Navigational precision of an autonomous ground vehicle using multiple sensors

by
Frederik Jacobus Potgieter

Thesis presented in partial fulfilment of the requirements for the degree of Master of Engineering (Mechatronic) in the Faculty of Engineering at Stellenbosch University

Supervisor: Dr WJ Smit

March 2016

Declaration

By submitting this thesis electronically, I declare that the entirety of the work contained therein is my own, original work, that I am the sole author thereof (save to the extent explicitly otherwise stated), that reproduction and publication thereof by Stellenbosch University will not infringe any third party rights and that I have not previously in its entirety or in part submitted it for obtaining any qualification.

March 2016

Abstract

This thesis investigated the navigational precision of an autonomous ground vehicle by fusing different sensors as a means of localization and navigation. Different GPS (Global Positioning System) modules (regular, RTK (Real Time Kinematic) and differential GPS) in conjunction with a digital compass and optical encoders were used as sensors for capturing data regarding the robot's position. The Arduino Mega 2560 with an 8-bit Atmel microcontroller was used to control all robot functions while MATLAB was used to plot all navigational output data.

To implement the localization and navigation, background information had to be gained regarding the functioning of the GPS, motor speed control, fusion of sensor data and algorithms used by the sensors. After this was done all the hardware required to implement navigation was purchased, compatibility between all the components was ensured, housings for the sensors were manufactured, the current platform was modified and a power source sufficient to power everything was selected. Next software was implemented to: control the hardware, capture all the data from the sensors, fuse sensor data, map the environment, establish localization and navigate between waypoints and finally display all the captured data to the user.

Before determining the navigational precision of the robot, it needed to be confirmed whether the Piksi RTK GPS could be used as a benchmark for precision comparison of the other sensors. Next case studies tested the navigational precision when: doing multiple runs of the same map, using different complimentary filter values, enabling differential GPS, altering the robot's speed, introducing wheel slippage, magnetic interference and GPS drift is present and when sensors fail.

The high precision with which the Piksi RTK GPS is able to locate the robot gives it the ability to be implemented in various other autonomous and navigation scenarios. Multiple runs of the same map concluded that the consistency of the navigational precision was good enough that data between different runs could be compared. The optimal complimentary filter constant was found experimentally, it was seen that differential GPS resulted in more precise navigation and that the lowest robot speed resulted in the most precise navigational results. Wheel slippage and magnetic interference had a large effect on the robot's position estimation while GPS drift had little effect. Finally it was seen that any single sensor failure resulted in the robot being unable to navigate.

Future work that affects the navigational precision can include: use of different data fusion algorithms, fusion of Piksi RTK GPS data with odometry data, more stable or different robot platform, additional sensor to detect wheel slippage, algorithm to detect magnetic interference and the use of stronger Piksi RTK GPS direct communication antennas.

Opsomming

Hierdie tesis ondersoek die navigasie presisie van ’n outonome voertuig deur die integrasie van verskillende sensors as ’n wyse van lokalisering en navigasie. Verskillende GPS (globale posisioneringstelsel) modules (gewoon, intyds kinematies en differensiële GPS) in samewerking met ’n digitale kompas en optiese enkodeerders is gebruik as sensors vir die insameling van data aangaande die robot se posisie. Die Arduino Mega 2560 met ’n 8-bis Atmel mikrobeheerder is gebruik om al die robot funksies te beheer terwyl MATLAB gebruik is om die navigasie uitset data te vertoon.

Om die lokalisering en navigasie te implementeer het hulle eerstens agtergrond kennis aangaande GPS, motor snelheid beheer, integrasie van sensor data en algoritmes wat deur sensors gebruik word ingesamel. Na afloop daarvan is al die nodige hardeware om navigasie te implementeer aangekoop, versoenbaarheid tussen al die komponente verseker, omhulsels vir die sensors vervaardig, die huidige platform aangepas en daar is besluit op ’n voldoende kragbron om alles aan te dryf. Daarna is die sagteware geïmplementeer wat: al die hardeware beheer, al die data van die sensors ontvang, die sensor data saamsmelt, ’n kaart van die omgewing skep, tussen koördinate navigeer en uiteindelik al die ingesamelde data aan die gebruiker vertoon.

Voor hulle kon kyk na die navigasie presisie van die robot, het hulle eers bepaal of die intyds kinematiese GPS gebruik kan word as ’n maatstaf vir die vergelyking van presiesheid van die ander sensors. Volgende is daar deur gevallestudies die navigasie presisie getoets wanneer: herhaaldelike lopies van dieselfde kaart gedoen is, verskillende komplimentêre filter waardes gebruik is, die differensiële GPS aangeskakel is, die robot se snelheid verander is, wielglip ingesluit is, magnetiese inmenging en GPS dryf teenwoordig is asook wanneer enige van die sensors faal.

Die hoë presisie waarmee die Piksi intyds kinematiese GPS in staat was om die robot te lokaliseer gee dit die vermoë om in verskeie ander outonome en navigasie verwante situasies geïmplementeer te word. Verskeie lopies van dieselfde kaart het gewys dat die konsekwentheid van die navigasie presisie voldoende was om data tussen verskillende lopies met mekaar te vergelyk. Die optimale komplimentêre filter konstante is eksperimenteel gevind, dit is waargeneem dat differensiële GPS tot meer presiese navigasie gelei het en dat die stadigste robot snelheid die mees presiese navigasie resultate gelewer het. Wielglip en magnetiese inmenging het ’n groot invloed op die robot se posisie vasstelling gehad, terwyl GPS dryf ’n klein effek gehad het. Uiteindelik is waargeneem dat ’n enkel sensor faling veroorsaak het dat die robot nie kan navigeer nie.

Toekomstige werk wat die navigasie presisie affekteer kan die volgende insluit: die gebruik van verskillende data integrasie algoritmes, die integrasie van die Piksi RTK GPS data met verplasingsmeter data, meer stabiele of ander platform, addisionele sensor om wielglip waar te neem, algoritme om magnetiese inmenging waar te neem en sterker Piksi RTK GPS kommunikasie antennas.

Dedicated to my family and friends
who are always there for me

Acknowledgements

The financial assistance of the National Research Foundation (NRF) towards this research is hereby acknowledged. Opinions expressed and conclusions arrived at, are those of the author and are not necessarily to be attributed to the NRF.

I wish to thank my supervisor, Dr WJ Smit, for his advice and guidance in my research.

Table of Contents

List of figures

List of tables

Nomenclature

$\bar{A}$	state transition matrix that applies the system parameters from the previous epoch on the current ones when process noise is absent
$\bar{A}_s$	symmetrical matrix
B	geomagnetic field strength
$\bar{B}$	input matrix that determines how much each system input affects the state vector
c	speed of light in a vacuum, taken as $299\,792\,458\ \mathrm{m \cdot s^{-1}}$
d	distance of a line on the surface of the sphere
$\bar{D}_{comb}$	combination of $\bar{D}_{soft}$, $\bar{D}_{scale}$ and $\bar{D}_{misal}$
$\bar{D}_{comb,cal}$	estimate of the soft-iron distortion
$\bar{D}_{hard}$	hard-iron distortion matrix
$\bar{D}_{hard,cal}$	estimate of the hard-iron distortion
$\bar{D}_{misal}$	misalignment matrix
$\bar{D}_{scale}$	scale factor matrix
$\bar{D}_{soft}$	soft-iron distortion matrix
e	difference between the setpoint and plant output
f	probability density function
g	gravity of earth, taken as $9{,}81\ \mathrm{m \cdot s^{-2}}$
$\bar{G}$	gravitational vector
$\bar{G}_m$	misalignment matrix
$\bar{G}_o$	offset
$\bar{G}_p$	combination of $\bar{G}_m$, $\bar{G}_s$ and $\bar{G}_o$
$\bar{G}_s$	scale factor
$\bar{H}$	transformation matrix which maps and converts the state vector values into the current equation domain
I	ionospheric distance error
i	current
$\bar{I}$	identity matrix
k	epoch
K_c	gain at the point of oscillation
K_d	differential gain coefficient
K_i	integral gain coefficient
K_p	proportional gain coefficient
$\bar{K}_k$	Kalman gain, which minimizes $\bar{P}_{k-1}$
M	payload
$\bar{M}$	magnetic field vector
$\bar{M}_r$	rotated magnetic field vector
$\bar{M}_{r,corrected}$	corrected magnetometer values
N	carrier phase ambiguity / integer ambiguity
n	number of satellites
n_m	number of measurements
P_c	period of the oscillations
$\bar{P}_k$	posteriori estimate error covariance matrix
$\bar{P}_k^-$	priori estimate error covariance matrix
$\bar{Q}$	process noise covariance matrix
$\bar{Q}_0$	centre of the ellipsoid

$\bar{Q}_{om}$	orthogonal matrix that has the eigenvectors of $\bar{A}$ as column entries
R	resistance
r	radius of the earth, taken as $6\ 378{,}137$ km
$\bar{R}$	measurement noise covariance matrix
$\bar{R}_p$	pitch rotation matrix
$\bar{R}_r$	roll rotation matrix
$\bar{R}_y$	yaw rotation matrix
T	tropospheric distance error
t	time
t_b	time difference between receiver clock and satellite clock
t_{dT}	satellite offset from reference time
t_{dt}	receiver offset from reference time
t_i	satellite i time
t_r	time message was received as recorded by the receiver
t_t	message true reception time
$\bar{\mathrm{T}}$	arbitrary rotation
$\bar{u}_k$	input vector that contains all the system inputs
V	voltage
$\bar{v}_k$	measurement noise vector containing noise values for all the system measurement values
$\bar{w}_k$	is the process noise vector containing noise values for all the state vector values
x_{avg}	measurement average
x_i	earth centered system x coordinate for satellite i
x_j	measurement j
x_r	earth centered system x coordinate for receiver
$\bar{x}_k$	state vector storing the model system parameters
$\bar{x}_k^-$	priori state estimate
y_i	earth centered system y coordinate for satellite i
y_r	earth centered system y coordinate for receiver
z_i	earth centered system z coordinate for satellite i
z_r	earth centered system z coordinate for receiver
$\bar{z}_k$	vector containing all the system measurements
β	inclination angle
χ	latitude
$\Delta\delta$	heading between two sets of coordinates
$\Delta\lambda$	absolute difference between the longitude of two coordinates
$\Delta\sigma$	angle between the two sets of coordinates with its origin at the earth centre
ε	carrier phase measurement noise, satellite ephemeris errors and multipath errors
γ	carrier phase measurement
λ	carrier wavelength
μ	mean
ω	current heading
ϕ	roll
ψ	yaw
ρ	straight distance between receiver and satellite

σ	standard deviation
σ^2	variance
σ_x	standard deviation in a northern direction
σ_y	standard deviation in an eastern direction
τ	time integration variable between 0 and the current time t
θ	pitch

1 Introduction

This section will provide background information regarding autonomous navigation, ways of robot localization, the original aim of the research and the current systems in use. The objectives – which include a literature study that must be completed, hardware and software design and implementation, and obtaining experimental results – will be listed. Lastly the motivation for conducting this research will be discussed.

1.1 Background

Extensive research has been done in the field of autonomous navigation for both indoor and outdoor situations using different platforms (aerial, ground and underwater). Let's first give a more definitive description for the term "autonomous navigation". This includes condition monitoring, continuous sensing of the environment, navigation by processing sensor data and / or previously known data of the environment and finally task execution. Condition monitoring in the scope of this research topic only includes the robot being able to sense when operating power is low and being able to navigate to the base station for recharging. Sensing the environment will be done using various sensors, including optical encoders, a compass and different GPS modules. Using this sensor data the robot can navigate to a goal state. Task execution entails the robot being able to autonomously navigate between waypoints and return to its base station.

When looking at robot localization there are two methods that can be applied, these are the absolute and relative approaches. The absolute approach requires prior knowledge of the environment and some way for the robot to take measurements with regard to known parameters present in the environment. Using GPS is a way of applying the absolute approach. The relative approach only uses sensors on the robot to determine its position with regard to its starting point. Two popular ways to get relative data is by using odometry and / or a compass as a way of dead reckoning as discussed in (Borenstein and Liqiang Feng, 1996). By combining these two methods one can fuse the data received from the various sensors to obtain a more accurate position estimate of the robot as discussed in (Goel et al., 2015).

This reseach was started with the aim of ultimately being implemented on an autonomous ground vehicle in a CSP (concentrating solar power) plant. The STERG (Solar Thermal Energy Research Group) at Stellenbosch University currently investigates solar energy as a way of generating electricity on a large scale. A common way of generating electricity using this approach is by implementing a CSP system, where a typical $11\,\mathrm{MW}$ power plant (for example the PS10 Solar Power Plant) has in excess of 600 individual heliostats (Power Technology, 2015). Each heliostat consists of a collection of mirrors with adjustable orientation, reflecting the sun to the central solar power tower. These mirrors need to be cleaned and inspected on a regular basis (daily), to ensure optimal functioning and efficiency.

Current systems comprise of personnel operating the machinery responsible for cleaning the mirrors, but can possibly be replaced by an autonomous ground vehicle. Such an autonomous vehicle will therefore consist of a motorized robot platform, a navigation module and the cleaning component. This part of the research will be limited to the navigation module and specifically the fusion of different sensors as a means of localization and navigation. The navigation module will consist of different sensors, a control module driven by software and a control interface to the motorized robot platform. With all this done the robot must be able to navigate between waypoints, travel to a specific waypoint and return to the base station if necessary, all of this being done autonomously while in a dynamic environment. The dynamic environment entails that there will be stationary objects such as the heliostats that all have known positions prior to the robot starting navigation.

1.2 Objectives

Because of the sheer size of a project such as developing an autonomous system, only the navigation of such a system will be investigated in this research. This will be done on scale since an existing robot platform is available and access to a full scale CSP plant is not possible. The objectives can be broken into a literature study that must be completed to gain an understanding of the mathematics driving the sensors, the hardware that must be purchased to localize the robot, the software that must be implemented to execute this and finally capturing the results after completion of all the integration.

1.2.1 Literature study

Before any hardware purchases can be made a study must be conducted to determine the following:

1. An understanding of the different GPS techniques used.
2. Ways in which accurate motor speed control over a varying speed range can be accomplished.
3. Ways to accomplish accurate motor acceleration and deceleration to avoid wheel slippage and to accomplish steering actions.
4. Different ways of fusing sensor data, for direct implementation in fusing sensor data and towards gaining an understanding behind the algorithms used in individual sensors.
5. The type of compass to be used in conjunction with encoders to obtain distance and attitude measurements for localization.
6. The theoretical accuracy of all sensors.

1.2.2 Hardware

The following list of objectives must be completed before any software can be implemented:

1. Define hardware that is currently present on the robot.
2. Purchase all additional sensors.
3. Purchase microcontroller and ensure compatibility between microcontroller and all sensors present.
4. Design and manufacture housings for sensors.
5. Analyse current platform to determine what modifications must be made to house the additional sensors.
6. Determine battery specification, battery life, robot power budget and battery discharge rate.

1.2.3 Software

To obtain experimental data the following software must be defined, developed and implemented:

1. Determine which software platforms will be used to navigate the robot, gather experimental data and show the final results.
2. Software for robot platform propulsion and steering.
3. Software to retrieve data from all attached sensors while simultaneously navigating the robot.
4. Software to fuse all sensor measurements.
5. Software to create a map of the environment containing known heliostat locations while having the ability to add known static obstacles.
6. Software that uses the fused data to determine the current robot position.
7. Software to successfully navigate from one waypoint to the next.
8. Software to capture and display all positional output data.

1.2.4 Experimental data

Once the robot platform and coding has been completed the following experiments will be conducted:

1. The viability of using the RTK GPS as a means of measuring precision of other sensors.
2. A comparison in relative navigational accuracy by navigating the same map multiple times.
3. A comparison of relative navigational accuracy when the complimentary filter values are altered.
4. A comparison of relative navigational accuracy when differential GPS (EGNOS (European Geostationary Navigation Overlay Service)) is enabled and disabled on the Adafruit GPS module.

5. A comparison of relative navigational accuracy when the ground speed of the robot is increased.
6. Relative navigational accuracy when wheel slippage is introduced.
7. Relative navigational accuracy when magnetic interference and GPS drift is present.
8. Relative navigational accuracy when one or multiple sensors fail.

1.3 Motivation

CSP systems generate electricity by converting the energy of the sun into heat, and then using this heat to generate electricity. A central tower power plant is a type of CSP system where electricity is generated by focusing sunlight from heliostats on a central tower receiver where a heat transfer medium (liquid or air) is used to generate steam which drives turbines, which in turn generates electricity. Thus the cleanliness of the mirrors on the heliostats has a direct impact on the efficiency of the power plant as a whole. A case study was conducted in Morocco where a drop of $45\,\%$ in reflectivity was recorded for mirrors mounted horizontally over a three month period (Merrouni et al., 2015). Therefore it is of critical importance to keep the mirrors clean to ensure the plant can maintain an optimum output level.

In central tower power plants the mirrors on individual heliostats are cleaned by machinery, operated by people. An AGV is an autonomous ground vehicle that can navigate through the power plant and execute tasks given to it while avoiding obstacles (permanent fixtures as well as temporary obstructions). Currently autonomous robots, which have no prior information about their environment, implement SLAM (Simultaneous Localization and Mapping) techniques as a way of navigation by building a map of the environment while localizing itself on the created map (Leonard et al., 1991). Because the heliostats' positions are known, a prior map of their coordinates can be created, thus the focus shifts towards the robot being able to localize itself rather than trying to navigate with no prior knowledge of its environment.

By implementing an AGV the operator(s) and the current machinery can be declared redundant, thus minimizing the risk of mirror breakage due to human error during operation of the machinery. A further advantage is the fact that in the overall operation of the power plant, less unnecessary human interaction, where repetitive tasks are present, is required, thus minimizing the operating effort placed on people. When using an AGV, the cleaning process can possibly be completed during night-time, thus not interrupting the power plant during daytime by obstructing the reflection of the mirrors while cleaning them. This allows the power plant to operate at full efficiency during daytime. People are also kept safe from a potential hazardous working environment, which includes heavy machinery, high ambient temperature and the reflection of the sun from the mirrors.

It must be realized that machines are more reliable than people in cases such as these where repetitive tasks are present. By implementing such a cleaning system, the efficiency of the whole power plant will be improved, which at the end of the day is the ultimate goal of any successful power plant.

2 Literature study

This section will give an explanation of how satellite navigation works, specific types of GPS technologies, PID controllers, the fusion of sensors using the complimentary and Kalman filters and finally the tilt compensated compass.

2.1 Satellite navigation

First an overview of how general satellite navigation works is given, followed by the more accurate differential GPS and finally proceeding to the highly accurate RTK GPS.

2.1.1 General satellite navigation

Satellite navigation gives an electronic receiver the ability to determine its time and location in terms of longitude, latitude and altitude using signals transmitted by satellites orbiting the earth. The receiver has to be in clear sight of at least four satellites, with more satellites resulting in a higher precision reading. When a satellite navigation system can give global coverage, it can be coined as a global navigation satellite system (GNSS). The GPS is the American GNSS and consists of up to 32 satellites orbiting earth at an altitude of $20\ 000\ \mathrm{km}$.

Every satellite has an atomic clock that is synchronized to each other and to stations on the ground. Each satellite continually transmits a signal containing the exact time the signal is sent from this satellite, and when the signal reaches the receiver the time it arrived is recorded. Thus the receiver knows how far it is from the satellite, based on the time the signal travelled. The exact position of the satellite in orbit is also known at the exact moment the signal is sent. Thus by using trilateration (the intersection of three spheres from three satellites) the position of the receiver can be calculated. The fourth satellite is necessary since the receiver does not have an accurate atomic clock. This is shown in equation 2.1 where the receivers' position $[x_r, y_r, z_r]$ is in the coordinate system shown in Figure 2-1. The true time the message was received is simply the difference between when the receiver says the message was received (t_r) and the difference between the receiver and satellite clocks (t_b), $t_t = t_r - t_b$. Thus t_b is the same for all received signals if the assumption is made that the satellites clocks are in sync. The distance the message travelled from the satellite to the receiver is denoted by $[t_r - t_b - t_i]\,c$. Thus the following equation can be solved for the four unknowns $[x_r, y_r, z_r, t_b]$ by using data from at least four satellites (n).

$$(x_r - x_i)^2 + (y_r - y_i)^2 + (z_r - z_i)^2 = ([t_r - t_b - t_i]\,c)^2\ ,\ i = 1,2,\dots,n \qquad (2.1)$$

This solution can be solved using either algebraic or numerical methods.

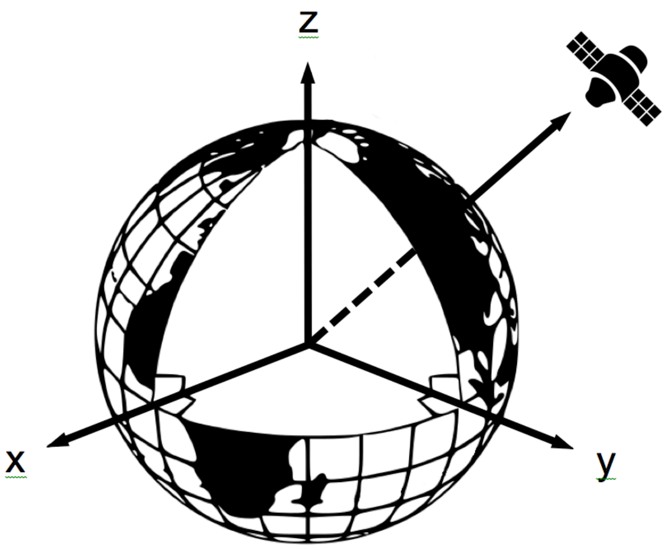


Figure 2-1: GPS coordinate system[1]

2.1.2 Differential GPS

Regular GPS will give a pseudo range accuracy of $7{,}8\ \mathrm{m}$ at a confidence level of $95\ \%$ (Gps.gov, 2015). For the purpose of AGV navigation this accuracy is not sufficient, therefore other GPS techniques will be investigated to improve GPS accuracy. Differential GPS works on the concept of having a base station at a highly accurate known location, that measures errors in signals from satellites and transmitting corrections to receivers (known as rovers) in close vicinity. These base stations also transmit the corrections to SBAS (satellite-based augmentation systems) that then in turn again broadcast the corrections to receivers. The EGNOS has four monitor stations in southern Africa, thus enabling corrections in this region (Merry, 2007). The measurements from the base station and rover are subtracted to get a differential measurement that is used to get a differential position. There are three sources of error when looking at GPS measurements. These are: ones originating on the satellite, ones on the path from the satellite to the receiver and ones where the signal is received. These errors, which can be removed by differential GPS, will be discussed.

Before the year 2000 the dominant error originating on the satellite was SA (Selective Availability). It is a technique that was implemented to provide a degraded signal on the L1 frequency ($1\ 575{,}42\ \mathrm{MHz}$) for non-military users, which created a random offset in the clock signal equivalent to $100\ \mathrm{m}$ of navigation accuracy (1996 Federal Radionavigation Plan, 1997). This has now been turned

[1] Adapted from U.S. Department of Transportation Federal Aviation Administration - Airway Facilities Division, 2013, https://en.wikipedia.org/wiki/Earth-centered_inertialAccessed

off but was the main reason differential GPS was implemented. Furthermore the orbit of the satellite and the difference between satellite clocks also have an effect on the error originating on the satellite. Because of uncertainties in the precise path of the satellite's orbit, the accuracy is $8\ \mathrm{m}$ on a $95\ \%$ interval (Varner, 2000). The satellite clock can be out by as much as $11\ \mathrm{ns}$ (Parkinson, 1996) resulting in an error of $3\ \mathrm{m}$.

The error generated while the signal travels from the satellite to the receiver can be separated into ionospheric, tropospheric and stratospheric delays. The ionosphere is a layer of the atmosphere between 70 and $1\,000\ \mathrm{km}$ with varying thickness that is dependent on the time of the day and solar activity. The angle at which the signal travels through the ionosphere has an effect on accuracy, between $10 - 30\ \mathrm{m}$ (Oc.nps.edu, 2015). The troposphere is from the earth's surface to $10\ \mathrm{km}$ above it and the stratosphere extends to a height of $50\ \mathrm{km}$. These spheres have different refractive indexes because of changes in the dryness of the atmosphere. This causes the GPS signal to travel at different speeds through the different layers resulting in inaccuracies in location measurements. A tropospheric model can be used to predict the effect of the dry atmospheric part which forms $90\ \%$ of the combined error between the trophoshere and stratosphere. If such a model is applied the remaining delay through the wet atmosphere is less than $300\ \mathrm{mm}$ (Li et al., 2014).

Errors generated at the receiver have been greatly reduced with new technologies. For example receivers with the ability to get L1 $(1\,575{,}42\ \mathrm{MHz})$ and L2 $(1\,227{,}60\ \mathrm{MHz})$ messages can eliminate ionospheric errors. Furthermore multipath errors – where the receiver sees reflected signals of surfaces as direct signals from the satellite – can cause errors of up to $15\ \mathrm{m}$ (Edu-observatory.org, 2015). By using an antenna that is less sensitive to low level signals this error can be minimized. Various algorithms can also be implemented to reduce this error and can be reduced to 1 to $2\ \mathrm{m}$ as shown by (Edu-observatory.org, 2015).

Table 2-1 gives a summary of the different error types common to all new receivers and their effect:

Table 2-1: Different error types

Description	Error with confidence level of $95\ \%$ (m)
Selective Availability	100
Satellite orbit	8
Satellite clock	3
Ionospheric	$10 - 30$
Troposphere and stratosphere modelled	$0{,}3$
Multipath	$1 - 2$

2.1.3 RTK GPS

RTK GPS is a differential global navigation satellite system that consists of a base station and one or more rovers with a direct line of communication between the base station and rovers in addition to their individual communication capabilities with the overhead satellites. RTK GPS can be used to a distance of 10 to $20\ \mathrm{km}$ (Rietdorf et al., 2006) with an accuracy of up to $30\ \mathrm{mm}$ horizontally (Roze et al., 2015). Because this is a differential technique it has all the benefits of a differential system, which include removal of satellite orbit errors, satellite clock errors, ionospheric delays, tropospheric delays and stratospheric delays. All these errors are varying as the rover moves around with the exception of the satellite clock error. There are three main errors that still need to be corrected; these are multipath, interference and thermal noise at the receiver.

As stated earlier there are two frequencies on which navigation messages are sent from the satellite to the receiver, these are L1 ($1\,575{,}42\ \mathrm{MHz}$ with a wavelength of $190\ \mathrm{mm}$) and L2 ($1\,227{,}60\ \mathrm{MHz}$ with a wavelength of $244\ \mathrm{mm}$). These navigation messages consists of two groups, coarse / acquisition (C/A) code and precision (P) code. The C/A code is available to everyone while the P code is encrypted and used for military purposes mainly, this encrypted signal is known as P(Y) code. The C/A code is at a frequency of $1{,}023\ \mathrm{MHz}$ (with a wavelength of $293{,}1\ \mathrm{m}$) over the L1 frequency, while the P(Y) code is sent at a frequency of $10{,}23\ \mathrm{MHz}$ (with a wavelength of $29{,}31\ \mathrm{m}$) over both the L1 and L2 frequencies. As discussed in Section 2.1.2 the ability to receive messages on both frequencies remove the ionospheric error.

Regular satellite navigation works on the principle that both the C/A and P codes are modulated onto the L band waves as a binary sequence being generated by a complicated algorithm. This is done on both the satellite and the receiver. Since the signal takes time to reach the receiver, the received message is delayed until the sequences match up, meaning that the required delay can be used to calculate the distance to the satellite. The difference with RTK is that the signal's wavelength is used as a signal instead of the signal itself as shown in Figure 2-2. This means that the C/A code at $1{,}023\ \mathrm{MHz}$ with a wavelength of $293{,}1\ \mathrm{m}$ can be disregarded and the L1 wave at a frequency of $1\,575{,}42\ \mathrm{MHz}$ and wavelength of $190\ \mathrm{mm}$ used instead as the source of distance calculation between the satellite and the receiver. The only problem is that the number of whole wavelengths on the L1 frequency is unknown, but this integer ambiguity can be resolved by comparing measurements from C/A codes that will lead to a consistent position solution as time passes and the satellites change position.

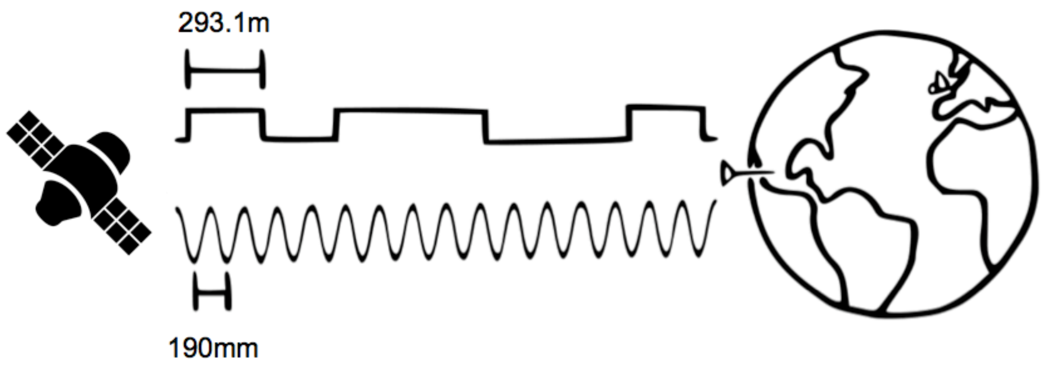


Figure 2-2: RTK GPS signal[2]

Because the RTK base station and rover are relatively close to one another the ionospheric error can also be reduced by assuming that the ionospheric delay will be common for both the base station and rover. Hence the difference (shown in Figure 2-3) can be calculated. When one has this difference value for four or more satellites one can calculate an ionospheric error free distance between the base station and rover.

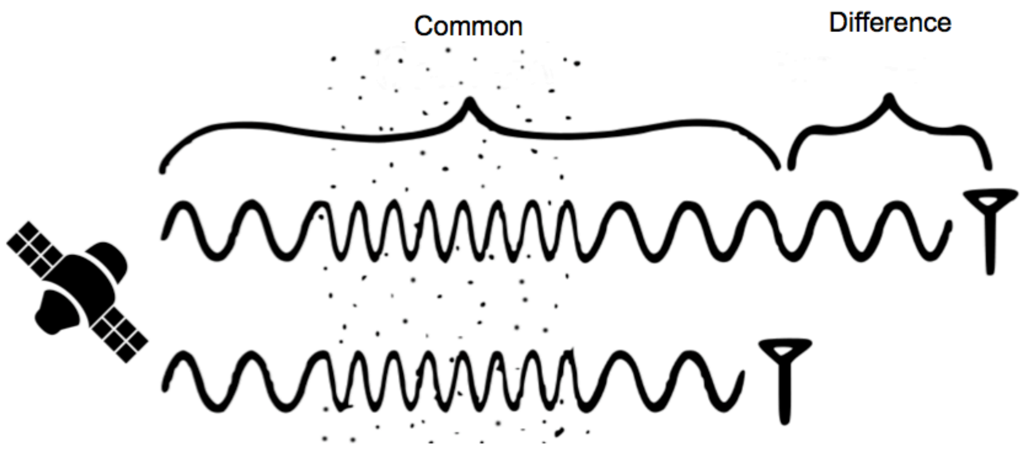


Figure 2-3: Ionospheric error[3]

[2] Adapted from Swiftnav, 2015, https://www.kickstarter.com/projects/swiftnav/piksi-the-rtk-gps-receiver/description

[3] Adapted from Swiftnav, 2015, https://www.kickstarter.com/projects/swiftnav/piksi-the-rtk-gps-receiver/description

A simplified version of the RTK algorithm will be shown to explain how the integer ambiguity problem is solved. The carrier phase measurement (γ) is a combination of the distance between the receiver and the satellite (ρ), ionospheric distance error (I), tropospheric distance error (T), receiver offset distance error $(c \cdot t_{dt})$, satellite offset distance error $(c \cdot t_{dT})$, the integer ambiguity (N) and noise (ε) and can be equated as follows:

$$\gamma = \rho - I + T + c\,(t_{dt} - t_{dT}) + N\,\lambda + \varepsilon \tag{2.2}$$

Where $\rho = \sqrt{(x_i - x_r)^2 + (y_i - y_r)^2 + (z_i - z_r)^2}$

Getting the difference in the carrier phase measurement between two satellites (1) and (2) and between the base station (b) and receiver (r) results in equation 2.3. Because the base station and receiver has a common clock and uses the same set of satellites the receiver and satellite clock errors can be eliminated. This difference can be written as:

$$\gamma_r^{12} - \gamma_b^{12} = \rho_r^{12} - \rho_b^{12} - I_r^{12} + I_b^{12} + T_r^{12} - T_b^{12} + \lambda\,(N_r^{12} - N_b^{12}) + \varepsilon_r^{12} - \varepsilon_b^{12} \tag{2.3}$$

Since the non-integer terms – the receiver and satellite clock error – have been eliminated, the integer ambiguity $(N_r^{12} - N_b^{12})$ is an integer value and can now be used to calculate the whole number of wavelengths as mentioned earlier if data from sufficient satellites is available.

2.2 PID controllers

A PID (proportional-integral-derivative) controller attempts to minimize the difference between the plant output and a desired setpoint. A PID controller (Figure 2-4) has three parts; they are proportional (K_p), integral (K_i) and derivative (K_d). Each one of these elements is controlled by the feedback signal from the plant or system that is being controlled. These elements are then added in a weighted manner as shown in equation 2.4. With the updated coefficients the plant is run again and the new output compared to the setpoint again, this process keeps on repeating to reduce the error to a value as small as possible.

$$u(t) = K_p\, e(t) + K_i \int_0^t e(\tau)\, d\tau + K_d\, \frac{de}{dt} \tag{2.4}$$

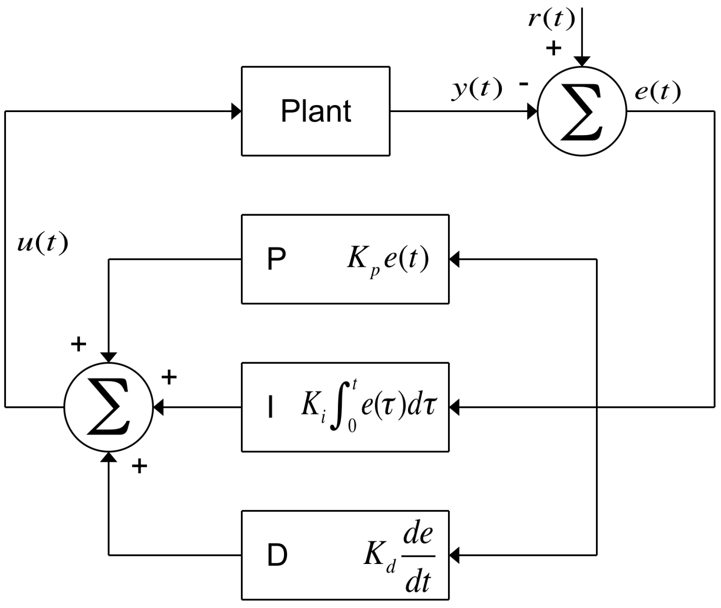


Figure 2-4: PID loop

The proportional part is simply the system error (e) multiplied by some constant and fed back into the system. Generally an increase in the proportional part will make the response of the system faster, and by increasing it too much the system output will start oscillating. The proportional part should contribute the most to a change in the system output (Kiran et al., 2014).

The integral part takes the effect of the error over the whole time the plant has been active into account. Thus this term keeps on increasing over time unless the system error is zero, so to minimize this part one must decrease the error and the time over which the error is accumulated. The integral part has an accelerating effect on the system, trying to get the system to the setpoint as fast as possible, but because it accumulates the error it has a tendency to overshoot the setpoint if the integral gain coefficient is too high (Åström et al., 2008).

The derivative part tries to decrease the settling time as well as stability for the system as discussed in (Wescott, 2000) by looking at the rate of change of the error. This part also decreases the overshoot introduced by the integral part but the trade-off here is that the system becomes more sensitive to noise.

For the system to give the desired response one must optimally tune the proportional, integral and derivative coefficients. The system must be stable, not oscillate and achieve this state in the shortest amount of time possible, and adjust accordingly if the setpoint is changed. To tune the loop there are various techniques that can be used, including the manual tuning method and the Ziegler Nichols method.

The manual tuning method is a trial and error method where the integral and differential terms are first set to zero. Next the proportional term is increased until the system output starts to oscillate. The larger the proportional term the faster the system starts responding but this also makes the system less prone to being stable. Now the integral term can be increased to stop the system output from oscillating. Because this term accumulates the system error the amount of overshoot will increase but this is necessary to help the system respond to changes without much delay. The term is adjusted to make the difference between the system output and the setpoint a minimum. Finally the derivative term is increased until the settling time of the system is within an acceptable range. Table 2-2 shows a summary for what effect increasing each of the tuning terms has on various aspects of the system output.

Table 2-2: Effect of tuning PID parameters

Response	Rise Time	Overshoot	Settling Time	Steady-state Error
K_p	Decrease	Increase	No trend	Decrease
K_i	Decrease	Increase	Increase	Eliminate
K_d	No trend	Decrease	Decrease	No trend

The Ziegler Nichols method is quite similar to the manual tuning method in that the integral and differential parts are first set to zero and the proportional part increased until the system output starts to oscillate. Whenever this state is achieved the period of the oscillations (P_c) and gain at this critical point (K_c) is taken and the proportional, integral and differential terms are then adjusted according to optimum settings suggested in (Ziegler et al., 1993), shown in Table 2-3.

Table 2-3: Ziegler-Nichols optimal tuning parameters

Control type	K_p	K_i	K_d
P	$0.50\ K_c$	-	-
PI	$0.45\ K_c$	$\frac{1.2}{P_c}$	-
PID	$0.60\ K_c$	$\frac{2.0}{P_c}$	$\frac{P_c}{8}$

2.3 *Sensor fusion*

When receiving data from multiple sensors regarding the same attribute (for the example the position of the robot) one can fuse the data using algorithms to achieve a result that is more accurate than when the sensors are used individually. Different sensors have different attributes, meaning they are either more reliable on the long run or on the short run. This is where the filters are supposed to extract the most trustworthy information from the individual sensors and fuse this new information. Two main filters will be discussed; they are the complimentary filter and the Kalman filter.

2.3.1 Complimentary filter

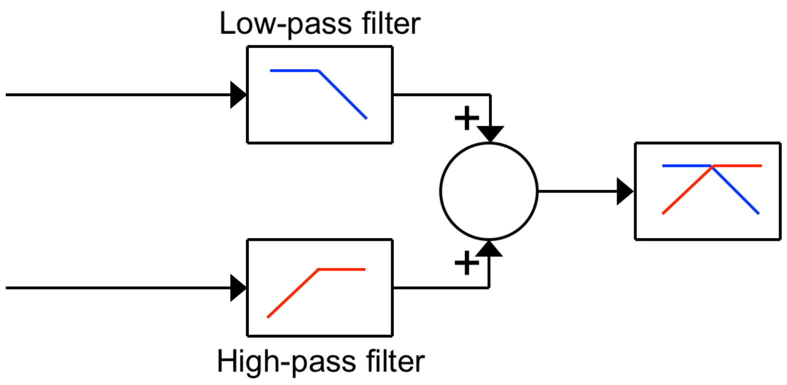


Figure 2-5: Complimentary filter[4]

The complimentary filter (Figure 2-5) is a simple filter that has a digital high-pass and a digital low-pass filter. The low-pass filter only lets through changes that have an effect on the long-term while short-term changes are filtered out. The high-pass filter does the exact opposite of the low-pass filter, reducing long term drift in the output. For this application the GPS data is more reliable on the long run than the odometry data, since the odometry data introduces drift. Thus the GPS data is fed through the low-pass filter while the odometry data is fed through the high-pass filter, resulting in the output highly relying on the odometry data while being corrected over the long run using the GPS data.

