

Simen Kartveit Bjerkestrand

UGV Dead Reckoning in GNSS-Denied Environments

Bachelor's thesis in Electrical Engineering

Supervisor: Erlend Coates

Co-supervisor: Eirik Fagerhaug

May 2022

Simen Kartveit Bjerkestrand

UGV Dead Reckoning in GNSS-Denied Environments

Bachelor's thesis in Electrical Engineering
Supervisor: Erlend Coates
Co-supervisor: Eirik Fagerhaug
May 2022

Norwegian University of Science and Technology
Faculty of Engineering
Department of ICT and Natural Sciences



NTNU

Kunnskap for en bedre verden

Preface

This bachelor thesis is written by a student completing a bachelors degree in electrical engineering and majoring in automation. This thesis gives 20 credits, completing the necessary 180 credits needed to complete the bachelors degree. This thesis is about localization, the ability of a robot to establish its position and orientation within a frame of reference. This type of technology is at the core of cybernetics and state estimation. My interest for this field of study is what intrigued me to do this project, and what motivates me to continue with further studies after the semester.

Acknowledgement

I would like to thank my supervisors Erlend Coates and Eirik Fagerhaug for valuable guidance, helpful discussions, and all types of help during the project. I would also like thank Nikolai Persen for helping me with technical problems I had during the project.

April 19, 2022, NTNU

Simen Bjerkestrand

Abstract

The goal of this project is to find a combination of sensors and filters to find out how few sensors are needed for dead reckoning of an ackerman based autonomous load handling vehicle. In this project, tests were performed on a simulated mobile robot model to see how wheel odometry and IMU performed in a dead-reckoning scenario. The sensors were individually evaluated based on when, and how fast the estimated position drifted away from the true position. Furthermore, applications of sensor fusion, and different state estimation filters were also evaluated. It was shown that both wheel odometry, and IMU performed relatively poorly on their own, and were sensitive to errors in initialization. The combination of wheel odometry, and a directional gyroscope provided the best performing localization, but still has room for improvement. Wheel odometry and IMU was also attempted, but showed little to no improvements over wheel odometry and directional gyroscope.

Contents

Preface	i
Acknowledgement	ii
Summary and Conclusions	iii
Acronyms	2
1 Introduction	5
1.1 Background	6
1.2 Problem Formulation	6
1.3 Literature Survey	6
1.4 Objectives	7
1.5 Limitations	7
1.6 Contributions	8
1.7 Structure of the Report	8
2 Theoretical basis	10
2.1 Navigation Technologies	10
2.1.1 Position fixing	10
2.1.2 Dead Reckoning	10
2.1.3 INS	11
2.1.4 Inertial sensors	12
2.1.5 Wheel Odometry	12
2.2 State Estimation	13
2.2.1 Standard Kalman Filter	13
2.2.2 Extended Kalman Filter	15

2.2.3	Unscented Kalman Filter	16
2.3	Position-Fixing and Dead-Reckoning integration	17
2.4	Improvement Techniques	17
2.4.1	Dynamic Process Noise Covariance	17
2.4.2	Steering Input	18
2.4.3	Motion Constraints	18
2.4.4	Systematic Calibration	18
2.5	Ackerman Vehicle	19
2.6	Ackerman Vehicle Kinematics	20
2.7	Quaternions	21
2.8	Simulation	22
2.8.1	Gazebo	22
2.8.2	Robot Operating Software (ROS2)	22
2.8.3	robot_localization package	23
2.8.4	Standard units of measure and Coordinate Conventions	23
2.9	Evaluation	25
2.9.1	Satisfactory performance	26
3	Method	28
3.1	Software	28
3.1.1	Operating System	29
3.1.2	Robot Operating Software 2	29
3.1.3	Gazebo	30
3.2	Simulation	30
3.2.1	Gazebo Plugins	30
3.2.2	Ackerman Vehicle	30
3.2.3	Ackerman Steering	31
3.2.4	Ackerman Wheel Odometry	31
3.2.5	Inertial Measurement Unit	34
3.2.6	Ground Truth	34

3.2.7	Architecture	35
3.3	State Estimation	37
3.3.1	Sensor fusion	37
3.3.2	Tuning	40
3.4	Testing	40
3.5	Evaluation	42
4	Result	44
4.1	Wheel Odometry	44
4.2	IMU	46
4.3	State estimation	48
4.3.1	Wheel Odometry	48
4.3.2	IMU	49
4.3.3	Sensor Fusion	51
4.3.4	Improvements	54
4.4	Performance	63
4.5	Challenges	69
4.5.1	Update Rate Errors	69
4.5.2	Qos (Quality of service) parameter mismatch	70
5	Discussion	73
5.1	Sensors	73
5.1.1	Wheel Odometry	73
5.1.2	Inertial Measurement Unit	73
5.1.3	Sensor Fusion	74
5.2	Unscented Kalman Filter	74
5.3	Process Noise Covariance	74
5.4	Steering Input	75
5.5	Dynamic Process Noise Covariance	75
5.6	Shortcomings	75
5.7	Objectives	76

<i>CONTENTS</i>	1
6 Conclusions and further work	77
Bibliography	79
Appendices	82
A Preproject report(Norwegian)	82
B Robot Localization proposed bachelor or master thesis	90
C Gazebo URDF	93
D Ackerman Odometry Plugin Source Code	98
E Kalman Filter Parameter File(YAML)	104

Terminology

ackerman A type of vehicle steering geometry(see chapter 2.5). iii

dead reckoning The use of integrated Odometry to determine position(see chapter 2.1.2). iii,
5, 7

Odometry The use of motion sensors to determine change in position. 2

Position fixing Type of positioning measurement that gives absolute measurements of position(see chapter 2.1.1). 5, 10

Notation

Δd_l Arc Length of left wheel. 20

Δd_r Arc Length of right wheel. 20

$\Delta \theta_l$ Rotational change in left wheel in radians. 20

$\Delta \theta_r$ Rotational change in right wheel in radians. 20

Δc_l Encoder Measurements of left wheel. 20

Δc_r Encoder Measurements of right wheel. 20

E_{res} Encoder resolution. 20

WR_l Radius of left wheel. 20

WR_r Radius of right wheel. 20

Abbreviations

EKF Extended Kalman Filter. 7, 16, 17, 23

GNSS Global Navigation Satellite Systems. 5, 10, 16, 17, 26, 27

GPS Global Positioning System. 7

IMU Inertial Measurement Unit. iii, vi, 7, 12, 39, 65, 73

INS Inertial Navigation System. iv, 7, 11, 16, 17

UKF Unscented Kalman Filter. 7, 16, 17, 23, 64

Chapter 1

Introduction

The need for Global Navigation Satellite Systems (GNSS) in most navigation systems, is the inherent reason why GNSS plays a vital role in today's society. GNSS receivers allow us to find the approximate position of an agent anywhere on earth, but there are many sources of error present in a GNSS receiver. For example, when there is no direct line of sight between the satellites and the receiver, or when signals are reflected off of buildings which causes multipath[3]. In absence of GNSS, or other Position fixing systems, dead reckoning is often used. Dead-reckoning is the method of calculating position from a known previous position, and incorporating estimates of speed and heading over elapsed time.

Redrock.AI, an organization that designs autonomous load handling vehicles, wants to implement dead-reckoning on some of the clients' autonomous vehicles. Their requirements can be easily met with costly localization methods, but they want to know how small of an investment is enough to satisfy their dead-reckoning needs. This is to say, how few sensors and how inaccurate sensors can be used but still be accurate enough. The accuracy of a particular dead-reckoning system can be a result of operational factors in the environment, the quality of the inertial sensors and the implemented localization algorithm. Therefore, one of the primary focuses of this thesis is to find a good localization algorithm. There are many things that can improve the performance of a localization algorithm. This could for example be using different filters, or taking into account inherent properties of the robot, like motion constraints, robot dynamics, or even the steering input of the robot. Using all the information we have at our disposal, could allow for the use of fewer sensors, and therefore reducing the overall cost.

1.1 Background

Redrock AI. is a firm that designs autonomous load handling solutions for their clients. This project tries to answer to one of several proposed bachelor or masters thesis that were given by Redrock AI. The proposed thesis is shown in Appendix B. The proposed thesis outlines a literary review of state-of-the-art localization models, and the testing of odometry, gyroscopes, and accelerometers on a simulated mobile robot.

1.2 Problem Formulation

The aim of this thesis is to evaluate the performance of localization that uses different sensors and filters on an Ackerman based vehicle. When the term "performance" is used, It is meant to signify to which degree some localization is accurate and precise to its true value. Another aim is to evaluate the performance of sensor fused dead-reckoning. The purpose of this, is to create a localization that is satisfactory, but still using the fewest amount of sensors possible. In this thesis, the term "satisfactory" is not strictly defined, but logical conclusions will be drawn to what is satisfactory(see chapter 2.9.1).

1.3 Literature Survey

Many papers have claimed that fusing absolute position measurements with odometry data provide more accurate and more reliable position estimation[8][13][18][7], and most researchers agree that odometry is an important part of mobile robot navigation systems[7]. Odometry data still has inherent drawbacks as stated by Everett B. Borenstein:

"Improvements in odometry techniques will not change their incremental nature, i.e., even for improved odometry, periodic absolute position updates are necessary."[7]

Some papers on the use of accelerometers have been written aswell[7][4][17]. A study investigated the suitability of accelerometers for mobile robot positioning at the University of Michigan, and it was found that there is a very poor signal-to-noise ratio in low speed turns[7]. *Barshan and Durrant-Whyte*[4] implemented a tilt compensated accelerometer that resulted in position drift between 1 to 8cm/s, which was deemed as unacceptable error for most mobile robot

applications. A newer study has claimed possible improvements to high-order noise accelerometers by coupling them with an Artificial Neural Network[17].

Many papers has also been written on the EKF and UKF[6][24][20][21]. *Vadim Bistrov and Ansis Kluga*[6] has analyzed a UKF-based navigation algorithm with low cost GPS and INS sensors, concluding that a UKF based algorithm has less velocity error than its EKF counterpart. Other papers have also discussed the differences in EKF and UKF based Navigation algorithms [24][20]. *Ridolfi, B. Allotta*[24] claims the superiority of the UKF, especially when there are few available sensors. *Sabet, Mohammad Taghi El al.*[21] utilized an EKF with low cost microelectromechanical inertial sensors.

1.4 Objectives

To evaluate the dead reckoning performance of a localization we first need to evaluate the capabilities of each specific sensor combination. In this thesis, the following research questions will be answered for the use of wheel odometry, IMU, as well as fused wheel odometry and IMU.

- When does the position start to drift?
- How much and how fast does the position drift?

The answers to these questions could help guide the further design of the localization, and answer the following research questions.

- How does dead-reckoning affect localization in absence of GNSS?
- Which sensors must be used to maintain acceptable performance?
- How few sensors are needed to maintain satisfactory performance?

1.5 Limitations

The scope of this project is primarily to simulate the localization, and although the noise models of the simulation are accurate to some degree, there is a limitation to how realistic the models

are. An example of this is how it is very hard to replicate a IMU(Inertial Measurement Unit) measurements, where IMUs in real life can exhibit complex high-order time varying noise. Further work could examine the validity of the results in the real world.

1.6 Contributions

This project will have as its goal to give insight into different possibilities of dead-reckoning. The results of wheel odometry is also especially useful as it shows the specific results for the vehicle model being used. This is useful because different vehicles have different slippage properties, depending on different dimensions of the vehicle and weight.

1.7 Structure of the Report

The rest of the report is structured as follows.

Chapter 2 - Theoretical basis: Chapter two first introduces the mathematical background to the Ackerman based vehicle. Furthermore, A literary review is done on different localization models, which includes types of navigation technologies, sensors, and filters. Lastly, a quick exploration on different ways to evaluate the results.

Chapter 3 - Method: Chapter three begins with the basics behind the software. Moreover, It is shown how the software can be used to simulate a vehicle. Then, it is shown how Gazebo models and plugins were implemented. Next, it is shown how the odometry of the simulated vehicle is filtered through different types of filters, and how the parameters of these filters were tuned.

Chapter 4 - Result: Chapter four contains the localization results of four different combination of sensors. Additionally, the results of changing different parameters and filters types are shown. Lastly, a overview of the challenges met during the project.

Chapter 5 - Discussion: In this chapter, the results are reviewed and compared to published literature that were mentioned in the literary review and theory chapters of the paper. Eventual sources of error are drafted and discussed.

Chapter 6 - Conclusions: This chapter presents a conclusion, where the research questions in the introduction are addressed, and possible further work.

Chapter 2

Theoretical basis

This chapter will cover the basis for localization technologies used today. State estimation filters, as well as eventual techniques that can be used to improve the overall performance of localization are drafted. Lastly, a brief overview of the mathematics that are needed later will be introduced, and then techniques for simulating and evaluating the mobile vehicle.

2.1 Navigation Technologies

2.1.1 Position fixing

Position fixing is a type of positioning that determines the absolute position of some agent. There are five different types of position-fixing technologies. Proximity, ranging, angular positioning, pattern matching, and doppler positioning [10]. GNSS receivers are considered proximity type position fixing devices.

2.1.2 Dead Reckoning

Dead-reckoning (derived from "deduced reckoning"[10]) uses measurements of change in position, or with integrated velocity, and uses the measurements to determine the position with respect to a known previous position.

When dead-reckoning, change in position is often in reference to the body-aligned axis of the agent being localized, and therefore needs a separate solution to produce position in rela-

relationship to the map or world frame. This makes sense, because inertial sensors will give measurements in respect to the body frame(see figure 2.7).

Some properties of dead-reckoning solutions are that they often have low short-term noise, continuous operation and fast update rates compared to traditional position fixing methods. The main drawbacks are that an initial position is needed, and that errors in measurements accumulate over time which gives large errors over large periods of time. Position fixing systems are often integrated with dead-reckoning systems and are used to correct and calibrate dead-reckoning errors(more in chapter 2.3).

2.1.3 INS

A INS(Inertial Navigation System) is a type of dead-reckoning system that consists of inertial sensors, gyroscope sensors, and an integrated circuit that handles the sensor data. Inertial sensors are coupled with a gravity model to compensate for the effect gravity has on measurements. Figure 3.2.4 shows an example of an INS.

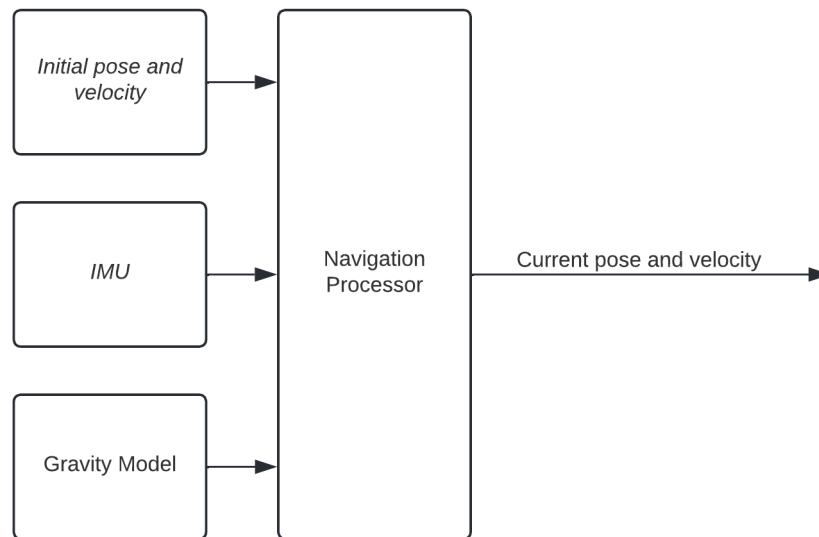


Figure 2.1: Example of an Inertial Navigation System(INS)

The most common applications of INS use strapdown INS. This is when the INS measurements are relative to some rigid body frame, as opposed to a map or odometry frame.

2.1.4 Inertial sensors

Accelerometers, and gyroscopes are two types of inertial sensors that measure linear and angular acceleration respectively. The accuracy of inertial sensors can vary on the scope of several orders of magnitude. There exists cheap inertial sensors that are often used in consumer devices, medium accuracy inertial sensors that can be used for navigation, and lastly, Military-grade inertial sensors that can have total position error drift that is less than 1500m in the span of an hour[10].

A subcategory of sensors that are very often used are called MEMS(Microelectromechanical systems)[25]. These sensors are often very inexpensive and are built up of components that are between 1 and 100 micrometers in size [9]. Although these sensors are relatively inexpensive, they can exhibit high levels of noise. MEMS type IMUs can exhibit errors that are of high-order complexity, and that have slow time-varying properties. Rate-ramp, random walk, and bias instability are among other things errors that can also arise in MEMS inertial sensors.

A possible solution to these errors are to model them with a ANN(Artificial neural network)[17]. Improvements to position accuracy have been demonstrated when integrating ANN/Kalman filters in the presence of MEMS IMUs[2]. Errors can also be analysed by applying Gauss-Markov, auto-regressive, Power Spectral Density methods[1]. Inertial sensors that have unknown noise can also be filtered with an adaptive Kalman filter[12]. Lastly, MEMS type inertial sensors are constantly developing, therefore, one should take care to use the newest technology available.

2.1.5 Wheel Odometry

Wheel odometry is claimed to be an important part of all mobile robot navigation systems[7]. Wheel odometry is the use of inexpensive incremental wheel encoders to produce estimates of velocity. An assumption that is made, is that rotational movement in the wheels, directly correlate with the velocity of the vehicle. Often, wheel odometry will produce false measurements of velocity when they slip. Driving wheels are especially susceptible to slippage as they are often wide, and support the weight of the vehicle. Wheel slip is an example of a Non-Systematic Error, which is one of the two types of odometry errors: systematic and non-systematic errors.

Non-Systematic Errors

As mentioned, wheel slippage is a type of non-systematic error. This is because these types of error are unpredictable and are dependent on the environment. Other types of non-systematic error includes travel over uneven surfaces, and over unexpected objects.

Solutions like following encoders[7] have been implemented with non load carrying, narrow aluminium wheels that have shown to produce more accurate odometry measurements as they reduce slippage. Slippage can also be attempted to be detected with integrity monitoring[10].

Systematic Errors

Systematic errors are types of error that make error accumulate constantly[7]. These errors consists of unequal or wrong wheel radii, wrong wheelbase, misaligned wheels, or encoder resolution and sampling rate. These are errors that are important to correct because they cause rapidly accumulated errors[7].

2.2 State Estimation

There are many estimation algorithms that can be used for the purpose of navigation, although the Kalman filter is the most widely used[10]. The Kalman Filter can provide a statistically optimal estimate of the states of a plant when the measurements have uncertainty.

The Kalman filter is a recursive Bayesian estimation(Bayes filter) filter[10]. The Bayes filter algorithm is used to predict the most likely position, along with the probability distribution of that prediction. When the distribution of the bayes filter states are normally distributed, the filter then becomes equivalent to a standard Kalman filter.

2.2.1 Standard Kalman Filter

As mentioned in chapter 2.2, the Kalman filter is a type of bayesian filter. In practice, this means that the filter is separated into two stages, the prediction and innovation stage(see figure 2.2).

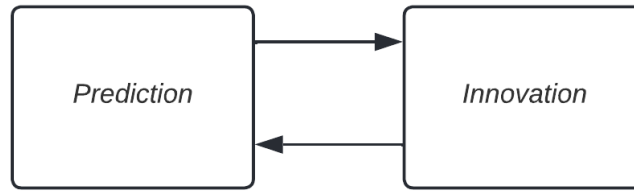


Figure 2.2: Kalman filter stages

The first stage of the filter is the prediction stage. Equations 2.1 and 2.2 are the equations used to calculate the predicted states(2.1) and their uncertainties(2.2).

$$\hat{x}_k^- = A\hat{x}_{k-1} + Bu_{k-1} \quad (2.1)$$

$$P_k^- = AP_{k-1}A^T + Q \quad (2.2)$$

In equation 2.1, the states of the plant(\hat{x}_k^-) are predicted based on previous state estimates(\hat{x}_{k-1}) and the model of the plant(A), as well as plant input¹. In the second equation 2.2, The error covariance estimate(P_k^-) is derived from the model of the plant(A) and the process noise covariance(Q).

The next stage of the filter is the innovation stage. This is the "Correction" stage of the filter, where the predicted states and uncertainties are corrected with updated sensor measurements. *Kalman Gain*(K_k)(calculated in equation 2.3) as it is often referred to is a matrix of weights that decides how much to weight new measurements, and how much to weight predictions.

$$K_k = P_k^- H^T (HP_k^- H^T + R)^{-1} \quad (2.3)$$

Equation 2.4 updates the state estimates with measurements, previous estimates, and kalman gain. Equation 2.5 updates the error covariances with the Kalman gain. This concludes the equations that are used for the standard Kalman filter.

¹the u_{k-1} is the input of the prediction and often contains acceleration terms

$$\hat{x}_k = \hat{x}_k^- + K_k (z_k - H\hat{x}_k^-) \quad (2.4)$$

$$P_k = (I - K_k H) P_k^- \quad (2.5)$$

2.2.2 Extended Kalman Filter

The Extended Kalman Filter is an extension of the standard Kalman Filter described in section 2.2.1. In the standard Kalman Filter, the system model of the plant(A) is assumed to be linear. When the model of the plant is not linear, an Extended Kalman Filter may be used instead.

The Filter equations are similar to standard kalman filter equations, the main difference being that the state transition matrix, and observation matrix has to be linearized(equation 2.7 and 2.9) for each iteration.

$$\hat{x}_{k|k-1} = f(\hat{x}_{k-1|k-1}, u_k) \quad (2.6)$$

$$F_k = \left. \frac{\partial f}{\partial x} \right|_{\hat{x}_{k-1|k-1}, u_k} \quad (2.7)$$

$$\tilde{y}_k = z_k - h(\hat{x}_{k|k-1}) \quad (2.8)$$

$$H_k = \left. \frac{\partial h}{\partial x} \right|_{\hat{x}_{k|k-1}} \quad (2.9)$$

Unlike the standard or linear kalman filter, the extended Kalman Filter is not an optimal estimator. If the initial state estimates are wrong, the filter can quickly diverge. The estimated covariance matrix often underestimates the true values of the covariance matrix, but can be improved[14]. Despite this, the extended Kalman Filter is the de facto standard for navigation given its reasonable performance[10].

