106
4.2 Software . . . 114
4.2.1 Point Cloud Library . . . 114
4.2.2 Android programming . . . 120
4.2.3 Nexus S Inertial Sensors . . . 124
4.2.4 Logging Application . . . 132
4.3 Performance Tests . . . 137
4.3.1 Nexus S Validation . . . 137
4.3.2 Kinect Validation . . . 144

5 System Verification **149**

5.1 Captured Data . . . 150
5.1.1 RGB Images . . . 150
5.1.2 RGB Histograms . . . 150
5.1.3 Depth Images . . . 150
5.1.4 Kinect Velocity . . . 157
5.1.5 GPS Position . . . 157
5.2 Output . . . 159

5.2.1 GPS Confidence 159
5.2.2 Kinect Confidence 160
5.2.3 Fused Position 160
5.3 Summary 160

6 Conclusions **165**
6.1 Further Work 166

A Matlab Histogram Generator **169**

B Various Transformations **173**

C Kinect Frame Logging Utility **177**

D Bibliography **181**

List of Figures

2.1 Electrostatic Field of Earth 14

2.2 Multipath GPS Signal 22

2.3 Microsoft® Xbox® Kinect Sensor 22

2.4 Primesense Overview 23

2.5 Primesense Operation and Specifications 24

2.6 Example Kinect Images 25

2.7 Comparison of Thermopile and MEMS Gyroscope 27

2.8 Inertial Measurement Unit 30

2.9 Attitude and Heading Reference System 31

2.10 Inertial Navigation System 31

2.11 Inertial Navigation System/Global Navigation Satellite System 32

2.12 Typical aircraft body coordinate system 38

2.13 ECEF XYZ and LLA, and Tangent Plane NED Coordinate Systems 40

3.1 Baseline Position Output vs XSens MTi-G Position Output 57

3.2 Baseline Comparison to MTi-G for X and Y Position 58

3.3 Least-Squares Integration 59

3.4 Total-State Cascaded Integration 61

3.5 Error-State Cascaded Integration 61

3.6 Total-State Centralised Integration 62
3.7 Error-State Centralised Integration 63
3.8 No-reset Federated Integration 64
3.9 Fusion-reset Federated Integration 65
3.10 Zero-reset Federated Integration 66
3.11 Cascaded Federated Integration 67
3.12 Error-state Cascaded Integration with GPS and Kinect as Aiding Sensors . . 68
3.13 Modified System Implementation 69
3.14 GPS Aiding Sensor Overview 69
3.15 GPS Navigation Processor Overview 70
3.16 Kinect Aiding Sensor Overview 71
3.17 Kinect Navigation Processor Overview 72
3.18 Reference Navigation System 73
3.19 Dilution of Precision (DOP) of GPS during Outdoor to Indoor Transition . . 77
3.20 Calculated Confidence of GPS during Outdoor to Indoor Transition 78
3.21 Kinect Comparison to Accelerometer over 5 metre Forward Movement 80
3.22 Example Kinect Depth Frames 01-08 81
3.23 Calculated Kinect Confidence over 5 metre Forward Movement 82
3.24 Error Introduced in Confidence Through Downsampling 84

4.1 Nexus S (Images from Samsung) 97
4.2 Nexus S Sensors (Images from iFixit.com) 98
4.3 Custom USB Power Injection Cable 101
4.4 PhoneDrone Android Accessory 102
4.5 Kinect Data Logging Platform 104
4.6 Commercial INS Devices 104

4.7 XSens MTi-G IMU Performance Statistics 105
4.8 XSens MTi-G GPS Performance Statistics 106
4.9 IG-500N IMU Performance Statistics 106
4.10 Piper Cub with On Board Computer Platform 108
4.11 GWS Formosa 109
4.12 Quadcopter PhoneDrone Experimental Platform 112
4.13 PCL Modules 114
4.14 NDT Output from Kinect over 5 metre Forward Movement 121
4.15 Frame Processing Times for ICP and NDT 122
4.16 Android Operating System Architecture 123
4.17 Nexus S Sensor Axes 124
4.18 Android Activity Lifecycle (Source: http://developer.android.com) 125
4.19 Android Logging Application 132
4.20 Stationary Inertial Sensor Bench Test 139
4.21 Sensor Output Comparison 140
4.22 Nexus S Rotational Benchtest Outputs 141
4.23 Nexus S GPS Warmup and Initial Fix 142
4.24 Nexus S Inertial Sensor Outputs during 5 Minute Flight Test 145
4.25 Estimated Orientation during 5 Minute Flight Test 146
4.26 Nexus S GPS Position during 5 Minute Flight Test 147

5.1 Example Kinect RGB Frames 01-12 151
5.2 Example Kinect RGB Frames 13-24 152
5.3 Histograms for Kinect RGB Frames 01-12 153
5.4 Histograms for Kinect RGB Frames 13-24 154
5.5 Example Kinect Depth Frames 01-12 155

5.6 Example Kinect Depth Frames 13-24 156
5.7 Kinect Velocity and Position during Outdoor to Indoor Transition 157
5.8 Kinect Velocity and Position during Outdoor to Indoor Transition 158
5.9 GPS Position during Outdoor to Indoor Transition 158
5.10 GPS Position during Outdoor to Indoor Transition 159
5.11 GPS HDOP And Confidence Estimate 161
5.12 Kinect Confidences during Outdoor to Indoor Transition 162
5.13 Position Estimation of System during Outdoor to Indoor Transition 163
5.14 Position Estimation of System during Outdoor to Indoor Transition 163

List of Listings

2.1 Example Point Cloud Data (PCD) Format 43

4.1 ICP implementation in PCL 115

4.2 NDT implementation in PCL 117

4.3 IMUListener Interface 126

4.4 IMUManagerr Constructor 126

4.5 IMUManager onSensorChanged 127

4.6 GPSListener Interface 129

4.7 GPSManager Interface 129

4.8 GPSManager start/stop Functions 130

4.9 GPSManager onLocationChange and onNmeaReceived 131

4.10 INSActivity onCreate 133

4.11 INSActivity onPosition callback function 133

4.12 INSActivity onAccelerometer callback function 134

4.13 INSActivity updateData function 135

A.1 MATLAB Depth Image Histogram Generator 169

C.1 GPSManager Interface 177

List of Acronyms

AHRS Attitude and Heading Reference System

API Application Programming Interface

CAD Computer-aided Design

COTS Commercial Off The Shelf

CPU Central Processing Unit

CSV Comma-Separated Values

DOF Degree(s) of Freedom

DOP Dilution of Precision

EEPROM Electronically Erasable Programmable Read Only Memory

ESC Electronic Speed Controller

FPGA Field Programmable Gate Array

GPIO General Purpose Input/Output

GPL General Public Licence

GPS Global Positioning System

HDOP Horizontal Dilution of Precision

I2C Inter-Integrated Circuit

IC Integrated Circuit

ICP Iterative Closest Point

IMU Inertial Measurement Unit

INS Inertial Navigation System

I/O Input/Output

IP Internet Protocol

ISM Industrial Scientific and Medical

ISP In-System Programming

ISR Interrupt Service Routine

JTAG Joint Test Action Group

LAN Local Area Network

LED Light Emitting Diode

LiPo Lithium Polymer

Li-Ion Lithium Ion

LiFe Lithium Iron

MAC Media Access Control

MCU Microcontroller Unit

MISO Master-In Slave-Out

MOSI Master-Out Slave-In

MPDU MAC Protocol Data Unit

MSDU MAC Service Data Unit

NDT Normal Distribution Transform

NED North East Down

NiMH Nickel-Metal Hydride

NTP Network Time Protocol

NWU North West Up

IEEE Institute of Electrical and Electronic Engineers

OS Operating System

PAN Personal Area Network

PC Personal Computer

PCB Printed Circuit Board

PCL Point Cloud Library

PDF Probability Density Function

PTP Precision Time Protocol

PWM Pulse Width Modulation

PPM Pulse Position Modulation

RAM Random Access Memory

RF Radio Frequency

RTOS Real-Time Operating System

SLAM Simultaneous Localisation and Mapping

SPI Serial Peripheral Interface

SS Subscriber Station(s)

ROS Robot Operating System

TCP Transmission Control Protocol

USART Universal Synchronous Asynchronous Receiver Transmitter

UAV Unmanned Aerial Vehicle

USB Universal Serial Bus

VDOP Vertical Dilution of Precision

WLAN Wireless Local Area Network

Acknowledgements

I would like to thank my supervisors, Dr Ben Upcroft and Dr Ryan Smith, for their guidance, expertise and trust throughout the various stages of the project.

To Ingrid

Statement of Original Authorship

The work contained in this thesis has not been previously submitted to meet requirements for an award at this or any other higher education institution. To the best of my knowledge and belief, the thesis contains no material previously published or written by another person except where due reference is made.

Signature: 

Date: 13/08/13

Copyright in Relation to this Thesis

Chapter 1

Introduction

1.1 Motivation

As the usage of robots spreads, the tasks they are asked to perform become more and more complex. In response a need is created for more adaptable sensory systems capable of operating under varying conditions with a level of accuracy higher than ever before. Traditional systems have fallen into a pattern of combining an inertial measurement module (typically accelerometer, gyroscope, and magnetometer) with GPS, which to date has seen much success. However, such systems are beginning to show their limitations as the demands put on them are increased in response to the advances in the robotic systems that they power.

Systems such as these function well in an outdoor environment, but considering the trend toward increased robotics usage in indoor environments the sole reliance on a satellite based

positioning system poses a problem. A potential solution already exists in the form of image based navigation. Techniques such as Simultaneous Localisation And Mapping (SLAM) can use a set of stereo cameras (or other source of depth information) to obtain images of their environment, and process these images to form a map of their surroundings as well as to position themselves within that map. This solution works well for localised situations, being able to provide a local position estimate, but is unable to provide a global positioning estimate without an external reference.

1.2 Research Objective

Through the utilisation of adaptable filtering algorithms and a variety of complementing sensors, a system that is capable of operating under many different environments and under varying conditions can be developed. Using an error-state cascaded integration filtering structure, a reference navigation system is aided by the combination of an Xbox Kinect sensor and a GPS module. The system generates on-line confidence levels in these aiding sensors, and uses them to combine the sensor data in the integration Kalman filter.

1.2.1 System Implementation

This system is implemented on a Nexus S Android powered smartphone. This implementation is for two reasons. Firstly, modern smartphones feature all of the basic sensors required for the implementation of an INS, and using a commodity platform such as this will facilitate a much wider user base than that of custom hardware. Secondly, the limited processing power

and memory constraints of the Nexus S platform demonstrate the efficiency and robustness of the system implementation.

1.3 Thesis Structure

In chapter one, existing literature is discussed and the potential solutions and problems presented are investigated. The relevant background information necessary to fully understand the remainder of the thesis is also presented. This includes relevant notation, common variables, and overviews on methods for representing rotations, filtering algorithms and their associated pros and cons. This chapter also includes a short introduction to depth imaging and the associated representations and techniques, as these methods make up a large part of the core work of the thesis.

Chapter two discusses the theoretical implementation of the system. A baseline inertial navigation system without any aiding sensors is first developed and compared in performance to an off the shelf INS system (an XSens MTi-G). Subsequently to this the viability of augmenting the system with an Xbox Kinect sensor is determined. The Kinect is evaluated, and methods for determining a confidence level in both the Kinect, and GPS are developed and verified.

In chapter three the practical implementation of the system is discussed. The decision to use a smartphone as the implementation platform is proposed, and an evaluation and comparison of potential implementation platforms is undertaken. The processing and sensory hardware of the selected implementation platform are then evaluated under a series of testing

methodologies, culminating in an evaluation while on board a UAV platform. As well, issues related to interfacing external hardware with a consumer smartphone are discussed and investigated.

Chapter four is dedicated solely to showing the system operating on a dataset which includes and outdoor to indoor transition. The system is demonstrated to be able to maintain an accurate positional estimate even throughout this disturbance, through fusion based on the confidence levels of the aiding sensors.

Finally, chapter five contains the conclusions of the thesis, as well as the future directions of the work.

Chapter 2

Background Information

To create a robust, reliable, and accurate navigation system, many differing approaches are available. The approach taken here is to investigate the usage of many varying sensor types, along with the novel structuring of specific sensor types to provide a layer of redundancy, as well as a source of additional information. This begins with the investigation and analysis of the methods used by biological systems. Further to this, special arrangements of sensors can provide the same information that is normally sourced from a different sensor (e.g. accelerometer pairs on a lever arm measuring rotation rates), these information sources can then be combined to compensate for both sensors inherent errors.

2.1 Literature Review

Many methods to provide redundant information are available, such as the detection of the visual horizon in video, or the estimation of attitude by measuring the difference in infrared temperature between Earth and space. Each of these methods has its positives and negatives, and by combining multiple methods at once into one coherent system, all the sensors can operate in unison to correct for the deficiencies of others, providing a system with better accuracy than any one sensor alone. Previous work on methods such as the above are explored in the following sections.

2.1.1 Biological Methods

Campolo et al. [6] investigate the usage of biologically-inspired sensors for dynamic attitude stabilisation of micro aerial vehicles. They propose a dynamic output feedback control scheme that is based on state estimation from redundant measurements, enabling it to be more robust to external disturbances and measurement noise. Although the main focus is centred towards robotic flying insects, it is general enough to be considered for use on traditional UAVs. The authors state that the highly specialised sensory system of flying insects is at the base of their extraordinary flying performance Epstein et al. [17], that flying insects rely on a heterogeneous set of redundant sensors for controlling their flight manoeuvres, and that this redundancy is the key to their the insects robustness. They go on to detail the various types of sensory inputs that insects utilise (halteres, ocelli, gravitometers, and magnetic and light compasses) and conclude that since the various kinematic quantities (e.g. angular velocity) are derived from more than one sensor, that the insects sensory system

is clearly redundant.

2.1.2 Redundant Sensor Configurations

Sukkarieh et al. [59] investigate redundant configurations of inertial sensors. They develop algorithms to determine an optimal configuration of sensors. This is accomplished by using a conical arrangement of sensors, the algorithms provide the necessary half-angle for the cone, as well as the sensor spacing, for a given number of sensors. Fault detection and isolation are also discussed. The authors use these algorithms to design a redundant system, the 'Tetrad' IMU. The Tetrad consists of 4 gyroscopes and 4 accelerometers on the faces of a tetrahedron. This gives the arrangement of a cone with 3 sensors spaced at 120 degrees, and a central sensor along the axis of the cone for both the accelerometers and gyroscopes. These examples of redundant sensor configurations can be utilised in the proposed system when instances of redundant sensors are detected.

In Tan et al. [62] the feasibility of an inertial navigation system that does not require the usage of gyroscopes is studied, the system only makes use of accelerometer readings. The authors argue that low-cost gyroscopes lack the accuracy that is required for precise navigation applications. They develop models for the accelerometers, and in the same vein as Sukkarieh et al. [59], a method to determine the feasibility of possible system configurations. However, unlike Sukkarieh et al. [59] the system configuration is not limited to conical shapes. The feasibility of the system determines whether it is possible to estimate all the variables needed, based on the number, positioning, and orientation of accelerometers on the platform. They conclude by confirming the feasibility of a six accelerometer configuration proposed in

Chen et al. [9], consisting of a one-axis accelerometer mounted on each face of a cube, with the sensitive axis aligned with the cube face diagonal, so as to form a regular tetrahedron.

Gebre-Egziabher et al. [21] propose a very similar system. The difference being that measurements of both Earth's magnetic field and gravity are used, rather than solely gravity. A method known as Wahba's problem is used to determine the attitude through vector matching, or solving for the rotation that aligns the two vectors in a base coordinate frame. This has the benefit of eliminating random walk errors resulting from error integration. Unlike Tan et al. [62] and Sukkarieh et al. [59], Gebre-Egziabher et al. [21] do not consider unique sensor arrangement configurations. The system's operation is verified with a real world flight test. The system's performance is shown to have a mean of less than 1 degree on all 3 axes, and a standard deviation of 11 degrees in yaw, and 3 degrees in roll and pitch. These systems which operate without the use of common inertial sensors provide a reference for possible system operation when an unexpected failure makes the output of a particular sensor unusable.

2.1.3 Vision Based Methods

Cornall and Egan [10] details the use of video processing for autonomous flight control of small UAVs. This method proposed uses a small colour camera to obtain images of the horizon, from which the vehicles roll angle can be estimated. The determination of the horizon angle is a multi-stage process. Firstly, the image is sectioned into sky and ground regions, through the use of a simple thresholding method on the blue channel of the image. The blue channel of the image is preferred due to the natural colour of the sky being blue.

A circular mask is then applied to the image and the centroid of both the sky and ground regions found. The authors state that the horizon is perpendicular to the line joining the sky and ground centroids. Through simulated trials of this method, an accuracy of better than 1% has been achieved in determining the roll angle. It is proposed to also make use of the texture of the image to assist in the determination of which pixels are sky and which are ground, typically the sky has much lower texture than the ground.

Thurrowgood et al. [65] also estimate the attitude of a UAV by monitoring the visual horizon. Their method improves on the system proposed by Cornall and Egan [10] by utilising novel methods for sky/ground contrast enhancement, and automatic determination of the optimal thresholding value for ground/sky separation. Thurrowgood et al. [65] note that while many other horizon detection methods such as Cornall and Egan [10] focus solely on the blue channel of the image, making use of all three channels can provide an improved estimate. After the contrast enhancement and thresholding operations, a 3D plane is aligned with the horizon using an iterative linear least squares fit procedure. This system has been tested on a small model plane under manual control, and via comparison to a reference Microstrain inertial measurement system, achieved a roll and pitch error standard deviation of 3.31 degrees, and 3.01 degrees respectively. The authors note that the total execution time of the algorithm on a 1GHz processor is approximately 2.4 milliseconds, allowing the system to operate in real-time.

Hol et al. [27] develop a system consisting of the fusion of inertial measurements and vision, capable of estimating its attitude to within 1 degree, and position to 2 centimetres. The authors note that using vision alone is capable of estimating the attitude and position, however the addition of IMU fusion makes for a more robust system, by being able to utilise the IMU to correct for deficiencies in vision Hartley and Zisserman [24], Ma et al. [40] (e.g.

movements faster than the frame rate). Vision provides an accurate estimate of orientation and position at a lower rate, while an IMU provides a short-term accurate estimate, at a higher rate. The authors show that the fusion of these two systems can provide robust estimates of the position and orientation. To verify the operation of the system, they mount their experimental setup onto a high precision 6 DoF industrial robotic arm, by moving the arm through predefined patterns, the system's accuracy is confirmed.

Similarly to Hol et al. [27], Rutkowski et al. [52] discuss the usage of the fusion of airspeed and optical-flow to estimate self-motion so as to assist or replace a GPS system. They take a biologically inspired approach, discussing the sensory inputs that insects utilise to perform the tasks required for self-motion estimation. Previous research by Srinivasan et al. [57] has shown that insects use optical flow to detect and to help quantify their self-motion. The authors use an optical flow algorithm developed by Srinivasan [56] to obtain estimations of translational motion along the x and y axes, and rotational motion about the z axis. They derive algorithms to give the relationship between airspeed, wind speed, and optical flow as well as ground speed and optical flow. As these algorithms are dependant on the platform altitude, the process is to first estimate the altitude, and then by using the optical flow information, estimate the wind speed and ground speed. To test the algorithm, the authors used digitised positional data of a moth tracking and odour plume in a wind tunnel. On this test data, the algorithm achieved a standard deviation of less than 0.5m for altitude estimation, and less than 0.6m/s for wind speed and ground speed estimation.

2.1.4 Infrared Measurements

Egan and Taylor [15] explore the use of infrared sensors to determine the attitude of a UAV. The principle idea behind using infrared sensors to measure the attitude of a given platform is that Earth has a higher temperature than space. The sensors used in Egan and Taylor [15] have a field of view of 100 degrees, and are used in groups of two to form a differential measurement. The arrangement used is a commercial quad sensor unit Direct [12] with a baseline of a few centimetres, the downside of this arrangement is that the sensor must be mounted clear of any obstructions (e.g. aircraft wings and/or tail) to ensure a good measurement from the sensor. Instead of deriving a mathematical model of the temperature differential relating to elevation, a exponential curve fit is used on previously gathered data.

One of the flaws of the hardware design used is that only the differential measurements are available, rather than each individual sensors. To accurately estimate the roll and pitch of the platform, the maximum differential is required to be known beforehand. This quantity can be easily obtained by simply rotating the sensing device to 90 degrees before flight, however this method is cumbersome, and can result in large under or over estimation of the roll/pitch angles if the differential changes during flight. A preferred solution is to have a third set of two sensors oriented vertically such that they provide a continuous update of the maximum temperature differential. The authors state that their results indicate an infrared measurement system alone is capable of estimating the platform attitude to within a few degrees given reasonable weather conditions.

Taylor et al. [63] continue the work discussed in Egan and Taylor [15], and utilise infrared sensors for the estimation of roll angles for a photo reconnaissance UAV platform. By using

the differential between two thermopiles oriented at 180 degrees to one another, the system estimates the platform roll angle. The proposed system was tested successfully onboard a UAV under manual control, with the infrared thermopiles assisting in auto-levelling the aircraft. The authors conclude that the proposed system is a highly effective and low cost alternative to inertial systems, and that the infrared thermopiles are capable of operating in both day and night under visual meteorological conditions (VMC). Despite their performance, infrared measurement devices all suffer from several common flaws. They can be sensitive to weather and geological conditions, e.g. large cloud cover obstructing view of the sky, or a large mountain range to one side. As well as this Taylor et al. [63] reports that they experience some issues with thermopile readings when the sun is particularly close to the horizon. This does not, however, preclude their use as part of the proposed multi-level sensor fusion suite.

2.1.5 Electrostatic Measurements

The surface of Earth is covered with an electric field, the strength of which varies depending on the altitude at which the measurement is taken. By attaching a pair of electrostatic sensors to an aircraft, one at the end of each wing, the roll angle can be estimated by calculating the difference between the two measurements. The variations in the field are distinct enough that this angle can be measured with a wingspan of as little as 1 metre, given suitable amplification of the electrical inputs. This operation is similar in nature to that of the IR thermopiles detailed in subsection 2.3.6.

Work was initially performed on this method to determine the roll and pitch of an air-

craft was developed by Hill [26]. This method involves the usage of safe radioactive materials present on the wingtips of the plane to measure the Earth's electrostatic field, and by measuring the voltage difference an attitude estimate could be calculated. Obviously, using even safe radioactive materials is out of the question in modern systems, and is shown to be unnecessary by Chao [8, Note: Written in Chinese] through the use of field-effect transistors (FETs). Hill [26] fixed an experimental system to a full size Cessna aircraft along with a gyroscope, and compared the outputs. It is shown that the outputs are nearly identical, suggesting the use of measurements of the electrostatic field as a viable method for attitude determination. As shown in Figure 2.1 downsides to the method include thunderstorms causing an inversion in the polarity of the field in their vicinity, as well as various ground sources of interference such as power transmission lines. This technique is an example of one of the more exotic methods to determine an airborne platforms attitude.

2.1.6 Horizon Detection

The method operates by first enhancing the sky/ground contrast of the image. By rotating the UV space by 38 degrees it is noted that the majority of data is aligned with the U' axis, reducing the data to 1D. The authors note that performing a threshold on either the brightness or colour alone results in 30.5% misclassified pixels if brightness is used, and 22.2% for colour. By plotting the histogram of 4U' versus Y and rotating by 40 degrees, the authors state that the projection of the brightness axis onto the colour axis results in a decrease of pixel overlap to 8.6%. The sum of all applied transformations reduces down to a simple formula to be applied to the image: $C = -1.16R + 0.363G + 1.43B - 82.3$ Thurrowgood et al. [65].

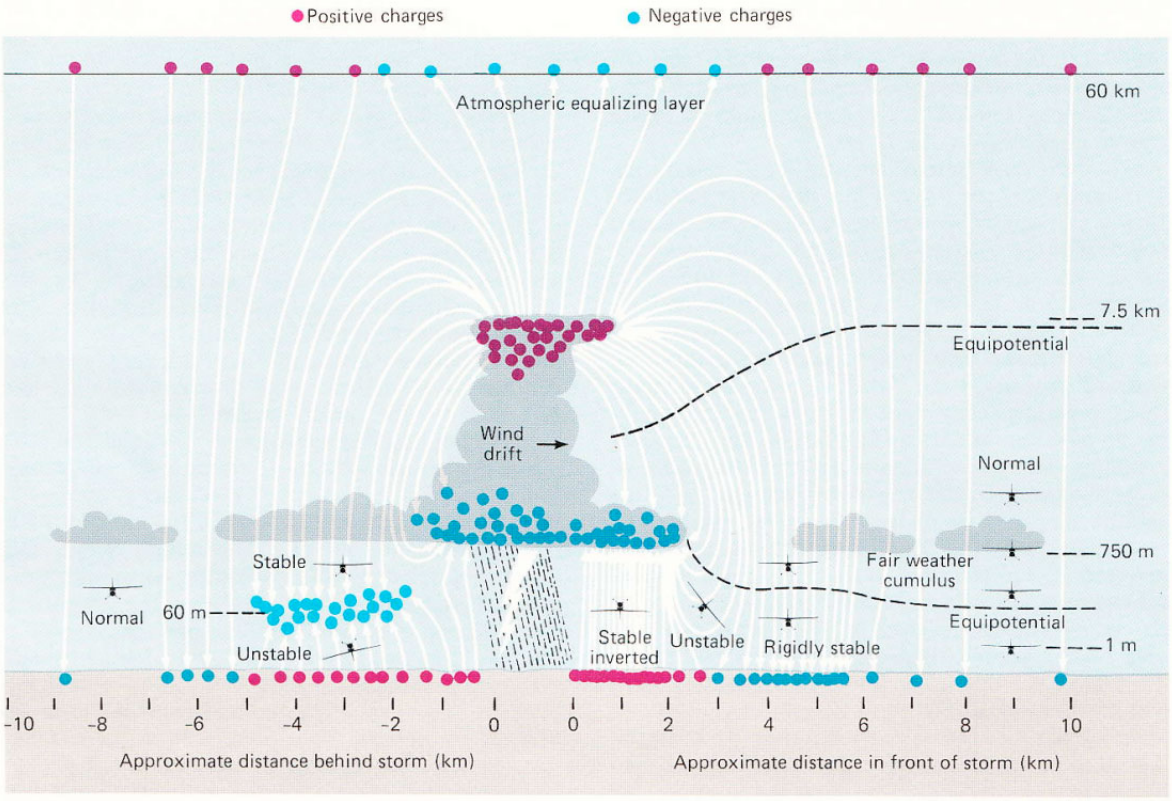


Figure 2.1: Electrostatic Field of Earth

The assumption is made that the sky and ground share half of the image each, and it is stated that the most likely location for the optimal thresholding value is at positions that correspond to local minima of the histogram. It is also shown that even if the aforementioned assumption is violated, the algorithm remains robust, performing equally as well on a sample image with only 4% of the image being sky.

2.2 Notation

The notation used throughout this document is standard for the vast majority of writings encountered while researching this topic. Transformation matrices are typically represented as $\mathbf{R}_a^b$, where b is the frame being transformed to, and a is the current frame, e.g. $\mathbf{R}_b^n$ would transform from the body frame into the navigation frame. Measurement vectors will be represented like so: ω_{ib}^b, in this case this would be the rotation of frame b (the body frame) with respect to frame i (the inertial frame) represented in frame b. Typically ω is used for rotation rate measurements and a used for acceleration measurements.

$\mathbf{F}$	System Matrix from Linearisation
$\mathbf{G}$	Input Matrix from Linearisation
$\mathbf{H}$	Measurement Matrix from Linearisation
$\vec{x}$	State Vector

2.3 Sensors

This section provides a brief overview on a variety of sensors available for use, and details their respective advantages and disadvantages.

2.3.1 Accelerometer

Accelerometers are one of the most commonly used sensors on robotic platforms. They are used to measure the acceleration (typically in three axes) from which the velocity and position of the sensor can be estimated. In addition to measuring accelerations, accelerometers can be used to estimate the orientation (roll and pitch) of the sensor by calculating the angles of the gravity vector. This estimation is valid provided that the sensor is not undergoing any additional accelerations.

A model of the accelerometer output as a combination of linear accelerations and gravity is shown in Equation 2.1

$$\mathbf{y}_a = \mathbf{a}_{ib}^b - \mathbf{g}^b \tag{2.1}$$

$$\mathbf{g}^n = \begin{bmatrix} 0 & 0 & g \end{bmatrix} \tag{2.2}$$

Where $\mathbf{y}_a$ is the measured acceleration, and $\mathbf{a}_{ib}^b$ is the acceleration of the body with reference to the inertial frame. Under the assumption that there are no external accelerations, the equation reduces to $\mathbf{y}_a = \mathbf{g}^b$. As gravity is known in the navigational frame (Equation 2.2 where g is the acceleration due to gravity) the equation can be transformed as shown in Equation 2.3 and solved for roll and pitch.

$$\mathbf{y}_a = \mathbf{g}^b = \mathbf{R}_n^b \mathbf{g}^n \tag{2.3}$$

An interesting phenomena is the manner in which gravity manifests itself in an accelerometer reading. Gravity can be defined as a force attracting objects towards Earth, and as a result of this definition a common misconception is that gravity will be read as a positive downward acceleration. However, the opposite is true, gravity will measure as 9.8m/s in an upward direction ([0, 0, 9.8] for NWU or [0, 0, –9.8] for NED), the reason for this being the way in which accelerometers measure acceleration. The easiest method of measuring acceleration is to place a mass at the end of a beam, and by measuring the deflection of the beam the acceleration can be estimated. Thinking about this in more detail, the beam will be forced downwards both on its own and by the mass attached to its end due to the force of gravity, the very same effect which would occur if the measurement apparatus were to accelerate upwards in zero gravity. Therefore, as the accelerometer is unable to tell the difference between forces due to gravity and acceleration, gravity will manifest itself as an acceleration in an upward direction.

Typically accelerometers are used as part of an inertial navigation system to allow for fast-rate position updates between an aiding positional input, such as GPS. Under these conditions, the accelerometer data must undergo a double integration, which causes the estimate to drift very rapidly. This drift is usually cancelled at each aiding sensor input, as that sensor is considered to be the authority for position. If the update rate of the aiding sensor is of a sufficient frequency, the potential problems this drift can cause are minimised. However, if the aiding sensor updates at an insufficient rate, or if it is lost entirely (e.g. Loss of GPS signal) the positional estimate will lose accuracy very quickly. Existing systems such

as the MTi-G [4] and the IG–500N [61] respond to a loss of GPS by simply ceasing positional estimates, and degrading their orientation estimates.

2.3.2 Gyroscope

Gyroscopes are sensors that detect the rate of rotation (angular velocity) around an axis. Whereas traditional gyroscopes were mechanical and relied on the gyroscopic forces of a rotating mass to detect rotation, modern gyroscopes are typically Micro-Electro-Mechanical Systems (MEMS) based, allowing for low-cost mass production. A MEMS based gyroscope takes advantage of a vibrating structure in the silicon of the microchip to determine the rotational rates about the sensors axis. The ability to produce MEMS based sensors so easily has led to their commonplace use in many different modern devices, including handheld game consoles and mobile phones.

Along with MEMS based gyroscopes, there exists several other types and methods for measuring angular velocity. A Ring Laser Gyroscope (RLG) takes advantage of the Sagnac effect[1] and is able to measure rotation rates from the interference pattern of lasers. A Fiber Optic Gyroscope (FOG) implements the same principle, through as much as 5 kilometres of coiled fibre optics, each turn multiplying the Sagnac effect, although it is slightly less sensitive than a RLG.

Electrically Suspended Gyroscopes (ESG) operate by suspending a spinning ball within a vacuum through an electric field. This spinning ball is deliberately constructed slightly off balance, and by measuring the sinusoidal pulses that it's rotation produces a measurement

[1] http://en.wikipedia.org/wiki/Sagnac_effect

of the rotational rate of the device can be calculated.

Table 2.1 classifies the various types of gyroscopes and provides a rough outline as to their inherent accuracy.

Table 2.1: Classification of various gyroscope types.

Class	**Gyro Technology**	**Drift Rate**
Military Grade	ESG, RLG, FOG	< 0.005 deg/h
Navigation Grade	RLG, FOG	0.01 deg/h
Tactical Grade	RLF, FOG	1 deg/h
AHRS	MEMS, RLG, FOG	1–10 deg/h
Control System	MEMS	10–1000 deg/h

2.3.3 Magnetometer

Magnetometers are able to detect the intensity of magnetic fields around them. Their most common use is to detect the Earth's magnetic field, and thus provide a source of heading information. A compass can be considered a form of a one axis magnetometer. In an INS the magnetometer typically has three axes, so as to be able to determine the heading regardless of the orientation of the sensor (a compass will not work if held sideways for instance). In this manner, if some broad assumptions are made regarding the magnetic field of Earth a magnetometer could theoretically be used to determine the roll and pitch of a robotic platform in addition to heading. In reality however, the use of a magnetometer in this manner is impeded by several issues.

Magnetometers are prone to interference from stray magnetic fields created by the operation of electronic devices and presence of metallic objects near the sensor, these are called soft iron and hard iron effects respectively. Hard iron effects can be calibrated for (assuming

the metallic objects are stationary), however electrical interference cannot be, and presents a particular problem when a magnetometer is used on an electric aircraft.

2.3.4 GPS

The Global Positioning System (GPS) is a form of Global Navigation Satellite System (GNSS) put in place by the US Navy. The system consists of 24 satellites orbiting Earth in six 12 hour orbital planes. These six orbital planes are spaced evenly around the equator, giving a plane separation of 60 degrees. This configuration of satellites ensures that no matter the position on Earth, at least four satellites will be visible at all times. In GPS terminology these satellites are known as Space Vehicles (SVs), and they are continually broadcasting encoded signals which a suitable receiver can use to estimate it's position.

GPS receivers typically consist of two important segments, a baseband, and a navigation segment. The baseband is responsible for tracking the signal from each individual SV and decoding the data contained within that signal. Modern receivers will have up to 50 channels even though there are only 24 SVs broadcasting data. These extra channels allow the receiver to obtain a lock on SV signals faster, as well as to achieve a higher level of positional accuracy. The navigation segment takes the signals decoded by the baseband, and uses the information within to estimate the receiver's position. The information the SVs broadcast includes SV position and velocity, clock error, and the health of that satellite.

GPS has two levels of service, the Standard Positioning Service (SPS), and the Precise Positioning Service (PPS). The SPS is the service available for anyone to use, while the PPS

is intended to be used by the US military. Higher end receivers with advanced processing algorithms are able to track the precise positioning without PPS authorisation, albeit with a slight accuracy penalty.

Since GPS receivers operate passively (that is, they do not send signals to the SVs), the system has an unlimited number of simultaneous users. GPS is a line of sight system. Without an unobstructed path between the SV and the receiver, the signal may not be received. As an example, a GPS receiver will not typically function indoors, underwater, or even under significant tree cover.

Multipath

Depending on the location of the receiver, multiple reflections of the same signal from an SV may be received at different times. This is known as multipath error, and can cause degradation in positional accuracy. The receiver can attempt to filter the unwanted reflections out by using the signal with the highest strength, but often times the only signal received is a reflected one, particularly in urban environments between buildings (commonly referred to as urban canyons). Figure 2.2 illustrates the reflection of a signal from a tall building. Due to the additional time taken for the reflected signal to reach the receiver, an error will be introduced into the positional estimate, typically on the order of a few metres. Modern receivers are more robust at filtering reflected signals and disregarding them entirely even if the reflection is the only signal received from the SV.

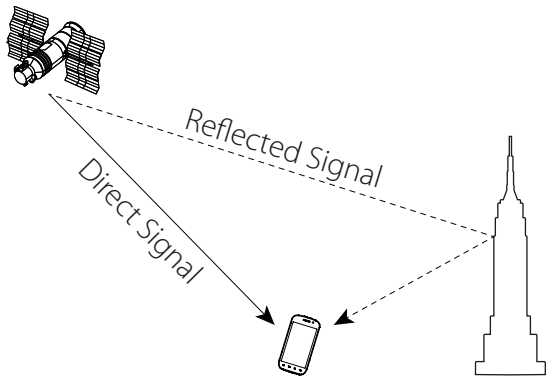


Figure 2.2: Multipath GPS Signal

2.3.5 Kinect

Figure 2.3: Microsoft® Xbox® Kinect Sensor

The Kinect is a sensor developed by Microsoft for the Xbox and is shown in Figure 2.3. The software solution was developed as an in-house project and based on hardware technology from PrimeSense[2]. The Kinect is capable of producing depth maps of the scene it is observing, with a maximum depth capability of approximately 5 metres. The device itself consists of four primary components, a colour imaging camera, a monochrome infrared

[2]http://www.primesense.com/

camera, an infrared projector with a fixed projection pattern, and an image processing chip which processes the IR frames to provide a depth map. Figure 2.4 provides an overview of the Primesense based hardware within the Kinect.

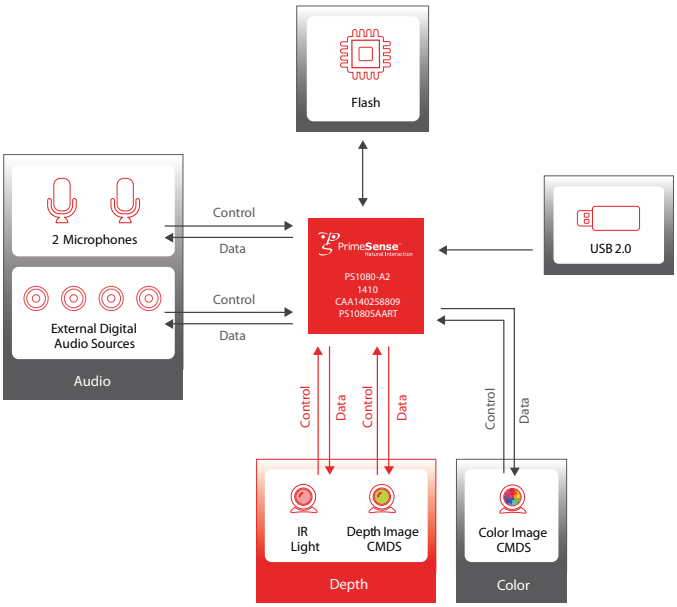


Figure 2.4: Primesense Overview

Traditionally to obtain a depth map of a particular scene, a stereo camera vision rig consisting of two cameras with a known baseline and converging views would be required. By matching features between these two images the depth can be calculated. Even though the Kinect does indeed have two cameras on board, this is not the method it utilises to calculated the depth map.

