The University of Western Australia

School of Electrical, Electronic and Computer Engineering

Methods for Improving the Absolute Localisation of an Autonomous SAE Vehicle

Christopher C. Blignaut
20936977

Supervised by: **Professor Dr. Thomas Bräunl**
Submitted: **Monday 26th October 2015**

Word Count: 7467

Final Year Project Report for Partial Fulfilment of Bachelor of Engineering

‘The REV Project – Autonomous SAE Team’

Abstract

In 2013, an autonomous high level control system was developed on a Formula SAE race car, enabling the vehicle to autonomously drive along paths defined by GPS waypoints. This provided UWA with a platform as a foundation for further research into driverless performance vehicles. In 2015, the Autonomous SAE Team of the REV project set out to improve a number of aspects on the functionality and robustness of the vehicle. Subsequent to preliminary investigation, it was determined that a limiting factor in the vehicles performance, ease of use and propensity for advanced obstacle avoidance was its poor localisation. A robotic vehicles localisation describes its ability to accurately determine its position in either an absolute (global) frame, or its relative position from its initialized position.

This report outlines the design and implementation of a unique low cost Real-time Kinematic GPS solution, integrated into the SAE vehicle's localisation scheme. This integration is adapted to vehicles current controller and configured to enable survey grade absolute positioning during times of GPS availability. The tests on the RTK implementation indicated significant improvements in the accuracy of absolute position, with a slight compromise in reliability. However more importantly, the capacity for the vehicle to localize accurately identified masked problems in the vehicle control, and thus allowed for progression towards a better functioning autonomous vehicle.

Whilst the complete implementation of the RTK GPS was successful, it highlighted the need for the vehicle to be able to operate even in times of a loss of GPS availability. For a complete localisation solution, methods of dead-reckoning and filter driven interpolation between GPS corrections were investigated.

The second phase of the report aims to implement a tightly coupled recursive Extended Kalman Filter into a sensor fusion scheme using odometry as the main element of positioning. The fusion method software and hardware infrastructure is designed and partially implemented, outlining all possible issues that might arise in the full implementation and testing.

Acknowledgements

I would like to thank the following people for their assistance throughout the course of this project:

Prof. Thomas Bräunl for his guidance, advice and supervision throughout the project.

The other two members of the 2015 Autonomous SAE Team, Thomas Churack and Sonia Miranda for their assistance and collaboration.

The 2015 Autonomous SAE Team Leader Thomas Drage for his support and project management.

My friends and family for their support and encouragement over the course of the year.

See Appendix 9.8 for acknowledgements regarding proprietors or various software and hardware used in this project.

Table of Contents

List of Figures

Abbreviations

Abbreviation	Description
GPS	Global Positioning System
IMU	Inertial Measurement Unit
SLAM	Simultaneous Localisation and Mapping
LIDAR	Light Detection and Ranging
SAE	Society of Automotive Engineers
REV	Renewable Energy Vehicle
RTK	Real-Time Kinematic
GNSS	Global Navigational Satellite System
DGPS	Differential Global Positioning System
EKF	Extended Kalman Filter
INS	Inertial Navigational System
A/R	Acquisition Rate
LLA	Longitude Latitude Altitude
NMEA	National Marine Electronics Association

1. Introduction

1.1 Background and Motivation

1.1.1 Autonomy and Localisation

The automotive industry has targeted vehicles as a platform for automation from as early as the 1920's [1]. In the past decades, major industry leaders and academic institutes have accelerated improvements and emphasized a shift to complete self-sufficient autonomy [2, 3]. This change encompasses an overarching goal of developing commercialized autonomous vehicles that are able to navigate a variety of environments, including urban and dense locations. Whilst trying to reach this goal, it is important to ensure ultimate safety and reliability of the vehicle when in robotic control, and thus new methods are constantly being developed to improve upon existing foundations[4]. The ability for a nonholonomic vehicle to accurately localize itself within both a local and absolute global frame is the underlying requirement for autonomous control[5]. It thus acts as both a precursor to advanced path planning and obstacle avoidance techniques, as well as a constant target to be holistically redesigned to adhere to lower costs and robustness[6].

Current autonomous vehicle localisation methods are numerous in implementation, mostly due to the requirement for adaptability to the specific vehicle platform they are employed on[7] as well as cost constraints. Commonality in the methodologies is however realized through the fusion of at least two or more sensors, such as GPS and IMU [8, 9]. This has been proven to achieve autonomous compliant accuracies whilst still allowing for multiple redundancies and environmental changes. In accordance with the immediate purpose or dependency of the autonomous vehicle, localisation can be tightly coupled with mapping techniques such as SLAM, incorporating sensors such as LIDAR[10]. Cameras and night vision are often used in conjunction with various computer vision techniques to allow for specific application such as detecting road signs and pedestrians [11]. These mapping techniques are often used to aid localisation by identifying landmarks and 'fence posts' so that a vehicle can employ pattern recognition to localize itself in a previously known environment [12].

Absolute positioning is an extension of relative positioning that applies a global correction to the localisation frame of the vehicle. If a vehicle is to drive in unknown environments or by plotted global waypoints, it would require some form of global correction. This is however a large restriction as any sky cover will disable the use of GPS, thus requiring intermediate relative positioning. The ability for a vehicle to interpolate its relative position during global corrections is known as dead reckoning [9]. The most common form of dead reckoning is the use of odometry techniques enabled by wheel speed sensors [13]. One of the more prominent challenges in dead reckoning methodology is the ability to mitigate cascading errors that result in expanding ellipsoidal-form error over time. Attempts have been made to minimize the errors, however this requires computationally expensive processes similar to neural networks and machine learning [14].

1.1.2 Project Context

Formulae SAE [15] is a long-running annual competition organized by the Society of Automotive Engineers, running various events around the world. The new branch of the competition focuses in electric vehicles, from which the vehicle discussed in this report spawns. The vehicle however has since expanded out of the specifications of the competition by becoming fully-autonomous. In 2015, the Autonomous SAE Vehicle Project is orientated towards the improvement of various aspects and short-comings of vehicle control, path planning, obstacle detection, safety procedures and interfaces. These have been identified from a combination of previous work recommendations [16] and preliminary testing on the current performance issues. The project is carried out under the banner of the REV Project at UWA [17] in collaboration with a team, using an electric SAE vehicle with an existing autonomous specification. The focus of the project is to conduct research on an autonomous vehicle platform relating to control algorithms, sensor interfacing and low-cost compliance. The vehicle must maintain its ability to be controlled manually, whilst being able to navigate autonomously to global waypoints and monitor its surroundings for obstacles[18].

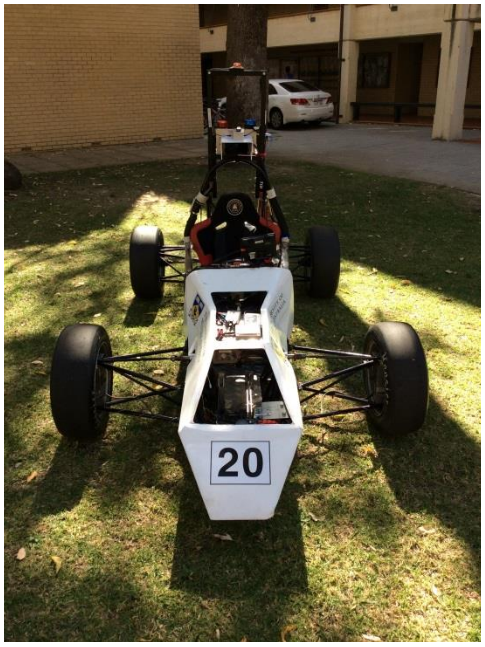


Figure 1 2015 Autonomous SAE Race Car

1.2 Purpose

This report focuses on an encompassing approach to improving the absolute localisation of the SAE vehicle, with respect to both accuracy and functionality. This is initially investigated by the complete implementation of RTK GPS to improve the vehicles absolute localisation. Subsequently, through the aspect of functionality in GPS outage, the effectiveness of integrating odometry into a non-linear sensor fusion scheme is investigated. This ultimately results in the dynamism towards a tightly coupled, closed loop fusion scheme driven by a recursive Extended Kalman Filter. Adopted methodology is integrated into a new novel type of control platform (Raspberry Pi 2) with limited data acquisition interfaces and computational capability.

1.3 Project Delimitations

This project will outline the complete implementation of integrating RTK GPS into the localisation scheme, as well as detail the resulting improvements to the overall functionality. As a direct outcome of the results from this change, further evocations of extended methods are investigated and designed. The implementation of an Extended Kalman Filter with tightly coupled odometry integration is modeled for adaption to the localisation scheme. The software and hardware design is completed sufficiently for primary tests and discussion, however the complete implementation is not covered and tested due to scope constraints. Furthermore, throughout the course of the project, a multitude of auxiliary changes were made to the SAE vehicle as tangential improvements to its overall autonomy. These changes will not be covered, as they do not strictly adhere to the goal of the research to remain self-contained and modular.

1.4 Report Layout

Section 2 will cover the various navigation and positioning systems associated with the sensors to be used throughout the project. This invokes understanding on the relative emphasis placed on both the choice of methodology, as well as design criteria surrounding the limitations of each sensor. Section 3 will briefly cover the theory background to the fusion techniques referred to in the design of the new fusion implementation.

Section 4 of the report will outline the rationalization and final design choices for the RTK-GPS interface. A considerable extension to the project was to design an interface to the control platform for each sensor that is both computationally compatible and functionally sound. The design choice is rationalized against various methods, and a brief comparison is drawn between each.

Section 5 outlines the design of the Extended Kalman Filter, which is outlined together with the required inputs from each sensor.

In section 6, the RTK GPS implementation is tested against the previous QStarz GPS using various metrics that are important for localization. Results are detailed and discussed in regards to the overall implications towards the initial goal and to the system performance as a whole. Furthermore the primary tests involved in the design of the EKF and odometry are described, with the issues uncovered relating to the full implementation.

Finally, in section 7 conclusions are drawn about the effectiveness of both designs and using the results and discussions. Recommendations are made for further work, which covers the full implementation of the EKF, as well as the next steps for the Autonomous SAE project.

2 Navigation and Positioning System

2.1 GNSS

2.1.1 Global Co-ordinate System

Global Navigational Satellite System (GNSS) is an autonomous system of satellites that provide geo-spatial positioning with global coverage to ground based receivers. It allows the receivers to determine their longitude, latitude and altitude using temporal signals transmitted from satellites carrying ephemeris data. There is a multitude of GNSS receivers available at varying costs in the consumer grade level capable of varying levels of accuracy. The more commonly used GNSS in Australia and other western countries is GPS, developed by the US in 1973 [19].

Differential GPS is a method of providing differential corrections from a base station receiver to a roving receiver in order to improve the accuracy of the solution. This tends to be a more expensive alternative to standard GPS, however able to provide better accuracy and faster solutions. To achieve accuracy compliant to autonomous on-road driving, a form of differential GPS is recommended[20], as this allows for accuracies below a meter. An extended form of differential GPS is RTK GPS. This involves setting up a base station at a surveyed known position at close proximity to a roving receiver. The base station and roving receiver will attempt to receive signals from the same satellites and compare carrier cycles in the signal. This is explained in detail in Appendix B.

The SAE Vehicle had an existing positioning system it used for basic autonomous driving in large open areas, achieved by using a standard low cost GPS (QStarz BT-Q818X) and IMU. It required a constant GPS fix, and used an IMU to interpolate movement during incoming GPS data. This was sufficient for basic waypoint navigation; however sensor drift made it unviable for on-road driving. Incoming data from the two sensors was directly read in by the car controller, and used a C++ implementation for interfacing and fusion.

2.1.2 Limitations

A number of factors can lead to poor position or track angle solutions from a GPS, resulting in either inaccurate values or none at all. The largest contributor to a poor solution is a lack of satellites available to the receiver. If the satellites used in the solution is less than six the solution could deviate by up to ten meters [8, 9] and if below four, the fix can be lost entirely. When in differential or RTK fix, the only constant limitation in positioning capability is the reliability to maintain that fix. Other limitations such as poor satellite dilution of precision (DoP) and age of corrections are inevitabilities that are 'force majeure' in that they cannot be removed. This however shifts design criteria implied by these limitations into the second part of the project, which is integrating dead reckoning (Detailed in section 3.1).

2.2 IMU/INS

2.2.1 Inertial Navigational System

Inertial Measurement Units measure inertial forces on their frame to determine accelerations in relative Cartesian co-ordinates. It does this with the aid of 3 degrees of freedom accelerometers, gyroscopes and magnometers. If the initial forces on an object are known as well as initial position and velocity, an IMU can be used to determine position when GPS is unavailable[21]. Furthermore, the IMU will always output a heading relative to true north after calibration, generated by either quaternions or Euler angles (Pitch/Roll/Yaw).