2.2.3 Unscented Kalman Filter

Fredrik. G. and Gustaf H.(2012)[11] says that the UKF has become a popular alternative to the EKF during the last decade.

It is shown that when the state transition and observation models are highly non-linear, that the extended Kalman Filter can give poor performance. In comparison, certain systems using UKF result in more accurate estimates of the true mean and covariance[15][11].

Julier Uhlmann[16] claims that "The extended Kalman filter (EKF) is probably the most widely used estimation algorithm for nonlinear systems. However, more than 35 years of experience in the estimation community has shown that is difficult to implement, difficult to tune, and only reliable for systems that are almost linear on the time scale of the updates. Many of these difficulties arise from its use of linearization."

The Unscented Kalman Filter(UKF) is another type of Kalman Filter used for nonlinear systems. In This filter, The probability density is determined by a sampling of points which represents the mean and covariance of the GRV(Gaussian Random Variable), and propagated through a true non-linear model[23].

This is unlike the Extended Kalman filter where a linearized model is used. Several Papers have explored the differences of the EKF and the UKF for navigation[6][24][20]. *Vadim Bistrov and Ansis Kluga*[6] implemented a GNSS/INS integrated navigation system, and concluded that the UKF had lower velocity estimation error in comparison to the EKF. Furthermore, *Alessandro Ridolfi, B. Allotta*[24] claims the superiority of UKF, especially when there is a reduced set of available sensors.

As mentioned, the probability distribution in the UKF is represented by a sample of points. These are called sigma points[10].

The parameters α , κ , and β are used to tune the UKF. α and κ are used to control the spread of sigma points, and β is used to define the distribution of the state vector.

Lastly, although the UKF is proved to be more accurate in many situations, the EKF can be simpler, and less computationally expensive. Some papers also claim the contrary, claiming that

the computational complexity of a third order UKF is the same as of a first order EKF[23].

2.3 Position-Fixing and Dead-Reckoning integration

There are many ways to integrate position-fixing and dead-reckoning systems into a combined system. The most common way is to implement what is called "Loosely coupled" integration. A Loosely Coupled GNSS-INS is where position-fixing and dead-reckoning have two separate filters and two separate estimates. These are then combined to form a joint estimate of the position and uncertainty.

This thesis will only implement the dead-reckoning portion of this integration. The use of position-fixing is beyond the scope of this thesis, but the results of this thesis could in theory be combined with position-fixing measurements in a Loosely coupled integration model.

2.4 Improvement Techniques

2.4.1 Dynamic Process Noise Covariance

As mentioned in chapter 2.1.2, an dead-reckoning system will have deteriorating accuracy as time passes. In a Kalman filter, the covariance of the states also increases the longer the filter has been dead-reckoning as shown in figure 2.3.

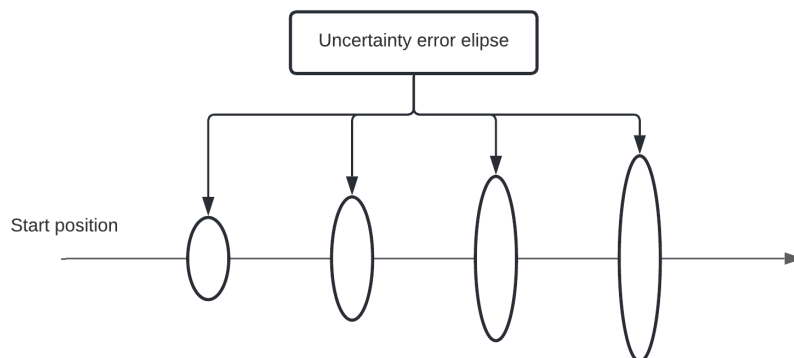


Figure 2.3: Error ellipse increase over time

An edge-case is when the vehicle is not in motion, where increasing the covariance of the po-

sition isn't expedient. It would be better to stop the increasing covariance when the vehicle is standing still.

2.4.2 Steering Input

Steering input can be used to predict the likely changes in position. Usually, the final velocity in the prediction state of the Kalman filter is a weighted average of the old and the new velocity. If control is also used, it can be converted to an acceleration term during the prediction. This can be useful if the vehicle is often accelerating or decelerating.

2.4.3 Motion Constraints

All vehicles have constraints, or common movement patterns that can be used to predict unlikely changes in position. A vehicle naturally has some limits to acceleration or speed, which can be used to lessen the uncertainty of the position. For vehicles that do not slip, velocity perpendicular to the vehicle can be interpreted as external sensor noise. Motion constraints go hand in hand with steering input(chapter 2.4.2) because steering inputs don't always directly correspond with vehicle movement. An example is when full throttle is given, but the predicted acceleration cannot be larger than the inherent maximum acceleration of the vehicle. This can also be affected by the weight of the vehicle, which will often change for a load handling vehicle.

2.4.4 Systematic Calibration

Everett B. Borenstein[7] mentions the use of systemic calibration. Systematic calibration is said to be the careful calibration of physical parameters to compensate for systematic errors in odometry(chapter 2.1.5). This is especially important for real life applications where wear can occur. It is also said to be a difficult to do since even minute deviations can cause substantial errors.

2.5 Ackerman Vehicle

An Ackerman vehicle has geometric arrangement of linkages to compensate for the need of inside and outside wheels of a vehicle to turn in circles with different radii. The intention of this is to avoid tire slippage in turns. An example of an Ackerman vehicle is shown in figure 2.5.

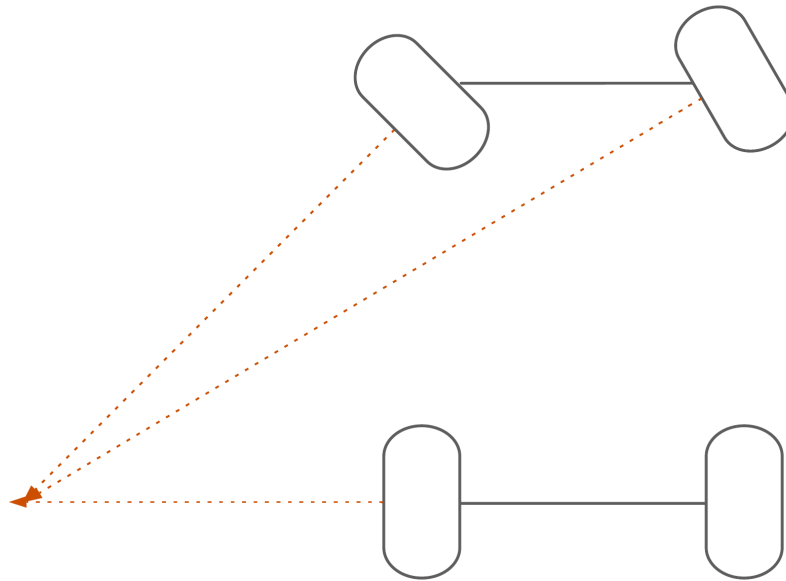


Figure 2.4: An Ackerman vehicle. The center of turning are the same for each front wheel. The turning angle for each wheel are different.

In comparison, a non-Ackerman vehicle is shown in figure 2.5.

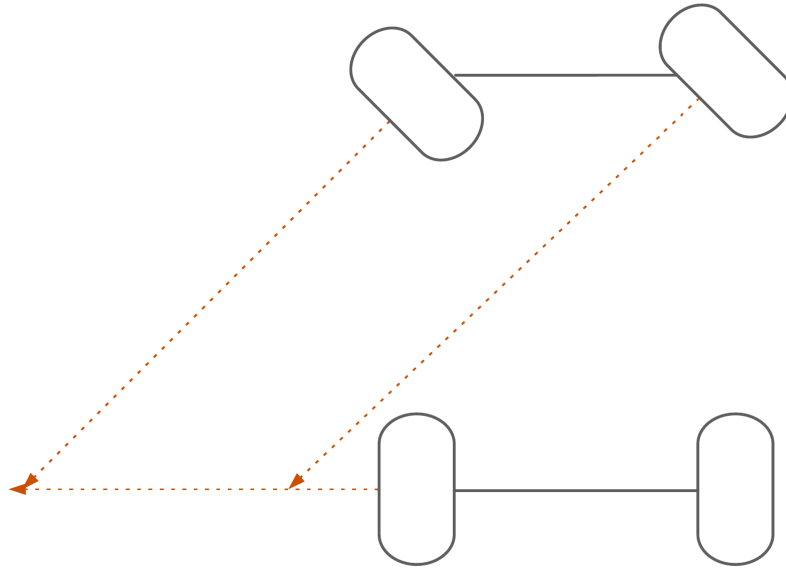


Figure 2.5: A non-Ackerman vehicle example. The center of turning are different for each front wheel, and the turning angles of each wheel is the same.

2.6 Ackerman Vehicle Kinematics

To calculate the odometry of an ackerman vehicle, we need to know the kinematics between the rotational movement of the wheels and the linear velocities of the vehicle. Firstly, the arc lengths($\Delta d_r, \Delta d_l$) for wheels have to be calculated using encoder measurements($\Delta c_r, \Delta c_l$)(E_{res} is the resolution of the encoder) and the radii of the wheels(WR_r, WR_l)(2.10)[5].

$$\begin{aligned}\Delta d_r &= \frac{2\pi(WR_r\Delta c_r)}{E_{res}} \\ \Delta d_l &= \frac{2\pi(WR_l\Delta c_l)}{E_{res}}\end{aligned}\quad (2.10)$$

When simulating a vehicle access to discrete encoder signals from the wheels might not be available. In order to use the rotation of the wheel directly, modification to equations 2.10 to use the change in radians($\Delta\theta_r, \Delta\theta_l$) as shown in equation 2.11 can be done instead.

$$\begin{aligned}\Delta d_r &= 2\pi(WR_r\Delta\theta_r) \\ \Delta d_l &= 2\pi(WR_l\Delta\theta_l)\end{aligned}\quad (2.11)$$

Given the Arc lengths, further calculation of the turning radius using equations 2.12[5] is done. Take note that since the timesteps are very small, a constant turning radius for each iteration can be calculated.

$$R_{\text{center}} = \frac{W_{\text{dis}}}{2} * \frac{\Delta d_r + \Delta d_l}{\Delta d_r - \Delta d_l} \quad (2.12)$$

The change in linear position can be extracted from the turning radius as shown in equation 2.13.

$$\begin{aligned} \Delta \Psi &= \frac{\Delta d_r - \Delta d_l}{W_{\text{dis}}} \\ \Delta x &= R_{\text{center}} (\cos(\Psi) \sin(\Delta \Psi) - \sin(\Psi)(1 - \cos(\Delta \Psi))) \\ \Delta y &= R_{\text{center}} (\sin(\Psi) \sin(\Delta \Psi) - \cos(\Psi)(1 - \cos(\Delta \Psi))) \end{aligned} \quad (2.13)$$

From here, it is only a matter of diving by the timestep to find the linear and angular velocities(see equations 2.14)

$$\begin{aligned} \dot{X} &= \frac{\Delta x}{\Delta t} \\ \dot{Y} &= \frac{\Delta y}{\Delta t} \\ \dot{\Psi} &= \frac{\Delta \Psi}{\Delta t} \end{aligned} \quad (2.14)$$

2.7 Quaternions

The quaternion is a mathematical notation for representing the rotation and orientation of a rigid body in three dimensions. It was a notation that was discovered by Sir William Rowan Hamilton in 1843[19]. This notation is represented by a fixed axis(also known as Euler's axis) along which the rigid body is pointing, and the rotation around Euler's axis.

In practice, the quaternion is given as a combination of four values. In some cases, the notation (x, y, z, w) is used, this is also the notation that will be used in this project. In the case of this notation, the first values (x, y, z) define Euler's axis, and the last value (w) represents the rotation around this axis.

2.8 Simulation

2.8.1 Gazebo

Gazebo is a simulation software with complex physics models to realistically simulate the real-world environment. Gazebo also has ROS2 compatibility.

2.8.2 Robot Operating Software (ROS2)

ROS2 is a middleware otherwise known as a DDS(Data Distribution Service For Real-Time Systems) and a framework for robot programming. ROS2 creates a foundation of functionality, so that users don't have to "reinvent the wheel" for each robotics project. ROS2 networks contain nodes and topics. Nodes are modules that can generate data or process data. A node can publish and subscribe to any number of topics. Topics enable data exchange between nodes. An example of a node publishing to one topic and subscribing to two other is shown in figure 2.6.

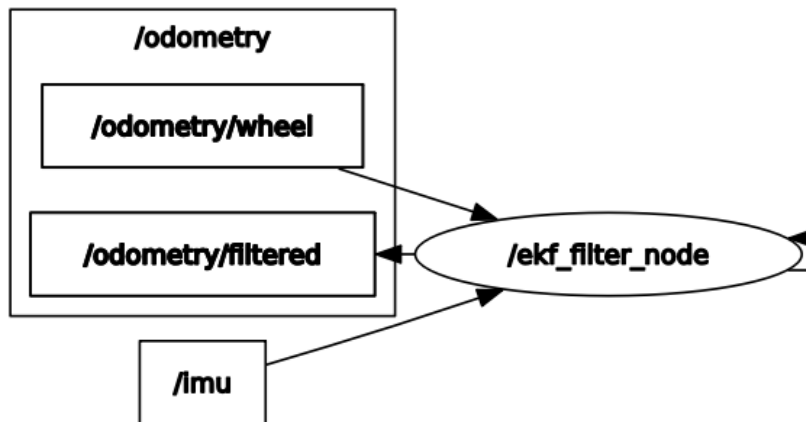


Figure 2.6: Example of a node publishing and subscribing to some topics. Square boxes are topics, and ellipses are nodes.

Nodes are language agnostic, which means that they can be coded in the users preferred coding language. This could for example be C++ or Python.

2.8.3 robot_localization package

The robot localization package² is a package for ROS2. It contains a collection of state estimation tools, each of which are nonlinear state estimator for moving robots. It features EKF, UKF type localization filters, which are implemented as nodes in the ROS2 architecture. There are also completed navigation implementations where parameters can be adjusted. As standard, the states used in the package are shown in 2.15.

The filter can handle an arbitrary amount of inputs of different data types (nav_msgs/Odometry, geometry_msgs/PoseWithCovarianceStamped, geometry_msgs/TwistWithCovarianceStamped, sensor_msgs/Imu).

$$(X, Y, Z, \text{roll}, \text{pitch}, \text{yaw}, \dot{X}, \dot{Y}, \dot{Z}, \dot{\text{roll}}, \dot{\text{pitch}}, \dot{\text{yaw}}, \ddot{X}, \ddot{Y}, \ddot{Z}) \quad (2.15)$$

2.8.4 Standard units of measure and Coordinate Conventions

REP 103³ and REP 105⁴ define the units, coordinate conventions, and coordinate frames that robot_localization uses.

REP 103 defines the units as SI units⁵, as well as SI derived units such as radians as angles.

ENU (East North Up) is the preferred coordinate convention, where the standard relation to a body is as follows. x is forward, y is left, and z is up (see figure 2.7). Rotation is also centered on the x-axis (East), therefore the heading is 0 when heading Eastbound.

²http://docs.ros.org/en/noetic/api/robot_localization/html/index.html

³<https://www.ros.org/reps/rep-0103.html> [accessed online 07.04.2022]

⁴<https://www.ros.org/reps/rep-0105.html> [accessed online 08.04.2022]

⁵Bureau International des Poids et Mesures (<http://www.bipm.org/en/home/>)

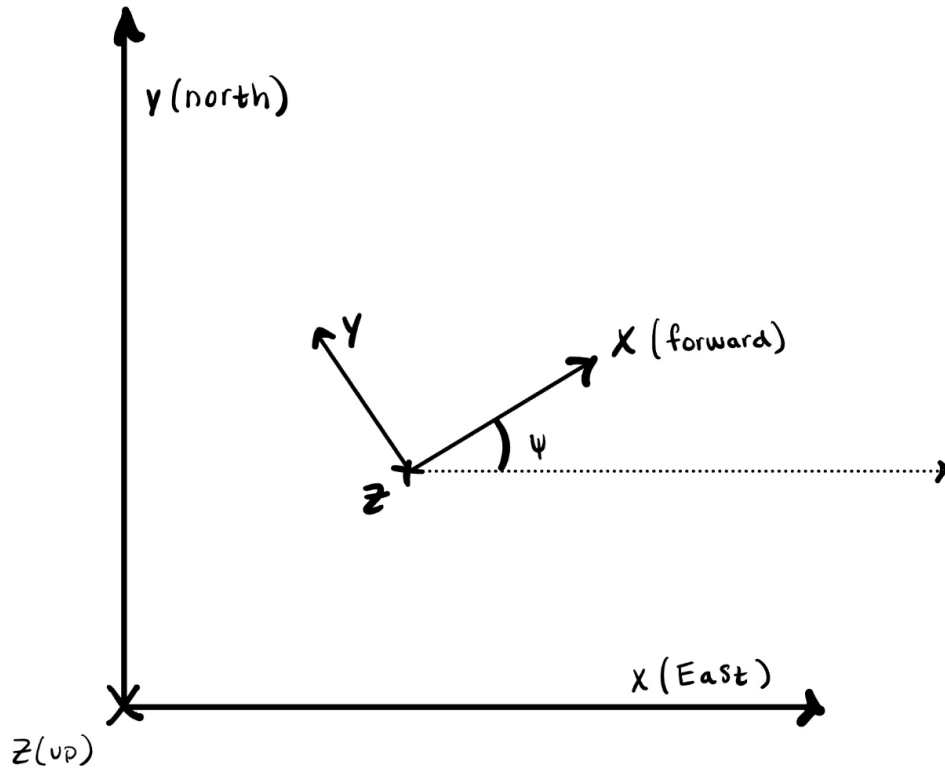


Figure 2.7: ENU convention. In this example, The outermost axes represent the map frame, and the innermost represent the odometry frame.

Rotations can be represented in several ways. REP 103 defines a prioritized order of representations respectively: Quaternions, rotation matrix, fixed axis roll, and euler angles.

REP 105 defines a hierarchy for coordinate frames for mobile platforms. The earth frame is the outermost frame, whilst the map is the second outermost and the odom is the last frame. Figure 2.8 shows an example of frame hierarchy without an earth frame.

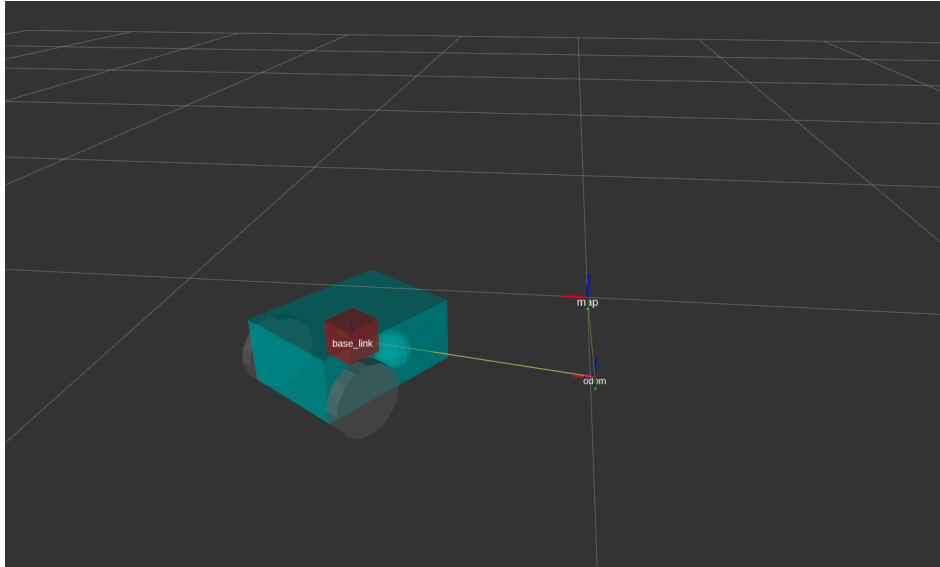


Figure 2.8: Map frame

2.9 Evaluation

To test performance of dead-reckoning in practice, it is often done by doing a trial over a closed-loop circuit and measuring the difference between the starting and end point. The results of such a trial can also be misleading by the fact that Initialization errors and biases have an impact on the performance, but don't impact the position of the end point (see figure 2.9).

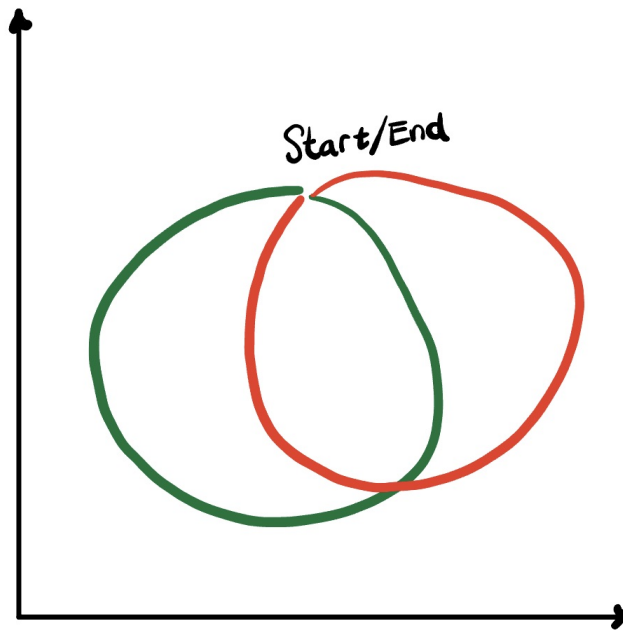


Figure 2.9: Example of dead-reckoning localization with bias in heading on a closed-loop circuit

Since this project will only simulate the localization, the true position will be easily available. Therefore the true and estimated position can be compared for each timestep.