[4] Adapted from Socialledge, 2013, http://www.socialledge.com/sjsu/index.php?title=F13:_Quad-copter

2.3.2 Kalman filter

The Kalman filter (Kalman, 1960) is a mathematical algorithm that makes a prediction of a value at some time by taking measurements over a time interval and then receiving feedback from the system as noisy measurements. There are thus two stages, a prediction and a measurement stage. The prediction stage predicts the state of the system in the future while the measurement step improves the prediction once data for that specific epoch is available. The state vector ($\bar{x}_k$) storing the model system parameters is a combination of the state vector from the previous epoch ($\bar{x}_{k-1}$), inputs to the system ($\bar{u}_k$) and process noise ($\bar{w}_{k-1}$). This is shown in the following equation:

$$\bar{x}_k = \bar{A}\,\bar{x}_{k-1} + \bar{B}\,\bar{u}_k + \bar{w}_{k-1} \tag{2.5}$$

The sensor measurement vector ($\bar{z}_k$) can be modelled by the state vector ($\bar{x}_k$) that is converted into the measurement domain by a transformation matrix ($\bar{H}$) with added measurement noise ($\bar{v}_k$).

$$\bar{z}_k = \bar{H}\,\bar{x}_k + \bar{v}_k \tag{2.6}$$

Once one has fitted the model into the Kalman filter one can start to estimate the parameters iteratively using equations 2.7 to 2.11. Assuming no noise a prediction of the state vector can be made:

$$\bar{x}_k^- = \bar{A}\,\bar{x}_{k-1} + \bar{B}\,\bar{u}_k \tag{2.7}$$

The error covariance matrix that will be used in the measurement update can be predicted to be:

$$\bar{P}_k^- = \bar{A}\,\bar{P}_{k-1}\,\bar{A}^T + \bar{Q} \tag{2.8}$$

The Kalman gain can be computed as:

$$\bar{K}_k = \frac{\bar{P}_k^-\,\bar{H}^T}{\bar{H}\,\bar{P}_k^-\,\bar{H}^T + \bar{R}} \tag{2.9}$$

Now the prediction made in equation 2.7 is updated:

$$\bar{x}_k = \bar{x}_k^- + \bar{K}_k\,(\bar{z}_k - \bar{H}\,\bar{x}_k^-) \tag{2.10}$$

And also the error covariance matrix of 2.8:

$$\bar{P}_k = (\bar{I} - \bar{K}_k\,\bar{H})\,\bar{P}_k^- \tag{2.11}$$

Equations 2.7 and 2.8 show the prediction stage, while equations 2.9 to 2.11 show the measurement stage. These two stages keep on repeating by first predicting the current state ahead of time and then adjusting this prediction through the measurement stage after which the cycle repeats itself. The big advantage of this approach is that only data from the current and previous epoch needs to be kept and the filter is recursive in a way that doesn't add complexity to the problem.

A simple one-dimensional example will be considered to show how the Kalman filter can be derived. Consider a point moving along a straight line where some input is given to move the point and the distance from the starting location to the current point position can be measured. The system parameters that are of interest are the point position and point velocity. Since the system parameters will have a measure of uncertainty it can be assumed that they will be Gaussian distributed. Initially the state of the system is known with a high certainty as shown in Figure 2-6.

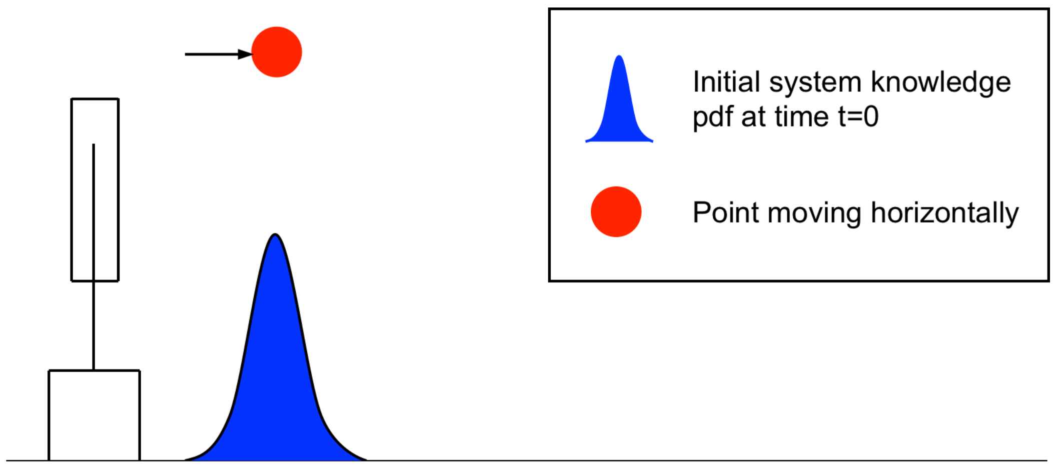

Initial system knowledge pdf at time t=0
Point moving horizontally

Figure 2-6: Initial system state

After a certain time interval the system state distribution will have a new mean (μ) and variance (σ^2) as shown in Figure 2-7. Because the input that caused the system to move to the current position is known, an estimate can be made regarding the current position of the point. But because this is only a prediction the system parameters will have a new mean and variance as shown in Figure 2-7. Equation 2.5 shows mathematically what happens between Figure 2-6 and Figure 2-7, and the variance is represented by equation 2.6. The change in variance is due to the process noise introduced by equation 2.5.

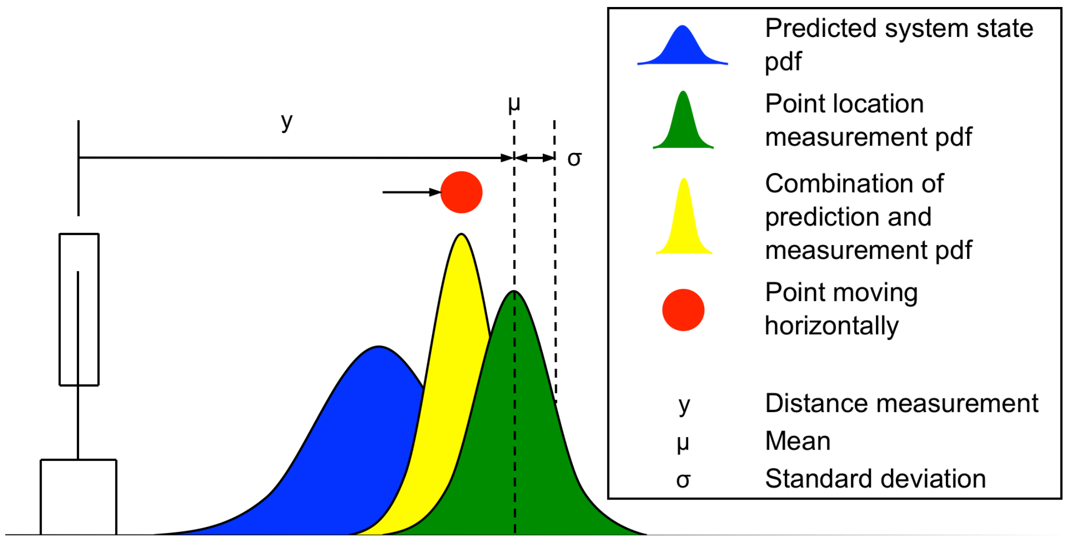

y
μ
σ
Predicted system state pdf
Point location measurement pdf
Combination of prediction and measurement pdf
Point moving horizontally
y Distance measurement
μ Mean
σ Standard deviation

Figure 2-7: System state after time interval

At the exact same time the prediction regarding the current position of the system is made, a measurement can also be taken as seen in Figure 2-7. The best estimate of the current position of the point can be made by combining the two Gaussian distributions in Figure 2-7 to form a new estimate shown by the yellow part. This new estimate is a multiplication between the two Gaussian distributions described above. It is known that the multiplication of two Gaussian functions yields another Gaussian function, as shown in (Bromiley, 2014). This is why the Kalman filter can be applied continually without increasing the complexity of the function, and will be illustrated by multiplying equations 2.12 and 2.13. The blue pdf (probability density function) can be given by equation 2.12:

$$f_1(y) = \frac{1}{\sigma_1\sqrt{2\,\pi}}\, e^{\frac{-(y-\mu_1)^2}{2\,\sigma_1{}^2}} \tag{2.12}$$

The green measurement Gaussian function can be given by equation 2.13:

$$f_2(y) = \frac{1}{\sigma_2\sqrt{2\,\pi}}\, e^{\frac{-(y-\mu_2)^2}{2\,\sigma_2{}^2}} \tag{2.13}$$

Multiplying equations 2.12 & 2.13 results in:

$$\begin{aligned} f_3(y) &= f_1(y)\, f_2(y) \\ &= \frac{1}{\sigma_1\sqrt{2\,\pi}}\, e^{\frac{-(y-\mu_1)^2}{2\,\sigma_1{}^2}}\, \frac{1}{\sigma_2\sqrt{2\,\pi}}\, e^{\frac{-(y-\mu_2)^2}{2\,\sigma_2{}^2}} \\ &= \frac{1}{2\,\pi\,\sigma_1\,\sigma_2}\, e^{-\left(\frac{(y-\mu_1)^2}{2\,\sigma_1{}^2}+\frac{(y-\mu_2)^2}{2\,\sigma_2{}^2}\right)} \end{aligned} \tag{2.14}$$

It is known that the combined standard deviation of the two Gaussian pdf's is as shown by (Bromiley, 2014) in equation 2.15:

$$\sigma_3 = \sqrt{\frac{{\sigma_1}^2\,{\sigma_2}^2}{{\sigma_1}^2 + {\sigma_2}^2}}$$
$$\therefore {\sigma_3}^2 = \frac{{\sigma_1}^2\,({\sigma_1}^2 + {\sigma_2}^2) - {\sigma_1}^4}{{\sigma_1}^2 + {\sigma_2}^2} \tag{2.15}$$
$$= {\sigma_1}^2 - \frac{{\sigma_1}^4}{{\sigma_1}^2 + {\sigma_2}^2}$$

And the combined mean of the two Gaussian pdf's is as shown by (Bromiley, 2014) in equation 2.16:

$$\mu_3 = \frac{\mu_1\,{\sigma_2}^2 + \mu_2\,{\sigma_1}^2}{{\sigma_1}^2 + {\sigma_2}^2}$$
$$= \frac{{\sigma_1}^2\,(\mu_2 - \mu_1) + \mu_1\,{\sigma_1}^2 + \mu_1\,{\sigma_2}^2}{{\sigma_1}^2 + {\sigma_2}^2} \tag{2.16}$$
$$= \mu_1 + \frac{{\sigma_1}^2\,(\mu_2 - \mu_1)}{{\sigma_1}^2 + {\sigma_2}^2}$$

Equation 2.14 shows that the multiplication of two pdf's is just another pdf, thus the new Gaussian pdf can be described by equation 2.17 with a standard deviation and mean as shown by equations 2.15 & 2.16.

$$f_3(y) = \frac{1}{\sigma_3\,\sqrt{2\,\pi}}\; e^{\frac{-\,(y-\mu_3)^2}{2\,{\sigma_3}^2}} \tag{2.17}$$

Equation 2.17 assumes that both the prediction and measurement stages are conducted within the same coordinate type. In the example used here the prediction stage occurs in the distance domain while the measurement stage occurs in the time domain. To get the correct fused Gaussian pdf one should convert one of these domains to the other, the standard way of accomplishing this is to convert the prediction domain into that of the measurement domain. This is the reason for the transformation matrix $\overline{H}$ as shown in equation 2.6. Thus converting equations 2.12 and 2.13 into the time domain results in:

$$f_1(y) = \frac{1}{\frac{\sigma_1}{c}\,\sqrt{2\,\pi}}\; e^{\frac{-\,(t-\frac{\mu_1}{c})^2}{2\left(\frac{\sigma_1}{c}\right)^2}} \tag{2.18}$$

And:

$$f_2(y) = \frac{1}{\sigma_2\,\sqrt{2\,\pi}}\; e^{\frac{-\,(t-\mu_2)^2}{2\,{\sigma_2}^2}} \tag{2.19}$$

Using these new functions as input for the new mean equation one obtains:

$$\begin{aligned} \frac{\mu_3}{c} &= \frac{\mu_1}{c} + \frac{\left(\frac{\sigma_1}{c}\right)^2 \left(\mu_2 - \frac{\mu_1}{c}\right)}{\left(\frac{\sigma_1}{c}\right)^2 + {\sigma_2}^2} \\ \therefore \mu_3 &= \mu_1 + \left(\frac{\frac{{\sigma_1}^2}{c}}{\left(\frac{\sigma_1}{c}\right)^2 + {\sigma_2}^2}\right)\left(\mu_2 - \frac{\mu_1}{c}\right) \end{aligned} \tag{2.20}$$

And by substituting $\overline{H} = \frac{1}{c}$ and $\overline{K}_k = \frac{\frac{{\sigma_1}^2}{c}}{\left(\frac{\sigma_1}{c}\right)^2 + {\sigma_2}^2}$ one can obtain:

$$\mu_3 = \mu_1 + \overline{K}_k \left(\mu_2 - \overline{H}\,\mu_1\right) \tag{2.21}$$

In a similar fashion for the standard deviation one obtains:

$$\begin{aligned} \frac{{\sigma_3}^2}{c^2} &= \left(\frac{\sigma_1}{c}\right)^2 - \frac{\left(\frac{\sigma_1}{c}\right)^4}{\left(\frac{\sigma_1}{c}\right)^2 + {\sigma_2}^2} \\ \therefore {\sigma_3}^2 &= {\sigma_1}^2 - \frac{\frac{{\sigma_1}^4}{c^2}}{\left(\frac{\sigma_1}{c}\right)^2 + {\sigma_2}^2} \\ &= {\sigma_1}^2 - \frac{{\sigma_1}^2}{c}\,\frac{\frac{{\sigma_1}^2}{c}}{\left(\frac{\sigma_1}{c}\right)^2 + {\sigma_2}^2} \end{aligned} \tag{2.22}$$

Again substituting for $\overline{H} = \frac{1}{c}$ and $\overline{K}_k = \frac{\frac{{\sigma_1}^2}{c}}{\left(\frac{\sigma_1}{c}\right)^2 + {\sigma_2}^2}$ one obtains:

$${\sigma_3}^2 = {\sigma_1}^2 - \overline{H}\,\overline{K}_k\,{\sigma_1}^2 \tag{2.23}$$

One can now compare the standard form of the Kalman filter as shown in equations 2.7 to 2.11 with the terms in equations 2.21 and 2.23 to see the following correlations:

μ_3	$\rightarrow$	$\bar{x}_k$
μ_1	$\rightarrow$	$\bar{x}_k^-$
μ_2	$\rightarrow$	$\bar{z}_k$
${\sigma_3}^2$	$\rightarrow$	$\overline{P}_k$
${\sigma_1}^2$	$\rightarrow$	$\overline{P}_k^-$
${\sigma_2}^2$	$\rightarrow$	$\overline{R}$
$\overline{H}$	$\rightarrow$	$\overline{H}$
$\overline{K}_k$	$\rightarrow$	$\overline{K}_k$

All variable descriptions are as described in the nomenclature. Placing these new values into equations 2.21 and 2.23 one can easily see how it relates to the standard Kalman filter equations 2.10 and 2.11 as an example:

$$\mu_3 = \mu_1 + \frac{\bar{H}\,{\sigma_1}^2}{\bar{H}^2\,{\sigma_1}^2 + {\sigma_2}^2}\,(\mu_2 - \bar{H}\,\mu_1) \tag{2.24}$$

$$\bar{x}_k = \bar{x}_k^- + \bar{K}_k\,(\bar{z}_k - \bar{H}\,\bar{x}_k^-) \tag{2.25}$$

$${\sigma_3}^2 = {\sigma_1}^2 - \bar{H}\,\frac{\bar{H}\,{\sigma_1}^2}{\bar{H}^2\,{\sigma_1}^2 + {\sigma_2}^2}\,{\sigma_1}^2 \tag{2.26}$$

$$\bar{P}_k = (\bar{I} - \bar{K}_k\,\bar{H})\,\bar{P}_k^- \tag{2.27}$$

By understanding the core mathematical principles behind the Kalman filter one can easily derive it as was done here.

2.3.3 Filter comparison

From literature (Higgins, 1975) it has been shown through two examples that the filter equations for both the complementary and Kalman filters are identical. Because the complementary filter does not calculate gains and measurement and time updates as the Kalman filter, it is less computationally expensive and thus ideal for the Arduino 8-bit microcontroller. Real world tests have been done to show a comparison between these two filters and according to (Letsmakerobots.com, 2016) it was concluded that the Kalman filter can be replaced by the simpler and faster complementary filter while obtaining the same results.

2.4 Tilt compensated compass

This section will discuss the method used to calculate the current heading of the tilt compensated compass. Details regarding accelerometer and magnetometer calibration are also given. Calibration is the process in which sensor outputs are compared to known reliable reference information and then the output is adjusted by coefficients to let it agree with the reference information (Artese et al., 2008).

2.4.1 Current heading

A 3-axis magnetometer (fixed in the XYZ coordinate frame) can be used to measure the earth's magnetic field and in doing so determine a magnetic field vector that points towards the magnetic north pole. When one is near the equator this vector is parallel to the surface of the earth but changes as one's latitude changes. Shown in Figure 2-8 is this magnetic field vector ($\overline{M}$) with an inclination angle (β) between the XY-plane and this vector.

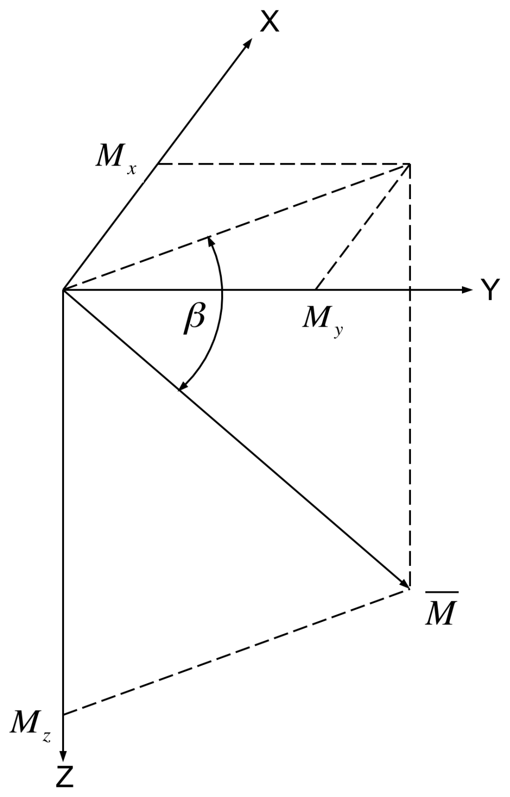


Figure 2-8: Magnetic field vector[5]

[5] Adapted from Coordinates, 2015, http://mycoordinates.org/operation-and-implementation-of-heading-reference-system/all/1/

To determine the current heading one can measure the two XY-plane components of the magnetic field vector, M_x and M_y, and then determine the angle ω (current heading) as shown in equation 2.28.

$$\omega = \tan^{-1}\left(\frac{M_y}{M_x}\right) \tag{2.28}$$

This equation is however only true as long as the magnetometer is held level, when the sensor is tilted the XY-plane components of the vector change, resulting in an erroneous heading. This effect can be countered by incorporating a 2-axis accelerometer that measures the angles between the tilted orientation of the magnetometer ($X'Y'Z'$ frame in Figure 2-9) and the gravity vector that is located in the XYZ coordinate frame. The measured M_x, M_y and M_z components are then converted from the $X'Y'Z'$ frame to the XYZ frame and the heading calculated using equation 2.28.

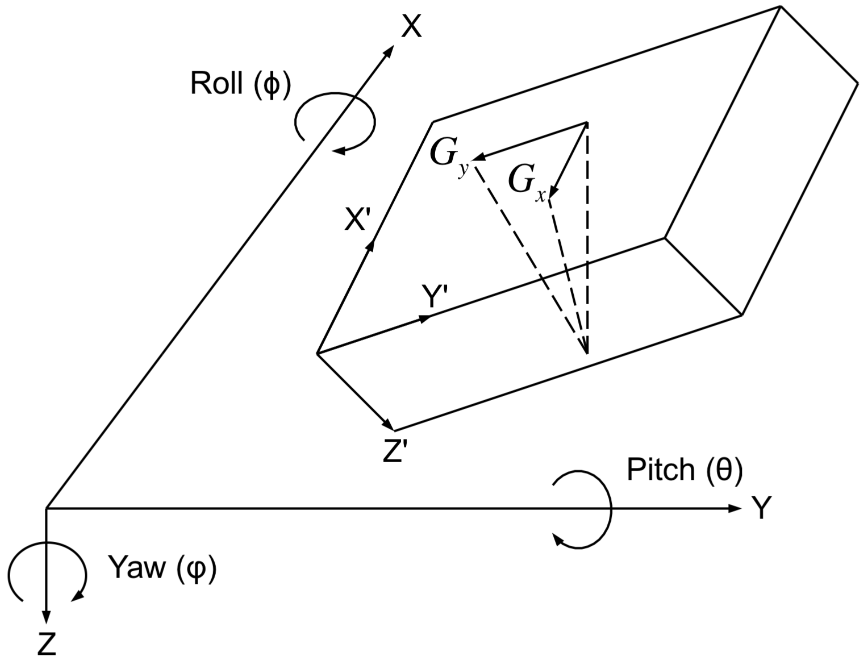


Figure 2-9: Conversion between coordinate frames[6]

The rotation of the coordinate frame will be described using Euler angles where roll (ϕ-rotation around the X-axis) and pitch (θ-rotation around the Y-axis) are

[6] Adapted from ST Microelectronics, 2015, http://www.st.com/st-web-ui/static/active/cn/resource/technical/document/application_note/DM00119044.pdf

present (as shown in Figure 2-9). Yaw (rotation about the Z-axis) is omitted because it's the heading and does not have an effect on the XY-plane tilt. Rotations around the axes follow the right-hand rule, and are always performed around the X-axis first and then around the Y-axis. From (Greenwood, 2003) it can be seen that the roll rotation matrix from the XYZ frame to the $X'Y'Z'$ frame is given by:

$$\bar{R}_r = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & \sin\phi \\ 0 & -\sin\phi & \cos\phi \end{pmatrix} \tag{2.29}$$

And the pitch rotation matrix is given by:

$$\bar{R}_p = \begin{pmatrix} \cos\theta & 0 & -\sin\theta \\ 0 & 1 & 0 \\ \sin\theta & 0 & \cos\theta \end{pmatrix} \tag{2.30}$$

Thus the overall transformation of the magnetic field vector from the XYZ frame to the $X'Y'Z'$ frame can be given by:

$$\begin{aligned} \begin{pmatrix} M_x{}' \\ M_y{}' \\ M_z{}' \end{pmatrix} &= \bar{R}_r\,\bar{R}_p \begin{pmatrix} M_x \\ M_y \\ M_z \end{pmatrix} \\ &= \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & \sin\phi \\ 0 & -\sin\phi & \cos\phi \end{pmatrix} \begin{pmatrix} \cos\theta & 0 & -\sin\theta \\ 0 & 1 & 0 \\ \sin\theta & 0 & \cos\theta \end{pmatrix} \begin{pmatrix} M_x \\ M_y \\ M_z \end{pmatrix} \\ &= \begin{pmatrix} \cos\theta & 0 & -\sin\theta \\ \sin\phi\,\sin\theta & \cos\phi & \sin\phi\,\cos\theta \\ \cos\phi\,\sin\theta & -\sin\phi & \cos\phi\,\cos\theta \end{pmatrix} \begin{pmatrix} M_x \\ M_y \\ M_z \end{pmatrix} \end{aligned} \tag{2.31}$$

To get the heading one must invert the matrix in equation 2.31 to obtain the inverses $\bar{R}_r{}^{-1}$ and $\bar{R}_p{}^{-1}$:

$$\bar{R}_r{}^{-1} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi & \cos\phi \end{pmatrix} \tag{2.32}$$

$$\bar{R}_p{}^{-1} = \begin{pmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{pmatrix} \tag{2.33}$$

Then one can get the inverted matrix:

$$\begin{pmatrix} M_x \\ M_y \\ M_z \end{pmatrix} = \bar{R}_p{}^{-1}\,\bar{R}_r{}^{-1} \begin{pmatrix} M_x{}' \\ M_y{}' \\ M_z{}' \end{pmatrix} \tag{2.34}$$

$$= \begin{pmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi & \cos\phi \end{pmatrix} \begin{pmatrix} M_x{}' \\ M_y{}' \\ M_z{}' \end{pmatrix}$$

$$= \begin{pmatrix} \cos\theta & \sin\phi\ \sin\theta & \cos\phi\ \sin\theta \\ 0 & \cos\phi & -\sin\phi \\ -\sin\theta & \sin\phi\ \cos\theta & \cos\phi\ \cos\theta \end{pmatrix} \begin{pmatrix} M_x{}' \\ M_y{}' \\ M_z{}' \end{pmatrix}$$

From equation 2.34 one can get:

$$M_x = M_x{}'\ \cos\theta + M_y{}'\ \sin\phi\ \sin\theta + M_z{}'\ \cos\phi\ \sin\theta \tag{2.35}$$

$$M_y = M_y{}'\ \cos\phi - M_z{}'\ \sin\phi \tag{2.36}$$

These two values from equations 2.35 and 2.36 represent the tilt-compensated values. But to get these one must first get the roll (ϕ) and pitch (θ) angles by using the accelerometer.

The accelerometer outputs are correlated to the gravitational vector shown in Figure 2-9 as G_y and G_x. To get the normalized accelerometer outputs $G_x{}'$ and $G_y{}'$ one can use equation 2.31 with gravitational acceleration only in the z-direction to obtain the following:

$$\begin{pmatrix} G_x{}' \\ G_y{}' \\ G_z{}' \end{pmatrix} = \begin{pmatrix} \cos\theta & 0 & -\sin\theta \\ \sin\phi\ \sin\theta & \cos\phi & \sin\phi\ \cos\theta \\ \cos\phi\ \sin\theta & -\sin\phi & \cos\phi\ \cos\theta \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \tag{2.37}$$

Thus:

$$G_x{}' = -\sin\theta \tag{2.38}$$

$$G_y{}' = \sin\phi\ \cos\theta \tag{2.39}$$

Using basic trigonometry identities one can say the following:

From equation 2.38:

$$\sin\theta = -G_x{}' \tag{2.40}$$

And:

$$\begin{aligned} &\sin^2\theta + \cos^2\theta = 1 \\ &\therefore \cos\theta = \sqrt{1-\sin^2\theta} \\ &\qquad\quad = \sqrt{1-G_x{}'^2} \end{aligned} \tag{2.41}$$

From equation 2.39:

$$\begin{aligned}\sin\phi &= \frac{G_y{}'}{\cos\theta}\\ &= \frac{G_y{}'}{\sqrt{1-G_x{}'^2}}\end{aligned} \tag{2.42}$$

And:

$$\begin{aligned}&\sin^2\phi+\cos^2\phi = 1\\ \therefore\ &\cos\phi = \sqrt{1-\sin^2\phi}\\ &\quad= \sqrt{1-\frac{G_y{}'^2}{1-G_x{}'^2}}\\ &\quad= \sqrt{\frac{1-G_x{}'^2-G_y{}'^2}{1-G_x{}'^2}}\end{aligned} \tag{2.43}$$

When looking at equation 2.28 one can see that to find the heading the only interest is the ratio between the two magnetic field vectors, thus equations 2.35 and 2.36 can be multiplied by the same value without having an effect on the heading. Multiply equations 2.35 and 2.36 by $\cos\theta$ to remove division since most microcontrollers do not have a native instruction for it. As an example it takes six times longer to do type long division than it takes to do type long addition on an Arduino 8-bit microcontroller (Learn.sparkfun.com, 2015). Thus the tilt compensation equations can be given as:

$$\begin{aligned}M_x &= M_x{}'\ \cos^2\theta + M_y{}'\ \sin\phi\ \sin\theta\ \cos\theta + M_z{}'\ \cos\phi\ \sin\theta\ \cos\theta\\ &= M_x{}'\ (1-G_x{}'^2) - M_y{}'\ \frac{G_y{}'}{\sqrt{1-G_x{}'^2}}\ G_x\sqrt{1-G_x{}'^2}\\ &\qquad - M_z{}'\sqrt{\frac{1-G_x{}'^2-G_y{}'^2}{1-G_x{}'^2}}\ G_x{}'\sqrt{1-G_x{}'^2}\\ &= M_x{}'\ (1-G_x{}'^2) - M_y{}'\ G_y{}'\ G_x{}' - M_z{}'\sqrt{1-G_x{}'^2-G_y{}'^2}\ G_x{}'\end{aligned} \tag{2.44}$$

$$\begin{aligned}M_y &= M_y{}'\ \cos\phi\ \cos\theta - M_z{}'\ \sin\phi\ \cos\theta\\ &= M_y{}'\sqrt{\frac{1-G_x{}'^2-G_y{}'^2}{1-G_x{}'^2}}\sqrt{1-G_x{}'^2} - M_z{}'\ \frac{G_y{}'}{\sqrt{1-G_x{}'^2}}\sqrt{1-G_x{}'^2}\\ &= M_y{}'\sqrt{1-G_x{}'^2-G_y{}'^2} - M_z{}'\ G_y{}'\end{aligned} \tag{2.45}$$

By then placing equations 2.44 and 2.45 back into equation 2.28 one can get the current heading. To get the true heading as a clockwise angle between magnetic north and the X-axis as shown in Figure 2-8 one can apply the following to adjust for different signs of x and y.

$$True\ heading = \begin{cases} 180 - \tan^{-1}\dfrac{M_y}{M_x} & M_x < 0 \\ -\tan^{-1}\dfrac{M_y}{M_x} & M_x > 0, M_y < 0 \\ 360 - \tan^{-1}\dfrac{M_y}{M_x} & M_x > 0, M_y > 0 \\ 90 & M_x = 0, M_y < 0 \\ 270 & M_x = 0, M_y > 0 \end{cases} \tag{2.46}$$

2.4.2 Accelerometer calibration

The gravitational vector that is given by the accelerometer has three main sources of error that has an effect on the accuracy of the roll and pitch values obtained, these are: offset, scale factor and misalignment. Offset refers to the sensor measurement when no gravity or motion is acting on it. Scale factor is the ratio between the electrical output of the accelerometer and the acceleration input. The misalignment error shows the difference between axes on which the accelerometer is mounted and the axes of the body on which the accelerometer is mounted. To transform the raw accelerometer data $(G_{x,raw}, G_{y,raw}, G_{z,raw})$ to normalized data $(G_x{}', G_y{}', G_z{}')$ on the $X'Y'Z'$ frame (same axis system as in Figure 2-9) one can use the misalignment matrix $(\bar{G}_m)$, scale factor $(\bar{G}_s)$ and offset $(\bar{G}_o)$ as used in (Parameters and calibration of a low-g 3-axis accelerometer, 2014).

$$\begin{pmatrix} G_x{}' \\ G_y{}' \\ G_z{}' \end{pmatrix} = \begin{pmatrix} G_{m11} & G_{m12} & G_{m13} \\ G_{m21} & G_{m22} & G_{m23} \\ G_{m31} & G_{m32} & G_{m33} \end{pmatrix} \begin{pmatrix} \dfrac{1}{G_{sx}} & 0 & 0 \\ 0 & \dfrac{1}{G_{sy}} & 0 \\ 0 & 0 & \dfrac{1}{G_{sz}} \end{pmatrix} \begin{pmatrix} G_{x,raw} - G_{ox} \\ G_{y,raw} - G_{oy} \\ G_{z,raw} - G_{oz} \end{pmatrix} \tag{2.47}$$

Equation 2.47 can be multiplied out to obtain the following where $\bar{G}_p$ is just the combination of the matrices used in equation 2.47:

$$\begin{pmatrix} G_x{}' \\ G_y{}' \\ G_z{}' \end{pmatrix} = \begin{pmatrix} G_{p11} & G_{p12} & G_{p13} \\ G_{p21} & G_{p22} & G_{p23} \\ G_{p31} & G_{p32} & G_{p33} \end{pmatrix} \begin{pmatrix} G_{x,raw} \\ G_{y,raw} \\ G_{z,raw} \end{pmatrix} + \begin{pmatrix} G_{p10} \\ G_{p20} \\ G_{p30} \end{pmatrix} \tag{2.48}$$

To solve equation 2.48 one can take six stationary positions (with coordinate frame as in Figure 2-9) as shown inTable 2-4.

Table 2-4: Accelerometer orientation

Stationary position	$G_x{}'$	$G_y{}'$	$G_z{}'$
X down	$+1g$	0	0
X up	$-1g$	0	0
Y down	0	$+1g$	0
Y up	0	$-1g$	0
Z down	0	0	$+1g$
Z up	0	0	$-1g$

Before equation 2.48 is solved a simplification is made as shown in equation 2.49. This is done to allow the use of the least squared approach that will be used later.

$$\begin{pmatrix} G_x{}' & G_y{}' & G_z{}' \end{pmatrix} = \begin{pmatrix} G_{x,raw} & G_{y,raw} & G_{z,raw} & 1 \end{pmatrix} \begin{pmatrix} G_{p11} & G_{p21} & G_{p31} \\ G_{p12} & G_{p22} & G_{p32} \\ G_{p13} & G_{p23} & G_{p33} \\ G_{p10} & G_{p20} & G_{p30} \end{pmatrix} \tag{2.49}$$

When one solves equation 2.49 six times with the orientations shown inTable 2-4 and raw accelerometer data $(G_{x,raw}, G_{y,raw}, G_{z,raw})$ the following is obtained:

$$\begin{pmatrix} 1 & 0 & 0 \\ -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & -1 \end{pmatrix} = \begin{pmatrix} G_{x,raw1} & G_{y,raw1} & G_{z,raw1} & 1 \\ G_{x,raw2} & G_{y,raw2} & G_{z,raw2} & 1 \\ G_{x,raw3} & G_{y,raw3} & G_{z,raw3} & 1 \\ G_{x,raw4} & G_{y,raw4} & G_{z,raw4} & 1 \\ G_{x,raw5} & G_{y,raw5} & G_{z,raw5} & 1 \\ G_{x,raw6} & G_{y,raw6} & G_{z,raw6} & 1 \end{pmatrix} \begin{pmatrix} G_{p11} & G_{p21} & G_{p31} \\ G_{p12} & G_{p22} & G_{p32} \\ G_{p13} & G_{p23} & G_{p33} \\ G_{p10} & G_{p20} & G_{p30} \end{pmatrix} \tag{2.50}$$

Now by using a least square approach one can estimate the parameters as being:

$$\hat{G}_p = (\bar{G}^T{}_{raw}\, \bar{G}_{raw})^{-1}\, \bar{G}^T{}_{raw}\, \bar{G}' \tag{2.51}$$

With:

$\hat{G}_p$ is $\begin{pmatrix} G_{p11} & G_{p21} & G_{p31} \\ G_{p12} & G_{p22} & G_{p32} \\ G_{p13} & G_{p23} & G_{p33} \\ G_{p10} & G_{p20} & G_{p30} \end{pmatrix}$

$\bar{G}_{raw}$ is $\begin{pmatrix} G_{x,raw1} & G_{y,raw1} & G_{z,raw1} & 1 \\ G_{x,raw2} & G_{y,raw2} & G_{z,raw2} & 1 \\ G_{x,raw3} & G_{y,raw3} & G_{z,raw3} & 1 \\ G_{x,raw4} & G_{y,raw4} & G_{z,raw4} & 1 \\ G_{x,raw5} & G_{y,raw5} & G_{z,raw5} & 1 \\ G_{x,raw6} & G_{y,raw6} & G_{z,raw6} & 1 \end{pmatrix}$

$\bar{G}'$ is $\begin{pmatrix} 1 & 0 & 0 \\ -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & -1 \end{pmatrix}$

2.4.3 Magnetometer calibration

A magnetometer measures the strength and attitude of the magnetic field (B) around it and outputs a vector containing this data. While capturing this data, four main sources of error are present, these are: hard-iron interference, soft-iron interference, scale factor and misalignment. When no distortions are present the magnetic measurements are a sphere as shown in the top left of Figure 2-10. Hard-iron magnetic fields refer to any permanently fixed magnet within the vicinity of the magnetometer that do not vary with time, these are shown in the top right of Figure 2-10 where the centre of the sphere in the top left of Figure 2-10 is shifted. Soft-iron magnetic fields are generated by any material that has the ability to be magnetized temporarily around the magnetometer and vary with time, these are shown in bottom left of Figure 2-10 where the sphere is deformed and rotated. The bottom right of Figure 2-10 shows the combination of hard and soft-iron distortions. The scale factor is the difference in sensitivity between the three axes of the magnetometer within the same magnetic field. And finally misalignment is the same as for the accelerometer, it is a misalignment between the magnetometer and the body axes.