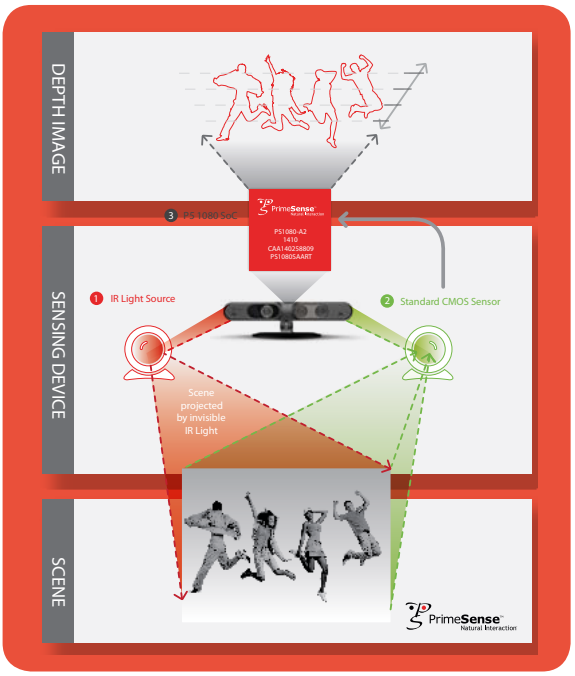
The IR projector has a fixed pattern which it projects onto the scene, this pattern actually consists of three sub-patterns. These sub-patterns have different arrangements and different sized points to allow the Kinect to operate over a larger range of depth (e.g. the smaller dots won't be captured by the camera past a certain distance, thus the next pattern up can serve this distance range). The ranges for the patterns are approximately 80 - 120cm, 120 -

200cm, and 200 - 350cm.

Since the relationship between the IR projector and the IR camera is known, and the projection pattern is programmed into memory at the factory, the information that would normally be provided by two cameras can be provided by one. The second image is 'virtual', since the projected patterns are known the offset between the patterns in the IR image and the hardcoded position can be found, after correcting for the effects of lens distortion. Figure 2.6 shows an example RGB, IR, and depth frame from the Kinect.

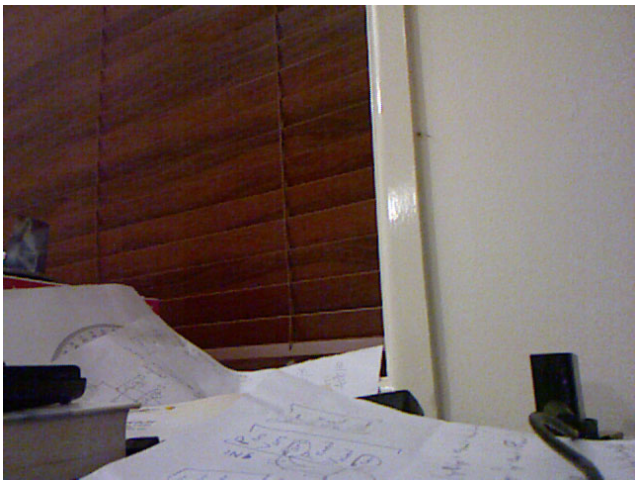
Sensor Specifications	
Field of view (horizontal, vertical, diagonal)	58° H, 45° V, 70° D
Depth image size	VGA (640x480)
Spatial x/y resolution (@2m distance from sensor)	3mm
Maximal image throughput (frame rate)	60fps
Average image latency in full VGA resolution	40ms
Operation range	0.8m-5.0m
Color image size	1280x960
Audio: built-in microphones	2 mics
Audio: digital inputs	4 inputs
Data interface	USB 2.0
Power supply	USB 2.0
Power consumption	2.25W
Dimensions (width x height x depth)	18cm x 2.5cm x 3.5cm
Operation environment (every light condition)	Indoor
Operating temperature	5°C – 40°C

(a) Primesense Overview

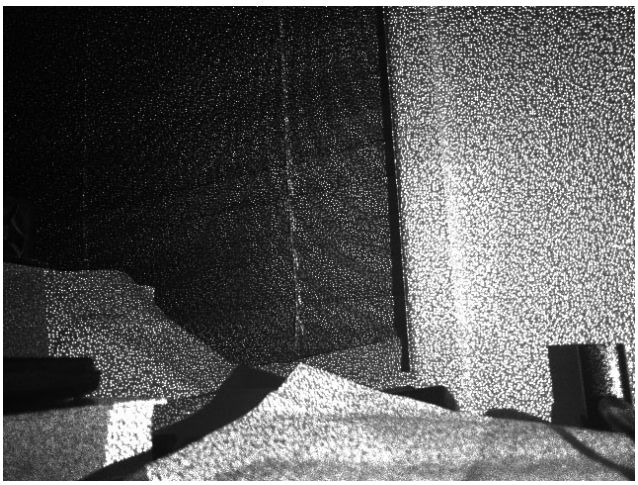


(b) Primesense Operation

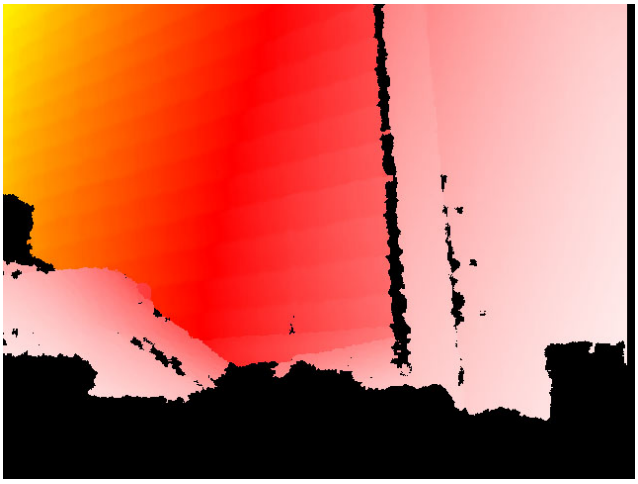
Figure 2.5: Primesense Operation and Specifications



(a) RGB Image



(b) IR Image



(c) Depth Image

Figure 2.6: Example Kinect Images

2.3.6 Thermopiles

Thermopile sensors are effectively light sensors operating in infrared (thermal) wavelengths between 5.5μm and 14.5μm, the sensors typically have viewing angles on the order of 60 degrees. By taking the difference between a pair of opposing sensors readings, their rotation angle can be estimated. This operation is generally limited to outdoor environments, and assumes the sensors have good visibility of the sky and the horizon. Combining two clusters of the sensors operating in this manner enables estimation of both roll and pitch angles. Figure 2.7 shows a dataset logged during the flight of the Piper Cub airframe (section 4.1.9) compared against MEMS gyroscope signals from the same flight, showing a close match between the two signals. As the thermopile data is drift free, they present a promising aiding sensor for correcting orientation estimates.

2.3.7 Inertial Sensor Modelling

To be able to utilise sensor readings in the filtering process, a mathematical model of the sensors behaviour is required. A general model is presented here, and more detailed models for some basic sensors in subsequent sections.

Let $\tilde{\mathbf{y}}^b$ be the reading from a fictitious sensor in the body frame, then a suitable general model of such a generic sensor is presented below.

$$\tilde{\mathbf{y}}^b = \mathbf{r}^b + \mathbf{x}_y + \nu_y$$

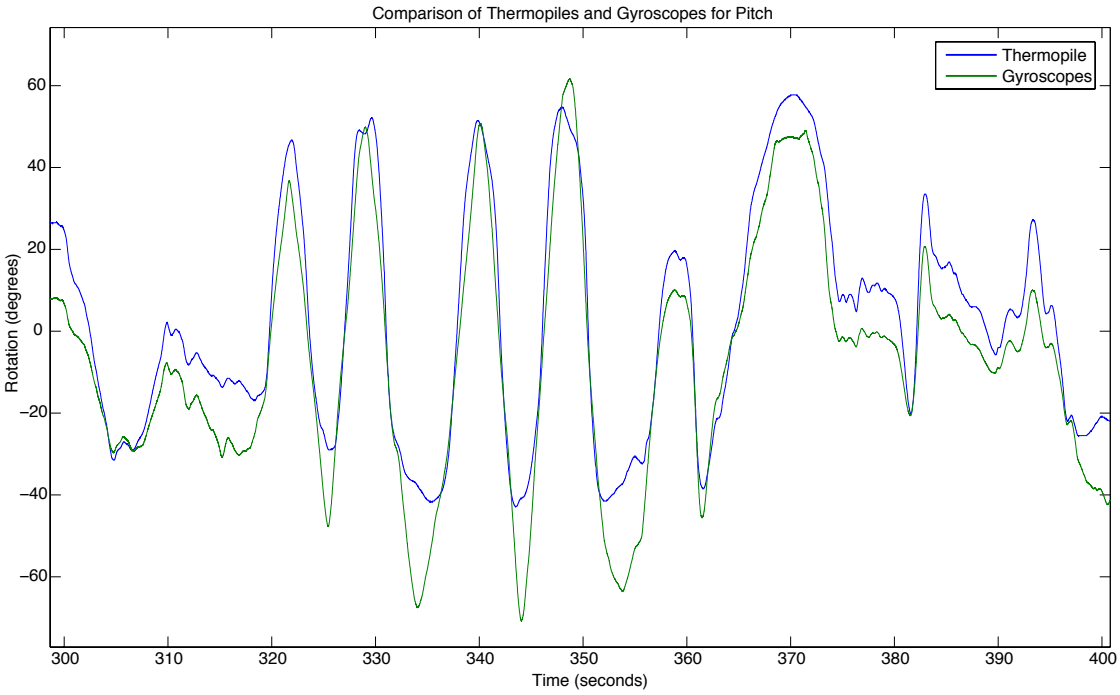


Figure 2.7: Comparison of Thermopile and MEMS Gyroscope

Here $\mathbf{r}^b$ represents the desired sensor reading, and $\mathbf{x}_y$ and ν_y represent different types of sensor errors. $\mathbf{x}_y$ describes the sensor errors that can be calibrated for, such as inherent drift on the sensor reading or temperature induced bias instabilities. ν_y represents other errors that will be modelled as white noise, this can include errors such noise introduced during the conversion from an analog reading to digital, or random fluctuations present in the sensor itself.

Gyroscope Model

Letting $\mathbf{u}$ denote the gyroscope measurement, a suitable model is:

$$\mathbf{u} = \omega_{ib}^{b} + \mathbf{x}_g + \nu_g$$

In this model ν_g is Gaussian white noise with a power spectral density (PSD) of $\sigma_{v_g}^2$, and $\mathbf{x}_g$ is an additive error. For the initial implementation of the inertial navigation system, the gyro additive error can be assumed to be a bias modelled as a first order Gauss-Markov process:

$$\dot{\mathbf{x}}_g = \mathbf{F}_g \mathbf{x}_g + \omega_g$$

Where $\mathbf{F}_g = -\lambda_g I$ and σ_g is a Gaussian white noise process with a PSD of $\sigma_{\omega_g}^2$.

Accelerometer Model

The accelerometer model is very similar to that used for the gyroscope, with the addition of gravity ($\mathbf{g}^b$) on the sensor measurements. The equations are repeated here in the form required for accelerometers.

$$\mathbf{y}_a = \mathbf{a}_{ib}^b - \mathbf{g}^b + \mathbf{x}_a + \nu_a'$$

$$\dot{\mathbf{x}}_a = \mathbf{F}_a \mathbf{x}_a + \omega_a$$

$$\mathbf{F}_g = -\lambda_g \mathbf{I}$$

Magnetometer Model

The model for magnetometers is shown below:

$$\mathbf{y}_m = \mathbf{m}_e^b + \mathbf{m}_b^b + \nu_m$$

where ν_m is Gaussian white noise with a PSD of $\sigma_{\nu_m}^2$. $\mathbf{m}_e^b$ is the vector representing the magnetic field of the Earth as measured in the body frame, and $\mathbf{m}_b^b$ is the magnetic field caused by the vehicle body itself measured in the body frame. In most cases the magnetometer can either be isolated from or compensated for the effects of $\mathbf{m}_b^b$ on the sensor reading. A key assumption with this model is that the magnetometer has been compensated for local magnetic field deviations based on the position of the sensor on the surface of the Earth. This assumption ensures that $\mathbf{m}^n = [m_e, 0, 0]$.

2.4 Systems

This section provides an introduction to the most common forms of systems found in inertial navigation, and explains the operation and utility of each.

2.4.1 IMU

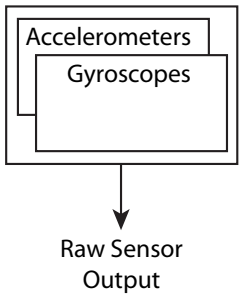


Figure 2.8: Inertial Measurement Unit

An Inertial Measurement Unit (IMU) is a device that incorporates accelerometers and gyroscopes, two types of inertial sensors. There is typically three of each sensor arranged in an orthogonal pattern so as to measure in each axis, and these clusters are also aligned to a common axis system. The IMU takes regular readings from the accelerometers and gyros and passes them on to a host system.

2.4.2 AHRS

Another type of sensor, the magnetometer, is usually used alongside an IMU. The magnetometer is able to take measurements of Earth's magnetic field, usually also in three axes.

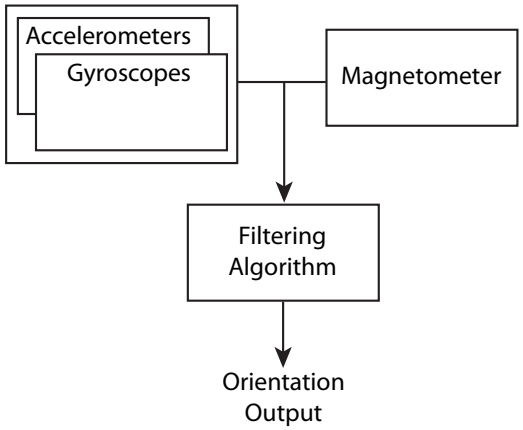


Figure 2.9: Attitude and Heading Reference System

When an IMU is combined with a magnetometer and a basic system for deriving orientation from the sensors, the system is usually referred to as an Attitude and Heading Reference System (AHRS).

2.4.3 INS

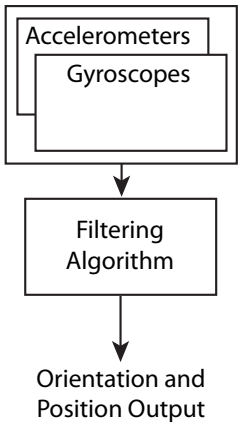


Figure 2.10: Inertial Navigation System

An Inertial Navigation System combines the IMU with advanced filtering techniques and algorithms to provide an accurate estimate of the attitude of the system. The most common

of these filtering algorithms is the Kalman filter. The sensors used in IMUs and AHRSs are susceptible to a multitude of errors. By being able to model these errors, they can be compensated for to provide a better estimate of the overall system[19] [22] .

2.4.4 INS/GNSS

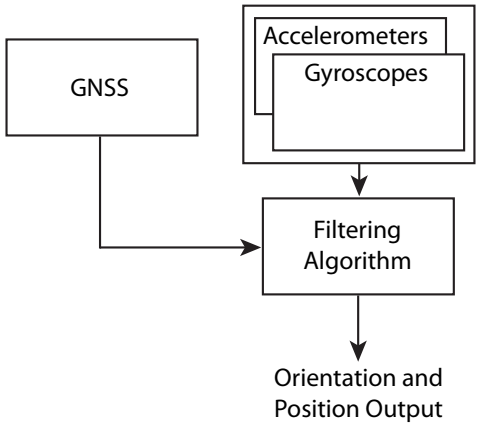


Figure 2.11: Inertial Navigation System/Global Navigation Satellite System

Alone, the INS can only provide an accurate estimate of its attitude, it can also provide an estimate of the position, however this will quickly diverge due to the accumulation of errors in the double integration required to turn the measured acceleration into position[66]. To overcome this problem, an INS is typically bound with a Global Navigation Satellite System (GNSS).

2.4.5 GNSS

The most commonly used of GNSS systems is the NAVSTAR Global Positioning System (GPS) put in place by the American government, although other countries are currently

deploying their own systems. The Russian Global Navigation Satellite System (Global'naya Navigatsionnaya Sputnikovaya Sistema, or GLONASS) was restored to full operation in September 2010, the EU's Galileo system is estimated to be completed in 2012, and China has proposed to expand their currently regional only BeiDou Navigation System into a global system named COMPASS by 2020. The abundance of differing satellite systems is a good thing for robotic navigation, as all systems can be utilised simultaneously to provide a much better estimate than any one system could provide alone.

GNSS systems operate by comparing signals received from satellites with locally generated reference copies, and along with a prediction of the satellites orbit the distance to the satellite can be deduced. Mathematically, three ranging measurements can be used to provide an estimate of the users position relative to the sources, however for GPS four measurements are needed, the extra being used to solve for an additional variable, the time offset in the receiver clock [11].

To integrate GPS into a navigation system, several approaches can be taken. The two main ones to be discussed here are defined as either loosely or tightly coupled systems. In a loosely-coupled system, the positional output from a stand-alone GPS unit is simply input to the navigation system, and used as a reference position to correct an integrated position. In tightly-coupled systems, the range measurement for each satellite is instead fed directly into the system Kalman filter. This brings the advantage that the minimum of four satellite fixes is no longer required to estimate the systems position. By integrating the inertial sensors to obtain a position estimate, any number of satellite range measurements can be combined with it to help constrain the error on this estimate. Tightly-coupled systems have a much higher complexity than loosely-coupled, and at least four satellites are still required for a complete positional solution, but the benefit of being able to use individual fixes to constrain

errors can often-times outweigh these minor flaws[32].

2.5 Rotation Representations

The representation of the orientation of the platform is an important factor in the development of an INS. This section provides a brief overview on the potential methods for storing and representing rotations, and the associated benefits and downsides.

2.5.1 Euler Angles

Euler angles are one of the most commonly used methods for representing rotations. Euler angles consist of three consecutive rotations around the principal axes x, y, and z. For example the vector $[\phi, \theta, \psi]$ can represent a sequence of rotations such as ϕ around the x axis, θ around the y axis, and ψ around the z axis. This rotation order is entirely arbitrary, and different orders such as $z - y - x$ or $z - x - z$ are allowable.

In the application of an INS these angles are commonly referred to as **roll** (ϕ), **pitch** (θ), and **yaw** (ψ) (less commonly **bank**, **elevation**, and **azimuth**) which represent the rotation about the x, y, and z axis respectively. The primary advantage to this representation is the ease of interpretation by human operators, the angles are easily visualised in terms of an object's movement. However, a large downside is that Euler angles are not always unique, a flaw which can lead to a situation known as *gimbal lock*, where a degree of freedom is lost. For example, consider an aircraft that is initially flying horizontally, if the aircraft were to

pitch upward to 90 degrees, the roll (rotation about the aircraft's longitudinal axis) and yaw (rotation about the aircrafts vertical axis) are no longer distinguishable. Because of this phenomena, Euler angles are usually avoided for internal orientation representations.

2.5.2 Rotation Matrix

Rotation matrices (also known as a Direct Cosine Matrix, or DCM) are a 3×3 matrix representing a rotation, an example of a rotation matrix is shown in Equation 2.4. Multiplying two rotation matrices together produces another rotation matrix, the rotation of which is equivalent to applying the original matrices in order. Euler angles are commonly implemented as a series of three rotation matrices, one for each axis.

$$\mathbf{R}_x\mathbf{R}_y\mathbf{R}_z = \begin{bmatrix} \cos(\psi)\cos(\theta) & \sin(\psi)\cos(\theta) & -\sin(\theta) \\ -\sin(\psi)\cos(\phi) + \cos(\psi)\sin(\theta)\sin(\phi) & \cos(\psi)\cos(\phi) + \sin(\psi)\sin(\theta)\sin(\phi) & \cos(\theta)\sin(\phi) \\ \sin(\psi)\sin(\phi) + \cos(\psi)\sin(\theta)\cos(\phi) & \cos(\psi)\sin(\phi) + \sin(\psi)\sin(\theta)\cos(\phi) & \cos(\theta)\cos(\phi) \end{bmatrix} \tag{2.4}$$

It is important to ensure that the rotation matrix remains orthogonal throughout its use. Over time, the product of several rotation matrices will end up drifting due to numerical errors and small inaccuracies. A drifted rotation matrix not only contains a rotational transformation, but also a skew transformation which will corrupt the product.

2.5.3 Euler-Axis

Euler-axis rotations are represented as a vector of 4 elements, illustrated in Equation 2.5. The downside to the Euler-axis representation is that four elements are used, but a 3D rotation only has three degrees of freedom. This same rotation can be represented using an infinite number of axes, provided their directions are the same. To remedy this issue a requirement may be that the axis vector is constrained to unit length.

$$(\vec{\mathbf{r}}, \phi) = \begin{matrix} os(\phi/2) \\ \vec{\mathbf{r}}\sin(\phi/2) \end{matrix} \tag{2.5}$$

2.5.4 Quaternions

Quaternions are the most complex of the representations, but are the most mathematically efficient and robust. Quaternions are a four dimensional non-commutative extension of complex numbers. They have one entry for the real component, and three entries for the imaginary components. The real component represents the cosine of the half angle ($\cos(\phi/2)$) and the imaginary part represents a vector along the axis of rotation multiplied by the sin of the half angle ($\vec{\mathbf{r}} \cdot \sin(\phi/2)$).

Quaternions avoid the pitfall of gimbal lock, and provide a straight forward algorithm for integrating the gyroscope signals. However, similarly to rotation matrices they must be normalised every so often to maintain accuracy. As well as this, quaternions have the downside that they are not able to be intuitively understood as rotations, in the way that

Euler angles can be. The solution to this is to limit the use of quaternions to internal representations, and convert them to Euler angle representations whenever presented to a user.

Generally, the quaternion is preferred to rotational matrices because of it's reduced complexity and computational load[19] . In the development of this system, the quaternion will be used for attitude representation.

2.6 Reference Frames

The rotation angles between the body fixed frame of the system and a suitable reference frame are commonly represented using three Euler angles, known as roll, pitch, and yaw. These frames and a variety of others relevant to the development of the system are detailed in the following sections.

2.6.1 Body Frame

The body fixed frame is simply the coordinate system aligned with the 'body' of the vehicle or platform on which the inertial navigation system is installed. (Since the application of this thesis is aimed toward UAVs, the majority of examples will be for aircraft) Shown in figure Figure 2.12 is the body frame of an aircraft. In aircraft the standard coordinate system consists of the x axis pointing in the forward direction, the y axis pointing out the right wing and the z axis pointing down. The body frame rotation rates are commonly expressed as

$\omega_{ib}^{b} = [p, q, r]$ for the x, y, and z axes respectively in aerospace nomenclature.

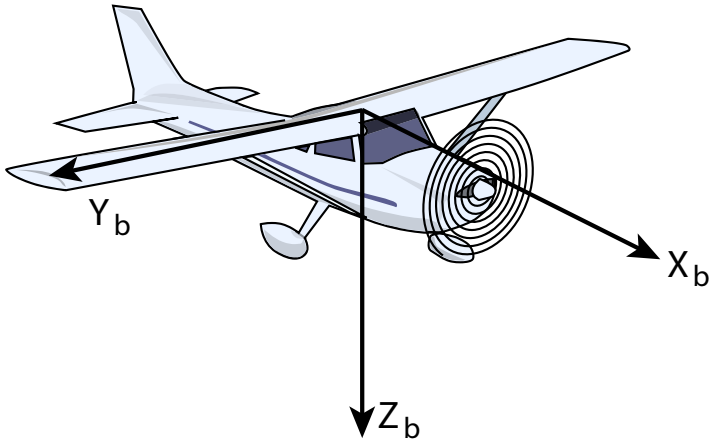


Figure 2.12: Typical aircraft body coordinate system

2.6.2 Inertial Frame

An inertial reference frame is one in which Newton's laws of motion apply Farrell [19]. Both the position of the origin and orientation of the frame are arbitrary. A commonly defined reference frame is the Earth centered inertial (ECI) frame, a frame which, at an initial time, has it's origin aligned with the centre of mass of the earth, it's z axis pointing to the north pole, its x axis pointing through the equator at 0 degrees longitude, and the y axis completing the orthogonal set. The measurements of inertial sensors are produced relative to an inertial frame resolved along the sensors sensitive axes Farrell [19].

2.6.3 ECEF Frame

The Earth centred earth fixed (ECEF) frame is a reference frame that is aligned with Earth in the same manner as the ECI frame at an initial time, the difference being that the ECEF frame is fixed with the Earth. The rotation of the ECEF frame with respect to the ECI frame is usually denoted as ψ_e^i and naturally has a period of 1 day. The most common forms of representation of position within the ECEF frame are spherical coordinates (Latitude, Longitude, Altitude) and rectangular coordinates (North, East, Down).

2.6.4 Tangent Plane

Since the vast majority of navigation occurs within a small section of the Earth's surface, the assumption that the surface is flat can be validly made. This is accomplished by forming a tangent plane with its origin at a useful point, such as an airport runway.

2.7 Filtering Algorithms

2.7.1 Kalman Filtering

Kalman filtering is an iterative estimation method that uses a stochastic model of the linear system in question along with measurements of components of that system to provide an estimate of the desired system states. The usual form of a Kalman filter is:

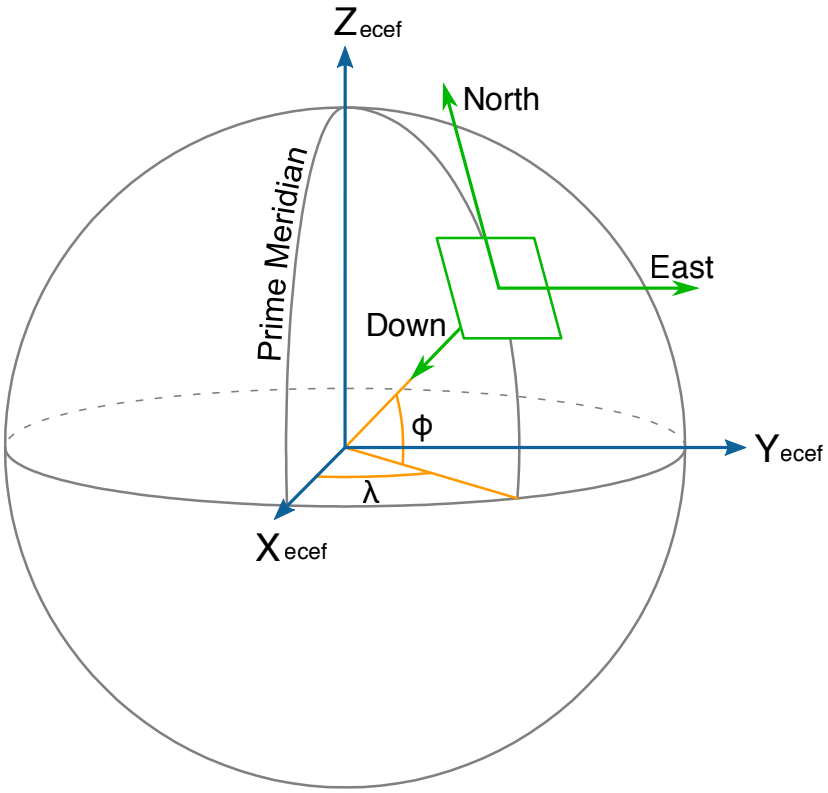


Figure 2.13: ECEF XYZ and LLA, and Tangent Plane NED Coordinate Systems

$$x(t) = F(t)x(t-1) + G(t)u(t) + w(t)$$

$$y(t) = H(t)x(t) + v(t)$$

where $x(t)$ is the system state and $y(t)$ is the measurement vector.

The Kalman filter is a recursive estimation technique. This recursion occurs in two stages, first the filter uses provided models to produce estimates of the current state. Along with these estimates uncertainties are generated as an indicator for how accurate the filter believes each estimate to be. The second stage occurs when the next measurement is observed. As

the measurement will contain an element of noise and other sources of errors, the estimates are updated to varying degrees using the calculated uncertainties as weighting values. This recursion process is briefly outlined below:

- Predict
  - Predict State Estimate
  - Predict Estimate Covariance
- Update
  - Calculate Measurement Residual
  - Calculate Measurement Covariance
  - Calculate Optimal Kalman Gain
  - Update State Estimate
  - Update Estimate Covariance

Two important forms of a Kalman filter are known as "total-state" and "error-state" implementations. In the total-state implementation, a typical IMU model would include both the attitude Euler angles, as well as the gyroscope/accelerometer bias in the system state, in the error-state implementation the integration of the gyroscope signals to attain an attitude estimate would take place externally to the filter, and the system state would consist of the errors between the estimated Euler angles, and the actual Euler angles. For example, such errors are typically estimated by making use of the gravity vector present in the accelerometer readings since it has a known value in the navigational frame.

2.7.2 Extended Kalman Filter

The Extended Kalman Filter (EKF) extends upon the base of the Kalman Filter to allow for the usage of non-linear models. The algorithm is modified such that the model is linearised at each iteration. The EKF is considered the *de facto* standard in navigation systems and GPS [30] . The updated equations are as follows:

$$x(t) = f\left(x(t-1), u(t), w(t)\right)$$

$$y(t) = h\left(x(t), v(t)\right)$$

The general principle of the EKF is that at each iteration the estimation is linearised around the current estimate using the partial derivatives of the process and measurement functions to compute estimates.

2.7.3 Unscented Kalman Filter

In the UKF, the probability density is approximated by a deterministic sampling of points which represent the underlying distribution as a Gaussian. The nonlinear transformation of these points are intended to be an estimation of the posterior distribution, the moments of which can then be derived from the transformed samples. The transformation is known as the Unscented Transform. The UKF tends to be more robust and more accurate than the EKF in its estimation of error [69] .

"The extended Kalman filter (EKF) is probably the most widely used estimation algorithm for nonlinear systems. However, more than 35 years of experience in the estimation community has shown that is difficult to implement, difficult to tune, and only reliable for systems that are almost linear on the time scale of the updates. Many of these difficulties arise from its use of linearization." [30]

2.8 Depth Imaging

2.8.1 Point Clouds

Point Clouds are a way of representing the depth information related to a particular scene. They consist of an array of points and their associated depth measurements, and can be thought of as a grayscale image where the intensity of each pixel corresponds to the depth of that surface. The format utilised in this thesis and by the open source Point Cloud Library is the Point Cloud Data (PCD) file format, an example of which is shown in Listing 2.1.

Listing 2.1: Example Point Cloud Data (PCD) Format

```
# .PCD v0.7 - Point Cloud Data file format
VERSION 0.7
FIELDS x y z
SIZE 4 4 4
TYPE F F F
COUNT 1 1 1
WIDTH 640
HEIGHT 480
```

```
VIEWPOINT 0 0 0 1 0 0 0
POINTS 307200
DATA ascii
-0.54520738 -0.41805041 1.003
-0.54346555 -0.41805041 1.003
...
```

2.8.2 Ranging Techniques

There are numerous different ways to obtain a depth map of a given scene. This section provides information on three such techniques:

- Laser Scanning
- Stereo Vision
- Projected Light Triangulation

Laser Scanning

A laser scanner (also commonly refered to as Light Detection And Ranging, LIDAR or Laser Detection And Ranging, LADAR) uses a laser beam to measure the distance to a target by illuminating it with pulses of laser light. A laser beam of a wavelength between 600 and 1000nm is scanned across the scene by a rotating mirror, either as a single axis (planar scanning) or a dual axis scanner capable of three dimensional imaging.

Because of the nature of scanning the scene in the way that laser scanners do, the update rate is typically limited to lower than alternative scanning methodologies such as stereo vision. However, laser scanning is generally able to achieve a higher quality estimate than other methods.

Stereo Vision

Stereo vision is able to estimate the depth of a given scene using a pair of cameras. Using a feature detection algorithm such as the Scale-Invariant Feature Transform (SIFT), features are identified in a frame that stand out from their surround areas, making them easy to recognise and track. A corresponding point is then found in the same frame from the other camera, and using the known baseline between the cameras a range estimate can be calculated. Depending on how many reliable features are able to be found in each frame the attainable resolution for stereo vision can be limited and context dependent.

The accuracy of ranges derived from stereo vision decreases with distance, and increases with the baseline length. However, stereo vision is limited by the features that are able to be detected, and therefore is unable to function with untextured surfaces such as plain walls.

Projected Light Triangulation

Structured light imaging is a technique whereby a known light pattern (typically a grid, or horizontal bars) is projected onto the scene, which is then captured by a camera. Presuming the relation between the projection device and the camera is known, the distortion observed

in the projected pattern can be used to estimate the three dimensional structure of the scene. The projected light is commonly in the infrared wavelength, so as to be imperceptible to users as well as to prevent interference with other observing cameras or sensors. This is sometimes referred to as invisible structured light.

This is the methodology the Kinect sensor uses to obtain it's depth images. The accuracy of structure light imaging is limited by the resolution of the observing camera, as well as the fidelity of the projected patterns. In the case of the Kinect, several patterns are projected at once to enable a consistent level of accuracy over a larger range of distances.

2.8.3 Registration

After reading the point cloud data from a ranging sensor, a process known as scan registration can be performed to match the current point cloud to the previous point cloud. This is a non-trivial problem, and there are numerous techniques available for achieving this match. The two techniques that are most applicable to the problem at hand, and the two that will be examined in further detail are known as Iterative Closest Point (ICP) and Normal Distributions Transform (NDT).

Iterative Closest Point

Iterative Closest Point (ICP) can be thought of as a brute-force approach to matching point clouds. In spite of it's brute-force nature, ICP is widely used as a technique for matching

3D point clouds. To match two point clouds together, ICP attempts to minimise the the sum of the squared distances between corresponding points in the scans using an iterative process. For each point in the current scan, the corresponding point is the closest point (by distance) in the target scan. This simple point matching process has the downside that if the two scans start fairly far apart, the closest point in the target scan may not necessarily be the correct corresponding point. The vast majority of the processing complexity of ICP is this corresponding point matching process.

Normal-Distributions Transform

The Normal-Distributions Transform (NDT) is a method of registration that models the likelihood of finding a particular point in a given position through a linear combination of normal distributions. This gives a piecewise smooth representation of the reference scan, and allows the application of standard numerical optimisation methods for registration. To accomplish this the point cloud is broken down into a grid of cubes, with a corresponding probability density function (PDF) being calculated for each cube based on it's point distribution. As such, the size of these cubes is an important parameter in the success of the NDT registration attempt.

NDT has the advantage over ICP that a computationally expensive nearest-neighbour search is not required since the points in the reference scan aren't used directly for matching. The normal distributions are calculated in a single pass over the points of the reference scan when it is received. The PDF of each cube is an approximation of the local surface, describing the surface's position, orientation, as well as smoothness. The NDT algorithm has also been extended to take advantage of colour information that may be present in a given point cloud.

To greatly simplify, rather than calculating one PDF per cube, a PDF is calculated per colour (typically RGB), per cube.

To find a registration between distinct point clouds, NDT attempts to find a translation that maximises the likelihood that the points of the current scan lie on the surface of the target scan.

Chapter 3

Algorithm Development

In this section the development of the aided filtering algorithm is detailed in progressive steps. First, a baseline INS is developed which estimates orientation from accelerometers, gyroscopes, and magnetometers and it's performance is compared to that of an commercial off the shelf INS.

Subsequently, the Xbox Kinect sensor is evaluated to determine whether it is suitable for integration as an aiding sensor, and whether it is capable of the required levels of accuracy and performance.

After this groundwork has been laid, the integration of the Kinect as an aiding sensor to the baseline INS can begin. The first element of this integration is devising a methodology for determining weights for each sensor depending on their current accuracy. For this task dynamic error estimation methods are devised for GPS and the Kinect. Once these estimates

are available, their integration into the filtering structure can commence and the performance of the resulting system can be analysed.

The overall performance of the final system is evaluated using a dataset featuring a transition from an outdoor environment to an indoor one. This dataset will exercise all aspects of the implementation, requiring a high level of performance from both the filtering algorithm as well as the error estimations of each sensor to succeed.

3.1 Baseline INS

As a basis of the system a baseline inertial navigation system will be developed. This system is intended to be the basis for the further development of the full aiding sensor system, and will also serve as a point of comparison when evaluating the performance of the selected hardware against commercial INS devices.

The system model used in this baseline filter is intended to be used on any robotic platform and as such does not incorporate specific vehicle kinematics or platform specific functions. The state vector includes a position and velocity estimate in a North-East-Down (NED) Earth centred Earth fixed (ECEF) frame, as well as the system attitude represented in a quaternion. In addition the state includes estimates of the sensor bias, which are modelled as random walks.

The state variables that the system will estimate are the position ($\mathbf{P}$), velocity ($\mathbf{V}$), and orientation (quaternion) ($\mathbf{q}$) vectors, as well as the gyroscope biases ($\mathbf{b}_\omega$).

$$\mathbf{x} = \begin{bmatrix} \mathbf{P} \\ \mathbf{V} \\ \mathbf{q} \\ \mathbf{b}_\omega \end{bmatrix}$$

The inputs to the system are are the measured rotation ($\boldsymbol{\omega}_m$) and measured acceleration ($\mathbf{a}_m$) vectors. The measured rotation is modelled as the true rotation rate ($\boldsymbol{\omega}$) with sensor noise ($\mathbf{w}_\omega$) and an additive bias ($\mathbf{b}_\omega$).