2.9.1 Satisfactory performance

It is difficult to set fixed requirements for the performance of a localization. The requirements are different according to the scale at which the vehicle is operating, how fast the vehicle is traveling and how often absolute position updates arrive.

The vehicle model provided by Redrock AI is of a relatively slow moving industrial load handling vehicle, although Redrock AI may wish an algorithm that can be used on several types of vehicles. The results in chapter 4 will be representative for the vehicle provided, and things like wheel odometry will be affected by the characteristics of the wide and load carrying wheels that are on the vehicle, therefore there can be no guarantee for the applicability to other vehicles.

There are two scenarios where dead-reckoning is used. The first scenario is in between absolute position-fixing measurements with relatively low frequencies. A typical consumer grade GNSS receiver only updates once a second but more advanced GNSS receivers are often between

5 or 20hz⁶. Dead-reckoning between the measurements can enable faster navigation. The other scenario is when absolute position-fixing measurements are lost. This could be when a vehicle has entered a tunnel or building. It is difficult to know how long absolute measurements are lost, but an estimate could be around 5s to 1minutes before the vehicle has done enough driving to exit the building or past the obstacle. If a vehicle is driving inside buildings or behind obstacles for an extended amount of time, another position fixing method than GNSS should be used.

⁶<https://learn.sparkfun.com/tutorials/gps-basics/all>

Chapter 3

Materials and methods

In this chapter the methods for simulating the vehicle, implementing sensors, sensor fusing, filtering sensor values, testing, and evaluating the localization will be shown.

3.1 Software

This first section will go over the software that was used.

software	version	comments
Oracle vm virtualbox	version 6.1	https://www.virtualbox.org/
Ubuntu desktop	20.04.03 LTS	Linux based operating system
Robot Operating System 2 (ROS2)	Galactic Geochelone	https://docs.ros.org
Virtual Studio Code (VSC)	version 1.63.2	IDE
C/C++ extension VSC	v1.7.1	Language support for C/C++
CMake extension VSC	v0.0.17	Language support for CMake
Python extension VSC	v2021.12.1559732655	Language support for Python
Python	version 3.8	Programming language
pip3	version 20.0.2	package manager for Python
colcon-core	version 0.7.1	ROS2 package compiler
CMake	version 3.16.3	C++ dependancy manager
GNU Make	version 4.2.1	C++ compiler
git	version 2.25.1	github downloader

Table 3.1: Software versions

This section of the thesis will go over software installation, and why all the specific software's are being used. The majority of this section will be brief, as there are many other resources

available.

3.1.1 Operating System

Ubuntu version 20.04 was used in this project for its compatibility with the ROS2 architecture and its Tier 1 support¹.

3.1.2 Robot Operating Software 2

ROS2 was installed using Debian packages. This is the fastest way to install ROS2 using only a few command lines. When installing ROS2, you also have the option to choose between the base version or the desktop version. For this application the desktop version was used as it includes demos and graphical tools for troubleshooting.

setting up environment

To use the ROS2 commands in the terminal, the ROS2 setup.bash file has to be sourced. The following command will write the source command at the bottom of the ".bashrc" file, so that it is run each time a terminal is opened.

```
\$ echo "source /opt/ros/galactic/setup.bash" > ~/.bashrc
```

creating a workspace

Before we start downloading packages, we have to create a directory to put them in. It is common practice to create a directory named "/ros2_ws/src/" in the root directory:

```
\$ mkdir ~/ros2_ws/src/
```

robot_localization package

Next is the installation of the robot_localization package shown in the following command:

```
\$ sudo apt-get ros-galactic-robot-localization
```

¹<https://docs.ros.org/en/galactic/Installation/Ubuntu-Development-Setup.html>

colcon compilation

To compile the packages, and to build them, the colcon compiler is needed. The following line of code install colcon on the system.

```
\$ sudo apt install python3-colcon-common-extensions
```

3.1.3 Gazebo

Installation of Gazebo is quite simple as only one line is needed²

```
curl -sSL http://get.gazebosim.org | sh
```

3.2 Simulation

In this chapter, the method for simulating the ackerman vehicle in Gazebo(chapter 2.5) is shown. This entails, among other things, existing plugins, the plugins that were used, the ones that had to be created, and the vehicle models.

3.2.1 Gazebo Plugins

There exists a variety of different open-source plugins for ros2 integration with gazebo. These plugins can be used to publish sensor output and subscribe to motor inputs on ROS topics[].

Gazebo plugins can also be easily created to fit unmet requirements. Some of the following subchapters describe the plugins that were used in this project.

3.2.2 Ackerman Vehicle

For this thesis, an ackerman vehicle model from RedRock AI by agreement of a non-disclosure document was provided. Although pictures can be shown of the model as shown in figure 3.1, the code behind the model is restricted. The main concept of creating a vehicle is defining the links and rotational joints of the vehicle.

²https://classic.gazebosim.org/tutorials?tut=install_ubuntu#Defaultinstallation:one-liner



Figure 3.1: Gazebo truck model

3.2.3 Ackerman Steering

Having a model for the ackerman vehicle, the `gazebo_ros_ackermann_drive` plugin³ was applied to be able to drive the vehicle. To use this plugin, the ackerman vehicle joints were used as parameters, as well as update rate. Once the simulation has started, the steering can be applied by publishing to the `cmd_vel` ros topic, with the `geometry_msgs/msg/twist` datatype.

3.2.4 Ackerman Wheel Odometry

The `gazebo_ros_ackermann_drive` plugin (chapter 3.2.3) that was used in chapter 3.2.3 has a parameter that enables the plugin to publish odometry data. This could have been suitable, but the published odometry is just the actual position and velocities of the vehicle, which are not fetched from the wheels of the vehicle. This makes them unrealistically accurate. Therefore, in absence of any other plugins, a gazebo plugin had to be created. This plugin would need to use the rotation of the wheels to publish wheel odometry.

To create the gazebo plugins a c++ class called "AckermanOdom" was created. Gazebo plugin classes can have a "Load()" function, and a "OnUpdate()" function. In practice, these are both callback functions that are given to the Gazebo runtime. The Load() function is passed the

³https://github.com/ros-simulation/gazebo_ros_pkgs/blob/galactic/gazebo_plugins/include/gazebo_plugins/gazebo_ros_ackermann_drive.hpp

model pointer⁴ and sdf from gazebo runs on startup. The OnUpdate is a standard notation for the function that runs for every gazebo simulation iteration. This function needs no parameters as we can easily bind it to the gazebo runtime in the Load() function.

```
void Load(physics::ModelPtr _model, sdf::ElementPtr _sdf) override;
void OnUpdate();
```

When implementing the plugin, the parameters of the model can be defined in the .urdf gazebo file(appendix C). An example of how we define the arbitrary parameter "wheel_radius" is shown in the following snippet of code.

```
<gazebo>
  <plugin name="ackerman_odom_plugin" filename="libackerman_odom.so">
    <wheels_radius>0.2615475</wheels_radius>
  </plugin>
</gazebo>
```

These parameters are then passed on to the plugin through the _sdf parameter in the Load() function. To extract the parameters we can use the following code inside the Load() function.

```
float wheels_radius_ = _sdf->GetElement("wheel_radius")->Get<float>();
```

Defining the rest of the parameters(3.2) in the same manner(appendix D), allowed us to start implementing the kinematics of the ackerman vehicle.

Parameter	Value
Wheel radius	0.2615475
Wheel seperation	1.990400
rotation joint left	rot_left
rotation joint right	rot_right

Table 3.2: Ackerman odometry parameters

⁴http://osrf-distributions.s3.amazonaws.com/gazebo/api/2.2.1/classgazebo_1_1physics_1_1Model.html [accessed online 01.05.2022]

The heading from an IMU is used to calculate the linear velocities of wheel odometry. This is because the calculated heading from the odometry quickly diverges as any error in the angular velocity is integrated into the heading. Therefore, a subscription was created in the Ackerman odometry plugin to subscribe to the heading of the IMU. The IMU heading was given as a Quaternion, but the heading is needed as a Euler angle in order to calculate odometry. In order to convert the quaternion to a Euler angle, a callback function for the IMU heading was created(see following code snippet).

```
void AckermanOdom::Callback(sensor_msgs::msg::Imu imu){
    auto q = imu.orientation;
    double siny_cosp = 2 * (q.w * q.z + q.x * q.y);
    double cosy_cosp = 1 - 2 * (q.y * q.y + q.z * q.z);
    theta = std::atan2(siny_cosp, cosy_cosp);
}
```

An example of the resulting Euler angle while driving an arbitrary route is shown in figure 3.2. The figure shows that the Euler angle jumps from -180 to 180 degrees(This is one of the reasons that the Quaternion are more stable as mentioned in chapter 2.7).



Figure 3.2: Quaternion after being converted to Euler angle

To use this odometry with a Kalman filter, the covariances of the states also need to be defined. Only the velocities of the vehicle are being used in the kalman filter(chapter 3.3.1), the covariances can be defined as constants as the velocity uncertainty doesn't vary with time.

One way to find starting points for covariances is to find them empirically. For this application, they will be used as tuning parameters later. The following covariance matrix is an example of how it is defined in the Gazebo plugin. The order of states is $\dot{X}, \dot{Y}, \dot{Z}, \dot{\phi}, \dot{\theta}, \dot{\psi}$ ⁵.

```
odom_.twist.covariance = {0.06, 0, 0, 0, 0, 0,
                          0, 0.07, 0, 0, 0, 0,
                          0, 0, 0.02, 0, 0, 0,
                          0, 0, 0, 0.011, 0, 0,
                          0, 0, 0, 0, 0.012, 0,
                          0, 0, 0, 0, 0, 0.05};
```

3.2.5 Inertial Measurement Unit

*GazeboRosImuSensor*⁶ is used for simulating a IMU sensor. This plugin permits changes in update rate, and gaussian noise. To apply this plugin, it has to be attached to a link⁷ as shown in Appendix C. The plugin will publish to a topic with the *sensor_msgs/Imu*⁸ datatype. This datatype contains orientation, linear acceleration, and angular acceleration respectively as well as their covariance matrices.

3.2.6 Ground Truth

To evaluate a localization, it's needed to know what the true position and velocities of the vehicle are. The *P3D(3D Position Interface for Ground Truth)*⁹ plugin is used to do this. This plugin

⁵http://docs.ros.org/en/noetic/api/geometry_msgs/html/msg/TwistWithCovariance.html [accessed online 02.05.2022]

⁶[https://classic.gazebosim.org/tutorials?tut=ros_gzplugins#IMU\(GazeboRosImu\)](https://classic.gazebosim.org/tutorials?tut=ros_gzplugins#IMU(GazeboRosImu))

⁷links are the building blocks of gazebo models

⁸http://docs.ros.org/en/diamondback/api/sensor_msgs/html/msg/Imu.html

⁹[https://classic.gazebosim.org/tutorials?tut=ros_gzplugins#P3D%20\(3D%20Position%20Interface%20for%20Ground%20Truth\)](https://classic.gazebosim.org/tutorials?tut=ros_gzplugins#P3D%20(3D%20Position%20Interface%20for%20Ground%20Truth))

publishes the true odometry of a vehicle with the *nav_msgs/Odometry*¹⁰ datatype.

3.2.7 Architecture

Figure 3.3 shows the simulation architecture.

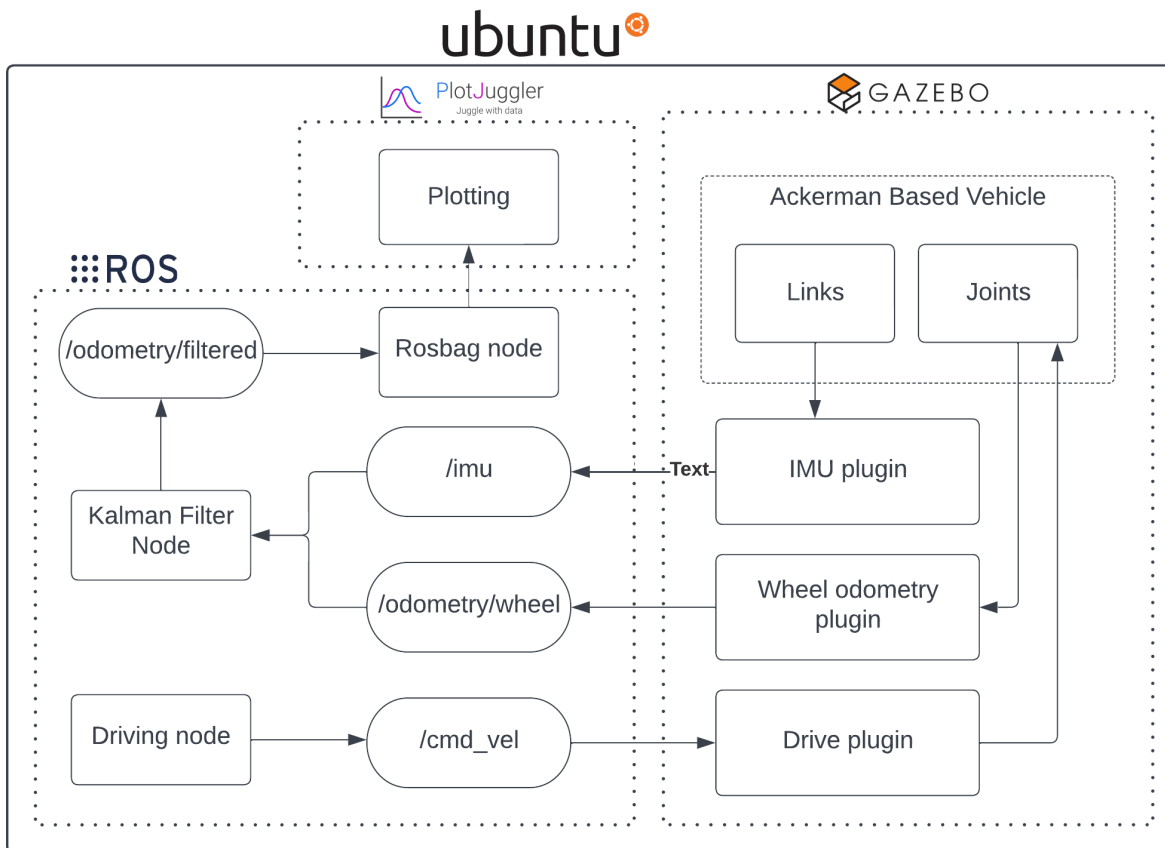


Figure 3.3: Simulation Architecture

In some cases, the same driving pattern was used for different Kalman filters, and it was desirable to apply the same data to each filter. Therefore, all the topics were recorded to a Rosbag (figure 3.4) and then played back to different kalman filters (figure 3.5).

¹⁰http://docs.ros.org/en/noetic/api/nav_msgs/html/msg/Odometry.html [accessed online 05.05.2022]

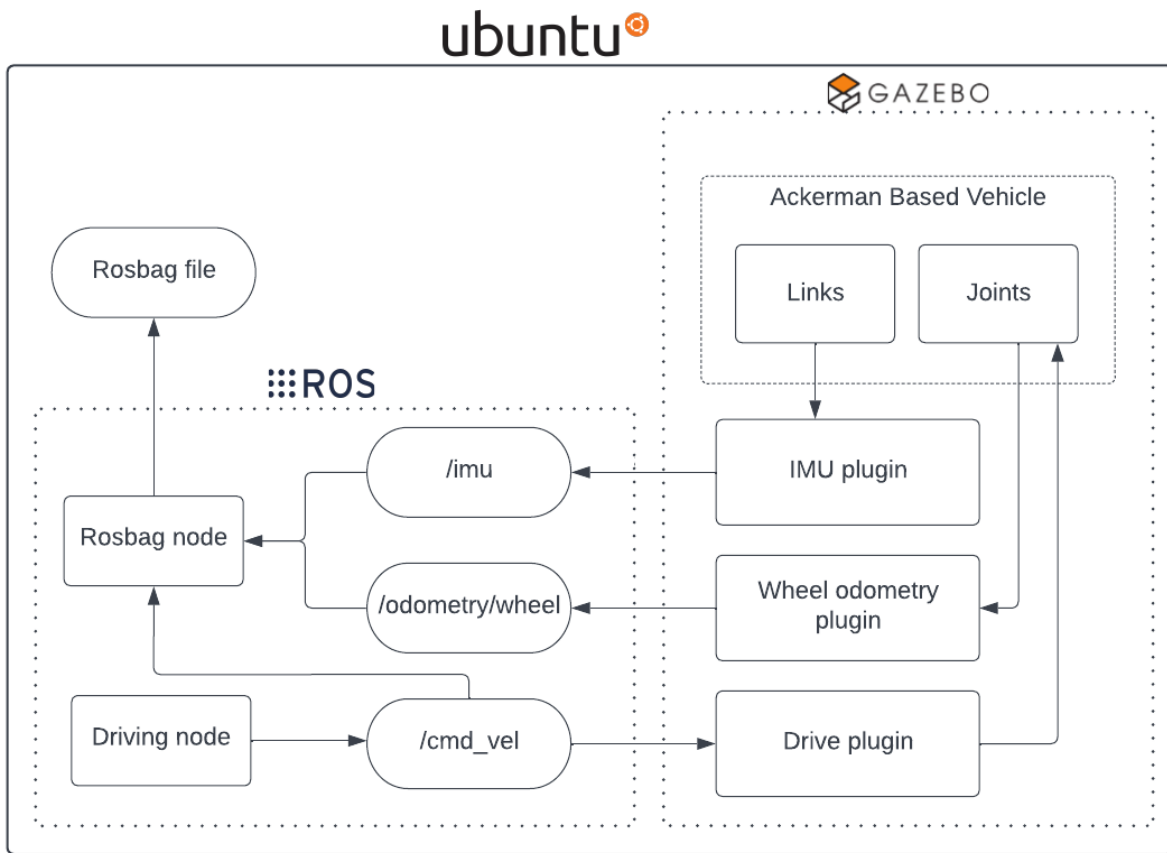


Figure 3.4: Recording gazebo topics

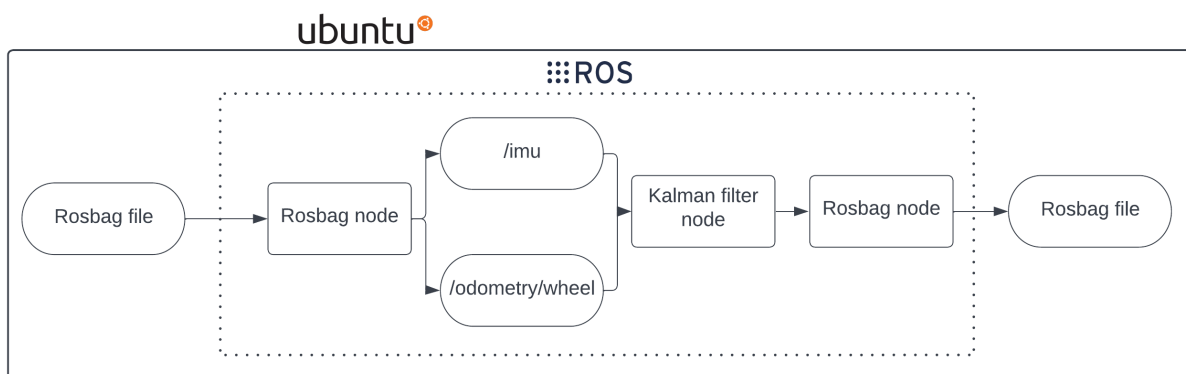


Figure 3.5: Playback and recording of rosbag data with arbitrary Kalman filter.

3.3 State Estimation

This chapter mainly covers the use of the `robot_localization` package, and how different filters and parameters are configured.

3.3.1 Sensor fusion

This sub-chapter covers the configuration of different combinations of sensors for state estimation. To apply state estimation, the `robot_localization` package is used. This package enables the use of EKF and UKF as well as an arbitrary combination of sensors.

Wheel odometry and IMU

The first fusion that is applied is Wheel odometry and IMU. Starting with the wheel odometry, we can conclude that the information we need to extract is the linear velocities, and the horizontal angular velocity as the angular velocity, together with the imu can create a better joint value. We therefore write the wheel odometry parameters as follows¹¹:

```
odom0_config: [false, false, false,
               false, false, false,
               true,  true,  false,
               false, false, true,
               false, false, false]
```

In this case, the position is not important because it is just the integrated velocities we are already integrating. The following shows the enables states from the IMU sensor.

```
imu0_config: [false, false, false,
              true,  true,  true,
              false, false, false,
              true,  true,  true,
              true,  false, false]
```

¹¹the order of states is shown in figure 2.15

Note that only the linear acceleration in the X direction is being used. This is because the accelerometer sensor outputs acceleration in respect to the rigid vehicle body as a reference frame. Therefore only the X direction is being used as this is the axis which points in the direction of travel. Also, roll, pitch and yaw are all being used as they are all needed to negate gravitational acceleration in the linear acceleration term, like the example shown in figure 3.6. The resulting system is shown in figure 3.6.

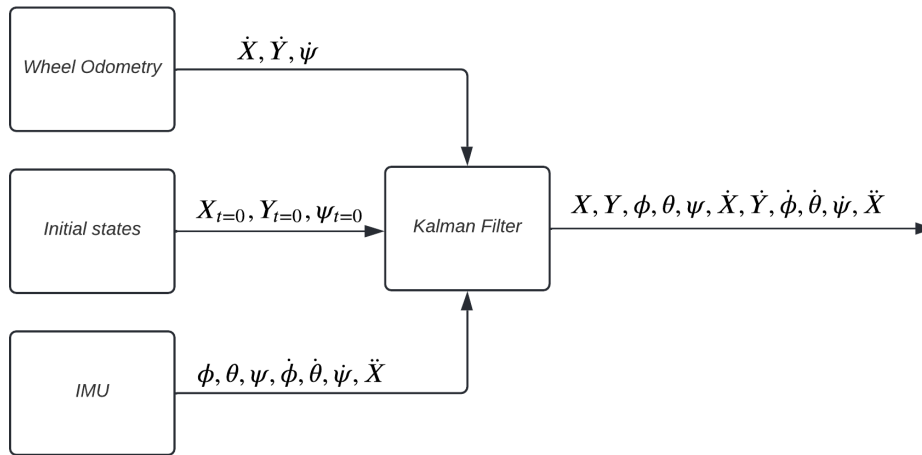


Figure 3.6: Kalman Filter input and output states flow chart

Measurements for the innovation stage (equation 2.4) are shown in equation 3.1.

$$z_k = \left[\phi \quad \theta \quad \psi \quad \dot{X} \quad \dot{Y} \quad \dot{\phi} \quad \dot{\theta} \quad \dot{\psi} \quad \ddot{X}_{body_frame} \right]^T \quad (3.1)$$

Wheel odometry and directional gyroscope (heading)

This fusion setup is shown in figure 3.7.