The IMU on the SAE Vehicle is an Xsens MTi IMU [22], which has already been integrated in 2013 by a previous student team. It is currently being used to output heading and for interpolating acceleration, velocity and position in the fusion scheme. It is thus using a method of minimal loose coupling with GPS as it is only being used to aid GPS based localisation.

2.2.2 Limitations

The main fallback of an IMU is its inability to acknowledge any errors accumulated over time. The unit will experience a cascading effect on sensor drift and without a global correction such as GPS, error will increase over time[7]. This limitation is only noticeably reduced when entering the high price range bracket for IMUs, and is highly restrictive in long or high speed driving. The implication of this is a necessity to tightly couple an IMU with some form of odometry like wheel speed sensors or to constantly receive global GPS corrections.

2.3 Odometry

2.3.1 Wheel speed sensors

The use of wheel speed sensors are generally considered essential for many autonomous platforms. They can be used to accurately determine velocity of all four wheels, which can be integrated to provide position. More importantly, they are able to effectively mitigate position drift when the vehicle is stationary. There are numerous simple methods of implementing them into localisation and also using them in feedback control of acceleration and steering. In conjunction with wheel speed, it is beneficial to use sensors such as Hall Effect sensors on the steering position to feed back into the controller. This can be used in conjunction with wheel velocities to accurately determine the direction of travel and the distance travelled.

In the past, wheel speed sensors on the SAE car has been implemented in foundation through both hardware and software. This involved setting up 7 magnets on each wheel in a location that a hall sensor would detect them. The signals are compared with an expected voltage increase (through a comparator), which combined with logic, outputs a high voltage signal into an Arduino Embedded Controller. This year, a few changes had to be made to the initial wiring to get functioning signals, as well as a few software changes to the Arduino to calculate and output the wheel speeds to the main controller.

2.3.2 Limitations

Wheel speed sensors can be inaccurate when the data from them is not properly acquisitioned at set intervals. This makes it important to calculate wheel speed in real time and on a separate dedicated controller rather than the main controller. It can then work through the use of software and hardware interrupts on every pulse count of a sensor on the wheel. The data should be synchronously outputted to the main controller so that the controller can construct an accurate integration on velocity.

3 Sensor Fusion

3.1 Dead Reckoning

Dead Reckoning is the process of obtaining position using the previously known position and advancing that position based on dynamics including velocity, acceleration and heading and is the most common way to localize any robotic system [9, 23, 24]. It is a form of relative positioning, as it can only determine the change in positon from a starting position. Dead reckoning can use a number of sensors such as wheel speed, IMUs, cameras and LIDAR (See introduction).

In this project, dead reckoning is investigated in regards to providing interpolating distances in between times of GPS unavailability. Wheel speed velocity and IMU acceleration is used to construct a measurement vector for the Extended Kalman Filter, configured to advance the system state vector according the a process model defining the vehicle dynamics.

3.2 Kalman Filters

The Kalman filter was developed and published in 1960 by Rudolph Kalman as a recursive filtering method for discrete sets of data [25]. The filter uses a three step algorithm to estimate past, present and future states of a linear system. This is enabled through using actual measurements of various states and processing them with the goal of minimization of the least mean squared error [26]. This results in 'smoothing' out a data set by trying to remove predicted noise and drift and can be used to more accurately determine changes in position and velocity.

The existing Kalman filter in the car is used in both the path planning and path tracking algorithms. This is a standard 3rd order position, velocity and acceleration filter implemented in C++ by Adrian Boeing [27]. This is able to filter the dynamical state of the vehicle to some extent, however the inherent non-linear relationship of the vehicle dynamics will negatively impact the effectiveness of the filter. Whilst there exists multiple flavors of Kalman filters such as unscented Kalman filters, recursive Kalman filters and Extended Kalman Filters, they all require the model on which they act to ultimately be linear. A Kalman filter, namely the Extended Kalman Filter can be modified however to act upon a non-linear system by first linearizing about a covariance of the measurements. This is explained in detail in appendix 9.2 and section 5.1.

4 Data Interface Design and Collection

4.1 RTK GPS – Piksi

4.1.1 Control Structure

For use in this project, the Autonomous SAE Team has acquired a relatively low cost RTK GPS module built by a San Francisco based company, Swift Navigation [28]. It is a first generation product with limited documentation and example code. In an effort to maintain homogeneity in system infrastructure, the interface was programmed into the controller with C++ and a C based message parser. Furthermore, maintaining the GPS receiver and IMU on the same platform, where latency can be mitigated through a common clock will improve synchronization and signal processing capability [24].

This system has bi-lateral communication requirements, as RTK GPS needs to maintain communication between both base station and the roving receiver, as well as between the receiver and the control hardware on the vehicle. The communication between the base station and receiver units simplifies into two feasible options after considering the maximum reached distance between the two (approximately 500m). The first of these is using Wi-Fi to communicate correction data from the base station to the receiver, however this was not chosen so to minimize IP traffic into the Pi for separate reasons. The chosen implementation was to use the existing hardware on the Piksi GPS modules to communicate through two 915MHz 3DR Radio Receivers. Furthermore, the Piksi units each have an external GPS antenna, which will be described in section 4.1.3 further. Figure 2 shows the block diagram with all associated communication channels involved in the setup.

The Raspberry Pi 2 controller is a relatively new product, and runs a Linux based operating system. This had to have extensive work done to it to enable fully functioning connectivity to the RTK GPS, which was documented and shared with other researchers online.

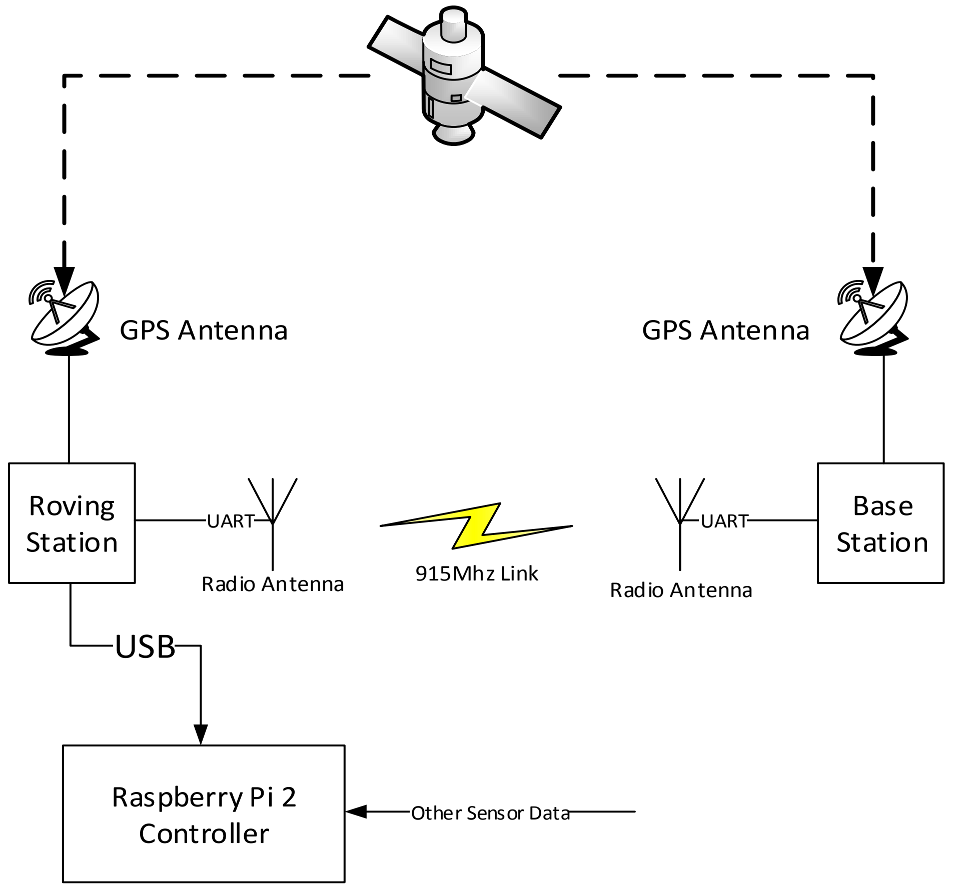
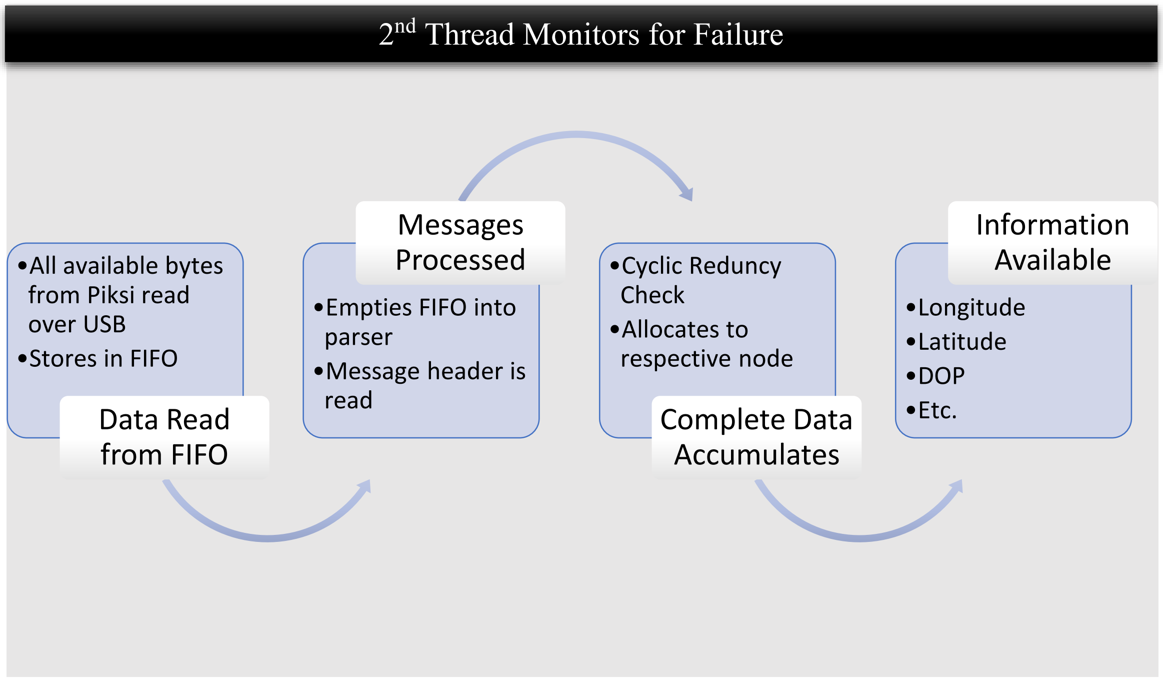


Figure 2 Piksi RTK GPS Set up Block Diagram

4.1.2 Serial Interface

Communication between the roving receiver and base station is limited to USB. The communication protocol being carried through USB is not the standard NMEA 0183 strings commonly used by GPS. The Piksi GPS units use their own proprietary protocol, SBP (SwiftNav Binary Protocol), which is able to communicate more information than NMEA, as well as baseline RTK compatible messages which NMEA is not. This however results in the requirement to develop a specific message parser. This parser was developed in C, using the supplied SwiftNav Piksi libraries for running cycling redundancy checks on the incoming messages, and for generic node allocation. The code for this is shown in appendix 9.4 and is compiled and ran on the Raspberry Pi 2.

When designing the interface, multiple polling and interrupt based methods were investigated and tested. The main considerations in investigating the method was to note the data from the GPS was synchronous, fast and important. The final decision was chosen to poll serial data into a FIFO structure that would be emptied at set intervals. This ensured that no data would be missed and the most recent solutions would be available to the controller. The code process implementation is shown in figure 3.



2nd Thread Monitors for Failure
•All available bytes from Piksi read over USB
•Stores in FIFO
Data Read from FIFO
Messages Processed
•Empties FIFO into parser
•Message header is read
•Cyclic Redundcy Check
•Allocates to respective node
Complete Data Accumulates
Information Available
•Longitude
•Latitude
•DOP
•Etc.

Figure 3 Code Structure for Piksi Serial Interface (See Appendix for Actual Code)

The software implementation uses a multi-threaded approach as well as class orientated programming. A thread separate to the main car control thread is spawned to carry out dedicated processing of the Piksi messages, as well as dedicated thread to monitor for GPS failure. This is compliant with keeping the software running in modular fashion, as well as utilizing the full processing capacity of the Raspberry Pi 2.

4.1.3 Physical Implementation

As described in section 5.1.1, the Piksi units have an external GPS antenna. This requires a metallic ground plane, which has been investigated to provide a large improvement on the antenna signal gain [29]. Furthermore, this antenna cannot be placed close to any magnetic interference such as an IMU's magnometers, as it will reduce both the quality of the IMU as well as the GPS signal. The final design is shown in figure 4, using an enclosure designed by a previous student, Ruvan Muthu-Krishna [29].