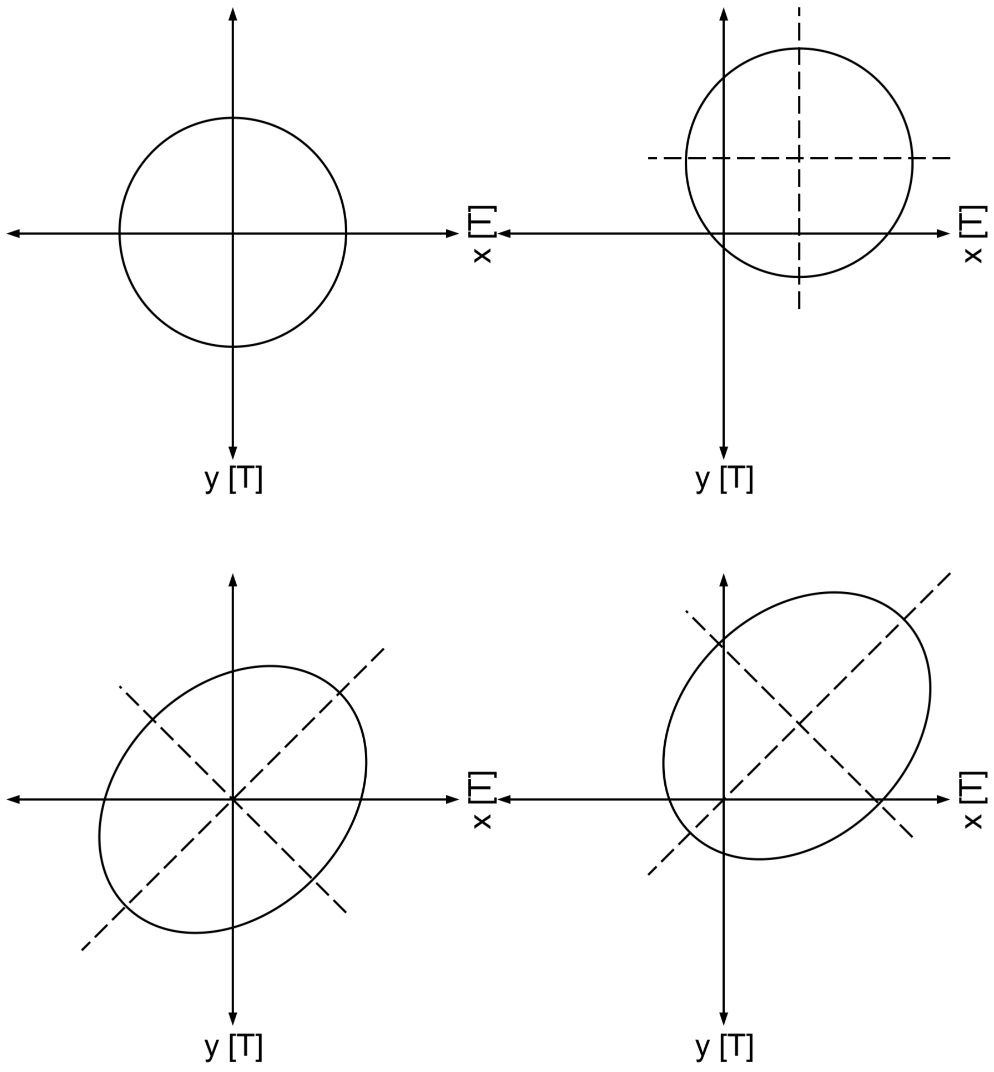


Figure 2-10: Top left: no distortion, top right: hard-iron deviation, bottom left: soft-iron deviation, bottom right: hard and soft-iron deviation

Looking back at Figure 2-8 the magnetometer is orientated towards magnetic north resulting in the following magnetic field vector ($\bar{M}'$):

$$\bar{M}' = B \begin{pmatrix} \cos\beta \\ 0 \\ \sin\beta \end{pmatrix} \tag{2.52}$$

During the calibration process the magnetometer will be rotated around all three axes in a roll ($\bar{R}_r$), pitch ($\bar{R}_p$) and yaw ($\bar{R}_y$) motion as shown in Figure 2-9. After this rotation process the new rotated magnetometer values ($\bar{M}_r$) can be given as:

$$\bar{M}_r = \bar{R}_r \, \bar{R}_p \, \bar{R}_y \, \bar{M}' \tag{2.53}$$

The yaw rotation matrix is given by:

$$\bar{R}_y = \begin{pmatrix} \cos\psi & \sin\psi & 0 \\ -\sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{2.54}$$

Equation 2.53 ignores the effect of all four sources of disturbance described above. The hard-iron interference can be countered by adding a vector ($\bar{D}_{hard}$) since this disturbance only causes an offset to the sphere. The soft-iron interference is assumed to be related linearly to the rotated magnetometer values by a 3x3 matrix $\bar{D}_{soft}$. The scale factor is accounted for by a diagonal matrix $\bar{D}_{scale}$, and the misalignment can be adjusted for by a 3x3 matrix $\bar{D}_{misal}$. Because these last three matrices are of the same size they can be combined to form:

$$\bar{D}_{comb} = \bar{D}_{soft} \, \bar{D}_{scale} \, \bar{D}_{misal} \tag{2.55}$$

Now the hard-iron vector $\bar{D}_{hard}$ and $\bar{D}_{comb}$ can be added to equation 2.53 to get the magnetometer reading after an arbitrary amount of rotations.

$$\bar{M}_r = \bar{D}_{comb} \, \bar{R}_r \, \bar{R}_p \, \bar{R}_y \, B \begin{pmatrix} \cos\beta \\ 0 \\ \sin\beta \end{pmatrix} + \bar{D}_{hard} \tag{2.56}$$

To get the de-rotated magnetometer values one can take equation 2.56 and arrange it in such a way that the magnetic field vector ($\bar{M}' = B \begin{pmatrix} \cos\beta \\ 0 \\ \sin\beta \end{pmatrix}$) is rotated around the Z-axis ($\bar{R}_y$) in Figure 2-9. The inverse of a rotation matrix is simply the negative angles of all terms as stated in (McCarthy et al., 2011, p.190). Thus:

$$\begin{gathered} \bar{R}_y \, B \begin{pmatrix} \cos\beta \\ 0 \\ \sin\beta \end{pmatrix} = \bar{R}_p^{\;-1} \, \bar{R}_r^{\;-1} \, \bar{D}_{comb}^{\;\;-1} \, (\bar{M}_r - \bar{D}_{hard}) \\ \therefore \begin{pmatrix} \cos\psi & \sin\psi & 0 \\ -\sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} B \ \cos\beta \\ 0 \\ B \ \sin\beta \end{pmatrix} \\ = \bar{R}_p^{\;-1} \, \bar{R}_r^{\;-1} \, \bar{D}_{comb}^{\;\;-1} \, (\bar{M}_r - \bar{D}_{hard}) \\ \therefore \begin{pmatrix} \cos\psi \ B \ \cos\beta \\ -\sin\psi \ B \ \cos\beta \\ B \ \sin\beta \end{pmatrix} = \bar{R}_p^{\;-1} \, \bar{R}_r^{\;-1} \, \bar{D}_{comb}^{\;\;-1} \, (\bar{M}_r - \bar{D}_{hard}) \end{gathered} \tag{2.57}$$

Let:

$$\begin{pmatrix} M_x \\ M_y \\ M_z \end{pmatrix} = \bar{R}_p^{\;-1} \, \bar{R}_r^{\;-1} \, \bar{D}_{comb}^{\;\;-1} \, (\bar{M}_r - \bar{D}_{hard}) \tag{2.58}$$

Where:

$\begin{pmatrix} M_x \\ M_y \\ M_z \end{pmatrix}$ represents the magnetometer readings that have been corrected for all four disturbances to the XY-plane as shown in Figure 2-9.

Thus from equations 2.57 and 2.58 the following can be said:

$$\cos\psi \ B \ \cos\beta = M_x \tag{2.59}$$

$$-\sin\psi \ B \ \cos\beta = M_y \tag{2.60}$$

This can then be substituted back into equation 2.28 to find the current compass heading. Thus all the disturbances can be removed if one knows the hard-iron vector and the inverse of the soft-iron combined matrix in the $\bar{D}_{comb}{}^{-1} \, (\bar{M}_r - \bar{D}_{hard})$ term in equation 2.57. The locus that describes the magnetometer measurements on the sphere as shown in Figure 2-10 will be used to determine the unknown vector and matrix. Looking at equation 2.56 and realizing the magnetometer measurements lie on a surface the following can be deducted.

$$\bar{D}_{comb}{}^{-1} \, (\bar{M}_r - \bar{D}_{hard}) = \bar{R}_r \, \bar{R}_p \, \bar{R}_y \, B \begin{pmatrix} \cos\beta \\ 0 \\ \sin\beta \end{pmatrix} \tag{2.61}$$

Now the following transpose multiplication will be made using equation 2.61 as base to extract the magnetometer measurements and in the process creating a relationship to the unknown vector and matrix.

$$\begin{aligned} &\left(\bar{R}_r \, \bar{R}_p \, \bar{R}_y \, B \begin{pmatrix} \cos\beta \\ 0 \\ \sin\beta \end{pmatrix}\right)^T \bar{R}_r \, \bar{R}_p \, \bar{R}_y \, B \begin{pmatrix} \cos\beta \\ 0 \\ \sin\beta \end{pmatrix} \\ &= \begin{pmatrix} \cos\beta \\ 0 \\ \sin\beta \end{pmatrix}^T B^T \, \bar{R}_y{}^T \, \bar{R}_p{}^T \, \bar{R}_r{}^T \, \bar{R}_r \, \bar{R}_p \, \bar{R}_y \, B \begin{pmatrix} \cos\beta \\ 0 \\ \sin\beta \end{pmatrix} \\ &= B^2 \, (\cos\beta \quad 0 \quad \sin\beta) \begin{pmatrix} \cos\beta \\ 0 \\ \sin\beta \end{pmatrix} \\ &= B^2 \, (\cos^2\beta + \sin^2\beta) \\ &= B^2 \end{aligned} \tag{2.62}$$

Looking at the first line of equation 2.62 and using this in conjunction with equation 2.61 one can get the following:

$$\left(\bar{D}_{comb}{}^{-1} \, (\bar{M}_r - \bar{D}_{hard})\right)^T \, \bar{D}_{comb}{}^{-1} \, (\bar{M}_r - \bar{D}_{hard}) = B^2 \tag{2.63}$$

If one has a symmetrical matrix $\bar{A}_s$ and an orthogonal matrix $\bar{Q}_{om}$ that has the eigenvectors of $\bar{A}_s$ as column entries, the following can be said according to (Meyer, 2000):

$$\bar{Q}_{om}{}^T \bar{A}_s \bar{Q}_{om} = constant \tag{2.64}$$

The centre of the ellipsoid containing all the measurements can have coordinates $\bar{Q}_0$. Now the expression that defines the locus of points on the ellipsoid can be given as:

$$(\bar{Q}_{om} - \bar{Q}_0)^T \bar{A}_s (\bar{Q}_{om} - \bar{Q}_0) = constant \tag{2.65}$$

It still needs to be proofed that $\bar{A}_s$ (which will be related to the soft-iron distortion part) is symmetrical:

$$\begin{aligned}
\bar{A}_s &= \left(\bar{D}_{comb}{}^{-1}\right)^T \bar{D}_{comb}{}^{-1} \\
\therefore \bar{A}_s{}^T &= \left(\left(\bar{D}_{comb}{}^{-1}\right)^T \bar{D}_{comb}{}^{-1}\right)^T \\
&= \left(\bar{D}_{comb}{}^{-1}\right)^T \left(\left(\bar{D}_{comb}{}^{-1}\right)^T\right)^T \\
&= \left(\bar{D}_{comb}{}^{-1}\right)^T \bar{D}_{comb}{}^{-1} \\
&= \bar{A}_s
\end{aligned} \tag{2.66}$$

Thus $\bar{A}_s$ is symmetrical. Now it can be seen that the magnetometer measurements form part of the ellipsoid surface that is defined by equations 2.62, 2.63 and 2.65. The centre of the ellipsoid is at $\bar{Q}_0 = \bar{D}_{hard}$, capturing the hard-iron distortions, the shape of the ellipsoid is defined by matrix $\bar{A}_s$, that captures the transpose of the inverse of the squared soft-iron distortions, and finally the size of the ellipsoid is captured by B, the magnetic field strength.

The hard-iron distortion can now be directly captured. But to calibrate the magnetic field measurements successfully the soft-iron distortion term $\bar{D}_{comb}{}^{-1}$ in $\bar{D}_{comb}{}^{-1} (\bar{M}_r - \bar{D}_{hard})$ is still needed. It is easy to calculate $\bar{A}_s$ given $\bar{D}_{comb}{}^{-1}$, but the reverse is not true. To get around the problem of having to use $\bar{D}_{comb}{}^{-1}$ to solve the problem another approach to the problem can be taken. The soft-iron distortion, $\bar{D}_{comb}$, is the product of three independent 3x3 matrices with 9 independent variables as shown in equation 2.55, but in equation 2.66 it was shown that the soft-iron distortion matrix is in fact symmetrical, meaning it only has 6 independent variables. The assumption is made that the inverse of the soft-iron distortion matrix, $\bar{D}_{comb}{}^{-1}$, is also symmetrical with 6 degrees of freedom. This assumption will later be confirmed. If the magnetometer values, $\bar{M}_r$, are corrected by an estimate of the hard-iron distortion, $\bar{D}_{hard,cal}$, and an estimate of the soft-iron distortion, $\bar{D}_{comb,cal}$, the corrected magnetometer values, $\bar{M}_{r,corrected}$, can be obtained, which can be given by the following equation using equation 2.61 as basis for it:

$$\begin{aligned}\bar{M}_{r,corrected} &= \bar{D}_{comb,cal}{}^{-1}\,(\bar{M}_r - \bar{D}_{hard,cal}) \\ &= \bar{D}_{comb,cal}{}^{-1}\,\left(\bar{D}_{comb}\,\bar{R}_r\,\bar{R}_p\,\bar{R}_y\,B\begin{pmatrix}\cos\beta\\0\\\sin\beta\end{pmatrix} + (\bar{D}_{hard} \right.\\ &\quad \left. - \bar{D}_{hard,cal})\right)\end{aligned} \tag{2.67}$$

If this correctly estimates the hard and soft-iron distortions then $\bar{D}_{comb,cal}{}^{-1}\,\bar{D}_{comb,cal} = \bar{I}$ and $\bar{D}_{hard,cal} = \bar{D}_{hard}$, meaning equation 2.67 reduces to:

$$\begin{aligned}&\bar{M}_{r,corrected}{}^{T}\,\bar{M}_{r,corrected} \\ &\quad = \left(\bar{R}_r\,\bar{R}_p\,\bar{R}_y\,B\begin{pmatrix}\cos\beta\\0\\\sin\beta\end{pmatrix}\right)^T \left(\bar{R}_r\,\bar{R}_p\,\bar{R}_y\,B\begin{pmatrix}\cos\beta\\0\\\sin\beta\end{pmatrix}\right)\end{aligned} \tag{2.68}$$

And from equation 2.62:

$$\left(\bar{R}_r\,\bar{R}_p\,\bar{R}_y\,B\begin{pmatrix}\cos\beta\\0\\\sin\beta\end{pmatrix}\right)^T \left(\bar{R}_r\,\bar{R}_p\,\bar{R}_y\,B\begin{pmatrix}\cos\beta\\0\\\sin\beta\end{pmatrix}\right) = B^2 \tag{2.69}$$

It was previously stated that the inverse of a rotation is the same as the inverse of the angle in question. Such a rotation is introduced, $\bar{T} = \bar{D}_{comb,cal}{}^{-1}\,\bar{D}_{comb}$, to equation 2.68 to obtain:

$$\begin{aligned}&\bar{M}_{r,corrected}{}^{T}\,\bar{M}_{r,corrected} \\ &\quad = \left(\bar{T}\,\bar{R}_r\,\bar{R}_p\,\bar{R}_y\,B\begin{pmatrix}\cos\beta\\0\\\sin\beta\end{pmatrix}\right)^T \left(\bar{T}\,\bar{R}_r\,\bar{R}_p\,\bar{R}_y\,B\begin{pmatrix}\cos\beta\\0\\\sin\beta\end{pmatrix}\right) \\ &\quad = B^2\end{aligned} \tag{2.70}$$

When the assumption that $\bar{D}_{comb}{}^{-1}$ is symmetrical is enforced it is impossible that an error in compass heading can be introduced to equation 2.70 because for a rotation to be present the rotation matrix must be anti-symmetric. Because $\bar{D}_{comb}{}^{-1}$ is symmetric now, the soft-iron distortion can be related to the ellipsoid of equations 2.62, 2.63 and 2.65 by:

$$\begin{aligned}\bar{A}_s &= \left(\bar{D}_{comb}{}^{-1}\right)^T\,\bar{D}_{comb}{}^{-1} \\ &= \bar{D}_{comb}{}^{-1}\,\bar{D}_{comb}{}^{-1} \\ \therefore\ \bar{D}_{comb}{}^{-1} &= \bar{A}_s{}^{\frac{1}{2}}\end{aligned} \tag{2.71}$$

Now there are terms for the hard-iron interference, soft-iron interference, scale factor and misalignment, and the calibrated compass heading can be calculated.

3 Design

The design of the autonomous navigation system can be split into hardware and software sections. On the hardware side the robotic platform was constructed by an undergraduate student but had to be reinforced to support the weight of the battery and mounting points had to be added for the sensors as shown in Figure 3-1. The two DC motors, four wheels, four bearing housing and bearings, two timing pulleys and belts, shafts, dual motor driver and robot frame were present when the platform was received. The following was still required: a power source, a microcontroller to control all robotic functions and sensors to capture the required data.

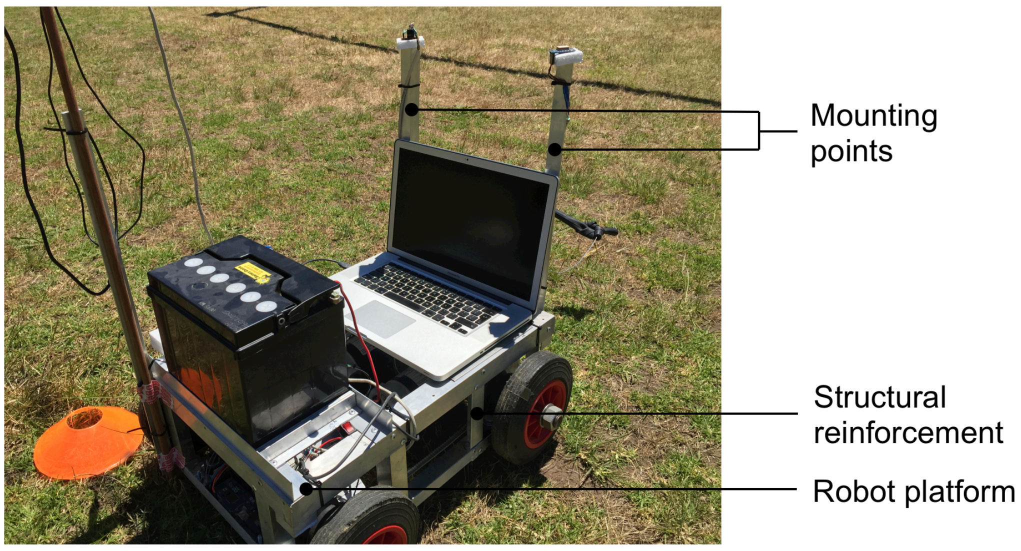


Figure 3-1: Robotic platform

3.1 Hardware

First a summary of all the hardware presently on the robot is given to ensure a complete system picture when system integration and power source selection is done. Next all the individual additional sensors added to capture the experimental data is discussed, then a microcontroller capable of capturing all the sensor data is considered and finally a power source able to power all the hardware is chosen.

3.1.1 DC motors

The motors present are the Bircraft EC100.120 permanent magnet motors. They are $140\ \mathrm{W}$, $12\ \mathrm{V}$ DC motors with 2 inside brushes, drawing a maximum of $16{,}8\ \mathrm{A}$ and weighs $2{,}7\ \mathrm{kg}$ each. The DC motors are mounted such that differential steering of the robot is possible by connecting the front and rear wheels of each side with timing pulleys and belts.

3.1.2 Motor driver

The motor driver present is the Sabertooth 2x25 V2 model. This driver can deliver $25\,\mathrm{A}$ continuous current for each motor at a nominal input voltage of $6 - 30\,\mathrm{V}$, has thermal and overcurrent protection, allows analog, R/C, simplified serial and packetized serial input modes. The datasheet suggests that when requiring a steady voltage at currents larger than $20\,\mathrm{A}$ the user should consider using a high capacity lead-acid battery.

Figure 3-2 shows the connections between the microcontroller, the two DC motors and the battery. M1A and M1B connect to one motor and M2A and M2B to the other, reversing the wires simply reverses the motor direction. B- connects to the negative battery pole and B+ to the positive pole. GND connects to the ground of the microcontroller and the serial transmit line from the microcontroller connects to S1.

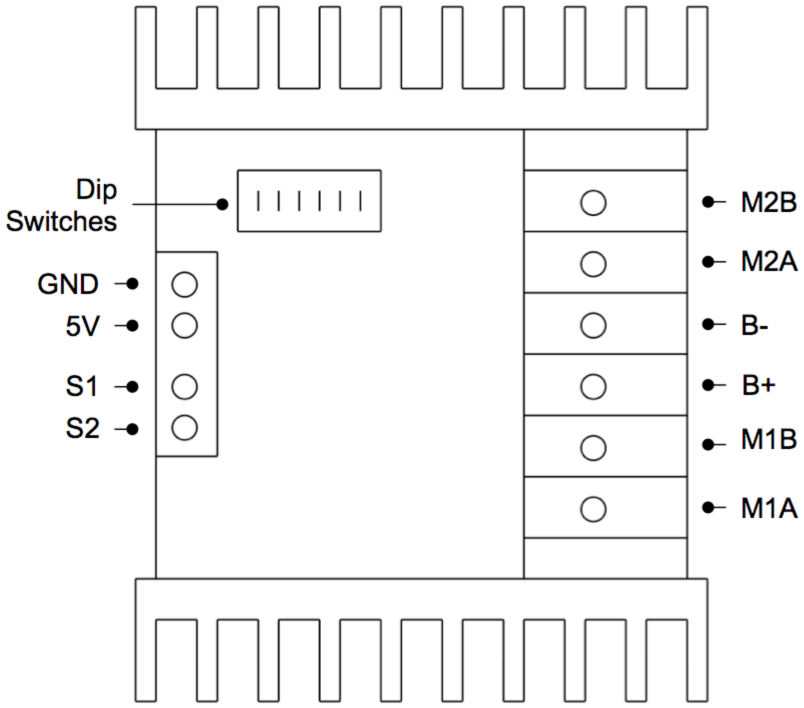


Figure 3-2: Motor driver

To control the speed and direction of the two DC motors the packetized serial mode is chosen that uses TTL level multi-byte serial commands to control the motors. This mode works in one direction only, data is only received, no feedback is given to the microcontroller. The data packet that is sent has the following format: an address byte, a command byte, a data byte and a 7-bit checksum. The address byte has a value greater than 127, while the command and data bytes have values less than 128. Table 3-1 shows the packet that must be sent to move the motor backwards at $75\ \%$. The Checksum can be calculated as follows, where 0b01111111 is the mask of decimal 127 in the 8-bit system.

$$\begin{aligned} Checksum &= (address\ byte + command\ byte \\ &\quad + data\ byte)\ \&\ 0b01111111 \\ &= (128 + 1 + 96)\ \&\ 0b01111111 \\ &= 0b11100001\ \&\ 0b01111111 \\ &= 0b01100001 \\ &= 97 \end{aligned} \tag{3.1}$$

Table 3-1: Data packet

Address	128
Command	1
Data	96
Checksum	97

The dip switches shown in Figure 3-2 are used to set the mode (switch 1 & 2 down for packetized serial), select the lithium cut-off (but not present here, thus switch 3 is up) and the address chosen as 128 (switch 4, 5 & 6 in up position).

3.1.3 Piksi GPS

The Piksi GPS is a carrier phase RTK GPS with centimetre level relative positioning accuracy, consisting of two modules: one known as the rover that will be mounted on the moving robot and the other is the base station and will be kept stationary as a point of reference. These GPS modules will mainly be used to verify the accuracy of the data captured by the other sensors. Table 3-2 shows the important electrical characteristics of a module:

Table 3-2: Piksi GPS electrical properties

Supply Voltage	$3{,}5 - 5{,}5$ V
Power Consumption	500 mW (max)
Position / velocity / time update rate	50 Hz

Figure 3-3 shows one of the modules and all the connections used. Each module has a 433 Mhz 3DR radio (transmit power of up to 100 mW) connected to its UART (universal asynchronous receiver / transmitter) A port to enable two-way communication between the two modules. The base station module is powered through the micro-USB with a USB power bank and an external antenna is attached. The external antenna is used to allow easier positioning of the antenna. The rover module is connected to the microcontroller via UART B and also has an external antenna. This module is also powered from the microcontroller. The status LEDs show whether an RTK fix is present and how many satellites are used in the solution.

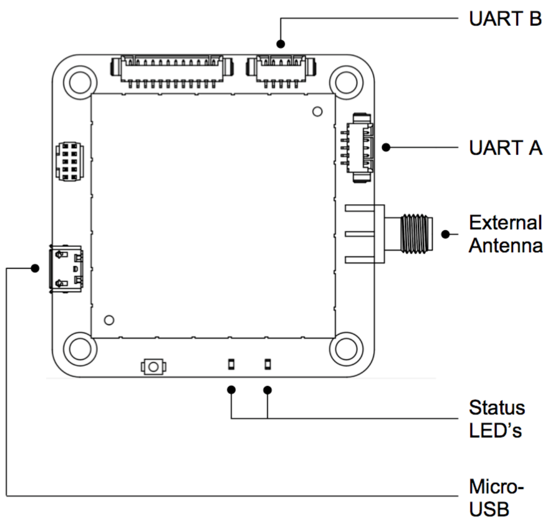


Figure 3-3: Piksi GPS[7]

3.1.4 Optical encoders

Optical encoders use a source of light that is projected at a disc with opaque and transparent areas mounted on a rotating shaft to measure the angular position of the disc. The enclosure, disk and optical switch are shown in Figure 3-4.

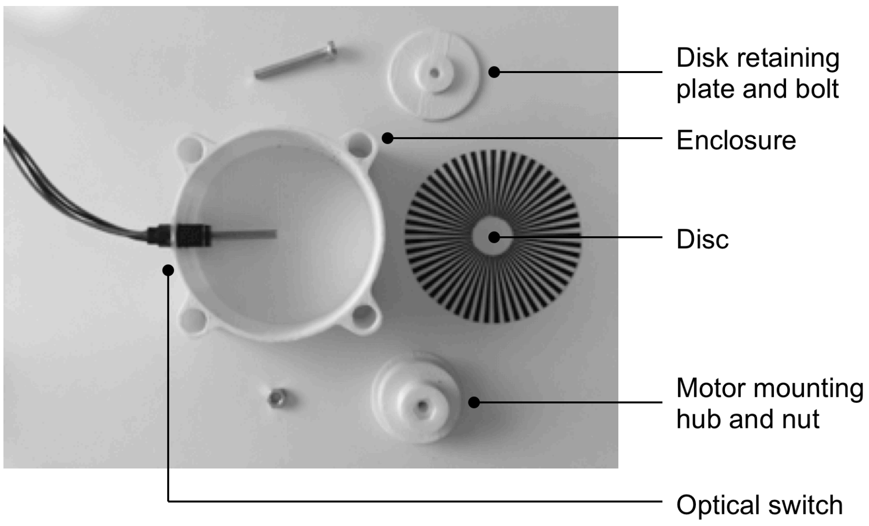


Figure 3-4: Optical encoder

[7] Adapted from Swiftnav, 2013, http://docs.swiftnav.com/pdfs/piksi_datasheet_v2.3.1.pdf

The optical switch used here is the OPTEK OPB980L55Z, which houses an OP240 LED to emit light and an OPL560 detector to retrieve light, with the electrical characteristics shown in Table 3-3:

Table 3-3: Optical encoder electrical properties

OP240 LED	
Forward Voltage	$1{,}70\ \mathrm{V}\ (@\ 20\ \mathrm{mA}\ \&\ 298{,}15\ \mathrm{K})$
OPL560 Detector	
Supply Voltage	$4{,}5 - 16\ \mathrm{V}$
Supply Current	$12\ \mathrm{mA}\ (@\ 4{,}5 - 16\ \mathrm{V})$

Figure 3-5 shows a block diagram of the sensor package, where A & K denote the anode and cathode of the LED, V_{cc} is the $+5\ \mathrm{V}$ supply voltage to the detector, OUT is the detector output (where $0\ \mathrm{V}$ denotes no light observed and $5\ \mathrm{V}$ the opposite) and GND grounds the sensor.

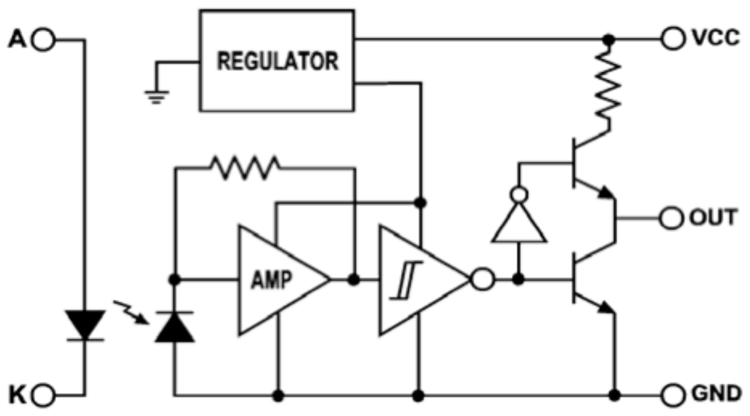


Figure 3-5: Totem-pole output buffer[8]

On the LED side the current must be limited according to the datasheet. Since the forward voltage over the LED is $1{,}70\ \mathrm{V}$ and the LED is powered by $5\ \mathrm{V}$, a resistor is needed between the anode and cathode to drop $3{,}30\ \mathrm{V}$ over it. The $1{,}70\ \mathrm{V}$ of the LED was measured at $20\ \mathrm{mA}$ (half of the absolute maximum forward current) and if this is taken as the current through the resistor as well, the resistor value must be:

[8] Adapted from OPTEK Technology, 2013, http://optekinc.com/datasheets/opb960-990_series.pdf

$$\begin{aligned} V &= i\,R \\ \therefore R &= \frac{V}{i} \\ &= \frac{3{,}30}{0{,}020} \\ &= 165\ \Omega \end{aligned} \tag{3.2}$$

Thus choose the closest standard resistor, a $160\ \Omega$ resistor.

The disc has a total of 106 slots, meaning that over one rotation of the wheel the slotted optical switch will tick 106 times. The wheels have a diameter of $200\ \text{mm}$. This means that every time the switch ticks, the robot has travelled the following distance:

$$\begin{aligned} Distance\ travelled &= \frac{wheel\ circumference}{106} \\ &= \frac{\pi\,d}{106} \\ &= \frac{\pi * 200}{106} \\ &= 5{,}9275\ \text{mm} \end{aligned} \tag{3.3}$$

3.1.5 Digital compass

The Compass Click unit used as tilt compensated compass houses a LSM303DLHC module. This module consists of a 3-axis accelerometer and 3-axis magnetometer. The module communicates with the microcontroller using an I^{2}C serial bus interface. The module has the electrical characteristics shown in Table 3-4:

Table 3-4: Tilt compensated compass electrical properties

Supply voltage	$2{,}16 - 3{,}6\ \mathrm{V}$
Current consumption (magnetic sensor @ 7,5 Hz and accelerometer @ 50 Hz)	$110\ \mu\mathrm{A}$
SCL clock frequency	$100\ \mathrm{kHz}$

Figure 3-6 shows the SDA (Serial Data Line) and SCL (Serial Clock Line) channels of the serial interface, and where the module is supplied with power and grounded.

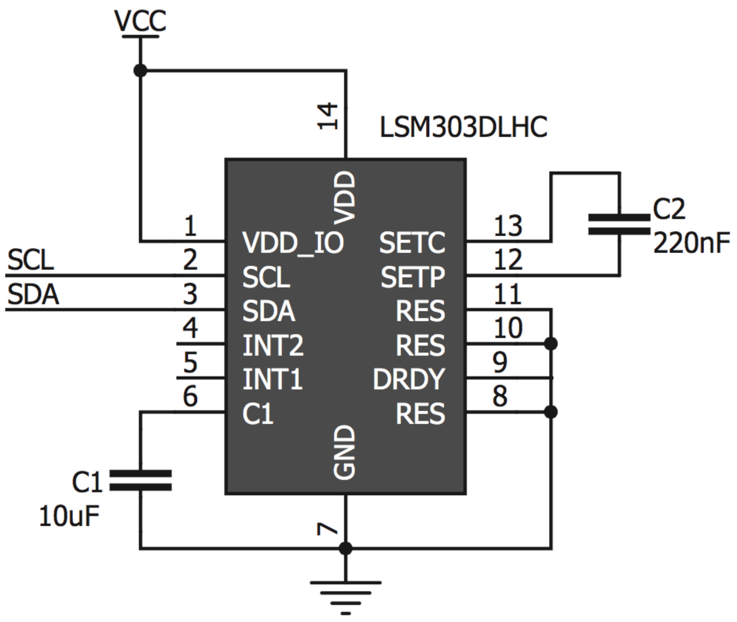


Figure 3-6: LSM303DLHC module

Since the power supplied to all other sensors will be $+5\ \mathrm{V}$, a way will have to be found to step this voltage down to $3{,}3\ \mathrm{V}$. This can be done using a bi-directional logic level converter. Shown in Figure 3-7 is such a device that has four level-shifting channels. The circuit for each channel consists of a single N-channel MOSFET with two pull-up resistors. For the first channel HV1 indicates the high voltage data channel $(+5\ \mathrm{V})$, LV1 the low voltage data channel $(+3{,}3\ \mathrm{V})$, HV the high voltage input, LV the low voltage input and GND on both sites the respective grounds.

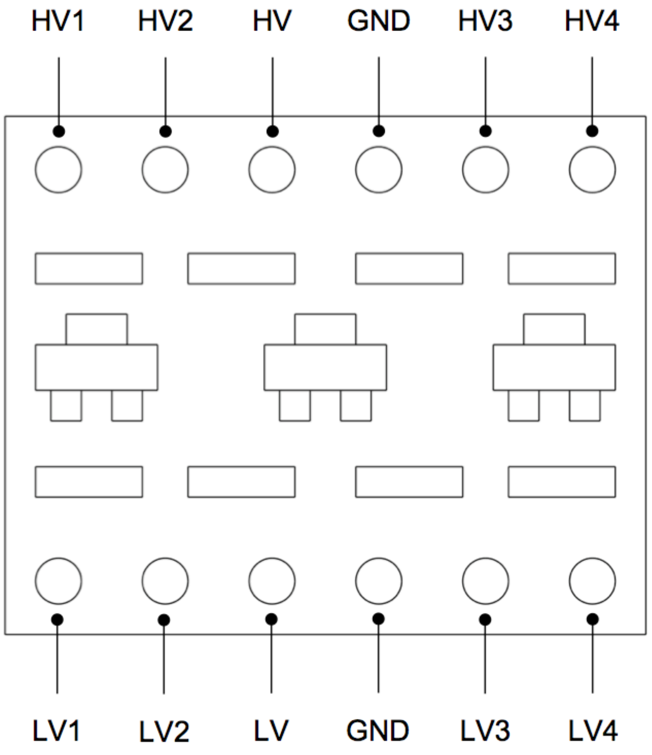


Figure 3-7: Bi-directional logic level converter

3.1.6 Adafruit GPS

The Adafruit GPS is manufactured by Adafruit, which houses a MTK3339 chipset that is capable of tracking up to 22 satellites on 66 channels, with a high-sensitivity receiver and an integrated antenna. The GPS can be battery powered to keep the RTC (real time clock) running to reduce the time it takes to get a fix when the GPS is powered on. The module has DGPS support and features serial communication and has the electrical characteristics shown in Table 3-5:

Table 3-5: Adafruit GPS electrical properties

Supply Voltage	$3{,}0 - 5{,}5\ \mathrm{V}$
Operating Current	$25\ \mathrm{mA}$ tracking $20\ \mathrm{mA}$ navigating
Position / velocity / time update rate	$10\ \mathrm{Hz}$

Figure 3-8 shows an overview of the module. The TX and RX pins are connected to the RX and TX pins of the microcontroller, the module is grounded with GND, and the $+5\ \mathrm{V}$ from the microcontroller is supplied to VIN. The FIX LED blinks at $1\ \mathrm{Hz}$ until a fix is found, and thereafter once every $15\ \mathrm{s}$. Because the other pins are not used, they will not be discussed.

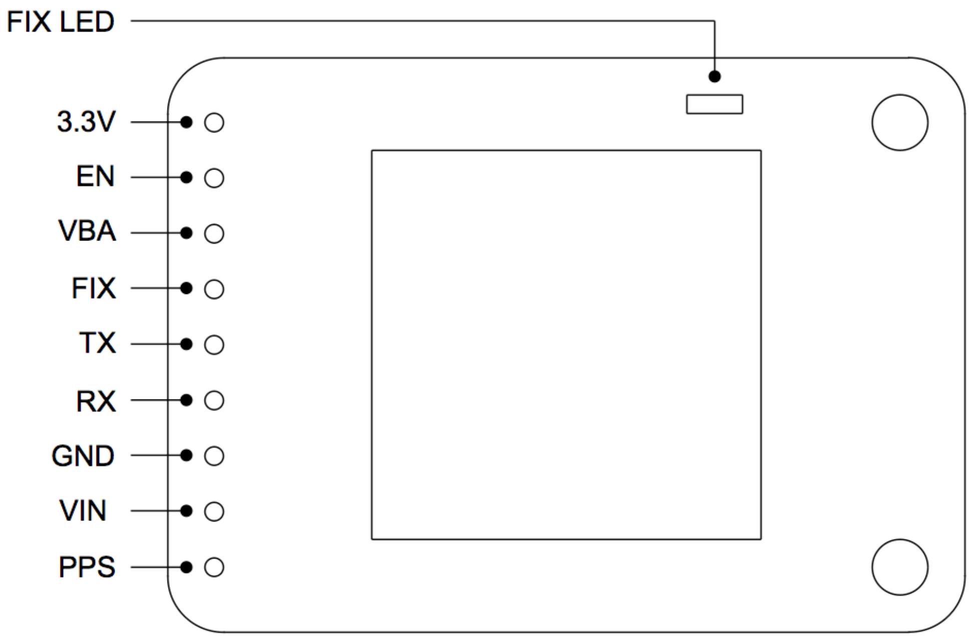


Figure 3-8: Adafruit GPS

3.1.7 Microcontroller

The microcontroller used to control all robotic movement and capture sensor data will have to comply to the following requirements:

1. Support at least two external interrupts on digital pins to capture encoders data.
2. Be able to read interrupts fast enough.
3. Have at least four serial ports for RX and TX TTL serial data transmission between microcontroller and Piksi GPS, motor driver, Adafruit GPS and communication to computer.
4. Support I^{2}C serial bus interface for digital compass.
5. Supply $+5\ \mathrm{V}$ to all sensors and meet current requirements.

The Arduino platform will be considered as the microcontroller to control the robot, specifically the Arduino Mega 2560. First it will be confirmed whether all the requirements are met:

The Arduino Mega supports six external interrupts, has four serial ports and has an I^{2}C serial bus interface, thus requirements 1, 3 and 4 are met.

To determine whether the interrupts can be read fast enough an example where the robot moves at a maximum speed of $2\ \mathrm{m \cdot s^{-1}}$ will be considered. At this speed the slotted optical switch will tick every $5\,927{,}5\ \mu\text{m}$ as shown in equation 3.4 at an interval of:

$$\begin{aligned} Interval &= \frac{2000\ \text{mm/s}}{5.9275\ \text{mm}} \\ &= 0.00296375\ \text{s} \\ &= 2963750\ \text{ns} \end{aligned} \tag{3.4}$$

The external interrupts are tied to the I/O (input / output) Clock that in turn is the same as the CPU Clock, it is $16\ \text{MHz}$. Thus an external interrupt can be triggered every $62{,}5\ \text{ns}$, thus requirement 2 is met.

To determine the current required one can take the sum total of all the sensors connected to the $+5\ \text{V}$ line as shown in Table 3-6:

Table 3-6: Current requirements of sensors

Piksi GPS	$100\ \text{mA}\ (500\ \text{mW} / 5\ \text{V})$
Optical Encoders	$64\ \text{mA}\ (2\ (20\ \text{mA} + 12\ \text{mA}))$
Digital compass	$110\ \mu\text{A}$
Adafruit GPS	$25\ \text{mA}$
TOTAL	$189{,}11\ \text{mA}$

Because the Arduino Mega is powered from the laptop via USB there is a $500\ \text{mA}$ thermofuse that limits the available current on the $+5\ \text{V}$ line, thus the total current drawn by the sensors is well within the current limit and requirement 5 is met.

3.1.8 Battery

It was decided to use a $12\ \text{V}$ deep cycle marine lead-acid battery because of the motor driver suggestion as well as the discharge characteristics of the deep cycle battery mentioned earlier. It is known that the laptop with attached Arduino Mega can be powered for one hour before the laptop battery is flat. Thus the lead-acid battery will also only have to last for one hour. The DC motors draw a maximum of $16{,}8\ \text{A}$ each, meaning that one can run the two motors from the battery for one hour at $33{,}6\ \text{A} \cdot \text{h}$. The smallest available deep cycle marine lead-acid battery of $50\ \text{A} \cdot \text{h}$ was chosen as power source for the DC motors.

3.1.9 Complete system

Now that all the components have been chosen the complete breadboard layout and wiring of the system is given in Figure 3-9.