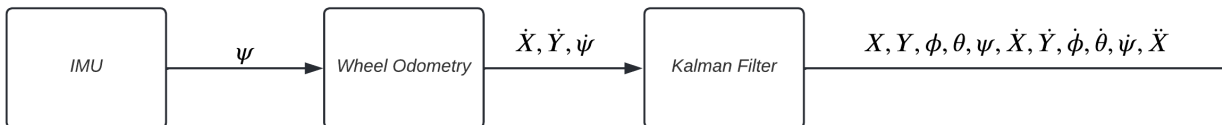


Figure 3.7: Sensor fusion with directional gyroscope heading fed into wheel odometry algorithm

For this sensor fusion model, only the wheel odometry states are being fed into the Kalman filter.

```
odom0_config: [false, false, false,
               false, false, false,
               true,  true,  false,
               false, false, true,
               false, false, false]
```

To implement the heading into the wheel odometry plugin, simple logic was created to replace the heading that was estimated by the plugin itself(chapter 3.2.4).

IMU(Directional gyroscope and Accelerometers)

This sensor fusion model only includes the measurements from the IMU module. Therefore, the following states are used: heading and angular velocity from a directional gyroscope, and linear acceleration from an accelerometer(See figure 3.8).

```
imu0_config: [false, false, false,
              true,  true,  true,
              false, false, false,
              true,  true,  true,
              true,  false, false]
```

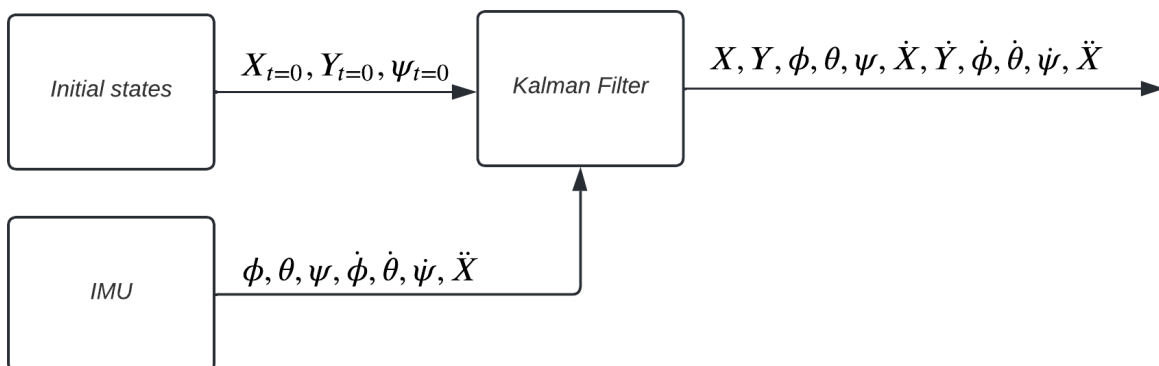


Figure 3.8: IMU application flowchart

Wheel Odometry

Lastly, the lone states of the wheel odometry module are used. Figure 3.9 shows the resulting data flow, and then the enabled states are shown.

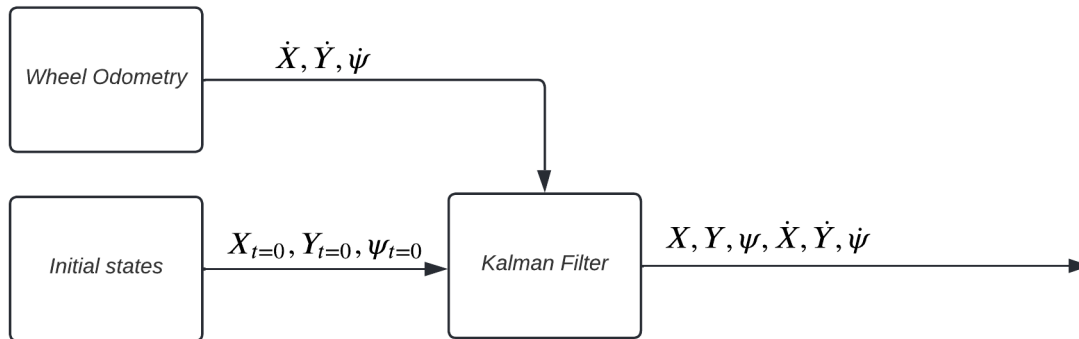


Figure 3.9: Kalman filtering of wheel odometry

```
odom0_config: [false, false, false,
               false, false, false,
               true, true, false,
               false, false, true,
               false, false, false]
```

3.3.2 Tuning

The process noise covariance matrix, commonly denoted Q , represents the uncertainty of the estimated states in a system. This parameter can also be interpreted as the uncertainties that aren't modeled in the state transfer matrix. This parameter is important for tuning kalman filters, because they decide how much to weight measurements, and how much to weight state estimates.

3.4 Testing

Rosbag, a tool used to record ROS topics, was used to record the odometry of the vehicle. Figure 3.10 shows the command used to record all the topics(see figure 3.3).

```

simen@simen-ubuntu: ~/rrai_project/bags/misc
simen@simen-ubuntu:~/rrai_project/bags/misc$ ros2 bag record -a
[INFO] [1650706569.627752457] [rosbag2_storage]: Opened database 'rosbag2_2022_04_23-11_36_09
/rosbag2_2022_04_23-11_36_09_0.db3' for READ_WRITE.
[INFO] [1650706569.627868703] [rosbag2_recorder]: Listening for topics...
[INFO] [1650706569.628679992] [rosbag2_recorder]: Subscribed to topic '/set_pose'
[INFO] [1650706569.629480568] [rosbag2_recorder]: Subscribed to topic '/odometry/wheel'
[INFO] [1650706569.630207962] [rosbag2_recorder]: Subscribed to topic '/tf_static'
[INFO] [1650706569.630850335] [rosbag2_recorder]: Subscribed to topic '/simulation/p3d'
[INFO] [1650706569.631480161] [rosbag2_recorder]: Subscribed to topic '/odometry/filtered'
[INFO] [1650706569.632340292] [rosbag2_recorder]: Subscribed to topic '/rosout'
[INFO] [1650706569.633230415] [rosbag2_recorder]: Subscribed to topic '/imu'
[INFO] [1650706569.633855555] [rosbag2_recorder]: Subscribed to topic '/simulation/cmd_vel'
[INFO] [1650706569.634601738] [rosbag2_recorder]: Subscribed to topic '/parameter_events'
[INFO] [1650706569.635343404] [rosbag2_recorder]: Subscribed to topic '/diagnostics'
[INFO] [1650706569.635919922] [rosbag2_recorder]: Subscribed to topic '/tf'
[INFO] [1650706569.636601334] [rosbag2_recorder]: Subscribed to topic '/clock'
    
```

Figure 3.10: Using ros2 bag to record all topics

After a recording was started, the vehicle was driven arbitrarily. A

"rqt_bag" is used to playback the recorded data simultaneously as the filter is started, and a new recording is begun. Figure 3.12 shows the resulting bag data when the bag data in 3.11 is filtered with a kalman filter.

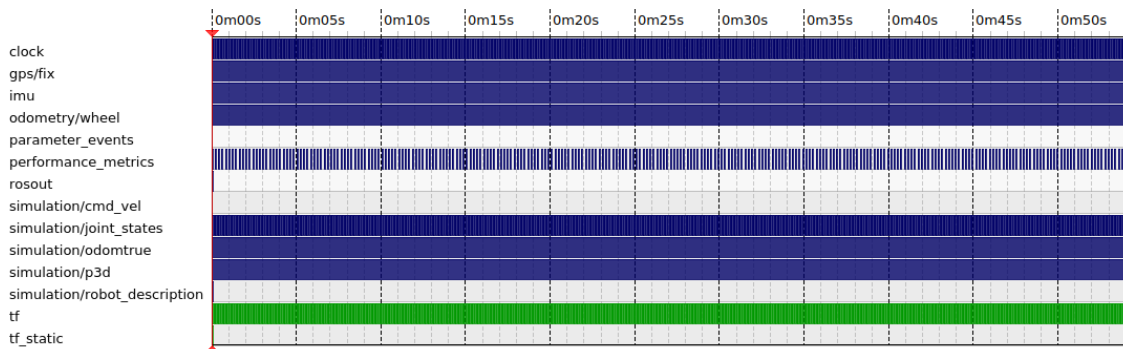


Figure 3.11: Bag data

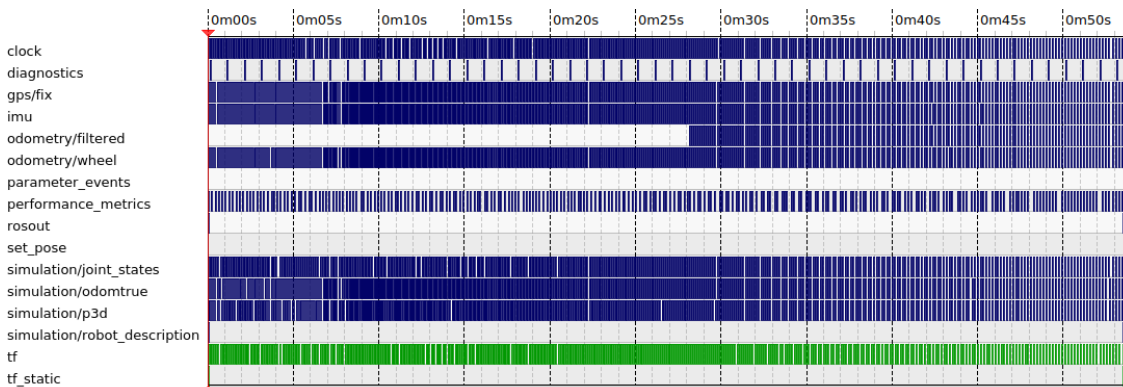


Figure 3.12: Bag data with filtered odometry(with "/odometry/filtered")

With this method, the same sensor readings can be used for different filters. This makes them easier to compare, as it becomes an apples to apples comparison.

3.5 Evaluation

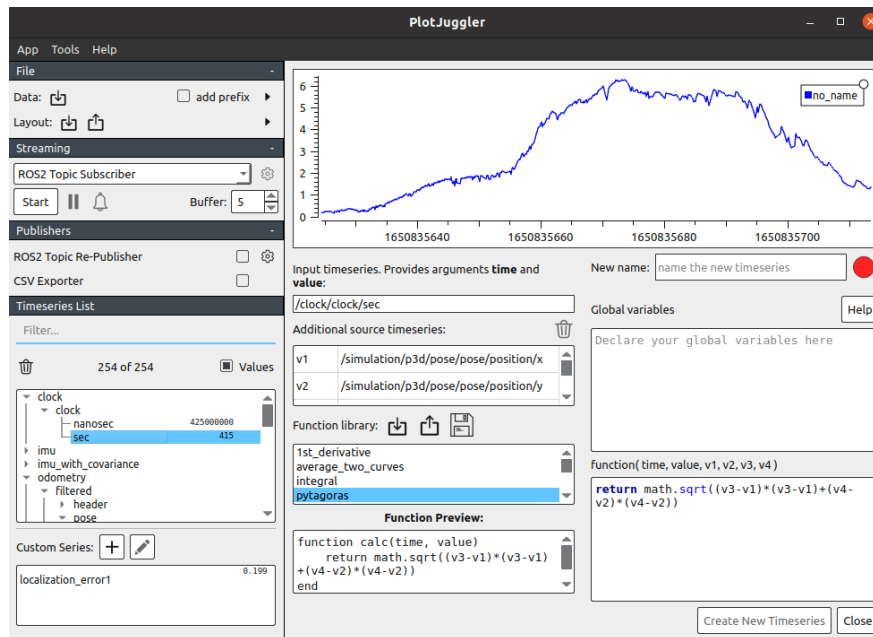


Figure 3.13: Custom timeseries in plotjuggler

Plotjuggler was used, as it was seemingly one of the few plotting softwares that could plot ros2 bag data natively. Its real-time plotting capabilities are also helpful when testing and diagnosing problems. Its inability to modify axis labels, and titles, among other things are some of the reasons that it is was difficult to use. In the results section of the thesis, the axis labels had to be added manually with a graphical tool. Custom functions can also be created and used to edit timeseries in plotjuggler (figure 3.13). Functions can be written in the programming language lua¹². A function implementing the pythagorean theorem (figure 3.14) was created to plot the error in position.

¹²<https://www.lua.org/about.html> [accessed online 06.05.2022]


```
function calc(time, value)
  return math.sqrt((v1-v3)*(v1-v3)+(v2-v4)*(v2-v4))
end
```

Figure 3.14: Function for calculating the difference in position

Chapter 4

Result

This chapter contains the results from this project. Firstly, the raw output of the wheel odometry and the IMU measurements(chapter 4.1 and 4.2) are shown for an arbitrary driving pattern. This helps get a grasp of the performance that can be expected. Then they are both fed into a state estimation filter in chapter 4.3 and the resulting localization is shown. Furthermore, some the localization results from several combinations are shown, and then other improvements that were attempted.

4.1 Wheel Odometry

The states that are calculated in the ackerman wheel odometry plugin are \dot{X} , \dot{Y} , $\dot{\psi}$. Comparisons between the calculated wheel odometry and the true odometry is shown in figure 4.1, 4.2, and 4.3

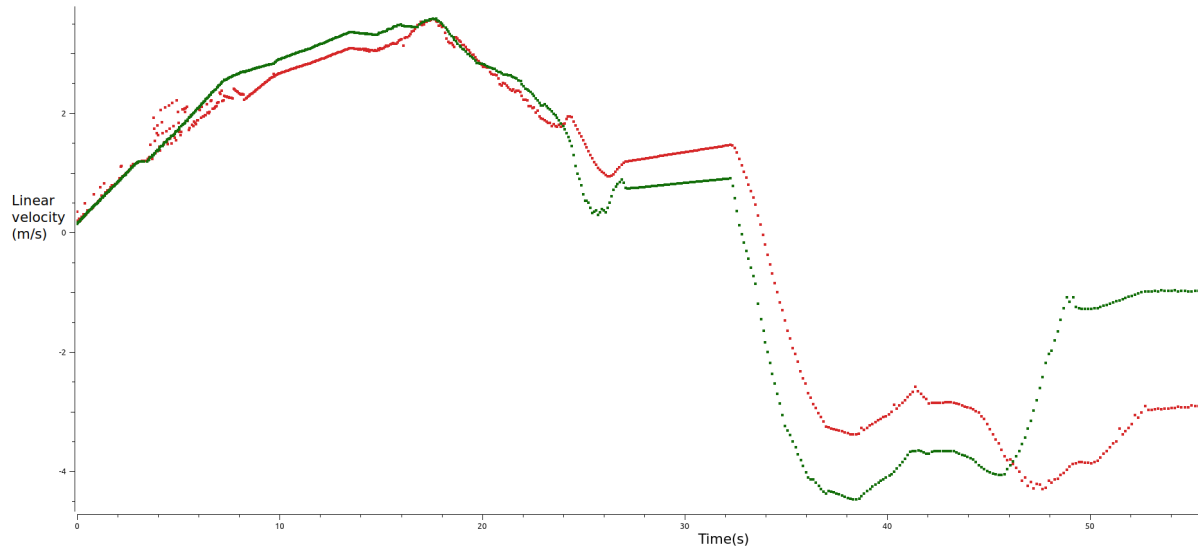


Figure 4.1: Linear velocity in x(north) direction. The true velocity is shown in green, and the estimated ackerman wheel odometry velocity is shown in red.

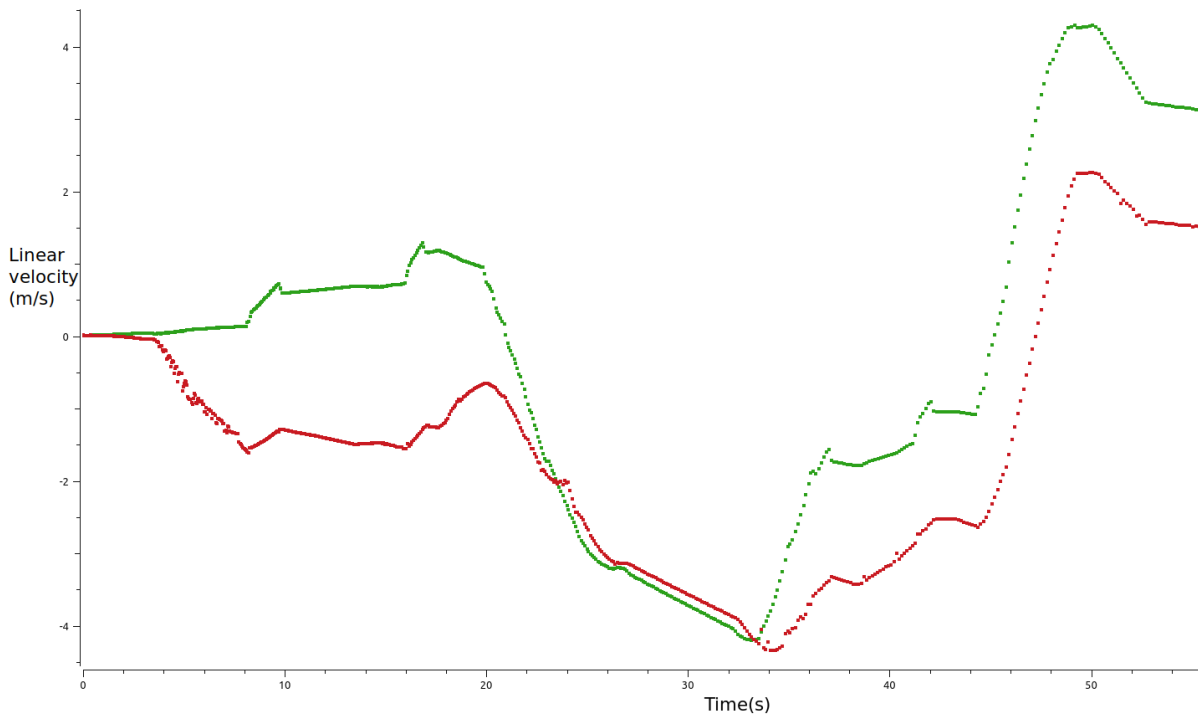


Figure 4.2: Linear velocity in y(east) direction. The true velocity is shown in green, and the estimated ackerman wheel odometry velocity is shown in red.

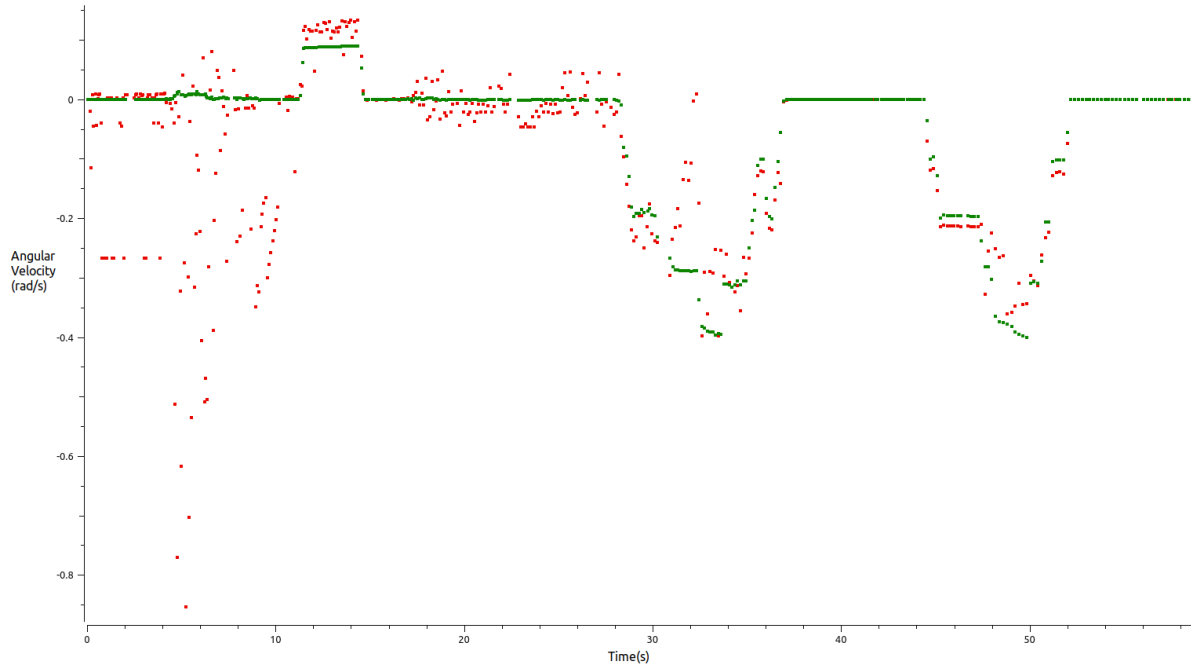


Figure 4.3: Angular velocity($\dot{\psi}$). The true angular velocity is shown in green, and the estimated ackerman wheel odometry angular velocity is shown in red.

4.2 IMU

The IMU measurements that are used are ϕ , θ , ψ , $\dot{\psi}$ and \ddot{X} . Their comparisons to their true counterparts are shown in the following figures respectively.