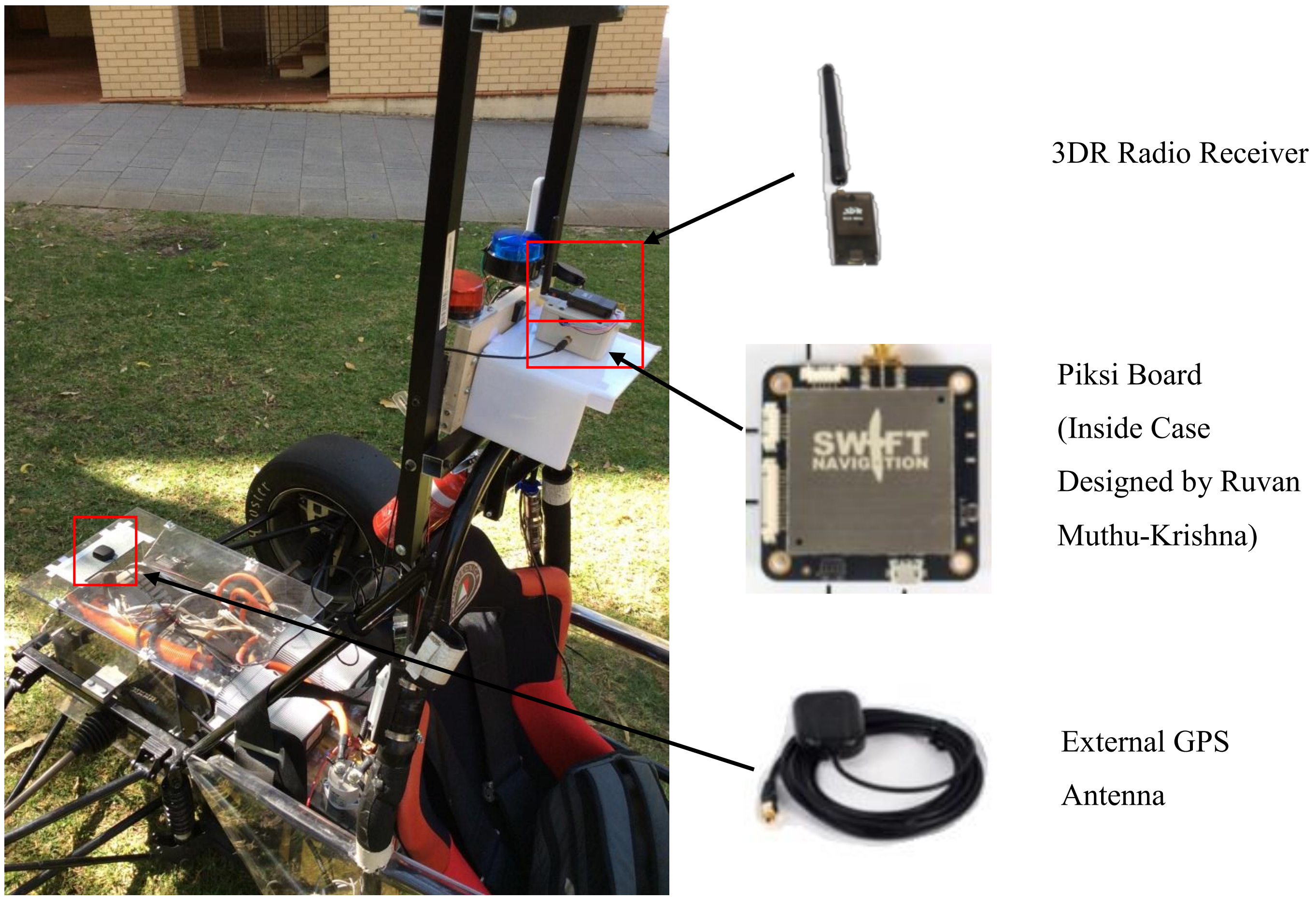


Figure 4 GPS Physical Implementation

4.2 IMU

The IMU was calibrated and configured in previous years to send pitch, roll and yaw to the car controller. The only change made in this configuration was to reduce the rate at which the car controller accepts messages from the IMU from 100Hz to 10Hz. This was done to further increase computational efficiency of the other components of the controller. Whilst increased sensor reading rates are generally beneficial for localisation, there is a diminishing returns factor involved [30], especially if other sensors are incapable of generating solutions at an equally large rate.

4.3 Odometry

The wheel speed sensors are a simple configuration of hall sensors detecting the passing of seven magnets attached to each wheel. As described in section2.3.1, the hardware for this was already on the car, as well as the required circuitry for inputting the received signals into the current Low Level Control. Small wiring changes had to be made to integrate all four wheels into the circuitry, as well as some minor changes to the logic. This was done by the help of another SAE Team member, Sonia Miranda. The Arduino used in the Low Level Control was programmed to calculate the wheel speed for each wheel individually, and then to transmit those values, along with the current steering position through serial to the car controller.

5 Fusion Implementation

5.1 EKF

The extended Kalman Filter is a derivative of the Kalman Filter, with the only difference being its effectiveness in a non-linear environment. This is through computing Jacobian matrices of the non-linear relationship in the state vector evolution to conduct linearization about the estimate of current covariance and mean [31]. The localisation of an autonomous vehicle following complex paths is inherently non-linear, and so the new adaption of the EKF is required.

The most imperative step in developing an EKF is to accurately model the system through a relevant state vector and transition matrix. In the case of this project, a model has been designed and outlined based on an 8 state EKF model which remains simple, yet effective. The model (fully described in appendix 9.2) defines the set of matrices in table 1 to be used in the algorithm described in figure 5.

The vehicles dynamical state is can be completely described by the state vector, $\boldsymbol{x_k}$.

$$\boldsymbol{x_k} = [\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{v}, \boldsymbol{a}, \boldsymbol{\varphi}, \dot{\boldsymbol{\varphi}}]^{\boldsymbol{T}}$$

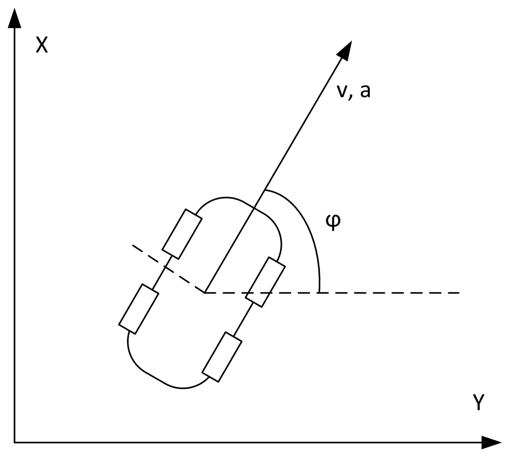


Figure 5 Vehicle Local Co-ordinate frame

The evolution of this state vector is described by the process model, f, using the previous state, a control vector and a process noise. The state vector is also related to the actual measured values of the system, z through a measurement model function, h.

$$\boldsymbol{x_k} = f(\boldsymbol{x_{k-1}}, \boldsymbol{u_{k-1}}, \boldsymbol{w_{k-1}})$$

$$\boldsymbol{z_k} = h(\boldsymbol{x_k}, \boldsymbol{v_k})$$

For our system, the process model is described by the set of equations below.

$$x_k = x_{k-1} + \cos(\varphi_{k-1})(v_{k-1}T + a_{k-1}\,T^2/2) - \sin(\varphi_{k-1})(v_{k-1}\dot{\varphi}_{k-1}\,T^2/2) \\ + 2a_{k-1}\ddot{\varphi}_{k-1}\,T^3/6$$

$$y_k = y_{k-1} + \sin(\varphi_{k-1})(v_{k-1}T + a_{k-1}\,T^2/2) + \cos(\varphi_{k-1})(v_{k-1}\dot{\varphi}_{k-1}\,T^2/2) \\ + 2a_{k-1}\ddot{\varphi}_{k-1}\,T^3/6$$

$$\varphi_k = \varphi_{k-1} + \operatorname{sign}(v_{k-1})(\dot{\varphi}_{k-1}T + \ddot{\varphi}_{k-1}\,T^2/2)$$

$$v_k = v_{k-1} + \mathrm{a}_{k-1}T$$

$$a_k = a_{k-1}$$

The measurement model is described by the matrix.

$$\boldsymbol{H_{i,j}} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

In our system, there is no implementation yet of a control vector **u**, or consideration towards the noise implications towards state evolution. All of the other matrices involved in the algorithm and the already discussed are outlined in table 1.

The covariance matrices $\boldsymbol{Q_k}$ and $\boldsymbol{R_k}$ will need to be individually constructed when the Kalman filter is optimized, so that they capture the noise associated with the sensors involved in processing and measuring the states discussed above.

Table 1 EKF Definitions used in this model

$\boldsymbol{x_k}: [\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{v}, \boldsymbol{a}, \boldsymbol{\varphi}, \dot{\boldsymbol{\varphi}}]^{\boldsymbol{T}}$	State Vector
$\boldsymbol{z_k}: [\boldsymbol{x}, \boldsymbol{y}, \dot{\boldsymbol{x}}, \dot{\boldsymbol{y}}, \ddot{\boldsymbol{x}}, \ddot{\boldsymbol{y}}, \boldsymbol{\varphi}]^{\boldsymbol{T}}$	Measurement Vector
$\boldsymbol{f}(\boldsymbol{x_{k-1}}, \boldsymbol{u_{k-1}}, \boldsymbol{w_{k-1}})$	Non-linear process function (of state vector, control vector and process noise vector respectively)
$\boldsymbol{h}(\boldsymbol{x_k}, \boldsymbol{v_k})$	Non-linear measurement function (of state vector and measurement noise vector respectively)
$\frac{\partial \boldsymbol{f_i}}{\partial \boldsymbol{x_j}} = \boldsymbol{F_{i,j}}$	Transition Matrix: Jacobian of process function with respect to state vector
$\frac{\partial \boldsymbol{h_i}}{\partial \boldsymbol{x_j}} = \boldsymbol{H_{i,j}}$	Measurement Matrix: Jacobian of measurement function with respect to state vector
$\frac{\partial \boldsymbol{f_i}}{\partial \boldsymbol{w_j}} = \boldsymbol{W_{i,j}}$	Jacobian of process function with respect to process noise vector
$\frac{\partial \boldsymbol{h_i}}{\partial \boldsymbol{v_j}} = \boldsymbol{V_{i,j}}$	Jacobian of measurement function with respect to measurement noise vector
$\boldsymbol{Q_k}$	Process Noise Covariance Matrix
$\boldsymbol{R_k}$	Measurement Noise Covariance Matrix

The input to the recursive EKF algorithm (shown in figure 6) is the current measurement vector, as well as the previous state vector and previous estimation covariance P. The first step is to make a prediction of the current state vector and error based on the matrices defined in table 1. The next step is to correct the prediction based on the actual measured values and resulting gain. The output is then a more accurate definition of the current state and corresponding covariance.

```
Recursive EKF Algorithm

Input : $z_k, x_{k-1}, P_{k-1}$
    Data: $f$ and $h$
    for current time $k$ do
        Prediction Step:
        $\hat{x}_k = f(\hat{x}_{k-1}, \hat{u}_{k-1})$
        $F_k = \frac{\partial f}{\partial x} | X_{k-1}, u_{k-1}$
        $H_k = \frac{\partial h}{\partial x} | X_{k-1}$
        $W_k = \frac{\partial f}{\partial w} | X_{k-1}, u_{k-1}$
        $V_k = \frac{\partial f}{\partial v} | X_{k-1}$
        $P_k = F_k P_{k-1} F_k^T + W_k Q_{k-1} W_k^T$
        Correction Step:
        $K_k = P_k H_k^T (H_k P_k H_k^T + V_k R_k V_k^T)^{-1}$
        $\hat{x}_k = \hat{x}_k + K_k (z_k - h(\hat{x}_k)$
        $P_k = (I - K_k H_k) P_k$
    end
Output: $x_k, P_k$
```

Figure 6 Recursive EKF Algorithm [32]

The measurement vector used in the algorithm requires position from the GPS, velocity from the wheel speed sensors and acceleration from the IMU. The car heading is determined by a fusion of the yaw from the IMU and the track angle from the GPS. The algorithm runs on a set time interval, which is calibrated to the rate at which data from the sensors can reliably be interpreted, and that the computed covariance is associated with.

6 Results and Discussion

6.1 RTK GPS

A complete testing procedure was developed at the start of the project for testing various GPS receiver metrics against each other as well as against the performance specification supplied by the manufacturer. This included either quantitative statistical analysis or qualitative analysis on the following:

1. Position Accuracy
2. Velocity Accuracy
3. Orientation Accuracy
4. Availability and Reliability
5. Speed of initialization (Time to acquire feasible fix)
6. Long term dead reckoning tests
7. Integration with current control system
8. Integration with fusion auxiliaries *if required*
9. Additional interfaces

Each metric had a pre-defined requirement checklist, as well as a template for the required data, of which as example is shown in appendix 9.6. The below results are comparing the Piksi to the previously used QStarz GPS on various platforms, including the SAE vehicle and other vehicles that could maintain a faster speeds. This is only a summary of the tests conducted, with the resulting implications.