Similarly, the measured acceleration is modelled as the true acceleration ($\mathbf{a}$) with sensor noise ($\mathbf{w}_a$) and with Earth's gravity subtracted (after being rotated into the body fixed frame) $\mathbf{R}_{be}.\begin{bmatrix} 0 & 0 & g \end{bmatrix}^T$ The accelerometer bias is not modelled as it is assumed to be accounted for in the initialisation procedure of the filter implementation (e.g. by taking samples for several seconds).

$$\mathbf{u} = \begin{bmatrix} \boldsymbol{\omega}_m \\ \mathbf{a}_m \end{bmatrix} = \begin{bmatrix} \boldsymbol{\omega} - \mathbf{w}_\omega + \mathbf{b}_\omega \\ \mathbf{a} - \mathbf{w}_a - \mathbf{R}_{be} \begin{bmatrix} 0 & 0 & g \end{bmatrix}^T \end{bmatrix}$$

The state equation is the derivative of the state variables, in this case the first three elements comprise the kinematic equations for a six degree of freedom (DOF) rigid body. The position variable is updated with the velocity from the last iteration, and similarly the velocity is updated with the acceleration from the last iteration (rotated into the ECEF frame from the body frame) and the orientation updated with the gyroscope values through the strapdown equation $\left(\dot{\mathbf{q}} = \frac{1}{2}\boldsymbol{\Omega}(\mathbf{q})\omega\right)$.

$$\dot{\mathbf{x}} = \begin{bmatrix} \dot{\mathbf{P}} \\ \dot{\mathbf{V}} \\ \dot{\mathbf{q}} \\ \dot{\mathbf{b}}_\omega \end{bmatrix} = \begin{bmatrix} \dot{\mathbf{V}} \\ \mathbf{R}_{eb}\mathbf{a} \\ \frac{1}{2}\boldsymbol{\Omega}\boldsymbol{\omega} \\ \mathbf{w}_b \end{bmatrix}$$

By solving the input equation for the true rotation rates and acceleration we can formulate the state equation as a function of previous states, inputs, and process and disturbance noise.

$$\boldsymbol{\omega} = \boldsymbol{\omega}_m + \mathbf{w}_\omega - \mathbf{b}_\omega$$

$$\mathbf{a} = \mathbf{a}_m + \mathbf{w}_a + \mathbf{R}_{be} \begin{bmatrix} 0 & 0 & g \end{bmatrix}^T$$

Combining these equations with the previous state equation gives the state equations in the appropriate function format:

$$f(\mathbf{x}, \mathbf{u}, \mathbf{w}) = \begin{bmatrix} \dot{\mathbf{P}} \\ \dot{\mathbf{V}} \\ \dot{\mathbf{q}} \\ \dot{\mathbf{b}}_\omega \end{bmatrix} = \begin{bmatrix} \dot{\mathbf{V}} \\ \mathbf{R}_{eb}(\mathbf{q}) \cdot (\mathbf{a}_m + \mathbf{w}_a) + \begin{bmatrix} 0 & 0 & g \end{bmatrix}^T \\ \frac{1}{2}\boldsymbol{\Omega}(\mathbf{q}) \cdot (\boldsymbol{\omega}_m + \mathbf{w}_\omega - \mathbf{b}_\omega) \\ \mathbf{w}_b \end{bmatrix}$$

The outputs of the system must also be modelled in a similar fashion. These outputs

from the system model are estimates of the values that the sensors will provide as input, and are used as part of the filter correction process. They are formed as a function of the state variables:

$$\mathbf{y} = \mathbf{h}(\mathbf{x}) = \begin{bmatrix} \mathbf{P} \\ \mathbf{V} \\ \mathbf{B}_b \\ \mathbf{A}_b \end{bmatrix} = \begin{bmatrix} \mathbf{P} \\ \mathbf{V} \\ \mathbf{R}_{be}(\mathbf{q})\,\mathbf{B}_e \\ -P_z \end{bmatrix}$$

Each element of the output mathbftor corresponds to an expected sensor input. In this case the expected position and velocity are simply the current estimates of these mathbftors. The expected magnetic field mathbftor in the body frame ($\mathbf{B}_b$) is modelled as the Earth's constant magnetic field ($\mathbf{B}_e$) transformed into the body frame via a rotation matrix derived from the current orientation estimate. The altitude element ($\mathbf{A}_b$) is simply expressed as the down component of the position mathbftor.

This output mathbftor ($\mathbf{y}$) is used in combination with a measurement mathbftor ($\mathbf{z}$) composed of the readings from the respective sensors, and a sensor noise mathbftor ($\mathbf{v}$).

$$\mathbf{z} = \mathbf{y} + \mathbf{v}$$

The difference between the measurement mathbftor and the output mathbftor is used by the filter's feedback mechanisms to provide corrections to the predicted state.

The implementation of the EKF requires the linearisation of the state equations at the calculation of each prediction step and the linearisation of the output equations at the calculation of each correction step. The Jacobian matrices below are the results of this linearisation.

$$\mathbf{F} = \frac{\delta \mathbf{f}}{\delta \mathbf{x}}$$

$$\mathbf{G} = \frac{\delta \mathbf{f}}{\delta \mathbf{w}}$$

$$\mathbf{H} = \frac{\delta \mathbf{h}}{\delta \mathbf{x}}$$

Where

$$\mathbf{F} = \begin{bmatrix} \mathbf{0}_{3\times3} & \mathbf{I}_{3\times3} & \mathbf{0}_{3\times7} \\ & & \mathbf{F}_{Vq} \quad \mathbf{0}_{3\times3} \\ \mathbf{0}_{10\times6} & & \mathbf{F}_{qq} \quad \mathbf{F}_{qb} \\ & & \mathbf{0}_{3\times7} \end{bmatrix}$$

$$\mathbf{F}_{Vq} = \begin{bmatrix} F_{Vq0} & F_{Vq1} & F_{Vq2} & F_{Vq3} \\ -F_{Vq3} & -F_{Vq2} & F_{Vq1} & F_{Vq0} \\ F_{Vq2} & -F_{Vq3} & -F_{Vq0} & F_{Vq1} \end{bmatrix}$$

$$
\begin{aligned}
F_{Vq0} &= 2\left(q_0 a_{mx} - q_3 a_{my} + q_2 a_{mz}\right) \\
F_{Vq1} &= 2\left(q_1 a_{mx} + q_2 a_{my} + q_3 a_{mz}\right) \\
F_{Vq2} &= 2\left(-q_2 a_{mx} + q_1 a_{my} + q_0 a_{mz}\right) \\
F_{Vq3} &= 2\left(-q_3 a_{mx} - q_0 a_{my} + q_1 a_{mz}\right)
\end{aligned}
$$

$$
\mathbf{F}_{qq} = \frac{1}{2}\begin{bmatrix}
0 & b_{\omega x} - \omega_{mx} & b_{\omega y} - \omega_{my} & b_{\omega z} - \omega_{mz} \\
\omega_x - b_{\omega x} & 0 & \omega_z - b_{\omega x} & b_{\omega y} - \omega_{my} \\
\omega_y - b_{\omega y} & b_{\omega z} - \omega_{mz} & 0 & \omega_x - b_{\omega x} \\
\omega_z - b_{\omega z} & \omega_y - b_{\omega y} & b_{\omega x} - \omega_{mx} & 0
\end{bmatrix}
$$

$$
\mathbf{F}_{qb} = -\frac{1}{2}\mathbf{\Omega}(\mathbf{q})
$$

$$
\mathbf{G} = \begin{bmatrix}
\multicolumn{3}{c}{\mathbf{0}_{3\times 9}} \\
\mathbf{0}_{3\times 3} & \mathbf{R}_{eb}(\mathbf{q}) & \multirow{3}{*}{$\mathbf{0}_{7\times 3}$} \\
\mathbf{\Omega}(\mathbf{q})/2 & \mathbf{0}_{4\times 3} & \\
\multicolumn{2}{c}{\mathbf{0}_{3\times 6}} & \mathbf{I}_{3\times 3}
\end{bmatrix}
$$

$$\mathbf{H} = \begin{bmatrix} \mathbf{I}_{6\times6} & \mathbf{0}_{6\times7} \\ \mathbf{0}_{3\times6} & \mathbf{H}_{Bq} \quad \mathbf{0}_{3\times3} \\ 00-1 \quad \mathbf{0}_{1\times3} & \mathbf{0}_{1\times7} \end{bmatrix}$$

$$\mathbf{H}_{Bq} = \begin{bmatrix} H_{Bq0} & H_{Bq1} & H_{Bq2} & H_{Bq3} \\ H_{Bq3} & -H_{Bq2} & H_{Bq1} & -H_{Bq0} \\ -H_{Bq2} & -H_{Bq3} & H_{Bq0} & H_{Bq1} \end{bmatrix}$$

$$\begin{aligned} H_{Bq0} &= 2\left(q_0 B_{ex} + q_3 B_{ey} - q_2 B_{ez}\right) \\ H_{Bq1} &= 2\left(q_1 B_{ex} + q_2 B_{ey} + q_3 B_{ez}\right) \\ H_{Bq2} &= 2\left(-q_2 B_{ex} + q_1 B_{ey} - q_0 B_{ez}\right) \\ H_{Bq3} &= 2\left(-q_3 B_{ex} + q_0 B_{ey} + q_1 B_{ez}\right) \end{aligned}$$

3.1.1 Performance Comparison

To validate the performance of the baseline INS algorithm, data logs were obtained from a commercial INS, the XSens MTi-G. The MTi-G provides both the unfiltered inertial sensor measurements as well as filtered outputs for both orientation and position. By feeding these inertial measurements into the baseline INS algorithm, it's estimate output can be compared with that of the MTi-G, and determine whether the algorithm is able to function at the required level.

The testing scenario used to obtain the dataset was a simple square pattern. The test was conducted outdoors, and the MTi-G was held level throughout the movement pattern. The logged data was fed into a MATLAB implementation of the baseline INS algorithm, the output of which is shown and compared to the filtered outputs of the MTi-G in Figure 3.1 and Figure 3.2. The position has been converted from geodetic coordinates to ECEF and is shown in metres.

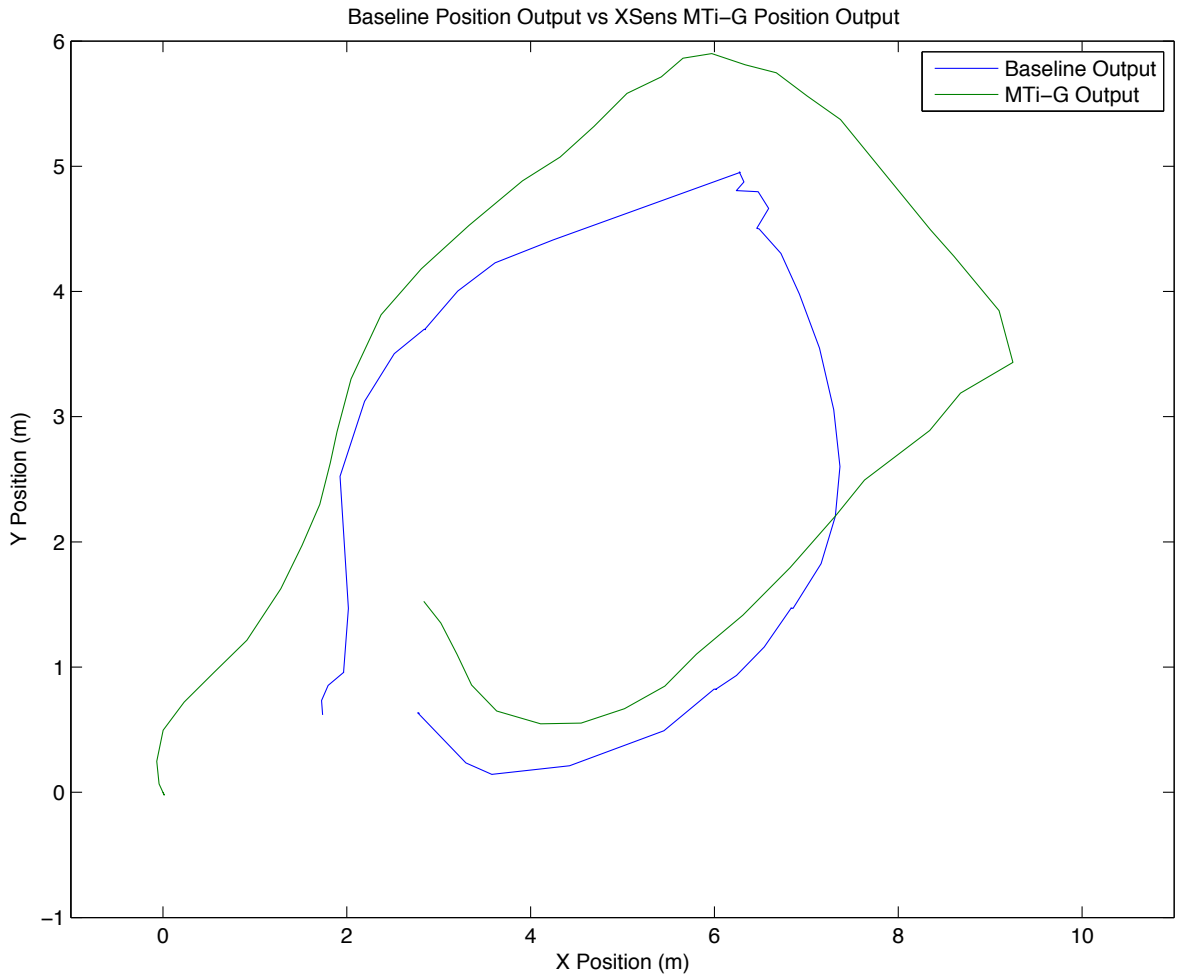
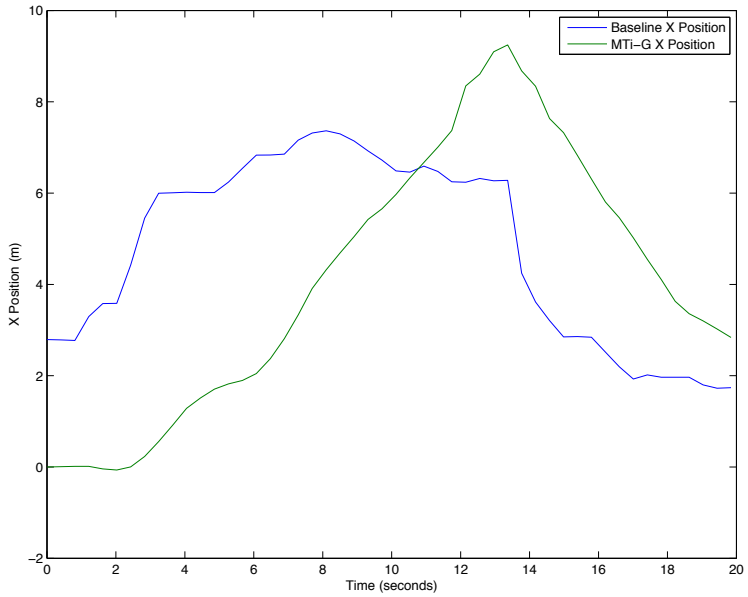
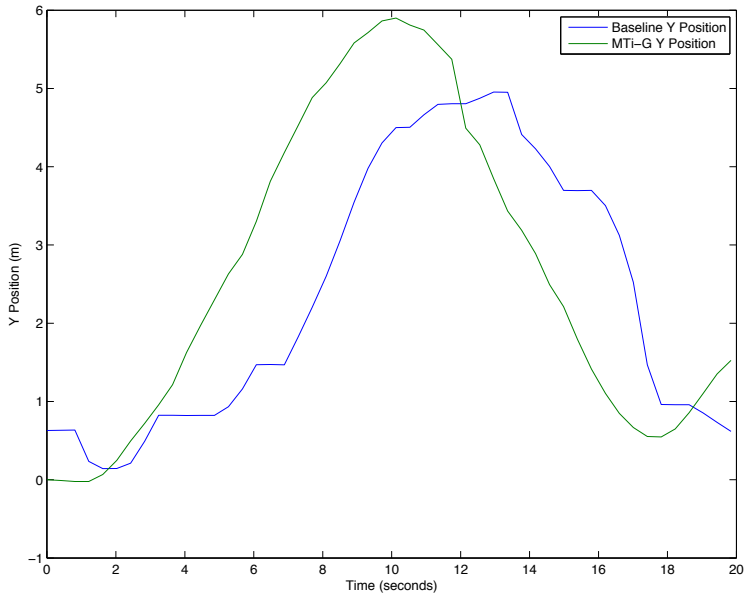


Figure 3.1: Baseline Position Output vs XSens MTi-G Position Output



(a) X Positions



(b) Y Positions

Figure 3.2: Baseline Comparison to MTi-G for X and Y Position

3.2 INS Structure

In order to combine the measurements from a variety of sensors a suitable structure is required for the system. There are many ways to structure and combine the elements of an aided filtering system, and several of the most relevant are presented and compared in the following sections. The general difference between the various structures comes down to one of three main points, a trade-off between computational requirements, robustness and accuracy, and the algorithmic complexity. Groves [22] includes much more detailed analyses on the alternate forms of navigational system structures, and is the main inspiration and source for this section.

3.2.1 Least-Squares Integration

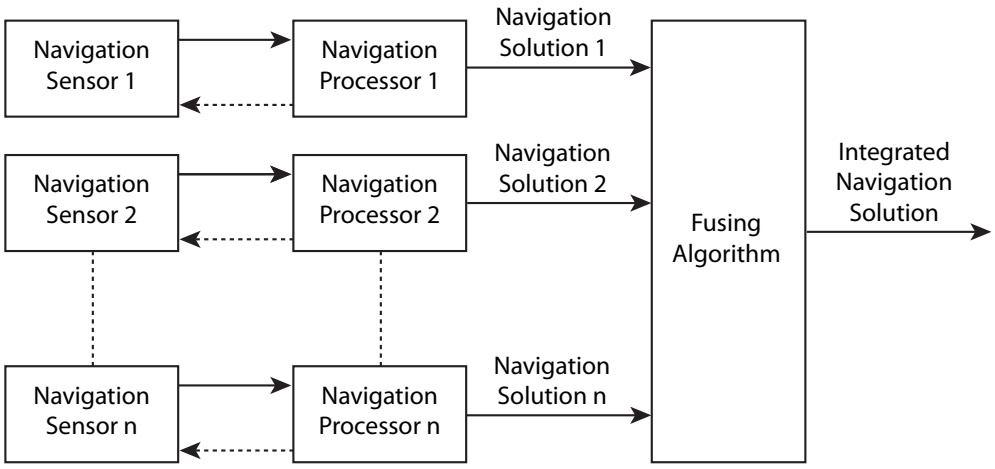


Figure 3.3: Least-Squares Integration

Least-Squares Integration is the simplest of the INS Structures to be presented. Each sensor and processor pair provides both a value for that sensor, along with an estimated error covariance matrix. The data from each sensor is combined in a snapshot fusing algorithm

through a simple weighted least-squares algorithm. The least-squares approach is well suited to 'black box' sensors, sensors that do not provide information on how their errors vary with time, and do not accept feedback from the host system.

The least-squares integration method has the advantage that it is extremely simple, and as such has a low computational requirement. An aside to this simplicity is that it also facilitates easier monitoring of the input sensors, as they are all treated separately. There are severe disadvantages however, the algorithm is unable to deal with sensors with differing data rates or sensory lags, and because there is no feedback mechanism available, least-squares cannot facilitate a dead-reckoning system since it is unable to correct for drift in data e.g. position or velocity. Instead, least-squares will weight the data correspondingly to how much it has drifted, trusting it less and less as time goes on.

3.2.2 Cascaded Integration

Cascaded integration is very similar to least-squares integration, except for the replacement of the snapshot-fusing algorithm with a Kalman filter. The utilisation of a Kalman filter remedies many of the deficiencies that the least-squares integration method suffers from. The Kalman filter allows for different time of arrivals in data, as it can use other sources to compensate for this. It also allows for corrections to be calculated, and fed back to the sources.

Due to the fact that the Kalman filter retains historic data from the sensors, it is able to perform dead-reckoning and maintain a positional solution while keeping track of drift

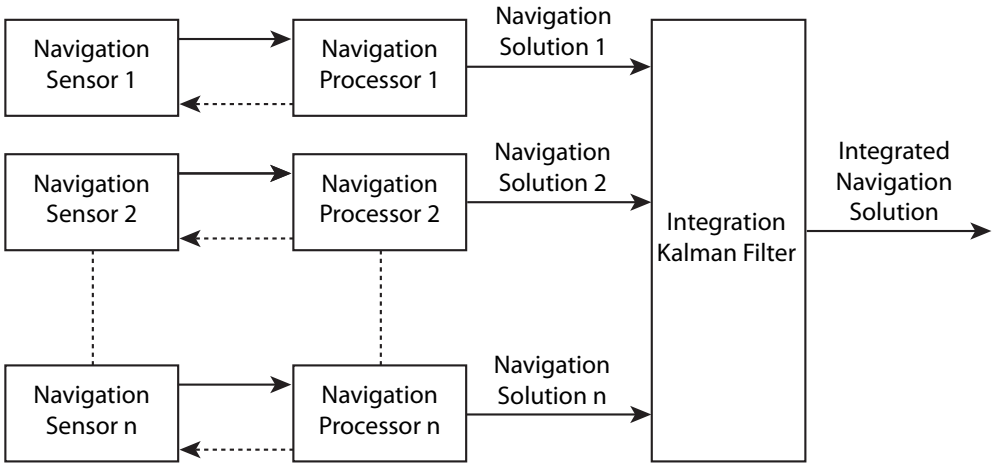


Figure 3.4: Total-State Cascaded Integration

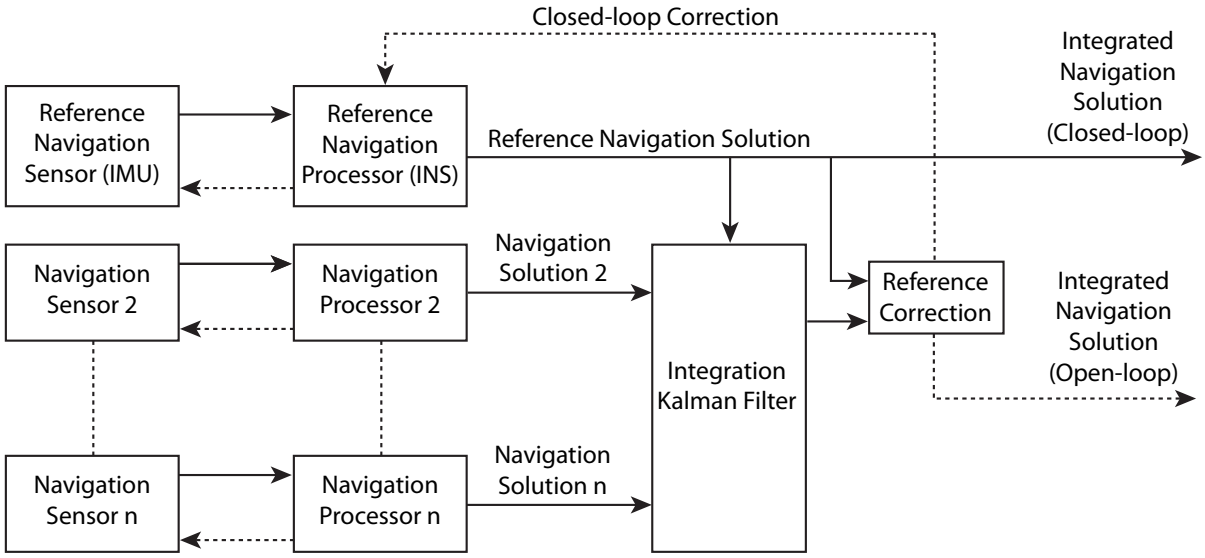


Figure 3.5: Error-State Cascaded Integration

and other errors. The error-state version of cascaded integration is shown in Figure 3.5. As shown in the figure, the reference system is an INS, this reference system is accompanied by various 'aiding' sensors such as GPS. One of the advantages of the error-state form of cascaded integration is that the reference solution can be integrated at a faster rate than the Kalman filter, both reducing computational load and providing a shorter time interval. A typical example of a cascaded integration scheme is a loosely-coupled INS/GNSS system.

3.2.3 Centralised Integration

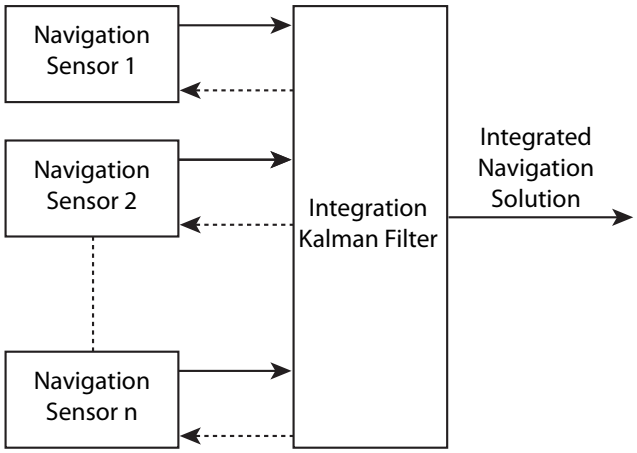


Figure 3.6: Total-State Centralised Integration

In centralised integration, the Kalman filter deals directly with sensor measurements, rather than processed measurements or solutions. A tightly-coupled INS/GNSS system is an example of centralised integration. Due to dealing with the sensor measurements directly rather than a processed form, the centralised integration Kalman filter models the noise and errors of every sensor. This allows the filter to properly correlate all the errors, and to utilise the maximum possible level of available information to help correct each sensor. For this reason the centralised integration architecture is considered the optimal architecture in terms of accuracy and robustness.

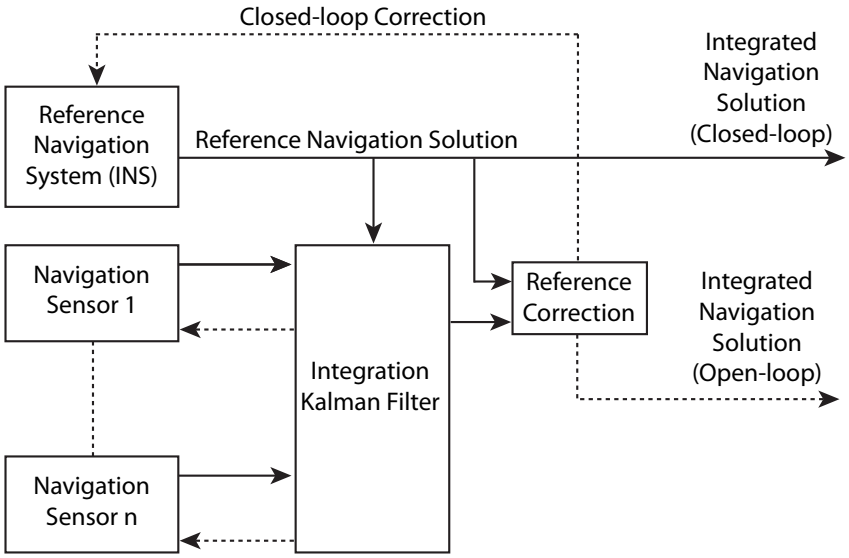


Figure 3.7: Error-State Centralised Integration

Since it models all sensors in the one filter, centralised integration has a higher computational processing load than other architectures. It is also more contingent on correct modelling of sensors and their associated errors, being sensitive to small deviations. Centralised integration is also not well suited for dealing with 'black box' sensors due to the fact it requires accurate modelling of the sensor and associated error sources.

3.2.4 Federated Integration

In the federated integration architecture the reference system is integrated separately, and each aiding sensor is fed through a local Kalman filter. The local Kalman filter outputs can be combined in a number of different manners. The no-reset, fusion-reset, zero-reset, and cascaded federated integration architectures are discussed below.

3.2.5 No-Reset

In the federated no-reset (FNR) integration architecture, the aiding sensors and their associated local Kalman filters are used to estimate the error of a reference navigation system. The navigation solution from the reference system and the estimated errors are combined in a snapshot fusing algorithm, as illustrated in figure Figure 3.8.

The main concern with FNR integration is that since the reference navigation system is common to all local Kalman filters, the estimated errors are no longer independent, and as such the off diagonal elements of the covariance matrix are no longer zero. This can cause the snapshot fusing algorithm to weight the aiding sensor inputs sub-optimally, which in turn can cause a corresponding decrease in the error covariance matrix, falsely identifying the system as more accurate than it actually is.

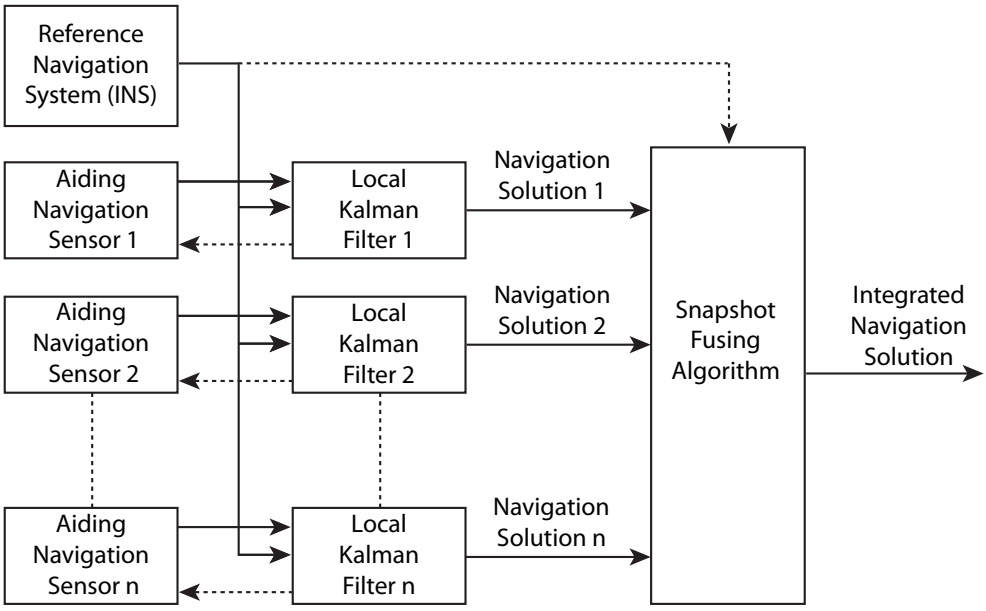


Figure 3.8: No-reset Federated Integration

3.2.6 Fusion-Reset

The federated fusion-reset (FFR) integration architecture allows for the feedback of state estimates and covariance matrices from the snapshot fusing algorithm to the local Kalman filters. By doing this, calibration information is able to be shared between all aiding systems while utilising a lower processing load than the centralised integration architecture. However, the FFR integration architecture is not with its problems. FFR integration is unable to utilise 'black-box' aiding devices and also does not directly support integrity monitoring of the component aiding subsystems. Along with this it provides poorer performance than the centralised architecture, and at a cost of higher system complexity.

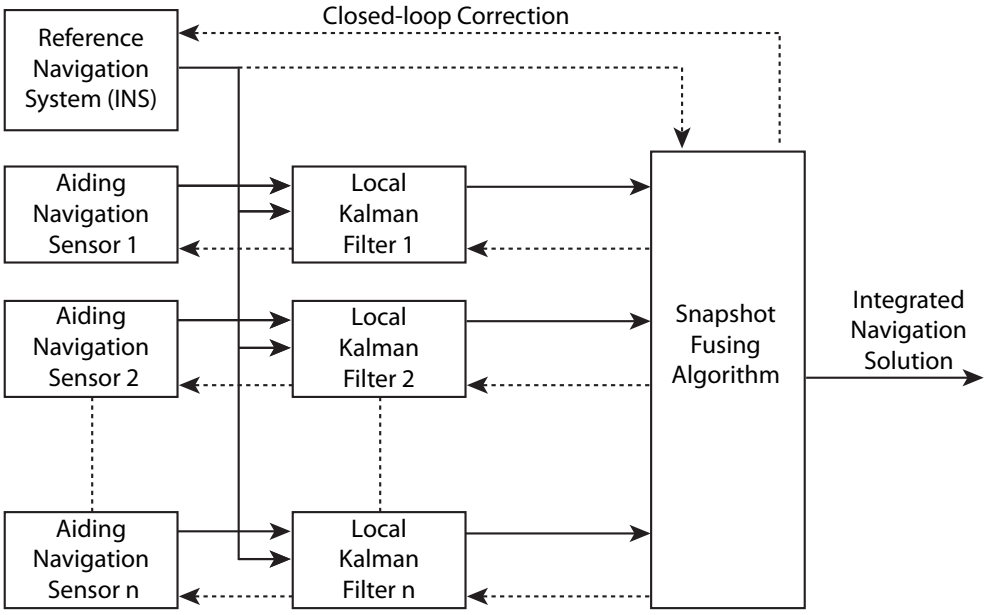


Figure 3.9: Fusion-reset Federated Integration

3.2.7 Zero-Reset

Federated zero-reset (FZR) integration replaces the snapshot fusing algorithm used in FNR and FFR integration with a master Kalman filter. This replacement allows the FZR architecture to deal with differing times of arrival of aiding information. The FZR integration architecture resets all state estimates of the aiding systems local Kalman filters to zero on each iteration, and resets the covariance matrices to their corresponding initialisation values. This resetting prevents the same data from being passed to the master Kalman filter multiple times. The FZR integration architecture can be useful when there is a need to process the measurements faster than the integration algorithm can be executed, e.g. for higher performance integrity monitoring.

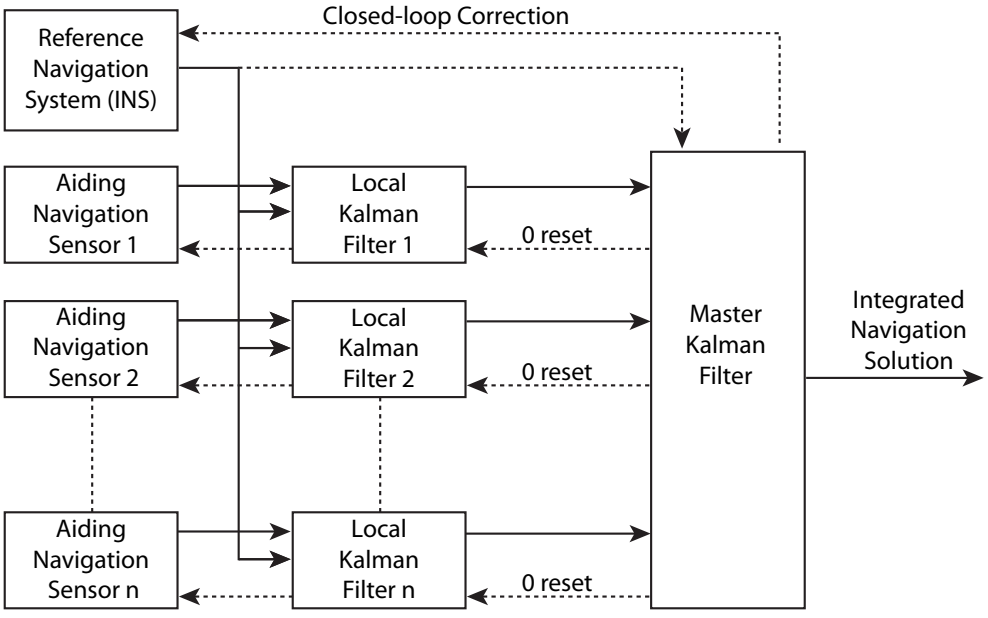


Figure 3.10: Zero-reset Federated Integration

3.2.8 Cascaded

Cascaded federated integration operates on the local Kalman filters without any resets. This causes the integration of the local filters with the master filter to be cascaded, and may cause time-correlated measurement noise. The cascaded federated integration architecture needs extremely precise tuning to operate effectively, and it is due to this fact that it is mostly avoided in practical use. The reason the architecture is presented is due to it's compatibility with aiding systems with 'black-box' local Kalman filters.

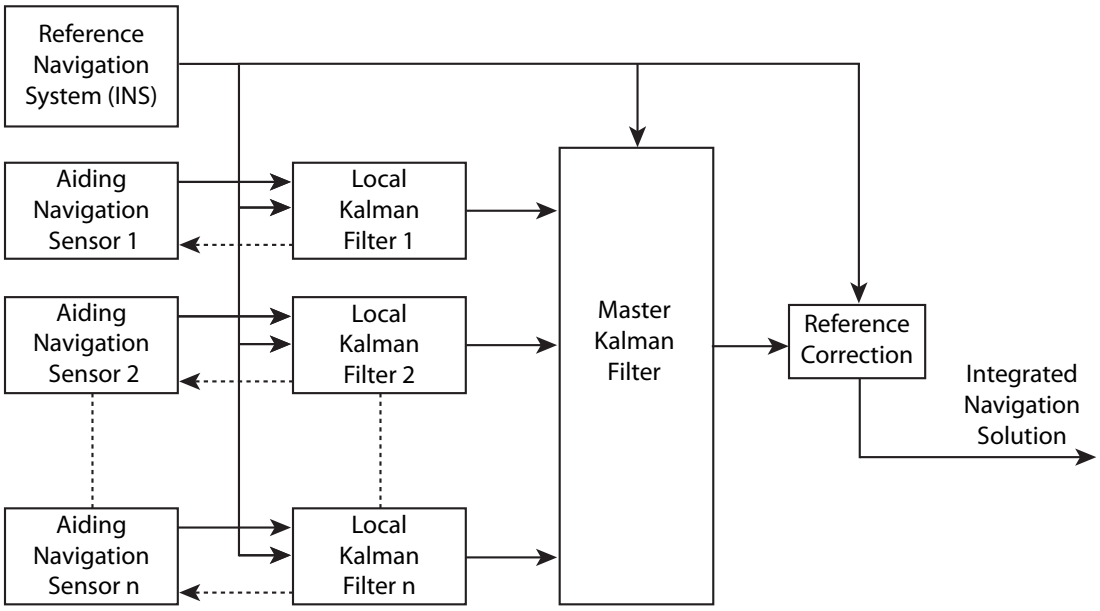


Figure 3.11: Cascaded Federated Integration

3.2.9 Summary

The architecture chosen for the implementation of the dynamic fusion filter is Error-state Cascaded Integration. This structure presents a highly configurable nature, structuring the core INS functions (such as IMU integration) as a separate component and allowing for

the aiding sensors to provide a correction feedback mechanism. A key benefit to the usage of cascaded integration is the ability to iterate the reference navigation solution completely separately from the integration Kalman filter. This naturally lends itself to usage in a system such as this, where the various aiding sensors may have slower than usual update rates.

The proposed demonstration implementation of the fusion algorithm using GPS and Kinect is shown in Figure 3.12 adapted for the cascaded integration structure. The GPS and Kinect act as aiding sensors, and each feed their data into a processing algorithm, which converts that data into a format suitable for the Integration Kalman filter, as well as providing an estimate as to the confidence level of the sensor. The particular methods for determining these confidence levels are discussed in section 3.3.