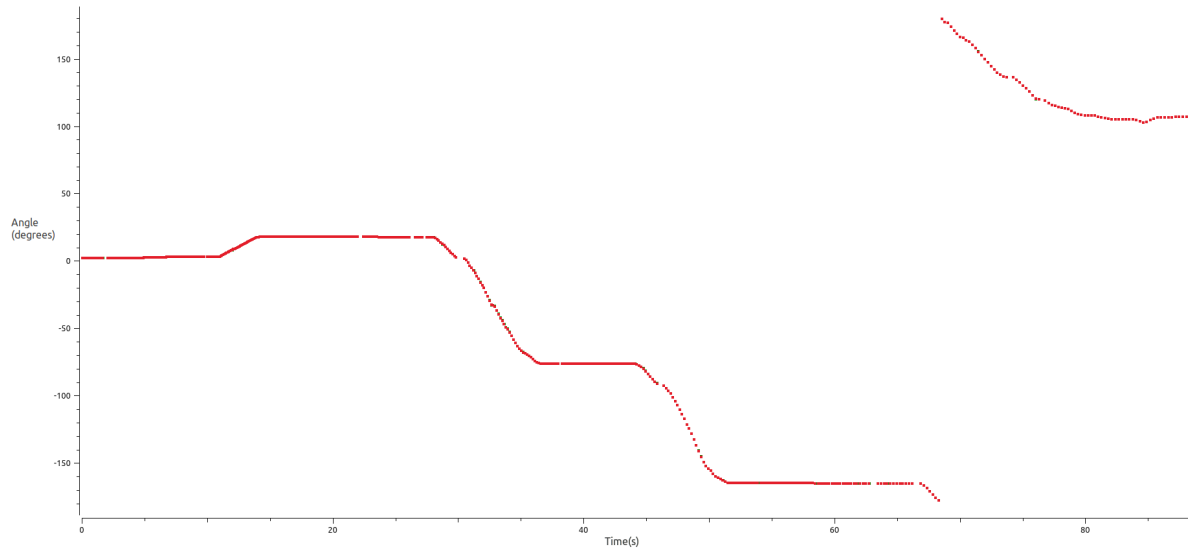


Figure 4.4: Yaw(ψ). True yaw is in green, and IMU yaw is in red. Gaussian noise of 1

Figure 4.4 shows that the IMU heading is very accurate, and very often within ± 1 degree of rotation. This is why the true value for heading is not visible.

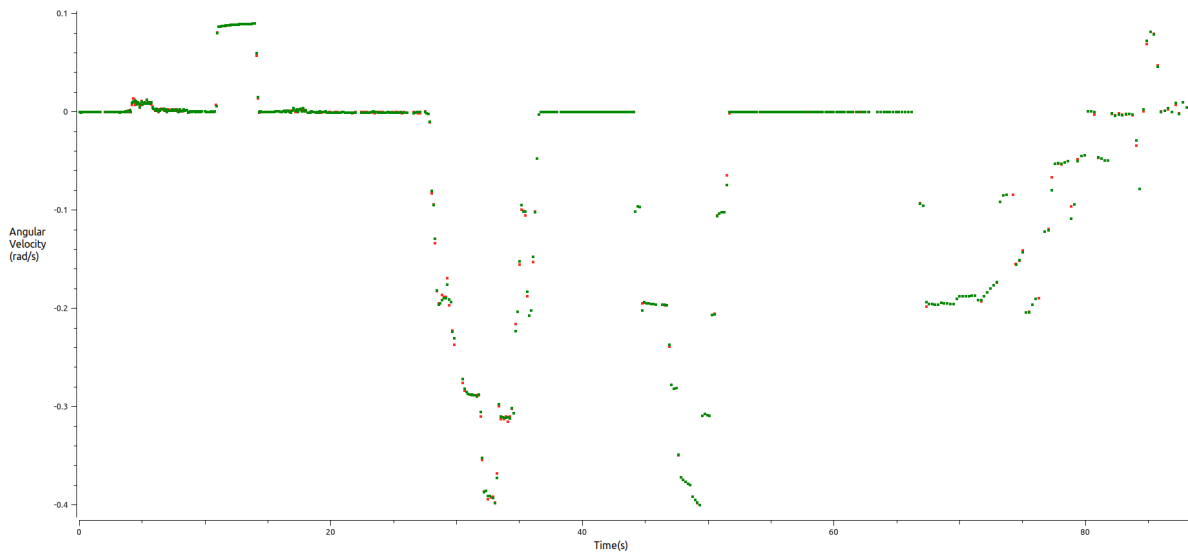


Figure 4.5: Angular velocity($\dot{\psi}$). True angular velocity is green, and IMU angular velocity is in red. Gaussian noise of 1

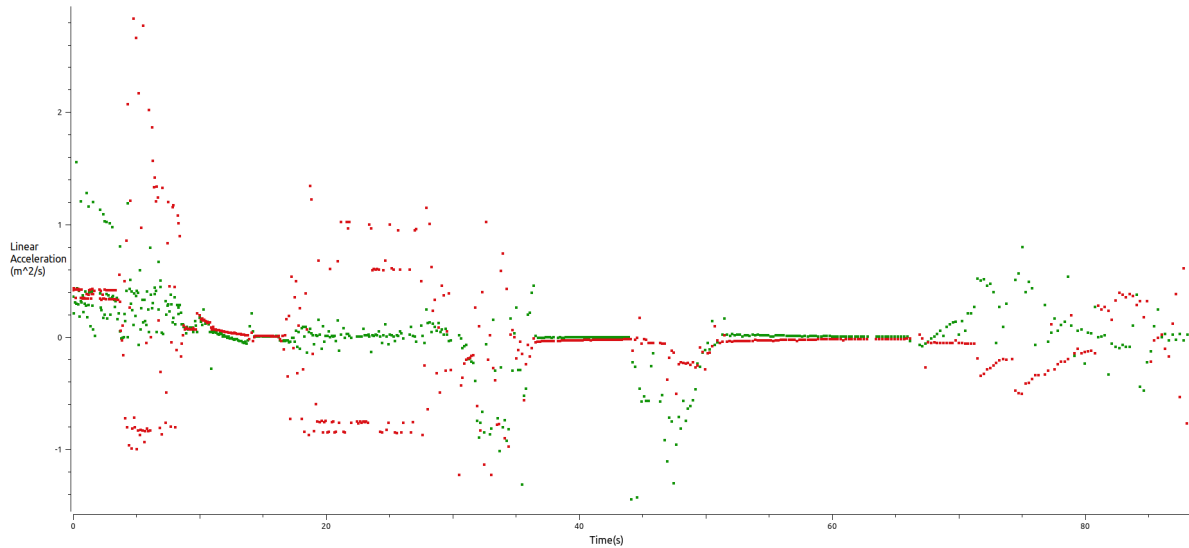


Figure 4.6: Linear acceleration. True linear acceleration is in green, and IMU linear acceleration is in red. Gaussian noise of 1

4.3 State estimation

This subchapter covers the results of Kalman filters applied to the Ackerman Odometry and IMU states in different combinations, and with different tuning parameters.

4.3.1 Wheel Odometry

This subchapter contains the result of running an extended kalman filter with only the states from the wheel odometry(\dot{X} , \dot{Y} , ψ).

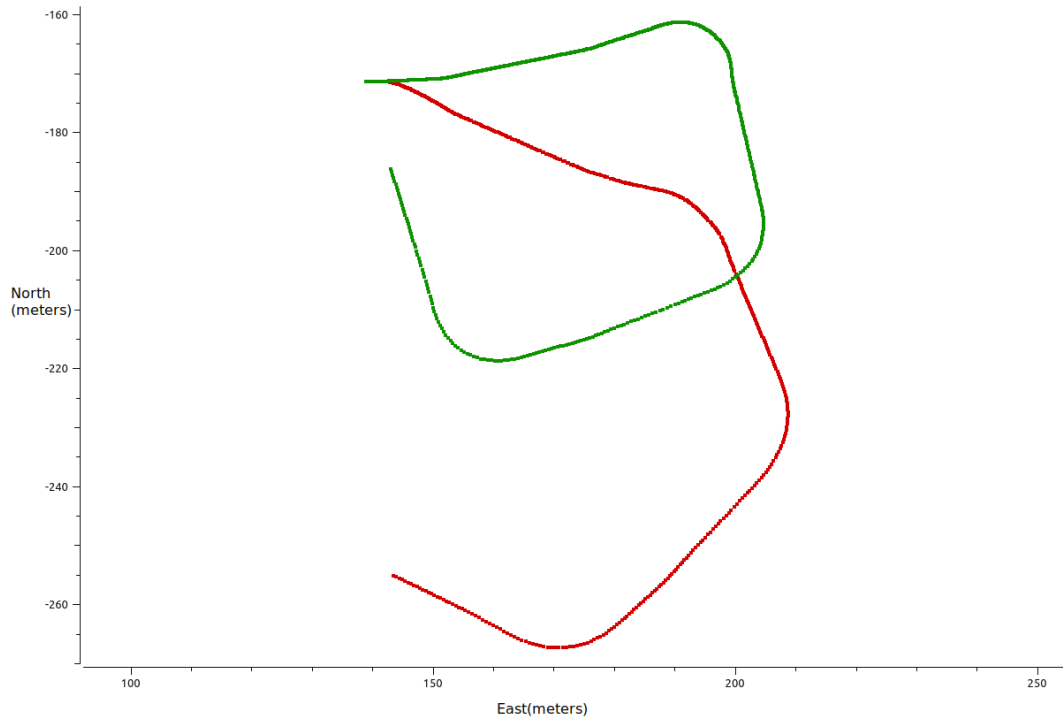


Figure 4.7: Wheel odometry only state estimation.

Figure 4.7 shows the filtered position and true position, while figure 4.1 and 4.2 shows the corresponding velocities.

4.3.2 IMU

Figure 4.8 shows the linear velocity in the x direction of a IMU only application. The estimated velocity is very accurate to begin with, with only a 0.025m/s drift after 5 seconds. After this point, the estimated velocity diverges. Figure 4.9 shows a very similar situation for the linear velocity in the y direction.

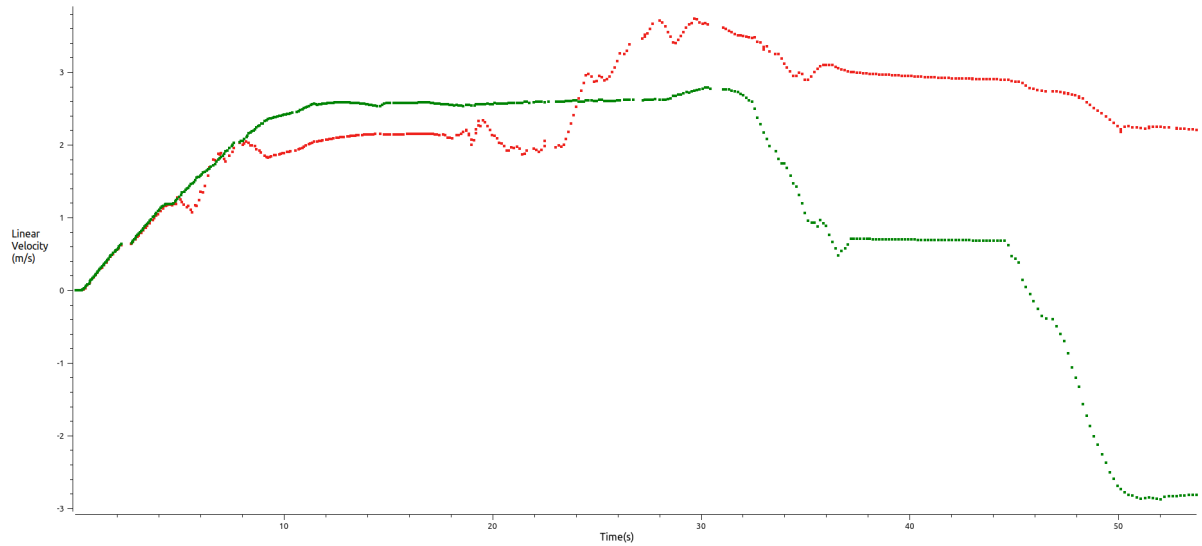


Figure 4.8: Linear velocity in x direction with only IMU states as input to kalman filter. Estimated velocity is red, true velocity is green.

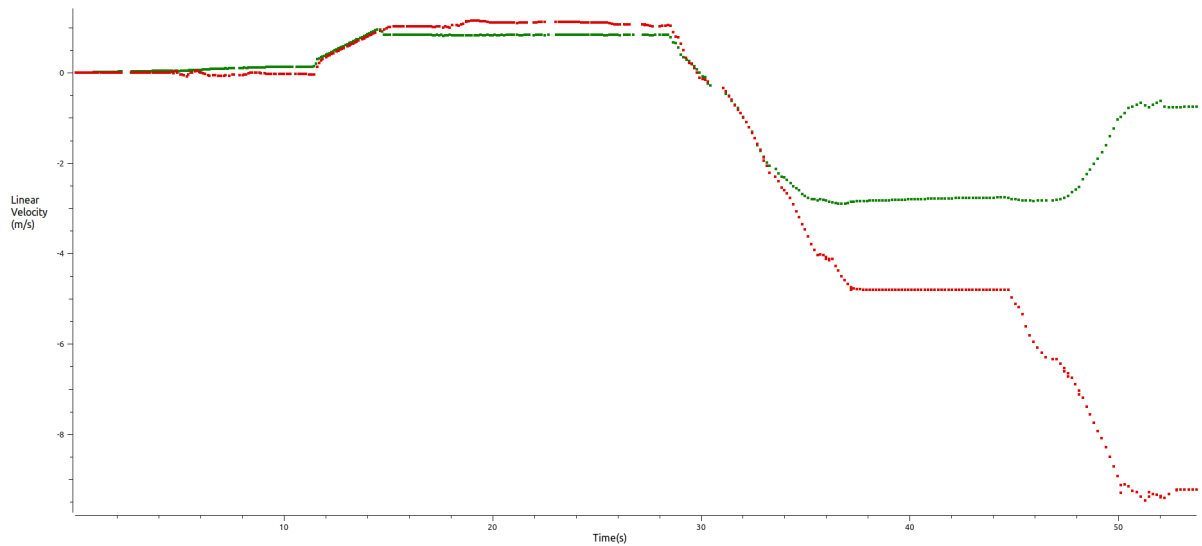


Figure 4.9: Linear velocity in y direction with only IMU states as input to kalman filter. Estimated velocity is red, true velocity is green.

The resulting localization is shown in figure 4.10. Localization drifts 0.1563 meters after 5 seconds, 13.4m after 30 seconds, and 95 meters after 60 seconds.

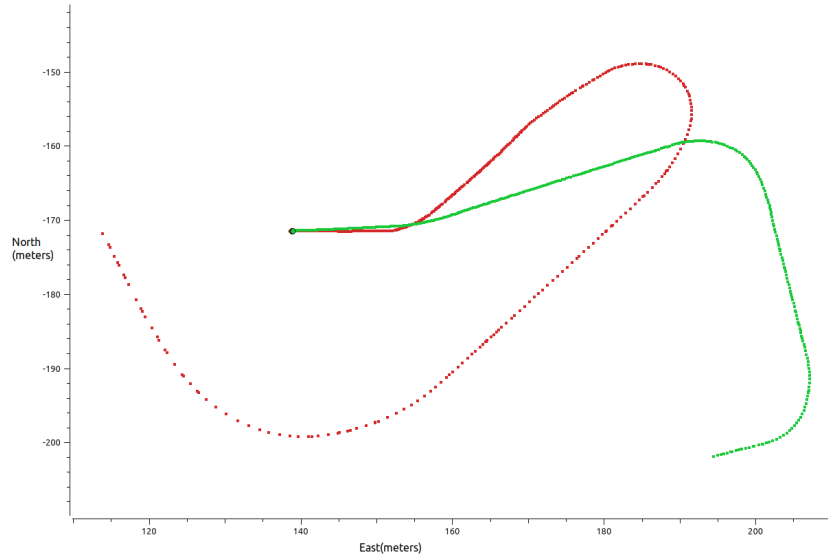


Figure 4.10: Localization with IMU states. Estimated position is red, true position is green.

4.3.3 Sensor Fusion

wheel odometry and directional gyroscope

For this first localization example, the wheel odometry fetched the heading from the IMU plugin. In other words, \dot{X} , \dot{Y} , ψ were the wheel odometry inputs to the kalman filter, while ψ was the IMUs input to the odometry plugin(see figure 3.7). The resulting localization is shown in figure 4.13. The velocities are shown in figure 4.11 and 4.12

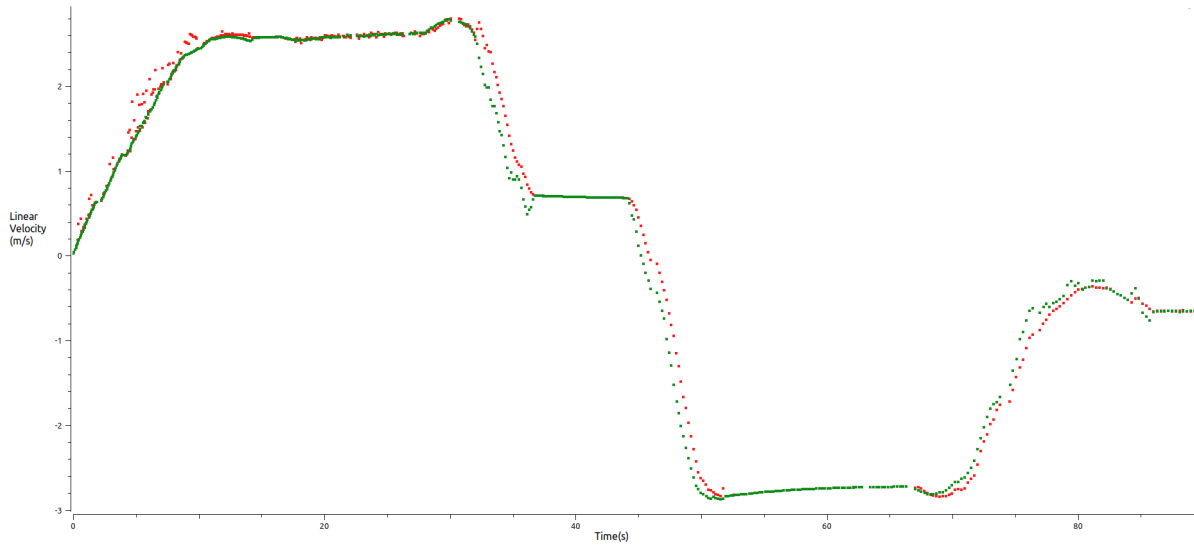


Figure 4.11: Wheel Odometry fused with IMU heading. Estimated velocity is red, true velocity is green.

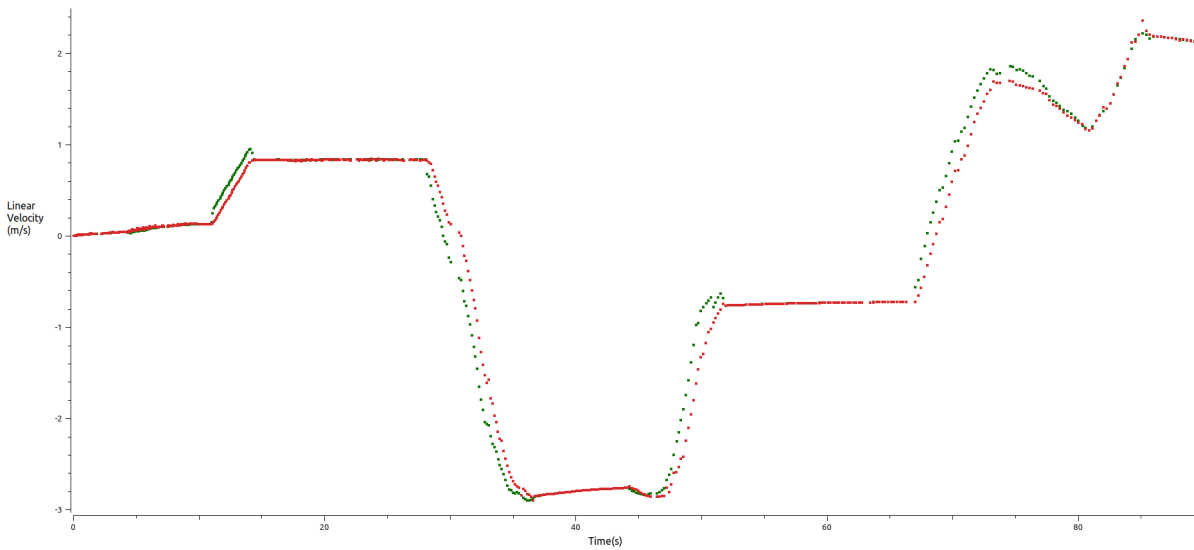


Figure 4.12: Wheel Odometry fused with IMU heading. Estimated velocity is red, true velocity is green.