6.1.1 System Reliability

The reliability was a lumped measurement of both acquisition rate from a cold or hot start and utilization of available satellites over time. The test platform for each GPS was the SAE vehicle itself, as this is able to incorporate the final design location of the receivers on the vehicle, which is expected to effect the results.

6.1.1.1 Acquisition Rate

The acquisition rate for a cold start (after powering on) and a hot start (temporarily loosing signal) can be measured for both receivers simultaneously. This was done using GPSD, a Linux based C++ standard GPS wrapper [Refer to Computer Listing], and the software developed for the Piksi on the Autonomous SAE vehicle. Table 2 shows the averaged results from 10 attempts under various cloud cover and sky density conditions. A two satellite fix was chosen as metric as this is generally when the GPS receiver has initialized and is able to start receiving data.

Table 2 Acquisition Rates - Cold and Hot Start

	QStarz Cold/Hot	**Piksi Cold/Hot**
2 Satellite Fix	28s ± 3s / 1s ± 1s	15s ± 5s / 1s ± 0.5s
Single Point	35s ± 5s / 2s ± 1s	26s ± 6s / 1s ± 1s
Differential	Not Available	46s ± 8s / 2s ± 3s
Floating RTK	Not Available	50s ± 11s / 2s ± 3s
Fixed RTK	Not Available	64s ± 13s / 4s ± 2s

The Piksi has a faster cold start A/R than the QStarz GPS for baseline single point solutions by roughly 34% and for two satellites by roughly 50%. This is likely due to the use of an external antenna on a ground plane for the Piksi, where the QStarz uses an internal smaller antenna without a ground plane. The cold start acquisition rates for differential and RTK solutions (available to only the Piksi) is roughly double that of the single point solution, however the compromise is acceptable to obtain the improved accuracy as detailed in earlier sections. Hot start is consistently low for both the QStarz and Piksi, which is expected. The increase from the nominal 1 second from single point solution to 2 or 3 seconds for differential and RTK is due to delay in correction data being computed by the Piksis' roving receiver. This is important for when the sky above the receiver is blocked by the environment or a building. When the line of sight is re-established, the Piksi receiver is able to generate a temporary solution in a similar interval to the QStarz. It is then able supply a more accurate solution when the RTK algorithm reaches acceptable thresholds for a generated solution.

6.1.1.2 Available Satellites Used

The type and quality of the obtained fix is directly correlated to the number of satellites used in the solution. Receivers will acknowledge available satellites and attempt to use them in a solution if they comply with a low dilution of precision (DOP) and can be interpreted [8]. This test was carried out using the two receivers placed at close proximity, and monitoring the satellites they saw available and what they used in solution simultaneously, with results displayed in figure 7.

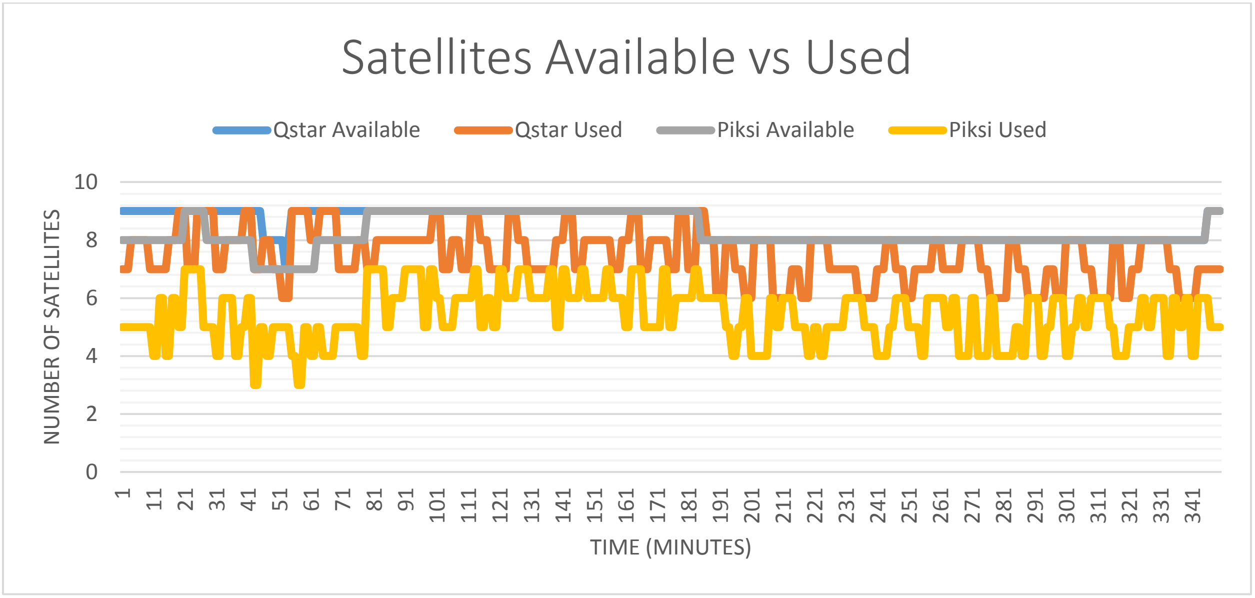


Figure 7 Satellites Available and Used over approximately 6 hours

The number of available satellites remains relatively constant and similar between both receivers. However, it can be seen that the Piksi consistently uses 2 to 3 less satellites in the solution relative to the QStarz. This would not be a problem during an RTK fix as the solution quality is superior even with low satellites numbers, however the single point solution from the Piksi would be poorer than the QStarz. The implication of this is aligned with limitations described in section 2.1.2, and is another motivator for the complete integration of Odometry and EKF.

6.1.2 System Performance

6.1.2.1 Positioning

After setting up the Piksi GPS according to section 4, it could be tested against both the previous GPS and the Piksi's themselves when not in an RTK fix. Since the RTK correction is applied on single point position data, it is possible to log the single point standard solution against the RTK solution simultaneously for a single run.

The base station was setup at a known GPS location and initialized to start sending correction data to the roving receiver. The results of the RTK solution against single point position are shown in figure 8 for a single run.

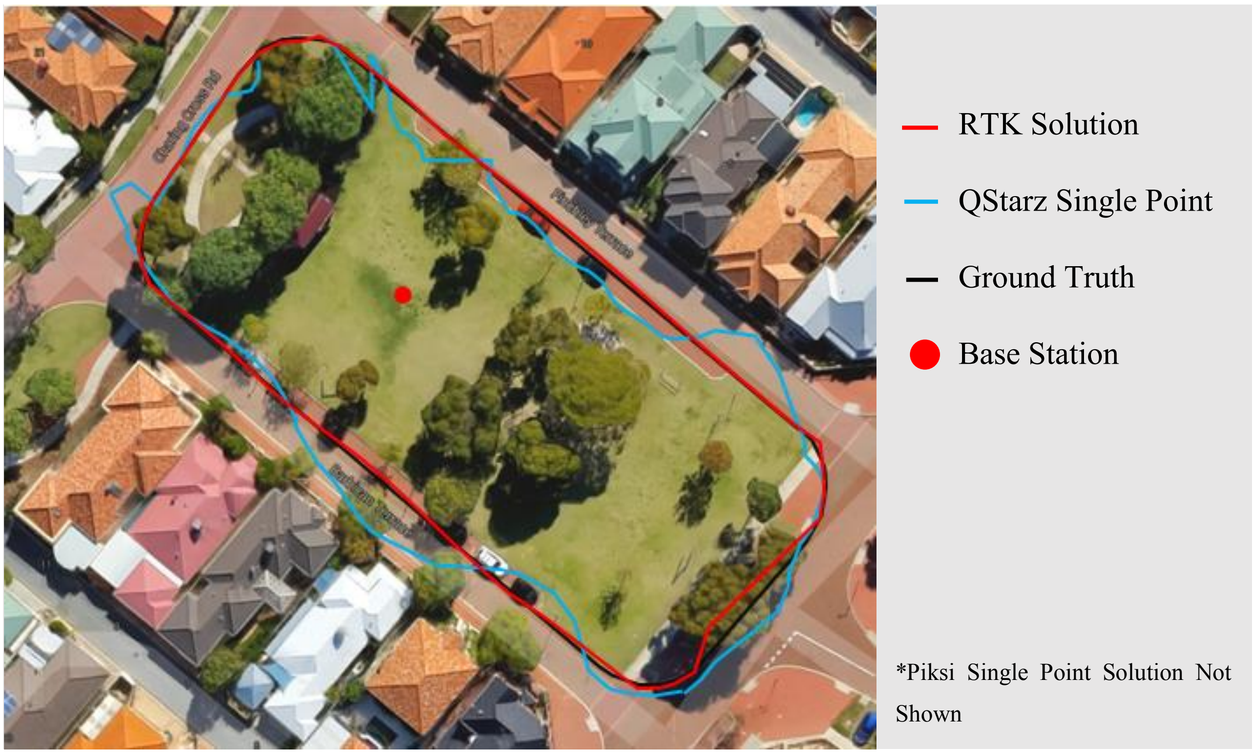


Figure 8 RTK Position Performance vs Single Point Solution

Table 3 Position Accuracy Max, Average and Std

Solution	Maximum Deviation	Average Deviation	Std of Deviation
Piksi RTK Solution	**0.23 m**	**0.04 m**	**0.01 m**
Single Point Solution Piksi	**5.45m**	**3.45m**	**2.42m**
Single Point Solution QStar	**4.42m**	**3.31m**	**1.14m**
*Deviation was determined by linearizing path segments and rotating along an axis such that the perpendicular distance between two paths could be compared. (Mathematica Used)			

The RTK solution shows significant improvements over the single point solution throughout the course of the run. It should be noted that the single point solution from the Piksi was on average less accurate than the solution from the QStarz, despite its RTK solution being vastly superior. This highlights the fact that even with less satellites and poorer triangulation, RTK still outperforms single point solutions. There is deviation regardless of the solution when the rover is close to an object that may cover the sky line of sight (like a tree), however the RTK solution is less perturbed. This is because the GPS receiver will always attempt to generate a single point solution as a baseline, regardless of how many satellites it currently has access to. The RTK solution on the other hand will receive correctional data from the base station over radio during the intermittency and be able to accurately position itself as soon as satellites ephemeris is re-established.

6.1.2.2 Velocity

The receivers were placed on a vehicle maintaining a constant speed on a flat road to measure the velocity jitter over approximately 4 minutes.

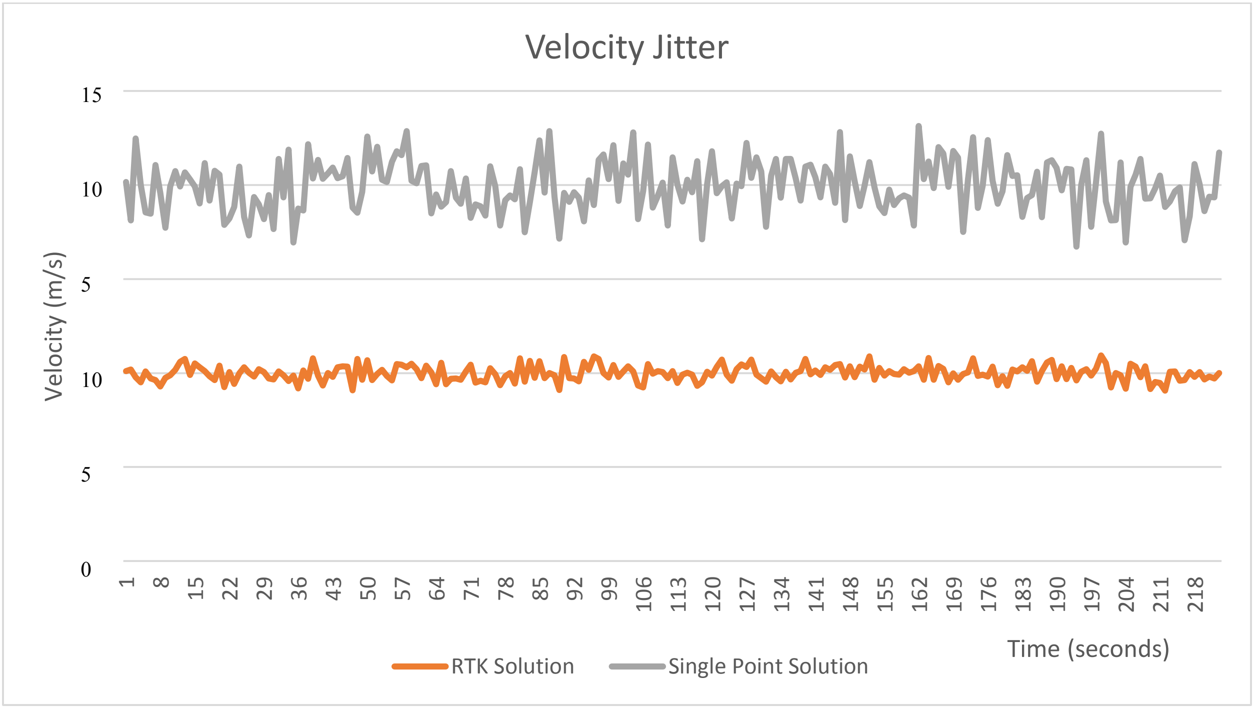


Figure 9 Stacked Scatter Graph for Velocity Jitter

As expected, the baseline velocity measurements from an RTK fix is more stable about the constant value. The standard deviation from the single point solution is 1.279 m/s whereas the RTK GPS has a deviation of only 0.4m/s. The fusion scheme could not previously use the velocity measurement from the only available single point solution, as it was too inaccurate, which agrees with the above result. However, if an RTK fix is used, the velocity is much more reliable, to the extent that it could be used with the Kalman filter to effectively provide position from velocity integration.