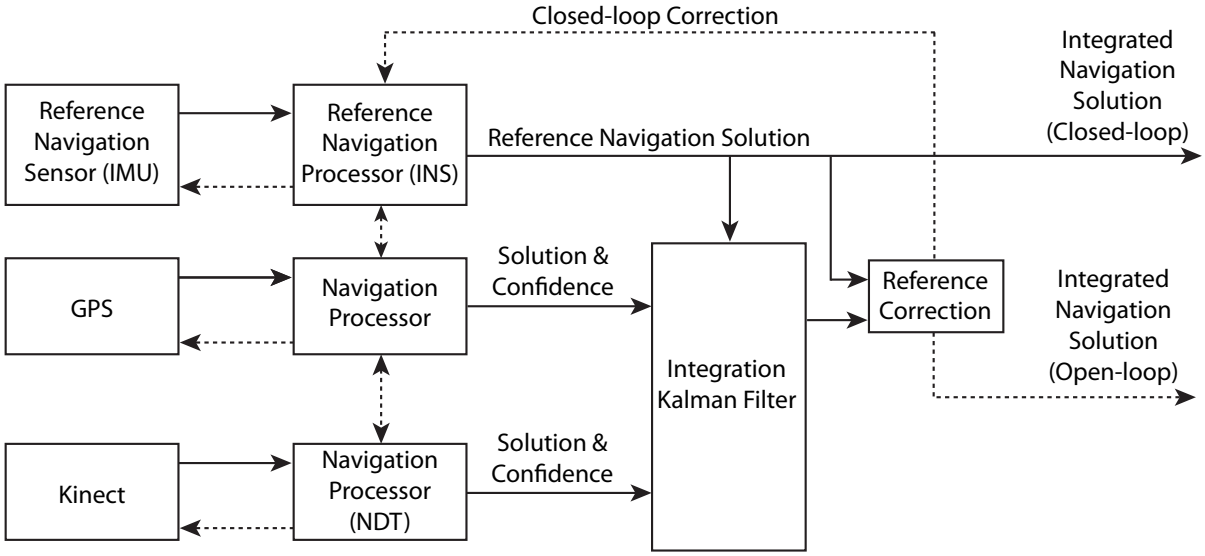


Figure 3.12: Error-state Cascaded Integration with GPS and Kinect as Aiding Sensors

GPS

Figure 3.14 illustrates in more detail the specific interface of the GPS aiding sensor to the

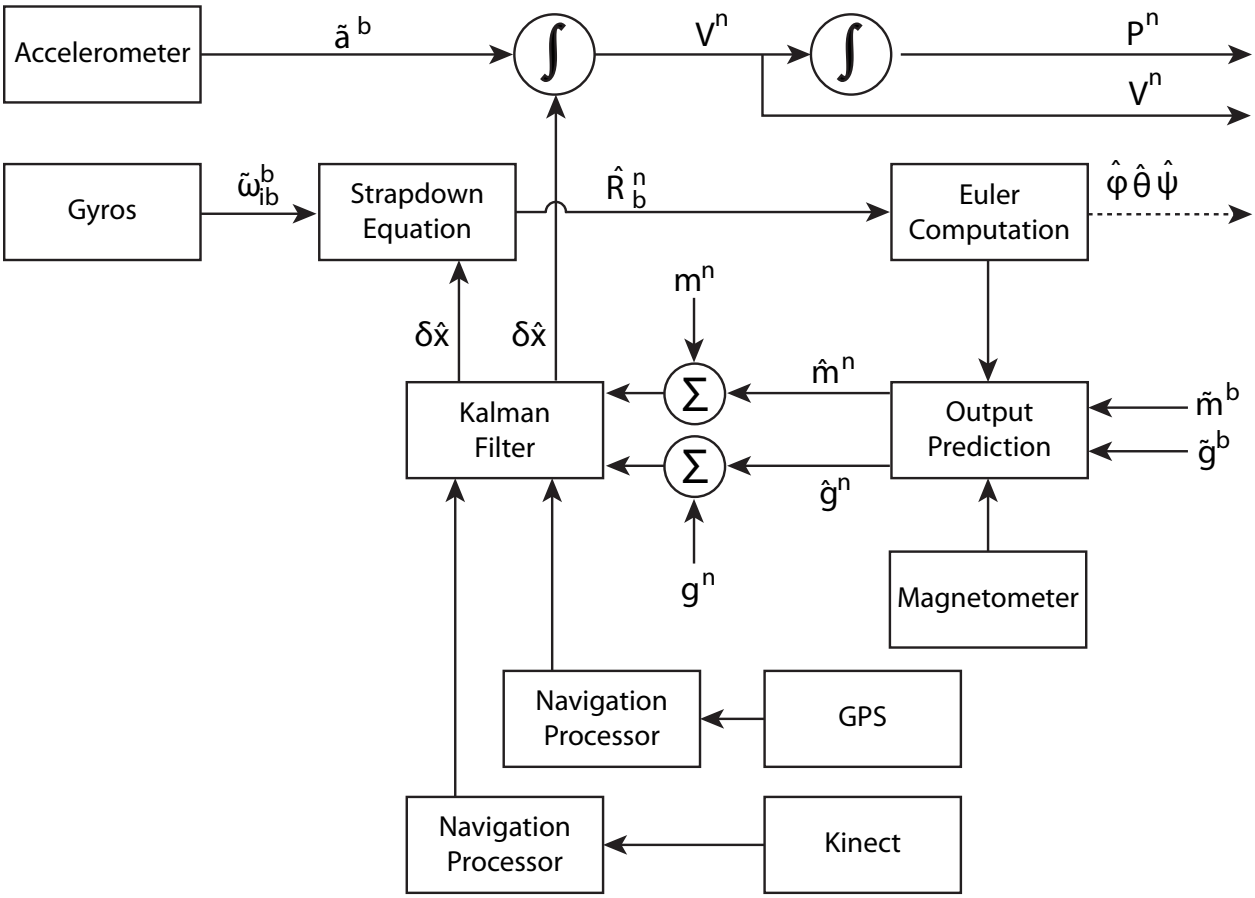


Figure 3.13: Modified System Implementation

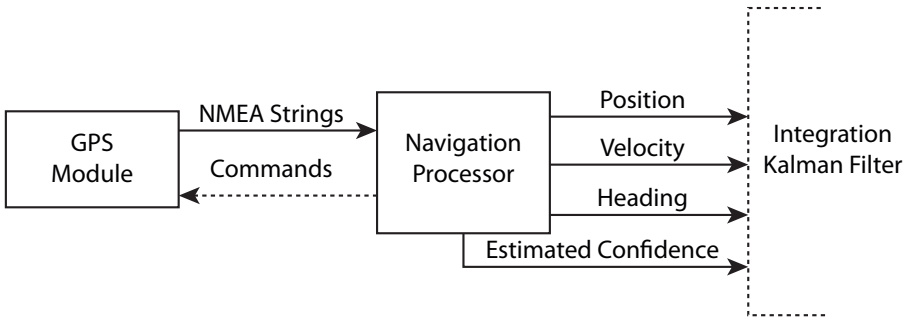


Figure 3.14: GPS Aiding Sensor Overview

integration filter. The output of the GPS module is fed into a processing block which operates on this data in order to output a position, velocity, and heading estimate. In addition this processing block provides a confidence level for the information provided by the GPS.

The specific workings of the processing block are shown in Figure 3.15. Typically a GPS will provide it's data in the form of a sequence of NMEA strings which must be decoded in order to get to the actual data. In the case of GPS, the processing is relatively simple, with the NMEA decoder retrieving the position, velocity, and heading from the NMEA input, as well as passing the current number of satellites and the horizontal and vertical dilution of precision to the confidence estimation block. The specific operation of the confidence estimation will be discussed in subsection 3.3.1.

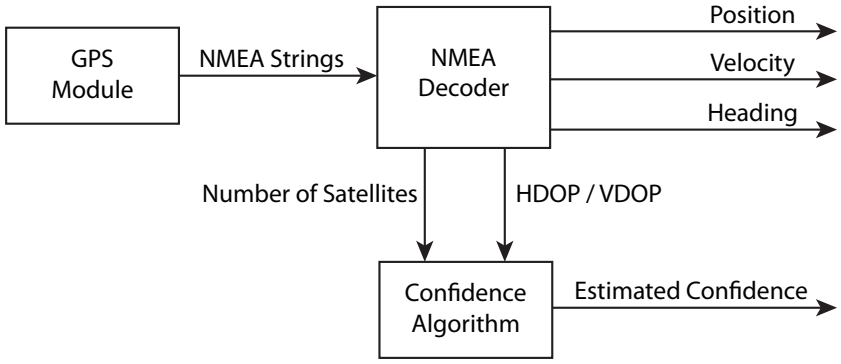


Figure 3.15: GPS Navigation Processor Overview

Kinect

The general principle behind the system augmentation with the Kinect is that as each new depth map arrives, it can be compared to the previous and a transformation between the two can be established. This transformation can then be decomposed into the relevant rotational and translational movements, which can be incorporated into the next iteration

of the filtering algorithm. The Kinect is suitable for augmenting the performance of the gyroscopes, as it provides an alternate source for rotational rate information, but more so the accelerometers. This is due to the translational output of the Kinect being a measurement of velocity, rather than acceleration, and thus removing one level of potential integration errors. The acceleration readings combined with the velocity estimates from the Kinect can be used in concert to provide a much more resilient positional estimate without GPS corrections.

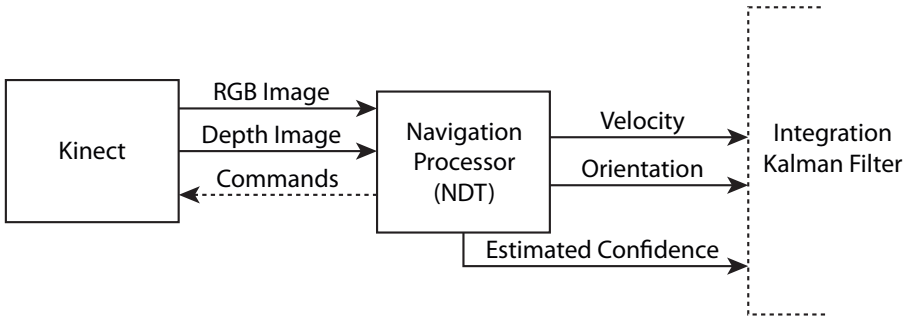


Figure 3.16: Kinect Aiding Sensor Overview

The Kinect interface to the cascaded integration filter is shown in Figure 3.16. Similarly to the operation of the GPS as an aiding sensor, the Kinect module provides input to a processing block, in this case both a colour (RGB) and grayscale image, representing the scene and it's depth respectively. These images are further processed as shown in Figure 3.17 to obtain estimates for the velocity and orientation of the Kinect sensor. In addition these frames are used to provide a confidence level in the outputs from the processing block.

The Kinect navigation processor shown in Figure 3.17 is more involved than that of it's GPS counterpart. First, the depth image from the Kinect sensor is fed to an algorithm which converts it from a bitmap image to a three dimensional point cloud representing the same scene. This point cloud representation is then passed to the NDT (Normal Distributions Transform) algorithm, along with an optional estimate of the current system pose (provided by the Reference Navigation Processor of the Cascaded Integration Filter) and used to

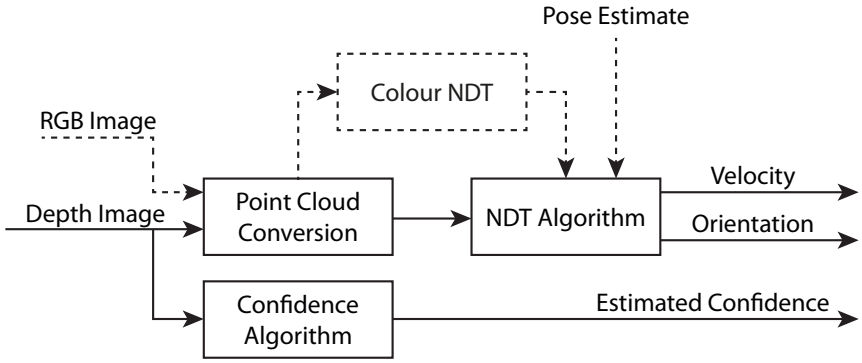


Figure 3.17: Kinect Navigation Processor Overview

estimate the current orientation and velocity of the system. Optionally, the colour image can be passed into the NDT algorithm as well, using the Colour-NDT variant of the algorithm for better registration.

In addition to this processing, the depth image is also passed to the confidence estimator algorithm which uses it to determine a confidence level for the outputs generated from this image. The operation of this confidence algorithm is outlined in subsection 3.3.2.

3.2.10 Reference Navigation Processor

The structure of the cascaded filter calls for the implementation of a reference navigation system. It is this reference system that will accept the corrections produced by the integration filter from the aiding sensors, and use them to correct it's own internal representations of position and velocity. An example reference navigation system is shown in Figure 3.18.

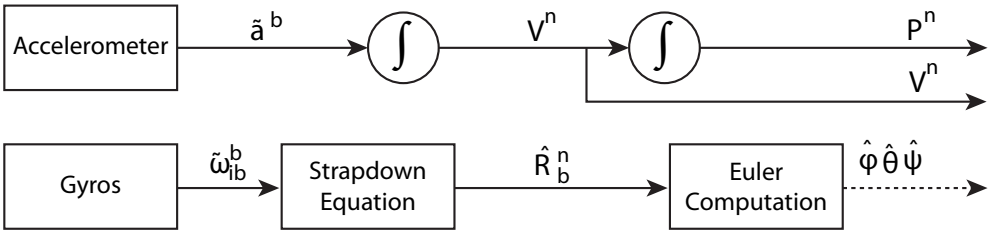


Figure 3.18: Reference Navigation System

3.3 On-line Confidence Estimation

With a suitable filtering structure selected, only one required element remains: an estimate of the accuracy of the aiding sensors. For the purposes of this example implementation, the GPS and the Kinect are serving as aiding sensors. A method for estimating the current integrity of each sensor needs to be formulated based on the data that sensor is provided. Ideally this error estimate is calculated solely based on the output of the sensor which it represents, but in some instances (such as the GPS/Kinect combination) it can prove worthwhile to utilise values from another sensor to aid in the estimation of another's error.

In the combination of GPS with a Kinect, the transition point ideally occurs when the GPS loses it's satellite fix while the Kinect enters it's maximum ranging distance. In reality, these events will rarely occur simultaneously. Due to their usage in consumer devices where the user's perception of reliability is important, almost all GPS modules will continue to output an estimated position for several seconds after losing their satellite fix. As mentioned above, this instance is where using the information from another sensor can be useful in determining the reliability of another. A simple rule for this system implementation which bypasses the false GPS outputs is to utilise primarily the Kinect positional information whenever it is available, and fall back to the GPS when it isn't.

For each of the two sensors an algorithmic method for providing an error estimate needs to be determined, the development of which is detailed in the following sections.

3.3.1 GPS Confidence Estimation

In estimating the error of the GPS solution there are several obvious approaches. Most GPS modules will output their own internal estimates of accuracy, in the form of a horizontal and vertical dilution of precision (HDOP and VDOP respectively) values. An overview of the various values and their meanings is shown in Table 3.1. As mentioned earlier, there is a trend amongst GPS devices to continue outputting previous values (or even interpolating or calculating new ones via dead reckoning) upon loss of a valid satellite fix. In addition, there is typically a lag between the time the satellite signals are received and the calculated position being output, on the order of several update cycles (usually 4Hz) meaning that by the time the position and associated DOP values are updated several seconds may have passed.

An alternate method for estimating the GPS error could be based on the number of satellites currently being used for localisation. The problem with this technique is that it will behave similarly to a step function, rather than a gradual transition, making it harder to transition cleanly between sensors. This is because a GPS module requires a minimum of four satellites in view to solve for position (latitude, longitude, altitude, and clock offset), any more than this and the positional accuracy will be increased, but the increase in accuracy depends on multiple factors and as such is not very predictable. Three satellites will provide a 2D fix (typically altitude is assumed to be a fixed value), but any less and the position of

Table 3.1: Meaning of DOP Values (Source: Wikipedia)

DOP Value	**Rating**	**Description**
1	Ideal	This is the highest possible confidence level to be used for applications demanding the highest possible precision at all times.
1–2	Excellent	At this confidence level, positional measurements are considered accurate enough to meet all but the most sensitive applications.
2–5	Good	Represents a level that marks the minimum appropriate for making business decisions. Positional measurements could be used to make reliable in-route navigation suggestions to the user.
5–10	Moderate	Positional measurements could be used for calculations, but the fix quality could still be improved. A more open view of the sky is recommended.
10–20	Fair	Represents a low confidence level. Positional measurements should be discarded or used only to indicate a very rough estimate of the current location.
>20	Poor	At this level, measurements are inaccurate by as much as 300 meters with a 6 meter accurate device (50 DOP $\times$ 6 meters) and should be discarded.

the receiver cannot be estimated.

Shown in 5.11(a) is the DOP output of a GPS as it undergoes a loss of GPS signal. In this instance, the GPS was moved from an outdoor environment with a full and clear view of the sky to an indoor environment. The transition occurs at approximately 80 seconds into the log, and the graph illustrates that it takes up to 10 seconds (approximately 40 GPS samples) before this loss of signal is reflected in the DOP information.

The algorithm for calculating a confidence level suitable for incorporation in the filtering algorithm was determined empirically from a variety of such datasets. Referring to Table 3.1, the algorithm essentially maps DOP values between 1 and 10 to a range of 1.0 to 0.0 and clamps values higher than 10 to 0.0. This algorithm is shown in Equation 3.1 and it's operation will be specific to the Nexus S GPS module used in the system implementation. Adjustments to the scaling parameters can be made to adapt it's operation to alternate GPS modules in a short time.

$$e_{GPS} = \begin{cases} \frac{10 - DOP_{GPS}}{7} & 0 \leq DOP_{GPS} \leq 10 \\ 0 & DOP_{GPS} > 10 \end{cases} \tag{3.1}$$

3.3.2 Kinect Confidence Estimation

The Kinect employs a technique of projected light triangulation to provide three dimensional scans of the environment. The specifics of this operation are detailed in section 2.8.2. As a

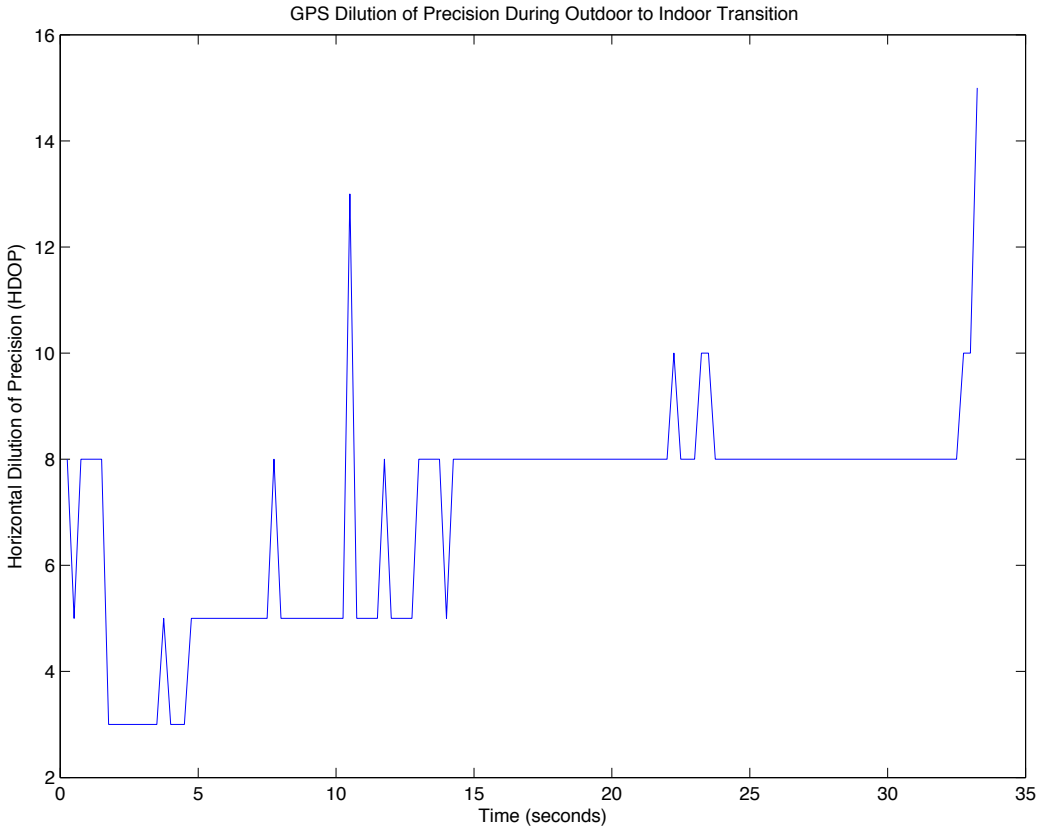


Figure 3.19: Dilution of Precision (DOP) of GPS during Outdoor to Indoor Transition

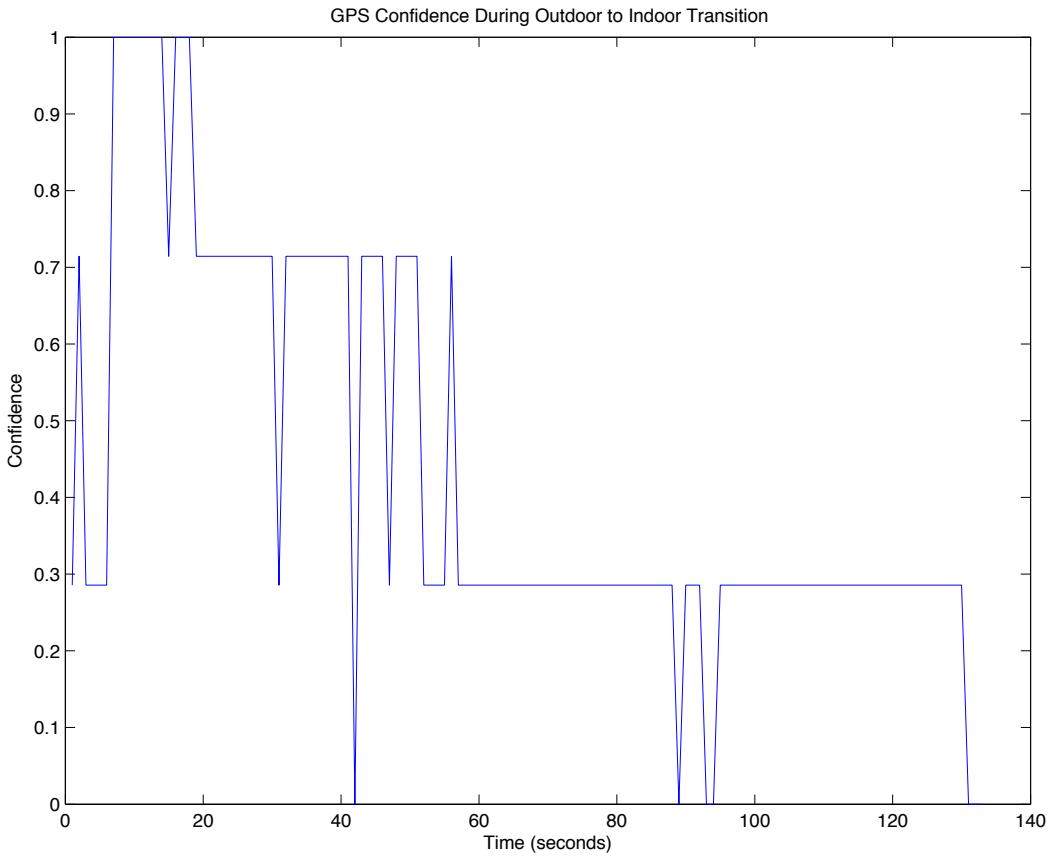


Figure 3.20: Calculated Confidence of GPS during Outdoor to Indoor Transition

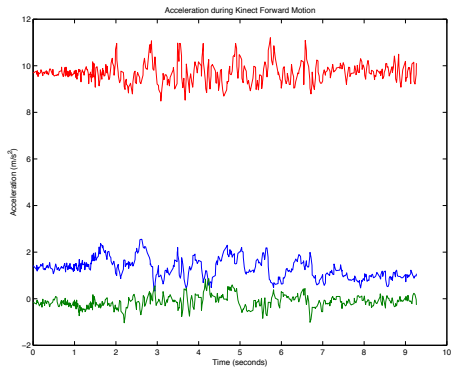
result of the specific variations of techniques the Kinect uses, the operational range is limited to a maximum of 5 metres. In addition, the closer objects are to the Kinect sensors the more accurate their ranging will be. With these constraints in mind a simple methodology for estimating the accuracy of the information the Kinect is providing can be devised.

To establish a baseline for the transitional behaviour of the Kinect a series of tests were conducted. In these tests, the Kinect data was recorded while moving toward a stationary wall. The dataset was passed through the NDT algorithm, with the translational output being compared against an accelerometer recording of the same movement. The results of this test are shown in Figure 3.21.

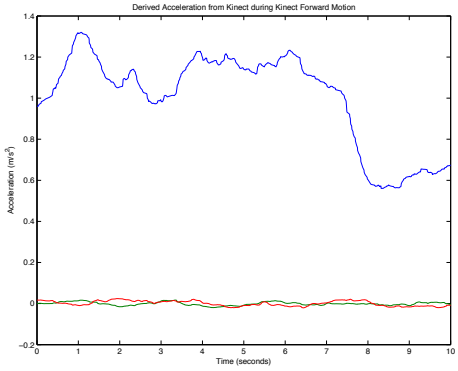
As the Kinect is more accurate the closer it is to the objects it is detecting, it follows intuitively that some measure of the depth of a frame could be used to calculate a confidence estimate for each frame. Shown in Figure 3.22 is a calculated histogram for a subset of frames (every 34th frame) from the captured dataset. What is immediately visible from these graphs is that there is a definite shift in histogram based on the average depth of the frame.

Mean Frame Depth

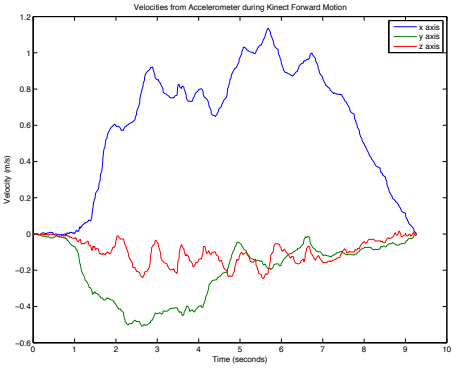
Given the clear relationship between the average depth of the frame, and the accuracy that can be expected from the processing of that frame via NDT, it follows that the mean depth of an image can be used as the basis for determining a confidence level in the Kinect sensor. As each frame arrives, the average depth of all pixels in the image are calculated, and scaled (out of full intensity, or 256 in the case of an 8-Bit image).



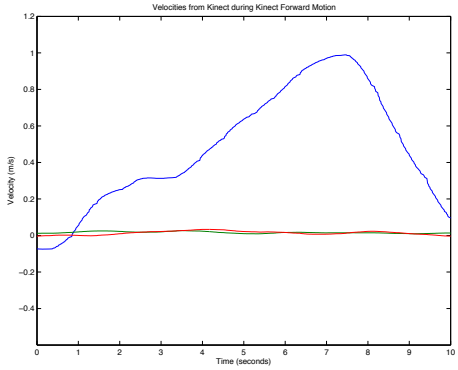
(a) Accelerometer Accelerations



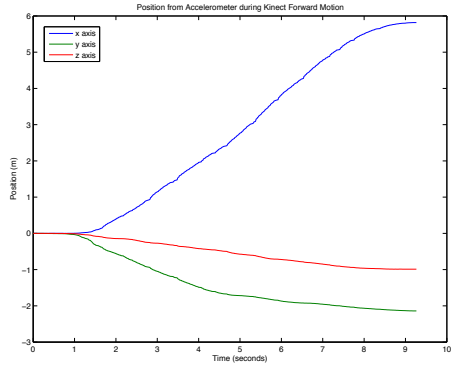
(b) Kinect Accelerations (Derived)



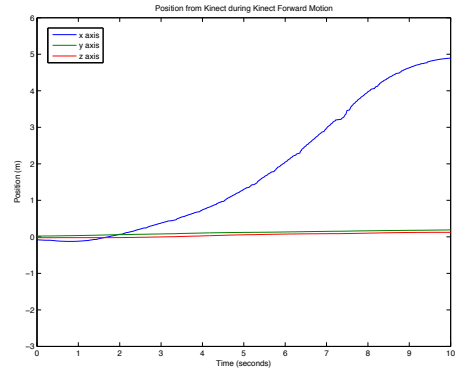
(c) Accelerometer Velocities (Integrated)



(d) Kinect Velocities

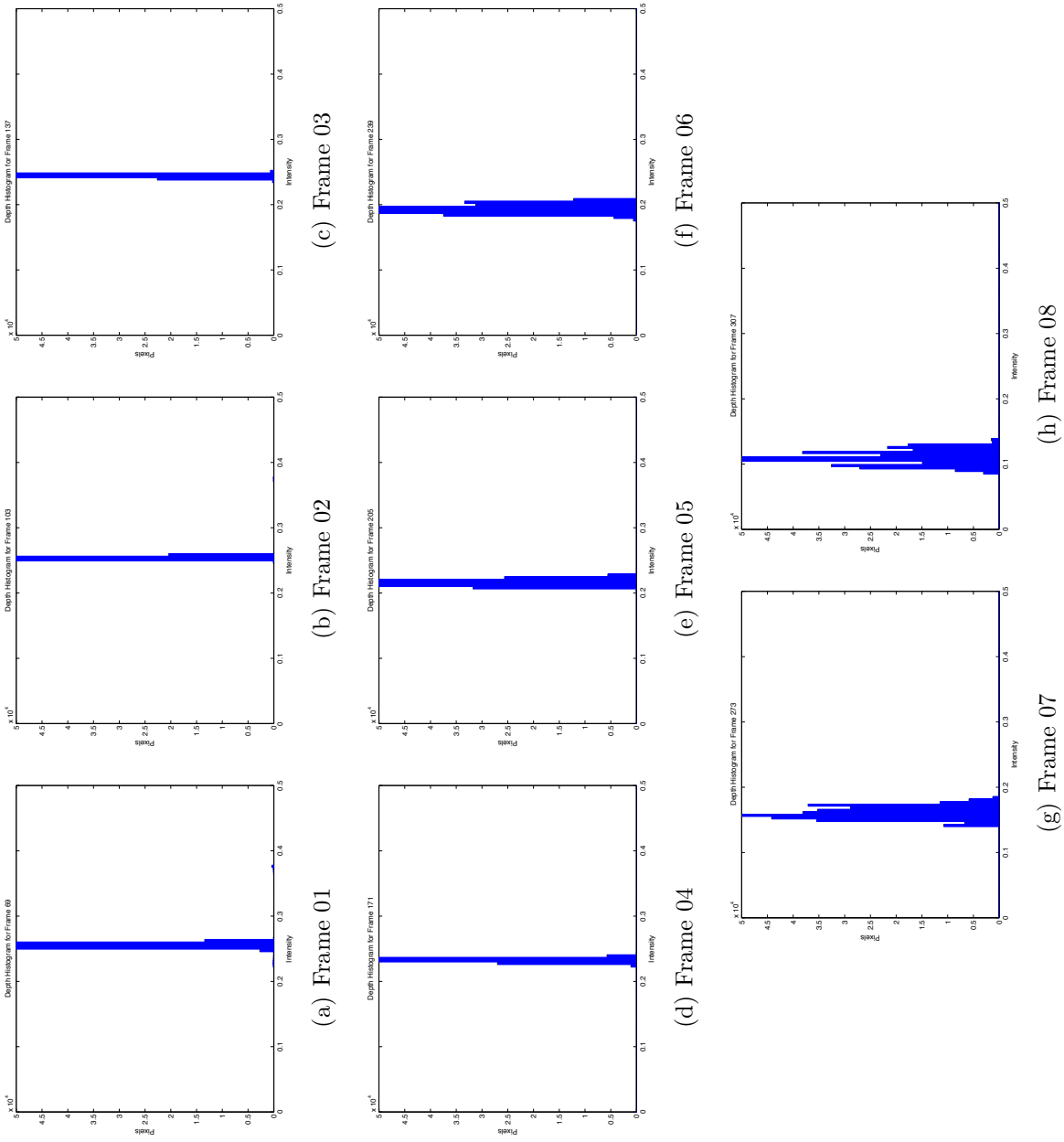


(e) Accelerometer Position (Integrated)



(f) Kinect Position (Integrated)

Figure 3.21: Kinect Comparison to Accelerometer over 5 metre Forward Movement



(a) Frame 01 (b) Frame 02 (c) Frame 03

(d) Frame 04 (e) Frame 05 (f) Frame 06

(g) Frame 07 (h) Frame 08

Figure 3.22: Example Kinect Depth Frames 01-08

Using this method, Figure 3.23 shows the calculated confidence for the 5 metre forward movement dataset.

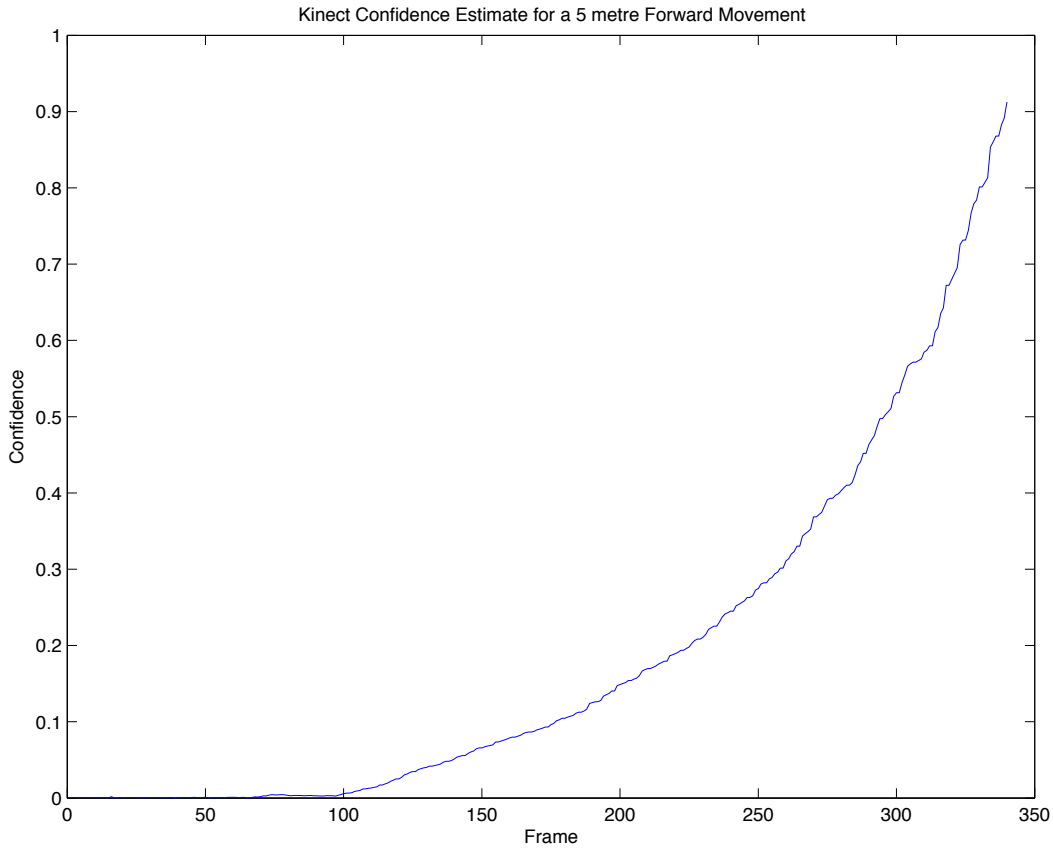


Figure 3.23: Calculated Kinect Confidence over 5 metre Forward Movement

However, given the size of the Kinect depth frames (640 × 480) pixels, or 307,200 pixels total, calculating the average of each frame at the Kinect's full rate of 30fps will take a significant amount of computational power. To this end, the effects of downsampling on the calculated mean were investigated. An example frame was resized to 10% and 5% of it's original size (64 × 48, and 32 × 24) and the mean calculated for each of these frames. The results are shown in Table 3.2.

Table 3.2: Average Depth for Downsampled Frames

Image Scale	Calculated Mean
640 × 480	122.9065
64 × 48	123.0117
32 × 24	123.1224

These results show that for the 32 × 24, the image size is reduced by 90%, but the calculated mean is only altered by 0.18%. Furthering this, the entire example dataset of the Kinect approaching a wall was resized to 5% of it's original size, and the mean calculated for each frame. As the results in Figure 3.24 illustrate, the loss of accuracy introduced by downsampling the input depth frames when calculating the confidence is so small as to be negligible, but the speed gains achieved reduce processing time drastically. However, this method does impose limitations in that it is somewhat dependant on a well formed surface. In situations where strong individual features are present (e.g. fencing posts) additional information must be considered.

3.4 Integration Kalman Filter

The Integration Kalman Filter is responsible for combining the outputs provided by the independent aiding sensor navigation processors. In this case the filter combines the estimated error from the GPS position and velocity, the estimated error from the Kinect integrated position and velocity, and the filter's own error estimates of position and velocity. The confidence values calculated for each aiding sensor are used to weight their respective errors when iterating the filter.

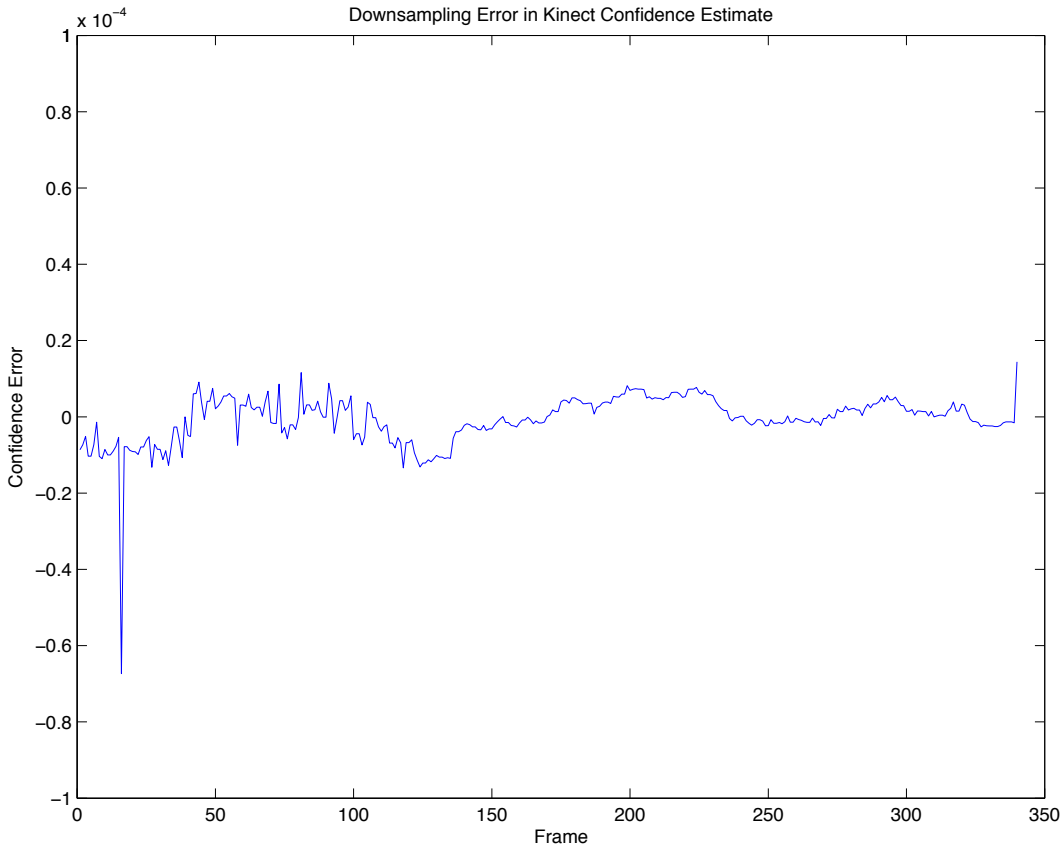


Figure 3.24: Error Introduced in Confidence Through Downsampling

$$\mathbf{x} = \begin{bmatrix} \boldsymbol{\delta}\mathbf{P} \\ \boldsymbol{\delta}\mathbf{V} \end{bmatrix} \tag{3.2}$$

The state vector of the filter is shown in Equation 3.2. This consists of the positional error estimate vector ($\boldsymbol{\delta}\mathbf{P}$) and the velocity error estimate vector ($\boldsymbol{\delta}\mathbf{V}$).