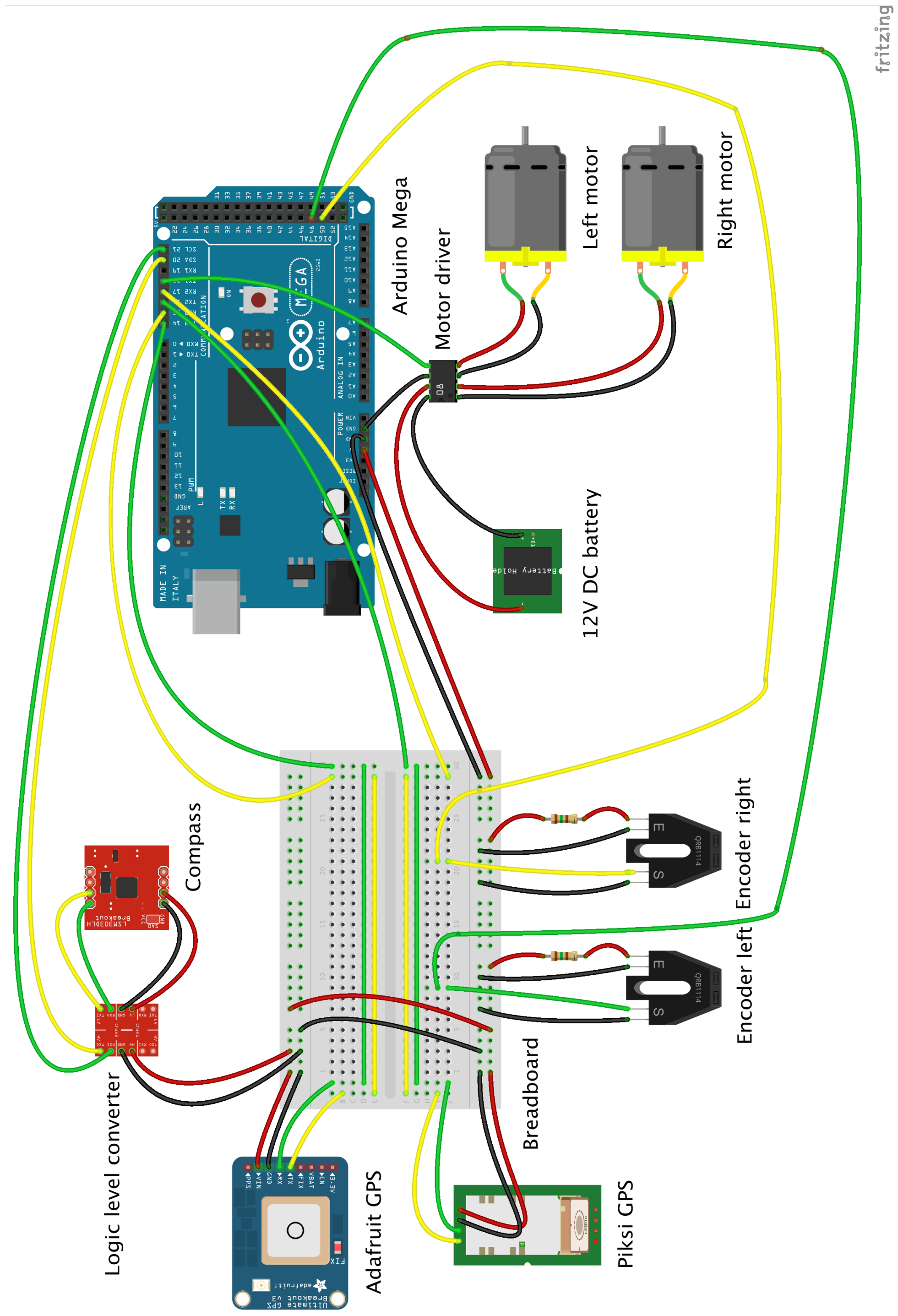


Figure 3-9: Complete breadboard layout

Figure 3-10 shows the individual components discussed in this chapter to give the reader an idea as to the size of each component.

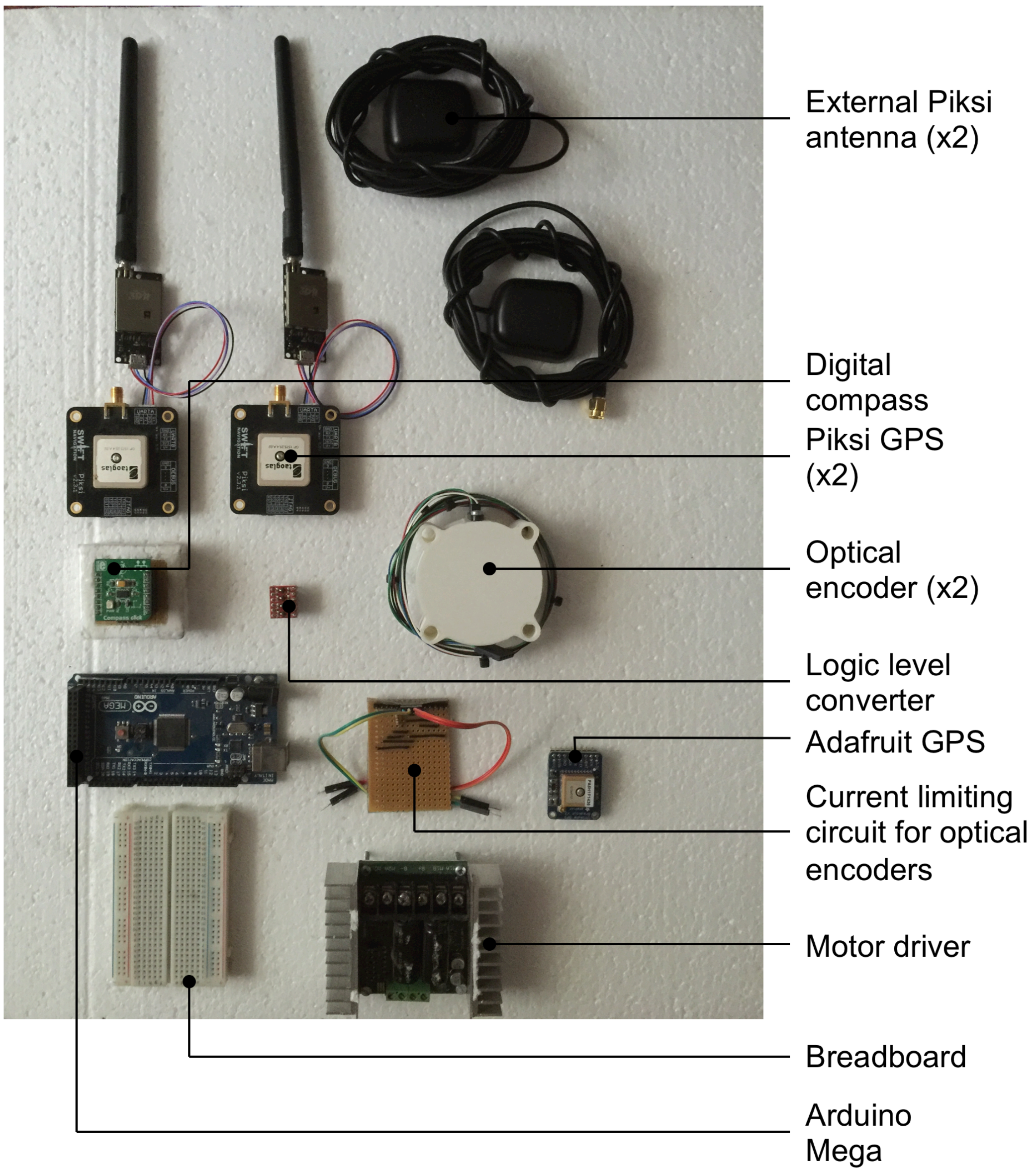


Figure 3-10: Individual components

3.1.10 Cost

The cost of the complete robot is shown in Table 3-7, excluding the parts already present when the robot was received.

Table 3-7: System cost

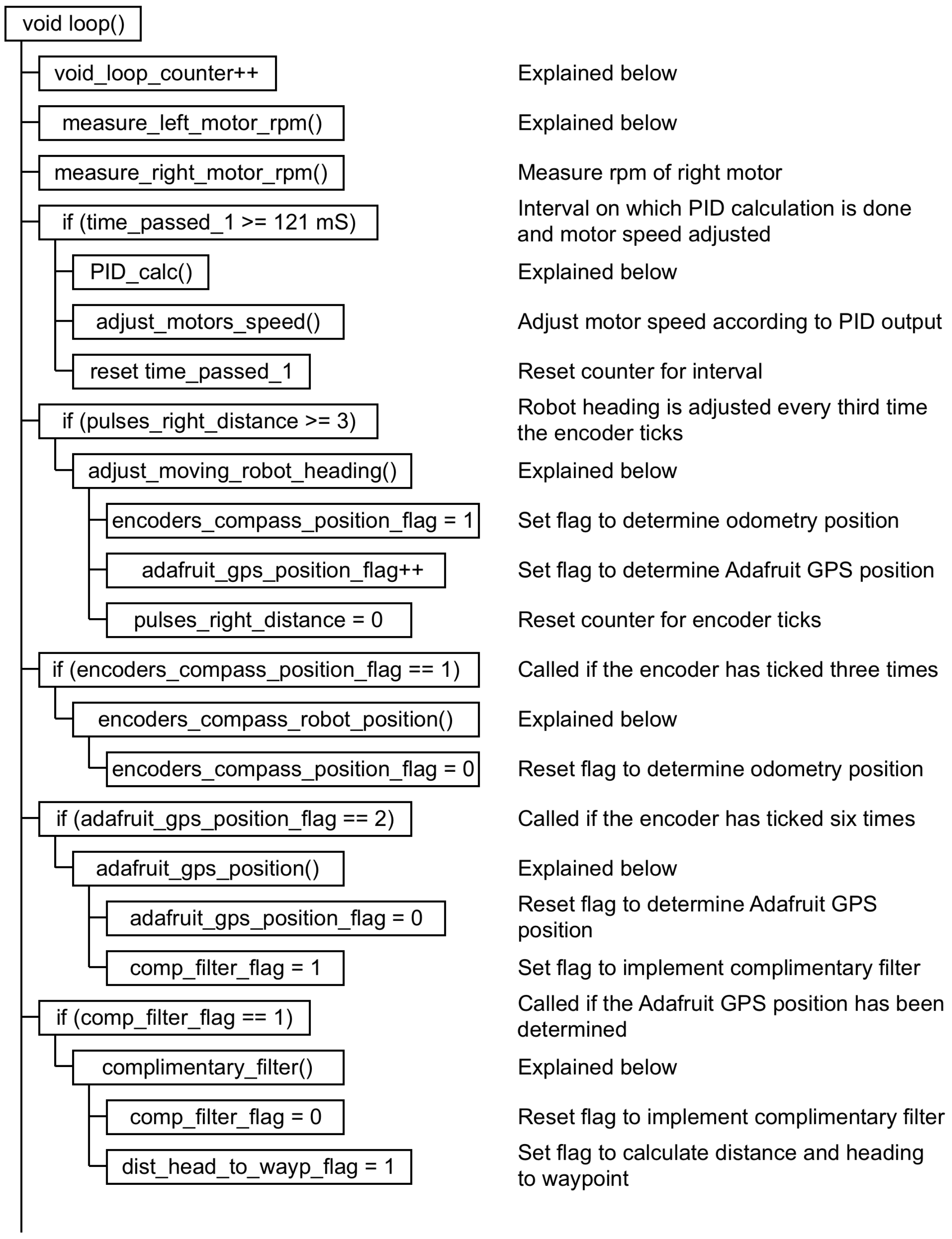
Item	Cost (R)
Arduino Mega	495,00
Compass	426,80
Logic level converter	44,95
Adafruit GPS	690,00
Piksi GPS	14 165,77
Encoders	210,00
12V battery	1 319,09
Wiring	100,00
Breadboard	43,00
3D printing	Free
TOTAL	**17 494,61**

3.2 Arduino code

This section will look at the Arduino code that was developed from scratch to enable the Arduino IDE (integrated development environment) to localize and navigate the robot between waypoints. The code controls the robot hardware, captures data from the sensors, fuses this sensor data and maps the environment. The Arduino IDE gives the user the ability to write Arduino programs in C or C++. The bulk of the time spent on this study went into the development of the software. The software program consists of various parts and can be broken down into the following:

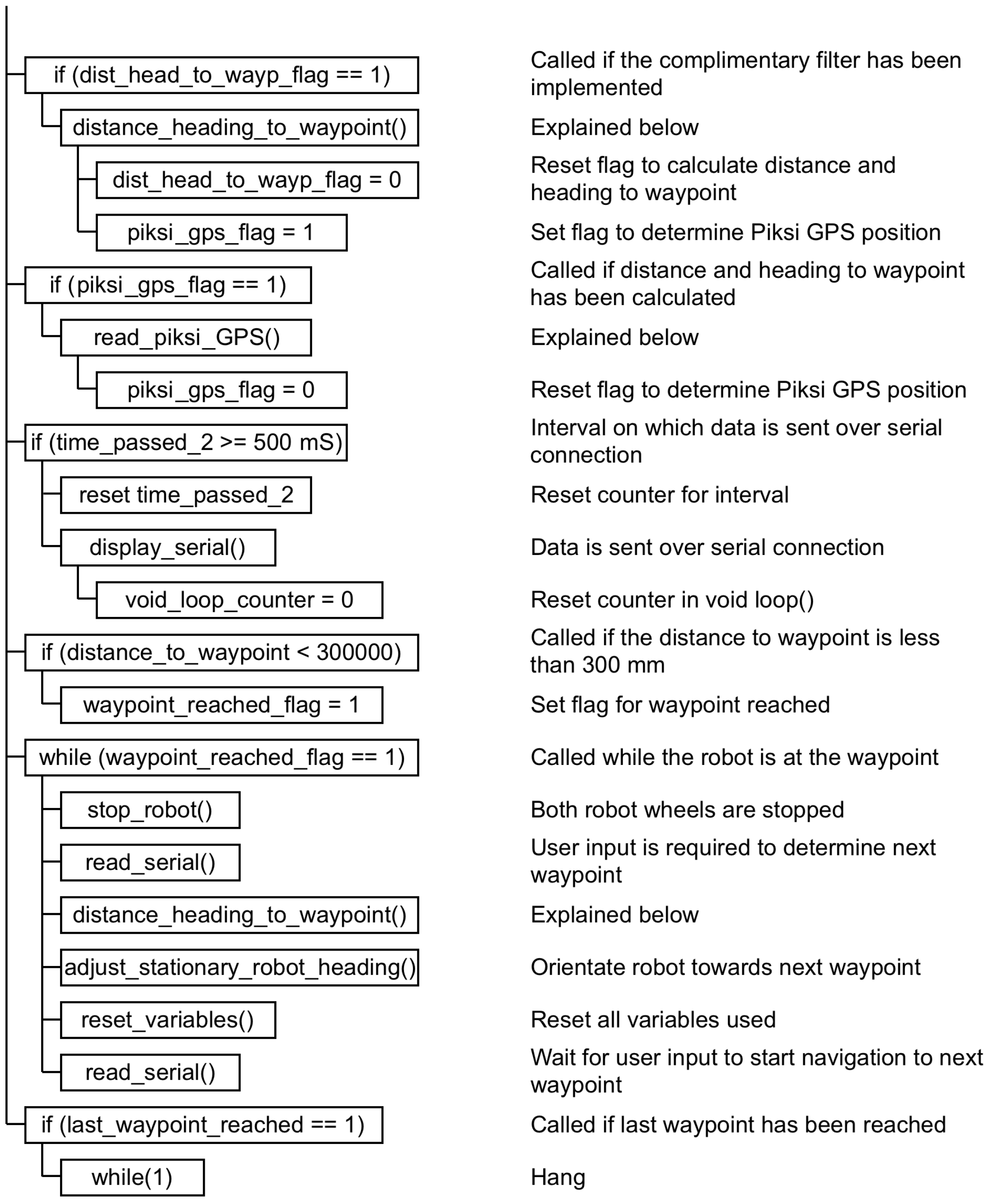
1. Libraries: All the libraries used to communicate with sensors and complete calculations are included here.
2. Variables: All the global variables used are defined here.
3. Objects: After adding the libraries various parameters of the individual sensors must be set here.
4. Void setup(): This code is run once during the start-up process of the Arduino.
5. Void loop(): This code is run repeatedly until power to the Arduino is removed.
6. Functions: To simplify the void loop() various functions are written and only referenced in the void loop().

The void loop() section is where all the navigational procedures are present and can be described by Figure 3-11 and is continued in Figure 3-12.



```
void loop()
  void_loop_counter++                          Explained below
  measure_left_motor_rpm()                     Explained below
  measure_right_motor_rpm()                    Measure rpm of right motor
  if (time_passed_1 >= 121 mS)                 Interval on which PID calculation is done
                                               and motor speed adjusted
    PID_calc()                                 Explained below
    adjust_motors_speed()                      Adjust motor speed according to PID output
    reset time_passed_1                        Reset counter for interval
  if (pulses_right_distance >= 3)              Robot heading is adjusted every third time
                                               the encoder ticks
    adjust_moving_robot_heading()              Explained below
      encoders_compass_position_flag = 1       Set flag to determine odometry position
      adafruit_gps_position_flag++             Set flag to determine Adafruit GPS position
      pulses_right_distance = 0                Reset counter for encoder ticks
  if (encoders_compass_position_flag == 1)     Called if the encoder has ticked three times
    encoders_compass_robot_position()          Explained below
      encoders_compass_position_flag = 0       Reset flag to determine odometry position
  if (adafruit_gps_position_flag == 2)         Called if the encoder has ticked six times
    adafruit_gps_position()                    Explained below
      adafruit_gps_position_flag = 0           Reset flag to determine Adafruit GPS
                                               position
      comp_filter_flag = 1                     Set flag to implement complimentary filter
  if (comp_filter_flag == 1)                   Called if the Adafruit GPS position has been
                                               determined
    complimentary_filter()                     Explained below
      comp_filter_flag = 0                     Reset flag to implement complimentary filter
      dist_head_to_wayp_flag = 1               Set flag to calculate distance and heading
                                               to waypoint
```

Figure 3-11: Navigational procedure coding part 1



```
if (dist_head_to_wayp_flag == 1)              Called if the complimentary filter has been
                                              implemented
    distance_heading_to_waypoint()            Explained below
        dist_head_to_wayp_flag = 0            Reset flag to calculate distance and
                                              heading to waypoint
        piksi_gps_flag = 1                    Set flag to determine Piksi GPS position
if (piksi_gps_flag == 1)                      Called if distance and heading to waypoint
                                              has been calculated
    read_piksi_GPS()                          Explained below
        piksi_gps_flag = 0                    Reset flag to determine Piksi GPS position
if (time_passed_2 >= 500 mS)                  Interval on which data is sent over serial
                                              connection
    reset time_passed_2                       Reset counter for interval
    display_serial()                          Data is sent over serial connection
        void_loop_counter = 0                 Reset counter in void loop()
if (distance_to_waypoint < 300000)            Called if the distance to waypoint is less
                                              than 300 mm
    waypoint_reached_flag = 1                 Set flag for waypoint reached
while (waypoint_reached_flag == 1)            Called while the robot is at the waypoint
    stop_robot()                              Both robot wheels are stopped
    read_serial()                             User input is required to determine next
                                              waypoint
    distance_heading_to_waypoint()            Explained below
    adjust_stationary_robot_heading()         Orientate robot towards next waypoint
    reset_variables()                         Reset all variables used
    read_serial()                             Wait for user input to start navigation to next
                                              waypoint
if (last_waypoint_reached == 1)               Called if last waypoint has been reached
    while(1)                                  Hang
```

Figure 3-12: Navigational procedure coding part 2

The void setup(), void loop() and functions written to enable autonomous robot navigation took up a total of 1119 lines, this excludes the libraries written to gather the sensor information. Furthermore the code written to gather the Piksi GPS data had to be altered constantly. This had to be done because when the product was received the SBP (Swift Navigation Binary Protocol) was not set and alterations were still made to it with new firmware iterations. The firmware had to be updated to improve the ability of the Piksi GPS to find an RTK fix.

The basic procedure followed in Figure 3-11 and Figure 3-12 will be described by giving detail to each of the blocks that is not self-explanatory.

void_loop_counter++

Keeps count of the amount of times the void loop() is repeated for each time serial data is written to the laptop via the USB connection.

measure_left_motor_rpm()

```
////////////////////////////////////////////////////////////////////////////////////

void measure_left_motor_rpm()
{
  // Check if standing still
  if (output_left == 0)
  {
    left_motor_rpm = 0;
  }
  // Otherwise
  // RPM updated every 2 ticks, it is 11.8mm
  if (pulses_left >= 2)
  {
    // Detach interrupt while calculating RPM
    detachInterrupt(1);
    left_motor_rpm = (double(pulses_left) * 60000000 /
double(pulses_per_rotation)) / (double(micros() - left_previous_time_rpm));
    left_previous_time_rpm = micros();
    pulses_left = 0;
    attachInterrupt(1, count_pulses_left, CHANGE);
  }
}

////////////////////////////////////////////////////////////////////////////////////
```

To measure the rpm of a motor the first check is to see whether the motor is standing still by looking at the output of the PID loop (`output_left`), if this is zero it means that the motor has been commanded to stop moving or the user has specified the motor speed to be $0\ \mathrm{rpm}$. If the motor is not standing still the motor rpm is updated every second time the encoder ISR (Interrupt Service Routine) is called. It is only updated every second time because it was found that when the motor rpm is calculated every time the encoder ISR is called the rpm calculation is inaccurate. This is due to the resolution limitation on the printed disc. Because a jump to the ISR during the rpm calculation is not desirable the interrupt is disabled for the time being, this interrupt is however still buffered by the hardware. The motor rpm is calculated by first getting the ratio between the number of times the encoder ticked and the total number of encoder pulses per rotation of the wheel. This is then divided by the time that passed since the previous time this calculation was completed and multiplied by $6\,000\,000$ to get from microseconds to minutes. Finally the interrupt is enabled again.

PID_calc()

```
///////////////////////////////////////////////////////////////////////////////////

void PID_calc()
{
  input_left = left_motor_rpm;
  left_PID.Compute();
  input_right = right_motor_rpm;
  right_PID.Compute();
}

///////////////////////////////////////////////////////////////////////////////////
```

Here the input for the PID loop is set and the output computed. The setpoint value can be declared anywhere in the code and the PID calculation is made every $121\ \mathrm{ms}$, using one of the Arduino timers to ensure a periodic interrupt. The PID calculation is made on the exact same time interval that the speed of the motors is adjusted.

adjust_moving_robot_heading()

```
///////////////////////////////////////////////////////////////////////////////////

void adjust_moving_robot_heading ()
{
  previous_heading = current_heading;
  read_compass();
  // Flag to call encoders_compass_robot_position ()
  encoders_compass_position_flag = 1;
   // Flag to call adafruit_gps_position (), called every second time, it is every
   6 ticks
  adafruit_gps_position_flag++;
  int heading_difference = current_heading - required_heading;
  // Adjust for negative degrees
  if (heading_difference < 0)
  {
    heading_difference = heading_difference + 360;
  }
  // Buffer of 10 degrees where robot just keep moving straight
  if (heading_difference <= 180 && heading_difference >= 5)
  {
    setpoint_left - 0.5;
  }
  else if (heading_difference > 180 && heading_difference <= 355)
  {
    setpoint_left + 0.5;
  }
  else
  {
  }
  // Reset counter for pulses_right_distance
  pulses_right_distance = 0;
}

///////////////////////////////////////////////////////////////////////////////////
```

Here the robot heading is altered by adjusting the PID setpoint value for the left motor. First the current compass value is read, then the difference between the previous heading and the current heading is calculated and used to determine whether the robot should turn more to the left or more to the right by altering `setpoint_left`. A $10\,°$ buffer is left within which the robot simply keeps heading in the direction it has been heading. The value of $10\,°$ was experimentally found to be the optimal value to accommodate variances in the compass heading.

encoders_compass_robot_position()

```
////////////////////////////////////////////////////////////////////////////////

void encoders_compass_robot_position ()
{
   // Encoders delta distance assuming that for the delta distance both wheels
   travelled the same distance, this is micro metres (m * 10^-6)
  int encoders_distance = 3 * 5906;
  // Adjust heading for trigonometry calculations
  previous_heading = previous_heading - 90;
  if (previous_heading >= 180)
  {
    previous_heading = previous_heading - 360;
  }
  previous_heading = previous_heading * -1;

  // Robot position in micrometres with conversion from degrees to radians
  robot_x_position_encoders = robot_x_position + encoders_distance *
cos((previous_heading * 71) / 4068);
  robot_y_position_encoders = robot_y_position + encoders_distance *
sin((previous_heading * 71) / 4068);

  // Reset flag for intermediate encoders compass position update
  encoders_compass_position_flag = 0;
}

////////////////////////////////////////////////////////////////////////////////
```

The robot's position according to odometry measurements is updated every third time the encoder ticks, meaning it is updated every $3\,(5{,}906) = 17{,}718\ \mathrm{mm}$ the robot travels. To determine the [x,y] coordinates of the robot the compass measurements must be adjusted from the axis system where north is zero degrees and clockwise positive (shown on the left in Figure 3-13) to an axis system where east is zero degrees and anti-clockwise is positive (shown on the right in Figure 3-13).

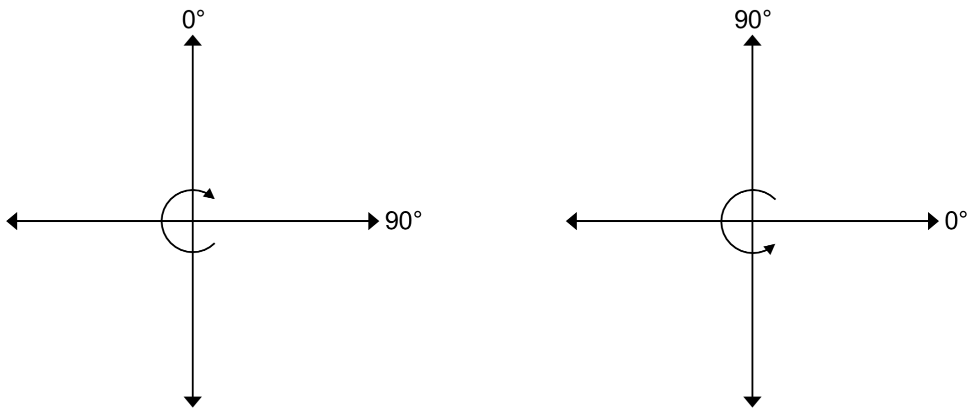


Figure 3-13: Axis system conversion

Simple sin() and cos() angles are used to determine the [x,y] coordinates of the robot. The built-in Arduino trigonomic functions work in radians while the compass output is in degrees, thus the $(previous\_heading\,(71)/4\,068)$ conversion is used to get to radians.

adafruit_gps_position()

```
////////////////////////////////////////////////////////////////////////////////

void adafruit_gps_position ()
{
  // Get new GPS position
  read_adafruit_GPS();

  // Next the distance and heading from base to current gps position is calculated
  double distance_to_destination = TinyGPSPlus::distanceBetween(
base_station_latitude, base_station_longitude, -1 *
convertDegMinToDecDeg(GPS.latitude), convertDegMinToDecDeg(GPS.longitude));
  double course_to_destination = TinyGPSPlus::courseTo(base_station_latitude,
base_station_longitude, -1 * convertDegMinToDecDeg(GPS.latitude),
convertDegMinToDecDeg(GPS.longitude));
  // Convert distance to destination from m to micrometres
  unsigned long adafruit_gps_distance = distance_to_destination * 1000000;
   // Total x and y distance from base to current adafruit gps location in
   micrometres
   robot_x_position_adafruit_gps = waypoint_0_x + adafruit_gps_distance *
   cos((course_to_destination * 71) / 4068);
   robot_y_position_adafruit_gps = waypoint_0_y + adafruit_gps_distance *
   sin((course_to_destination * 71) / 4068);
  // Reset flag for intermediate adafruit gps position update
  adafruit_gps_position_flag = 0;
  // Set flag that will enable complimentary filter
  comp_filter_flag = 1;
}

////////////////////////////////////////////////////////////////////////////////
```

The robot's position according to the Adafruit GPS is calculated by using the two functions `distanceBetween` and `courseTo` from the TinyGPSPlus library. These two functions take the latitude (χ) and longitude ($\Delta\lambda$) of the origin and latitude and longitude of the goal as inputs and give the distance (d) and heading ($\Delta\delta$) between these points back. The inputs must be signed decimal-degree values. The formula used calculates the great-circle distance, which is the shortest distance between two points when one moves on the surface of a sphere. The Vincenty formula (Vincenty, 1975) can be used to calculate this distance as shown in equation 3.5:

$$\Delta\sigma = \tan^{-1}\left(\frac{\sqrt{(\cos\chi_2 \sin\Delta\lambda)^2 + (\cos\chi_1 \sin\chi_2 - \sin\chi_1 \cos\chi_2 \cos\Delta\lambda)^2}}{\sin\chi_1 \sin\chi_2 - \cos\chi_1 \cos\chi_2 \cos\Delta\lambda}\right) \tag{3.5}$$

Then:

$$d = r\,\Delta\sigma \tag{3.6}$$

The formula used by `courseTo` to determine the heading from one set of coordinates to another is given by:

$$\Delta\delta = \tan^{-1}\left(\frac{\sin\Delta\lambda\ \cos\chi_2}{\cos\chi_1\ \sin\chi_2 - \sin\chi_1 \cos\chi_2 \cos\Delta\lambda}\right) \tag{3.7}$$

Now the [x,y] coordinates of the robot according to the Adafruit GPS can be determined using simple sin() and cos() angles.

complimentary_filter()

```
////////////////////////////////////////////////////////////////////////////////

void complimentary_filter ()
{
  double odometry_const = 99.999;
  double gps_const = 100 - odometry_const;
  // Split division for more accurate results
  robot_x_position = (robot_x_position_encoders * odometry_const) / 100 +
(robot_x_position_adafruit_gps * gps_const) / 100;
  robot_y_position = (robot_y_position_encoders * odometry_const) / 100 +
(robot_y_position_adafruit_gps * gps_const) / 100;
  // Reset flag
  comp_filter_flag = 0;
  // Flag to get new distance and required heading to waypoint
  dist_head_to_wayp_flag = 1;
}

////////////////////////////////////////////////////////////////////////////////
```

The complimentary filter is implemented here, with a percentage defined by the user that determines how much each of the odometry and Adafruit GPS robot position estimations will be trusted.

distance_heading_to_waypoint()

```
////////////////////////////////////////////////////////////////////////////////

void distance_heading_to_waypoint ()
{
  double delta_y = (double) waypoint_y - (double) robot_y_position;
  double delta_x = (double) waypoint_x - (double) robot_x_position;
  required_heading = atan2(delta_y, delta_x);
  // Use sin or cos depending on which is more accurate
  if ((required_heading >= -1 * PI / 4 && required_heading <= PI / 4) ||
(required_heading >= 3 * PI / 4 && required_heading <= -3 * PI / 4))
  {
    distance_to_waypoint = delta_x / cos(required_heading);
  }
  else
  {
    distance_to_waypoint = delta_y / sin(required_heading);
  }
  // Convert to degrees
  required_heading = (required_heading * 4068) / 71;
  // Convert to coordinate system where North denotes zero degrees
  required_heading = required_heading - 90;
  if (required_heading >= 180)
  {
    required_heading = required_heading - 360;
  }
  required_heading = required_heading * -1;
  // Adjust for a range of 0 - 360 degrees
  if (required_heading < 0)
  {
    required_heading = required_heading + 360;
  }
  dist_head_to_wayp_flag = 0;
  // Flag to save piksi coordinates
  piksi_gps_flag = 1;
}

////////////////////////////////////////////////////////////////////////////////
```

This function calculates the distance and heading from the current robot position to the next waypoint. Through tests it was seen that when only one trigonometric function was used to calculate the distance to the waypoint, there were times when one of the axes values became so small that it resulted in large variations in the `distance_to_waypoint` value. Therefore both sin() and cos() functions were used depending on whether the angle was smaller or greater than $45°$.

read_piksi_GPS()

```
////////////////////////////////////////////////////////////////////////////////

// This function reads each byte and is analysed according to the supplied SPB
message system
void read_piksi_GPS ()
{
  piksi_message_flag = 0;
  // While data is being sent from Piksi
  while (Serial3.available () > 0 && piksi_message_flag == 0)
  {
    // Read incoming byte
    if (piksi_message_flag == 1)
    {
      piksi_gps_flag = 0;
      break;
    }
    piksi_message_flag = processIncomingByte (Serial3.read ());
  }
}

// This function reads each byte and is analysed according to the supplied SPB
message system
boolean processIncomingByte (const byte inByte)
{
  // Input message
  static byte input_msg [MAX_INPUT];
  // Hold position in message
  static unsigned int input_pos = 0;
  // Correct message received flag
  boolean piksi_message_flag = 0;

  // 0x55 shows the start of a message
  if (inByte == 0x55)
  {
    // When last byte of redundancy check has been reached
    if (input_pos == 29)
    {
      // This is the hex code for a MSG_BASELINE_NED message
      if ((input_msg[0] == 0x03) && (input_msg[1] == 0x02))
      {
        // Another function to extract data from the MSG_BASELINE_NED message
        msg_analyse(input_msg);
        piksi_message_flag = 1;
      }
    }
    // Reset the message position holder
    input_pos = 0;
  }

  // The else captures the other 29 bytes excluding the start byte of 0x55
  else
  {
    // Simple check to ensure a start byte has not been skipped
    if (input_pos < (MAX_INPUT - 1))
    {
      input_msg[input_pos] = inByte;
      input_pos = input_pos + 1;
    }
  }
  return piksi_message_flag;
}

// Function to analyse the received message
void msg_analyse (byte byte_msg[29])
{
  // Check for fix type, 0x00 = float RTK, 0x01 = fixed RTK
  fix_mode_type = byte_msg[26];
  // Shows nr of satellites
  byte nr_satellites = byte_msg[25];
  double angle = 0;
  double distance = 0;
  // Vertical deviation from base with conversion from bytes to integer
  robot_y_position_piksi_gps = bytesToInt(byte_msg[12], byte_msg[11],
byte_msg[10], byte_msg[9]) * 1000;
  // Horizontal deviation from base with conversion from bytes to integer
  robot_x_position_piksi_gps = bytesToInt(byte_msg[16], byte_msg[15],
byte_msg[14], byte_msg[13]) * 1000;
}

// Function to convert from bytes to integer
long bytesToInt (int b4, int b3, int b2, int b1)
{
  long result = 0;
  result = (long)b4 << 24;
  result += (long)b3 << 16;
  result += (long)b2 << 8;
  result += (long)b1;
  return result;
}

////////////////////////////////////////////////////////////////////////////////
```

To communicate with the Piksi GPS, code had to be written to implement the SBP. All data transmitted by the Piksi GPS consists of a message structure that has the format shown in Table 3-8:

Table 3-8: SBP (Swift Navigation Binary Protocol) message structure

Size (bytes)	Name	Description
1	Preamble	States start of message with $0x55$
2	Message type	Shows payload contents
2	Type Sender	ID of device sending message
1	Length	Define amount of bytes in payload
M	Payload	Binary content
2	CRC	Cyclic Redundancy Check

Only messages of type $0x0203$ are of interest, these messages include data about the GPS time, the north, east and down coordinates from the base station to the rover receiver, the number of satellites used to obtain the result and the type of fix achieved. A message of type $0x0203$ has 22 bytes, thus the total number of bytes received up to this point is $27\ (22 + 1 + 2 + 2)$. When the 29th byte is received the code knows the last byte of the cyclic redundancy check has been received and the message can be processed. The fix type and number of satellites used in the solution are both binary numbers and can be directly captured, while the north and east deviation values each consists of 4 bytes that must first be converted to integer values as shown in the code above.

4 Experimental Setup

First measurement uncertainty will be discussed. Next the two setups with different maps will be detailed. The first setup was used to capture only the RTK GPS data. The second setup was used to capture positional data for all sensors on the robot.

4.1 Measurement uncertainty

Before proceeding to the experimental setup the topic of measurement uncertainty will be discussed to get an idea of how reliable real world measurements are.

4.1.1 Overview

For the experimental setup various measurements (distance, time, heading) will be taken, and wherever measurements are taken there is an uncertainty as to the trueness of the measurement. This uncertainty can be expressed by two values, the interval or margin within which repeated measurements of the same object fall and the confidence level of the measurements falling in this interval.

If there is a change in repeated readings one can use an average as an estimate of the actual value. The more measurements one is able to take the better this estimate will be. To get an idea of the spread of the measurements one can get the standard deviation for all the measurements. To get the standard deviation for a set of measurements one takes the sum of the square of the difference between the average (x_{avg}) and each measurement (x_j) and divides this by the total number of measurements (n_m) -1. The standard deviation (σ) is then given by:

$$\sigma = \sqrt{\frac{\sum_{j-1}^{n_m}(x_j - x_{avg})^2}{n_m - 1}} \tag{4.1}$$

4.1.2 Origin

For this thesis one particular measurement will be investigated to determine its uncertainty. A class 1 $2{,}5\ m$ tape measure is used to measure the length of a steel guide wire over a maximum distance of $120\ m$. The errors and uncertainties present in this measurement can be broken into the following as given in (Bell, 1999):

1. The measurement instrument itself - a tape measure will be used to measure distances. From (Thetapestore.co.uk, 2016) a $2{,}5\ m$ class 1 measurement tape can only be accurate to $\mp\ 0{,}35\ mm$.

2. The item measured - a steel guide wire will be measured, but since it is lying on a grass surface that is not flat the measurement cannot be completely true.
3. The measurement process - the wire is in excess of $100\,m$, thus the $2{,}5\,m$ measurement tape will build up an error with consecutive measurements.
4. The operator skill - because the measurements must be taken using visual confirmation a parallex error can be introduced.
5. The environment - since measurements will be taken outside in the sun and wind, with varying temperatures, this can have an effect on the operator and measurement tape.

4.1.3 Measuring uncertainty

Before calculating measurement uncertainty one must identify all the sources of uncertainty, then estimate the contribution of each of these sources and finally combine these to get an overall uncertainty. The process of measuring uncertainty can be broken into the following eight steps as described in (JCGM 100:2008, 2008):

1. Make a decision regarding the exact information that is required from the measurements to get a result. This will include data regarding errors in the measurement device, errors in the object being measured and errors introduced in the process of measuring.

2. Next take the measurements while taking note of when it was done, under what conditions it was done and the exact measurement device used. Calculate the mean and standard deviation for all the measurements taken.

3. Make an estimate of the uncertainty of each parameter that will have an affect on the final measurement estimate. To be able to combine these uncertainties later, one has to convert each uncertainty to a value around the standard deviation, also called a standard uncertainty. A coverage factor is used to do this conversion. A coverage factor of 2 will give a confidence level of $95\,\%$ (Physics.nist.gov, 2016) assuming the combined standard uncertainty has a normal distribution. The errors and uncertainties present were identified earlier and can be assigned values here:

    - The $2{,}5\,m$ class 1 measurement tape can only be accurate to $\mp\,0{,}35\,mm$ as stated above. Thus over $2{,}5\,m$ with a coverage factor of 2 the standard uncertainty is $0{,}175\,mm$.
    - The smallest measurement possible on the tape is in millimeter. Thus the reading can fall anywhere within this $1\,mm$ interval, or $\mp\,0{,}5\,mm$. The distribution of these measurements is uniform and from (Bell, 1999) the standard uncertainty is the half-width ($0{,}5\,mm$) divided by $\sqrt{3}$, giving $0{,}28\,mm$.

- Since the wire does not lie straight it is assumed that the length is underestimated by $0{,}1\,\%$. Over a $2{,}5\,m$ distance this is $2{,}5\,mm$. Again the uncertainty is assumed to be uniform, with the half-width being $2{,}5\,mm$ and dividing this by $\sqrt{3}$ the standard uncertainty is $1{,}44\,mm$.