6.1.2.3 Heading

The heading generated from GPS is a standard message parsed in normal NMEA 0183 protocol [33], referred to as GPS track angle. The SBP message protocol used by the Piksi currently does not have track angle natively parsed, so it is calculated. The coding changes to the GPS class in control involved with both initializing the GPS and calculating the GPS track angle (along with LLA) is shown in appendix 9.5.

6.1.2.4 Drift

The sensor drift of each GPS was compared over an extended period of time when a nominal amount of satellites (6 or 7) were being used. This gives an indication on the effect the GPS position will have over long drives. It furthermore can be used later on in the modelling of noise variance in a Kalman filter. The Piksi console software (See computer listing) has been used to display drift over a period of 10 minutes.

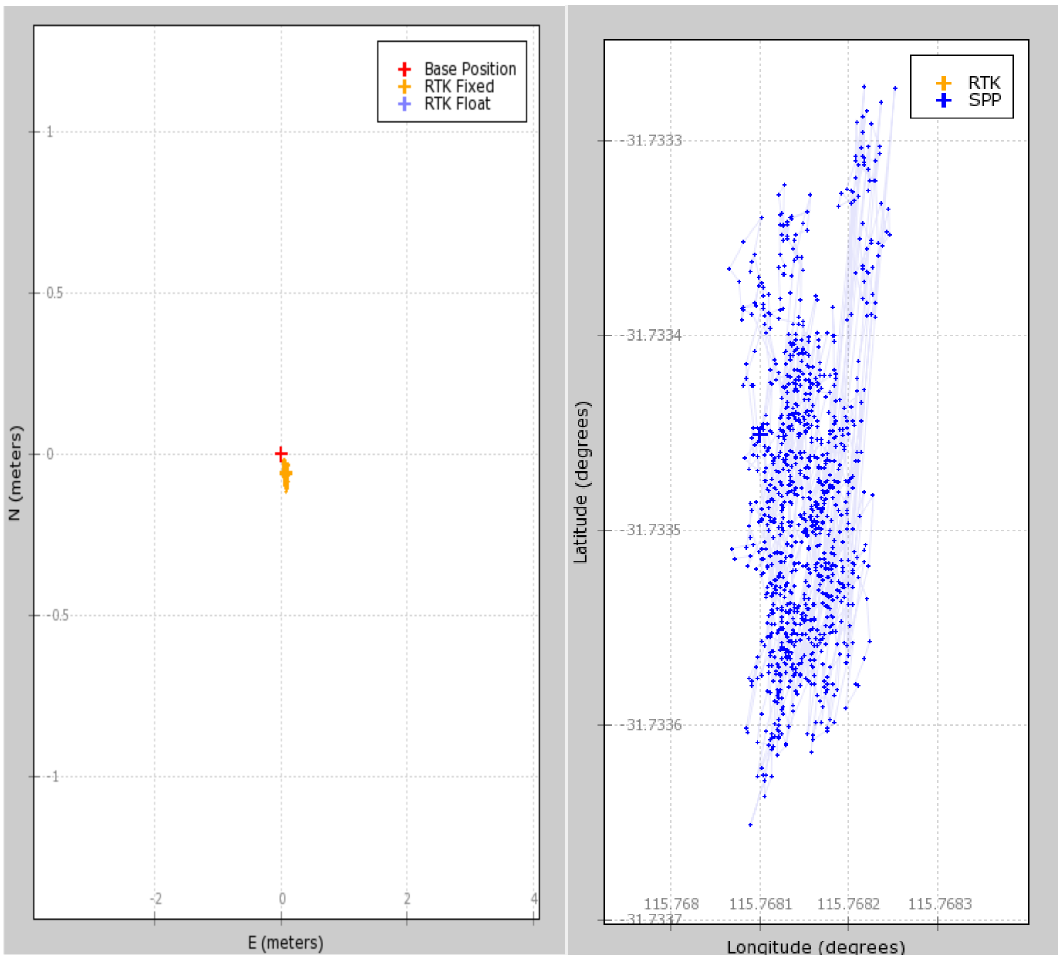


Figure 10 GPS Drift RTK Fixed vs Single Point

The drift from the Piksi in RTK fixed mode is maintained within a maximum of 10 centimeters vertically and similar horizontally. The single point solution on the other hand drifts to a maximum of 8 meters. The spread index improvement from RTK is in the order of 8000%, which is a clear significant improvement. This indicates that the solution from an RTK fix can be trusted to always be accurate regardless of when it is obtained, and no averaging is required. This synergizes well with a proposed Extended Kalman Filter that accepts event based global corrections (See Section).

6.2 Odometry and EKF

Whilst the EKF and odometry have not been fully implemented, primary tests and results will be outlined to identify uncovered issues that may arise during full implementation.

6.2.1 Odometry

The setup described in section 5.3 was tested on the two front wheel speed sensors by creating a temporary serial interface to the low level controller performing the pulse counts and speed calculations. One wheel had its pulse count (magnets per second) sampled every 10 milliseconds and plotted in figure 11.

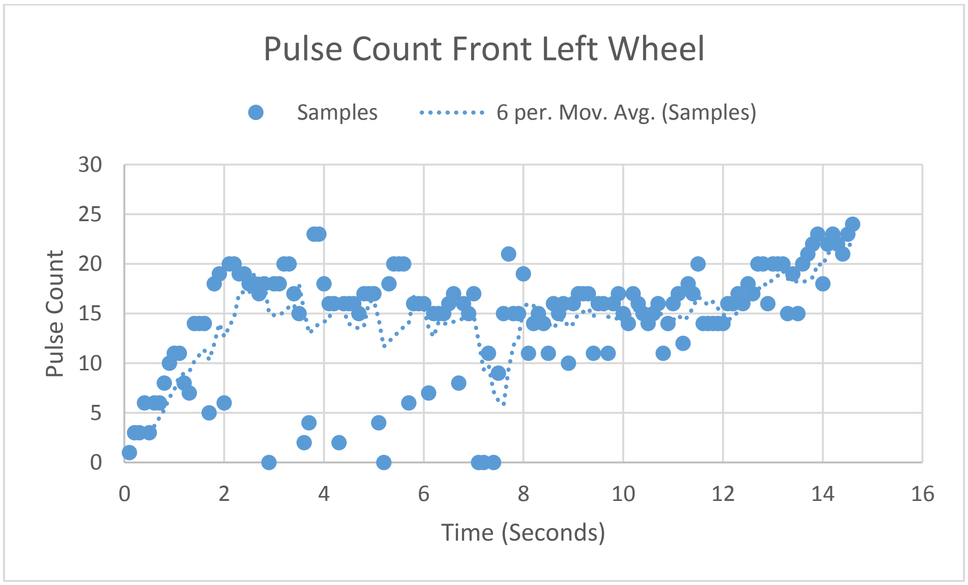


Figure 11 Wheel Speed Pulse Count

The 6 point moving average (60ms) stabilizes the sample somewhat, however it can be seen that there are significant number of samples that are outliers to the trend. This is likely due to the fact that the count works on an interrupt in a microcontroller that already has a number of interrupts and control loops running. The current configuration will indicate speeds sufficiently, however if odometry is to be used in a localisation scheme, especially as the main generation of position through velocity integration, it will have to be more accurate than shown in figure 11.

This indicates that in the future, if wheel speed is to be integrated into a controller, it should be performed by a single thread processor with no other allocations. Some controllers have inbuilt encoder pulse counting hardware that runs on a dedicated separate loop. This ensures that a calculation of wheel speed isn't interrupted by another important task.

6.2.1 EKF

The Extended Kalman Filter algorithm outlined in section 6.1 and fully described in appendix 9.2 has not been completely implemented to the point where it can be tested in conjunction with all available sensors. The code implementation has however been included in the appendix (9.7), which includes the process model and measurement model from section 5 integrated into it. This is a class based programming orientated approach, and enables the controlling software to spawn an EKF class with certain initialization parameters such as time interval and matrix dimensions. The covariance matrices can be created out of class and sent to the class to update the correction step inputs.

The measurements are sent to the class on an interval with time varying capacity, which triggers the algorithm to occur and output the predicted next state and error. Whilst this is a standard recursive approach, it can be expanded to be event triggered, as every step in the process is a public function from within the EKF class. For example, if GPS corrections are received, they can be injected into the correction step at any point, which will update the current state vector.

An issue that may arise from the implementation of the filter is the ability for the control processor to computer the prediction and correction steps at a reasonable speed. The code implementation attempts to reduce the computing strain by performing efficient matrix multiplications, however this will have to be tested.

7 Conclusions

7.1 Conclusion

At the beginning of the project, the localisation of the autonomous SAE vehicle was identified to be a contributing factor to failure in other aspects of control with regards to accuracy and robustness. This generated a clear goal to investigate both hardware and software changes required to improve functionality at low cost. Whilst there exists extensive research in the field of localisation for robotic and autonomous vehicles, no major work was found regarding the implementation of RTK GPS. This is partly due to RTK GPS receivers traditionally costing ten times that of the acquired Piksi receivers.

The Piksis' are a development board that is yet to reach commercialized maturity, and thus little documentation or other similar examples exist. Much of the work regarding successful serial interface with the Piksi, as well as integration into a separate control infrastructure has therefore been shared online.

The conducted tests in positioning demonstrated accuracies that is expected from RTK GPS signifying successful integration, resulting in significant improvements to standard GPS single point solution. This proves that absolute (global) positioning is available for autonomous vehicles at a low cost range, without the requirement for large processing power and strict sensor fusion. Acquisition rate and reliability from the Piksi was deemed to be slightly compromised, however the implications of this is minimal considering the improvements elsewhere.

The RTK GPS was therefore integrated and fully installed into the vehicle as a result. Other improvements made this year to the SAE vehicle were merged into the vehicle and autonomous on-road driving was tested. The improved localisation removed a large factor of instability in vehicle performance, and furthermore helped to identify the next set of improvements to be made to the vehicle.

The next step identified was to improve the ability for the vehicle to localize itself when there is no GPS available. The Extended Kalman Filter was modelled for the vehicle, and aligned with the current choice of IMU and wheel speed. The filter was implemented in C++ and added to the software repository for the vehicle. The existing wheel speed sensors have been extended to be fully implemented in both hardware and software, thus available for the Extended Kalman Filter when it is used. It is expected that the use of wheel speed for dead reckoning in conjunction with the EKF will allow the vehicle to localize itself during times of GPS unavailability.

7.2 Future Work

In regards to the vehicles positioning, the immediate next task should be to completely integrate both the wheel speed sensors and Extended Kalman Filter. As previously mentioned, this will enable the vehicle to remain localized even during long periods of GPS outages, as well as allow for longer drives and faster speeds. The Extended Kalman Filter could also be modified and optimized to various implementations of itself, for example an event based fusion instead of a recursive based fusion, the event being GPS correction. Furthermore the model used to describe the dynamics of the vehicle makes a number of assumptions about wheel slip and car dimensions. When turning the EKF, it would be beneficial to test the vehicles actual motion during various scenarios, and incorporate that into the process model of the filter.

Another possible overall system improvement would be to use the Radio communication channel between RTK base station and roving station for more than just corrections. The SAE vehicle needs a heartbeat signal from an external source during autonomous driving to act as both a failsafe and emergency stop. It is possible to transmit this heartbeat over radio using the Piksis instead of the current method of transmitting over wireless, which is not as reliable.

During an RTK fix, the position solution will be accurate to 10cm, and so this enables a multitude of opportunities to improve other aspects of vehicle control. Most importantly, the actual position of the vehicle can be accurately compared to the planned paths the vehicle attempts to track. This will not only allow for the steering and acceleration control loops to be optimized, but furthermore improve the capability for advanced obstacle avoidance.