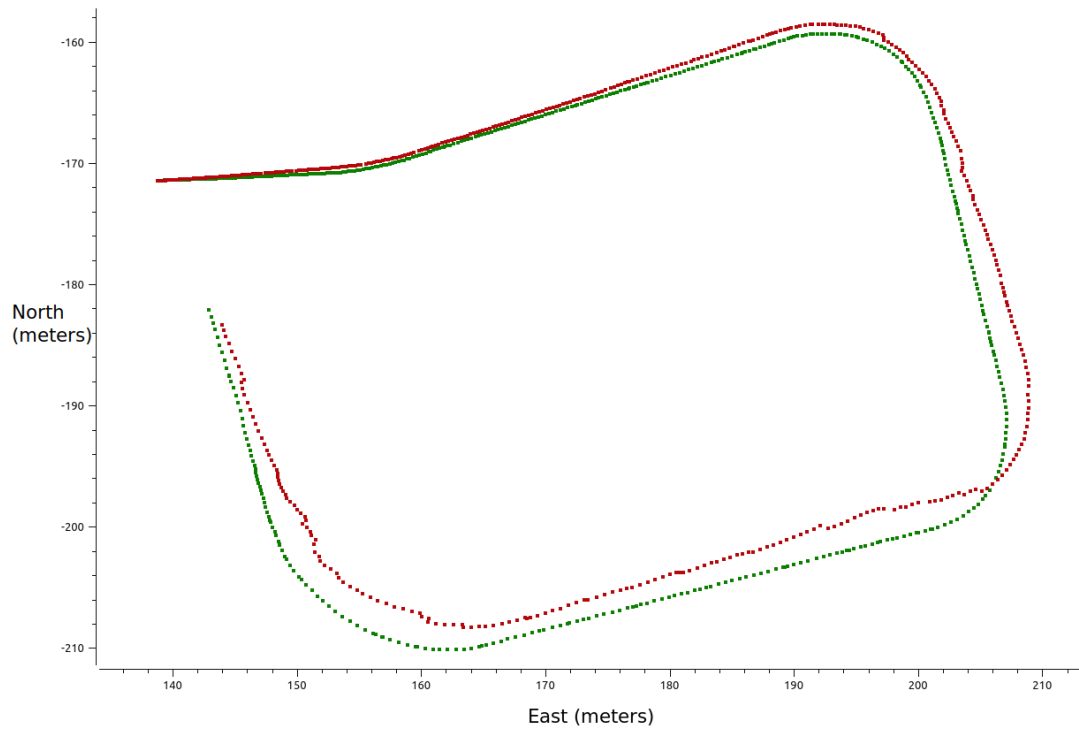


Figure 4.13: Wheel Odometry fused with IMU heading

wheel odometry, gyroscope and accelerometer

EKF and UKF

A EKF and a UKF was a applied to the same driving scenario. Figure 4.14 compares their estimated paths.

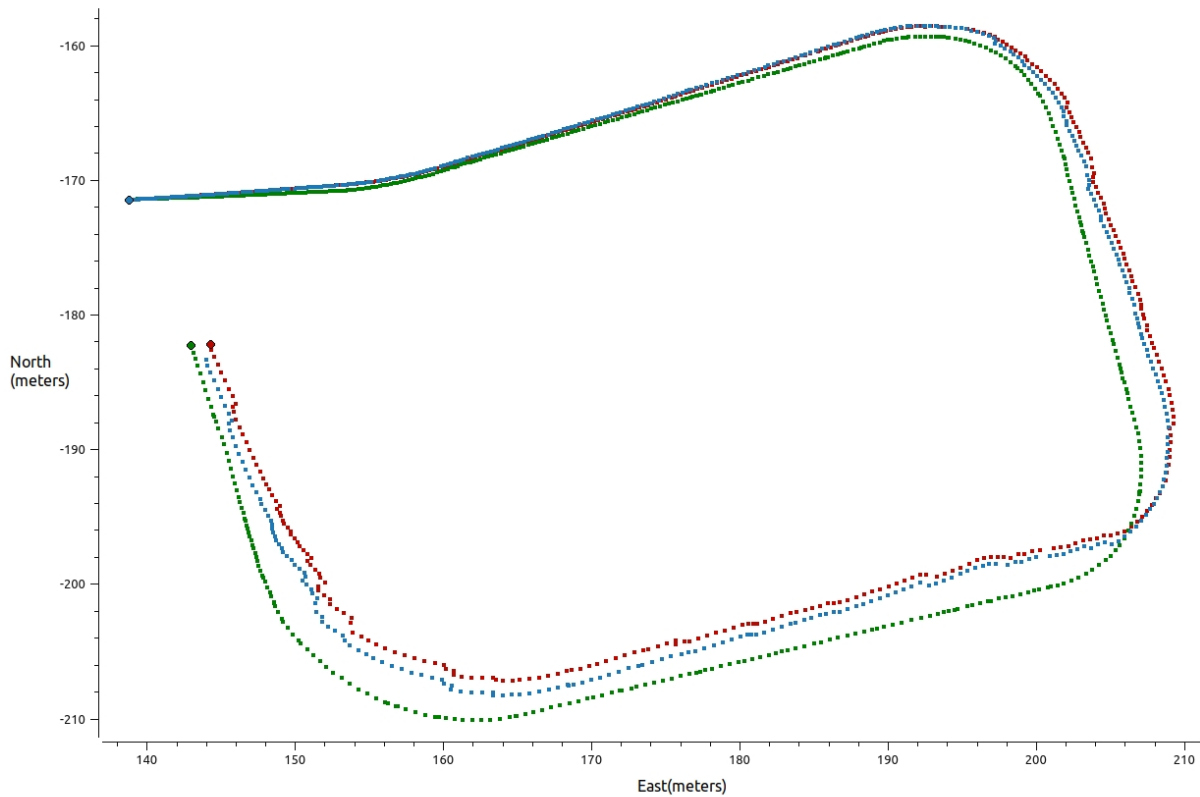


Figure 4.14: Driving scenario 3, with true position in green, ekf estimated position in red, and ukf estimated position in blue

4.3.4 Improvements

Tweaking Physical Parameters

When linear velocities are correct, but angular velocities are not, tweaking the radii of the wheels is an easy way to compensate for slip. Also, if odometry is good when driving in a straight line, but not as good when turning, adjusting the wheel base length is also an easy way to increase or decrease the angular velocities.

Figure 4.7 shows that state estimation with only wheel odometry results in understeering. To compensate for this, decreasing the wheelbase slightly was done to compensate for any slippage that was occurring. The resulting localization is shown in figure 4.15.

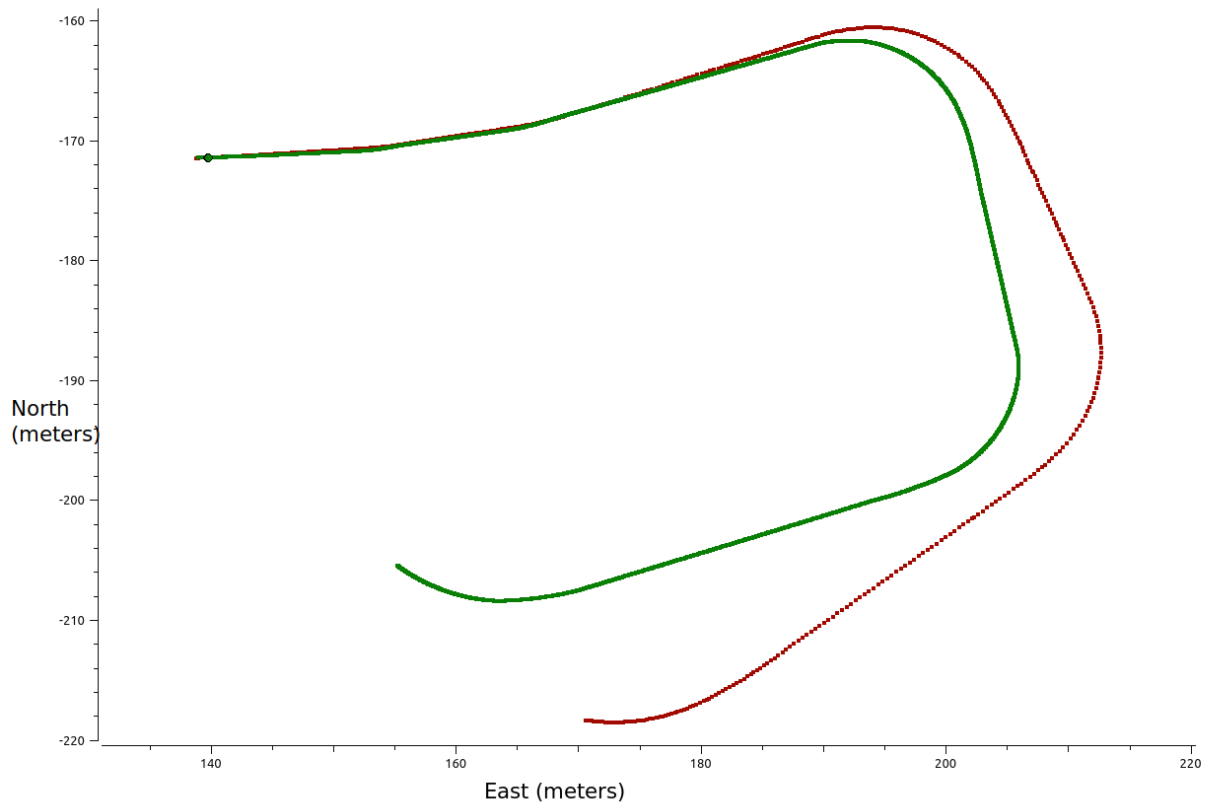


Figure 4.15: Wheel odometry only state estimation after adjusting wheelbase. Estimated position is red, true position is green.

further decreasing the wheelbase length resulted in localization shown in figure 4.16

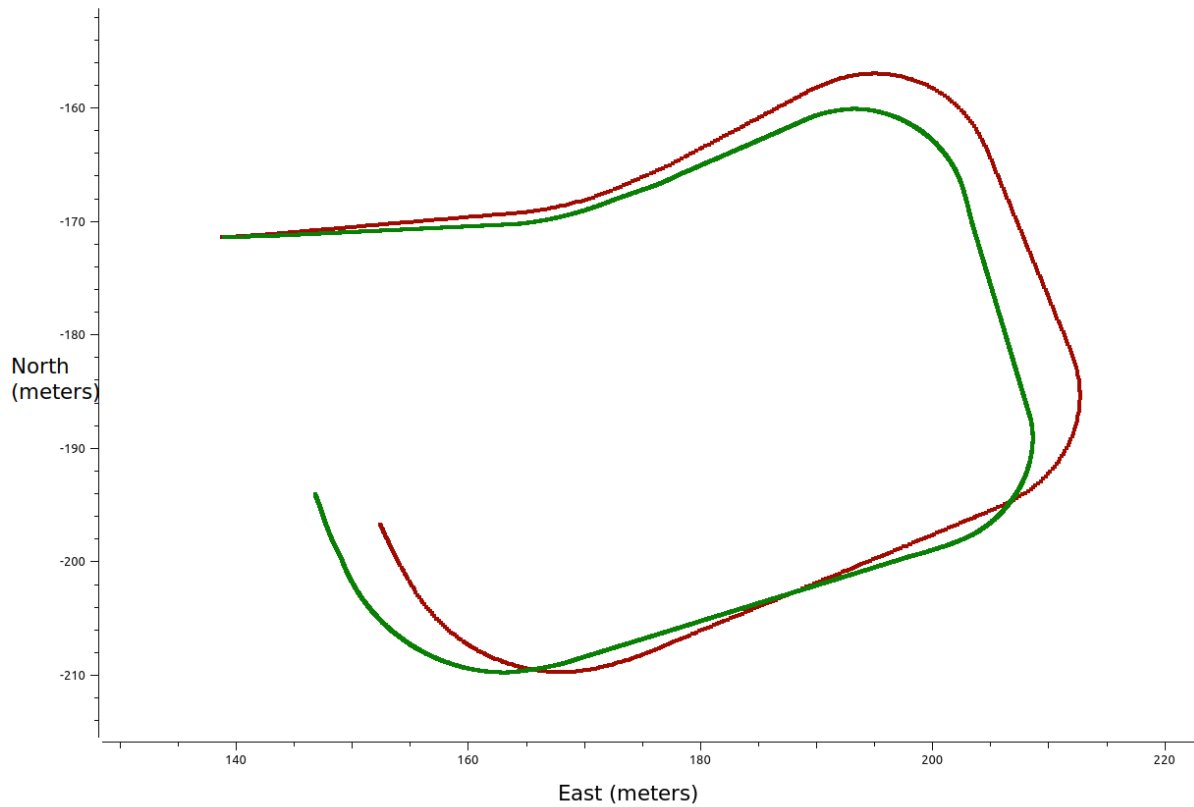


Figure 4.16: Wheel odometry only state estimation after adjusting wheelbase further. Estimated position is red, true position is green.

Process noise covariance

Figure 4.13 shows that the estimated position has what is suspected to be a bias in its heading. It is shown in figure 4.4 that the heading is very accurate, and therefore should not be the cause of the offset. Further investigation concludes that the source of error is a delay that is caused by the kalman filter not weighting wheel odometry measurements enough. This is shown in figure 4.17.

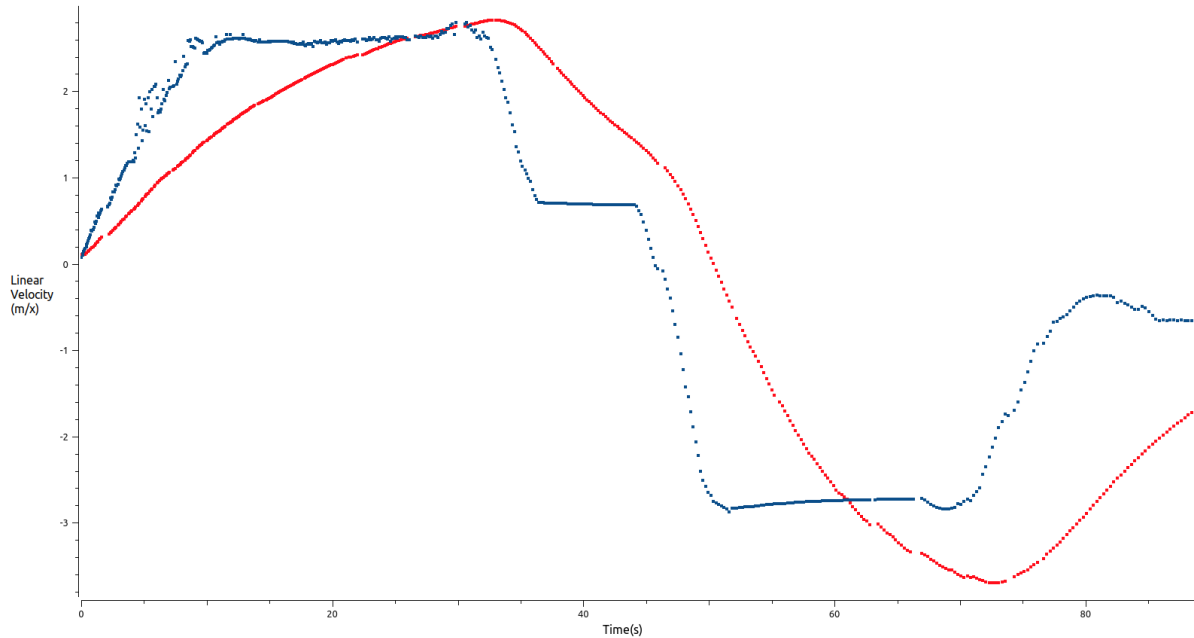


Figure 4.17: Linear velocity in x(East) direction. Kalman estimate is red, Wheel odometry measurement is blue. Before adjusting process noise covariance.

To make the kalman filter emphasize the contribution that the linear velocities that the wheel odometry plugin is contributing, the covariances for \dot{X} and \dot{Y} were adjusted to be larger. This new process covariance matrix had the effect that the filter followed the wheel odometry more closely as shown in figure 4.18.

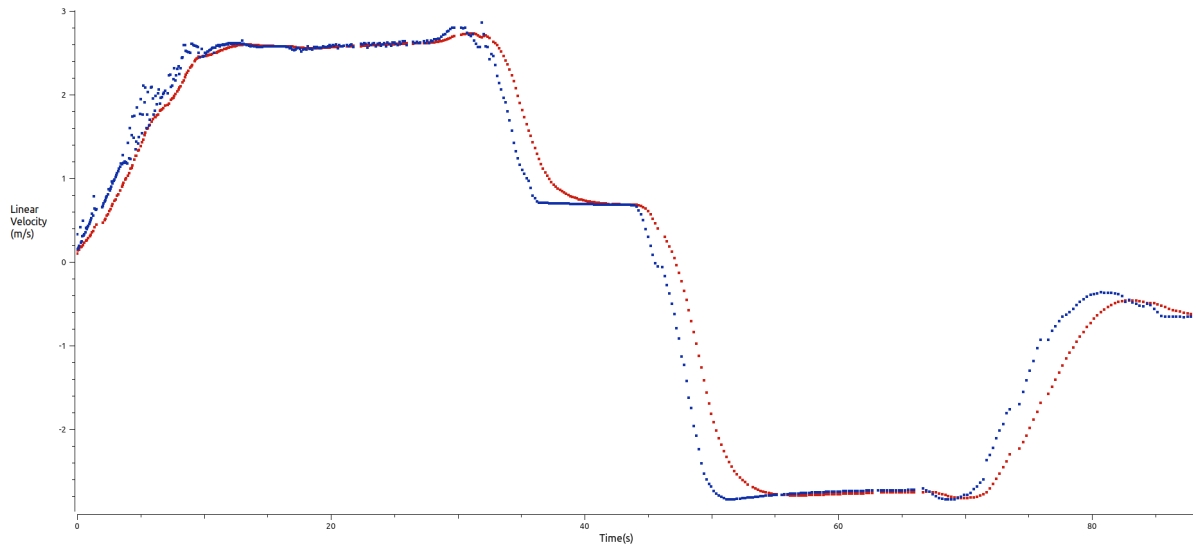


Figure 4.18: Linear velocity in x(East) direction. Kalman estimate is red, Wheel odometry measurement is blue. After adjusting process noise covariance.

This updated process noise covariance also gave the localization better performance as shown in figure 4.19.

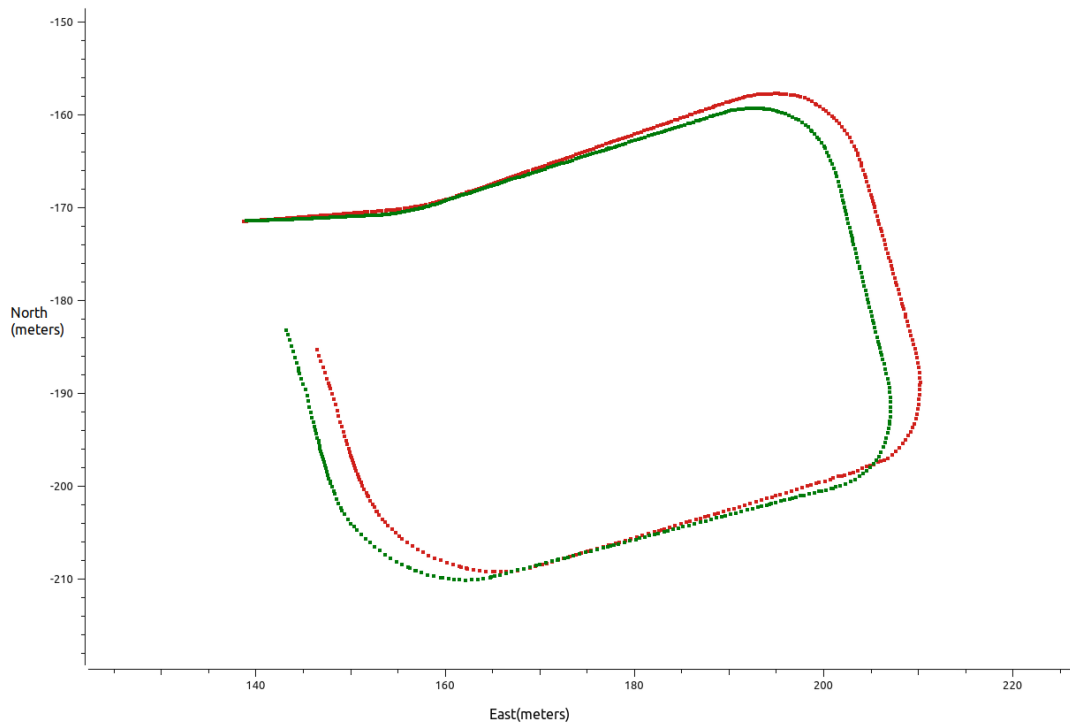


Figure 4.19: Localization with larger process noise covariance

Carefully tuning the process noise covariance further resulted in figure 4.20.

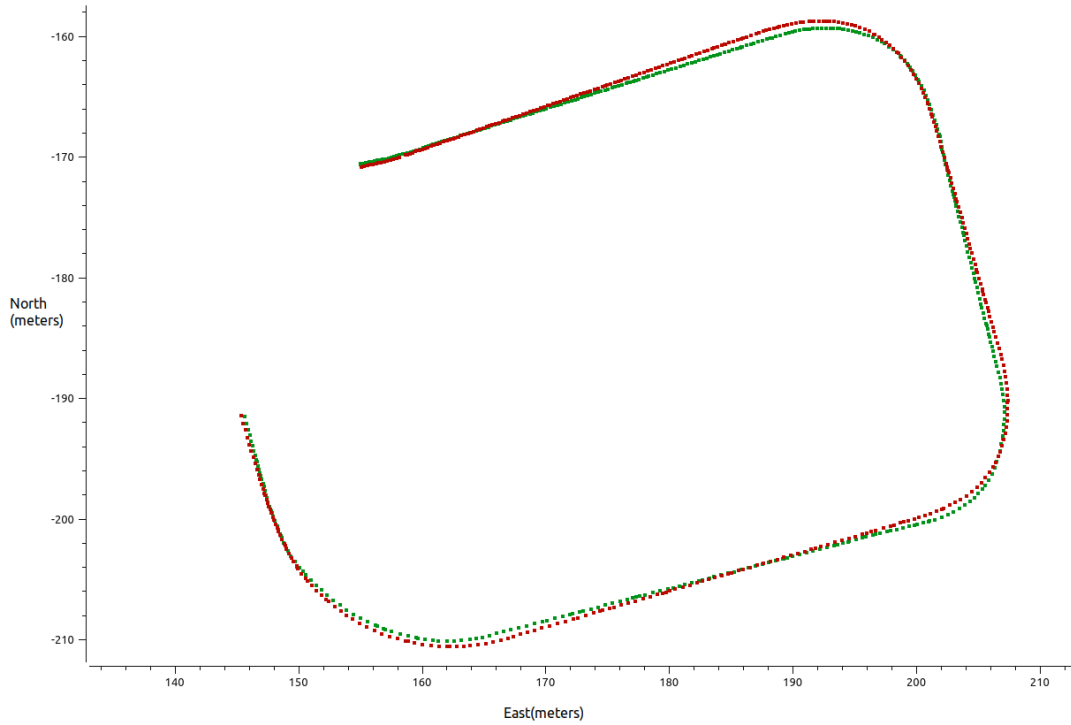


Figure 4.20: tuned process covariance, Kalman filtered estimate is red, true position is green.