4. Decide whether each of the parameters listed in point 3 is independent of one another, if this is not the case this dependency needs to be calculated and added as another parameter. In this case all parameters are independent.

5. Determine the result of the measurement by taking the mean measurement value and adding all known corrections. In this case the corrections only include the wire that is not completely straight. Thus the wire measured length over $2{,}5\,m$ should be $2500\,mm + 2{,}5\,mm = 2502{,}5\,mm$.

6. Now the combined standard uncertainty can be determined by taking the square root of the squared individual uncertainties.

$$\begin{aligned} Combined\ standard\ uncertainty &= \sqrt{0{,}175^2 + 0{,}28^2 + 1{,}44^2} \\ &= 1{,}48\,mm \end{aligned} \tag{4.2}$$

7. Now this uncertainty can be given in terms of a coverage factor of 2, meaning the uncertainty is $2\,(1{,}48\,mm) = 2{,}95\,mm$ with a confidence level of $95\,\%$.

8. Finally the measurement uncertainty can be written down. Up to step 7 all calculations worked over a length of $2{,}5\,m$, multiply this by 48 to get to $120\,m$ resulting in a new uncertainty of $141{,}6\,mm$. Thus it can be said that the length of the wire is $120\,m \mp 0{,}142\,m$ for a coverage factor of 2, resulting in a confidence level of $95\,\%$.

4.2 RTK GPS positional data

For this setup the base station was fixed to a stationary point as shown in Figure 4-1, which is also known as waypoint_0 as shown in Figure 4-3. To prevent interference and to get the best possible GPS readings the base station antenna was mounted at a height of $2{,}5\ \mathrm{m}$ above ground level. This is an active antenna meaning it has an LNA (Low-Noise Amplifier) that lowers the noise picked up by the receiver. The 3DR telemetry radio enables a direct line of communication between the base station and rover modules. The robot with the rover RTK GPS as shown in Figure 4-2 was then moved at a constant velocity of $0{,}313\ \mathrm{m \cdot s^{-1}}$ (or $30\ \mathrm{rpm}$ maintained within $1\ \mathrm{rpm}$) between the waypoints while capturing its location.

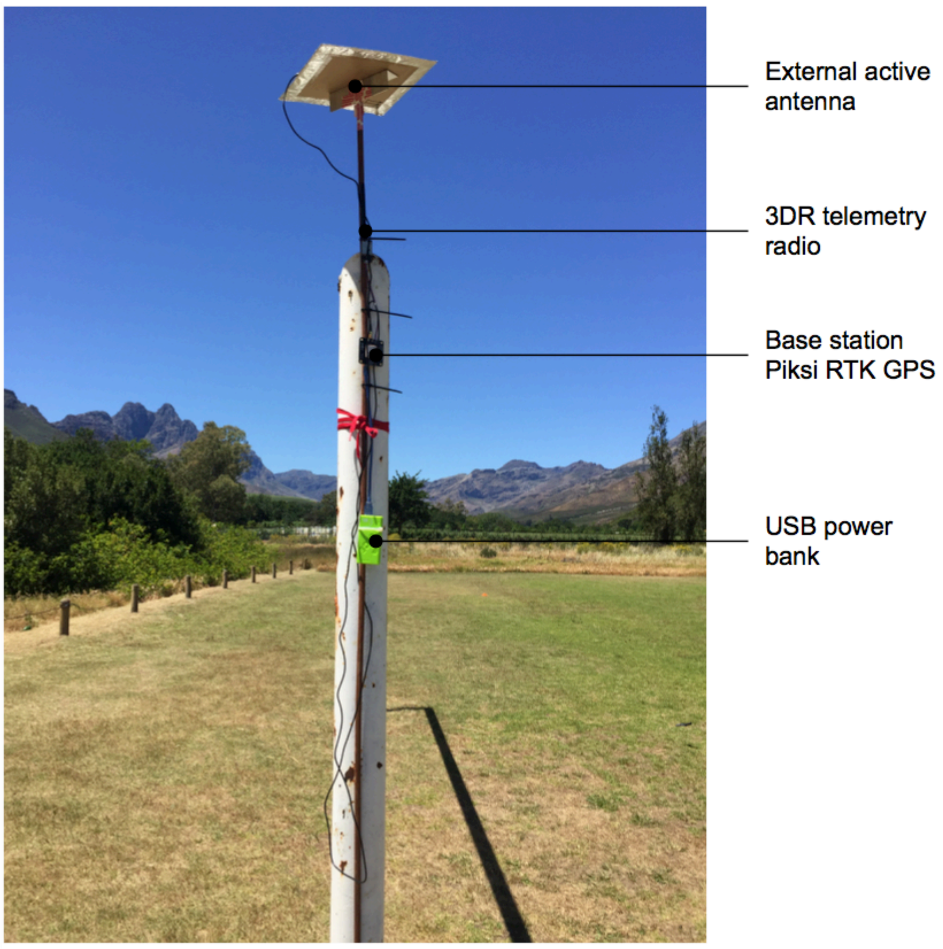


Figure 4-1: Base station Piksi RTK GPS with external power source and active antenna

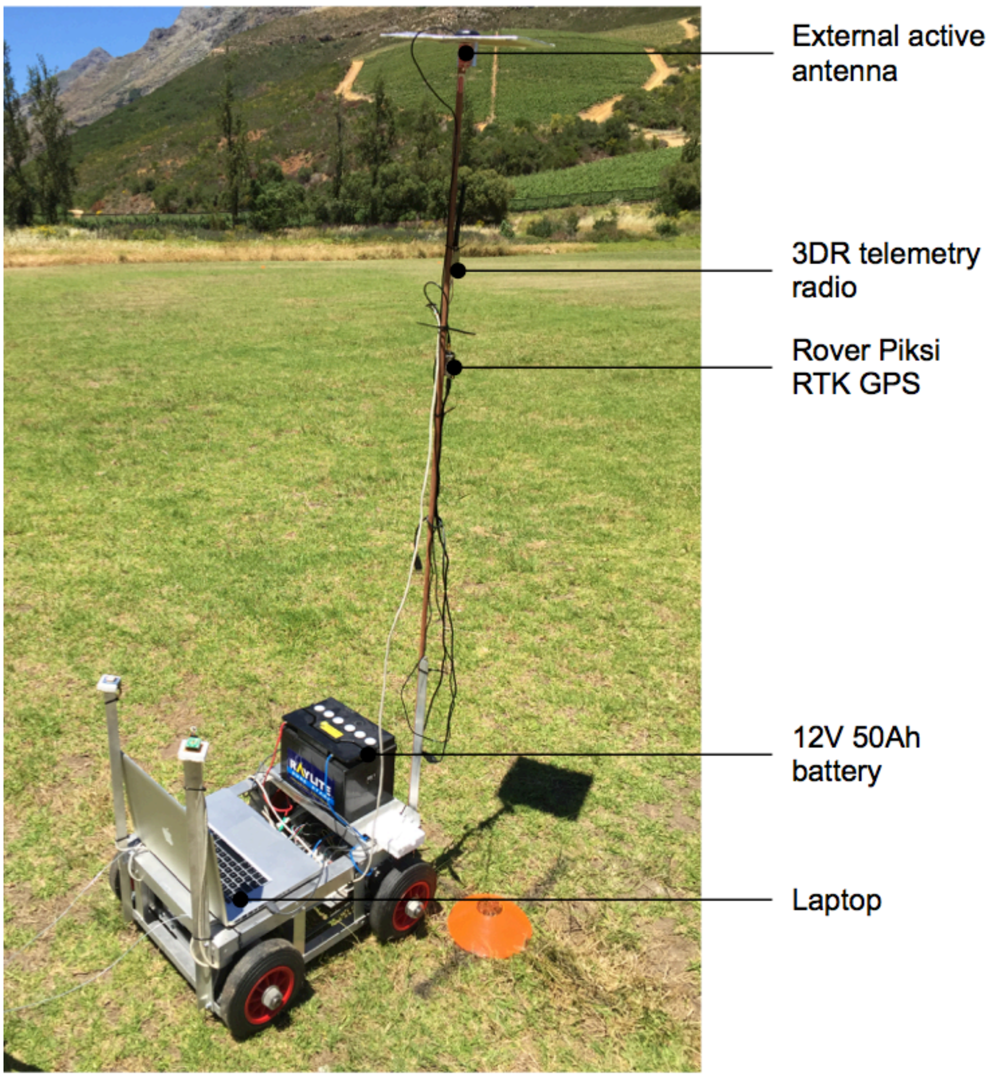


Figure 4-2: Robot with rover Piksi RTK GPS, active antenna, laptop and battery

Figure 4-3 shows the four waypoints between which the navigation was conducted. These waypoints were conveniently chosen as poles of four goal posts present on a sports field. The base station was fixed at waypoint_0 and the rover then moved from this waypoint to the next in a straight line (by guiding the robot platform on a fish line connected between all the waypoints) until it returned to the base station.

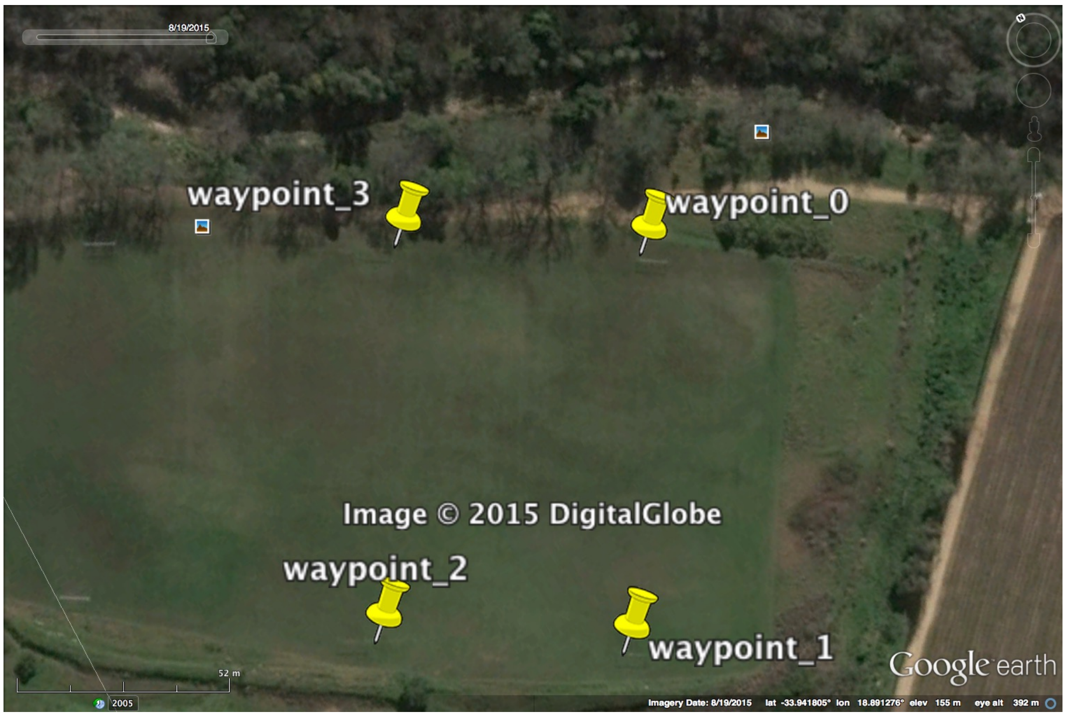


Figure 4-3: Waypoints for navigation[9]

To determine the coordinates of the waypoints a wire (that is used to guide the watering machine on the field) was used to measure the distance between waypoints to obtain the values in Table 4-1:

Table 4-1: Distances between waypoints

Waypoints	Distance (m)	Measurement uncertainty (m)
Waypoint 0 to 1	99,47	0,117
Waypoint 0 to 2	117,78	0,138
Waypoint 1 to 2	63,07	0,075
Waypoint 0 to 3	62,84	0,075
Waypoint 2 to 3	99,62	0,118

As discussed in Section 4.1 the distance measurements in Table 4-1 is given at a confidence level of $95\ \%$ and calculated by using the result of Section 4.1. Using this data triangulation was used to determine the relative position of each waypoint from waypoint_0.

[9] Adapted from Google Earth, 2015, https://www.google.com/earth/explore/products/plugin.html

4.3 All sensors positional data

A new map as shown in Figure 4-4 was navigated next because of the distance limitations of the 3DR telemetry radios that will be discussed in Section 5.1. These waypoints were measured out using the Piksi RTK GPS after the navigational precision of this GPS was confirmed. The map was navigated at a velocity of $0{,}313\ \mathrm{m \cdot s^{-1}}$ with the robot starting at waypoint_0 and the base station mounted on the pole at the origin of the coordinate system.

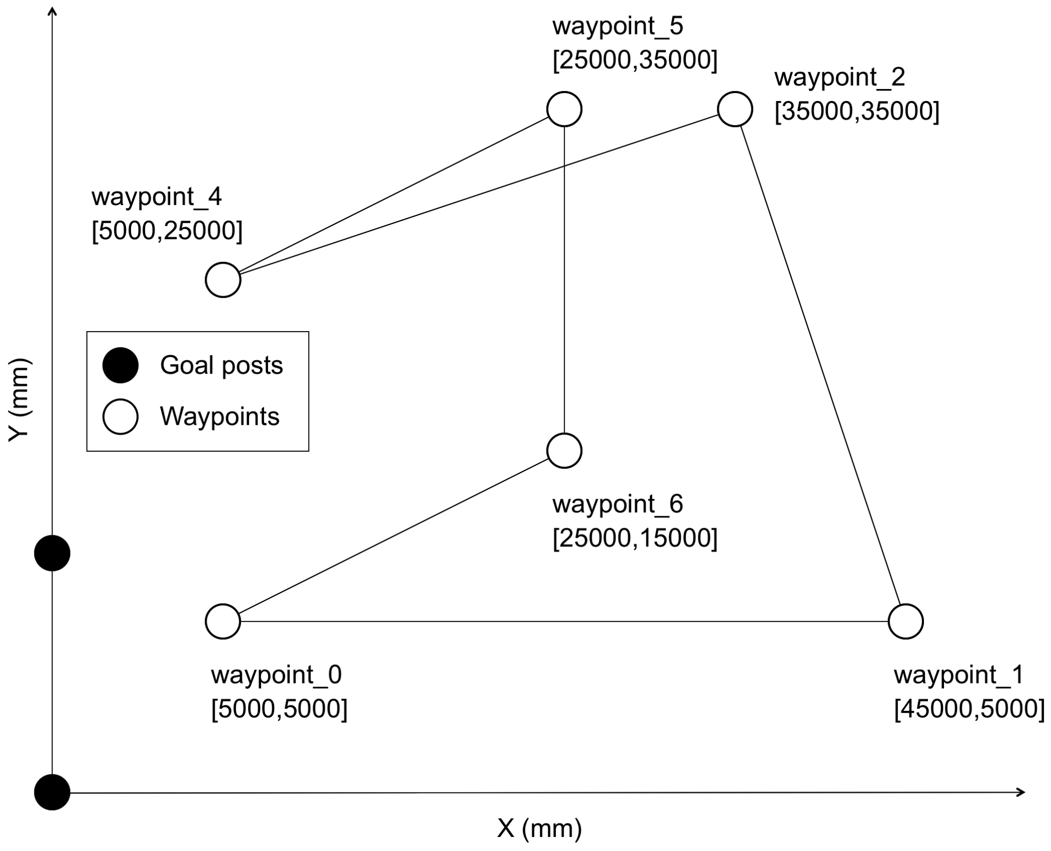


Figure 4-4: Map to be navigated

Figure 4-5 shows the robot, base station and starting waypoint on the field. The robot was then navigated from waypoint to waypoint, using the code described in Section 3.2, until it returned to waypoint_0 multiple times for all the case studies conducted.

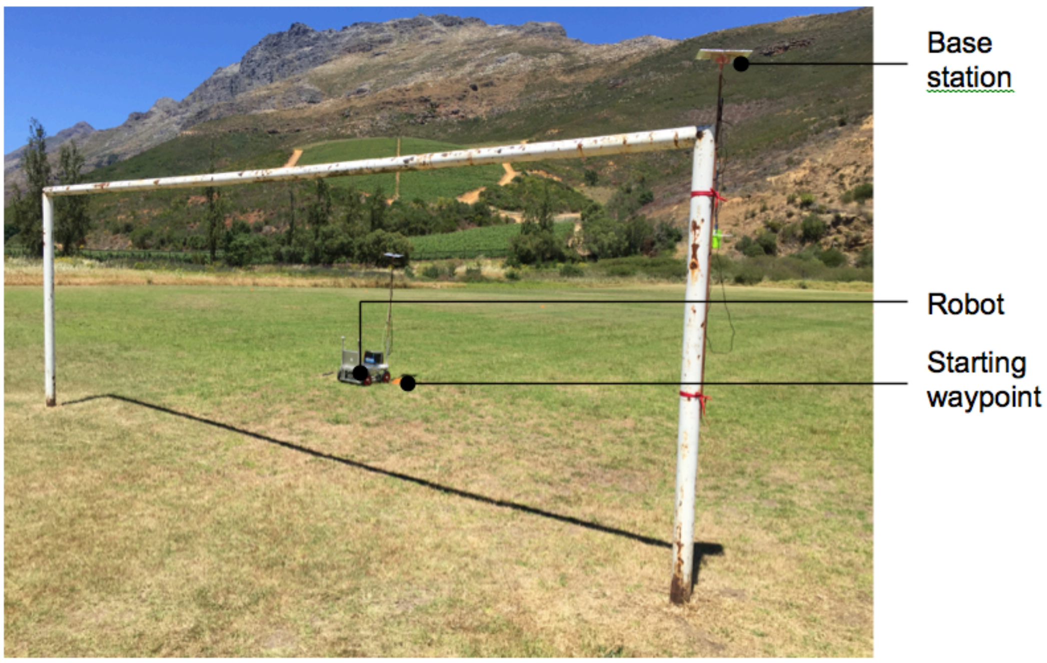


Figure 4-5: Complete experimental setup

5 Results

The results section will be split into two main parts. Firstly the RTK GPS test results are summarized, this is done first to confirm the accuracy of the RTK GPS since it will be used as a way of verifying the position of the robot. Secondly, the positional data of the robot using the encoders, tilt-compensated compass and Adafruit GPS will be shown. Seven case studies will be made, and the relative accuracy of this positional data will be determined by comparing it to the RTK position data.

5.1 RTK GPS positional data

The Piksi RTK GPS datasheet (Piksi Datasheet, 2013) claims that "centimetre accurate relative positioning" is possible. This is the only claim that can be found regarding the accuracy of the GPS. A study will be conducted to confirm this claim and determine the real world relative accuracy of the GPS. Before continuing it is important to have a clear understanding of the terms accuracy, relative accuracy and precision. In terms of position a measurement can be called accurate when it is close to the real position. Relative accuracy and precision are the same and mean that with repeated measurements the position will stay constant if the rover and base station are kept stationary.

The robot is moved as described in Section 4.2 through the map of Figure 4-3 and the result of the first run of five is shown in Figure 5-1, the plots of the other runs can be seen in Appendix A. The output of the RTK GPS is in units of m north and east of the base station. During each of the runs at least eight satellites were visible. The gaps show the positions where the RTK GPS fix was lost and the robot kept on moving in a straight line with constant velocity until the signal was re-established.

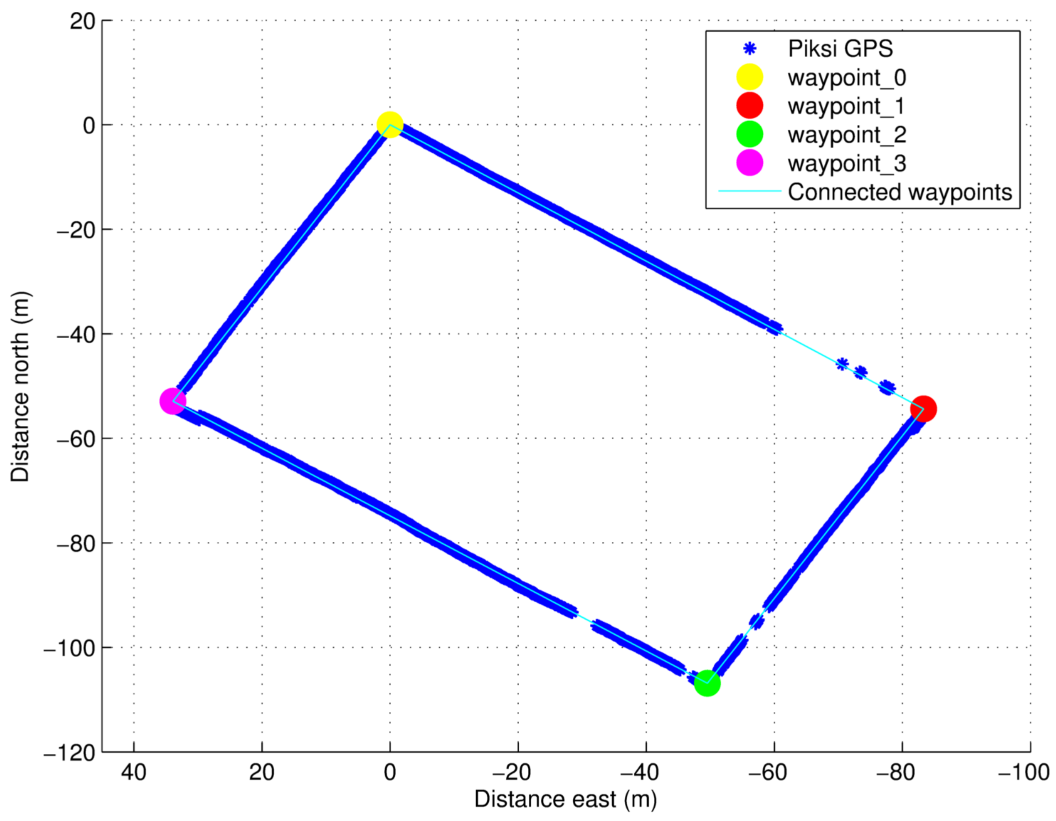


Figure 5-1: Run 1 RTK GPS positional data

Since the main concern is the two-dimensional GPS accuracy the DRMS (Distance Root Mean Square) will be calculated, which is the square root of the average of the squared horizontal deviations. The values have a probability of $65\ \%$ of falling in the probability circle. The DRMS is shown in equation 5.1:

$$DRMS = \sqrt{{\sigma_x}^2 + {\sigma_y}^2} \tag{5.1}$$

Since the velocity and exact time of measurements are known, the theoretical location of the robot is known and can be compared to the location measured by the Piksi GPS. The distance between each of the measured values and actual location values is taken and averaged for the complete path travelled. This is done for all five runs to get the following average deviations:

Table 5-1: Average deviations from actual path

	DRMS error (mm)
Run 1	$180{,}90$
Run 2	$222{,}40$
Run 3	$295{,}50$
Run 4	$269{,}20$
Run 5	$484{,}00$

Thus the average DRMS value for the Piksi RTK GPS is as in equation 5.1:

$$\begin{aligned} DRMS &= \sqrt{{\sigma_x}^2 + {\sigma_y}^2} \\ &= \sqrt{{\sigma_r}^2} \\ &= 290{,}4\ \mathrm{mm} \end{aligned} \tag{5.2}$$

It must be noticed that this DRMS value of $290.4\,mm$ is subject to the measurement uncertainty discussed in Section 4.1. A further note that must be made is that because of the terrain on which the robot moves there is roll and pitch movement present that has an effect on the position of the RTK GPS. This effect is further increased by the fact that the rover Piksi GPS module is positioned at a higher level than the robot platform as shown in Figure 4-2.

After testing the accuracy of the RTK GPS, information regarding this accuracy was also published by another source (Hirt, 2015). The standard deviation was found to be between 94 to $129\,mm$ when walking in a circle of 20 to $30\,m$. This error is smaller because their test setup ensured more accurate measurements. Thus from this point forward the RTK GPS will be used as a basis for the precision measurements of the other sensors.

An interesting phenomenon that is noticed in Figure 5-1 is that the RTK fix is lost at positions only far from the starting waypoint. This is caused by a break in communication between the two station radios. While the RTK fix is still present, the stations are not able to exchange information at that moment. When the robot is moving from waypoint_0 to waypoint_1 and from waypoint_1 to waypoint_2 the radio on the rover is positioned in a way that a direct line of sight is not present to the base station. This is the cause of the missing measurements on all five runs between those waypoints.

5.2 All sensors positional data

The seven case studies that will be examined can be divided into the following:

1. A comparison in relative navigational accuracy by navigating the same map multiple times.
2. A comparison of relative navigational accuracy when the complimentary filter values are altered.
3. A comparison of relative navigational accuracy when differential GPS (EGNOS) is enabled and disabled on the Adafruit GPS module.
4. A comparison of relative navigational accuracy when the ground speed of the robot is increased.
5. The relative navigational accuracy when wheel slippage is introduced.
6. The relative navigational accuracy when magnetic interference and GPS drift is present.
7. The relative navigational accuracy when one or multiple sensors fail.

From the results of Section 5.1 it was decided to use the Piksi RTK GPS as a way of verifying the robot position. For all the above case studies the Piksi RTK GPS will be used as the sole sensor for navigating the robot between waypoints, with the other sensors capturing data under the different scenarios. All values will be displayed in millimeter although the measurement uncertainty as discussed in Section 4.1 will make these results less accurate.

5.2.1 Case study 1: Multiple runs

To determine whether measurements are consistant over multiple runs the same map will be navigated five times and the Adafruit GPS data, odometry data and Piksi RTK GPS data will be compared to determine whether data captured in consequent tests can be compared to one another. Figure 5-2 shows a plot of the positional data where the distance north corresponds to the Y-axis and the distance east to X-axis in Figure 4-4.

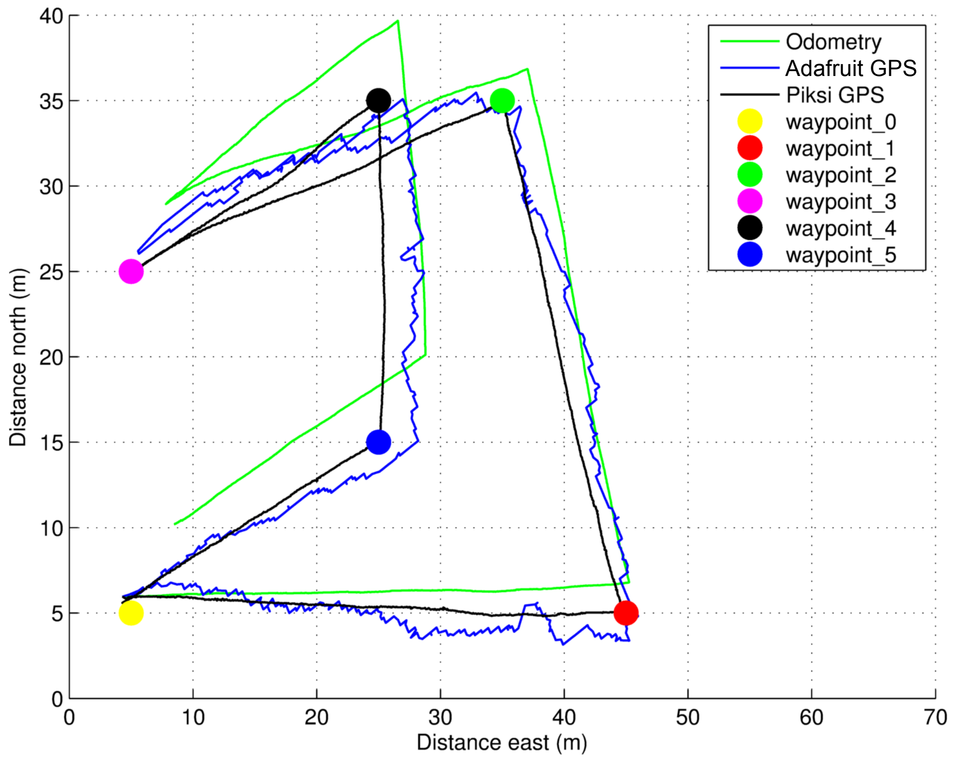


Figure 5-2: Case study 1 navigated map

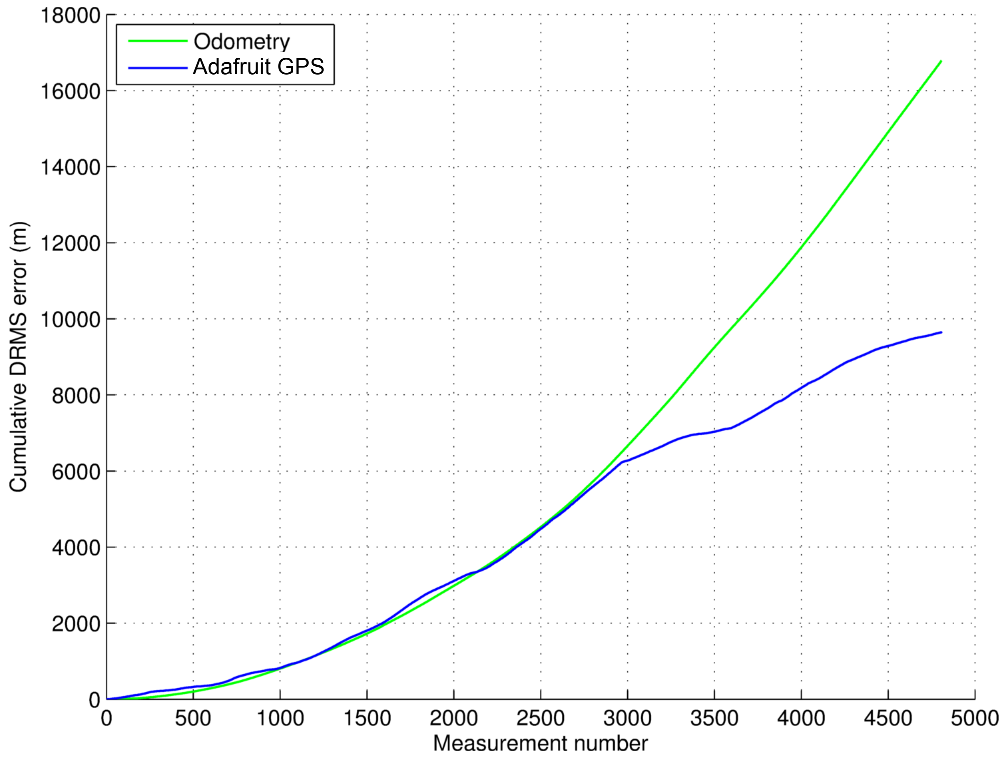


Figure 5-3: Case study 1 cumulative DRMS error

From Figure 5-2 it is clear that the odometry readings (which is the combination of encoder and compass measurements to get heading and distance for each increment) start drifting from the Piksi RTK GPS readings as the robot progresses through the map. The Adafruit GPS readings show less relative accuracy than the odometry readings but are on the long run more precise. To visualize the drifting effect one can plot the cumulative DRMS error value against the measurements taken, this is shown in Figure 5-3.

From Figure 5-3 it is clear that as the robot progresses the odometer error gets larger, confirming the drift, while the Adafruit GPS error does not follow the same linear trend, thus confirming it is more precise. To get a comparative idea of the DRMS errors, the mean of the cumulative DRMS error can be calculated for each of the five runs. This is shown in Table 5-2 with the plots of the other four runs documented in Appendix B.

Table 5-2: Case study 1 DRMS errors

Run	DRMS error (mm)	
	Odometry	Adafruit GPS
1	3 491	2 006
2	935	2 822
3	1 156	2 487
4	2 828	1 756
5	1 612	3 326
Mean	2 004	2 479

From Table 5-2 one can see the odometry errors vary by $2\,556\,\mathrm{mm}$. It can visually be confirmed that runs 1 and 4 show the largest error due to incorrect heading readings in the odometry calculation. Over all the runs the Adafruit GPS DRMS error varies by $1\,570\,\mathrm{mm}$ with run 5 showing the largest error because of erroneous readings between waypoints 3 and 4. Since regular GPS has a pseudo range accuracy of $7{,}8\,\mathrm{m}$ as stated earlier, the largest error of $3\,326\,\mathrm{mm}$ is acceptable. The mean odometry error is also less than that of the stated GPS error, thus in the further case studies data will be compared from different runs to one another with a higher certainty than the stated accuracy of the most inaccurate single sensor (Adafruit GPS with a stated position accuracy of less than $3\,\mathrm{m}$) on the robot.

5.2.2 Case study 2: Altering complimentary filter constant

The complimentary filter constant defines the percentage that the odometer and Adafruit GPS readings are trusted. A value of $0{,}99$ means that $99\,\%$ of the Adafruit GPS coordinates are used, and $1\,\%$ of the odometry coordinates. To determine the best constant the cumulative DRMS error will be calculated for different constant values. By plotting this data for three different runs it was seen that the optimal constant value lies between 0 and $0{,}02$. This interval will be evaluated to find the optimal complimentary filter constant. The data for the runs was only captured between waypoint_0 and waypoint_1 because when a change in robot heading is introduced there is a possibility that some of the odometry error can be cancelled out. This is not ideal in determining the filter constant that will also be used in cases when the odometry error is not cancelled out. All three cases showed odometry drift and Adafruit GPS readings within the stated position accuracy, thus an ideal scenario for implementing the complimentary filter.

Figure 5-4 shows the cumulative DRMS error for complimentary filter constants in the 0 to $0{,}02$ interval for the first of the three runs.

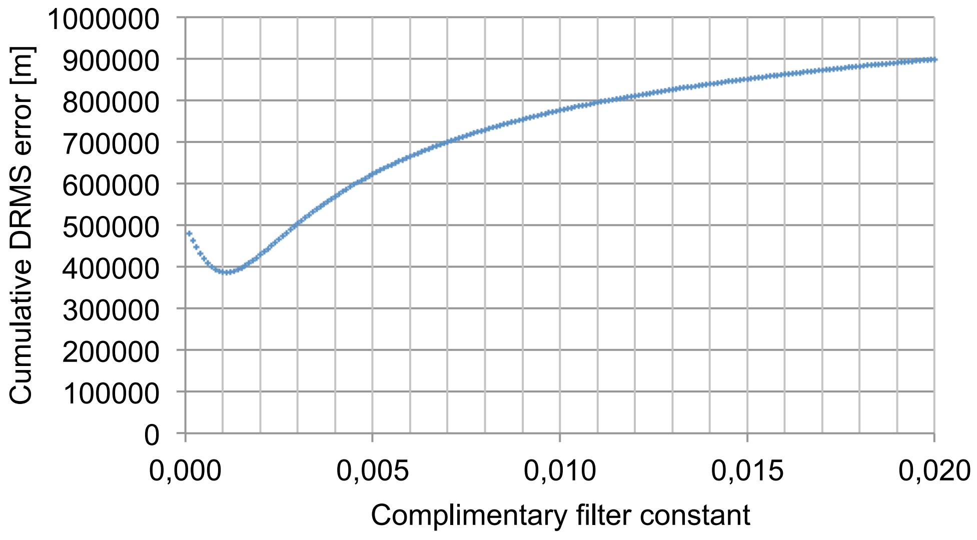


Figure 5-4: Cumulative DRMS error for different complimentary filter constants

From Figure 5-4 it can be seen that a constant of $0{,}001$ gives a minimum cumulative DRMS error. Figure 5-5 shows the navigated map for a constant of $0{,}001$ between waypoint_0 and waypoint_1, and Figure 5-6 the cumulative DRMS error.

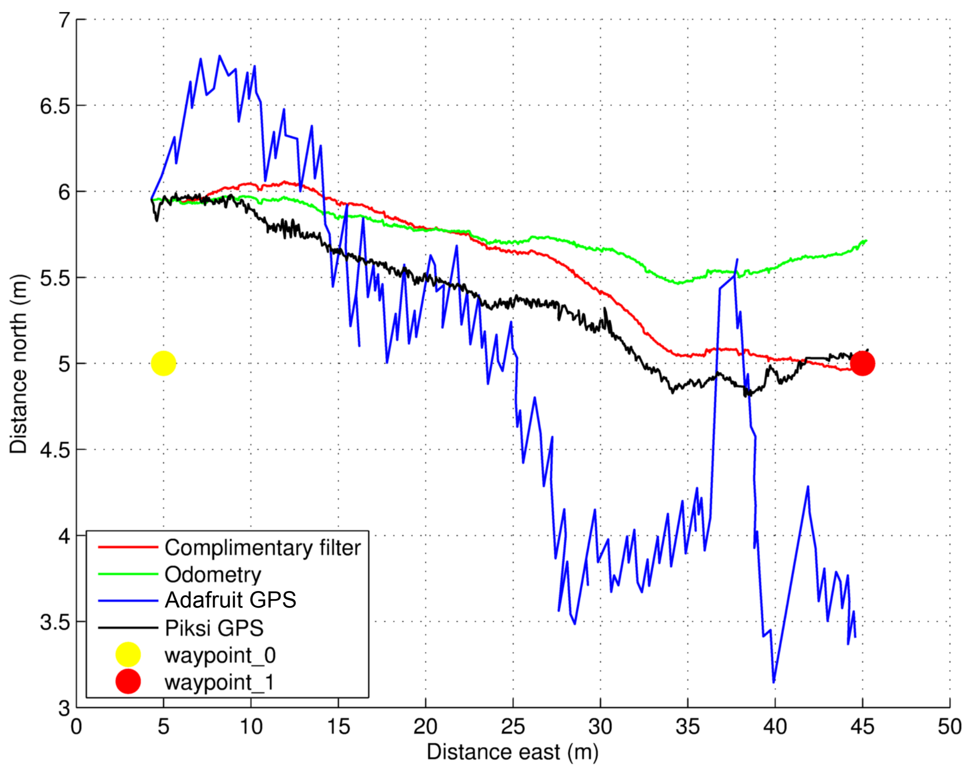


Figure 5-5: Case study 2 navigated map

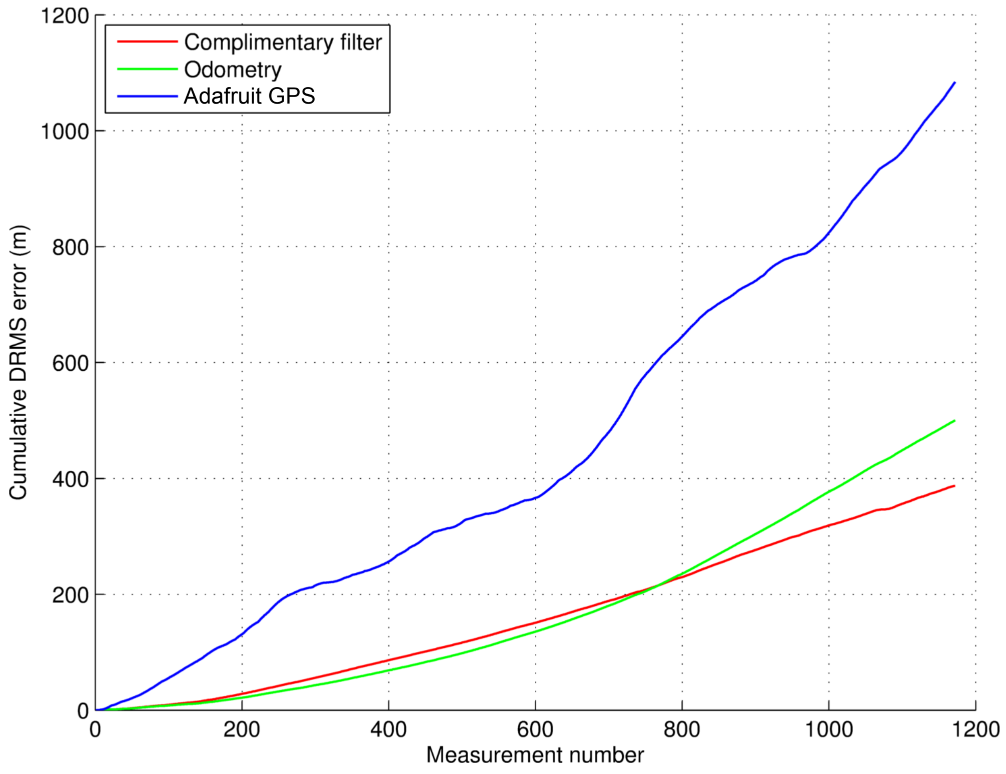


Figure 5-6: Case study 2 cumulative DRMS error

Appendix C contains the cumulative DRMS error for complimentary filter constants in the 0 to $0{,}02$ interval for the other two runs. For these runs the optimal complimentary filter constant was also found to be around $0{,}001$. The navigated maps and cumulated DRMS errors for these runs at the optimal complimentary filter constants are also included in Appendix C. Table 5-3 shows a summary of the mean accumulated DRMS errors for the three runs with a complimentary filter constant of $0{,}001$. All further case studies will use a complimentary filter constant of $0{,}001$.