The tests after the localisation improvements especially highlighted the need for the vehicle to be able to detect what is in front of it at a large distance. Planned paths work optimally when they are computed with knowledge of the full environment in front of the vehicle. This highlights the need for introducing advanced SLAM techniques from either integrating the LIDAR or a camera into the localisation scheme. If the vehicle is able to detect objects at far distances in front of it and map those objects into its local co-ordinate frame, it will be able to drive smoother curves.

8 References

[1] "Phantom Auto' will tour the city," *The Milwaukee Sentinel*, Google News Archive, 1926.

[2] R. Wallace, "First Results in robot road-following," in Proceedings of the 9th international joint conference on Artificial intelligence, 1985.

[3] T. Kanade, C. Thorpe, and W. Whittaker, "Autonomous land vehicle project at CMU," in Proceedings of the 1986 ACM fourteenth annual conference on Computer science, Cincinnati, Ohio, USA, 1986, pp. 71-80.

[4] L. Danielsson, *Tracking Theory for Preventive Safety Systems*: Chalmers University of Technology, 2008.

[5] B. Carey. "Shelley, Stanford's robotic racecar, hits the track," 19th May, 2015; http://news.stanford.edu/news/2012/august/shelley-autonomous-car-081312.html.

[6] O. K. Hamilton, T. P. Breckon, X. Bai *et al*., "A foreground object based quantitative assessment of dense stereo approaches for use in automotive environments," in Image Processing (ICIP), 2013 20th IEEE International Conference on, Melbourne, VIC, 2013, pp. 418-422.

[7] P. B. S. Roth, "Evaluating Path Tracker Performance for Outdoor Mobile Robots."

[8] G. H. Elkaim, M. Lizarraga, and L. Pedersen, "Comparison of low-cost GPS/INS sensors for Autonomous Vehicle applications." pp. 1133-1144.

[9] Y. Qingmei, and S. Jianmin, "A location method for autonomous vehicle based on integrated GPS/INS." pp. 1-4.

[10] G. Pandey, "An Information Theoretic Framework for Camera and Lidar Sensor Data Fusion and its Applications in Autonomous Navigation of Vehicles," Dissertation, Electrical Engineering - Systems, The University of Michigan, Michigan, USA, 2015.

[11] O. Tsimhoni, J. Bärgman, T. Minoda *et al*., *Pedestrian Detection with Near and Far Infrared Night Vision Enhancement*, The University of Michigan Ann Arbor, Michigan 48109-2150 2004.

[12] R. Siddiqui, and S. Khatibi, "Robust visual odometry estimation of road vehicle from dominant surfaces for large-scale mapping," *Intelligent Transport Systems, IET,* vol. 9, no. 3, pp. 314-322, 2015.

[13] A. Al-Mayyahi, W. Wang, and P. Birch, "Path tracking of autonomous ground vehicle based on fractional order PID controller optimized by PSO." pp. 109-114.

[14] X. Xin, Z. Hongyu, D. Bin *et al*., "Self-learning path-tracking control of autonomous vehicles using kernel-based approximate dynamic programming." pp. 2182-2189.

[15] S. International. "About Formula SAE Series," October 20, 2015; http://students.sae.org/cds/formulaseries/about.htm.

[16] T. H. Drage, "Development of a Navigation Control System for an Autonomous Formula SAE-Electric Race Car," Honours, School of Electrical, Electronic and Computer Engineering, University of Western Australia, Perth, Western Australia, 2013.

[17] "University of Western Australia REV Project," 19 May, 2015; http://therevproject.com/vehicles/sae2010.php.

[18] S. International, "Taxonomy and Definitions for Terms Related to On-Road Motor Vehicle Automated Driving Systems," *Automotive*, On-Road Automated Vehicle Standards Committee, 2014.

[19] N. R. Council, "The global positioning system: a shared national asset: recommendations for technical improvements and enhancements," N. A. Press, ed., Committee on the Future of the Global Positioning System; National Academy of Public Administration, 1995.

[20] M. Spcmgenberg, V. Calmettes, and J. Y. Tourneref, "Fusion of GPS, INS and odometric data for automotive navigation." pp. 886-890.

[21] C. Hide, T. Moore, and M. Smith, "Adaptive Kalman filtering algorithms for integrating GPS and low cost INS." pp. 227-233.

[22] X. Technologies, "Xsens MTi IMU Specifications," 2013.

[23] L. Lei, J. Zhurong, C. Tingting *et al*., "Optimal Model Predictive Control for Path Tracking of Autonomous Vehicle." pp. 791-794.

[24] J. Hol, “Sensor Fusion and Calibration of Inertial Sensors, Vision, Ultra-Wideband and GPS,” Linköping Studies in Science and Technology. Dissertations. No. 1368, Department of Electrical Engineering, Automatic Control, Linköping University, The Institute of Technology, 2011.

[25] R. E. Kalman, “A New Approach to Linear Filtering and Prediction Problems,” *Transactions of the ASME--Journal of Basic Engineering*, vol. 82, no. Series D, pp. 35-45, 1960.

[26] G. B. G. Welch, *An Introduction to the Kalman Filter*, University of North Carolina at Chapel Hill, Chapel Hill, NC 27599-3175, 2006.

[27] A. Boeing. "Kalman Filters," October 9th, 2015; http://adrianboeing.blogspot.com.au/2010/05/kalman-filters.html.

[28] S. Navigation. "About the Piksi," 24th October, 2015; http://www.swiftnav.com/piksi.html.

[29] R. Muthu-Krishna, “Autonomous SAE Car ‐ Real‐ ‐ time Kinematic GPS Integration and Passenger Feedback,” Engineering, Mathematics and Computing, Univeristy of Western Australia, 2014.

[30] O. J. Woodman, *An introduction to inertial navigation*, University of Cambridge, 2007.

[31] M. S. Schlosser, and K. Kroschel, “Limits in tracking with extended Kalman filters,” *Aerospace and Electronic Systems, IEEE Transactions on,* vol. 40, no. 4, pp. 1351-1359, 2004.

[32] L. Marín, M. Vallés, Á. Soriano *et al*., “Multi Sensor Fusion Framework for Indoor-Outdoor Localization of Limited Resource Mobile Robots,” *Sensors (Basel, Switzerland),* vol. 13, no. 10, pp. 14133-14160, 10/21

[33] N. M. E. Association, "Publications and Standards from the National Marine Electronics Association," *Standards*, 2008.

[34] Y. F. a. B. Li, “Wide area real time kinematic decimetre positioning with multiple carrier GNSS signals,” *China Earth Science,* vol. 53, no. 5, pp. 731-740, 2010.

9 Appendices

9.1 RTK GPS

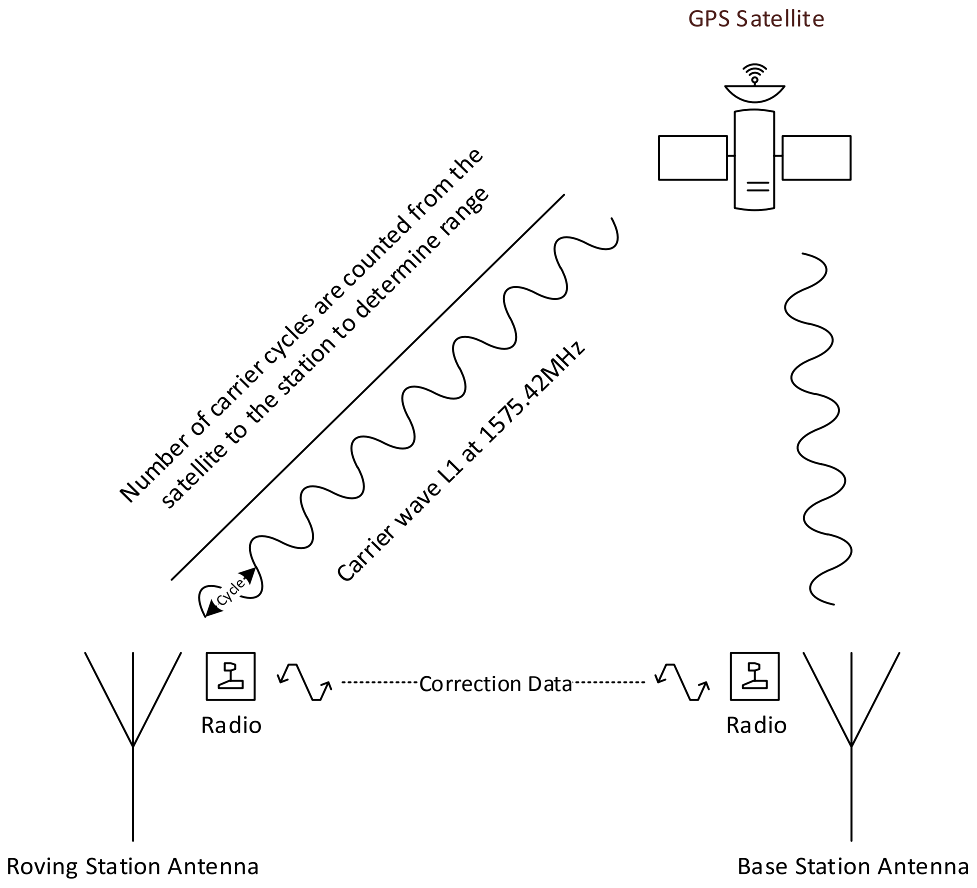

GPS Satellite
Number of carrier cycles are counted from the satellite to the station to determine range
Carrier wave L1 at 1575.42MHz
Cycle
Correction Data
Radio
Radio
Roving Station Antenna
Base Station Antenna

Figure 12 RTK GPS Configuration

Both the roving and base station receive the same carrier signal from a satellite. This receivers then count the carrier cycles it receives from the L1 carrier signal instead of the actual course acquisition data contained in the signal [34]. Signal delay errors resulting from ionospheric and tropospheric delays are constant, and so can be negated by comparing the data received by each receiver, and running an algorithm to fix the phase ambiguities to integer numbers.

At a given epoch for a given satellite, the carrier phase ambiguity observation equation is the following:

$$\emptyset = \rho - I + Tr + c(b_{Rx} - b_{Sat}) + N\gamma + \varepsilon$$

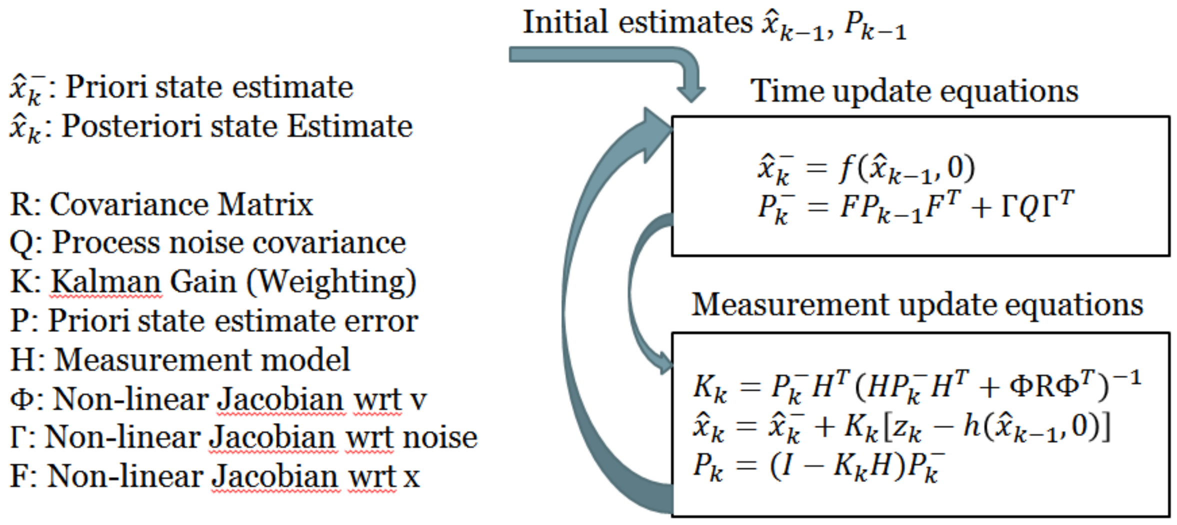
Where:

- ρ is the geometrical range between satellite and receiver
- I is the ionospheric delay, Tr the tropospheric delay
- c is the speed of light, $(b_{Rx} - b_{Sat})$ the clock offset differences
- $N\gamma$ is the phase ambiguity integer multipled by wavelength
- ε is the measurement noise components

Ambiguity Resolution Algorithm:

- First fixed to float numbers using standard least-square techniques
- Set of integer ambiguities are set to the one that optimizes residuals in the surroundings of the float solution
- Carrier measurements are corrected with the integer ambiguities and are used to obtain relative position