The inputs to the filter are shown in equation Equation 3.6. The GPS confidence (e_{GPS}) and Kinect confidence (e_{Kinect}) are scaled as shown in Equation 3.3 before being passed to the integration filter.

$$s = e_{GPS} + e_{Kinect} \tag{3.3}$$

$$\bar{e}_{GPS} = \frac{e_{GPS}}{s} \tag{3.4}$$

$$\bar{e}_{Kinect} = \frac{e_{Kinect}}{s} \tag{3.5}$$

$$\mathbf{u} = \begin{bmatrix} \bar{e}_{GPS} \\ \bar{e}_{Kinect} \end{bmatrix} \tag{3.6}$$

The measurements provided to the integration filter are shown in equation Equation 3.7. These are calculated by taking the difference between the value provided by the aiding sensor, and that provided by the reference navigation processor, as shown in Equation 3.8.

$$\tilde{\mathbf{z}} = \begin{bmatrix} \boldsymbol{\delta}\mathbf{P}_{GPS} \\ \boldsymbol{\delta}\mathbf{V}_{GPS} \\ \boldsymbol{\delta}\mathbf{P}_{Kinect} \\ \boldsymbol{\delta}\mathbf{V}_{Kinect} \end{bmatrix} \tag{3.7}$$

$$\boldsymbol{\delta}\mathbf{P}_{GPS} = \mathbf{P}_{GPS} - \mathbf{P} \tag{3.8}$$

$$\boldsymbol{\delta}\mathbf{V}_{GPS} = \mathbf{V}_{GPS} - \mathbf{V} \tag{3.9}$$

$$\boldsymbol{\delta}\mathbf{P}_{Kinect} = \mathbf{P}_{Kinect} - \mathbf{P} \tag{3.10}$$

$$\boldsymbol{\delta}\mathbf{V}_{Kinect} = \mathbf{V}_{Kinect} - \mathbf{V} \tag{3.11}$$

The state propagation equation is shown in Equation 3.12. The scaled confidence values for each sensor are simply combined to produce an estimate of the overall error present in the reference navigation position estimate.

$$\mathbf{f}(\mathbf{x}, \mathbf{u}, \mathbf{w}) = \begin{bmatrix} \boldsymbol{\delta}\mathbf{P} \\ \boldsymbol{\delta}\mathbf{V} \end{bmatrix} = \begin{bmatrix} (\bar{e}_{GPS} \times \mathbf{P}_{GPS} + \bar{e}_{Kinect} \times \mathbf{P}_{Kinect}) \\ (\bar{e}_{GPS} \times \mathbf{V}_{GPS} + \bar{e}_{Kinect} \times \mathbf{V}_{Kinect}) \end{bmatrix} \tag{3.12}$$

Equation 3.13 shows that the current overall error estimates are used to generate the difference between the next measurement, and update the Kalman gain.

$$\mathbf{h}(\mathbf{x}) = \begin{bmatrix} \boldsymbol{\delta}\mathbf{P} \\ \boldsymbol{\delta}\mathbf{V} \\ \boldsymbol{\delta}\mathbf{P} \\ \boldsymbol{\delta}\mathbf{V} \end{bmatrix} \tag{3.13}$$

3.5 Summary

In this section the development of the aided filtering algorithm is detailed in progressive steps. First, a baseline INS is developed which estimates orientation from accelerometers, gyroscopes, and magnetometers and it's performance is compared to that of an commercial off the shelf INS.

Subsequently, the Xbox Kinect sensor is evaluated to determine whether it is suitable for integration as an aiding sensor, and whether it is capable of the required levels of accuracy and performance.

After this groundwork has been laid, the integration of the Kinect as an aiding sensor to the baseline INS can begin. The first element of this integration is devising a methodology for determining weights for each sensor depending on their current accuracy. For this task dynamic error estimation methods are devised for GPS and the Kinect. Once these estimates are available, their integration into the filtering structure can commence and the performance of the resulting system can be analysed.

The overall performance of the final system is evaluated using a dataset featuring a

transition from an outdoor environment to an indoor one. This dataset will exercise all aspects of the implementation, requiring a high level of performance from both the filtering algorithm as well as the error estimations of each sensor to succeed.

Chapter 4

System Implementation

4.1 Hardware

4.1.1 Decision To Use A Smartphone

As the hardware requirements were outlined, and the selection of the system components began, it was realised that there already existed a platform with these capabilities - and more. For a far lower cost than developing custom hardware and with a user base far larger than any custom hardware could hope to achieve, smartphones presented the ideal implementation platform for the type of system aimed to be developed. As one of the key goals of the system is to be robust to low quality sensors, the commodity sensors present in the phones are more than sufficient. In addition, almost every phone sold today comes

equipped with a GPS module, and further, supports aided GPS by taking advantage of the fact that mobile phone towers are fixed locations. For these reasons, it was decided that investigation into a smartphone implementation deserved merit.

In addition, a phone has a built in battery, screen, and well established development environment. The ability to be self-powered is of enormous benefit, particularly when dealing with light weight data acquisition platforms such as the GWS Formosa aircraft detailed in section 4.1.9 or the Quadcopter detailed in section section 4.1.9.

4.1.2 Commercial IMUs

Contrary to the processing platform, the available options for a commercial IMU/AHRS unit are fairly limited (taking into account reasonable cost). Out of the available options, three commercial inertial measurement units were selected for further investigation and viability for usage, which are detailed in the following subsections.

Microstrain 3DM-GX1

The Microstrain 3DM-GX1[1] is a commercial AHRS that combines accelerometers, gyroscopes, and magnetometers with a 16-Bit ADC and microcontroller to provide inertial measurements as well as an estimate of orientation. The 3DM-GX1 supports both RS–232 and RS–485, and can output at 19,200, 38,400, and 115,200 baud rates. It features a 16-Bit ADC

[1] http://www.microstrain.com/inertial/3DM-GX1

for highly accurate analog voltage measurements from the sensors, and weighs 25.8 grams as an OEM package (no enclosure).

The 3DM-GX1's accelerometers are capable of measuring up to ±5g standard, with the options for either ±1.7g or ±18g. The accelerometers have a bias stability of ±0.005g and nonlinearity of 0.2%. The gyroscopes are able to measure rotational rates up to ±300deg/s standard, with options for ±1200, ±600, and ±50 deg/s available. The gyroscopes have a bias stability of 0.7 deg/s and nonlinearity of 0.2%.

At an approximate price of $1495, the 3DM-GX1 is affordable in comparison to other commercial AHRS given it's capabilities, but this price still presents a substantial barrier for entry.

VectorNav VN–100

The VectorNav VN–100[2] is a miniature AHRS sized at 24x22x2.5mm and weighing just 3 grams. In this minuscule package the VN–100 combines an accelerometer, gyroscope, magnetometer, and a 32-Bit processor. It is capable of outputting orientation estimates at up to 200Hz in full 32-Bit floating point precision. Both an SPI interface at up to 18MHz and an RS–232 interface at up to 921,600 baud are supported as output channels for the orientation estimate produced by the on board EKF algorithm.

The VN–100's accelerometers are capable of measuring ±8g, with a bandwidth of 260Hz and a noise density of $400\text{mg}/\sqrt{Hz}$. Similarly, the gyroscopes are capable of rates up to

[2]http://www.vectornav.com/index.php?option=com_content&view=article&id=42&Itemid=14

$\pm$2000 deg/s with a bandwidth of 256Hz and a noise density of 0.005deg/s/$\sqrt{Hz}$.

The VectorNav VN–100 is available for as low as $500 in an OEM chip module, or $800 (+$100 for interface cable) for an enclosed module. Both of these modules are only calibrated for operation at 25 degrees celsius. An expanded operational range of –40 degrees celsius to 85 degrees celsius is available for an additional $300. Similarly to the Microstrain 3DM-GX1, the VN–100 presents a low price for the level of performance offered, but also presents a high barrier to entry.

Razor IMU

The Sparkfun Razor IMU[3] features an accelerometer, gyroscope, and magnetometer combined with an Atmel ATmega328 microcontroller. The advantage the Razor IMU has over the previous devices is that the on board microcontroller is fully programmable and the firmware is open source. The ATmega328 operates at 8MHz and outputs the orientation estimate over an RS–232 connection at 57,600 baud by default. The Razor IMU is available directly from Sparkfun for $124.95. This low cost goes a long way to facilitate adoption, however the barebones nature of the board and immaturity of the firmware makes system integration more of a challenge.

The Razor IMU utilises Invensense ITG–3200 gyroscopes capable of measuring rotational rates of up to $\pm$2000 deg/s with a noise density of 0.03 deg/s/$\sqrt{Hz}$. It also uses an Analog Devices ADXL345 accelerometer capable of measuring up to $\pm$16g, and a Honeywell HMC5883L magnetometer.

[3]http://www.sparkfun.com/products/10736

The Razor IMU has several novel benefits towards it's usage. The ITG–3200 gyroscopes are designed with operational frequencies between 24kHz and 36kHz which is almost double that of most other MEMS based gyroscopes. The advantage this presents is in usage on electric motor powered aircraft, such as quadcopters. The frequencies that the motors are operating at frequently overlaps with typical gyroscope resonant frequencies, causing their rotational rates measurements to saturate and effectively blocking their operation. The higher operational frequencies of the ITG–3200 avoid this issue.

4.1.3 iOS vs Android

There are two main smartphone platforms available to consumers today, Apple's iPhone and Google's Android. Both platforms were evaluated to determine their suitability as an implementation platform for the sensor system, with hardware capabilities (required sensors, and external hardware interfacing) being a primary requirement.

iOS Evaluation

The original iPhone was released in 2007, and in terms of sensors only featured an accelerometer (STMicroelectronics LIS331DH) and a GPS module. Since then, Apple has added both a gyroscope (STMicroelectronics L3G4200D) and a magnetometer. The STMicroelectronics L3G4200D is a digital gyroscope, able to measure rotational rates at 250 deg/s, 500 deg/s, and 2000 deg/s. It uses a combination I2C/SPI interface for communication, and provides rate data in 16-Bit as well as 8-Bit temperature data.

The main benefit of choosing iOS as the platform for implementation is the standardisation of hardware. Similarly to development of a custom hardware component, each iPhone is identical, and thus similar behaviour can be expected of each device. However, the locked-down nature of the platform leads to the inability to interface external hardware with the phone which prevents the addition of external aiding sensors (at least in a simple manner). Apple does have a hardware program known as MFi[4] (Made For iPhone) that allows custom hardware to be interfaced to the phone, but this is only available to large hardware manufacturing partners.

Android Evaluation

Android is a Linux based smartphone operating system. It is primarily targeted at ARM processors, but also runs on X86 and a variety of other platforms. Google purchased the original form of the Android operating system from Android Inc. in 2005. The first commercial phone released that ran the Android operating system was the HTC Dream, released on October 22nd, 2008. Unlike iOS, where there is only one core hardware platform, Android is designed as a general operating system intended to run on a wide variety of devices. As of 2011, there are over 200 different hardware platforms running a version of the Android operating system. On one hand this large number of devices presents an impressively large potential install base, but on the other presents a very diverse range of hardware, not all of which has the minimum capabilities required. Some of the more prominent Android features are listed below:

- Application framework enabling reuse and replacement of components

[4] https://developer.apple.com/programs/mfi/

- Dalvik virtual machine optimised for mobile devices
- Integrated browser based on the open source WebKit engine
- Optimised graphics powered by a custom 2D graphics library; 3D graphics based on the OpenGL ES 1.0 specification (hardware acceleration optional)
- SQLite for structured data storage
- Media support for common audio, video, and still image formats (MPEG4, H.264, MP3, AAC, AMR, JPG, PNG, GIF)
- GSM Telephony (hardware dependent)
- Bluetooth, EDGE, 3G, and WiFi (hardware dependent)
- Camera, GPS, compass, and accelerometer (hardware dependent)
- Rich development environment including a device emulator, tools for debugging, memory and performance profiling, and a plugin for the Eclipse IDE

The Android platform was selected for the prototype implementation of the system for several reasons. It is an open source operating system, which allows for modifications if required, as well as greatly increasing the ease of debugging at the low level that is required for components of the system. As well, it supports development in both Java and C/C++, and the mixing of both code from both languages through the Java Native Interface (JNI). This allows for the use of the Java programming language for high level tasks that require verbosity and clarity, and the use of C for tasks which require absolute performance or very low latency.

Finally, Google have introduced an Android library known as ADK (Accessory Development Kit) which facilitates the interfacing of USB devices with an Android smartphone. Specially designed hardware can simply be plugged into the phone (ADK requires Android 2.3.4 or higher), and using standard library functions an application on the phone can communicate with it.

4.1.4 Nexus S

The Nexus S was selected as the hardware platform for implementation of the system. A key advantage that the Nexus S had at the time of hardware selection was that it was one of the only smartphones available with a gyroscope built in. In addition, it is a flagship phone for Google, and as such will receive ongoing support and firmware updates.

The processor of the Nexus S is a Samsung Exynos 3110 with an ARM Cortex A8 based CPU capable of up to 1GHz clock speeds. The GPU is a PowerVR SGX 540 designed by Imagination Technologies, it supports OpenGL ES 1.1/2.0 and has a fill rate of up to 20 Million polygons per second. The Nexus S has 512MB of RAM available and 16GB of on board NAND flash memory.

The full suite of sensors[5] is outlined in Table 4.1 and several key sensors and their approximate positions within the phone are shown in Figure 4.2.

The Nexus S has a 3.7V Lithium-Ion battery with a capacity of 1500mAh. This gives an

[5]KR3DM and K3G are the model numbers the Android operating system returns when queried about the Accelerometer and Gyroscope respectively. These models do not exist, the actual hardware is likely custom manufactured by STMicroelectronics.

Figure 4.1: Nexus S (Images from Samsung)

Table 4.1: Nexus S Built In Sensors

Sensor	Model
Accelerometer	KR3DM
Gyroscope	K3G
Magnetometer	AK8973
Light Sensor	GP2A
Proximity Sensor	GP2B
GPS	BCM4752
Front Camera	640×480
Rear Camera	2560×1920

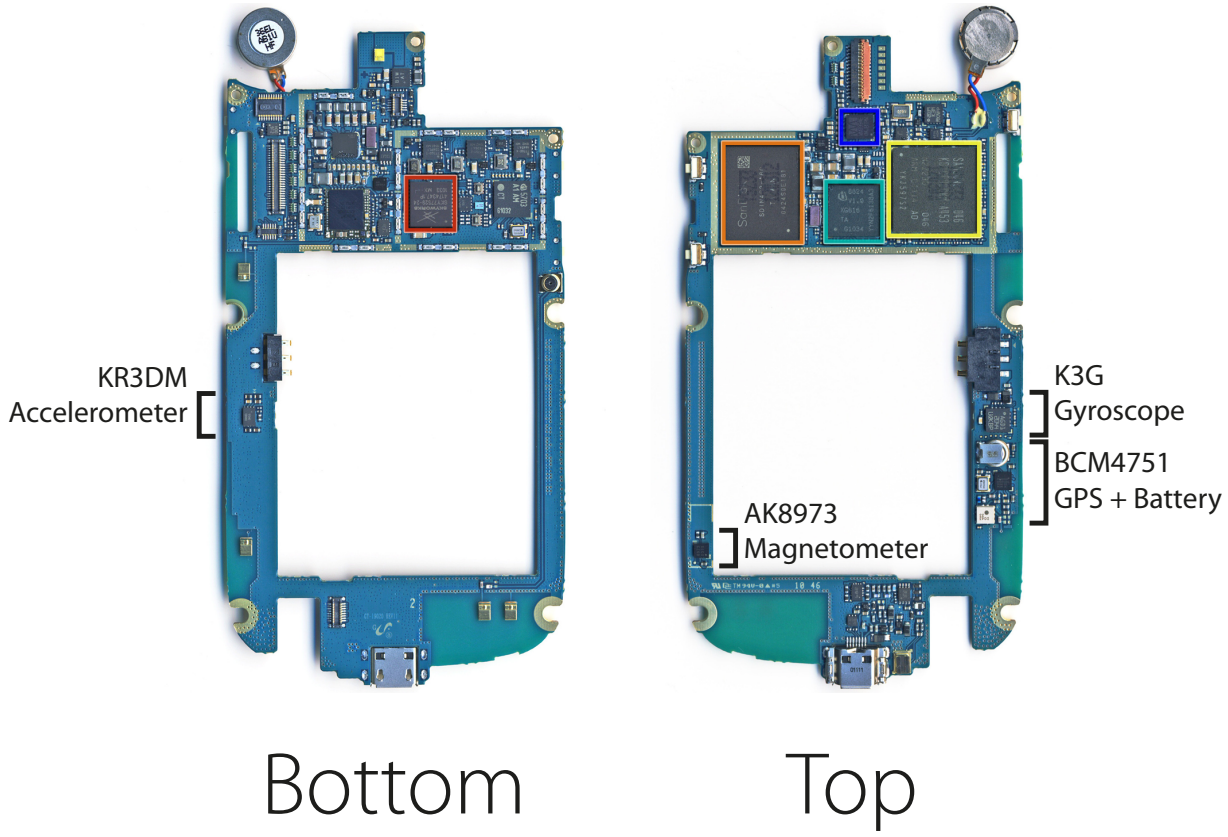


Figure 4.2: Nexus S Sensors (Images from iFixit.com)

approximate runtime of 4 hours with the inertial sensors and GPS powered on.

USB Host Mode

The USB standard specifies three operational modes for micro-usb connections:

- **Slave Mode**, where the device is controlled by the host (master)
- **Host Mode**, where slave devices can be connected and controlled
- **On-The-Go (OTG)**, designed for mobile devices to interface to simple slave hardware

The USB control chip used in the Nexus S is capable of operating in all three USB modes, but by default will only operate as a slave device for file transfers and charging. By modifying the USB driver, the Nexus S can be commanded to recognise attached USB devices. Due to the Android operating system being open source, and the Nexus S being Google's flagship phone, there are numerous repositories available for both the kernel and driver source code. The modification required to enable host (and OTG) mode are relatively minor, and are available in the following git repository: *https://github.com/sztupy/samsung-kernel-herring*

After the software modifications to the kernel have been made, the phone must be put into a special mode (known as 'rooting') to enable unsigned code to be installed. Again, due to the fact that the Nexus S is a flagship phone for Google, the process is well documented and straightforward. After the modified kernel has been loaded, an additional application is

available in the Android Marketplace to edit the settings of the USB device (e.g. forcing a specific USB mode, or enumerating all connected devices for debugging).

With the modified USB driver installed, accessing the Kinect device itself is very similar to that of a standard Linux operating system. The block device (typically /dev/USB*) is passed to the hardware interfacing library, in this case a modified version of the OpenNI driver, allowing for abstracted communication with the Kinect. From this point both RGB and IR frames are able to be retrieved from the Kinect device, converted into point cloud format, and passed to a processing library such as PCL where they can be transformed into inertial movements suitable for integration into the filtering algorithm.

4.1.5 Kinect

The Kinect sensor interfaces to an Xbox or computer through a standard USB connection, and presents itself as a slave device. In addition to the power it draws through the USB connection (up to 500mA) the Kinect requires an auxiliary power supply of 12V. In the tests conducted the 12V was supplied via the included power transformer, and for deployment on a robotic platform this power could easily be sourced from an on board battery. Even if the 12V supply is able to deliver enough power to operate the Kinect hardware, the device will not function unless it is able to draw sufficient current from the USB interface as well. As the Nexus S is unable to supply power through it's built in USB port, this necessitates the construction of a custom USB cable to inject 5V power into the Kinect.

The power injection cable is simply a USB extension cable, passing through the Data+

and Data- signals without modification, but splicing an additional USB male connection onto the power and ground lines (or in the case of deployment on a robot, a suitable power connection for 5V). This additional USB male connection can be plugged into an extra USB port, which will supply up to 100mA of current by default.

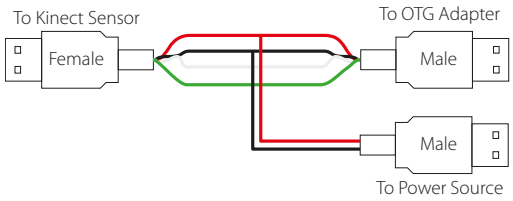


Figure 4.3: Custom USB Power Injection Cable

In addition to the constructed power injection cable, a USB OTG conversion cable is required to convert the standard USB type A connection into a Micro USB connection suitable for the Nexus S.

4.1.6 PhoneDrone Board

The most inhibiting limitation of using a a smartphone as an implementation platform for a robotic system is the lack of external hardware interfaces. The majority of Android powered devices have a micro-usb port, configured as a slave device for interfacing to a computer and charging.

In order for the Android ADK (Accessory Development Kit) to connect hardware to phones that will only enumerate as slave devices, the accessory hardware itself acts as a host to which the phone connects. For this to work correctly the accessory hardware needs to supply power to the phone. Shown in Figure 4.4 is an example of an Android accessory, the

3D Robotics PhoneDrone board. This board is based on the design of the ArduPilot, an open source Arduino based autopilot, with some modifications.

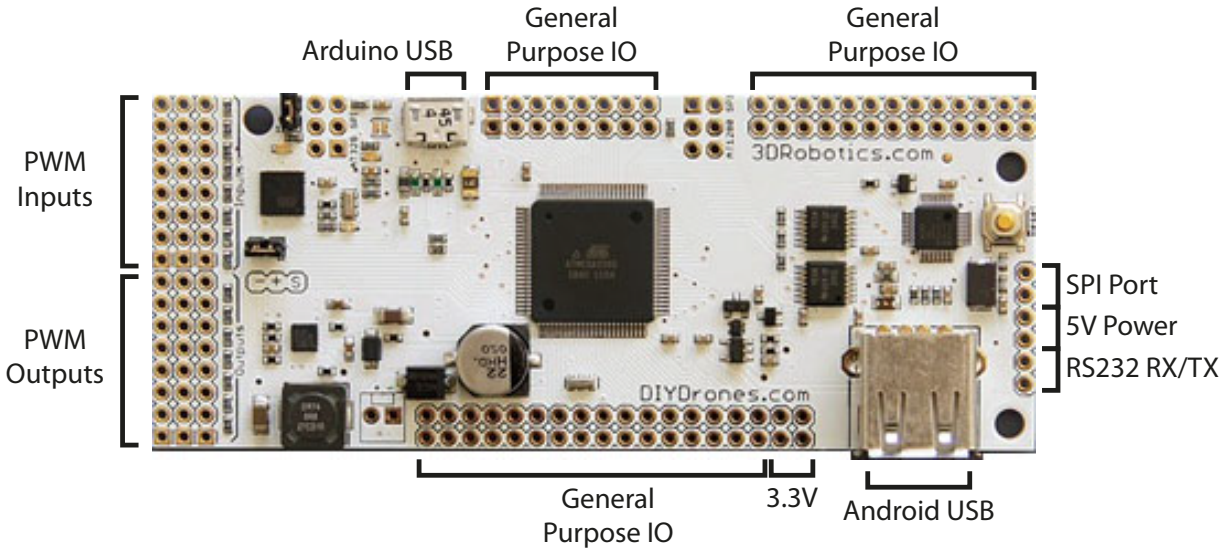


Figure 4.4: PhoneDrone Android Accessory

The specifications of the PhoneDrone board are listed below:

- 8 Input & Output PWM Signals
- Native USB Host Master (MAX3421)
- Native USB Slave (ATmega32-AU)
- Arduino Compatible
- ATmega2560 as Main Controller
- Three Serial Ports
- Built in 5V/2A Switch Power Regulator (Input Range 6V–36V)
- Built in 3.3V LDO Power Regulator

Software Interface

4.1.7 Kinect Data Platform

To obtain datasets of RGB and depth images from the Kinect a computer logging system was constructed as shown in figure Figure 4.5. The system consists of the Kinect connected to a Macbook Pro running Ubuntu Linux 10.10 with the Nexus S used as a reference for accelerations/velocities/rotations. The data from the Nexus S was logged alongside the images captured from the Kinect.

Initially the concept was to have the Kinect interfaced to the Nexus S, and operate the entire system as a standalone unit. Unfortunately, due to errors encountered during the usage of the OpenNI and PCL libraries on the Android platform, images were unable to be received from the Kinect sensor, although enumeration and rudimentary communication were achieved.

4.1.8 Commercial INS

For the purposes of evaluation and as a source of ground-truth data during the development of the two commercial inertial navigation systems were used. The XSens MTi-G and the SBG Systems IG–500N are shown in Figure 4.6.

Figure 4.5: Kinect Data Logging Platform

(a) SBG Systems IG-500N

(b) XSens MTi-G

Figure 4.6: Commercial INS Devices

XSens MTi-G

The XSens MTi-G features an integrated AHRS consisting of MEMS accelerometers, gyroscopes, and magnetometers, along with a GPS and a static pressure sensor (barometer). The sensor fusion algorithm runs on the on board Digital Signal Processor (DSP) at a rate of up to 120Hz. The filtered outputs are available through an RS–232 interface at up to 1MBaud. The MTi-G is priced at approximately $5,500AUD.

		rate of turn	accele-ration	magnetic field	tempe-rature	static pressure
Unit		[deg/s]	[m/s^2]	[mGauss]	[°C]	[Pa]
Dimensions		3 axes	3 axes	3 axes	-	-
Full Scale	[units]	+/- 300	+/- 50	+/- 750	-55 - +125	$30 – 120 \cdot 10^3$
Linearity	[% of FS]	0.1	0.2	0.2	<1	0.5
Bias stability	[units 1σ]	1	0.02	0.1	0.5	100 /year
Scale factor stability	[% 1σ]	-	0.03	0.5	-	-
Noise density	[units /√Hz]	0.05	0.002	0.5 (1σ)	-	4
Alignment error	[deg]	0.1	0.1	0.1	-	-
Bandwidth	[Hz]	40	30	10	-	-
A/D resolution	[bits]	16	16	16	12	9

Figure 4.7: XSens MTi-G IMU Performance Statistics

IG–500N

The IG–500N claims to be the world's smallest GPS enhanced AHRS, and is powered by an embedded Extended Kalman Filter (EKF). It has on board MEMS accelerometers, gyroscopes, and magnetometers, and a GPS and barometer. It is capable of providing it's output at a rate of 100Hz through an RS–232 interface at bauds up to 921,600. The IG–500N is calibrated for operation between –40 and 85 degrees celsius, and also has calibration methods for both soft and hard iron effects on the magnetometer. The IG–500N costs approximately

GPS Receiver specification		Attitude and Heading from XKF-6G	
Receiver Type:	50 channels GPS L1, C/A code GALILEO OpenService L1	**Dynamic Range:**	
GPS Update Rate:	4 Hz	**Pitch:**	± 90°
Pos/Vel Update Rate:	120 Hz	**Roll:**	± 180°
Accuracy Position SPS:	2.5 m CEP	**Heading:**	± 180° (0...360°)
SBAS:	2.0 m CEP	**Angular Resolution:**	0.05 deg
Start-up Time Cold start:	29 s	**Static Accuracy:**	
Re-acquisition:	<1 s	**Roll/Pitch:**	<0.5 deg
Tracking Sensitivity:	-160 dBm	**Heading:**	<1 deg
Timing Accuracy:	30 ns RMS	**Dynamic Accuracy:**	
Operational Limits:		**Roll/Pitch:**	1 deg RMS
Maximum Altitude:	18 km	**Heading:**	2 deg RMS
Maximum Velocity:	600 m/s (2160 km/h)	**Max update rate:** **Autonomously:**	 120 Hz
Max dynamics GPS:	4 g	**PC/raw data:**	512 Hz

Figure 4.8: XSens MTi-G GPS Performance Statistics

$4000AUD.

Standard Sensors	Accelerometers	Gyroscopes	Magnetometers	
Measurement range	± 5 g	± 300 °/s	± 1.2 Gauss	Refer to sensors options table
Non-linearity	< 0.2% of FS	< 0.1% of FS	< 0.2% of FS	
Bias stability	± 5 mg	± 0.5 °/s	± 0.5 mGauss	Over temperature range
	-	< 0.1 °/s	-	Kalman filter stabilized
Scale factor stability	< 0.1%	< 0.05%	< 0.5%	Over temperature range
Noise density	0.25 mg/√Hz	0.05 °/s/√Hz	0.01 mG/√Hz	
Alignment error	< 0.1°	< 0.1°	< 0.1°	
Bandwidth	50 Hz	40 Hz	500 Hz	Additional software filter available
Sampling rate	10 000 Hz	10 000 Hz	1 000 Hz	

Figure 4.9: IG-500N IMU Performance Statistics

4.1.9 Platforms

Throughout the development cycle of the system, a number of hardware platforms have been utilised for data gathering purposes. The first of these platforms was a 1/3 scale Piper Cub

model aircraft. This aircraft predates the introduction of the Kinect sensor, and so was only used for very early development of the system.

The next plane used was a small scale foam trainer craft, the GWS Formosa 3D. Normally this plane would be unusable for a project such as this as it doesn't have the payload capacity for the logging computer, batteries, INS, etc., however the light weight and self contained operation of the Nexus S bypasses these issues.

The final plane used is a Robin 2160. This plane was purchased as a replacement for the Piper Cub, and has a much more aerodynamic profile. It also has a greatly increased payload, allowing the installation of an expanded version of the computer system from the Cub with several improvements, as well as the permanent installation of a stereo vision camera rig down the tail of the aircraft.

In addition to the aforementioned planes, the system was also experimented with on a quadcopter platform. This work was focussed more on the implementation of Ardupilot on the Android operating system, and used the PhoneDrone board as a way to interface servos to the Nexus S. In addition to the quadcopter autopilot work, a model trainer powered by a glow engine was also used for Ardupilot purposes.

Piper Cub

The Piper Cub was used as an initial data gathering platform. Using the onboard XSens MTi-G INS, logs were gathered from the inertial sensors and GPS as well as the filtered output. This data was used to verify the correct operation of the INS algorithms independently of

the aiding sensor interfaces.

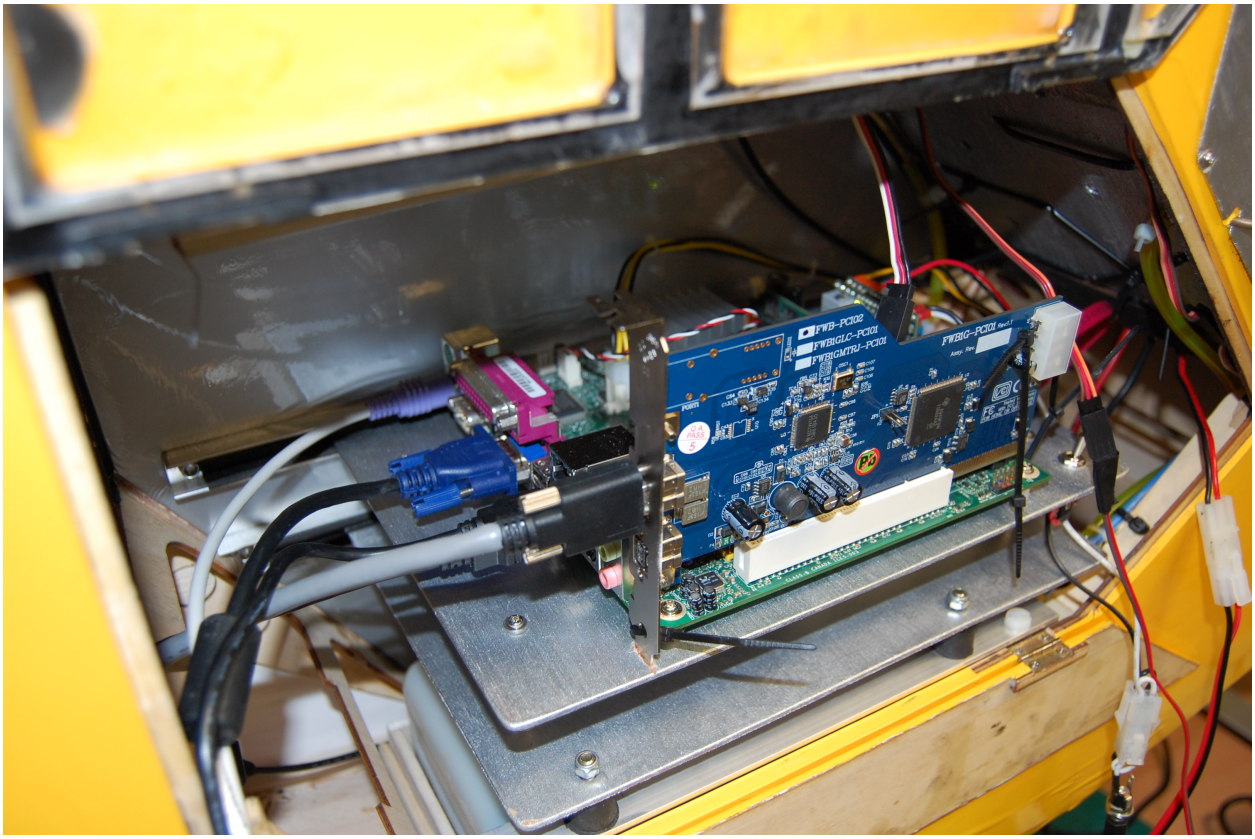


Figure 4.10: Piper Cub with On Board Computer Platform

GWS Formosa

The GWS Formosa is a 3D capable training plane. It is electric powered, and has a flight time of approximately 20 minutes from a 3S (3 Cell), 2200mAh battery. The Formosa was mainly used as a testing ground for the Ardupilot autopilot, but has also carried the Nexus S in flight. The small size and electric powered nature of the Formosa make it well suited to the rapid testing cycle the development of an autopilot necessitates.

Figure 4.11: GWS Formosa

Robin 2160

The Robin 2160 was brought in to replace the ageing Piper Cub platform. The Robin has several distinct advantages over the Piper Cub, namely:

- Far easier access to the internal bays
- Low wing (Wing is at the bottom of the body, more aerobatic)
- Much higher payload capacity

The Robin is powered by a Desert Aircraft 50CC two-stroke engine, and has a 500mL fuel tank which allows for up to 30 minutes of flight time, depending on flight conditions and payload.

The Robin is currently equipped with an onboard computer system powered by light-weight lithium polymer batteries. At the core of this computer system is an Intel D945GCLF2 motherboard powered by a dual core Intel Atom 330 clocked at 1.6GHz. The computer is equipped with 4GB of RAM and two SSD hard drives configured in Raid 0 for maximum throughput from the stereo vision system. The stereo vision cameras are Point Grey Flea2 CCD units connected to the computer through Firewire 800, and attached to a custom designed 1 metre baseline camera rig. Also attached to the rig to serve as a baseline for pose estimates is an XSens MTi-G INS, which interfaces to the computer through RS232 over a USB connection.

The Nexus S was flown on several test flights on the Robin platform, and recorded data from it's built in inertial sensors and GPS suitable for comparison with the Robin's inertial navigation system.

Quadcopter

The quadcopter platform was built specifically to verify the operation of the proposed system. A quadcopter is a natural platform for a vision based sensor such as a Kinect, as it is slow moving and is also able to move in any direction at any time.

The quadcopter was built from standard hobbyist components, the frame comes as a laser cut plywood kit, and a set of four motor and ESC combinations. The control board used for quadcopter specific stabilisations is a Hobby King Quadcopter Control Board (V1) which is based on the KKmulticopter open source design. This control board is responsible for control of the quadcopter itself, with navigational inputs coming from the RC receiver or the PhoneDrone board interfaced to the Nexus S.

Phoenix Trainer

The Phoenix Trainer platform was used as a proving ground for the Ardupilot autopilot systems before integration with the Robin was to take place. The Phoenix is a smaller model with a wingspan of approximately 1.4 metres. Intended primarily as a trainer plane for beginning pilots, it has excellent characteristics for evaluating the performance of an autopilot. Another aspect that was particularly suitable is that the Phoenix is powered by a

Figure 4.12: Quadcopter PhoneDrone Experimental Platform

nitro engine, and as such will experience vibrations similar (in some instances **more** severe) to that found on the Robin from the 50CC petrol engine.

This was regarded as preferable to an electric powered aircraft as it was presumed there would be less of a transition between the aircraft and the associated tuning values.

4.2 Software

4.2.1 Point Cloud Library

The Point Cloud Library (PCL) is an open source library for 3D point cloud processing. It consists of numerous state-of-the-art algorithms including filtering, feature estimation, surface reconstruction, registration, model fitting, and segmentation. PCL is available under the BSD license and is free for commercial and research use. It is a cross-platform library, having been compiled and used on Linux, Mac OS, Windows, and Android.

PCL is split into a series of smaller libraries that can be compiled separately, assisting in it's deployment on platforms with reduced computational power or storage space constraints. An overview of the structure of these libraries is shown in figure Figure 4.13

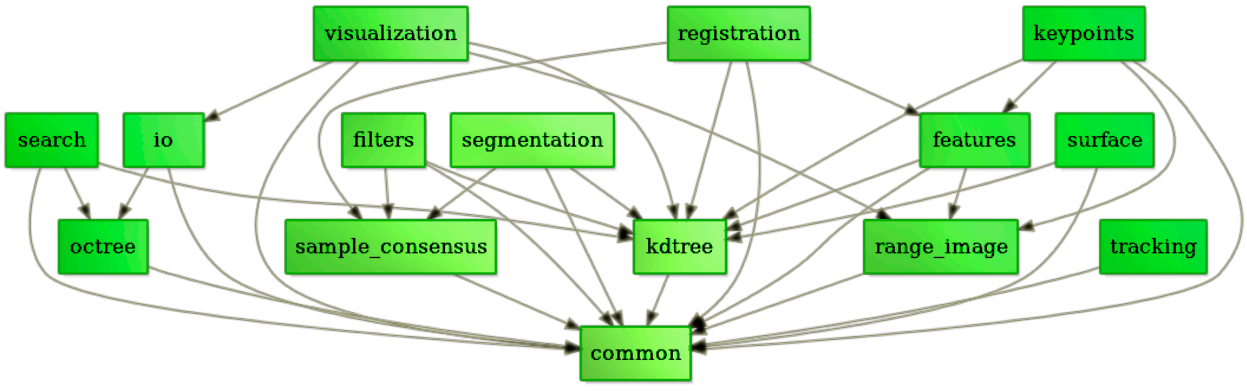


Figure 4.13: PCL Modules

Iterative Closest Point

A simple test program was written to evaluate the performance of the ICP algorithm using the Point Cloud Library (PCL). This program accepts the sequence of point clouds produced from the test dataset as input, and processes each one in turn through the ICP algorithm, attempting to find a translation from the current frame to the previous. The most relevant sections of the code are listed and explained in Listing 4.1.

```
if (downsample)
  {
    grid.setLeafSize (0.05, 0.05, 0.05);
    grid.setInputCloud (cloud_src);
    grid.filter (*src);

    grid.setInputCloud (cloud_tgt);
    grid.filter (*tgt);
  }
```

Before beginning the registration process, the point clouds are downsampled using a voxel grid filter utility in PCL. This will greatly speed up the registration process, and due to the voxel grid filter selecting the points appropriately, the impact to accuracy will be minimal.