Table 5-3: Case study 2 DRMS errors

	DRMS error (mm)		
	Odometry	Adafruit GPS	Complimentary filter
Run 1	426	925	330
Run 2	1 694	1 789	1 626
Run 3	1 325	1 274	1 297

5.2.3 Case study 3: Adafruit GPS DGPS

Here the effect on the DRMS error will be investigated when the DGPS capability of the Adafruit GPS is switched on. It must be noted it took between $12 - 15\ \mathrm{min}$ for the Adafruit GPS to find an EGNOS fix. The navigated map result is shown in Figure 5-7 and cumulative DRMS error plot in Figure 5-8.

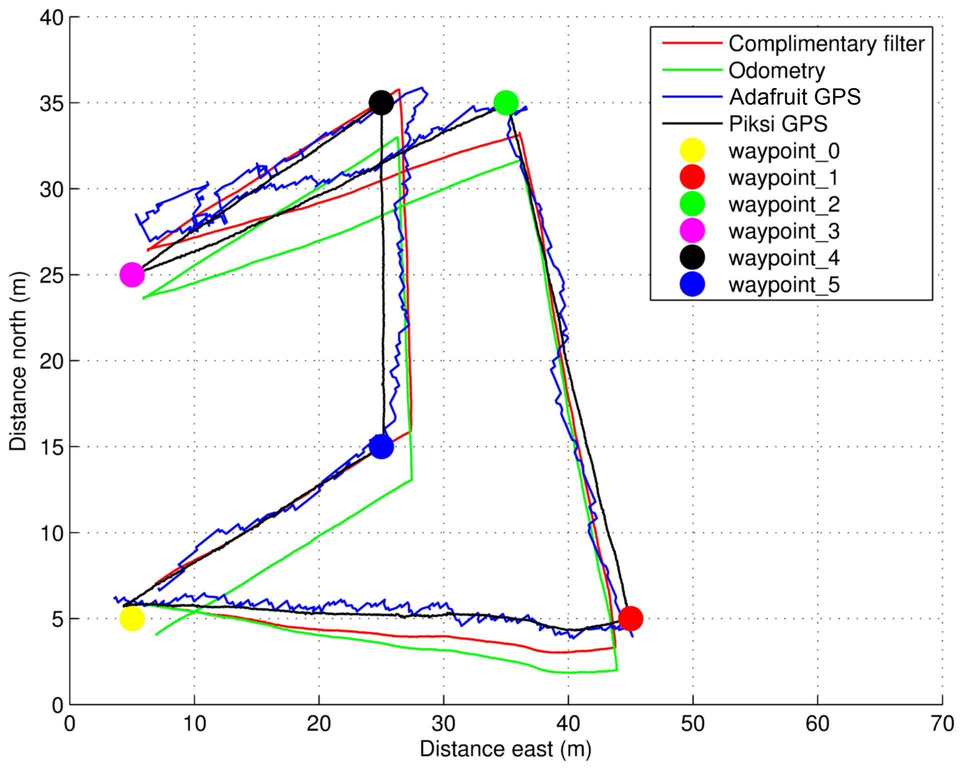


Figure 5-7: Case study 3 navigated map

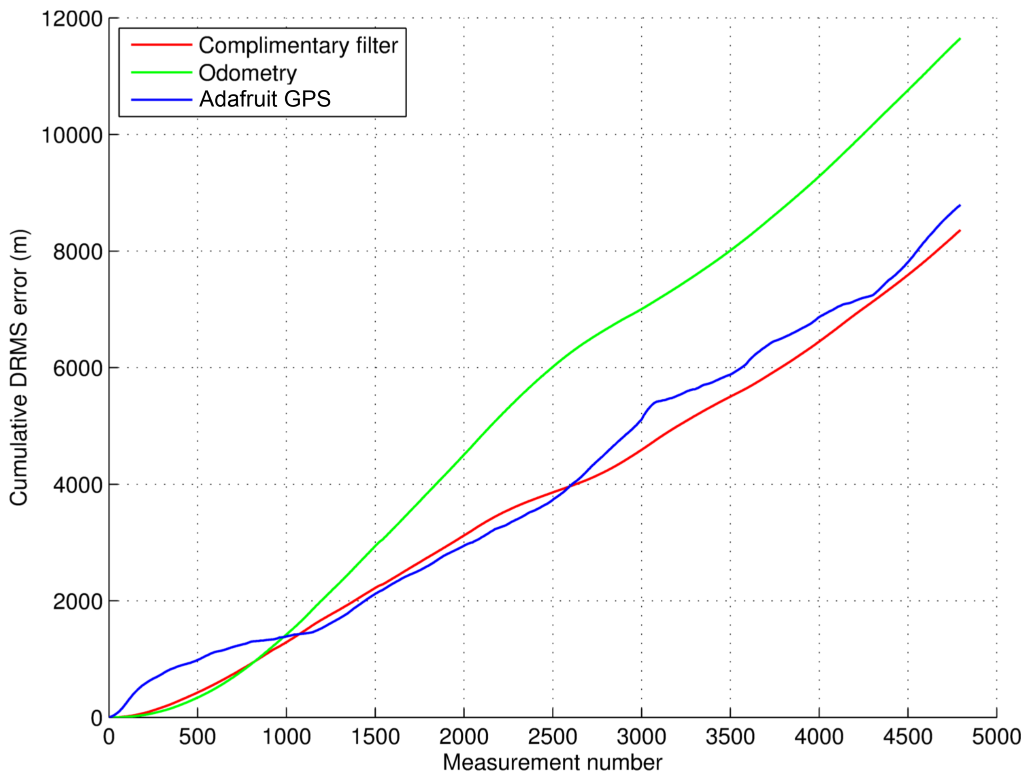


Figure 5-8: Case study 3 cumulative DRMS error

With the DGPS on, the mean accumulated DRMS errors as shown in Table 5-4 were calculated:

Table 5-4: Case study 3 DRMS errors

DRMS error (mm)		
Odometry	Adafruit GPS	Complimentary filter
$2\,429$	$1\,833$	$1\,743$

In the first case study it was seen that the mean deviation for the Adafruit GPS without DGPS enabled was $2\,479\ \mathrm{mm}$. This deviation has now dropped to $1\,833\ \mathrm{mm}$, meaning the complimentary filter output will also be more accurate.

5.2.4 Case study 4: Robot ground velocity

Here the effect on the DRMS error will be investigated when the wheel velocity is increased from the default $30\ \mathrm{rpm}$ (robot velocity of $0{,}313\ \mathrm{m \cdot s^{-1}}$), in increments of $10\ \mathrm{rpm}$ until $60\ \mathrm{rpm}$ is reached. Tests at higher velocities could not be conducted because the robot platform began moving too much to provide a stable platform for the laptop. With the robot moving at different velocities the following mean accumulated DRMS errors were calculated:

Table 5-5: Case study 4 DRMS errors

Run speed	DRMS error (mm)		
	Odometry	Adafruit GPS	Complimentary filter
$30\ \mathrm{rpm}$	$2\,854$	$1\,756$	$1\,631$
$40\ \mathrm{rpm}$	$2\,442$	$2\,134$	$2\,824$
$50\ \mathrm{rpm}$	$5\,515$	$1\,697$	$3\,007$
$60\ \mathrm{rpm}$	$7\,006$	$2\,286$	$4\,153$

From Table 5-5 it can be seen that only at $30\ \mathrm{rpm}$ the complimentary filter is effective, by yielding a result that is less erroneous than the other individual sensors. At $40\ \mathrm{rpm}$ there were long periods during which the Adafruit GPS gave the same positional data, resulting in the complimentary filter being more inaccurate than both the individual sensors. At 50 and $60\ \mathrm{rpm}$ the odometry information becomes very unreliable, this is because there is too much pitch and roll present and the compass is unable to compensate for this resulting in heading readings that vary even though the robot is moving in a straight line. The pitch and roll effect is even greater at the level of the compass since it is mounted higher than the robot platform. It is concluded that $30\ \mathrm{rpm}$ is the maximum speed at which experimental data can be gathered. The navigated maps and cumulated DRMS errors for these runs are included in Appendix D.

5.2.5 Case study 5: Wheel slippage

To get wheel slippage the robot was lifted from the ground for $10\ s$ at the halfway point between each of the waypoints. Figure 5-9 shows the result of the individual measurements, and Figure 5-10 shows the cumulative error.

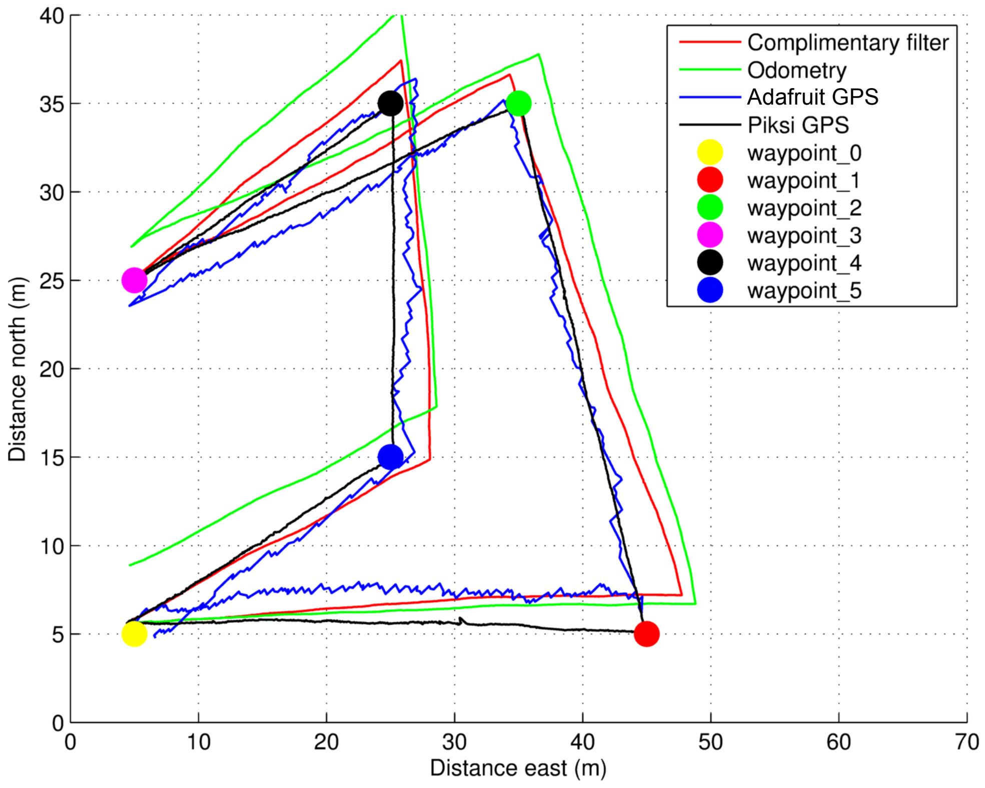


Figure 5-9: Case study 5 navigated map

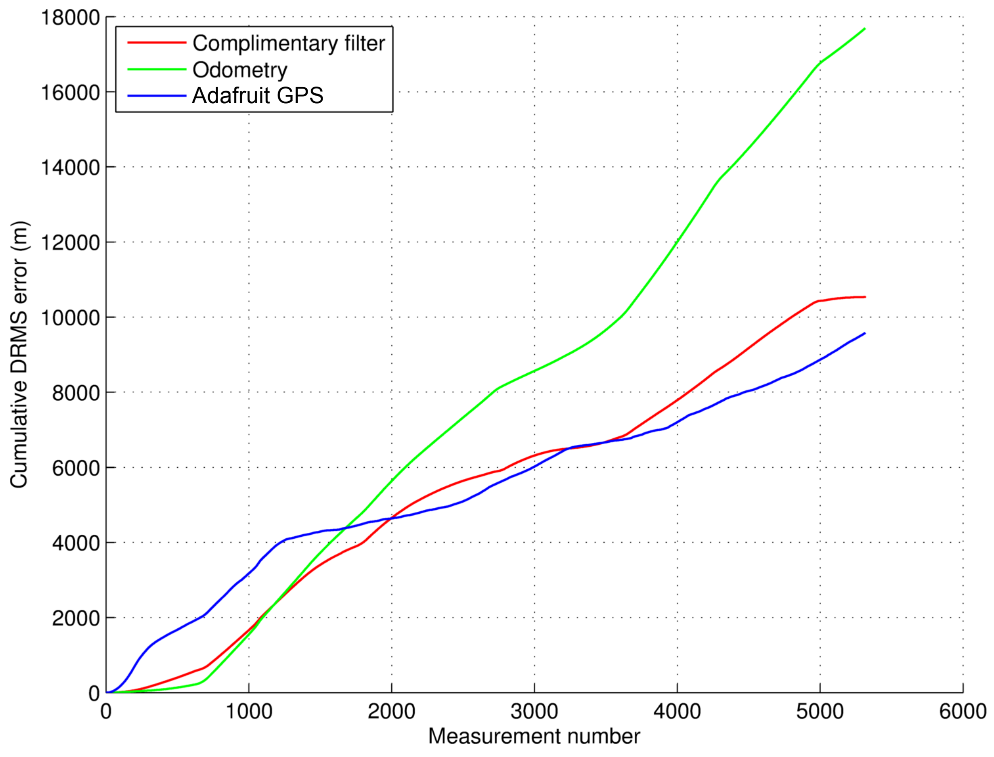


Figure 5-10: Case study 5 cumulative DRMS error

With the wheel slippage present the DRMS errors shown in Table 5-6 were calculated:

Table 5-6: Case study 5 DRMS errors

DRMS error (mm)		
Odometry	Adafruit GPS	Complimentary filter
3 328	1 801	1 981

The complimentary filter shows a larger error than the Adafruit GPS, this is because of the large amount the complimentary filter trusts the odometry readings. And in this case the odometry readings have a large error because it seems that the robot has travelled further than it actually has because of the slippage error that is introduced to the encoders.

5.2.6 Case study 6: Magnetic interference and GPS drift

For this study the measurement data of the fourth run of the first case study was taken and a virtual magnetic interference as well as GPS drift introduced separately. The interference and drift was implemented by post processing the odometry and GPS data to include these sources of error. The virtual magnetic interference point is shown in Figure 5-11. This point will simulate a new magnetic north and the compass will trust this simulated north between 20 to 30 % while trusting the real north the rest.

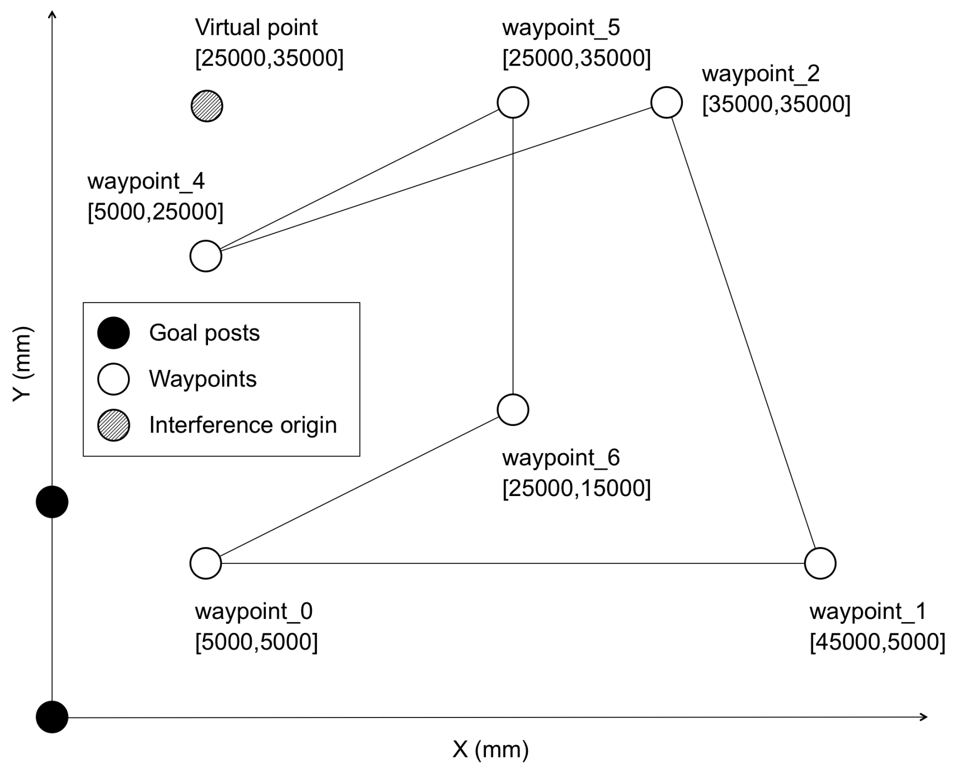


Figure 5-11: Map to be navigated

Figure 5-12 shows the result of the individual measurements when the magnetic interference is introduced. It is clear that the compass heading is greatly affected by the artificial magnetic north, resulting in the odometer measurements being of no use. The complimentary filter shows the robot being at waypoint 0 when navigation is complete, this is pure coincidence and a result of the odometry readings cancelling themselves out.

Figure 5-13 shows the cumulative error. Because of the amount the complimentary filter trusts the odometry readings the complimentary error is larger than that of the Adafruit GPS.

For the study of GPS drift the measurement data was taken and a virtual GPS drift of $3 - 4\ \mathrm{m}$ introduced. Since regular GPS has a pseudo range accuracy of $7{,}8\ \mathrm{m}$ as stated earlier, this is a drift of 38 to $50\ \%$ in GPS accuracy. Figure 5-14 shows the result of the individual measurements when GPS drift is present and Figure 5-15 shows the cumulative error.

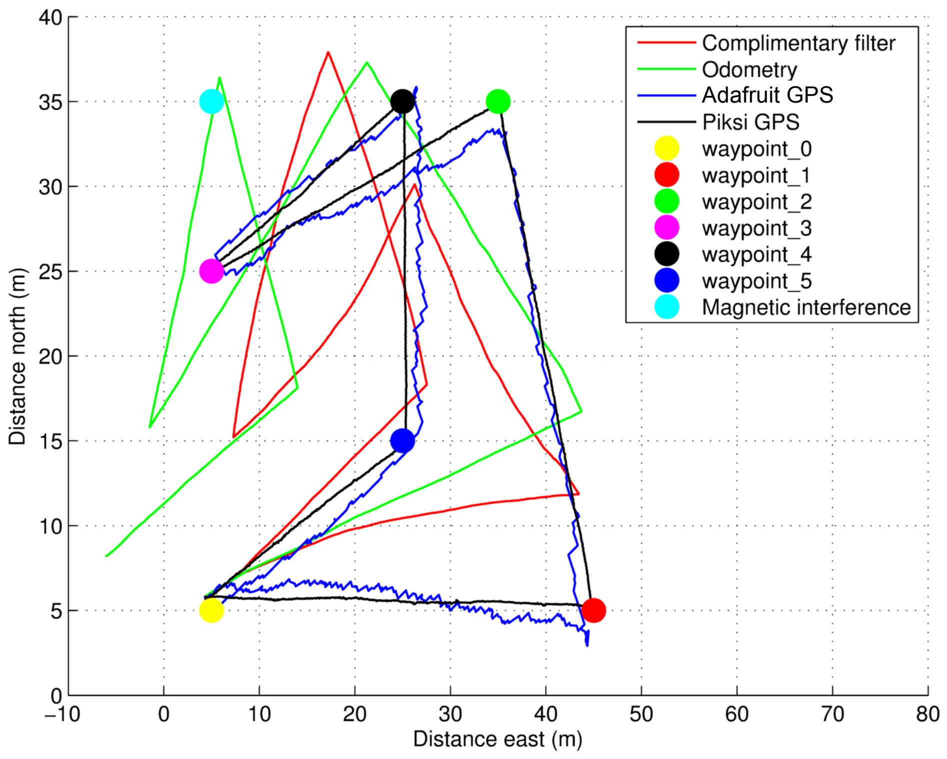


Figure 5-12: Case study 6 with magnetic interference navigated map

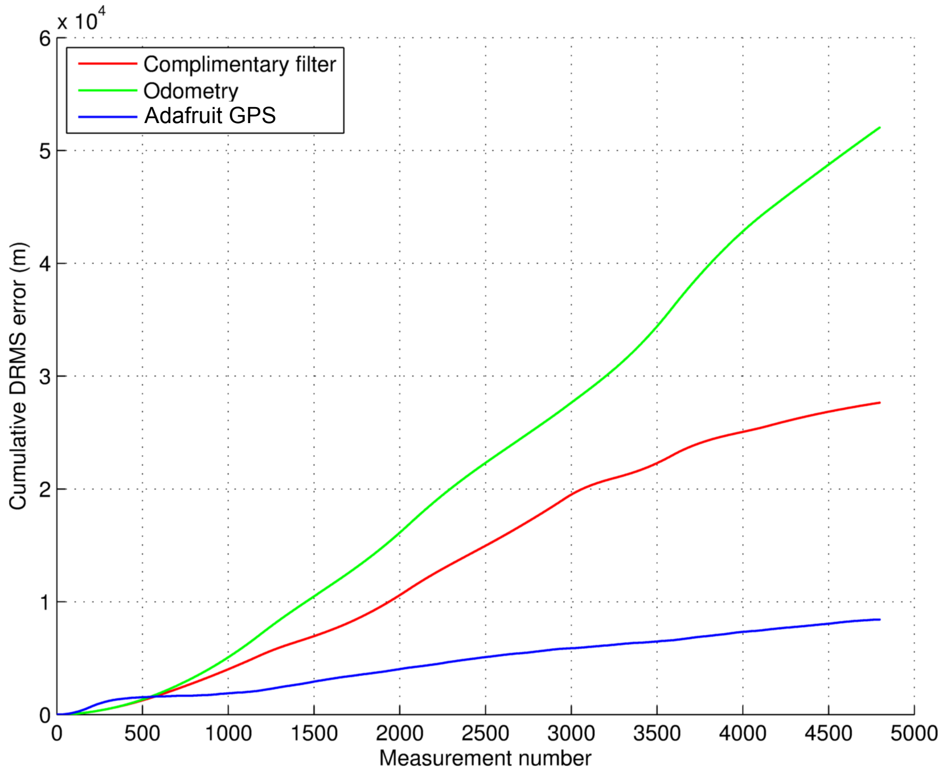


Figure 5-13: Case study 6 with magnetic interference cumulative DRMS error

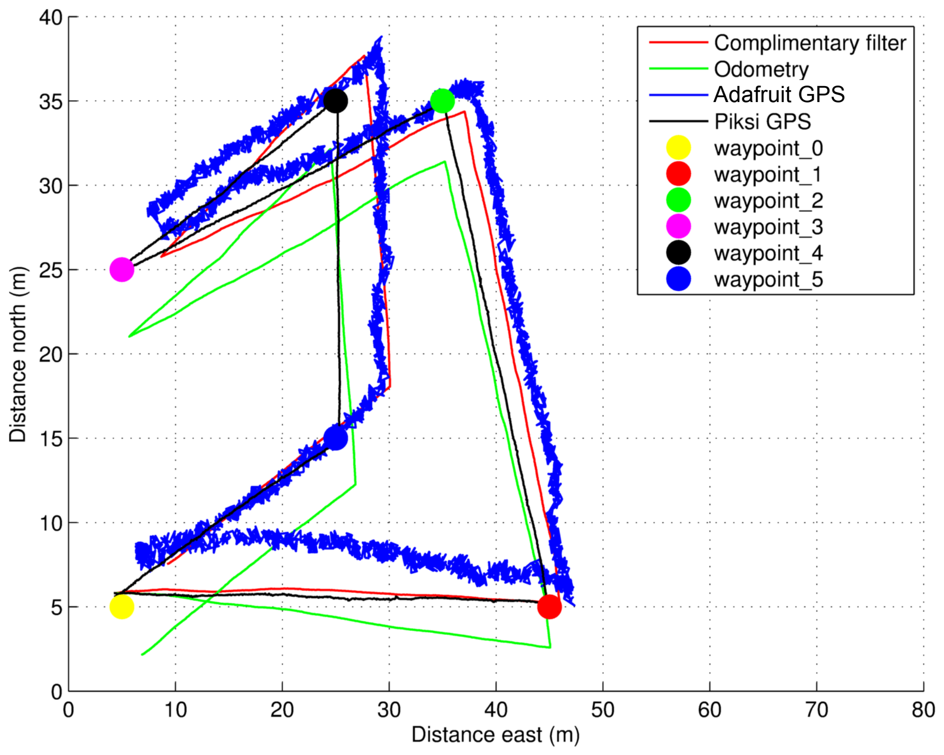


Figure 5-14: Case study 6 with GPS drift navigated map

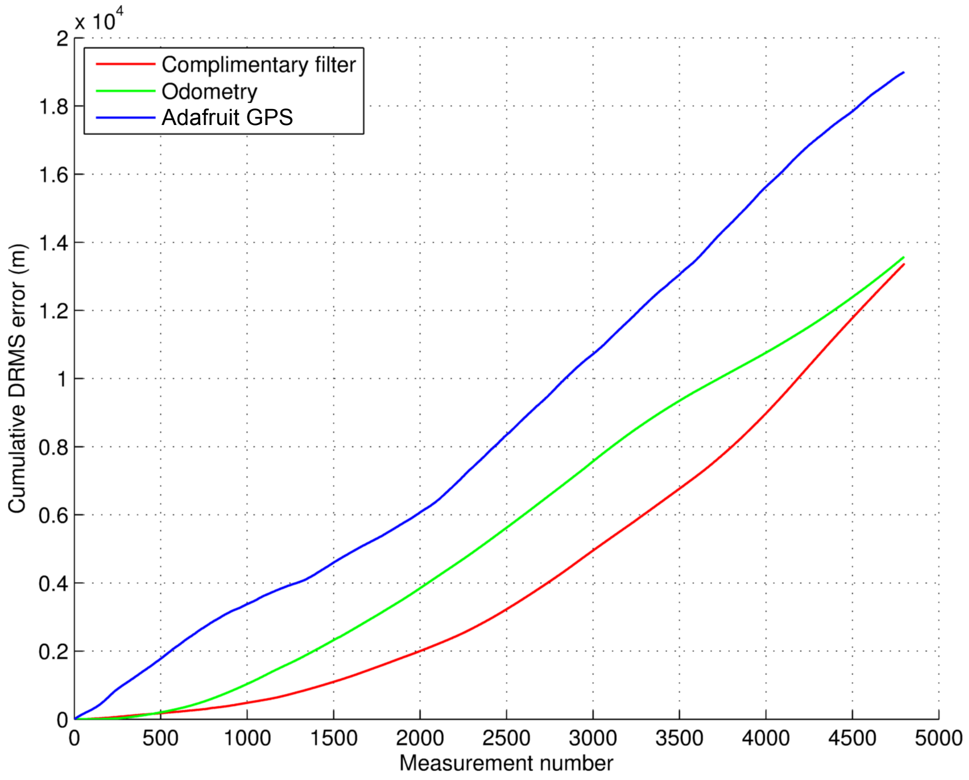


Figure 5-15: Case study 6 with GPS drift cumulative DRMS error

With the robot moving under the different scenarios, the following mean accumulated DRMS errors were calculated:

Table 5-7: Case study 6 DRMS errors

Scenario	DRMS error (mm)		
	Odometry	Adafruit GPS	Complimentary filter
Magnetic interference	10 844	1 756	5 761
GPS drift	2 828	3 958	2 784

From Table 5-7 it is clear that the magnetic interference has the largest effect on the precision with which the robot can navigate, showing an odometry DRMS error of over $10\ \mathrm{m}$, while the GPS drift has almost no effect because the complimentary output error is still less than that of the individual sensors.

5.2.7 Case study 7: One or multiple sensors fail

In this final case study the relative navigational accuracy will be compared when the Adafruit GPS, encoders and compass are individually powered off during a navigational run. To get consistency in the results data from the same run will be taken to compare the effect of sensor malfunction. For this study the data from the third run of the first case study will be taken and the results are shown in Table 5-8.

Table 5-8: Case study 7 DRMS errors

Scenario	DRMS error (mm)		
	Odometry	Adafruit GPS	Complimentary filter
All sensors functioning	1 156	2 487	2 117
No Adafruit GPS	1 156	26 263	20 689
No encoders	26 263	2 487	14 438
No compass	Inf	2 487	Inf

When no GPS is present the complimentary filter is skewed towards waypoint 0 because this point is reported by the Adafruit GPS as its location as shown in Appendix E. When no encoders are present the odometry information tells the complimentary filter that the robot is standing still, but because the Adafruit GPS is still sending valid data the robot position is known, but only with a precision of over $14\ \mathrm{m}$. When the compass is disabled the robot thinks it is heading directly north and the cumulated DRMS error quickly grows to a value that is too large to plot. It can be concluded that if any of the sensors fail the robot is unable to navigate successfully between the waypoints. The navigated maps and cumulated DRMS errors for these runs are included in Appendix E.

6 Conclusions

This thesis investigated the navigational precision of an autonomous ground vehicle by fusing different sensors as a means of localization and navigation. Different GPS modules (regular, RTK and differential GPS) in conjunction with a digital compass and optical encoders were used as sensors for capturing data regarding the robot's position. The Arduino Mega 2560 with an 8-bit Atmel microcontroller was used to control all robot functions while MATLAB was used to plot all navigational output data.

To implement the localization and navigation, background information had to be gained regarding the functioning of the GPS, motor speed control, fusion of sensor data and algorithms used by sensors. After this was done all the hardware required to implement navigation was purchased, compatibility between all the components was ensured, housings for the sensors were manufactured, the current platform was modified and a power source sufficient to power everything was selected. Next software was implemented to: control the hardware, capture all the data from the sensors, fuse sensor data, map the environment, establish localization and navigate between waypoints and finally display all the captured data to the user.

Once the robot platform was able to navigate using the software implemented experimental data was captured. First a way of measuring the precision of the data captured by the sensors as the robot navigated around the map was needed. To do this the precision with which the Piksi RTK GPS is able to keep the robot on track was tested. It was found that the Piksi RTK GPS is able to represent the position of the robot within a DRMS error of $290\ \mathrm{mm}$ (subject to measurement uncertainty). All the other tests were measured against the Piksi RTK GPS data to get a measure of their precision.

Seven case studies were completed, starting with the robot navigating the same map five times to determine consistency between runs. The mean DRMS errors for both the odometry and Adafruit GPS were less than the error of the most inaccurate sensor (Adafruit GPS), thus it was concluded that the consistency is sufficient for comparing data between runs. The complimentary filter constant needed to be determined and over three runs it was seen that a value of $0{,}001$ gives the best position estimate of the robot. It was also seen that enabling the DGPS capability of the Adafruit GPS showed a DRMS error decrease of $645\ \mathrm{mm}$ over the regular Adafruit GPS readings, meaning it is advantageous towards the complimentary filter to keep the DGPS capabilities enabled during navigation. It was experimentally found that the lowest ground speed of $0{,}313\ \mathrm{m \cdot s^{-1}}$ showed the smallest DRMS error, which makes logical sense. But by increasing the robot speed by a factor of two, the odometry error alone increased by $2{,}5$ times with an odometry DRMS error of $7\ \mathrm{m}$. When introducing wheel slippage the odometry error clearly increased and had a big effect on the result of the complimentary filter because of the filters large dependence on the odometry measurements. When magnetic interference was introduced the robot was completely unable to navigate, while GPS drift had little effect on navigational precision. Finally it was seen that whenever one of the sensors (Adafruit GPS, optical encoders and digital compass) fail the robot is also completely unable to navigate.

The high precision with which the Piksi RTK GPS is able to locate the robot gives it the ability to be implemented in various other autonomous and navigational scenarios. From all the results obtained it can be concluded that fusing sensor data does make the robot localization more precise and helps when one of the sensors collects erroneous data for a short duration during navigation.

Future work that affects the navigational precision can include the following:

1. The use of different data fusion algorithms.
2. The fusion of Piksi RTK GPS with odometry data.
3. Changes in the robot platform structure to keep it level and get more accurate compass measurements at higher ground speeds.
4. Sensor implementation that can detect wheel slippage and account for it.
5. Algorithms to detect magnetic interference and counter it.
6. Changing the ground vehicle to an aerial one.
7. Upgrading Piksi RTK GPS direct communication antennas and testing it over a greater distance.