The estimate error is shown in figure 4.21.

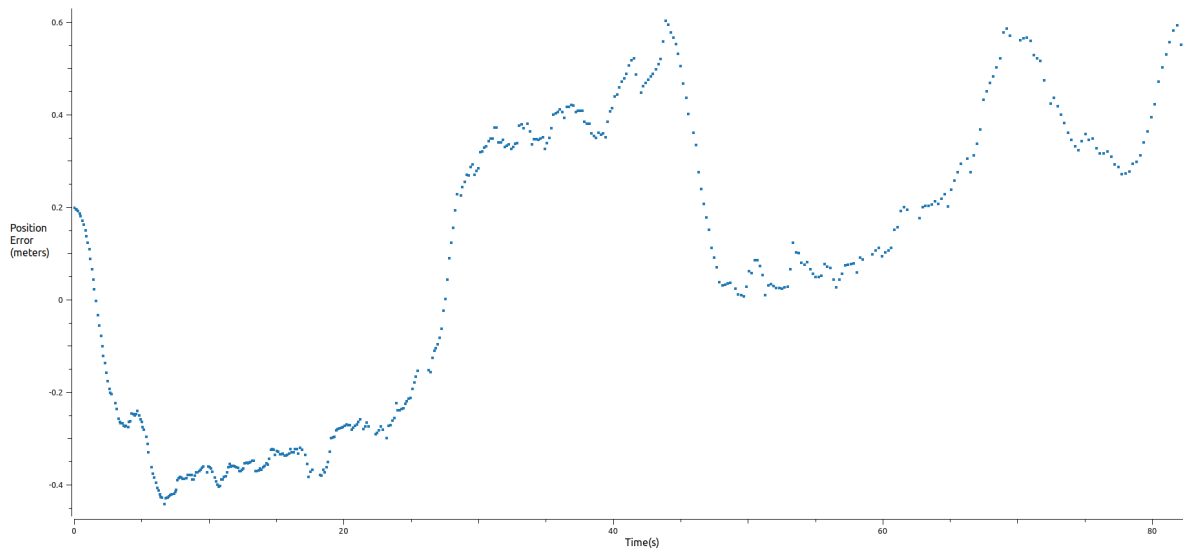


Figure 4.21: Tuned process covariance estimation error

Steering Input

Figure 4.22 shows the localization of the vehicle without using control as an acceleration term.

Figure 4.23 shows the same localization with steering input.

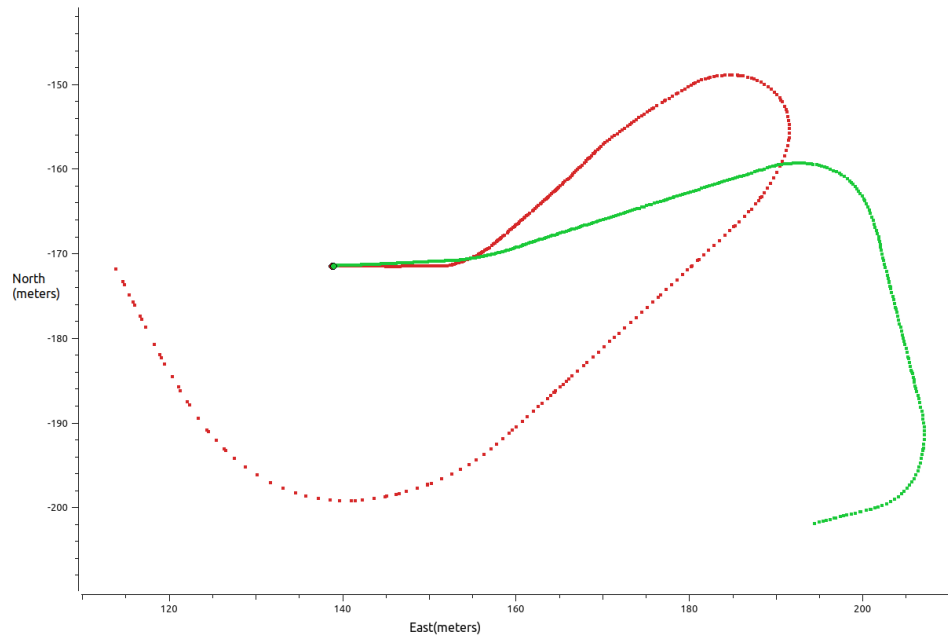


Figure 4.22: Localization without steering input

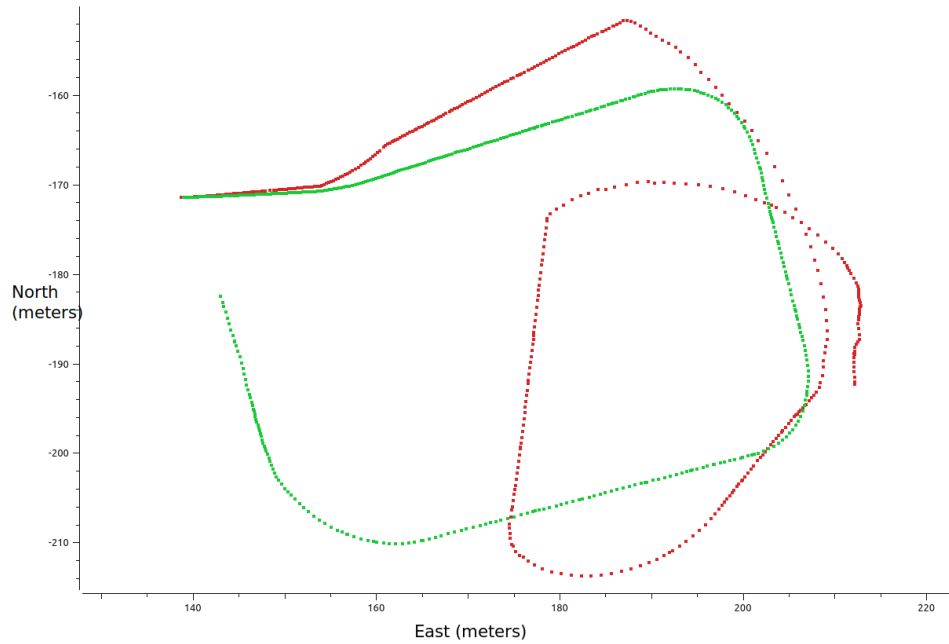


Figure 4.23: Localization with steering input

Dynamic process noise covariance

Figure 4.24 depicts the estimated covariance of the filtered position. In this figure it is growing constantly. When enabling dynamic process noise covariance, the result can be seen in figure 4.25.

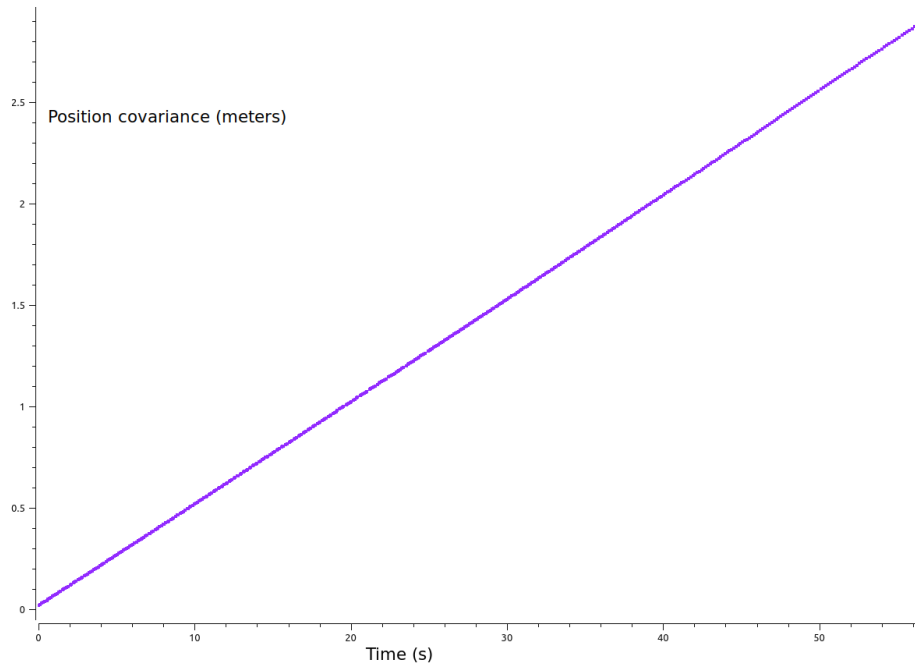


Figure 4.24: Position covariance without dynamic process noise covariance enabled

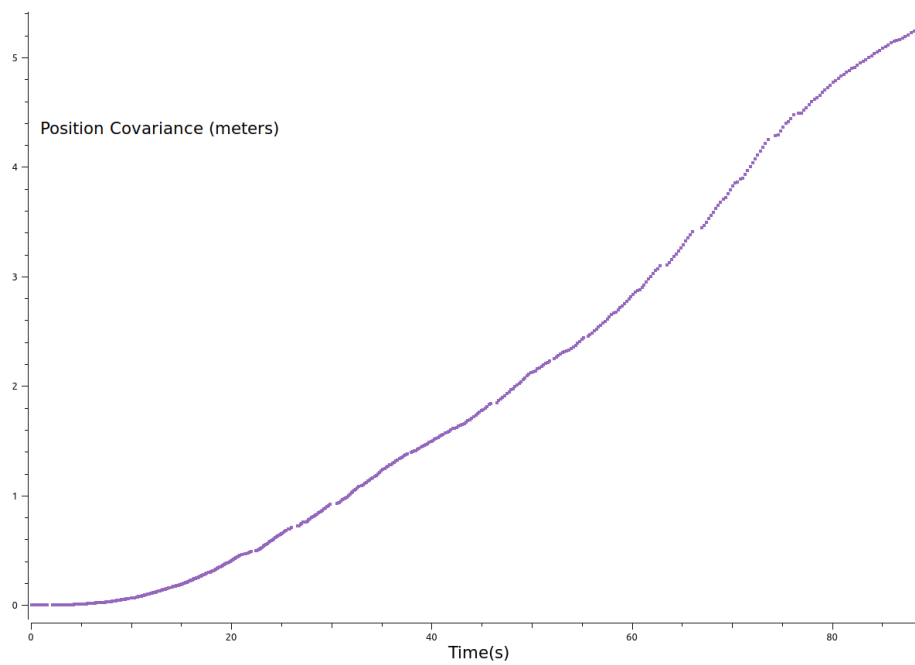


Figure 4.25: Position covariance without dynamic process noise covariance enabled

4.4 Performance

This section will attempt to convey the results of the different applications of localization in a clear way. Figure 4.1 shows all the combinations of sensors that were used, with the best performing filter chosen.

Wheel odometry

Figure 4.26 shows the localization of only wheel odometry localization. The resulting error in position is shown in figure 4.27.

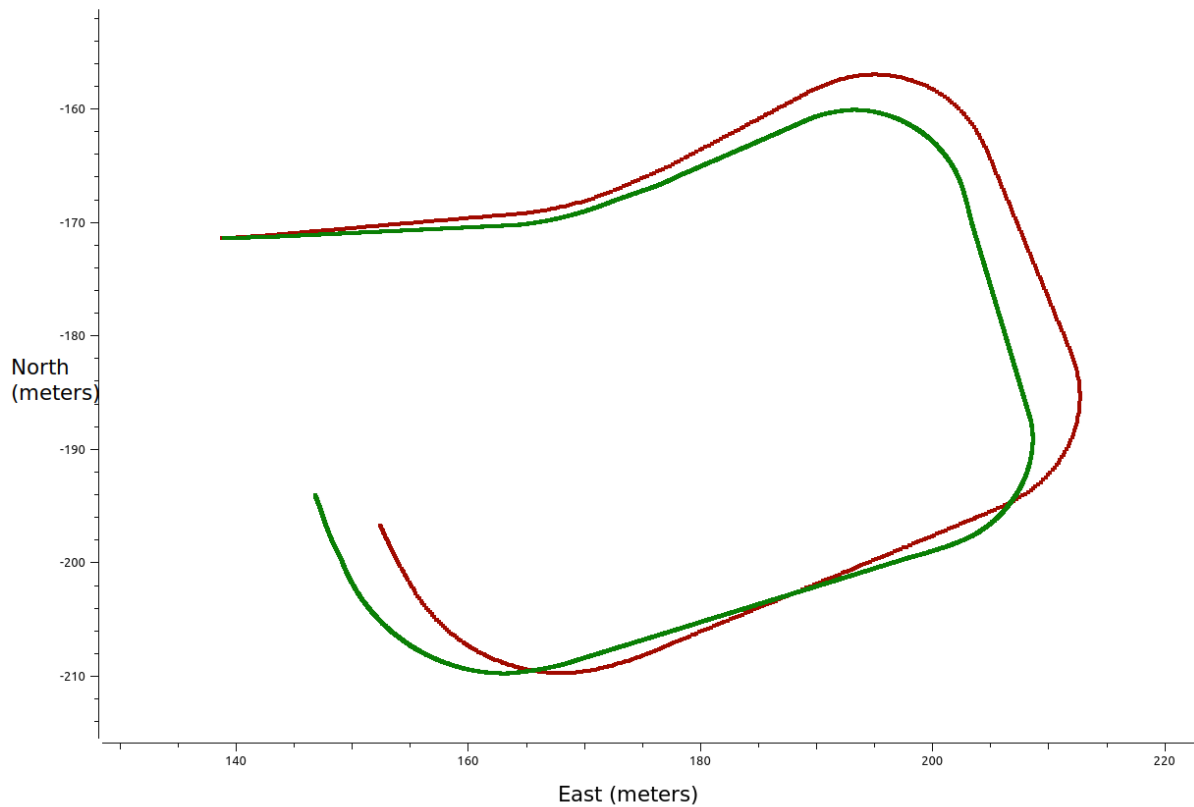


Figure 4.26: Localization with Wheel odometry. Estimated position is red, true position is green.

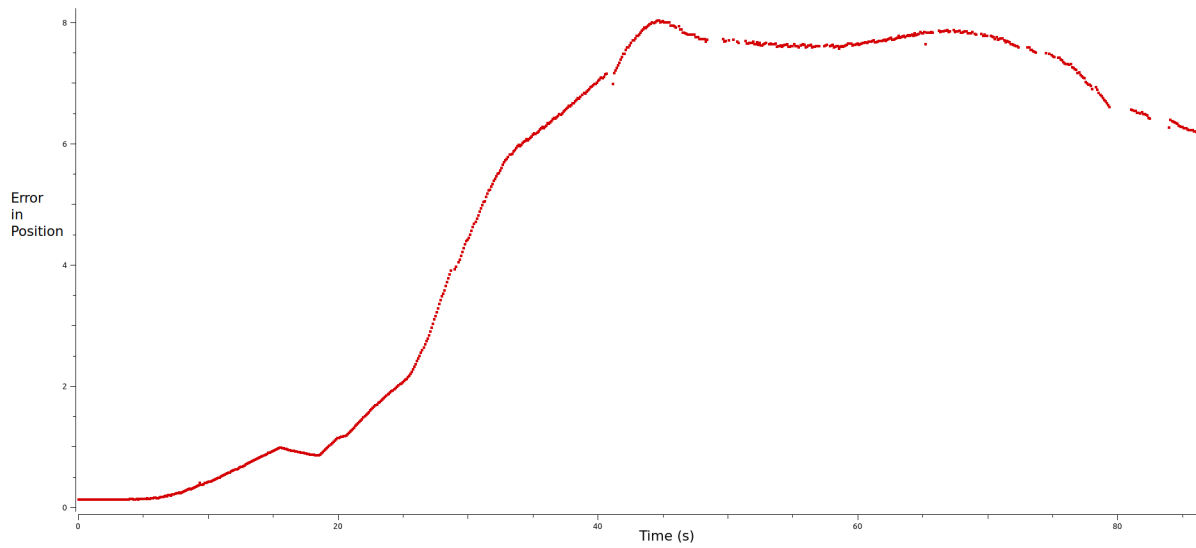


Figure 4.27: Error in position for localization with wheel odometry.

This localization is done with an UKF with default distribution values. The process noise covariance matrix can be seen in Appendix E, and the measurement noise covariance is shown in Appendix D.

IMU

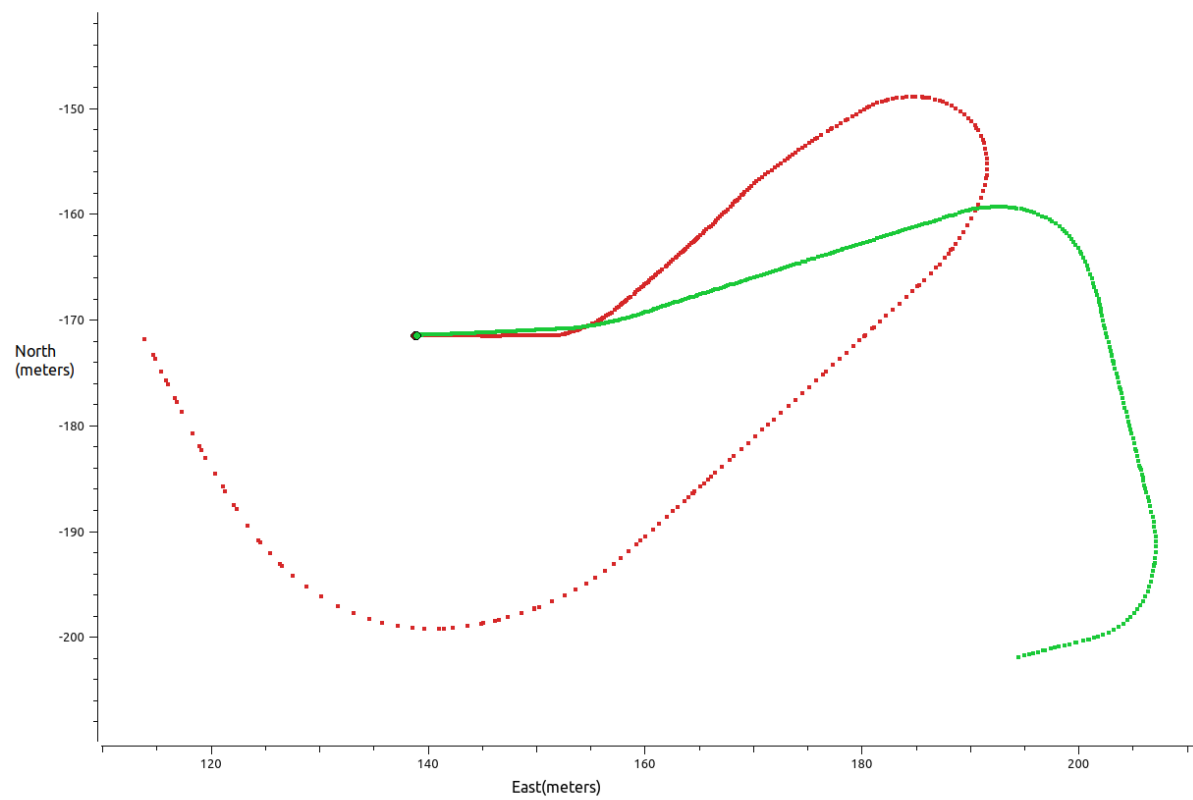


Figure 4.28: localization with IMU

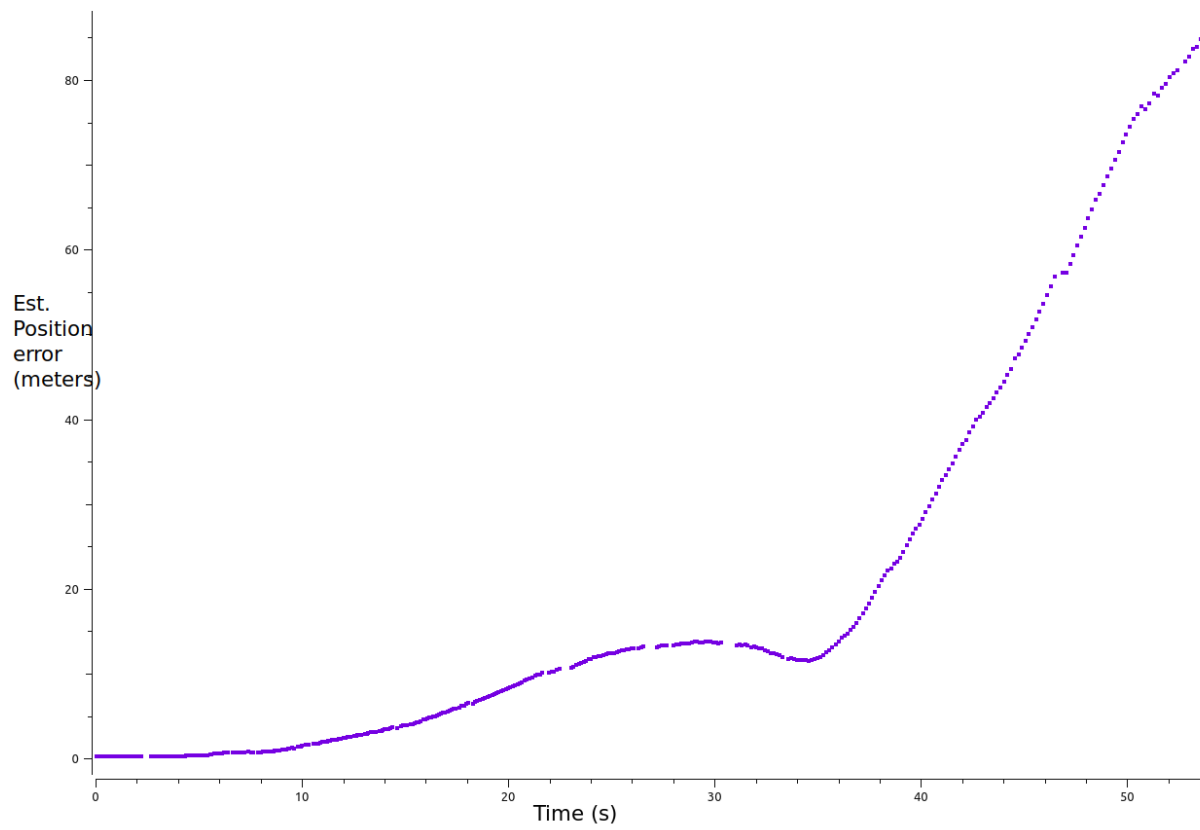


Figure 4.29: Error in estimated position with IMU.