9.2 EKF Model


$\hat{x}_k^-$: Priori state estimate
$\hat{x}_k$: Posteriori state Estimate
R: Covariance Matrix
Q: Process noise covariance
K: Kalman Gain (Weighting)
P: Priori state estimate error
H: Measurement model
Φ: Non-linear Jacobian wrt v
Γ: Non-linear Jacobian wrt noise
F: Non-linear Jacobian wrt x
Initial estimates $\hat{x}_{k-1}$, P_{k-1}
Time update equations
$\hat{x}_k^- = f(\hat{x}_{k-1}, 0)$
$P_k^- = FP_{k-1}F^T + \Gamma Q \Gamma^T$
Measurement update equations
$K_k = P_k^- H^T (HP_k^- H^T + \Phi R \Phi^T)^{-1}$
$\hat{x}_k = \hat{x}_k^- + K_k[z_k - h(\hat{x}_{k-1}, 0)]$
$P_k = (I - K_k H)P_k^-$

The state vector is combination of position, velocity and acceleration in x and y space, as well as angle φ.

$$x_k = [x \quad y \quad v \quad a \quad \varphi \quad \dot{\varphi}]^T$$

$$z_k = [x, y, \dot{x}, \dot{y}, \ddot{x}, \ddot{y}, \varphi]^T$$

The non-linear process model is described by the following set of equations theorized by Lars Danielsson [4].

$$x_k = x_{k-1} + \cos(\varphi_{k-1})(v_{k-1}T + a_{k-1}T^2/2) - \sin(\varphi_{k-1})(v_{k-1}\dot{\varphi}_{k-1}T^2/2) + 2a_{k-1}\ddot{\varphi}_{k-1}T^3/6$$

$$y_k = y_{k-1} + \sin(\varphi_{k-1})(v_{k-1}T + a_{k-1}T^2/2) + \cos(\varphi_{k-1})(v_{k-1}\dot{\varphi}_{k-1}T^2/2) + 2a_{k-1}\ddot{\varphi}_{k-1}T^3/6$$

$$\varphi_k = \varphi_{k-1} + \text{sign}(v_{k-1})(\dot{\varphi}_{k-1}T + \ddot{\varphi}_{k-1}T^2/2)$$

$$v_k = v_{k-1} + \mathrm{a}_{k-1}T$$

$$a_k = a_{k-1}$$

The above functions can be used to determine all the other required matrices for the model.

9.3 Receiver Performance Specifications

Table 4 GPS Receiver Specifications

		Ellipse-N	QstarZ BT-818X	Piksi RTK
Configuration		IMU Integrated w/ GNSS	Standard GPS (Fusion with XSens MTi IMU)	RTK GPS
Channels		72	66	
GPS Bands		L1 C/A GPS, GLONASS, Galileo, Beidou	-	L1 GPS, GLONASS, Galileo
GPS Antenna		Internal or External (Model Dependent) Dual Compatibility	Internal with LNA	Internal/External 3.3V
Acquisition Rate	**Cold Start**	26s	35s	-
	Hot Start	<1s	1s	-
Update Rate		10Hz	5Hz	-
Positional Accuracy		2m	10m	1cm
Velocity Accuracy		0.1m/s	0.1m/s	-
Heading Accuracy		$<0.5^\circ$	$<1^\circ$ (IMU Int.)	$<1^\circ$ (IMU Int.)
Power Consumption		<650mW	-	500mW
Compatibility with DGPS / RTK		DGPS / RTK	DGPS	DGPS / RTK
Protocols		Binary eCom, NMEA, ASCII, TSS	NMEA, GSA, GSV, RMC, VTG, GLL	-
Protocol Rates		Up to 921,600ps	115200 bps Baud, 8 bit	3 bit, 16.368 MS/s
Interface		RS-232, RS-422,USB,CAN	Bluetooth, USB	USB, dual UART
Sensors		3 Axis Gyroscopes 3 Axis Accelerometers 3 Axis Magnetometers Temperature Sensors Odometer Compat.	XSens IMU	XSens IMU

9.4 Piksi Message Parser

```c
#include "Piksi_Read.h"

//Serial FIFO
u32 serialread(u8 *buf, u32 n, void* context) {
    context_t ctx = *(context_t *)context;
    return read(ctx.fd, buf, n);
}

//Callback node allocation
void sbp_pos_llh_callback(u16 sender_id, u8 len, u8 msg[], void *context)
{
  pos_llh = *(msg_pos_llh_t *)msg;
}
void sbp_baseline_ned_callback(u16 sender_id, u8 len, u8 msg[], void *context)
{
  baseline_ned = *(msg_baseline_ned_t *)msg;
}
void sbp_vel_ned_callback(u16 sender_id, u8 len, u8 msg[], void *context)
{
  vel_ned = *(msg_vel_ned_t *)msg;
}
void sbp_dops_callback(u16 sender_id, u8 len, u8 msg[], void *context)
{
  dops = *(msg_dops_t *)msg;
}
void sbp_gps_time_callback(u16 sender_id, u8 len, u8 msg[], void *context)
{
  gps_time = *(msg_gps_time_t *)msg;
}

/* ACKNOWLEDGEMENTS TO SWIFTNAV LIBSBP
 * Set up SwiftNav Binary Protocol (SBP) nodes; the sbp_process function will
 * search through these to find the callback for a particular message ID.
*/
void sbp_setup(sbp_state_t *sbp_state, context_t *context)
{
  /* SBP parser state must be initialized before sbp_process is called. */
  sbp_state_init(sbp_state);
  sbp_state_set_io_context(sbp_state, context);
  /* Register a node and callback, and associate them with a specific message ID. */
  sbp_register_callback(sbp_state, SBP_MSG_GPS_TIME, &sbp_gps_time_callback,
                        NULL, &gps_time_node);
  sbp_register_callback(sbp_state, SBP_MSG_POS_LLH, &sbp_pos_llh_callback,
                        NULL, &pos_llh_node);
  sbp_register_callback(sbp_state, SBP_MSG_BASELINE_NED, &sbp_baseline_ned_callback,
                        NULL, &baseline_ned_node);
  sbp_register_callback(sbp_state, SBP_MSG_VEL_NED, &sbp_vel_ned_callback,
                        NULL, &vel_ned_node);
  sbp_register_callback(sbp_state, SBP_MSG_DOPS, &sbp_dops_callback,
                        NULL, &dops_node);
}

//Serial Opening Flags and File Descriptor
int openSerial(char *path, speed_t baud) {
    printf("Opening %s\b\n", path);
    int fd = open(path, O_RDWR | O_NOCTTY);
    struct termios toptions;
    bzero(&toptions, sizeof(toptions));

    if (tcgetattr(fd, &toptions) < 0) {
        perror("tcgetattr");
        return -1;
    }
    toptions.c_cflag = CRTSCTS | CS8 | CLOCAL | CREAD;
    toptions.c_iflag = IGNPAR;
    toptions.c_oflag = 0;
    toptions.c_lflag = 0;
    struct serial_struct ss;
    ioctl(fd, TIOCGSERIAL, &ss);
    ss.flags = (ss.flags & ~ASYNC_SPD_MASK) | ASYNC_SPD_CUST;
    ss.custom_divisor = ss.baud_base / baud;
    ioctl(fd, TIOCSSERIAL, &ss);

    cfsetispeed(&toptions, B38400);
    cfsetospeed(&toptions, B38400);
    toptions.c_cc[VMIN]      = 5;   // blocking read until 5 chars received
    toptions.c_cc[VTIME]     = 0;   // inter-character timer unused
    tcflush(fd, TCIFLUSH);
    if (tcsetattr(fd, TCSANOW, &toptions) < 0) {
        perror("tcsetattr");
        return -1;
    }
    return fd;
}

//Run to initialize Piksi
int OpenPiksi(char *path, int baud) {
    context.path = path;
    context.fd = openSerial(path, baud);
    sbp_setup(&sbp_state, &context);
    return context.fd;
}

//Run to Process Piksi through the addressed Queue
void ProcessPiksi() {
    sbp_process(&sbp_state, &serialread);
}
```

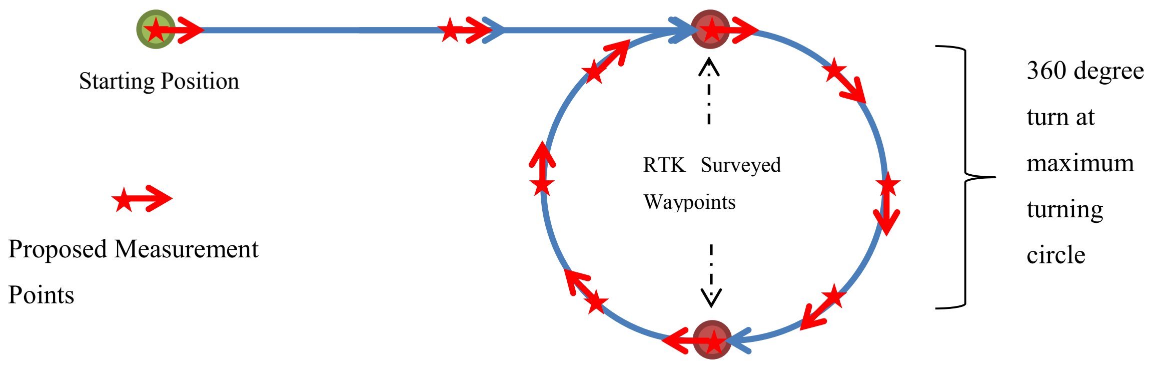
9.5 Changes to GPS Connection Class

```
/*
    Rest of class code Authored by Thomas Drage on 23/7/13
    Not included in this extract
*/
extern "C" {
#include "Piksi_Read.h"
}

//Opens and Intializes Piksi
bool GPSConnection::Open() {
    std::string LogPath = CarControl->LogDir + "/gps.log";
    GPSLog = new Logger(LogPath.c_str());

    fd = OpenPiksi(PIKSI_PATH, PIKSI_BAUD);

    if (fd < 0) {
        Log->WriteLogLine("No Piksi Found - Attempting GPSD...");
        GPSDFound();
    } else {
        Log->WriteLogLine("Piksi - GPS successfully opened and initialized");
        UsingGPSD = false;
        GPSState = true;
    }
    return true;
}
//Starts Multi-Threaded Processing and Monitoring
void GPSConnection::Start() {

    if(GPSState) {

        Run = true;
        if (! UsingGPSD) { m_Thread = boost::thread(&GPSConnection::ProcessMessages, this); }
                else { m_Thread = boost::thread(&GPSConnection::ProcessMessagesDep, this);}
        m_Thread.detach();

        s_Thread = boost::thread(&GPSConnection::Monitor, this);
        s_Thread.detach();
    }
}
//Processes Messages by Emptying FIFO into nodes and probing
void GPSConnection::ProcessMessages() {
    int nullcount = 0;

    while(Run) {

        ProcessPiksi();
        NumSat = pos_llh.n_sats

        if (pos_llh.lat == NULL) {
            nullcount += 1;
            if (nullcount == 1000) {
                Log->WriteLogLine("GPS - Read error! No Fix.");
            }
        } else {
            nullcount = 0;
            Time = gps_time.tow;
            velocity_n = double (vel_ned.n);
            velocity_e = double (vel_ned.e);
            if (vel_ned.n > 0) {
                if (vel_ned.e > 0) {
                    TrackAngle = (atan(velocity_e/velocity_n)* 180 / PI);
                } else {
                    TrackAngle = 360.0 + (atan(velocity_e/velocity_n)* 180 / PI);
                }
            } else if (vel_ned.n < 0) {
                if (vel_ned.e > 0) {
                    TrackAngle = 180.0 + (atan(velocity_e/velocity_n)* 180 / PI);
                } else {
                    TrackAngle = 180.0 + (atan(velocity_e/velocity_n)* 180 / PI);
                }
            }

            NewTrack();
            Speed = sqrt(vel_ned.e*vel_ned.e + vel_ned.n*vel_ned.n)*0.001; // convert to m/s

            Latitude = pos_llh.lat - CarControl->LatOffset;     //Lat
            Longitude = pos_llh.lon - CarControl->LongOffset;   //Long
            NewSpeedAndPosition();
            boost::this_thread::sleep(boost::posix_time::milliseconds(5));
        }
    }
}

void GPSConnection::Monitor() {

    while(Run) {

        if(CarControl->AutoRun) {
            if(! (Time > (OldTime + 1000))) { Log->WriteLogLine("GPS - GPS Fix is old."); }
        }

        OldTime = Time;

        boost::this_thread::sleep(boost::posix_time::milliseconds(500));
    }
}
```

9.6 Example GPS Test Process (Orientation/Track Angle)

Pre Test

1. Test vehicle maximum turning circle, and survey waypoints at maximum diameter


Starting Position
Proposed Measurement Points
RTK Surveyed Waypoints
360 degree turn at maximum turning circle

During Test

1. Initialize the GPS/INS and Logging for all devices
2. Mark measurement points before and after, to ensure the orientation of the car is correctly 'quantized'
3. Manually Drive along updated path waypoints
4. Log dynamic GPS/INS orientation measurements