7 Appendix A: RTK GPS

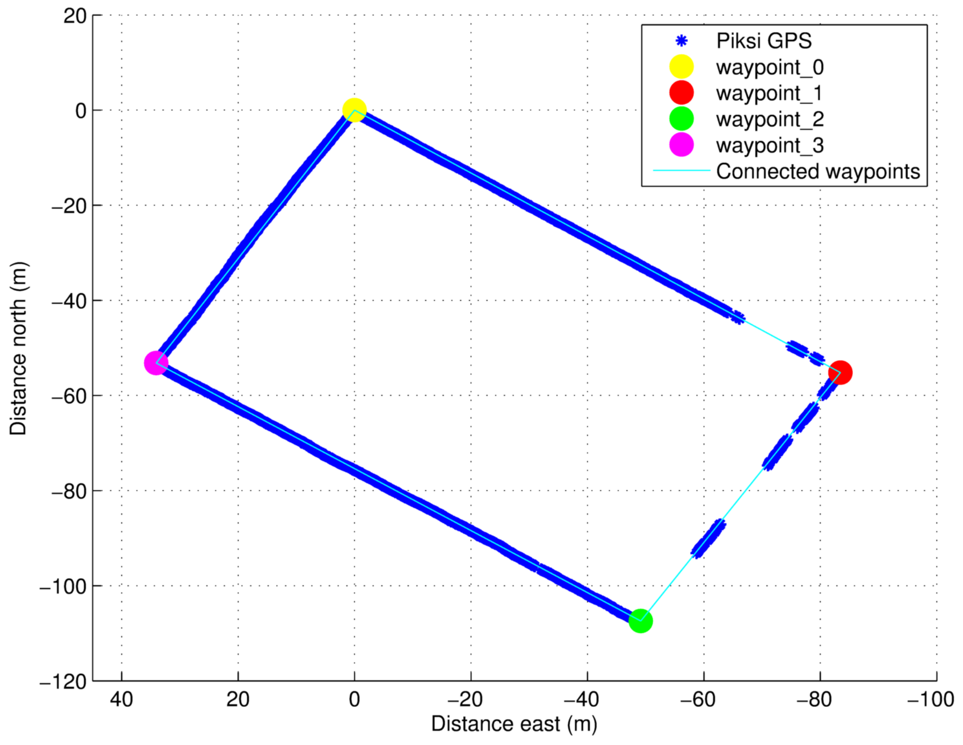


Figure 7-1: Run 2 RTK GPS positional data

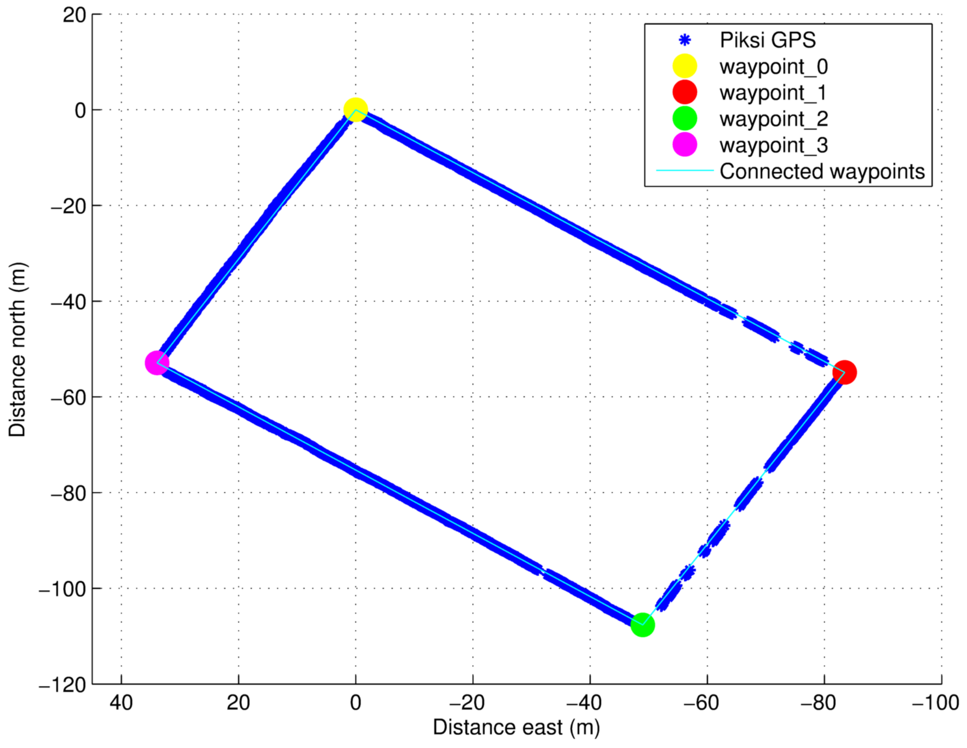


Figure 7-2: Run 3 RTK GPS positional data

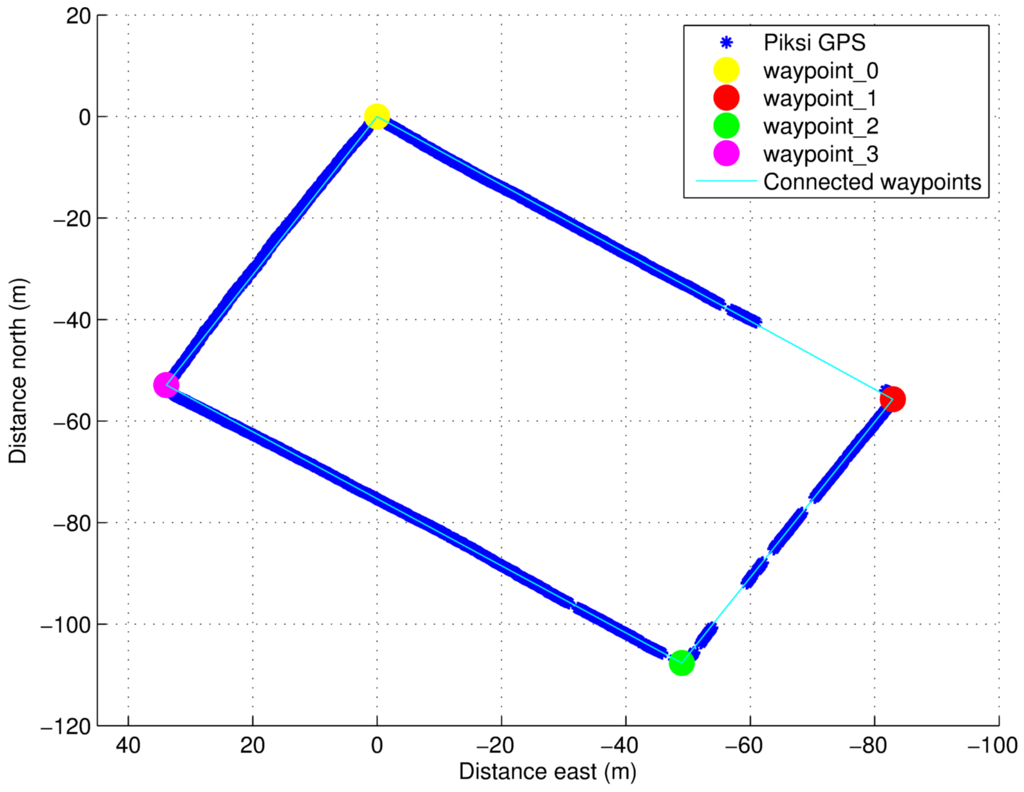


Figure 7-3: Run 4 RTK GPS positional data

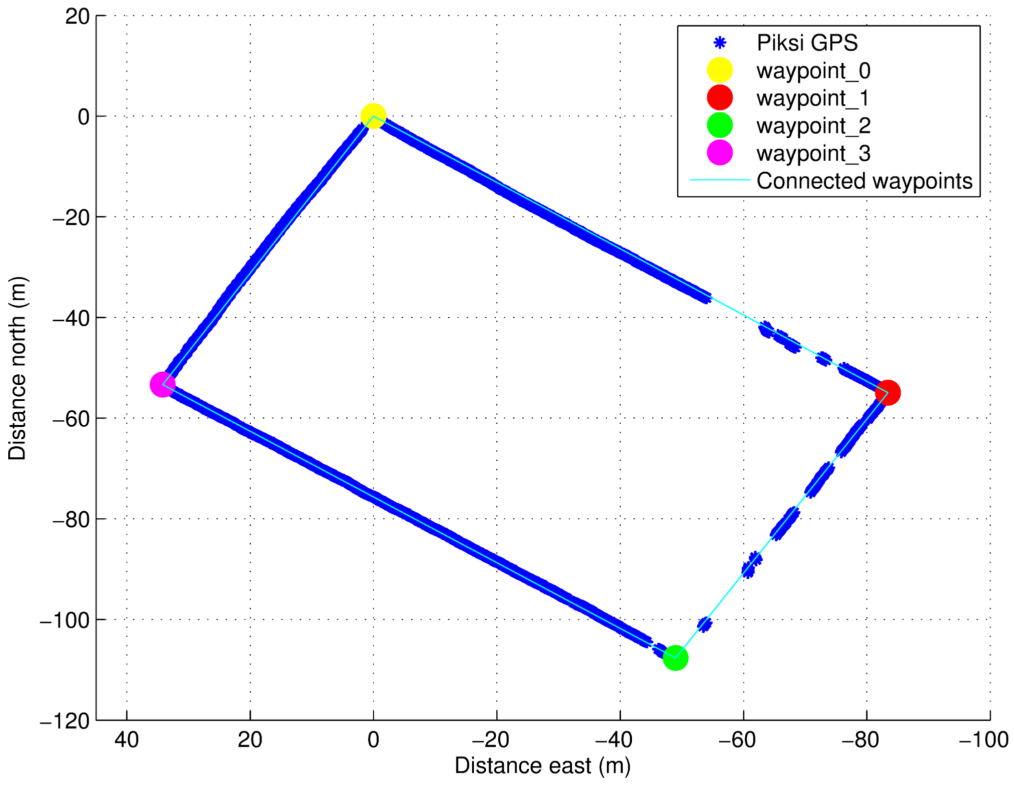


Figure 7-4: Run 5 RTK GPS positional data

8 Appendix B: Case study 1

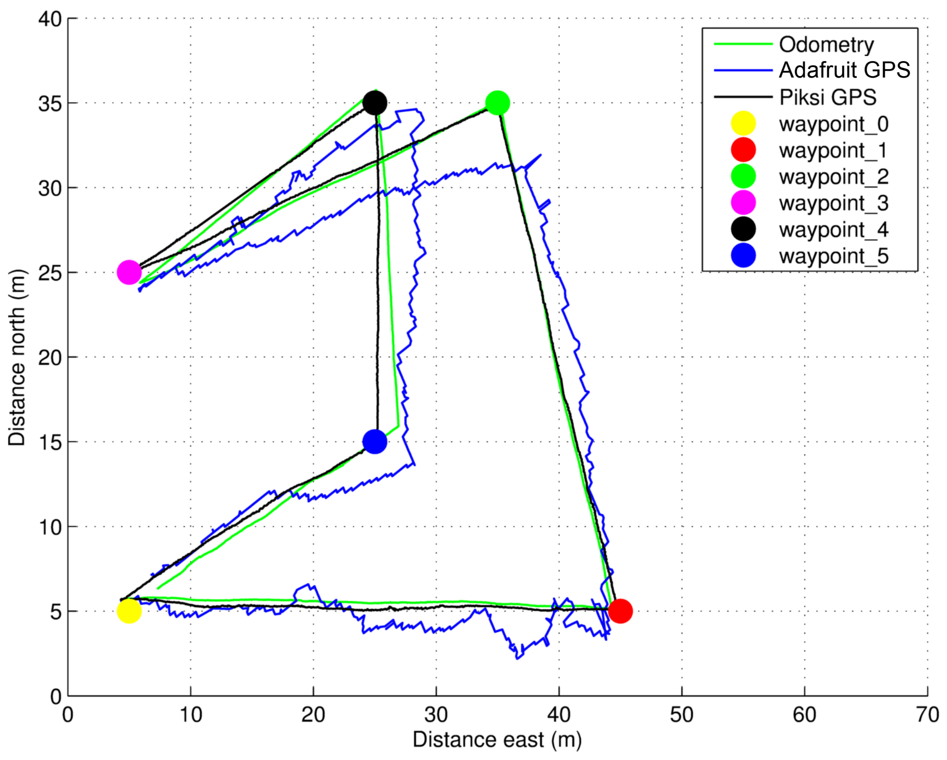


Figure 8-1: Case study 1 navigated map run 2

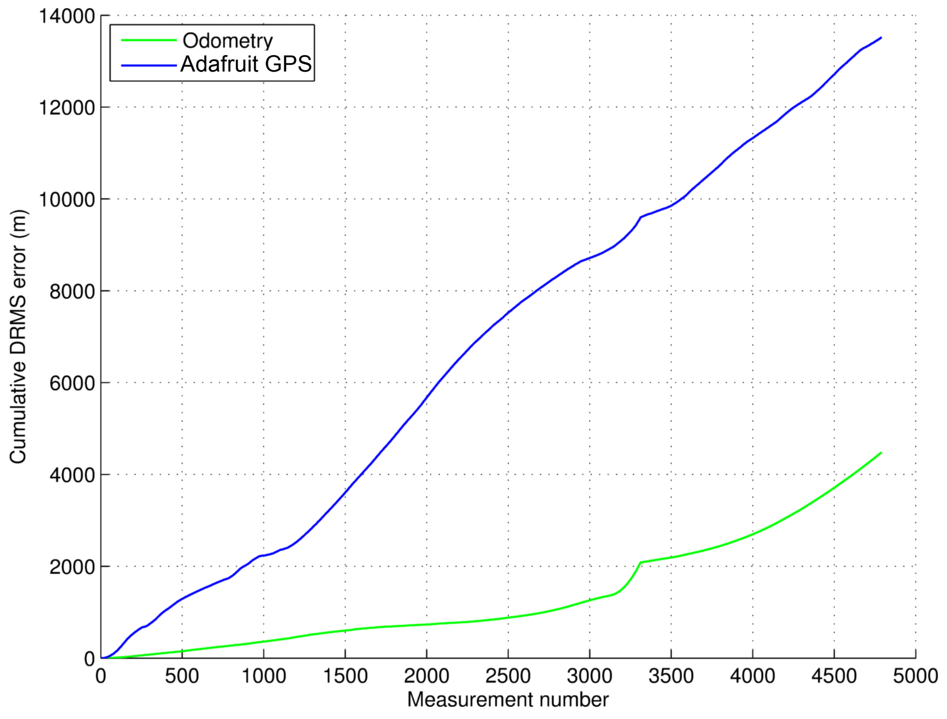


Figure 8-2: Case study 1 cumulative DRMS error run 2

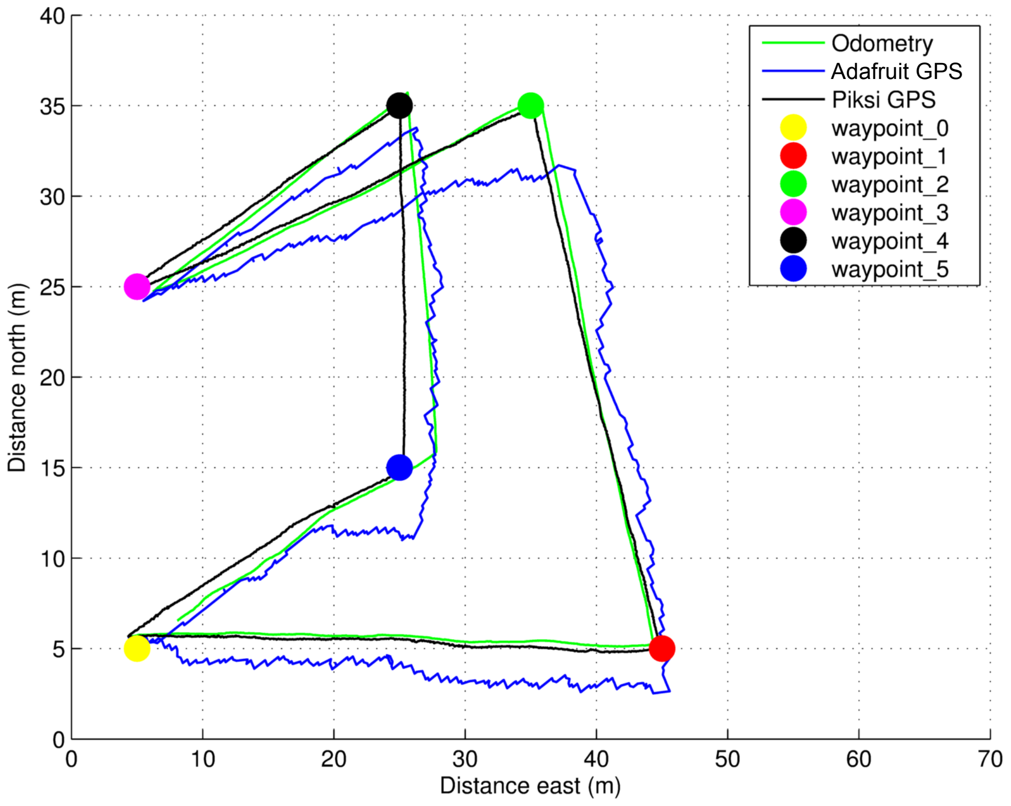


Figure 8-3: Case study 1 navigated map run 3

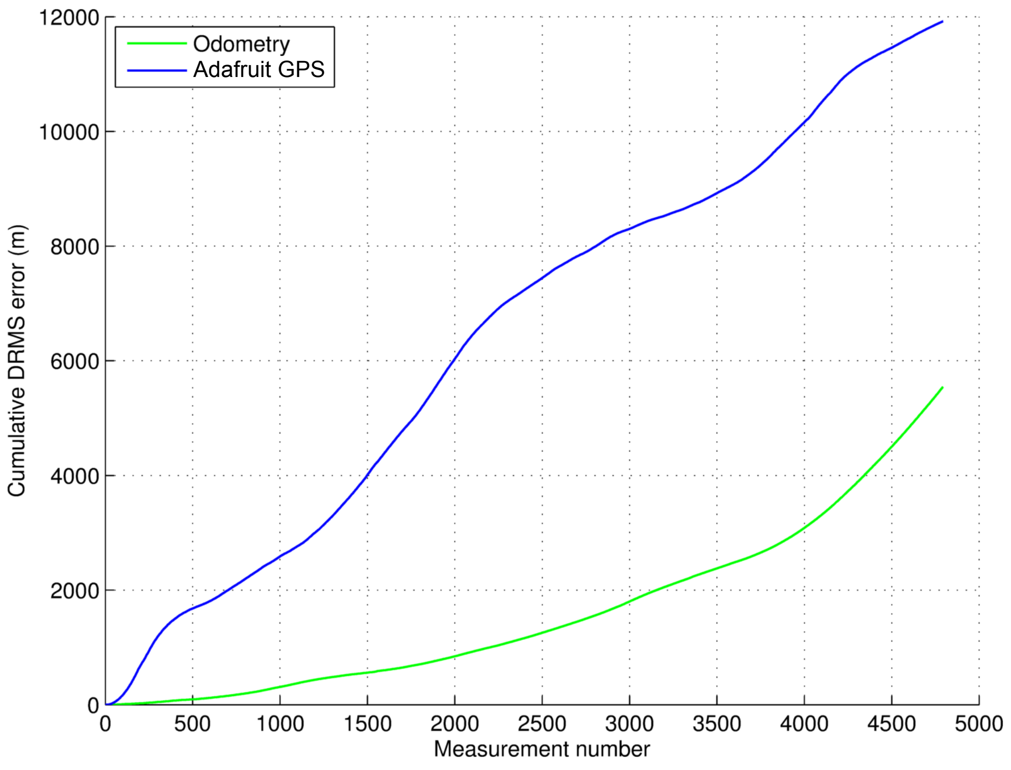


Figure 8-4: Case study 1 cumulative DRMS error run 3

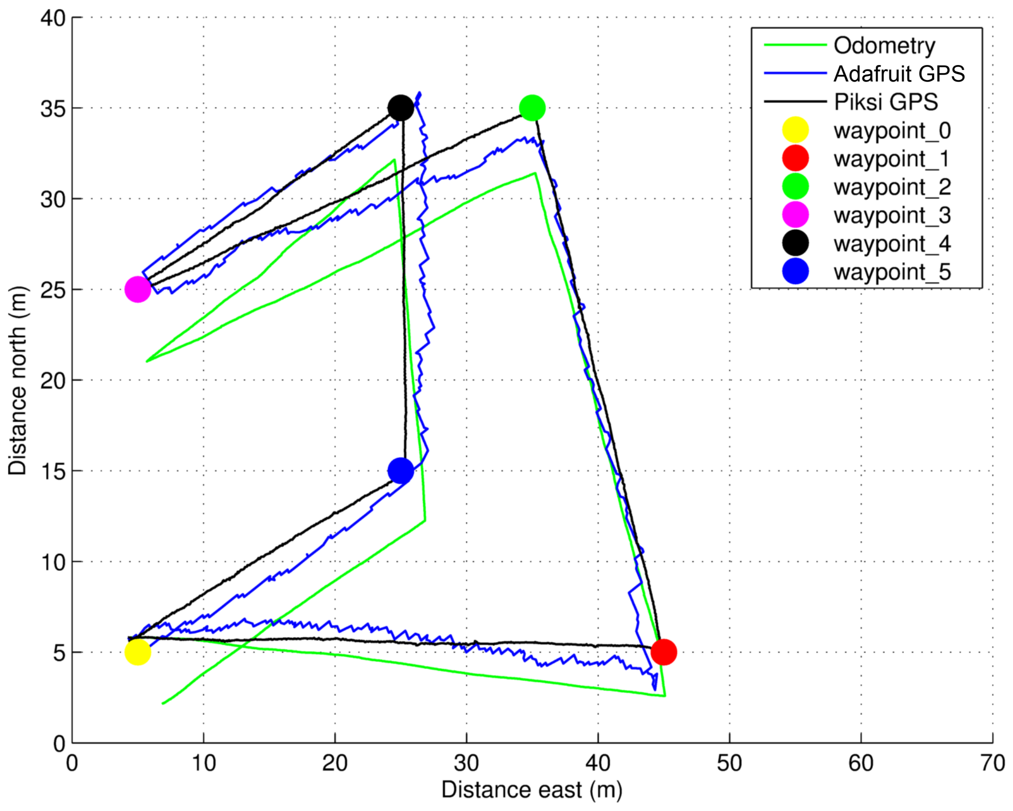


Figure 8-5: Case study 1 navigated map run 4

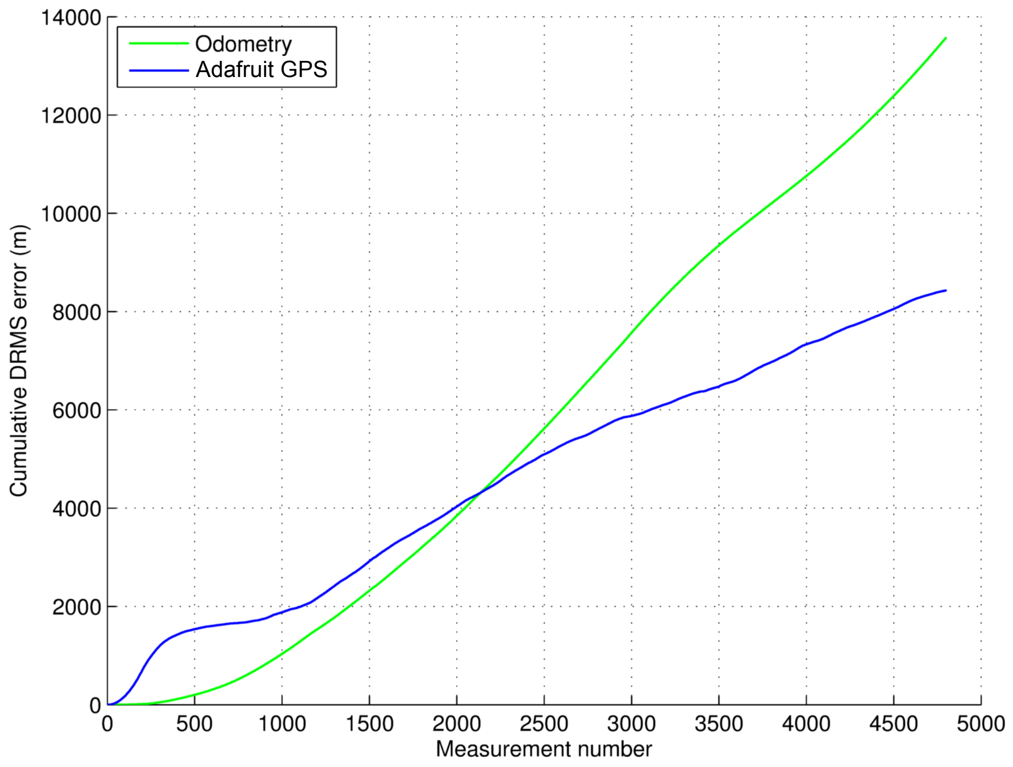


Figure 8-6: Case study 1 cumulative DRMS error run 4

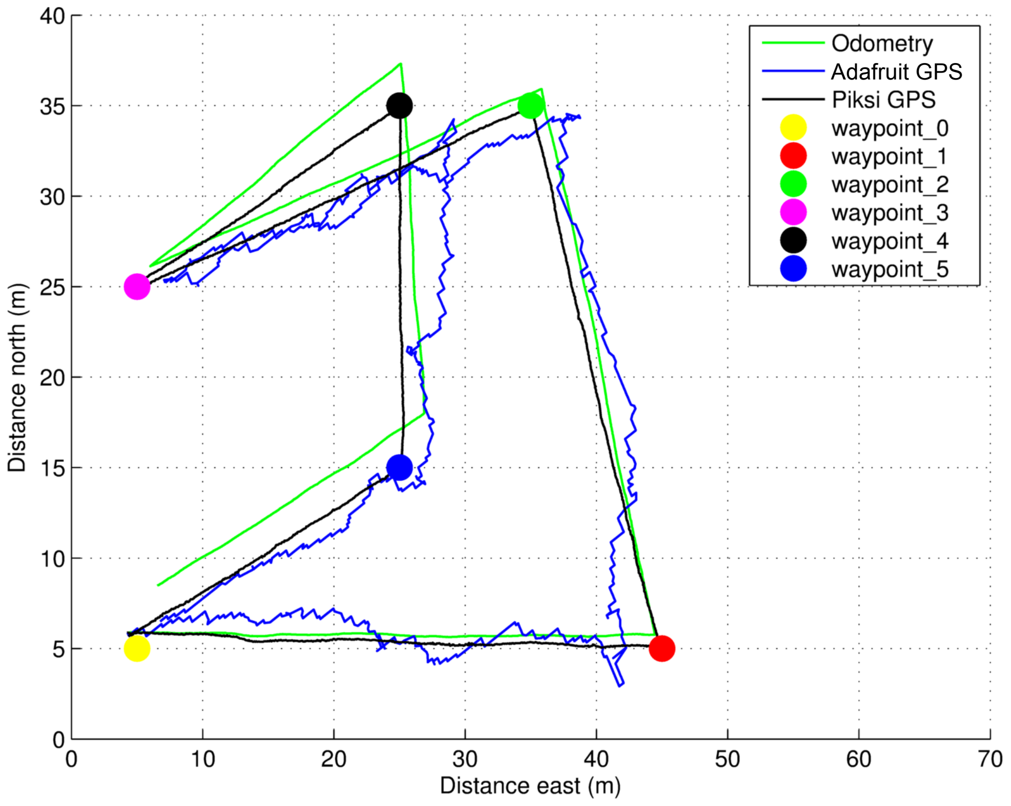


Figure 8-7: Case study 1 navigated map run 5

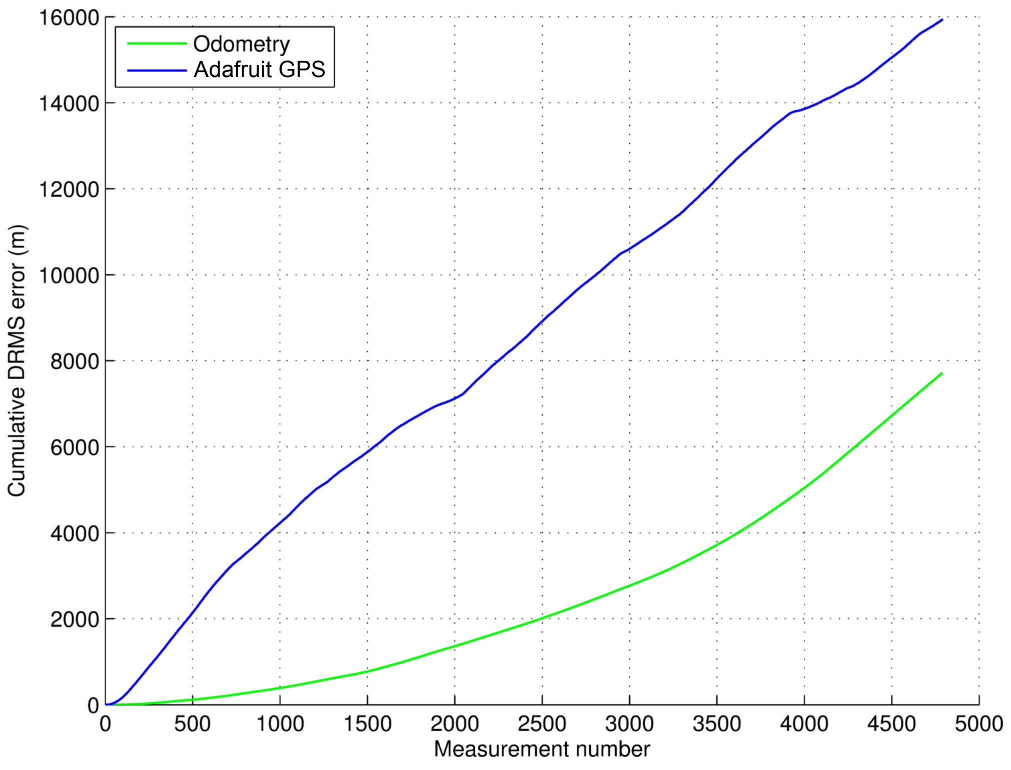


Figure 8-8: Case study 1 cumulative DRMS error run 5

9 Appendix C: Case study 2

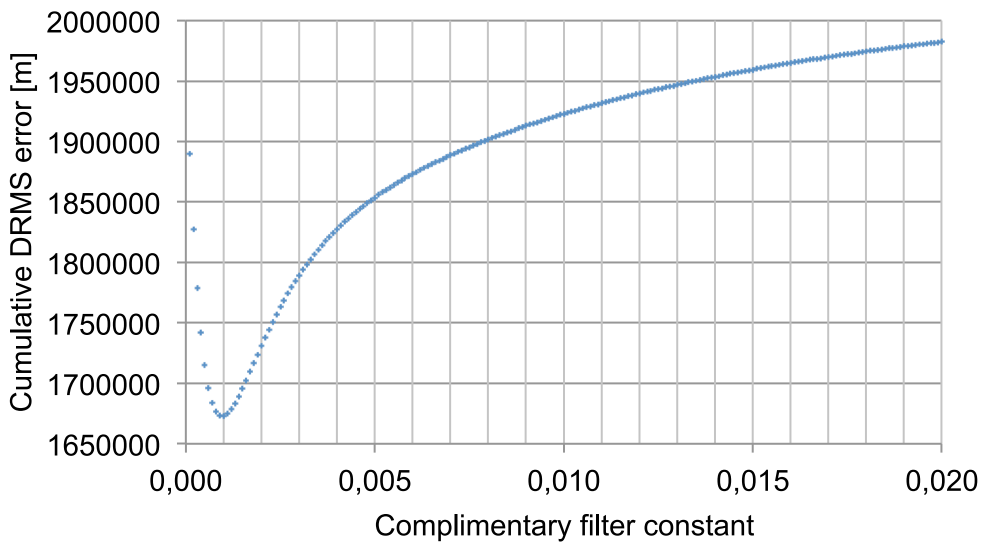


Figure 9-1: Cumulative DRMS error for different complimentary filter constants run 2

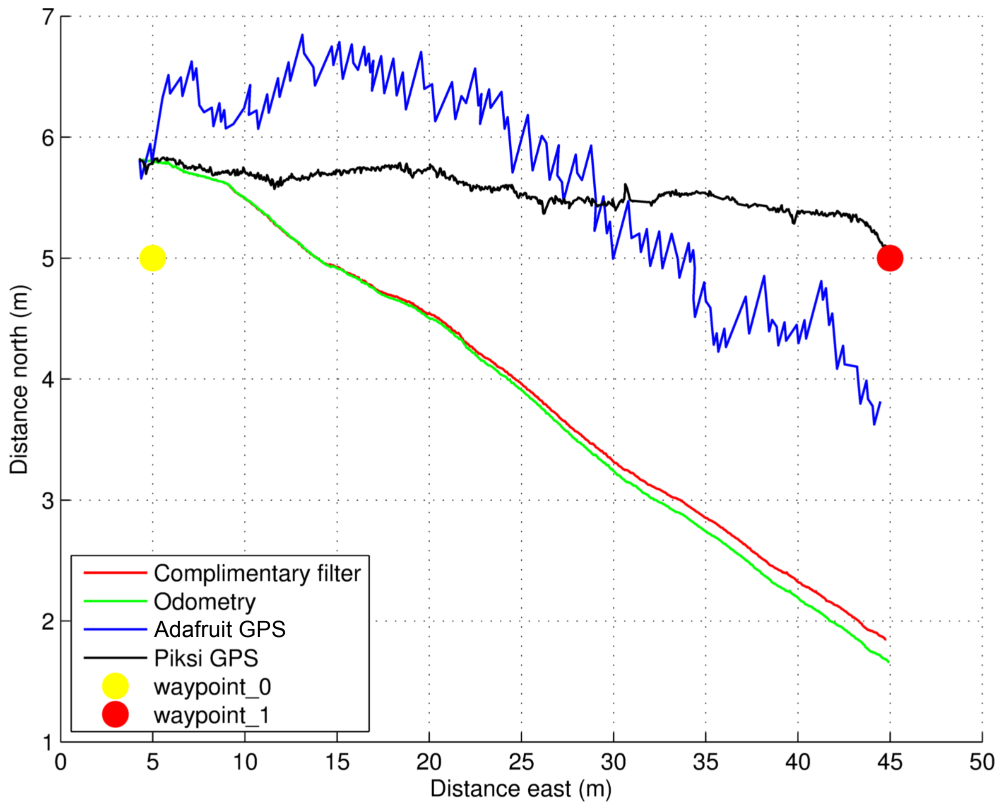


Figure 9-2: Case study 2 navigated map run 2

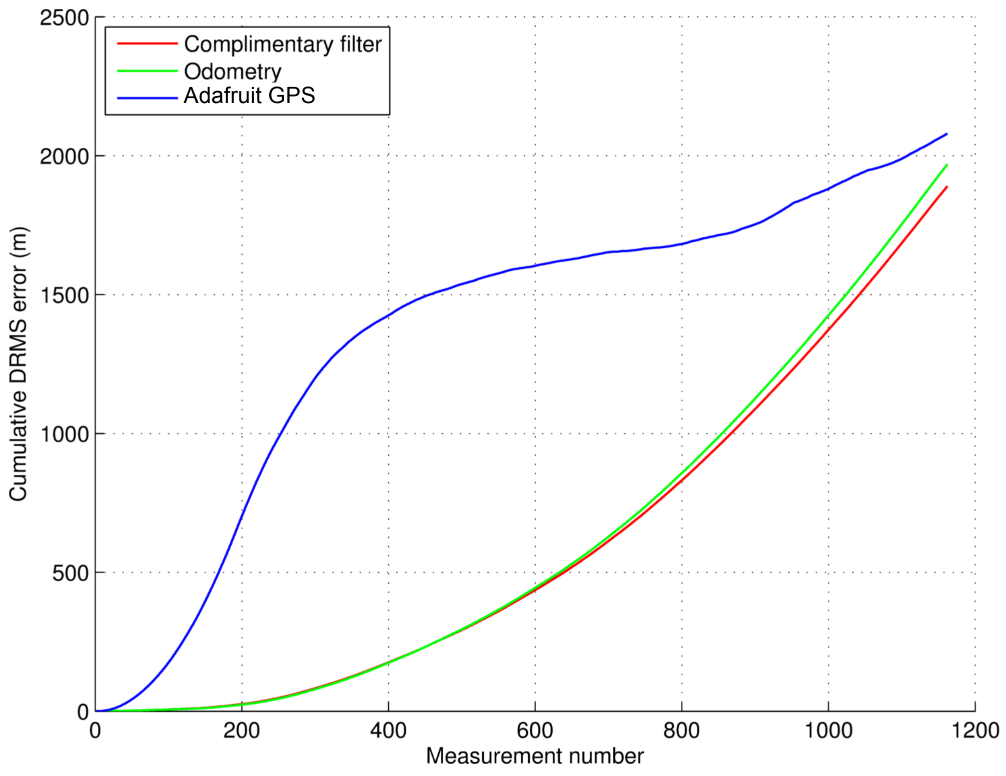


Figure 9-3: Case study 2 cumulative DRMS error run 2

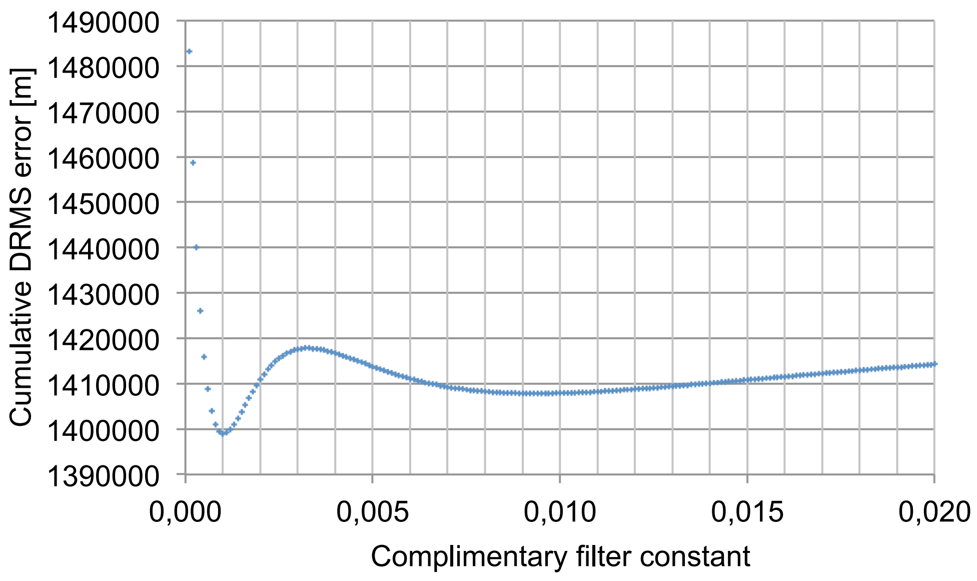


Figure 9-4: Cumulative DRMS error for different complimentary filter constants run 3