Listing 4.1: ICP implementation in PCL

```
Eigen::Matrix4f Ti = Eigen::Matrix4f::Identity (), prev, targetToSource;
PointCloudWithNormals::Ptr reg_result = points_with_normals_src;

for (int i = 0; i < 5; ++i) {
    PCL_INFO ("Iteration Nr. %d.\n", i);
```

```
    // save cloud for visualization purpose
    points_with_normals_src = reg_result;

    // Estimate
    reg.setInputCloud (points_with_normals_src);
    reg.align (*reg_result);

        //accumulate transformation between each Iteration
    Ti = reg.getFinalTransformation () * Ti;

        //if the difference between this transformation and the previous
            one
        //is smaller than the threshold, refine the process by reducing
        //the maximal correspondence distance
    if (fabs ((reg.getLastIncrementalTransformation () - prev).sum ()) <
       reg.getTransformationEpsilon ())
      reg.setMaxCorrespondenceDistance (reg.getMaxCorrespondenceDistance ()
          - 0.001);

    prev = reg.getLastIncrementalTransformation ();

    // visualize current state
    showCloudsRight(points_with_normals_tgt, points_with_normals_src);
}
```

In this example, the ICP code is set to 5 iterations, up to 30 iterations were used during testing, however the slight increase in accuracy was deemed to be not worth the vastly increased processing time, and 5 iterations was settled on as a fair compromise.

```
points_with_normals_src = reg_result;
```

At the beginning of each iteration, the result from the previous is loaded into the source point cloud.

```
reg.setInputCloud (points_with_normals_src);
reg.align (*reg_result);
```

This point cloud is then set as the input to the ICP registration function, and the `align` method called to perform the algorithm. The transformation matrix is then retrieved and stored for later reference.

Normal Distributions Transform

The Normal Distributions Transform (NDT) algorithm has a performance advantage over ICP in that it doesn't have to perform the computationally expensive nearest-neighbour search, and therefore presents a much more attractive choice for efficient operation. Similarly to the ICP algorithm, the point clouds are downsampled using a voxel grid filter before being processed through the NDT algorithm.

Listing 4.2: NDT implementation in PCL

```
std::cout << "Aligning " << filtered_cloud->size () << " data points to "
   << target_cloud->size () << " data points." <<std::endl;
```

```
        // Set initial alignment estimate found using robot odometry.
        Eigen::Quaternion init_rotation (cur_orientation);
        Eigen::Translation3f init_translation (tx, ty, tz);
        Eigen::Matrix4f init_guess = (init_translation * init_rotation).
            matrix ();

        // Calculating required rigid transform to align the input cloud to
             the target cloud.
        pcl::PointCloud<pcl::PointXYZ>::Ptr output_cloud (new pcl::
            PointCloud<pcl::PointXYZ>);
        ndt.align (*output_cloud, init_guess);

        std::cout << "Normal Distributions Transform has converged:" << ndt
            .hasConverged () << " score: " << ndt.getFitnessScore () << std
            ::endl;
```

The test program used to evaluate ICP was modified to make use of the NDT algorithm implementation of PCL as shown in Listing 4.2, and run over the test dataset. Figure 4.14 shows the acceleration, velocity, and positional outputs of a small test dataset after being processed through the NDT algorithm.

```
  // Setting minimum transformation difference for termination condition.
    ndt.setTransformationEpsilon (0.01);
    // Setting maximum step size for More-Thuente line search.
    ndt.setStepSize (0.1);
    //Setting Resolution of NDT grid structure (VoxelGridCovariance).
    ndt.setResolution (1.0);
  // Setting max number of registration iterations.
    ndt.setMaximumIterations (35);
```

```
    Eigen::Quaternion init_rotation (cur_orientation);
    Eigen::Translation3f init_translation (tx, ty, tz);
    Eigen::Matrix4f init_guess = (init_translation * init_rotation).matrix ()
        ;
```

First, the parameters to the NDT algorithm are initialised and the estimated transformation matrix between the two frames is formed. This transformation estimate is created using data from the inertial sensors in the reference navigation processor. This initial guess as to the transformation matrix isn't strictly required, but will reduce the complexity and time required to match the two frames considerably.

```
// Setting point cloud to be aligned.
  ndt.setInputCloud (filtered_cloud);
  // Setting point cloud to be aligned to.
  ndt.setInputTarget (target_cloud);

        // Calculating required rigid transform to align the input cloud to
            the target cloud.
        pcl::PointCloud<pcl::PointXYZ>::Ptr output_cloud (new pcl::
           PointCloud<pcl::PointXYZ>);
        ndt.align (*output_cloud, init_guess);
```

Next the NDT algorithm is invoked on the two point clouds, iterating until a solution is found that meets the requirements specified by the algorithm's initialisation parameters.

```
ndt.getFinalTransformation()
```

Finally, the transformation found via the NDT algorithm can be retrieved, which can be decomposed into translation and rotation estimates suitable for use in the filtering algo-

rithms.

Performance Comparison

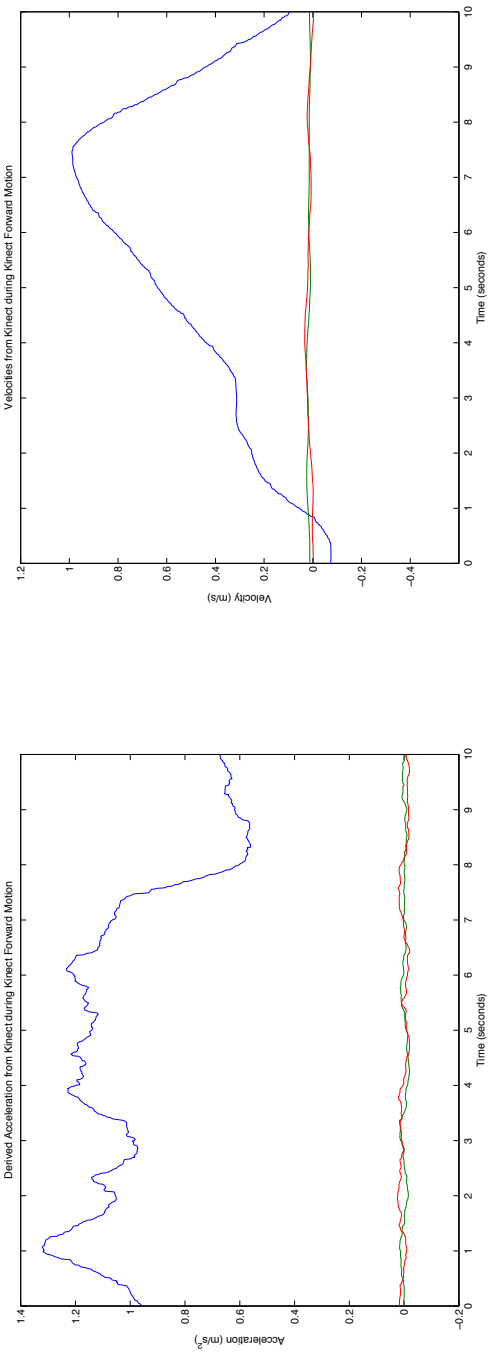
In order to profile the performance of the two test programs, they were both run over the same dataset of approximately 700 depth frames, and their execution times recorded. The code was compiled and executed on a Macbook Pro equipped with a dual core 2.66GHz Intel Core i7 processor and 8GB of RAM.

The frames were reduced to approximately 10% of their original size using a voxel grid filter in PCL before being passed to the processing algorithms. The results of this performance test are shown in Figure 4.15. The average time to process a frame was 10.02 seconds for ICP and 6.00 seconds for NDT, and the maximum processing time was 13.46 seconds for ICP and 7.08 seconds for NDT.

This illustrates the faster performance of NDT over ICP, and also brings to light the fact that NDT is much more consistent in it's processing time, as shown in 4.15(b) the times are grouped very tightly around a base value.

4.2.2 Android programming

The core of the Android operating system is based on Linux, with the primary method of application development being through Java.

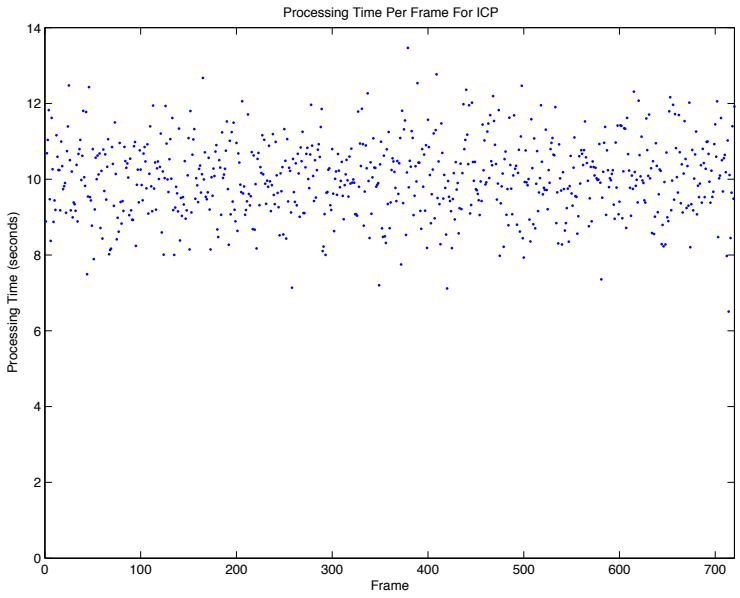


(a) Kinect Accelerations (Derived)

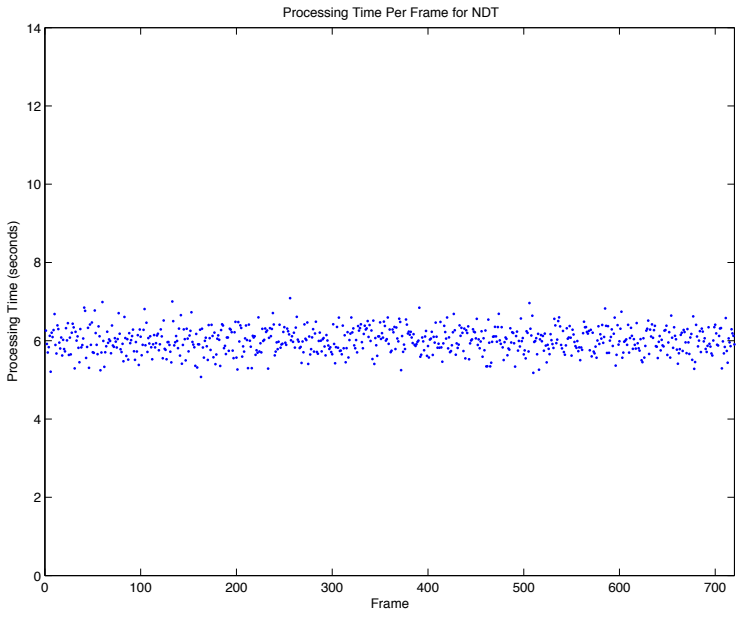
(b) Kinect Velocities

(c) Kinect Position (Integrated)

Figure 4.14: NDT Output from Kinect over 5 metre Forward Movement



(a) Processing Time Per Frame for ICP



(b) Processing Time Per Frame for NDT

Figure 4.15: Frame Processing Times for ICP and NDT

A brief overview of the structure of an Android application is presented, along with descriptions as to that components application within the proposed system.

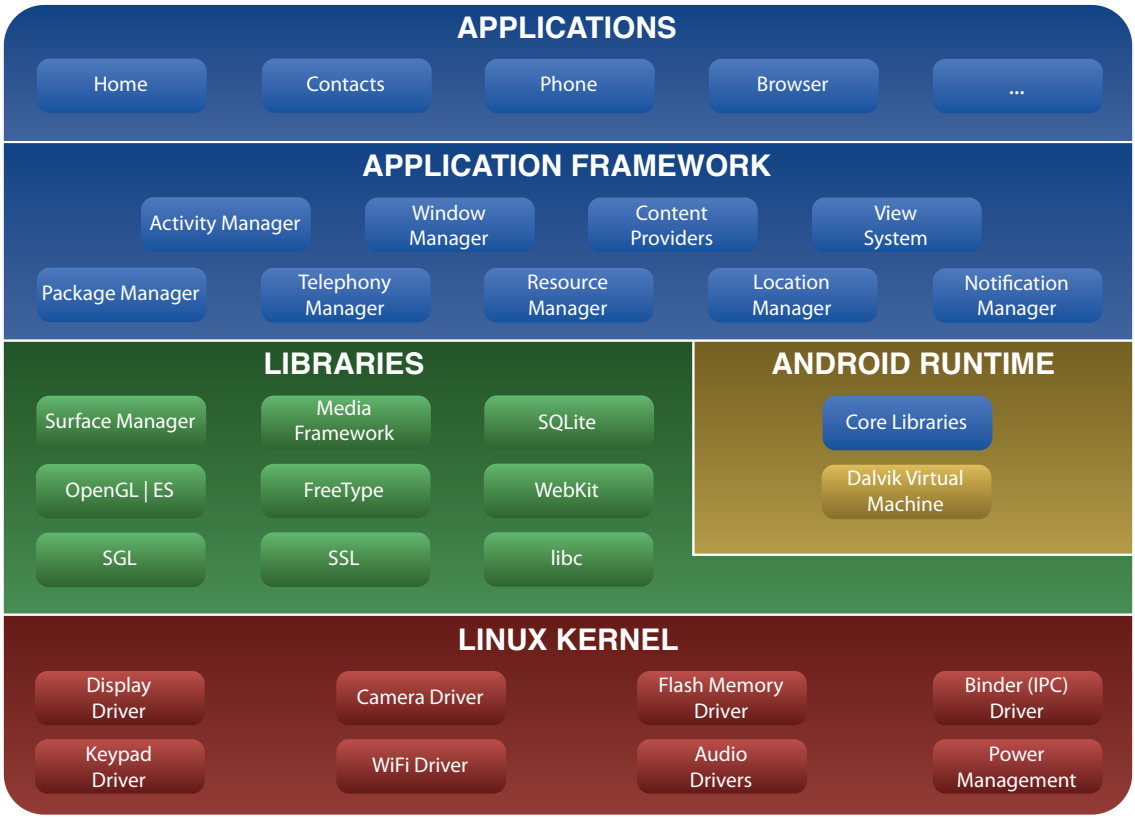


Figure 4.16: Android Operating System Architecture

Application Structure

The core of an Android application is the `Activity` class. This class is where the application logic begins, and where the user interaction events will arrive. The lifecycle of an Android `Activity` is shown in Figure 4.18.

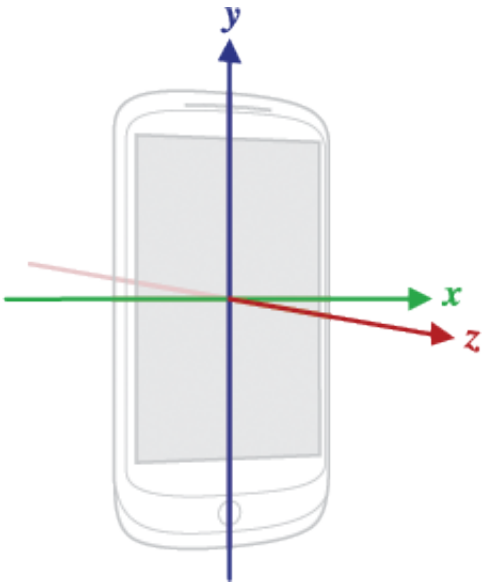


Figure 4.17: Nexus S Sensor Axes

4.2.3 Nexus S Inertial Sensors

As per the Android API, to receive measurements from the on board inertial sensors, the `SensorEventListener` interface has to be implemented. This interface specifies two public methods, `onAccuracyChanged` and `onSensorChanged` . A class that implements this interface can be registered with a `SensorManager` to receive the appropriate callbacks when sensor events are triggered. These functions are implemented and explained in section 4.2.3.

To implement an interface to the Nexus S inertial sensors and GPS in a clean, extensible, and portable manner a 'listener' interface and 'manager' class was written for both the sensors and the GPS. These implementations are explained in the following sections.

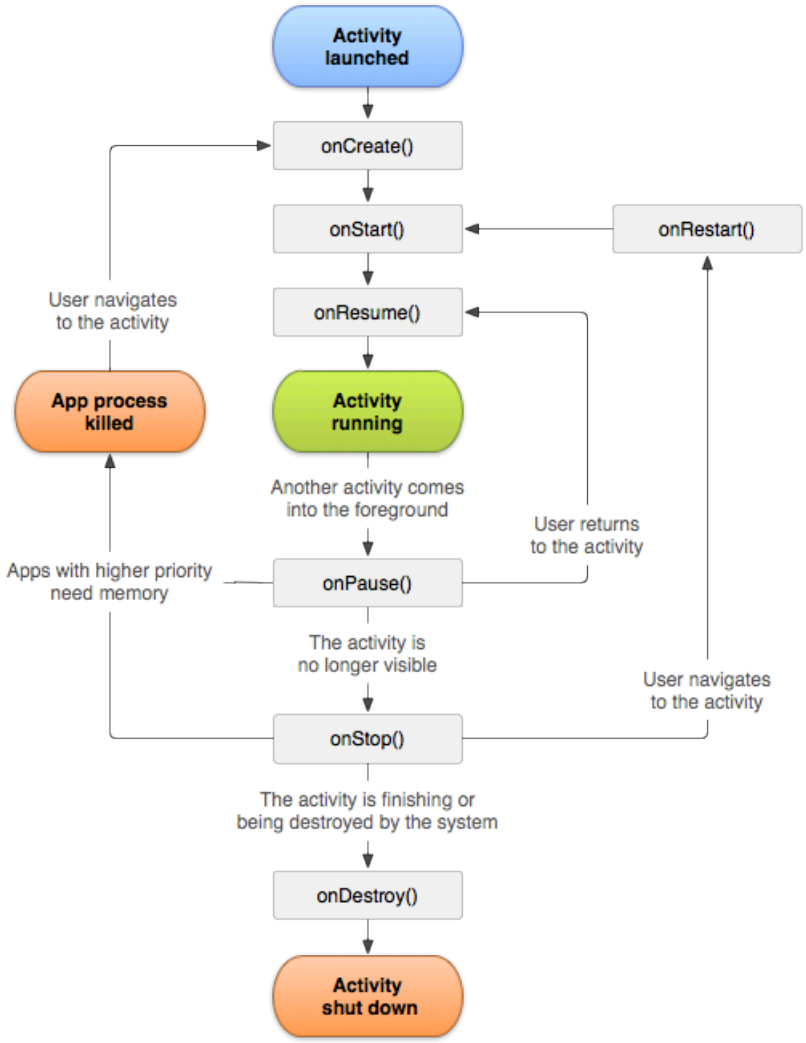


Figure 4.18: Android Activity Lifecycle (Source: http://developer.android.com)

IMUListener

The IMUListener interface defines the functions that must be implemented in order to receive the appropriate callbacks when data arrives from a particular sensor. The high-level class that requires the sensor measurements implements this interface, and registers itself with an IMUManager instance, which will trigger these callbacks when data is ready to be read from the sensors.

Listing 4.3: IMUListener Interface

```
public interface IMUListener {
    public void onAccelerometer(double[] data);
    public void onGyroscope(double[] data);
    public void onMagnetometer(double[] data);
}
```

IMUManager

The `IMUManager` class is responsible for processing the sensor updates from the operating system, and passing them to the associated listeners. To receive the sensor updates from the Android operating system, the class implements the SensorEventListener[6] interface.

Listing 4.4: IMUManagerr Constructor

```
    public IMUManager(Context c, IMUListener imuListener) {
     context = c;
```

[6] http://developer.android.com/reference/android/hardware/SensorEventListener.html

```
        listener = imuListener;

            sensorManager = (SensorManager) context.getSystemService(Context.
                SENSOR_SERVICE);

            accelerometer   = sensorManager.getDefaultSensor(Sensor.
                TYPE_ACCELEROMETER);
            gyroscope       = sensorManager.getDefaultSensor(Sensor.
                TYPE_GYROSCOPE);
            magnetometer        = sensorManager.getDefaultSensor(Sensor.
                TYPE_MAGNETIC_FIELD);
        }
```

The class constructor obtains a reference to the system sensor service through a `SensorManager` instance. Using this reference, the class can register it's interest in receiving accelerometer, gyroscope, and magnetometer signals. The `onSensorChanged` callback is triggered when data arrives from any sensor that class has a reference to.

Listing 4.5: IMUManager onSensorChanged

```
    @Override
    public void onSensorChanged(SensorEvent event) {
        //long time = System.currentTimeMillis();

        switch(event.sensor.getType()) {
            case Sensor.TYPE_ACCELEROMETER:
                for (int i=0 ; i<3 ; i++) {
                    accVals[i] = event.values[i];
                }
```

```
                listener.onAccelerometer(accVals);

                break;

            case Sensor.TYPE_GYROSCOPE:
                for (int i=0 ; i<3 ; i++) {
                    gyrVals[i] = event.values[i];
                }

                listener.onGyroscope(gyrVals);

                break;

            case Sensor.TYPE_MAGNETIC_FIELD:
                for (int i=0 ; i<3 ; i++) {
                    magVals[i] = event.values[i];
                }

                listener.onMagnetometer(magVals);

                break;

            default:
                break;
        }
    }
}
```

The `onSensorChanged` function passes a `SensorEvent` parameter which contains information as to which sensor trigger the data arrival callback, as well as the sensor data itself. In

this instance a switch case statement is used to determine which sensor has data ready, and that data is then passed to the appropriate callback function of the registered `IMUListener` .

GPSListener

Similarly to the `IMUListener` interface, the `GPSListener` simply outlines which functions a class must implement in order to receive callbacks related to the GPS module of the phone.

Listing 4.6: GPSListener Interface

```
public interface GPSListener {
    public void onPosition(double[] data, String provider);

    public void onNmea(long timestamp, String nmea);
}
```

GPSManager

The `GPSManager` class implements both `LocationListener` and `NmeaListener` interfaces. These interfaces are used to retrieve decoded GPS information, and raw NMEA strings from the device's GPS module.

Listing 4.7: GPSManager Interface

```
    public GPSManager(Context c, GPSListener gpsListener) {
```

```
        context = c;
        listener = gpsListener;

        locationManager = (LocationManager) context.getSystemService(
            Context.LOCATION_SERVICE);
    }
```

The `GPSManager` constructor simply keeps a reference to the GPSListener that spawned it, and obtains a reference to the system's location service through an instance of a `LocationManager` .

Listing 4.8: GPSManager start/stop Functions

```
    public void stop() {
        registered = false;
        locationManager.removeUpdates(this);
        locationManager.removeNmeaListener(this);
    }

    public void start() {
        locationManager.addNmeaListener(this);
        locationManager.requestLocationUpdates(LocationManager.GPS_PROVIDER
            , 0, 0, this);
    }
```

The `GPSManager` class implements two functions, `start` and `stop` which register and unregister the class to receive GPS and NMEA updates from the `LocationManager` service respectively.

When the GPS has new positional information, or a new NMEA string, the appropriate callback functions are triggered by the `LocationManager` service. The `onLocationChanged` function is passed a `Location` object as a parameter, which contains the relevant information (e.g. Latitude, Longitude, etc.), and similarly for `onNMEAReceived` with NMEA information.

Listing 4.9: GPSManager onLocationChange and onNmeaReceived

```
@Override
public void onLocationChanged(Location location) {
    // TODO Auto-generated method stub
    latitude = location.getLatitude();
    longitude = location.getLongitude();
    altitude = location.getAltitude();
    bearing = location.getBearing();
    speed = location.getSpeed();
    accuracy = location.getAccuracy();
    gpstime = location.getTime();
    provider = location.getProvider();

    listener.onPosition(new double[] {latitude, longitude, altitude,
        bearing, speed, accuracy, gpstime}, provider);
}

@Override
public void onNmeaReceived(long timestamp, String nmea) {
    // TODO Auto-generated method stub
    //if(listener != null) {
        listener.onNmea(timestamp, nmea);
    //}
}
```

4.2.4 Logging Application

In order to record the data from the Nexus S sensors a logging application was written which writes log files for each sensor, and the GPS, to the internal flash memory. This recorded data can then be retrieved over a USB connection for post-processing. A screenshot of the application running on the Nexus S is shown in Figure 4.19.

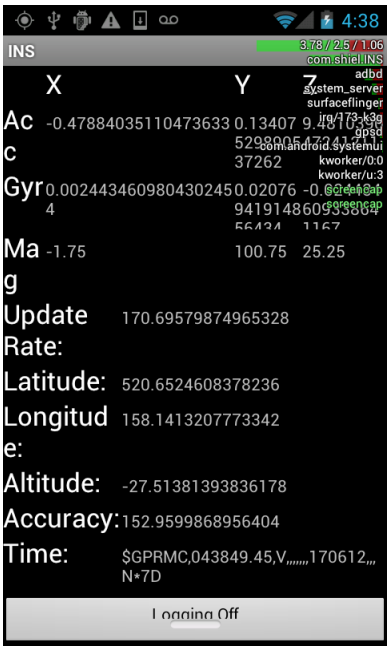


Figure 4.19: Android Logging Application

INSActivity

The INSActivity is the core activity of the logging application. As per the `Activity` lifecycle shown in Figure 4.18 the `onCreate` function is called as the activity is created. In this case, the `INSActivity` creates new instances of the `IMUManager` and `GPSManager` , passing along a reference to itself so as to receive the callback functions, as well as a reference to the

local application context, to allow these manager classes to obtain references to the system services (i.e. sensor and GPS access).

Listing 4.10: INSActivity onCreate

```
@Override
public void onCreate(Bundle savedInstanceState) {
    Log.i(tag, "onCreate");
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    imu = new IMUManager(this, this);
    gps = new GPSManager(this, this);

    accrate = gyrrate = magrate = orirate = gravrate = 0;
    latitude = longitude = altitude = 0;
}
```

Essentially, the `INSActivity` sits in an idle loop while waiting for new sensor data to arrive via the registered callback functions. When data does arrive, the respective function is called, and the data written to the appropriate log file. Shown in Listing 4.11 is an example of one of these callback functions. In this case it is the GPS data that is being received. The data is broken into the `INSActivity` member variables, and then written to the log file.

Listing 4.11: INSActivity onPosition callback function

```
@Override
    public void onPosition(double[] data, String _provider) {
        // TODO Auto-generated method stub
        //latitude, longitude, altitude, bearing, speed, accuracy, gpstime
```

```
        latitude = data[0];
        longitude = data[1];
        altitude = data[2];
        bearing = data[3];
        speed = data[4];
        accuracy = data[5];
        gpstime = data[6];
        provider = _provider;

        if(logging) {
            gpsLog.write(System.currentTimeMillis() + ", " + latitude + ",
                " + longitude + ", " + altitude + ", " + bearing + ", " +
                speed + ", " + accuracy + ", " +
                    gpstime + ", " + provider + "\n");
        }
    }
```

`INSActivity` refreshes the data on the screen via the `updateData` function (shown in Listing 4.13) each time a new accelerometer measurement is received. In addition, as the inertial sensors share a common log file, the accelerometer callback is also used to write all inertial sensor data to a file at once. The accelerometer callback was chosen for these tasks as out of the three inertial sensors, it has the most consistent update rate (50Hz).

Listing 4.12: INSActivity onAccelerometer callback function

```
@Override
    public void onAccelerometer(double[] data) {
        accVals = data;
```

```
        long t = System.nanoTime();
        accrate = 1000000000.0 / (t - at);
        at = t;

        updateData();

        if(logging) {
            imuLog.write(System.currentTimeMillis() + ", " + accVals[0] + "
                , " + accVals[1] + ", " + accVals[2] + ", " +
                    gyrVals[0] + ", " + gyrVals[1] + ", " + gyrVals[2] + ",
                        " +
                    magVals[0] + ", " + magVals[1] + ", " + magVals[2] + ",
                        " +
                    oriVals[0] + ", " + oriVals[1] + ", " + oriVals[2] + "\
                        n");
        }
    }
```

Listing 4.13: INSActivity updateData function

```
private void updateData() {

        ...

        ((TextView) this.findViewById(R.id.ax)).setText(String.format("%.4f
            ", oriVals[0]));
        ((TextView) this.findViewById(R.id.ay)).setText(String.format("%.4f
            ", oriVals[1]));
        ((TextView) this.findViewById(R.id.az)).setText(String.format("%.4f
            ", oriVals[2]));

        ...
```

```
    }
```

4.3 Performance Tests

4.3.1 Nexus S Validation

Before beginning the integration of the Kinect sensor, the existing system must be tested to ensure it's correct stand-alone operation. To accomplish this, the system was subjected to a series of tests and compared to a ground-truth estimate provided by the XSens MTi-G commercial INS.

The tests the system was subjected to include:

- Stationary bench test to ensure stability, and validate drift compensation
- Bench test involving rotation through a series of simple movements (e.g. roll 90 degrees)
- Outdoor positional test, to validate operation of the device's GPS modules
- Real world flight test where the system was strapped into the Robin 2160 airframe while it was flown in a simple search pattern

Stationary Bench Test

For this test, the Nexus S was positioned on a stationary desk beside the XSens MTi-G. This test was conducted indoors, and therefore is not GPS compensated in any way. The idea of this test is to validate the operation of the inertial sensors of each device (gyroscope,

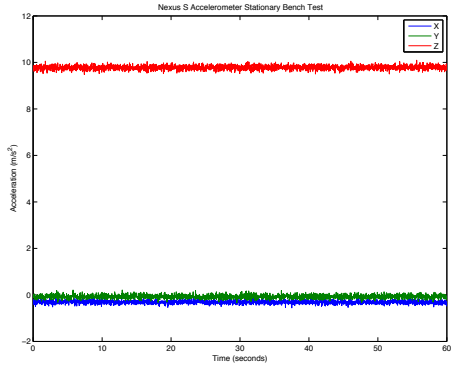
accelerometer, and magnetometer) and gather samples so as to determine their baseline noise levels and drift rates.

The devices were run for a period of 10 minutes, allowing them to reach operating temperature and ensuring that temperature was not a factor in the observed sensor drift. A selected one minute portion of these tests is shown below in Figure 4.20 and Figure 4.21.

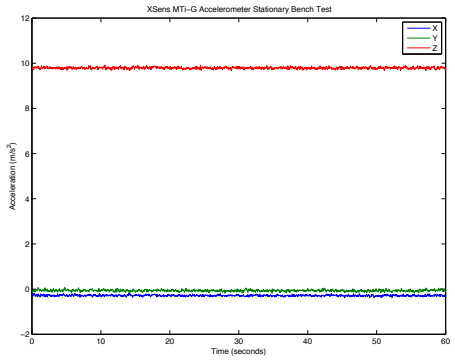
The figures show that the Nexus S inertial sensors have a slightly higher level of noise than that observed in the MTi-G. In addition, the smaller gradient over time indicates that they are slightly more robust to temperature changes. One other observation of note is that the magnetometer in the Nexus S is much more immune to interference than that in the MTi-G, this can be put to the fact that it operates in an extremely noisy environment (multiple radio frequencies for the phone) and as such has additional shielding and compensation.

Rotational Bench Test

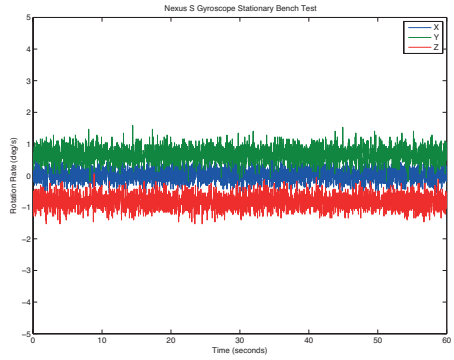
This test was primarily designed to evaluate the performance of the gyroscope within the Nexus S. The device was rotated through a roll motion of 90 degrees over a period of 5 seconds, held at 90 degrees for 2 seconds and then returned to a flat position. This was done by hand, and as such the results have a level of jittering introduced through the manual rotation, but are still sufficient to determine the sensor capabilities.



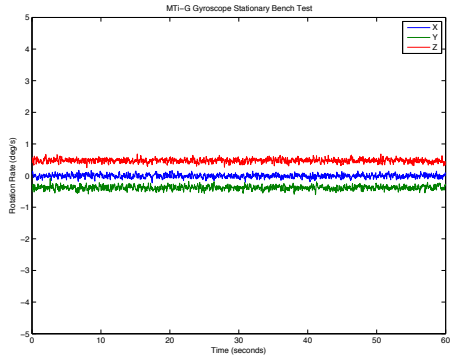
(a) Nexus S Accelerometers



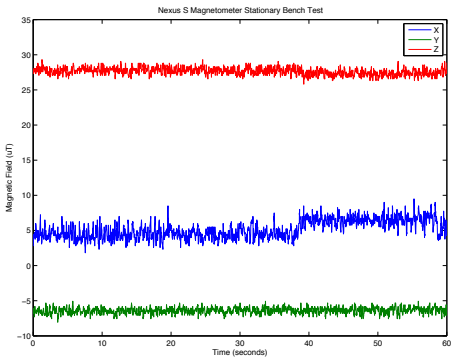
(b) XSens MTi-G Accelerometers



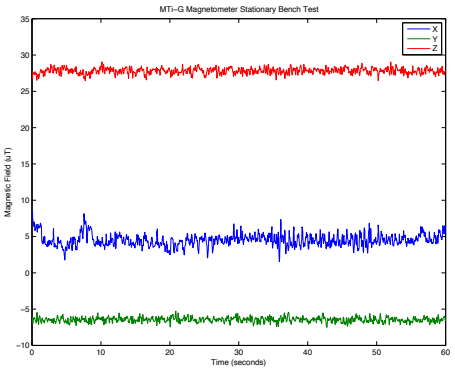
(c) Nexus S Gyroscopes



(d) XSens MTi-G Gyroscopes

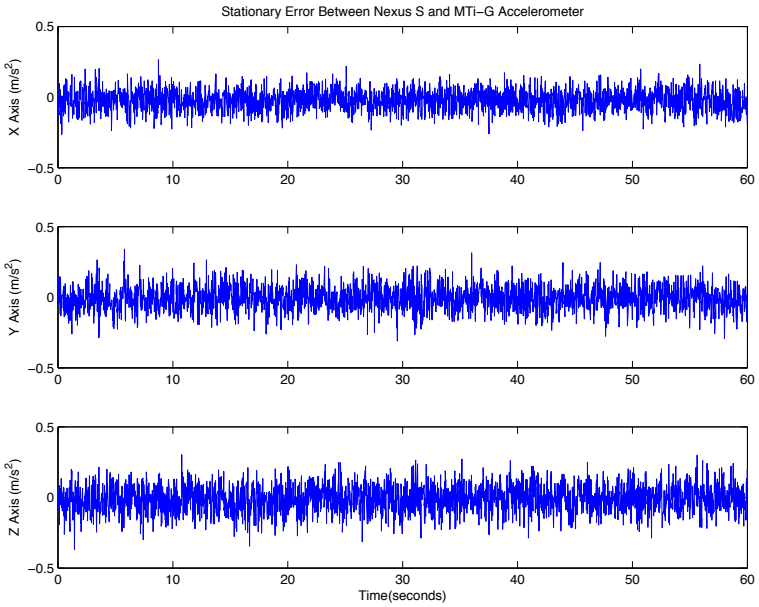


(e) Nexus S Magnetometers

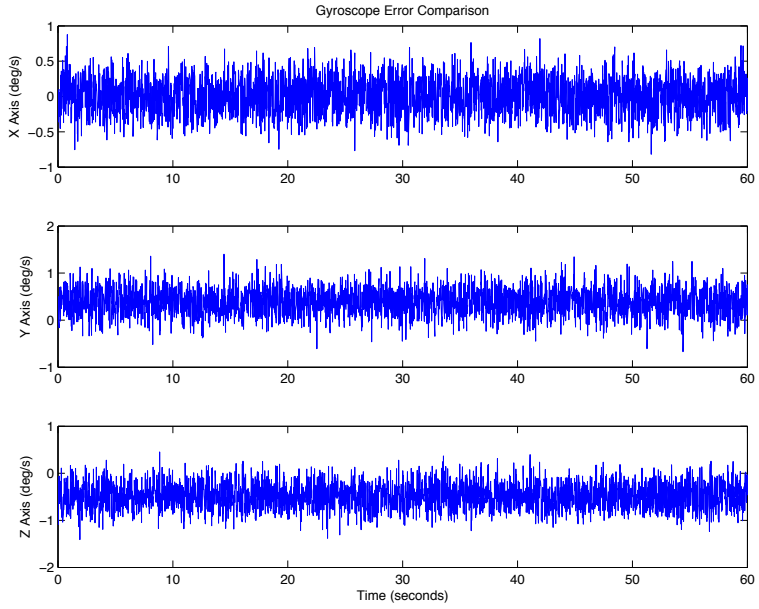


(f) XSens MTi-G Magnetometers

Figure 4.20: Stationary Inertial Sensor Bench Test

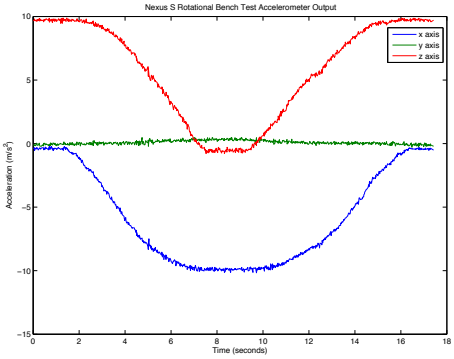


(a) Accelerometer Comparison

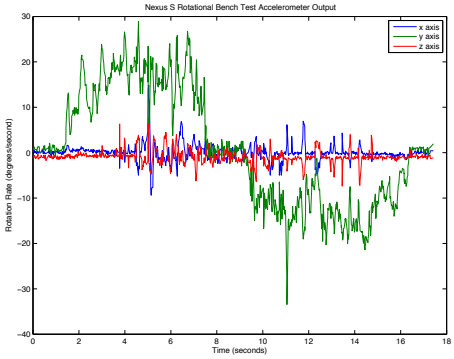


(b) Gyroscope Comparison

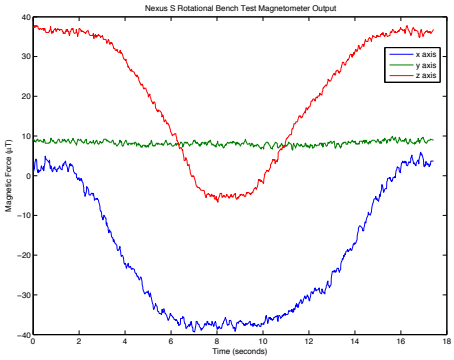
Figure 4.21: Sensor Output Comparison



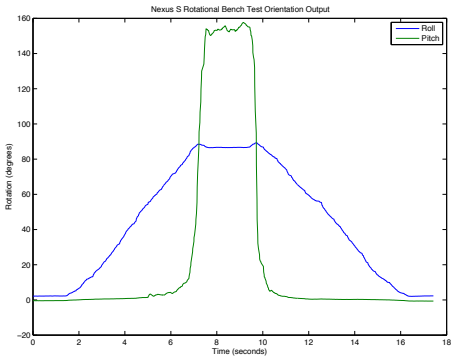
(a) Nexus S Accelerometers during Rotational Benchtest



(b) Nexus S Gyroscopes during Rotational Benchtest



(c) Nexus S Magnetometers during Rotational Benchtest

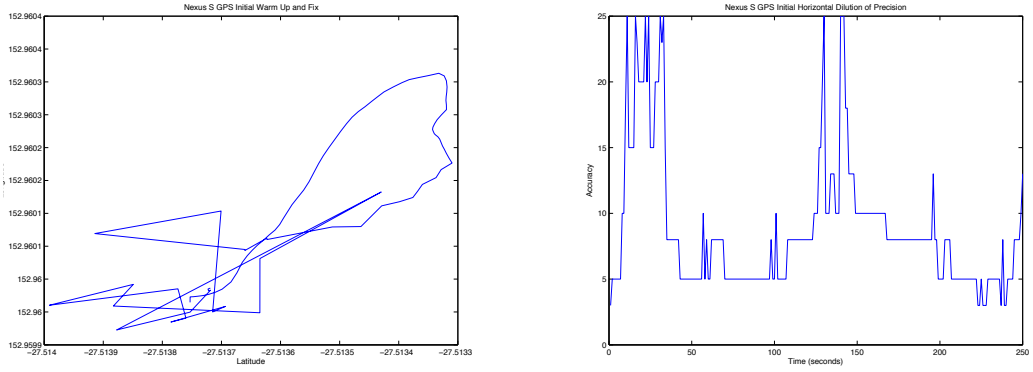


(d) Nexus S Estimated Orientation during Rotational Benchtest

Figure 4.22: Nexus S Rotational Benchtest Outputs

Outdoor Positional Test

To determine the level of accuracy of the Nexus S GPS module, an outdoor test was performed comparing it's raw output against that of the MTi-G. The devices were positioned outdoors (within 10cm of each other to allow for a simple comparison) and initially turned off. In addition, they were both forced to do a cold start so as to determine the worst-case time to first fix. Shown below is the positional information of each module, and the reported horizontal dilution of precision for the Nexus S, over a 5 minute period.