Post Test

1. Overlay GPS/INS data as orientation vectors onto path defined.
2. Use straight section to test orientation stability, and circle to test general accuracy
3. Orientation of the vehicle to be compared at well-defined equally spaced points on the path
4. Analyze mean and standard deviation of difference between planned path (Use computer algebra software, MATLAB/Mathematica)

Path and Waypoints with GPS Data Overlay	Difference Measurements		Comments / Uncertainty
	Static Mean		
	X:	Y:	
	Static STD		
	X:	Y:	
	Dynamic Mean		
	X:	Y:	
	Dynamic STD		
	X:	Y:	
	Maximum Difference		
	X:	Y:	

9.7 EKF Filter C++ Implementation

```
//
//  Authored by Christopher Blignaut
//

#include "EKFilter.h"
#include <iostream>
#include <fstream>
using namespace std;

EKFilter::EKFilter() {

    //Dimension of State Vector
    x_dimension=0;
    //Dimension of Measurement Vector
    z_dimension=0;
    //Dimension of Control Vector
    u_dimension=0;
    //Dimension of Process Noise Vector
    w_dimension=0;
    //Dimension of Measurement Noise Vector
    v_dimension=0;

    //State Vector
    x_last=NULL;
    x_predicted=NULL;
    x_updated=NULL;
    //Control Vector
    u=NULL;
    //Measurement Vector
    z=NULL;
    z_predicted=NULL;

    ////Matrices
    P_last=NULL;
    P_predicted=NULL;
    P_updated=NULL;
    A=NULL;
    W=NULL;
    Q=NULL;
    H=NULL;
    V=NULL;
    R=NULL;
    S=NULL;
    K=NULL;

}

EKFilter::~EKFilter()
{
    ClearMatrices();
}

int EKFilter::ClearMatrices()
{
    if (x_last!=NULL)
        x_last.release():
    if (x_predicted!=NULL)
        x_predicted.release();
    if (x_updated!=NULL)
        x_updated.release();
    if (u!=NULL)
        u.release();
    if (z!=NULL)
        z.release();
    if (z_predicted!=NULL)
        z_predicted.release();
    if (P_last!=NULL)
        P_last.release();
    if (P_predicted!=NULL)
        P_predicted.release();
    if (P_updated!=NULL)
        P_updated.release();
    if (A!=NULL)
        A.release();
    if (W!=NULL)
        W.release();
    if (Q!=NULL)
        Q.release();
    if (H!=NULL)
        H.release();
    if (V!=NULL)
        V.release();
    if (R!=NULL)
        R.release();
    if (S!=NULL)
        S.release();
    if (K!=NULL)
        K.release();

    return 0;
}

int EKFilter::SetDimensions(int x_dimension, int u_dimension, int
z_dimension, int w_dimension, int v_dimension)
{
    x_dimension=x_dimension;
    u_dimension=u_dimension;
    z_dimension=z_dimension;
    w_dimension=w_dimension;
    v_dimension=v_dimension;

    return 0;
}

int EKFilter::CheckDimension(Mat M1,Mat M2)
{
    if (M1==NULL || M2==NULL)
        return -1;
    if (M1->cols!=M2->cols ||M1->rows!=M2->rows)
        return -1;
    return 0;
}

int EKFilter::InitMatrices()
{
    ClearMatrices();

    //State Vector
    x_last.create(x_dimension,1,CV_32F);
    x_last = Mat::zeros(x_dimension,1,CV_32F);
    x_predicted.create(x_dimension,1,CV_32F);
    x_predicted = Mat::zeros(x_dimension,1,CV_32F);
    x_updated.create(x_dimension,1,CV_32F);
    x_updated = Mat::zeros(x_dimension,1,CV_32F);
    //Control Vector
    u.create(u_dimension,1,CV_32F);
    u = Mat::zeros(u_dimension,1,CV_32F);
    //Measurement Vector
    z.create(z_dimension,1,CV_32F);
    z = Mat::zeros(z_dimension,1,CV_32F);
    z_predicted.create(z_dimension,1,CV_32F);
    z_predicted = Mat::zeros(z_dimension,1,CV_32F);

    ////Matrices
    P_last.create(x_dimension,x_dimension,CV_32F);
    P_last = Mat::zeros(x_dimension,x_dimension,CV_32F);
    P_predicted.create(x_dimension,x_dimension,CV_32F);
    P_predicted = Mat::zeros(x_dimension,x_dimension,CV_32F);
    P_updated.create(x_dimension,x_dimension,CV_32F);
    P_updated = Mat::zeros(x_dimension,x_dimension,CV_32F);
    F.create(x_dimension,x_dimension,CV_32F);
    F = Mat::zeros(x_dimension,x_dimension,CV_32F);
    f.create(x_dimension,x_dimension,CV_32F);
    f = Mat::zeros(x_dimension,x_dimension,CV_32F);
    W.create(x_dimension,w_dimension,CV_32F);
    W = Mat::zeros(x_dimension,w_dimension,CV_32F);
    Q.create(w_dimension,w_dimension,CV_32F);
    Q = Mat::zeros(w_dimension,w_dimension,CV_32F);
    H.create(z_dimension,x_dimension,CV_32F);
    H = Mat::zeros(z_dimension,x_dimension,CV_32F);
    h.create(z_dimension,x_dimension,CV_32F);
    h = Mat::zeros(z_dimension,x_dimension,CV_32F);
    V.create(z_dimension,v_dimension,CV_32F);
    V = Mat::zeros(z_dimension,v_dimension,CV_32F);
    R.create(v_dimension,v_dimension,CV_32F);
    R = Mat::zeros(v_dimension,v_dimension,CV_32F);
    S.create(z_dimension,z_dimension,CV_32F);
    S = Mat::zeros(z_dimension,z_dimension,CV_32F);
    K.create(x_dimension,z_dimension,CV_32F);
    K = Mat::zeros(x_dimension,z_dimension,CV_32F);

    return 0;
}

int EKFilter::Set_NoiseCovariance(Mat Q,Mat R) {

    if (Q.empty() || R.empty())
```

```
        return -1;
    R.copyTo(R);
    Q.copyTo(Q);
    return 0;

}

int EKFilter::PredictionStep() {
    Get_x_predicted();
    Get_F();
    Get_W();
    Get_H();
    Get_V();
    x_predicted.copyTo(x_last);
    return 0;
}

int EKFilter::Get_x_predicted(float dt) {

    x_predicted.at<double>(0,0) = x_last.at<double>(0,0) +
cos(x_last.at<double>(2,0))*(dt*x_last.at<double>(3,0) +
dt*dt*x_last.at<double>(4,0)/2);
    x_predicted.at<double>(1,0) = x_last.at<double>(1,0) +
sin(x_last.at<double>(2,0))*(dt*x_last.at<double>(3,0) +
dt*dt*x_last.at<double>(4,0)/2);
    x_predicted.at<double>(2,0) = x_last.at<double>(2,0);
    x_predicted.at<double>(3,0) = x_last.at<double>(3,0) +
dt*x_last.at<double>(4,0);
    x_predicted.at<double>(4,0) = x_last.at<double>(4,0);

    Get_F(dt);

    return 0;

}

int EKFilter::Get_F(float dt)
{
    F = Mat::eye(x_dimension,x_dimension,CV_32F);
    F.at<double>(0,2) = -
1*sin(x_last.at<double>(2,0))*(dt*x_last.at<double>(3,0) +
dt*dt*x_last.at<double>(4,0)/2);
    F.at<double>(0,3) = dt*cos(x_last.at<double>(2,0));
    F.at<double>(0,4) = dt*dt*cos(x_last.at<double>(2,0))/2;
    F.at<double>(1,2) =
cos(x_last.at<double>(2,0))*(dt*x_last.at<double>(3,0) +
dt*dt*x_last.at<double>(4,0)/2);
    F.at<double>(1,3) = dt*sin(x_last.at<double>(2,0));
    F.at<double>(1,4) = dt*dt*sin(x_last.at<double>(2,0))/2;
    F.at<double>(3,4) = dt;

    return 0;
}

int EKFilter::Get_W()
{
    W = Mat::eye(x_dimension,w_dimension,CV_32F);
    return 0;
}

int EKFilter::Get_P_predicted()
{
    Mat Ft.create(x_dimension,x_dimension,CV_32F);
    transpose(F,Ft);
    Mat Wt.create(w_dimension,x_dimension,CV_32F);
    transpose(W,Wt);
    P_predicted = F * P_last * Ft + W * Q * Wt;

    return 0;
}

int EKFilter::Set_z(Mat* z)
{
    if (CheckDimension(z,z)!=0)
        return -1;
    //cvmCopy(z,z);
    cvReleaseMat(&z);
    z=cvCloneMat(z);
    return 0;
}

/*int EKFilter::Get_z_predicted()
*/

int EKFilter::Get_H()
{

    cvSetZero(H);
    return 0;
}

int EKFilter::Get_S()
{
    Mat* Ht.create(H->cols,H->rows,CV_32F);
    Mat* Vt.create(V->cols,V->rows,CV_32F);
    Mat* H_x_P.create(H->rows,P_predicted->cols,CV_32F);
    Mat* H_x_P_x_Ht.create(H->rows,H->rows,CV_32F);
    Mat* V_x_R.create(V->rows,R->cols,CV_32F);
    Mat* V_x_R_x_Vt.create(V->rows,V->rows,CV_32F);

    cvTranspose(H,Ht);
    cvTranspose(V,Vt);
    MatMul(H,P_predicted,H_x_P);
    MatMul(H_x_P,Ht,H_x_P_x_Ht);
    MatMul(V,R,V_x_R);
    MatMul(V_x_R,Vt,V_x_R_x_Vt);

    cvAdd(H_x_P_x_Ht,V_x_R_x_Vt,S);

    cvReleaseMat(&Ht);
    cvReleaseMat(&Vt);
    cvReleaseMat(&H_x_P);
    cvReleaseMat(&H_x_P_x_Ht);
    cvReleaseMat(&V_x_R);
    cvReleaseMat(&V_x_R_x_Vt);

    return 0;
}

int EKFilter::Get_K()
{

    Mat* Ht.create(H->cols,H->rows,CV_32F);
    Mat* Sinv.create(S->cols,S->rows,CV_32F);
    Mat* P_x_Ht.create(P_predicted->rows,Ht->cols,CV_32F);

    cvTranspose(H,Ht);
    cvInvert(S,Sinv);
    MatMul(P_predicted,Ht,P_x_Ht);
    MatMul(P_x_Ht,Sinv,K);

    cvReleaseMat(&Sinv);
    cvReleaseMat(&Ht);
    cvReleaseMat(&P_x_Ht);

    return 0;
}

int EKFilter::Get_x_updated()
{
    Mat* z_minus_z_predicted.create(z_dimension,1,CV_32F);
    Mat* K_x_z_minus_z_predicted.create(K->rows,1,CV_32F);

    cvmSub(z,z_predicted,z_minus_z_predicted);
    MatMul(K,z_minus_z_predicted,K_x_z_minus_z_predicted);
    cvmAdd(x_predicted,K_x_z_minus_z_predicted,x_updated);

    cvReleaseMat(&z_minus_z_predicted);
    cvReleaseMat(&K_x_z_minus_z_predicted);

    return 0;
}

int EKFilter::Get_P_updated()
{
    Mat* I.create(P_predicted->rows,P_predicted->rows,CV_32F);
    Mat* K_x_H.create(K->rows,H->cols,CV_32F);
    Mat* I_minus_K_x_H.create(K_x_H->rows,K_x_H->cols,CV_32F);

    MatMul(K,H,K_x_H);
    cvSetIdentity(I);
    cvmSub(I,K_x_H,I_minus_K_x_H);
    MatMul(I_minus_K_x_H,P_predicted,P_updated);

    //Make sure P_updated is symmetric
    for (int row=0;row<P_updated->rows;row++)
    {
        for (int col=row;col<P_updated->cols;col++)
        {
            double tmp;
            tmp=cvmGet(P_updated,row,col)+cvmGet(P_updated,col,row);
            tmp=tmp/2;
            cvmSet(P_updated,row,col,tmp);
            cvmSet(P_updated,col,row,tmp);
        }
    }

    cvReleaseMat(&I);
    cvReleaseMat(&K_x_H);
    cvReleaseMat(&I_minus_K_x_H);

    return 0;
}
```

9.8 Software Used

Author	**Software**
Swift Navigation	libSBP
Boost Project	Boost C++ Libraries
Ubuntu Community	Ubuntu
Linux Community	Linux/Lubuntu
Wolfram	Mathematica
Python	Python
Swift Navigation	Piksi Console
CATb	GPSD
XSens	Xsens libraries
Adrian Boeing	Kalman Filter
OpenPR	EKF Extension to CV
OpenCV	Kalman Filter Libraries
Chiark – OpenSource	Putty
Valgrind	Valgrind
Debian	Wheezy
Linux Community	Advanced Packaging Tool
Github	Github