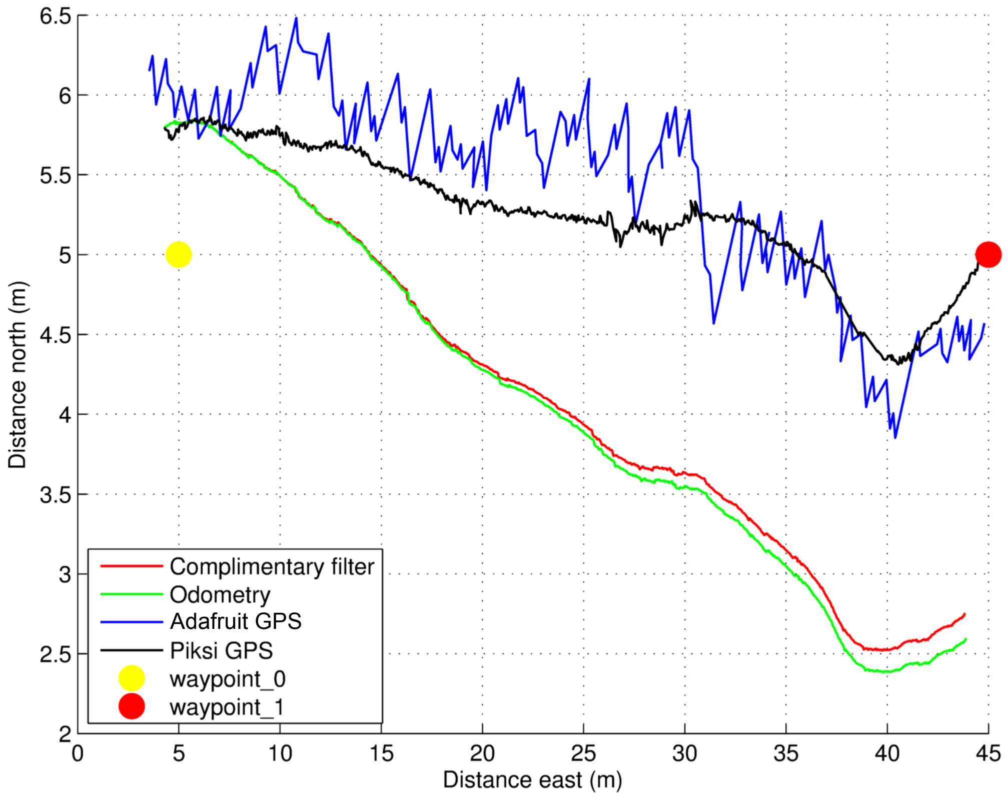


Figure 9-5: Case study 2 navigated map run 3

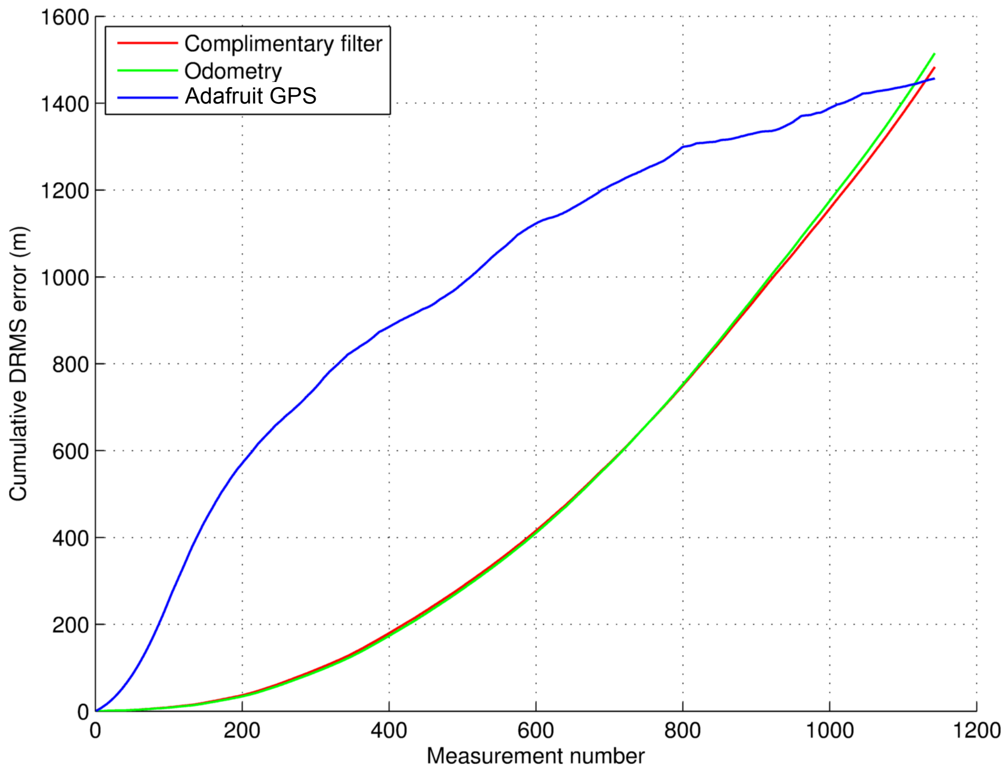


Figure 9-6: Case study 2 cumulative DRMS error run 3

10 Appendix D: Case study 4

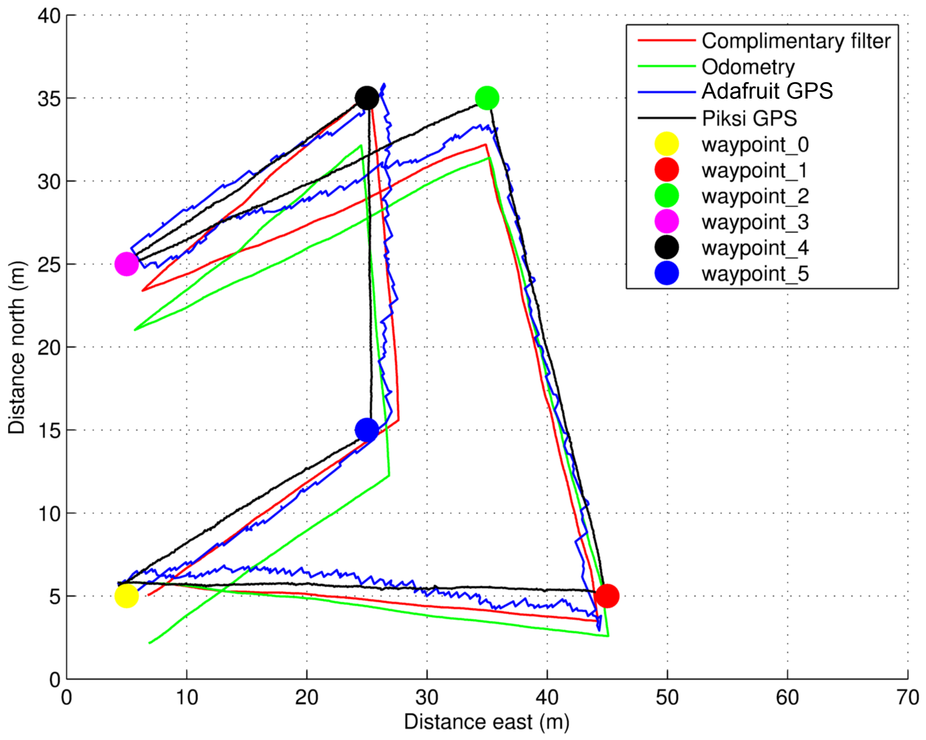


Figure 10-1: Case study 4 navigated map 30 rpm

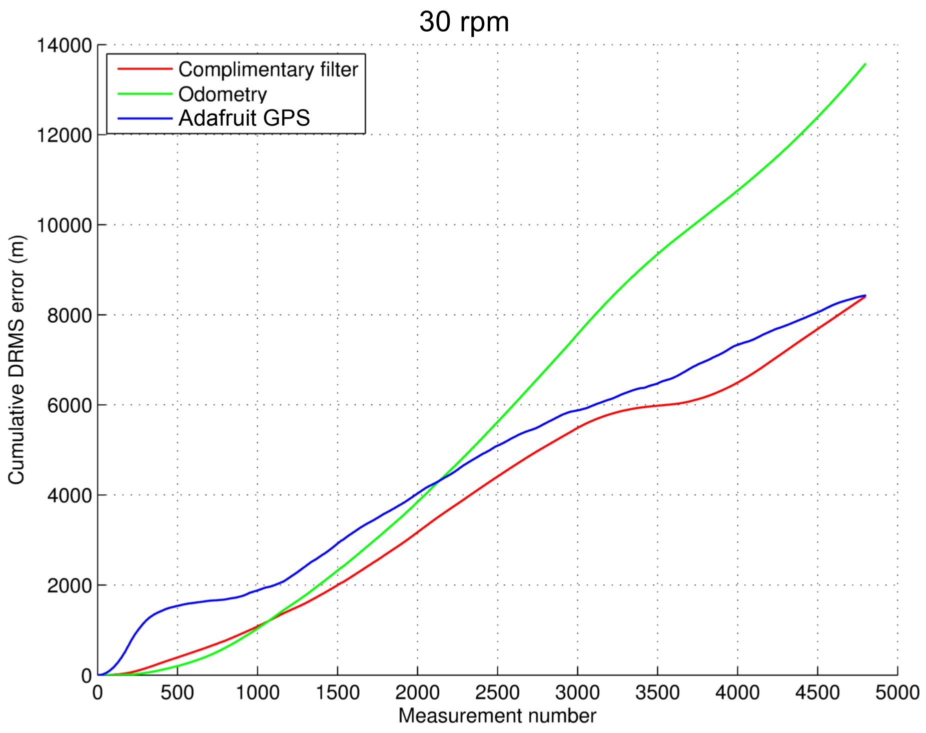


Figure 10-2: Case study 4 cumulative DRMS error 30 rpm

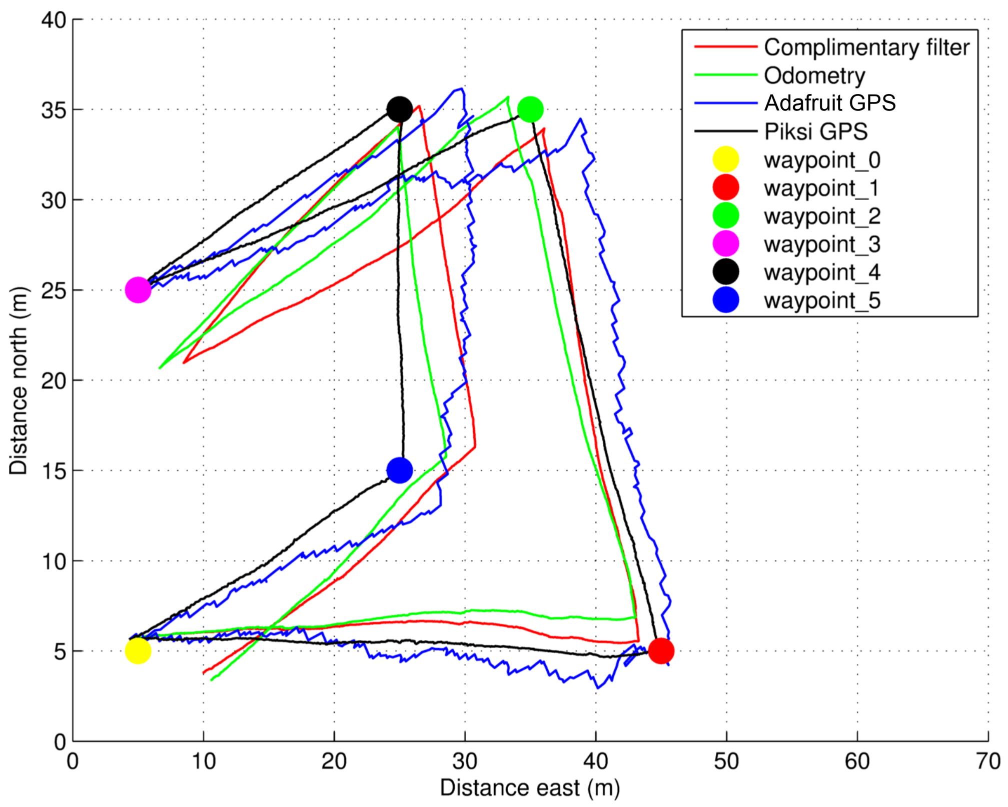


Figure 10-3: Case study 4 navigated map 40 rpm

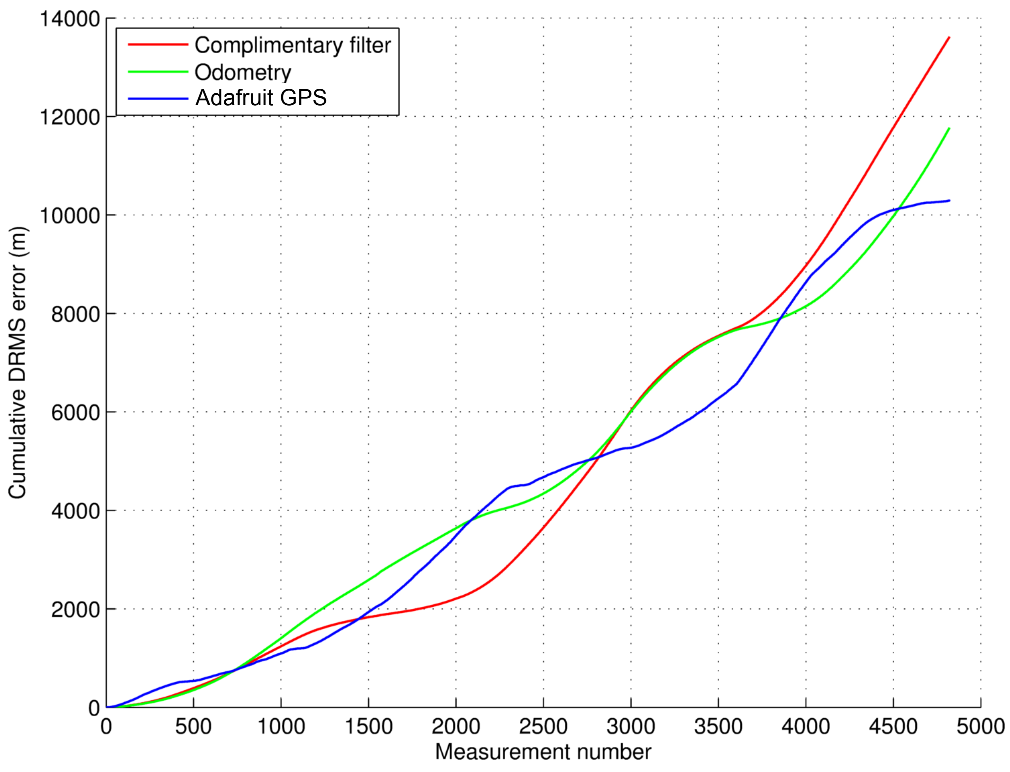


Figure 10-4: Case study 4 cumulative DRMS error 40 rpm

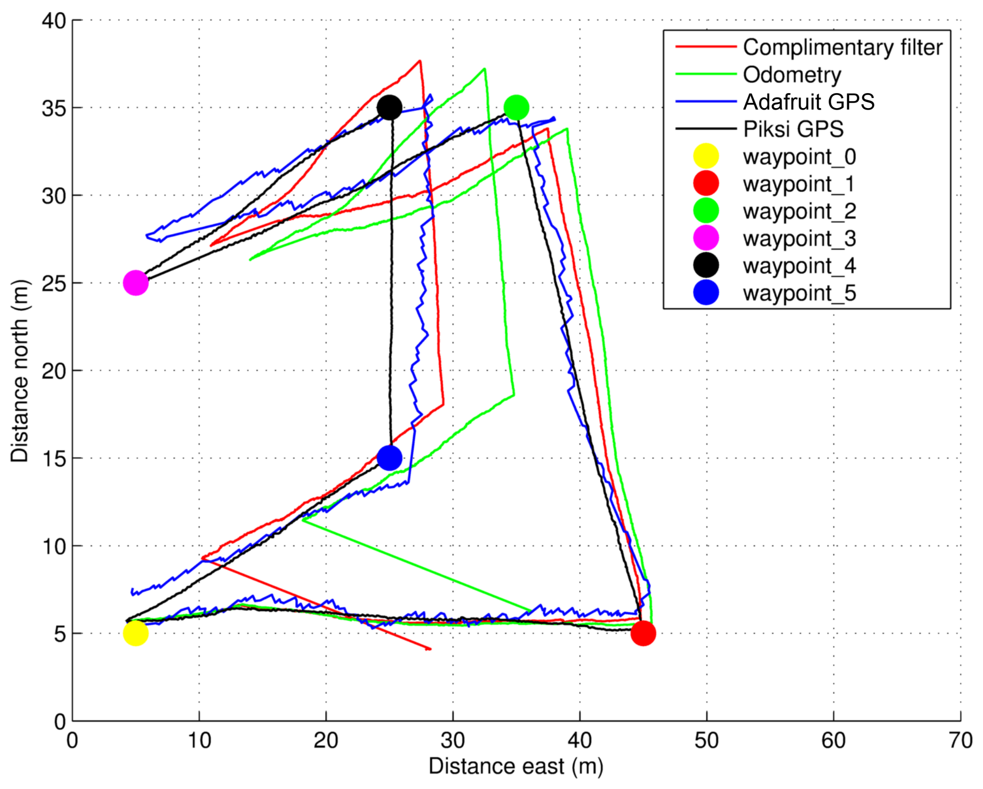


Figure 10-5: Case study 4 navigated map 50 rpm

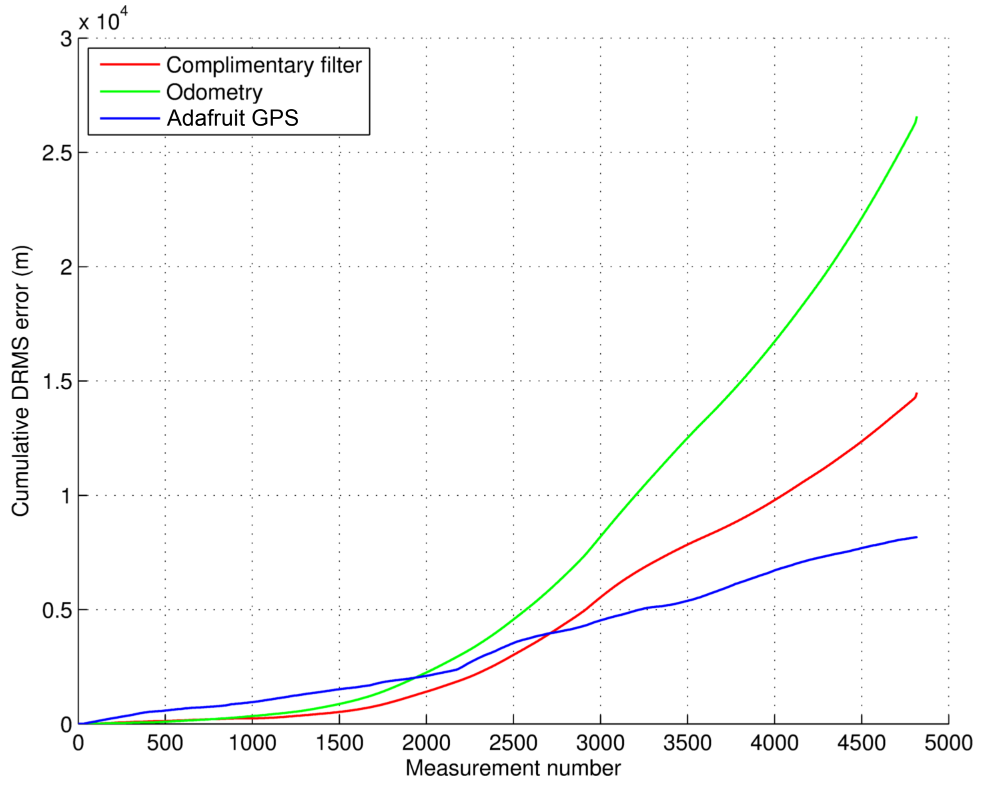


Figure 10-6: Case study 4 cumulative DRMS error 50 rpm

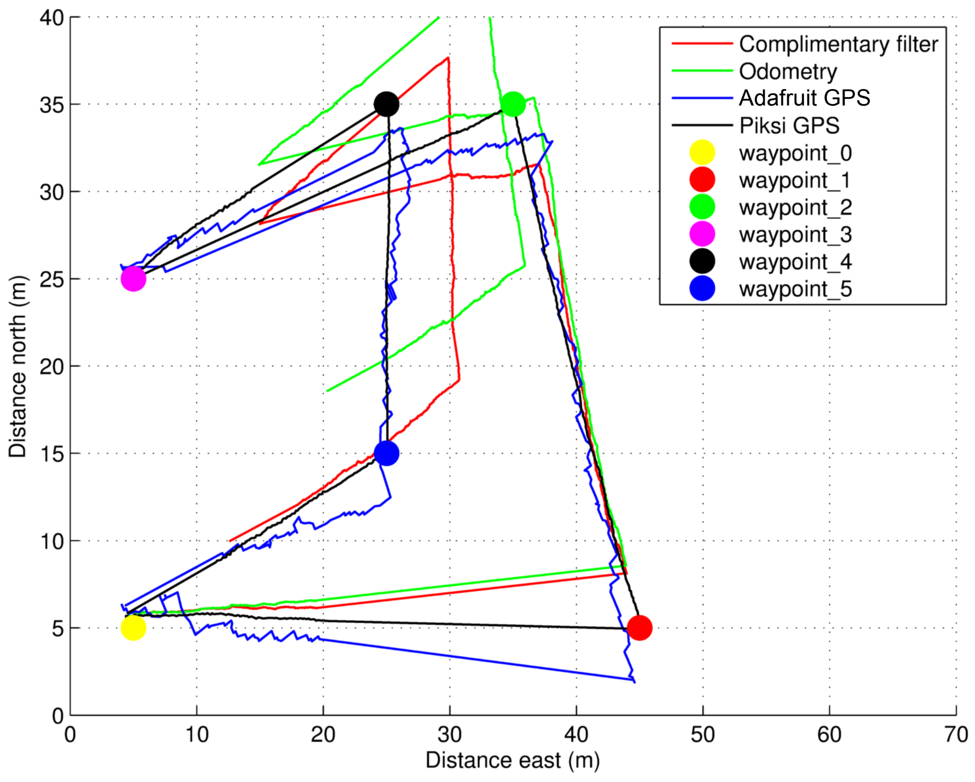


Figure 10-7: Case study 4 navigated map 60 rpm

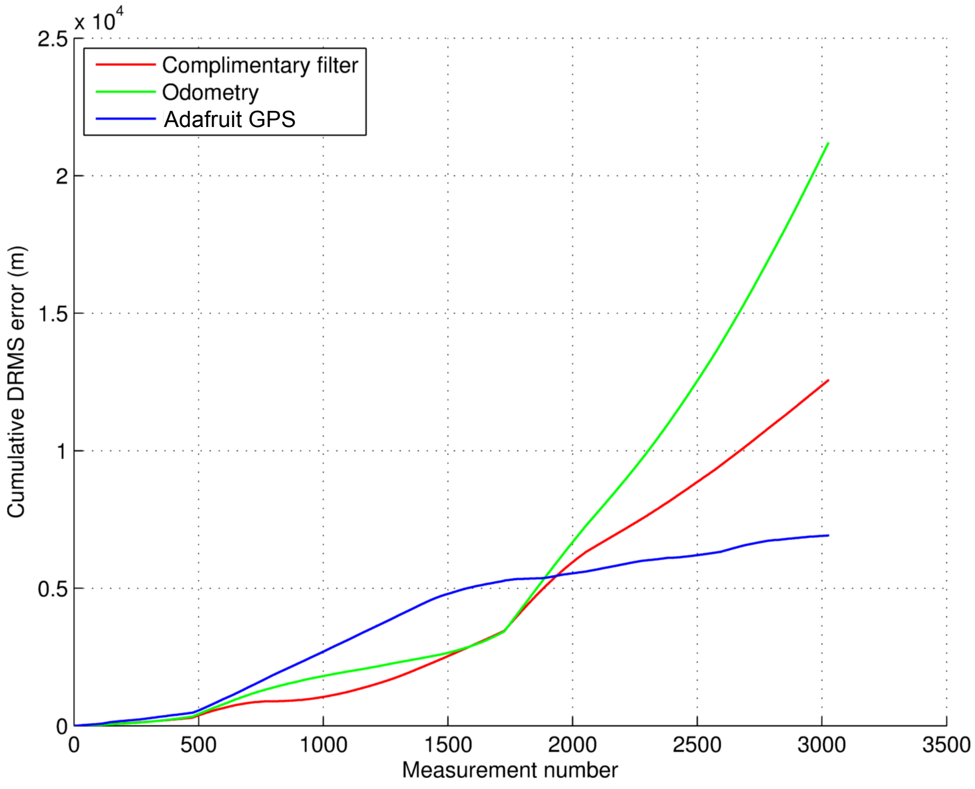


Figure 10-8: Case study 4 cumulative DRMS error 60 rpm

11 Appendix E: Case study 7

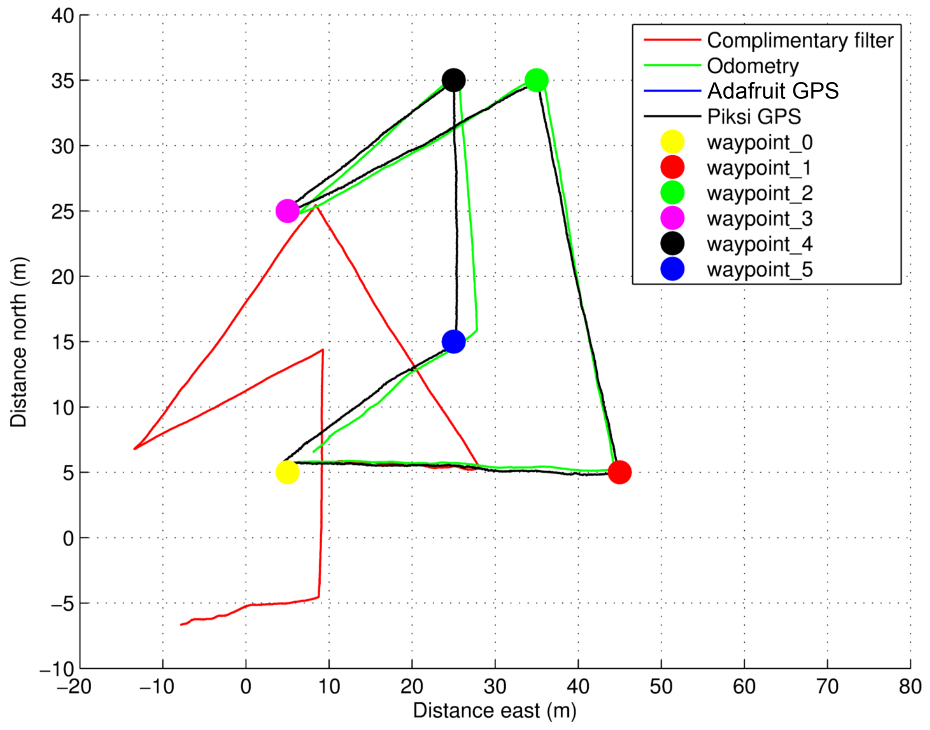


Figure 11-1: Case study 7 navigated map no Adafruit GPS

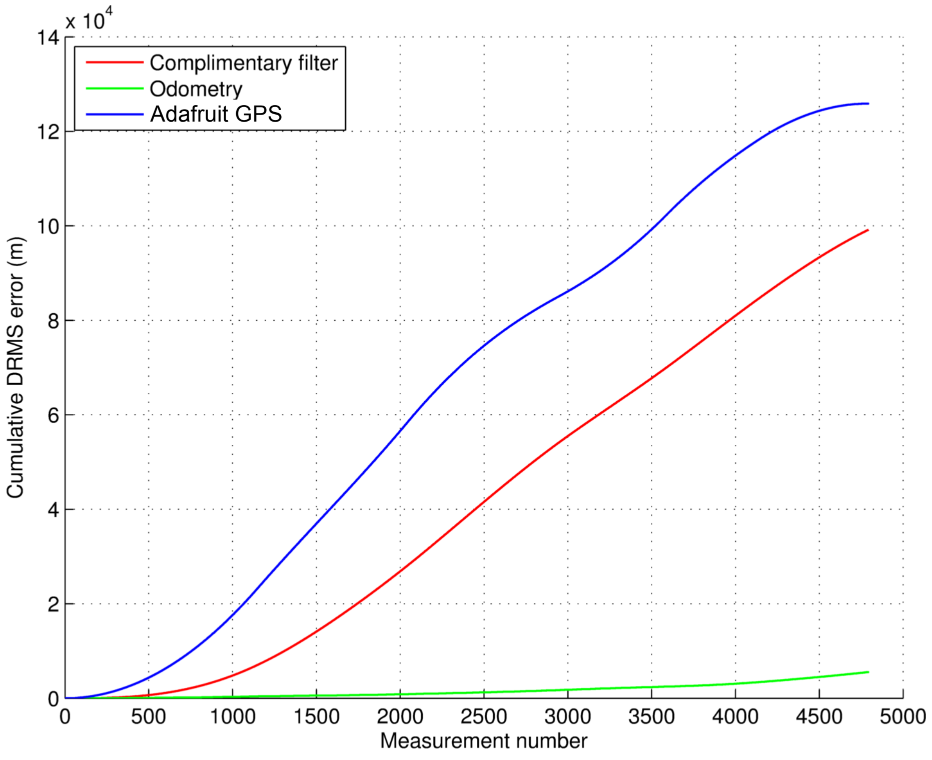


Figure 11-2: Case study 7 cumulative DRMS error no Adafruit GPS

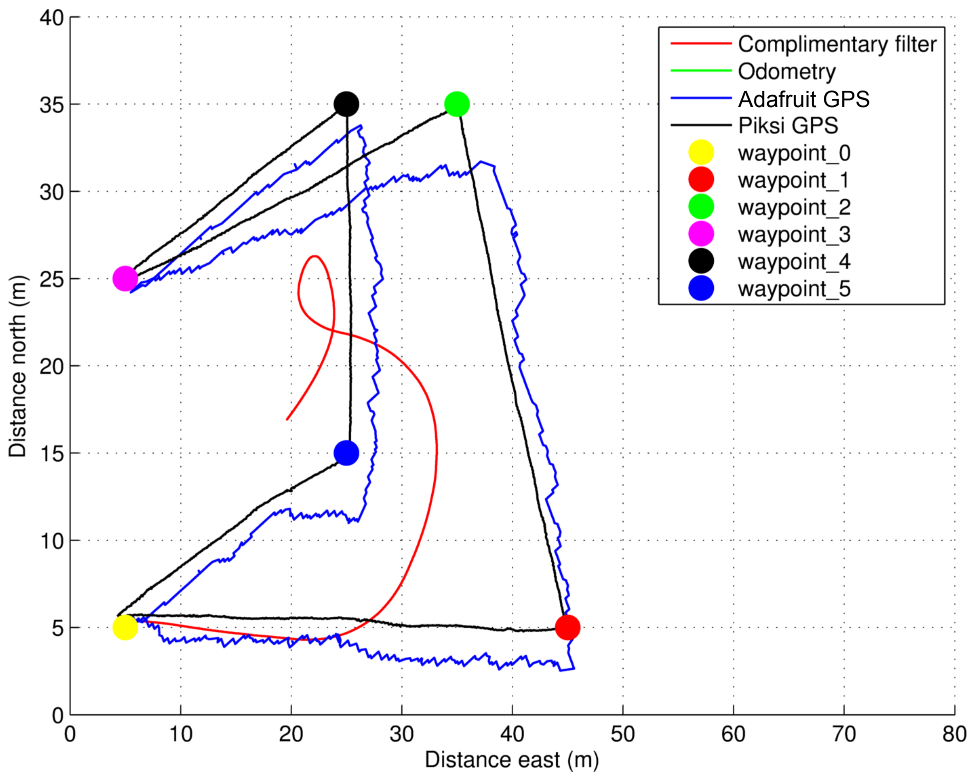


Figure 11-3: Case study 7 navigated map no encoders

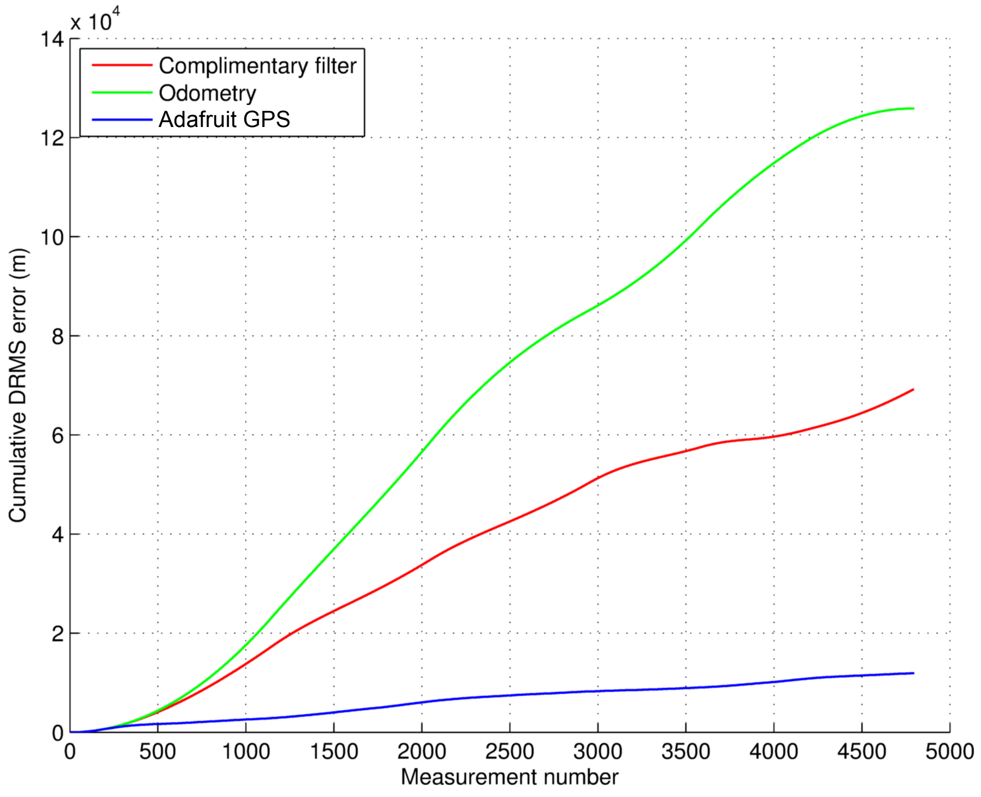


Figure 11-4: Case study 7 cumulative DRMS error no encoders

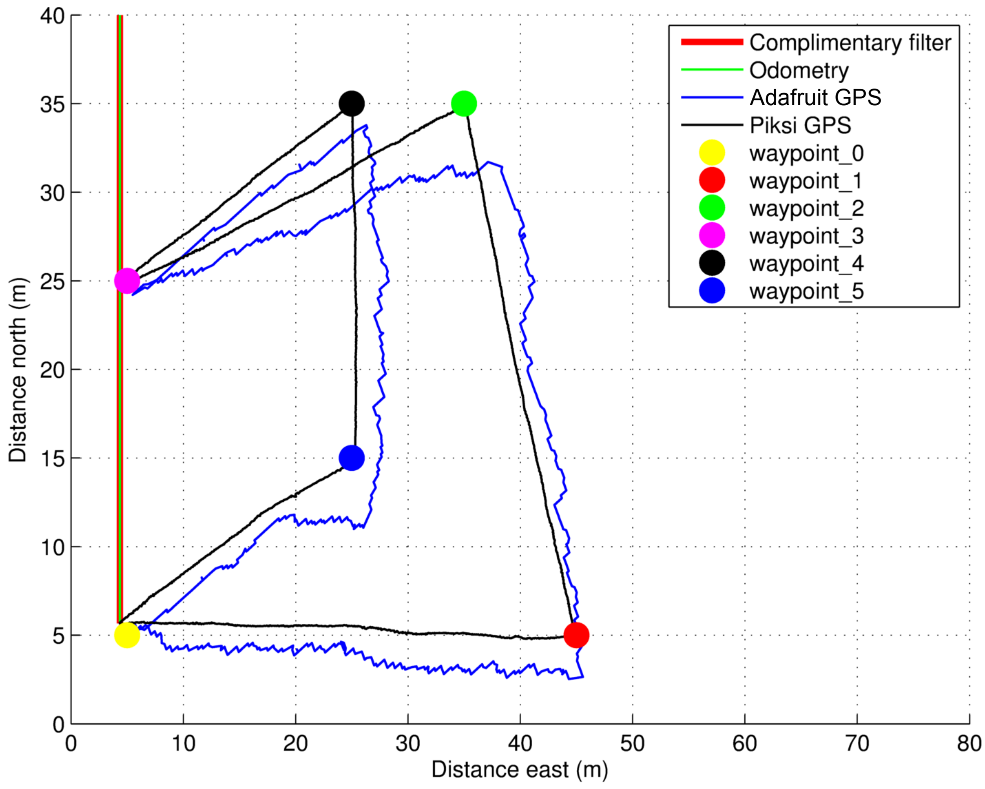


Figure 11-5: Case study 7 navigated map no compass

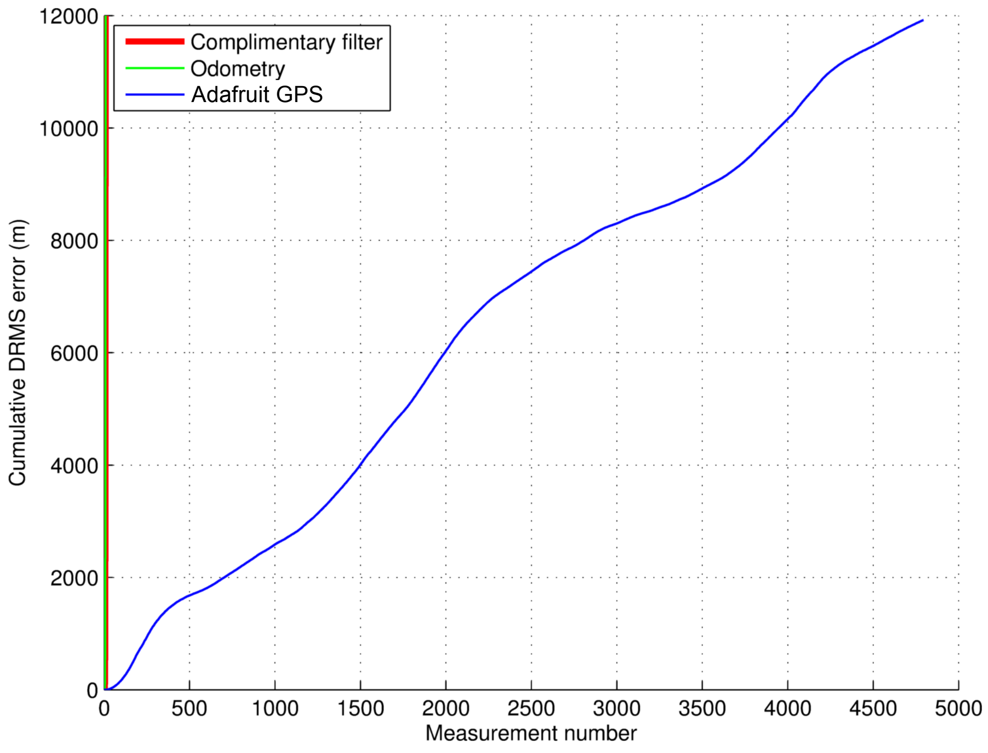


Figure 11-6: Case study 7 cumulative DRMS error no compass

12 References

1996 Federal Radionavigation Plan. (1997). 9th ed. Department of Transportation and Department of Defense, p.A-7.

Artese, G. and Trecroci, A. (2008). CALIBRATION OF A LOW COST MEMS INS SENSOR FOR AN INTEGRATED NAVIGATION SYSTEM. *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, XXXVII(Part B5), p.878.

Åström, K. and Murray, R. (2008). *Feedback systems*. Princeton: Princeton University Press, p.296.

Bell, S. (1999). *A Beginner's Guide to Uncertainty of Measurement*. [ebook] Available at: https://www.wmo.int/pages/prog/gcos/documents/gruanmanuals/UK_NPL/mgpg11.pdf [Accessed 25 Jan. 2016].

Borenstein, J. and Liqiang Feng, (1996). Measurement and correction of systematic odometry errors in mobile robots. *IEEE Trans. Robot. Automat.*, 12(6), pp.869-880.

Bromiley, P. (2014). *Products and Convolutions of Gaussian Probability Density Functions*. 1st ed. [ebook] Manchester: Imaging Sciences Research Group, pp.2-7. Available at: http://www.tina-vision.net/docs/memos/2003-003.pdf [Accessed 5 Apr. 2015].

Edu-observatory.org, (2015). *Sam Wormley's GPS Errors & Estimating Your Receiver's Accuracy*. [online] Available at: http://www.edu-observatory.org/gps/gps_accuracy.html [Accessed 11 Feb. 2015].

Goel, P., Roumeliotis, S. and Sukhatme, G. (2015). Robot Localization Using Relative and Absolute Position Estimates. p.6.

Gps.gov, (2015). *GPS.gov: Performance Standards & Specifications*. [online] Available at: http://www.gps.gov/technical/ps/ [Accessed 23 Feb. 2015].

Greenwood, D. (2003). *Advanced dynamics*. Cambridge, U.K.: Cambridge University Press, p.144.

Higgins, W. (1975). A Comparison of Complementary and Kalman Filtering. *IEEE Transactions on aerospace and electronic systems,* Vol. AES-11 No. 3, p.324.

Hirt, A. (2015). *Accuracy of the Swift-Navigation Piksi differential GPS*. [online] Available at: https://groups.google.com/group/swiftnav-discuss/attach/d3fb5e94b44f3/Validierung.pdf?part=0.1 [Accessed 26 Jan 2016].

JCGM 100:2008. (2008). Evaluation of measurement data - Guide to the expression of uncertainty in measurement, p.27.

Kiran, N. and Raja, C. (2014). Improved Dynamic Response of Buck Converter using Fuzzy Controller. *Bulletin of Electrical Engineering and Informatics*, 3(1), p.29.

Kalman, R. (1960). A New Approach to Linear Filtering and Prediction Problems. *Journal of Basic Engineering*, 82(1), p.35.

Learn.sparkfun.com, (2015). *Data Types in Arduino - learn.sparkfun.com*. [online] Available at: https://learn.sparkfun.com/tutorials/data-types-in-arduino [Accessed 12 Feb. 2015].

Leonard, J. and Durrant-Whyte, H. (1991). Mobile robot localization by tracking geometric beacons. *IEEE Trans. Robot. Automat.*, 7(3), pp.376-382.

Letsmakerobots.com, (2011). *Kalman filter vs Complementary filter*. [online] Available at: http://letsmakerobots.com/node/29121 [Accessed 26 Jan 2016].

Li, W., Yuan, Y., Ou, J., Chai, Y., Li, Z., Liou, Y. and Wang, N. (2014). New versions of the BDS/GNSS zenith tropospheric delay model IGGtrop. *J Geod*, 89(1), p.78.

McCarthy, J. and Soh, G. (2011). *Geometric design of linkages*. New York: Springer, p.190.

Merrouni, A., Wolfertstetter, F., Mezrhab, A., Wilbert, S. and Pitz-Paal, R. (2015). Investigation of Soiling Effect on Different Solar Mirror Materials under Moroccan Climate. *Energy Procedia*, 69, p.1957.

Merry, C. (2007). Augmentation systems for GPS. *PositionIT*, p.63.

Meyer, C. (2000). *Matrix analysis and applied linear algebra*. Philadelphia: Society for Industrial and Applied Mathematics, p.547.

Oc.nps.edu, (2015). *Differential GPS:*. [online] Available at: http://www.oc.nps.edu/oc2902w/gps/dgpsnote.html [Accessed 10 Feb. 2015].

Parameters and calibration of a low-g 3-axis accelerometer. (2014). 1st ed. [ebook] ST, p.8. Available at: http://www.st.com/st-web-ui/static/active/cn/resource/technical/document/application_note/DM00119044.pdf [Accessed 7 May 2015].

Parkinson, B. (1996). *Global Positioning System: Theory and Applications, Volume 1*. AIAA, p.67.

Physics.nist.gov, (2016). *Expanded uncertainty and coverage factor*. [online] Available at: http://physics.nist.gov/cuu/Uncertainty/coverage.html [Accessed 25 Jan. 2016].

Piksi Datasheet. (2013). 2nd ed. [ebook] Swift Navigation, p.1. Available at: http://docs.swiftnav.com/pdfs/piksi_datasheet_v2.3.1.pdf [Accessed 4 May 2015].

Power Technology, (2015). *Solar Tower, Seville*. [online] Available at: http://www.power-technology.com/projects/Seville-Solar-Tower/ [Accessed 16 Jan. 2015].

RIETDORF, A., DAUB, C. and LOEF, P. (2006). Precise Positioning in Real-Time using Navigation Satellites and Telecommunication. *PROCEEDINGS OF THE 3rd WORKSHOP ON POSITIONING, NAVIGATION AND COMMUNICATION*, 06, p.124.

Roze, A., Zufferey, J., Beyeler, A. and McClellan, A. (2015). *eBee RTK Accuracy Assessment*. 1st ed. [ebook] Available at: https://www.sensefly.com/fileadmin/user_upload/sensefly/documents/eBee-RTK-Accuracy-Assessment.pdf [Accessed 21 Apr. 2015].

Thetapestore.co.uk, (2016). *Class 1 Tape Measures - EC Class I*. [online] Available at: http://www.thetapestore.co.uk/tapes-rules/tape-measures/tape-accuracy/class-1-tape-measures [Accessed 24 Jan. 2016].

Varner, C. (2000). *DGPS Carrier Phase Networks and Partial Derivative Algorithms*. Graduate. University of Galgary, p.26.

Vincenty, T. (1975). DIRECT AND INVERSE SOLUTIONS OF GEODESICS ON THE ELLIPSOID WITH APPLICATION OF NESTED EQUATIONS. *Survey Review*, 23(176), p.89.

Wescott, T. (2000). *PID Without a PhD*. 1st ed. [ebook] Embedded Systems Programming, p.5. Available at: http://m.eet.com/media/1112634/f-wescot.pdf [Accessed 10 Apr. 2015].

Ziegler, J. and Nichols, N. (1993). Optimum Settings for Automatic Controllers. *J. Dyn. Sys., Meas., Control*, 115(2B), p.765.