(a) Nexus S GPS Position Initial Fix (b) Nexus S GPS HDOP During Warmup

Figure 4.23: Nexus S GPS Warmup and Initial Fix

The MTi-G uses a uBlox 5 series GPS module, which performs to a higher level than the Nexus S module. That being said, the GPS in the Nexus S is able to get a 3D fix in a short amount of time and it's positional accuracy is of a sufficient standard to be utilised for navigational purposes. In addition, the Nexus S features an internal antenna port for an auxiliary Bluetooth/GPS antenna which greatly boosts the SNR compared to what the built in antenna can achieve.

GPS Augmentation The Nexus S has several methods that it can utilise to augment the positional information from the GPS signal.

- Assisted GPS (A-GPS)
- WiFi Positioning
- Phone Tower Triangulation

Assisted GPS (A-GPS) takes advantage of the network resources the Nexus S is connected to via the radio modem, for example: downloading orbital data or almanac for the GPS satellites from a server rather than the satellites themselves, enabling the receiver to achieve a lock quicker. An alternate method exists where the receiver will capture a snapshot of the local GPS signal, which is then sent to a remote assistance server along with the associated log time. This remote server has vast computational power and a reliable satellite signal and therefore can analyse fragmented signals that are relayed to it.

WiFi Positioning takes a scan of the local WiFi signals (more specifically: Access Point SSIDs) and similarly to A-GPS transmits this data to a remote processing server. This remote server has an enormous database of SSIDs and approximate surveyed positions. Using the signal strengths of each SSID send, the server can determine an approximate location for the receiver based on it's known SSID locations.

Phone Tower Triangulation uses the highly accurate surveyed GPS positions of mobile phone towers, combined with the respective signal strengths to each tower, to estimate the current position of the phone. Each tower regularly broadcasts it's relevant information, including GPS position, within the normal network traffic to mobile phones.

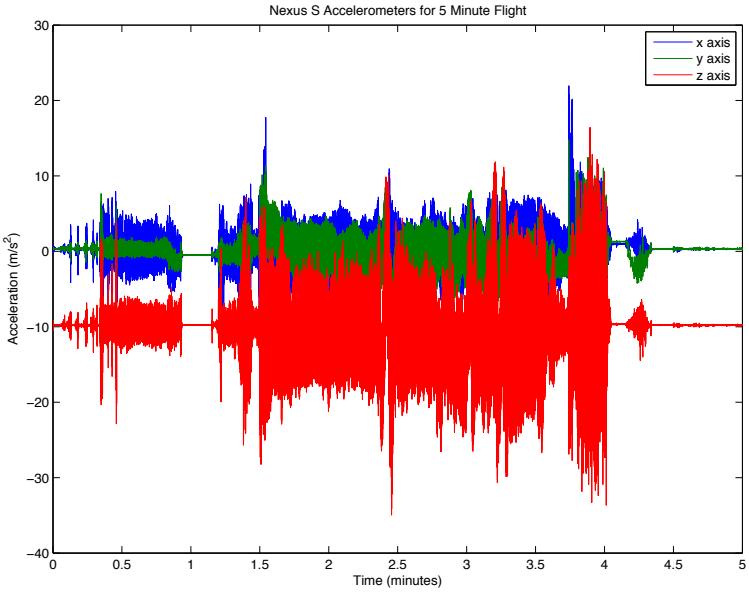
Flight Test

The final performance test of the Nexus S was conducted on board the Robin 2160 aircraft. This test validates the operation of all of the sensors of the Nexus S under real-world flight conditions. The Nexus S was attached to the internal wing spar of the Robin, approximately over the centre of gravity of the aircraft, and a valid GPS fix was ensured before take off. The aircraft was flown in a simple search pattern consisting of repeated circles for approximately 5 minutes.

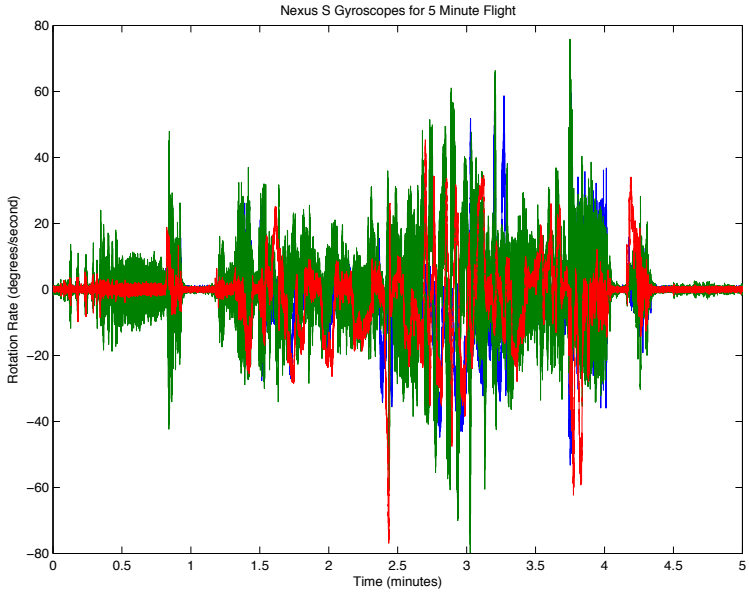
The flight pattern of the aircraft is clearly visible in the positional output shown in Figure 4.26, and the repeated circling is reflected in the wrap-around seen in the yaw plot in 4.25(c). Visible in Figure 4.24 is the large amount of vibrational noise imparted onto the Nexus S from the single cylinder engine powering the Robin. A potential future remedy to reduce these vibration levels is a specifically designed vibration resistant mount, rather than the rigidly fixed zip ties and foam block combination used for this evaluation. Figure 4.25 shows that the sensors in the Nexus S are capable of performing to a level sufficient to provide orientation estimates for an aircraft in flight.

4.3.2 Kinect Validation

In practice, even with a decimated point cloud and with the Kinect set to the lowest possible update rate, it was determined that the Nexus S wasn't able to process the frames fast enough to make real time operation feasible. Further investigation with debugging and profiling tools determined that the cause of this was not entirely due pure processing power limitations, as

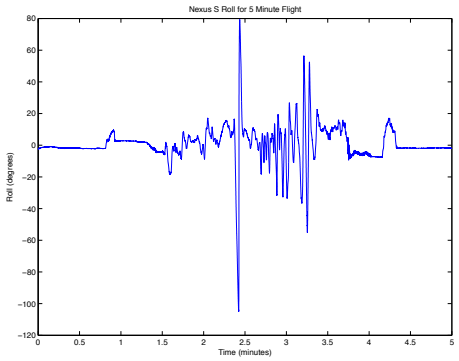


(a) Nexus S Accelerometers during 5 Minute Flight Test

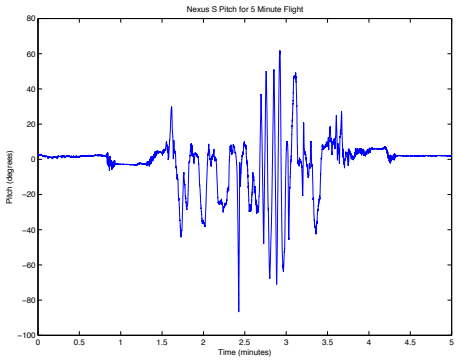


(b) Nexus S Gyroscopes during 5 Minute Flight Test

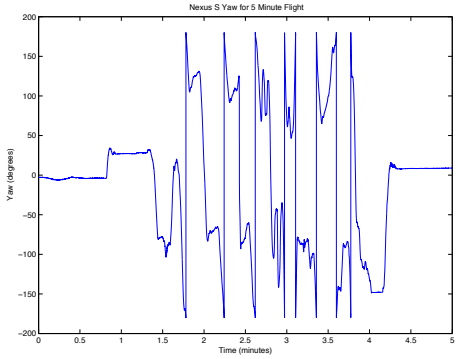
Figure 4.24: Nexus S Inertial Sensor Outputs during 5 Minute Flight Test



(a) Estimated Roll during 5 Minute Flight Test



(b) Estimated Pitch during 5 Minute Flight Test



(c) Estimated Yaw during 5 Minute Flight Test

Figure 4.25: Estimated Orientation during 5 Minute Flight Test

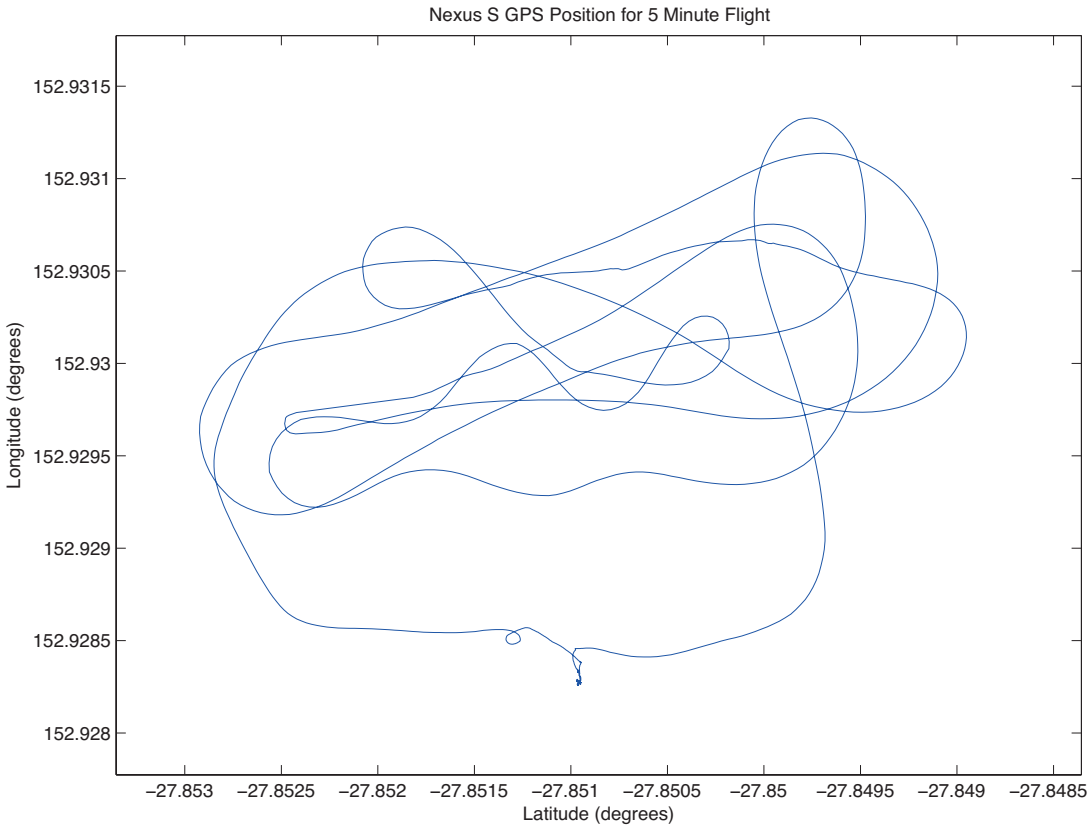


Figure 4.26: Nexus S GPS Position during 5 Minute Flight Test

PCL has an Android optimised port which takes advantage of the ARM processors SIMD instruction set, but rather limited by the sheer amount of processing required to receive the frames from the Kinect.

Further work will need to be performed to achieve a satisfactory level of performance, but the primary areas of focus relevant to this goal are the USB driver and associated processing, as well as the Kinect specific USB interfacing code. A potential solution is to limit the Kinect update rate at the hardware level, alleviating the load presented to the processor but further research into the specific operating parameters of the USB hardware is required.

For the remainder of the work, the Kinect frames will be logged separately from the Nexus S sensors, and post-processed on a desktop computer with the appropriate sensor inputs to produce the required data.

Chapter 5

System Verification

To verify the correct operation of the overall system, a final test dataset was compiled wherein the Kinect and INS undergo a transition from an outdoor environment to an indoor one. This test will test all aspects of the system, specifically the ability to transition between using the GPS positional measurements to the positional estimates produced by integrating velocity readings from the Kinect without a large impact on system accuracy.

This section goes through each step in the process of the system, illustrating how and why the decisions are being made, and showing the relevant data.

5.1 Captured Data

The initial sets of data are shown in the following sections. This data was captured using the Nexus S for inertial and GPS data, and a laptop to save the Kinect frames for offline processing.

5.1.1 RGB Images

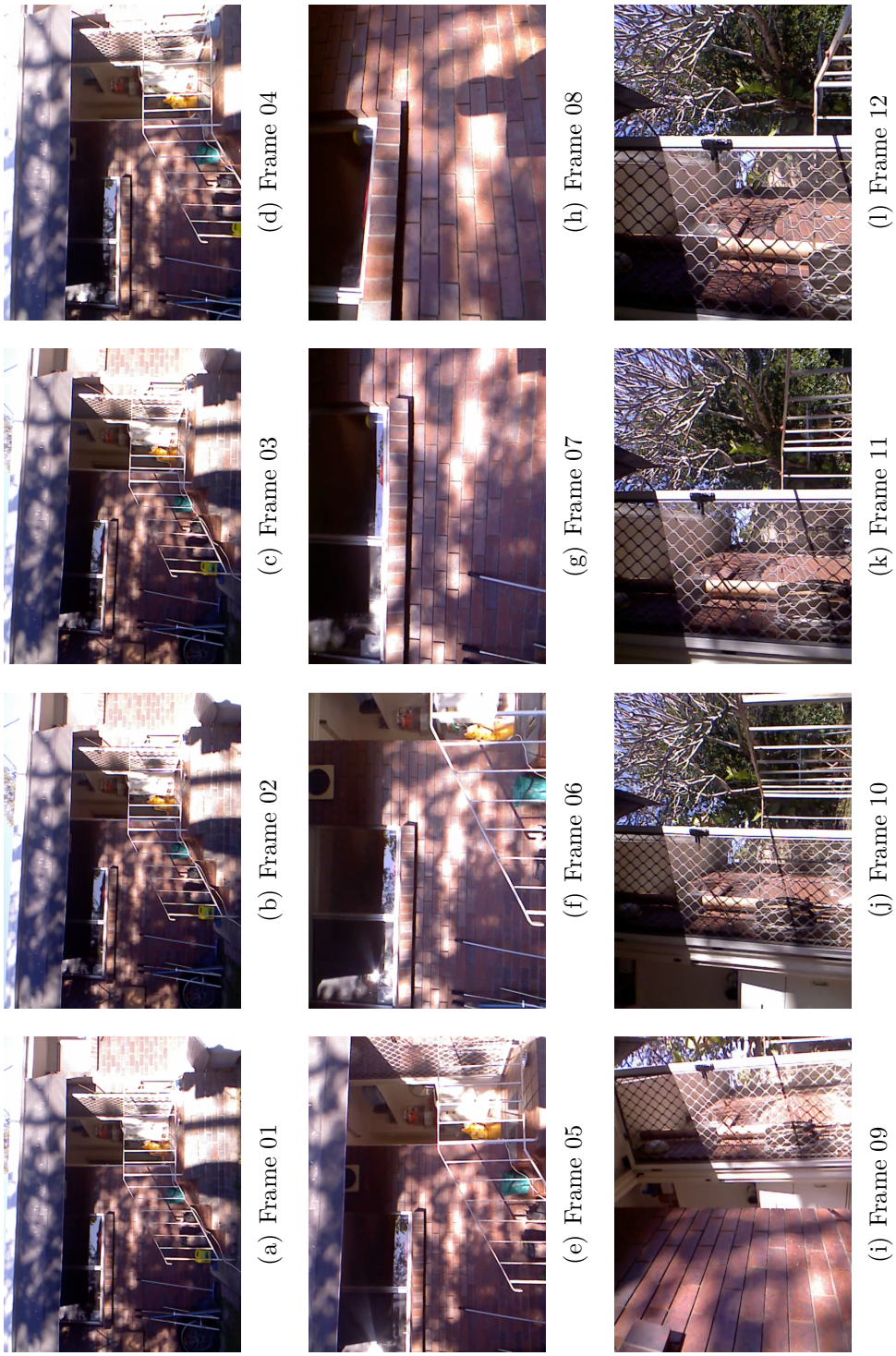
The Kinect RGB frames captured during the dataset are shown in Figure 5.1 and Figure 5.2.

5.1.2 RGB Histograms

The histograms are calculated for each frame captured during the dataset, and are shown in Figure 5.4 and Figure 5.4. The x axis values on each histogram range between 0.0 and 1.0, corresponding to the pixel intensity of the red (R) channel.

5.1.3 Depth Images

The Kinect depth frames captured during the dataset are shown in Figure 5.5 and Figure 5.6. While they are difficult to interpret by eye, these images illustrate the gradual increase in depth as the Kinect approaches the house.



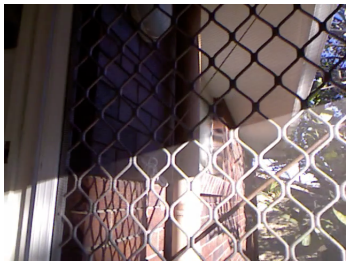
(a) Frame 01 (b) Frame 02 (c) Frame 03 (d) Frame 04

(e) Frame 05 (f) Frame 06 (g) Frame 07 (h) Frame 08

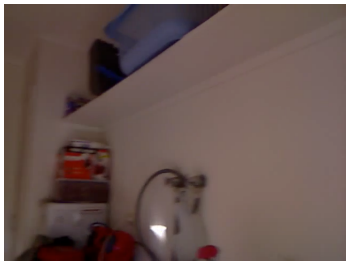
(i) Frame 09 (j) Frame 10 (k) Frame 11 (l) Frame 12

Figure 5.1: Example Kinect RGB Frames 01-12

(a) Frame 13

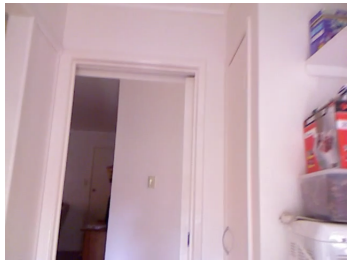


(b) Frame 14



(c) Frame 15

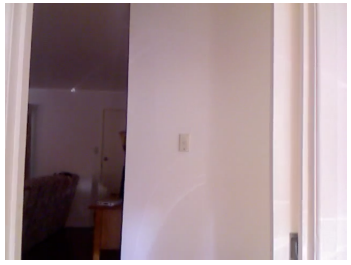
(d) Frame 16



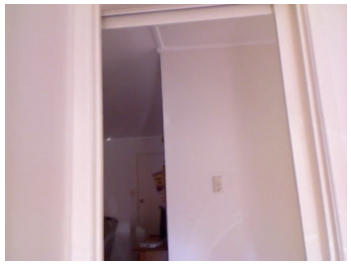
(e) Frame 17

(f) Frame 18

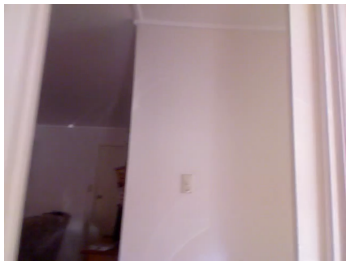
(g) Frame 19



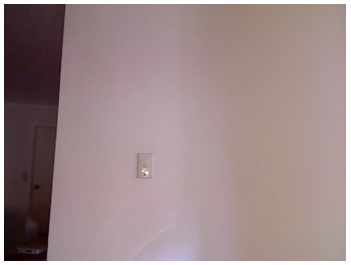
(h) Frame 20



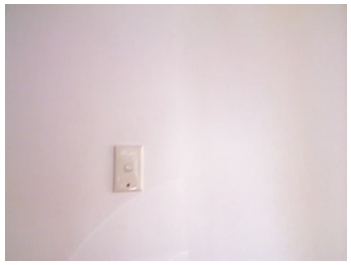
(i) Frame 21



(j) Frame 22

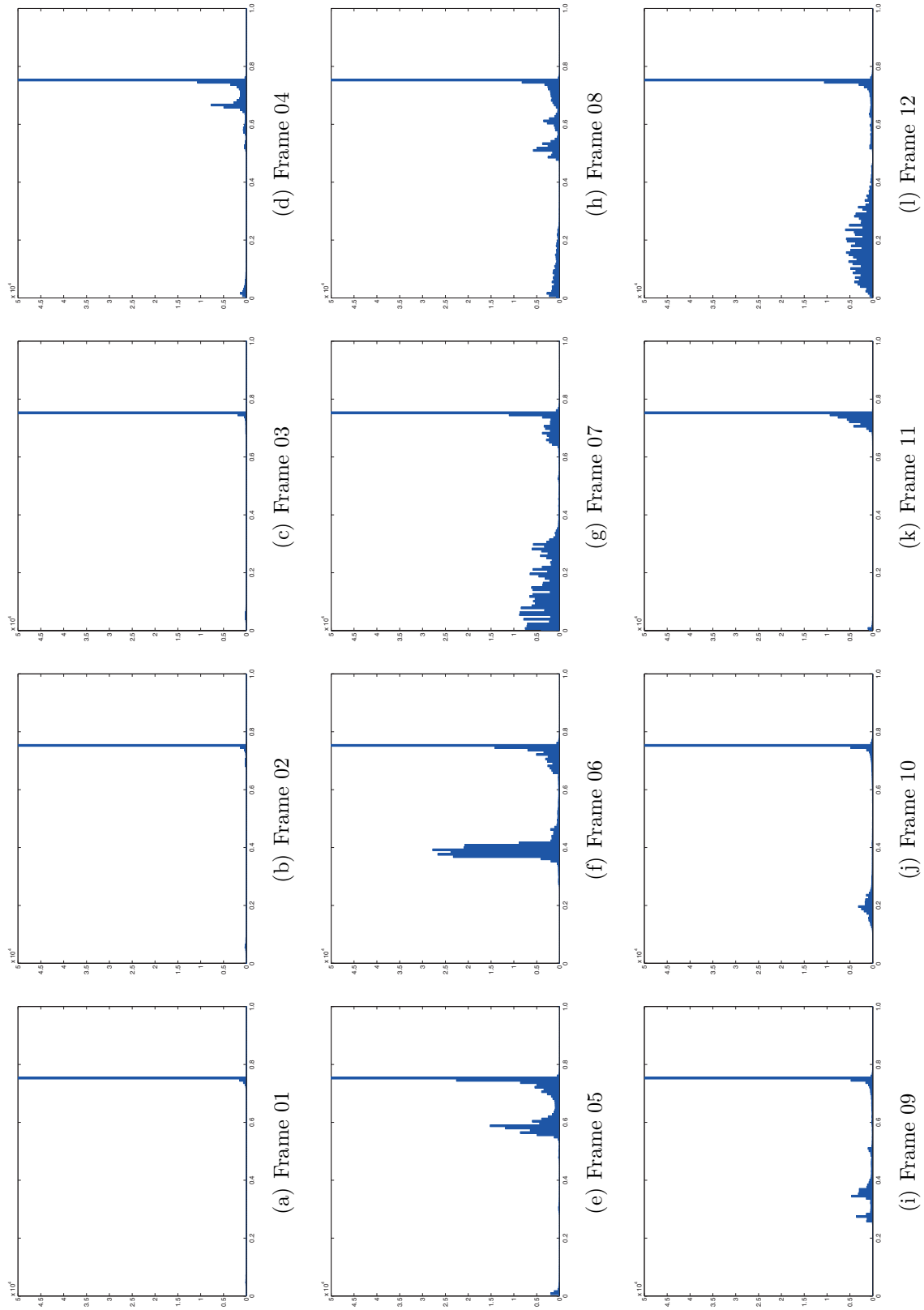


(k) Frame 23



(l) Frame 24

Figure 5.2: Example Kinect RGB Frames 13-24

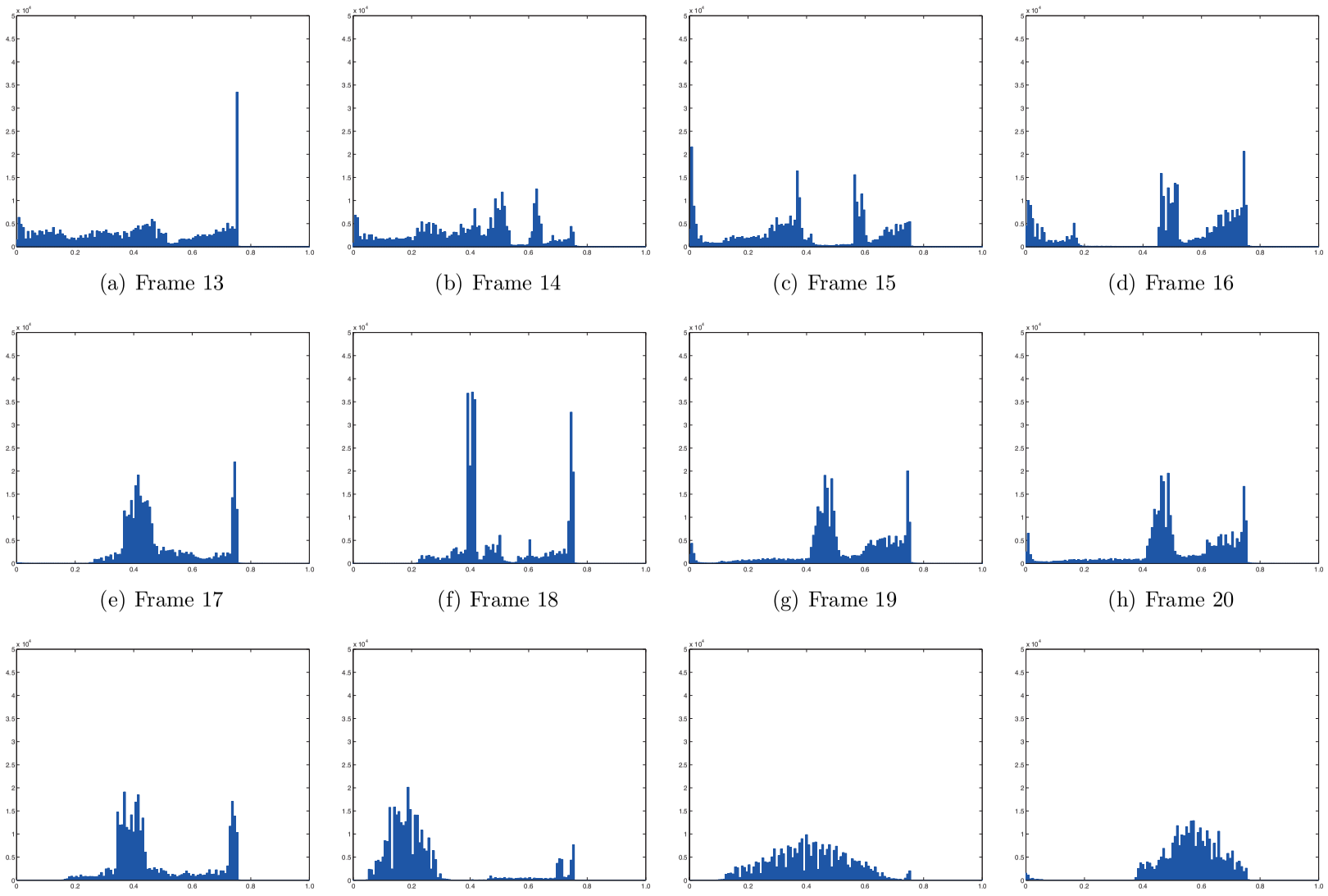


(a) Frame 01 (b) Frame 02 (c) Frame 03 (d) Frame 04

(e) Frame 05 (f) Frame 06 (g) Frame 07 (h) Frame 08

(i) Frame 09 (j) Frame 10 (k) Frame 11 (l) Frame 12

Figure 5.3: Histograms for Kinect RGB Frames 01-12

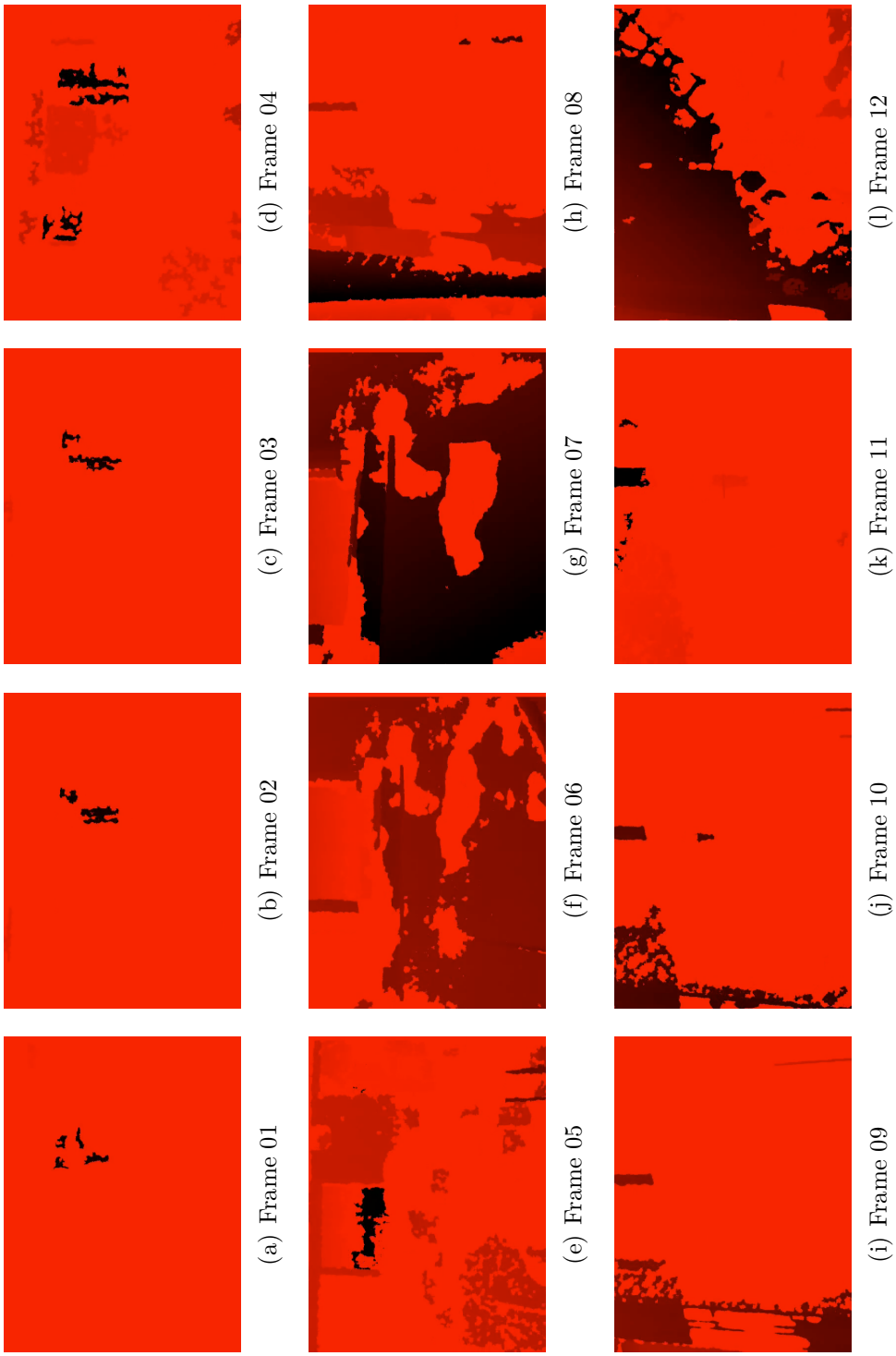


(a) Frame 13 (b) Frame 14 (c) Frame 15 (d) Frame 16

(e) Frame 17 (f) Frame 18 (g) Frame 19 (h) Frame 20

(i) Frame 21 (j) Frame 22 (k) Frame 23 (l) Frame 24

Figure 5.4: Histograms for Kinect RGB Frames 13-24



(a) Frame 01 (b) Frame 02 (c) Frame 03 (d) Frame 04
(e) Frame 05 (f) Frame 06 (g) Frame 07 (h) Frame 08
(i) Frame 09 (j) Frame 10 (k) Frame 11 (l) Frame 12

Figure 5.5: Example Kinect Depth Frames 01-12

(a) Frame 13 (b) Frame 14 (c) Frame 15 (d) Frame 16

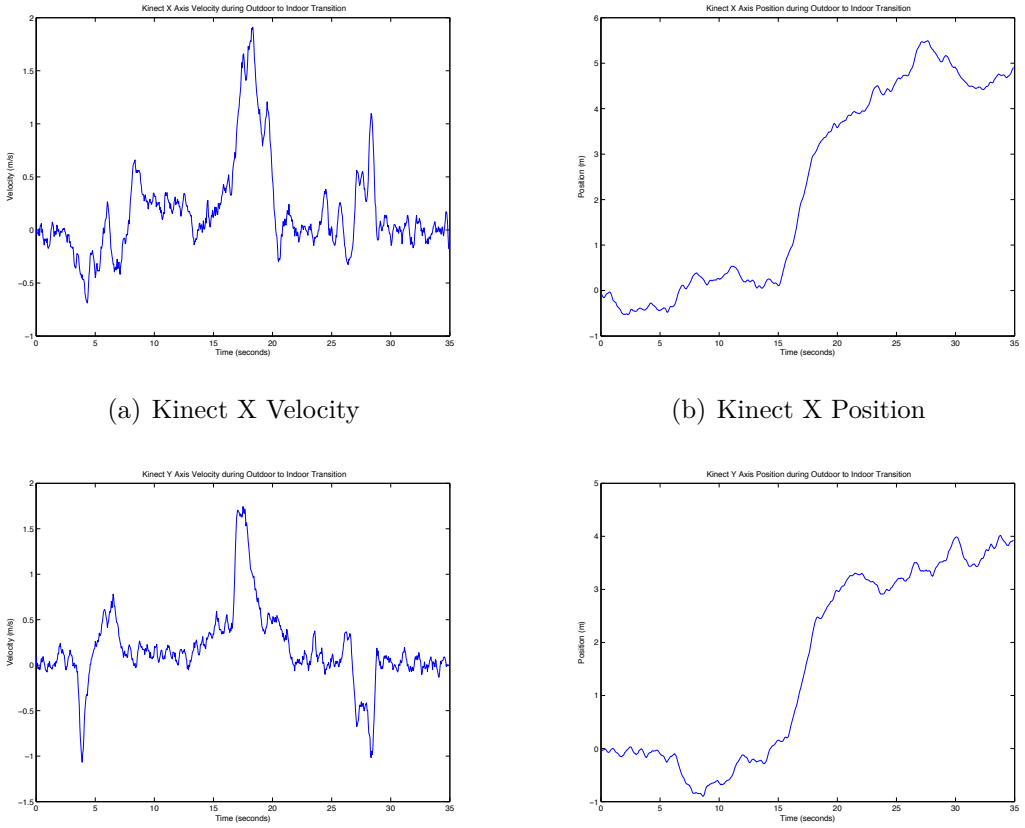
(e) Frame 17 (f) Frame 18 (g) Frame 19 (h) Frame 20

(i) Frame 21 (j) Frame 22 (k) Frame 23 (l) Frame 24

Figure 5.6: Example Kinect Depth Frames 13-24

5.1.4 Kinect Velocity

The velocity and position estimates from the Kinect were calculated using the NDT algorithm, the outputs of which are shown in Figure 5.12.



(a) Kinect X Velocity

(b) Kinect X Position

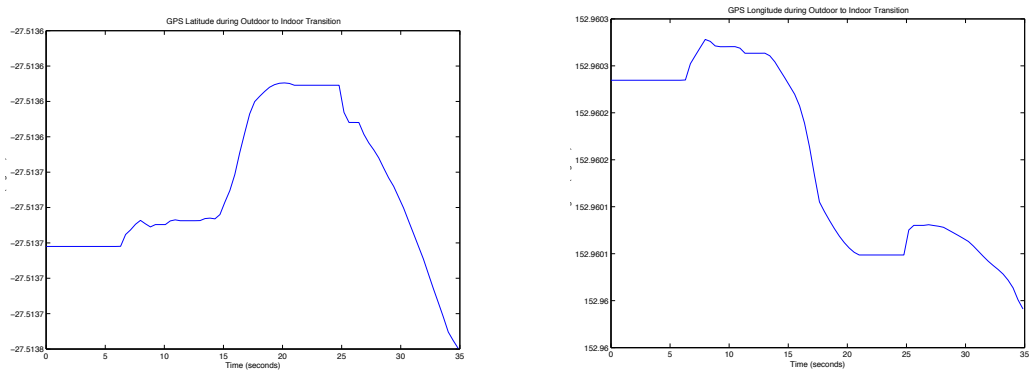
(c) Kinect Y Velocity

(d) Kinect Y Position

Figure 5.7: Kinect Velocity and Position during Outdoor to Indoor Transition

5.1.5 GPS Position

The known movement pattern taken when obtaining the GPS logs is shown as an overlay on the map in Figure 5.10 as a point of reference.



(a) GPS Latitude during Outdoor to Indoor Transition

(b) GPS Longitude during Outdoor to Indoor Transition

Figure 5.8: Kinect Velocity and Position during Outdoor to Indoor Transition

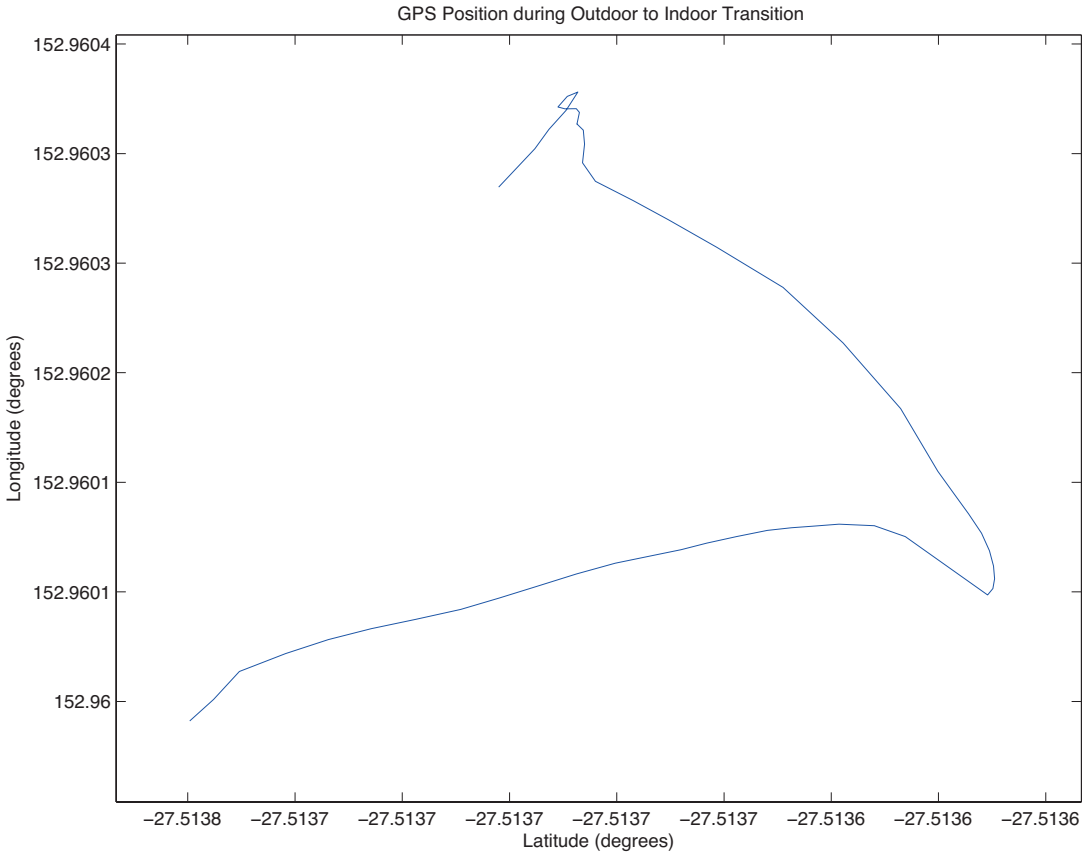


Figure 5.9: GPS Position during Outdoor to Indoor Transition

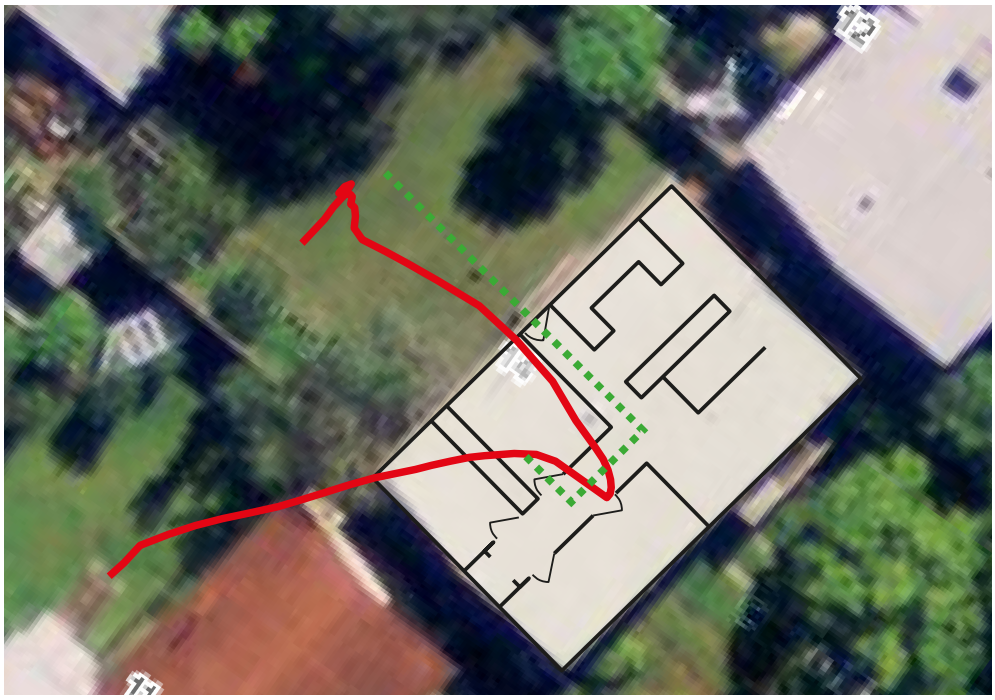


Figure 5.10: GPS Position during Outdoor to Indoor Transition

5.2 Output

5.12(a) shows the calculated confidence for the Kinect during the dataset. Clearly visible in this graph is the transition from an out of range depth to one that is in range at approximately frame 300. The spike that occurs between frames 100 and 200 is the detection of a stable wall surface, which was lost during a rotational movement which concluded at frame 300.

5.2.1 GPS Confidence

The HDOP of the GPS is shown in 5.11(a), and the corresponding confidence calculated is shown in 5.11(b). These signals illustrate that while the GPS is capable of reporting a quality

estimate, it is delayed for several seconds, impacting the performance of the confidence estimator.

5.2.2 Kinect Confidence

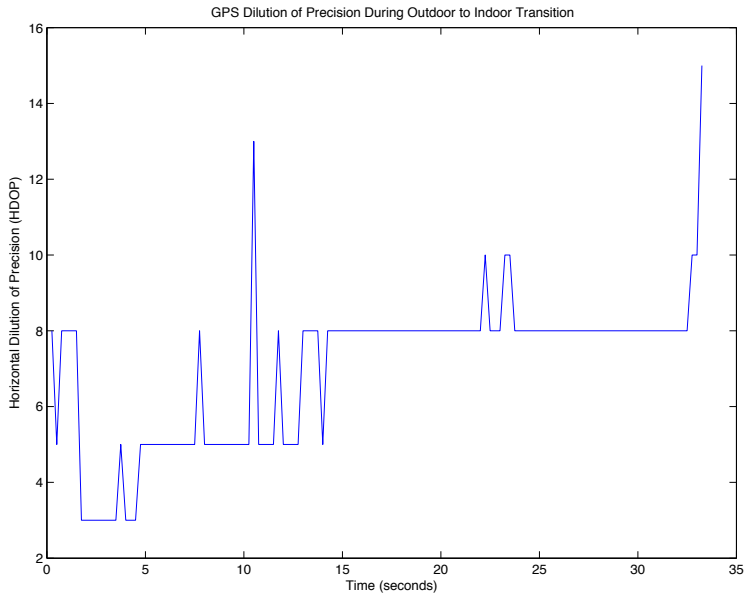
5.12(a) shows the calculated confidence level for the Kinect sensor during the transition. Two distinct peaks are visible on the graph, the first being a section of the house that was exposed in frame, allowing the Kinect to get a depth measurement briefly, the second being the actual transition into an indoor environment. The Kinect scaled confidence is shown in 5.12(b).

5.2.3 Fused Position

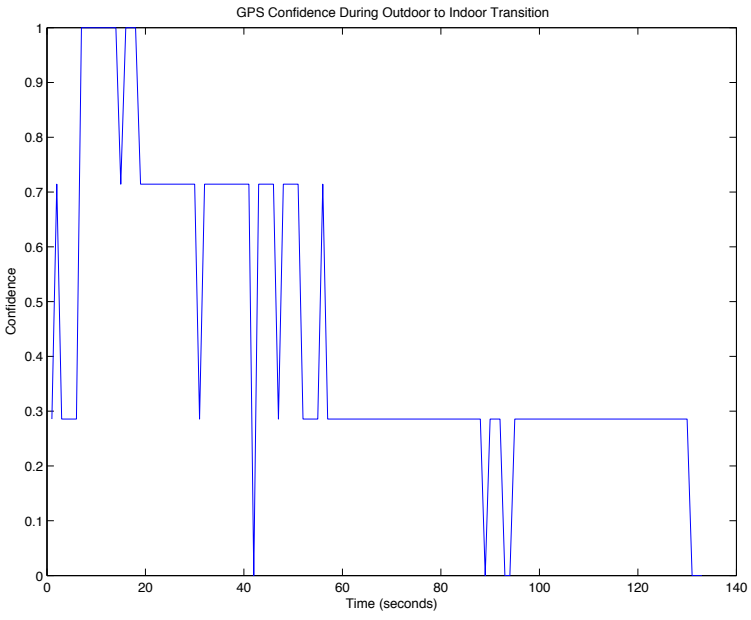
In order to provide a source of information to compare the produced results again, the known movement pattern taken when obtaining the dataset is overlaid onto the output map, along with a simple schematic of the house in which the test was conducted.

5.3 Summary

Using the on board inertial sensors and GPS of the Nexus S, combined with depth maps obtained from the Kinect sensor, the algorithm is capable of providing an estimate of position

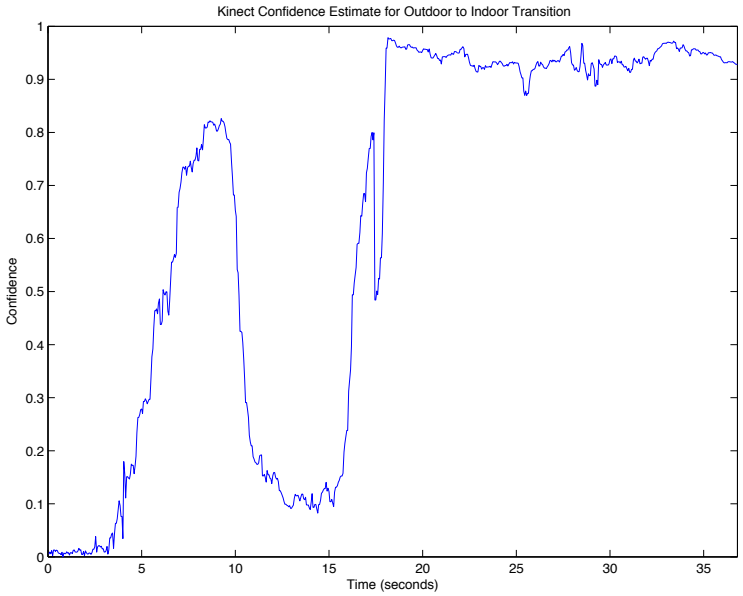


(a) HDOP of GPS during Outdoor to Indoor Transition

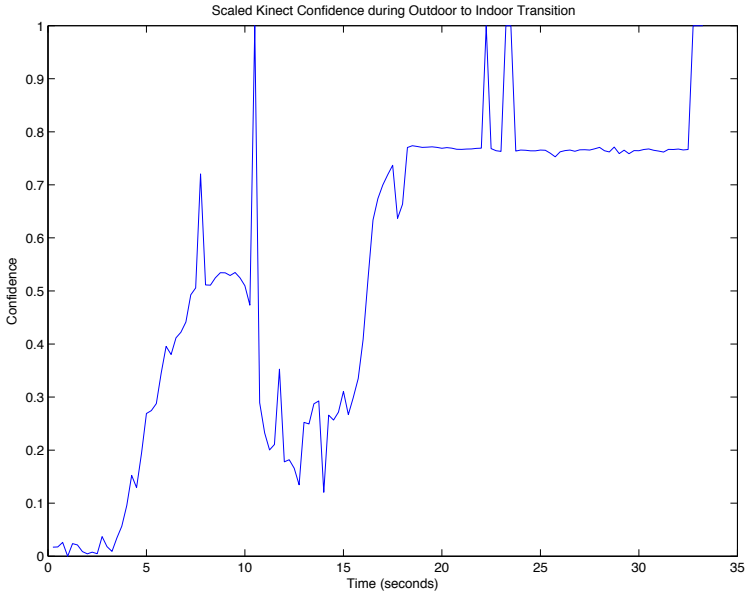


(b) Confidence of GPS during Outdoor to Indoor Transition

Figure 5.11: GPS HDOP And Confidence Estimate



(a) Kinect Confidence during Outdoor to Indoor Transition



(b) Scaled Confidence of Kinect during Outdoor to Indoor Transition

Figure 5.12: Kinect Confidences during Outdoor to Indoor Transition

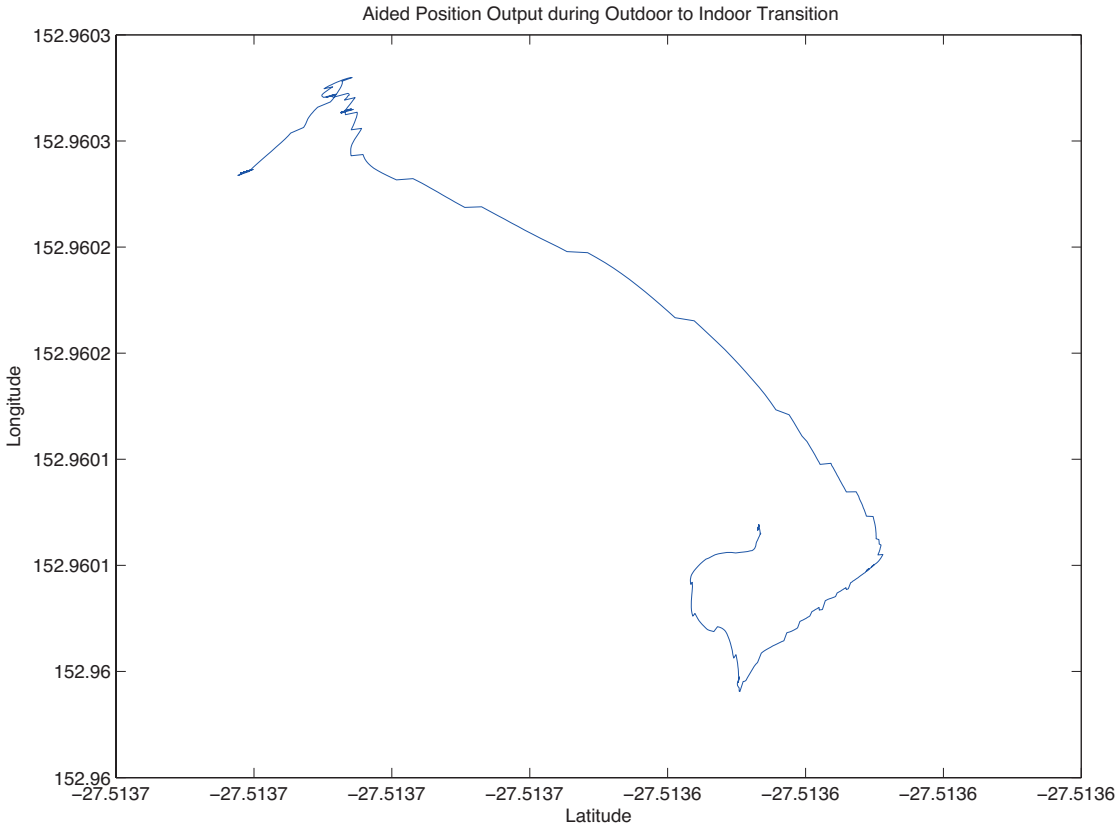


Figure 5.13: Position Estimation of System during Outdoor to Indoor Transition

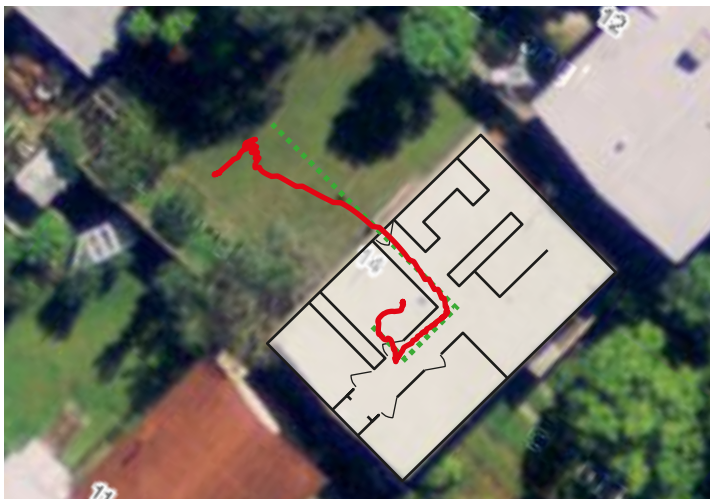


Figure 5.14: Position Estimation of System during Outdoor to Indoor Transition

that is robust to GPS loss.

Chapter 6

Conclusions

Traditional inertial navigation systems are unable to operate under conditions in which a GPS signal isn't available. A solution to this problem was developed and implemented in the form of a cascaded integration filter with on-line confidence estimation for the aiding sensors. A variety of alternative filtering structures were considered, but the requirement to share sensor data between navigation processors ultimately led to the selection of the error-state form of the cascaded integration filter.

The Xbox Kinect was selected as an aiding sensor, and algorithms and processes to transform it's provided RGB and depth images into translation and rotation information were devised. Two registration algorithms were evaluated, ICP and NDT. NDT was chosen for implementation, as it is more robust to larger mismatches between point clouds, and operates more efficiently than ICP. A method for calculating a confidence level in the Kinect sensor was devised, based on the average depth visible in the depth frame. It was shown

that this technique was robust to downsampling of the input images, allowing highly efficient operation. Similarly, a technique for determining a confidence level in the GPS sensor was developed based on the horizontal dilution of precision values provided by the receiver.

The position and velocity calculated from each sensor was used to calculate an error based off of the current position estimate in the reference navigation system. Then, the two confidence levels are scaled, and passed into the integration filter of the system where they are used to weight the corresponding position and velocity estimates. The filter then produces an overall estimate of the error on both the position and velocity, which is fed back into the reference navigation system as a correction update.

The operation of this system was demonstrated on a dataset wherein the system was transitioned from an outdoor environment where GPS was available, to an indoor environment where the GPS signal is blocked. The respective confidence estimates from each aiding sensor reflect this transition, and were used appropriately to correct the overall position estimate, verifying the correct operation of the system.

6.1 Further Work

This thesis has demonstrated a methodology for combining the information from a Kinect and a GPS module to provide a positional estimate robust to losses of GPS signal. This system could be made further robust through the introduction of additional aiding sensors. As well, the primary focus of the implementation of the system was on augmenting the system's estimate position. Future sensors could be expanded such that the orientation or

acceleration of the system were augmented as well. The current implementation of the system is intended to be a proof-of-concept. The aimed goal of achieving standalone operation on an Android smartphone was not achieved due to immature interfacing libraries required for interfacing the Kinect to the Android hardware. The issues encountered were minor, and only impacted on the ability to interface the Kinect to the phone due to restricted time constraints during development. Further work could be done on this area in order to provide a system that is truly standalone.

Appendix A

Matlab Histogram Generator

Listing A.1: MATLAB Depth Image Histogram Generator

```
function [meandepth, maxdepth, peak] = kinect_histogram(prefix, output,
    count)

fig = figure;

skip = 1;

meandepth = zeros(1, count+1);
maxdepth = zeros(1, count+1);
peak = zeros(1, count+1);

for i = 1:skip:count+1
    %subplot(4,6,i);
    %filename = sprintf('%s%02d.png', prefix, i)
    filename = sprintf('%s-%06d.png', prefix, i)
```

```
%if mod(i, 10) == 0
%    filename
%end

ff = imread(filename);
%ff = imresize(ff, 0.05, 'nearest');
fg = rgb2gray(ff);
[b, a] = imhist(fg);

high = b(1);

for j = 1:length(a)
    if b(j) > high
        high = b(j);
        peak(i) = j;
    end
end

meandepth(i) = mean(mean(fg));
maxdepth(i) = max(b);

bar(a/65536, b, 'b');

xlim([0, 0.5])
ylim([0, 50000])
set(gca, 'XTick', 0:0.1:1)
title(sprintf('%s %d', 'Depth Histogram for Frame', i));
xlabel('Intensity');
ylabel('Pixels');

if ~isempty(output)
```

```
            print(fig, '-deps', sprintf('%s-%06d.eps', output, i));
            k = k + 1;
        end
    end
```

Appendix B

Various Transformations

ECEF to Tangent Plane

$$\mathbf{R}_e^t = \begin{bmatrix} -\sin(\phi)\cos(\lambda) & -\sin(\phi)\sin(\lambda) & \cos(\phi) \\ -\sin(\lambda) & \cos(\lambda) & 0 \\ -\cos(\phi)\cos(\lambda) & -\cos(\phi)\sin(\lambda) & -\sin(\phi) \end{bmatrix} \tag{B.1}$$

$$\mathbf{R}_t^e = \left(\mathbf{R}_e^t\right)^T$$

Geodetic to ECEF XYZ

$$x = (R_N + h)\cos(\phi)\cos(\lambda) \tag{B.2}$$

$$y = (R_N + h)\cos(\phi)\sin(\lambda) \tag{B.3}$$

$$z = [R_N(1 - e^2) + h]\sin(\phi) \tag{B.4}$$

ECEF NED to Geographic

$$\begin{bmatrix} v_n \\ v_e \\ v_d \end{bmatrix} = \begin{bmatrix} (R_M + h)\dot{\phi} \\ \cos(\phi)(R_N + h)\dot{\lambda} \\ -\dot{h} \end{bmatrix} \tag{B.5}$$

Geographic to ECEF NED

$$\begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{h} \end{bmatrix} = \begin{bmatrix} \frac{v_n}{R_M + h} \\ \frac{v_e}{\cos(\phi)(R_N + h)} \\ -v_d \end{bmatrix} \tag{B.6}$$

Vehicle to Navigation Frame

$$R_g^b = \begin{bmatrix} \cos(\psi)\cos(\theta) & \sin(\psi)\cos(\theta) & -\sin(\theta) \\ -\sin(\psi)\cos(\phi) + \cos(\psi)\sin(\theta)\sin(\phi) & \cos(\psi)\cos(\phi) + \sin(\psi)\sin(\theta)\sin(\phi) & \cos(\theta)\sin(\phi) \\ \sin(\psi)\sin(\phi) + \cos(\psi)\sin(\theta)\cos(\phi) & \cos(\psi)\sin(\phi) + \sin(\psi)\sin(\theta)\cos(\phi) & \cos(\theta)\cos(\phi) \end{bmatrix} \tag{B.7}$$

DCM to Quaternion

$$\mathbf{b} = \begin{bmatrix} \frac{1}{2}\sqrt{1 + \mathbf{R}_a^b[1,1] + R_a^b[2,2] + R_a^b[3,3]} \\ \frac{R_a^b[3,2] - R_a^b[2,3]}{4b_1} \\ \frac{R_a^b[1,3] - R_a^b[3,1]}{4b_1} \\ \frac{R_a^b[2,1] - R_a^b[1,2]}{4b_1} \end{bmatrix} \tag{B.8}$$

Quaternion to DCM

$$\mathbf{R}_a^b(\mathbf{b}) = \begin{bmatrix} b_1^2 + b_2^2 - b_3^2 - b_4^2 & 2(b_2b_3 - b_1b_4) & 2(b_1b_3 + b_2b_4) \\ 2(b_2b_3 + b_1b_4) & b_1^2 - b_2^2 + b_3^2 - b_4^2 & 2(b_3b_4 - b_1b_2) \\ 2(b_2b_4 - b_1b_3) & 2(b_1b_2 + b_3b_4) & b_1^2 - b_2^2 - b_3^2 + b_4^2 \end{bmatrix} \tag{B.9}$$

Quaternion to Euler

$$\theta = asin(-2(b_2 b_4 + b_1 b_3)) \tag{B.10}$$

$$\phi = atan2(2(b_3 b_4 - b_1 b_2), 1 - 2(b_2^2 + b_3^2)) \tag{B.11}$$

$$\psi = atan2(2(b_2 b_3 - b_1 b_4), 1 - 2(b_3^2 + b_4^2)) \tag{B.12}$$

Appendix C

Kinect Frame Logging Utility

Listing C.1: GPSManager Interface

```
#include <pcl/point_cloud.h>
#include <pcl/point_types.h>
#include <pcl/io/openni_grabber.h>
#include <pcl/common/time.h>
#include <pcl/visualization/cloud_viewer.h>
#include <pcl/io/pcd_io.h>
#include <string>
#include <sstream>
#include <stdio.h>
#include <stdlib.h>

using std::string;

class OpenNILogger
{
```

```
    char *name;
    int frame;
    int skip;
    int saved;
public:
    OpenNILogger(char *filename, int _skip) : name(filename), frame(0),
        saved(0), skip(_skip) {}

    void cloud_cb_ (const pcl::PointCloud<pcl::PointXYZRGBA>::ConstPtr &
        cloud) {
        double now = pcl::getTime();

        if((skip == 0) || ((frame % skip) == 0)) {
            std::stringstream out;
            out << saved;

            pcl::io::savePCDFile(string(name + out.str() + string(".pcd")).
                c_str(), *cloud);
            std::cout << "Saving frame [" << frame << "] as " << saved <<
                std::endl;

            saved++;
        }

        std::cout << "Frame [" << frame << "]@ " << now << std::endl;

        frame++;
    }

    void run () {
        // create a new grabber for OpenNI devices
```

```
        std::cout << "Finding interface..." << std::endl;
        pcl::Grabber* interface = new pcl::OpenNIGrabber("", (pcl::
            OpenNIGrabber::Mode)5, (pcl::OpenNIGrabber::Mode)5);
        std::cout << "Found." << std::endl;

        // make callback function from member function
        boost::function<void (const pcl::PointCloud<pcl::PointXYZRGBA>::
            ConstPtr&)> f =
        boost::bind (&OpenNILogger::cloud_cb_, this, _1);

        // connect callback function for desired signal. In this case its a
             point cloud with color values
        boost::signals2::connection c = interface->registerCallback (f);

        // start receiving point clouds
        interface->start ();

        // wait until user quits program with Ctrl-C, but no busy-waiting
            -> sleep (1);
        while (1) {
             boost::this_thread::sleep (boost::posix_time::seconds (1));
        }

        // stop the grabber
        interface->stop ();
    }
};

int main (int argc, char *argv[])
{
    //std::cout << argc << " " << std::endl;
```

```
    if(argc < 2) {
        std::cout << "Usage: filename [skip]" << std::endl;
        return 0;
    }

    int skip = 0;

    if(argc >= 3) {
        skip = atoi(argv[2]);
        std::cout << "Skipping every " << skip << " frame." << std::endl;
    }

    OpenNILogger v(argv[1], skip);
    v.run ();
    return (0);
}
```

Appendix D

Bibliography

[1] Achtelik, M., Bachrach, A., He, R., Prentice, S., and Roy, N. (2009). Stereo vision and laser odometry for autonomous helicopters in GPS-denied indoor environments.

[2] Analog Devices, Inc. (2010). ADXRS620 (Rev. B). Technical report.

[3] Bryson, M., Johnson-Roberson, M., and Sukkarieh, S. (2009). Airborne smoothing and mapping using vision and inertial sensors. *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*, pages 2037–2042.

[4] B.V., X. (2012). Xsens bv mti-g miniature 6dof ahrs. http://www.xsens.com/en/general/mti-g.

[5] B.V., X. T. (2009). *MTi-G User Manual and Technical Documentation*. XSens Technologies B.V.

[6] Campolo, D., Barbera, G., Schenato, L., Pi, L., Deng, X., and Guglielmelli, E. (2009). Attitude Stabilization of a Biologically Inspired Robotic Housefly via Dynamic Multimodal Attitude Estimation. *Advanced Robotics*, 23(15):2113–2138.

[7] Chahl, J., Thakoor, S., Le Bouffant, N., Stange, G., Srinivasan, M., Hine, B., and Zornetzer, S. (2003). Bioinspired engineering of exploration systems: A horizon sensor/attitude reference system based on the dragonfly ocelli for mars exploration applications. *Journal of Robotic Systems*, 20(1):35–42.

[8] Chao, K. (2002). UAV Attitude Determination by Measuring Earth Electrostatic Field.

[9] Chen, J., Lee, S., and DeBra, D. (1994). Gyroscope free strapdown inertial measurement unit by six linear accelerometers. *Journal of Guidance, Control, and Dynamics*, 17(2):286–290.

[10] Cornall, T. and Egan, G. (2004). Measuring horizon angle from video on a small unmanned air vehicle. In *2nd international conference on autonomous robots and agents*, pages 339–344. Citeseer.

[11] Dana, Peter H. (1994). The global positioning system. http://www.colorado.edu/geography/gcraft/notes/gps/gps_f.html.

[12] Direct, F. (2010). Co-pilot cpd4 2-channel, 2 axis flight stabilization system. http://www.revolectrix.com/cpd4_description_tab.htm.

[13] Durrant-Whyte, H. (2001). Introduction to Estimation and the Kalman Filter. *Australian Centre for Field Robotics*.

[14] Eberly, D. (2007). *3D game engine design: a practical approach to real-time computer graphics*. Morgan Kaufmann Publishers.

[15] Egan, G. and Taylor, B. (2007). Characterisation of infrared sensors for absolute unmanned aerial vehicle attitude determination. *Systems Engineering*.

[16] Engelharda, N., Endresa, F., Hessa, J., Sturmb, J., and Burgarda, W. (2011). Real-time 3D visual SLAM with a hand-held RGB-D camera. *Proc. of the RGB-D Workshop on 3D Perception in Robotics at the European Robotics Forum, Vasteras, Sweden*, 2011.

[17] Epstein, M., Waydo, S., Fuller, S., Dickson, W., Straw, A., Dickinson, M., and Murray, R. (2007). Biologically inspired feedback design for Drosophila flight. In *American Control Conference, 2007. ACC'07*, pages 3395–3401. IEEE.

[18] Euston, M., Coote, P., Mahony, R., Kim, J., and Hamel, T. (2008). A complementary filter for attitude estimation of a fixed-wing uav. In *Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on*, pages 340–345. IEEE.

[19] Farrell, J. A. (2008). *Aided Navigation: GPS with High Rate Sensors*. McGraw Hill.

[20] Fioraio, N. and Konolige, K. (2011). Realtime visual and point cloud slam. *Proc. of the RGB-D Workshop on Advanced Reasoning with Depth Cameras at Robotics: Science and Systems Conf.(RSS)*.

[21] Gebre-Egziabher, D., Elkaim, G., Powell, J., and Parkinson, B. (2002). A gyro-free quaternion-based attitude determination system suitable for implementation using low cost sensors. In *Position Location and Navigation Symposium, IEEE 2000*, pages 185–192. IEEE.

[22] Groves, P. D. (2008). *Principles of GNSS, Inertial, and Multisensor Integrated Navigation Systems*. Artech House.

[23] Gumstix (2010). Gumstix computer-on-modules. http://www.gumstix.com.

[24] Hartley, R. and Zisserman, A. (2003). *Multiple view geometry in computer vision*. Cambridge Univ Pr.

[25] He, R., Prentice, S., and Roy, N. (2008). Planning in information space for a quadrotor helicopter in a GPS-denied environment. *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, pages 1814–1820.

[26] Hill, M. (1972). Introducing the Electrostatic Autopilot. *Astronautics and Aeronautics*, 10(11).

[27] Hol, J. D., Schön, T. B., Luinge, H., Slycke, P. J., and Gustafsson, F. (2007). Robust real-time tracking by fusing measurements from inertial and vision sensors. *Journal of Real-Time Image Processing*, 2(2-3):149–160.

[28] Incorporated, T. I. (2010). Stellaris® LM3S9B96 Microcontroller. Technical report.

[29] jbrunsch (2009). Microsoft Word - DSS-0822_Rev-C_NANO_IMU_Data_Sheet.docx. Technical report.

[30] Julier, S. and Uhlmann, J. (2004). Unscented filtering and nonlinear estimation. *Proceedings of the IEEE*, 92(3):401–422.

[31] Jung, D. and Tsiotras, P. (2007). Inertial attitude and position reference system development for a small UAV. *Jung*.

[32] Kaplan, E. D. (1996). *Understanding GPS: Principles and Applications*. Artech House.

[33] Kim, J. and Sukkarieh, S. (2004). SLAM aided GPS/INS navigation in GPS denied and unknown environments. *The 2004 International Symposium on GNSS/GPS*.

[34] Kionix (2007). Tilt sensing with kionix mems accelerometers.

[35] Koifman, M. and Bar-Itzhack, I. (1999). Inertial navigation system aided by aircraft dynamics. *Control Systems Technology, IEEE Transactions on*, 7(4):487–493.

[36] Lau, T. and Lin, K. (2011). Evolutionary tuning of sigma-point Kalman filters. *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 771–776.

[37] Levin, M. S. and Fimin, A. V. (2012). Design of modular wireless sensor. *arXiv.org*, cs.SE.

[38] Lewis, F. and Stevens, B. (2003). *Aircraft Control and Simulation*. John Wiley & Sons, Inc, Hoboken, N.J., second edition.

[Luukkonen] Luukkonen, T. *Modelling and control of quadcopter*. PhD thesis.

[40] Ma, Y., Soatto, S., Kosecka, J., Ma, Y., Soatta, S., Kosecka, J., and Sastry, S. (2004). *An invitation to 3-D vision*, volume 6. Springer.

[41] Mahony, R., Cha, S., and Hamel, T. (2006). A coupled estimation and control analysis for attitude stabilisation of mini aerial vehicles. In *Proceedings of the Australasian Conference on Robotics and Automation, Auckland, New Zealand*.

[Meier] Meier, L. Towards Visual Sensor Networks Related Research on Micro Air Vehicles.

[43] Meier, L., Tanskanen, P., Fraundorfer, F., and Pollefeys, M. (2011). Pixhawk: A system for autonomous flight using onboard computer vision. *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 2992–2997.

[44] MEMSense, L. (2009). Memsense interial measurement units. http://www.memsense.com/.

[45] Ojeda, L. and Borenstein, J. (2007). Personal dead-reckoning system for gps-denied environments. *Safety, Security and Rescue Robotics, 2007. SSRR 2007. IEEE International Workshop on*, pages 1–6.

[46] Orca Robotics (2010). Orca: Components for robotics. http://orca-robotics.sourceforge.net.

[47] phickey (2010a). Microsoft Word - eb_armat91_xxaxx_b rev 1.2. Technical report.

[48] phickey (2010b). Microsoft Word - ps-imu-3000a-00-01.1. Technical report.

[49] Phillips, W. (2004). *Mechanics of flight*. John Wiley & Sons Inc.

[50] Project, J. (2009). Jsbsim homepage. http://jsbsim.sourceforge.net.

[51] Rusu, R. B. and Cousins, S. (2011). 3D is here: Point Cloud Library (PCL). In *IEEE International Conference on Robotics and Automation (ICRA)*, Shanghai, China.

[52] Rutkowski, A., Quinn, R., and Willis, M. (2006). Biologically Inspired Self-motion Estimation using the Fusion of Airspeed and Optical Flow. *2006 American Control Conference*, pages 2712–2717.

[53] Sane, S. (2003). The aerodynamics of insect flight. *Journal of experimental biology*, 206(23):4191–4208.

[54] Shen, S., Michael, N., and Kumar, V. (2011). Autonomous multi-floor indoor navigation with a computationally constrained MAV. *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 20–25.

[55] Spanos, D. and Murray, R. M. (2005). DISTRIBUTED SENSOR FUSION USING DYNAMIC CONSENSUS. pages 1–6.

[56] Srinivasan, M. (1994). An image-interpolation technique for the computation of optic flow and egomotion. *Biological Cybernetics*, 71(5):401–415.

[57] Srinivasan, M., Poteser, M., and Kral, K. (1999). Motion detection in insect orientation and navigation. *Vision Research*, 39(16):2749–2766.

[58] Sukkarieh, S. (2000). Low cost, high integrity, aided inertial navigation systems for autonomous land vehicles. *Australian Center for Fields Robobtics, University of Sydney*.

[59] Sukkarieh, S., Gibbens, P., Grocholsky, B., Willis, K., and Durrant-Whyte, H. (2000). A low-cost, redundant inertial measurement unit for unmanned air vehicles. *The International Journal of Robotics Research*, 19(11):1089.

[60] Suzuki, T., Kitamura, M., Amano, Y., and Hashizume, T. (2011). High-accuracy GPS and GLONASS positioning by multipath mitigation using omnidirectional infrared camera. *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 311–316.

[61] Systems, S. (2012). Ig-500n miniature gps enhanced attitude and heading reference system. http://www.sbg-systems.com/products/ig-500n.

[62] Tan, C., Mostov, K., and Varaiya, P. (2000). Feasibility of a gyroscope-free inertial navigation system for tracking rigid body motion. *Transportation*.

[63] Taylor, B., Bil, C., Watkins, S., and Egan, G. (2003). Horizon sensing attitude stabilisation: A VMC autopilot. In *18th International UAV Systems Conference, Bristol, UK*.

[64] Texas Instruments, Inc. (2010). LM3S9B96 Microcontroller. Technical report.

[65] Thurrowgood, S., Soccol, D., Moore, R., Bland, D., and Srinivasan, M. (2009). A vision based system for attitude estimation of UAVs. In *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*, pages 5725–5730. IEEE.

[66] Titterton, D. and Weston, J. (2004). *Strapdown Inertial Navigation Technology, 2nd Edition*. The IEE.

[67] Tuck, K. (2007). Tilt sensing using linear accelerometers. *Freescale Semiconductor Application Note AN3107*.

[Walter et al.] Walter, O., Schmalenstroeer, J., and Engler, A. Smartphone-Based Sensor Fusion for Improved Vehicular Navigation. *nt.uni-paderborn.de*.

[69] Wan, E. (2006). Sigma-point filters: An overview with applications to integrated navigation and vision assisted control. In *Nonlinear Statistical Signal Processing Workshop, 2006 IEEE*, pages 201–202. IEEE.

[70] Wu, X. and Ma, S. (2011). Sensor-driven neural controller for self-adaptive collision-free behavior of a snake-like robot. *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 191–196.

[71] XSens Technologies B.V. (2010). Xsens : 3d motion tracking. http://www.xsens.com.