Wheel Odometry and Gyroscope

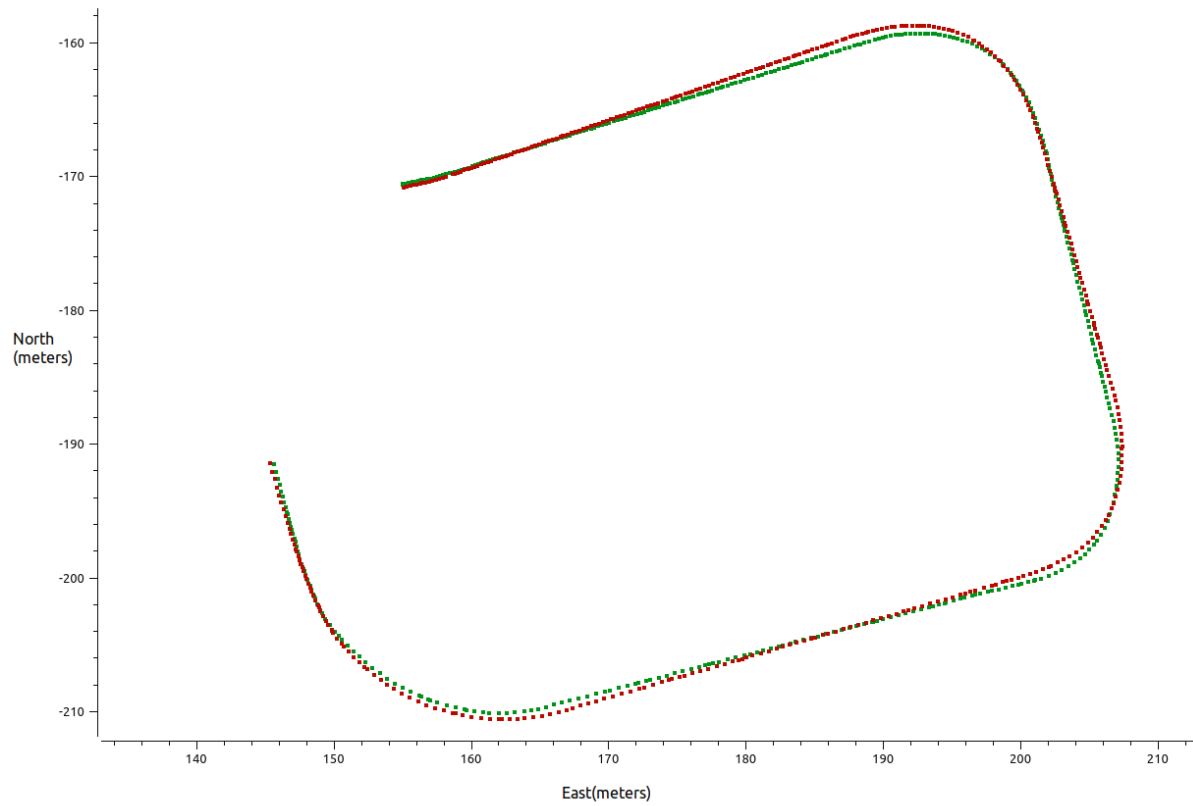


Figure 4.30: Localization with Wheel odometry and IMU

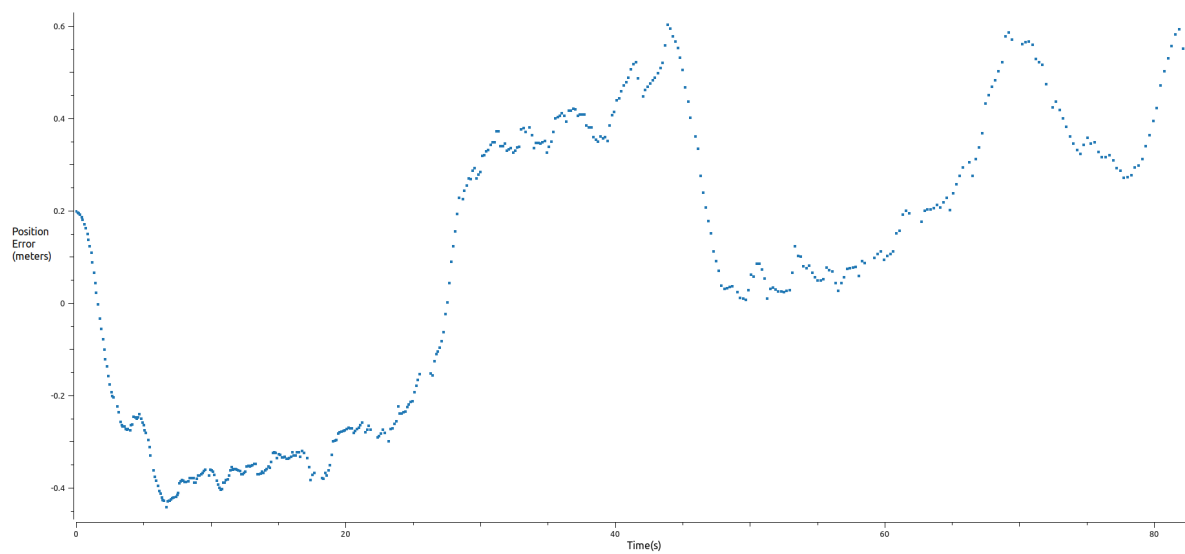


Figure 4.31: Localization error with wheel odometry and IMU

Wheel Odometry and IMU

Figure 4.32 shows the localization of combined wheel odometry and IMU. This localization shows little to performance benefits over Wheel odometry and gyroscope on their own, although there is a slight increase in accuracy over the short term(see table 4.1).

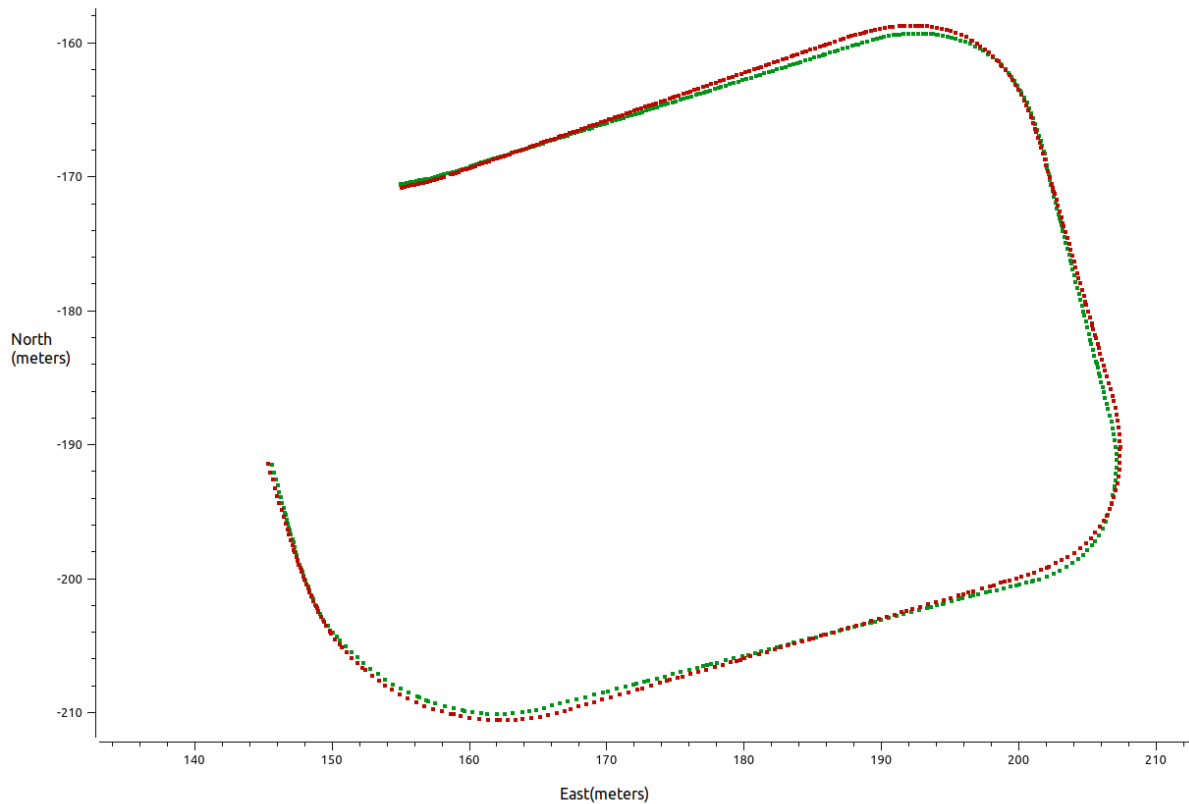


Figure 4.32: Localization with Wheel odometry and IMU. Estimated position is red, and true position is green.

Summary

Figure 4.1 shows the drift for the different combinations of filters at different time points.

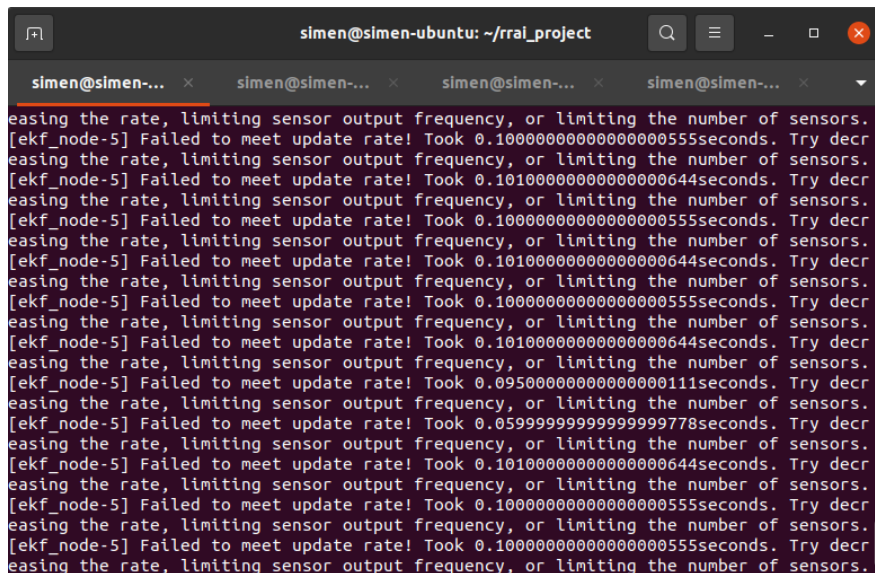
Application	drift after 5 seconds	30 seconds	60 seconds
Wheel odometry	0.13m	4.8m	7.657m
Wheel odometry, Gyroscope	0.31m	0.35m	0.14m
Wheel odometry, Gyroscope, Accelerometer	0.29m	0.36m	0.14m
Accelerometer, Gyroscope	0.09m	13.7m	89.1m

Table 4.1: Performance of localization

4.5 Challenges

4.5.1 Update Rate Errors

An error occurred with the `ekf_node` where it was stated that the filter failed to meet the update rate as shown in figure 4.33.

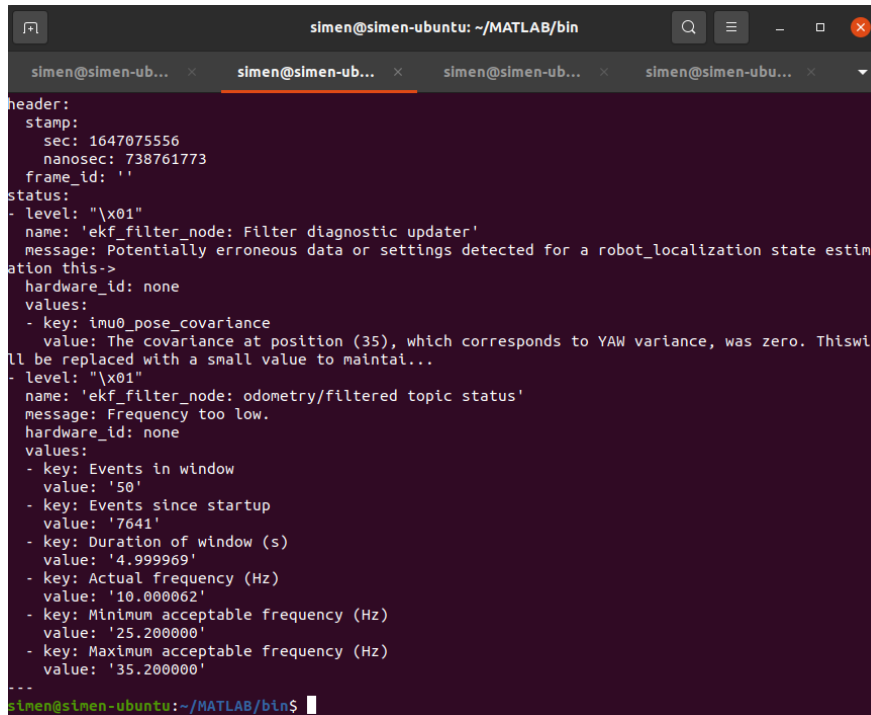


```

simen@simen-ubuntu: ~/rral_project
simen@simen-... x simen@simen-... x simen@simen-... x simen@simen-... x
easing the rate, limiting sensor output frequency, or limiting the number of sensors.
[ekf_node-5] Failed to meet update rate! Took 0.100000000000000555seconds. Try decr
easing the rate, limiting sensor output frequency, or limiting the number of sensors.
[ekf_node-5] Failed to meet update rate! Took 0.101000000000000644seconds. Try decr
easing the rate, limiting sensor output frequency, or limiting the number of sensors.
[ekf_node-5] Failed to meet update rate! Took 0.100000000000000555seconds. Try decr
easing the rate, limiting sensor output frequency, or limiting the number of sensors.
[ekf_node-5] Failed to meet update rate! Took 0.101000000000000644seconds. Try decr
easing the rate, limiting sensor output frequency, or limiting the number of sensors.
[ekf_node-5] Failed to meet update rate! Took 0.100000000000000555seconds. Try decr
easing the rate, limiting sensor output frequency, or limiting the number of sensors.
[ekf_node-5] Failed to meet update rate! Took 0.101000000000000644seconds. Try decr
easing the rate, limiting sensor output frequency, or limiting the number of sensors.
[ekf_node-5] Failed to meet update rate! Took 0.095000000000000111seconds. Try decr
easing the rate, limiting sensor output frequency, or limiting the number of sensors.
[ekf_node-5] Failed to meet update rate! Took 0.059999999999999778seconds. Try decr
easing the rate, limiting sensor output frequency, or limiting the number of sensors.
[ekf_node-5] Failed to meet update rate! Took 0.101000000000000644seconds. Try decr
easing the rate, limiting sensor output frequency, or limiting the number of sensors.
[ekf_node-5] Failed to meet update rate! Took 0.100000000000000555seconds. Try decr
easing the rate, limiting sensor output frequency, or limiting the number of sensors.
[ekf_node-5] Failed to meet update rate! Took 0.100000000000000555seconds. Try decr
easing the rate, limiting sensor output frequency, or limiting the number of sensors.

```

Figure 4.33: Robot_localization package update rate error



```

simen@simen-ubuntu: ~/MATLAB/bin
header:
  stamp:
    sec: 1647075556
    nanosec: 738761773
  frame_id: ''
status:
- level: "\x01"
  name: 'ekf_filter_node: Filter diagnostic updater'
  message: Potentially erroneous data or settings detected for a robot_localization state estimation this->
  hardware_id: none
  values:
    - key: imu0_pose_covariance
      value: The covariance at position (35), which corresponds to YAW variance, was zero. This will be replaced with a small value to maintain...
- level: "\x01"
  name: 'ekf_filter_node: odometry/filtered topic status'
  message: Frequency too low.
  hardware_id: none
  values:
    - key: Events in window
      value: '50'
    - key: Events since startup
      value: '7641'
    - key: Duration of window (s)
      value: '4.999969'
    - key: Actual frequency (Hz)
      value: '10.000062'
    - key: Minimum acceptable frequency (Hz)
      value: '25.200000'
    - key: Maximum acceptable frequency (Hz)
      value: '35.200000'
---
simen@simen-ubuntu:~/MATLAB/bin$

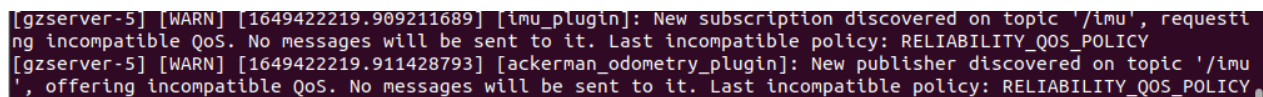
```

Figure 4.34: ROS2 "/diagnostic" topic output

By looking at the robot_localization diagnostic topic I could see that the filter rejected any filter update rate under 25hz and over 35hz (figure 4.34). In comparison, an update time of 0.1 seconds as shown in figure 4.33 corresponds to an update rate of 10hz. Suspecting that something was slowing down the filter, It turned out that debug mode being enabled caused the delay.

4.5.2 Qos (Quality of service) parameter mismatch

When subscribing to the imu topic from the ackerman odometry plugin, the follow error occurred when running the launch file.



```

[gzserver-5] [WARN] [1649422219.909211689] [imu_plugin]: New subscription discovered on topic '/imu', requesting incompatible QoS. No messages will be sent to it. Last incompatible policy: RELIABILITY_QOS_POLICY
[gzserver-5] [WARN] [1649422219.911428793] [ackerman_odometry_plugin]: New publisher discovered on topic '/imu', offering incompatible QoS. No messages will be sent to it. Last incompatible policy: RELIABILITY_QOS_POLICY

```

Figure 4.35: Qos parameter error

Figure 4.36 shows how the "imu_plugin" publishes to the "/imu" topic with the Qos reliability parameter "BEST EFFORT". On the contrary, the "ackerman_odometry_plugin" subscribes to

the `/imu` topic with the Qos reliability parameter `"RELIABLE"`. No topic data is being send to the subscriber either when this error message occurred.

```

stmengstmen-ubuntu:~/rrat_project$ ros2 topic info /imu --verbose
Type: sensor_msgs/msg/Imu

Publisher count: 1

Node name: imu_plugin
Node namespace: /
Topic type: sensor_msgs/msg/Imu
Endpoint type: PUBLISHER
GID: 12.2d.10.01.33.0b.16.82.31.6d.b5.a0.00.00.3e.03.00.00.00.00.00.00.00
QoS profile:
  Reliability: BEST_EFFORT
  Durability: VOLATILE
  Lifespan: 9223372036854775807 nanoseconds
  Deadline: 9223372036854775807 nanoseconds
  Liveliness: AUTOMATIC
  Liveliness lease duration: 9223372036854775807 nanoseconds

Subscription count: 1

Node name: ackerman_odometry_plugin
Node namespace: /
Topic type: sensor_msgs/msg/Imu
Endpoint type: SUBSCRIPTION
GID: 12.2d.10.01.33.0b.16.82.31.6d.b5.a0.00.00.71.04.00.00.00.00.00.00.00
QoS profile:
  Reliability: RELIABLE
  Durability: VOLATILE
  Lifespan: 9223372036854775807 nanoseconds
  Deadline: 9223372036854775807 nanoseconds
  Liveliness: AUTOMATIC
  Liveliness lease duration: 9223372036854775807 nanoseconds

```

Figure 4.36: Qos parameter mismatch debugging

Referring to the Ros2 Galactic Wiki, the compatibility table (figure 4.37) shows that this combination of parameters isn't compatible. This makes sense according to the wiki[].

Publisher	Subscription	Compatible
Best effort	Best effort	Yes
Best effort	Reliable	No
Reliable	Best effort	Yes
Reliable	Reliable	Yes

Figure 4.37: Compatibility of reliability QoS policies[]

For sensor data, it's important to receive readings as they are registered rather than making sure every reading is received. This is why the QoS reliability parameter `"BEST_EFFORT"` is best suited for sensor data as it prioritizes the timely arrival of readings[].

Luckily, Ros2 already has QoS parameter profiles for use with sensor data[].

To use this profile, the QoS profile has to be declared when creating the subscription. The documentation for the `"create_subscription"` function declaration, shows that the function argument `"qos"` can be defined as shown in figure 4.39.

```

46 // Here a subscriber is created to be used to fetch imu heading
47 heading_sub_ = node_ ->create_subscription<sensor_msgs::msg::Imu>("imu", "10",
48   std::bind(&AckermanOdom::Callback, this, std::placeholders::_1));

```

Figure 4.38: Using standard qos profile

```

43 // since the imu topic has non-matching QoS settings, we have to change them manually
44 auto qos_profile = rclcpp::QoS(rclcpp::KeepLast(10), rmw_qos_profile_sensor_data);
45
46 // Here a subscriber is created to be used to fetch imu heading
47 heading_sub_ = node_ ->create_subscription<sensor_msgs::msg::Imu>("imu", qos_profile,
48   std::bind(&AckermanOdom::Callback, this, std::placeholders::_1));

```

Figure 4.39: Using "sensor data" qos profile

After compiling and running the program again, there were no errors. Using the "ros2 topic info" command in the terminal shows that both the publisher and subscriber of the topic has the reliability parameter of "BEST_EFFORT".

```

simengsinen-ubuntu:~/rral_project$ ros2 topic info /imu --verbose
Type: sensor_msgs/msg/Imu

Publisher count: 1

Node name: imu_plugin
Node namespace: /
Topic type: sensor_msgs/msg/Imu
Endpoint type: PUBLISHER
GUID: 6b.29.10.01.cd.1e.8f.28.21.79.70.f5.00.00.3e.03.00.00.00.00.00.00.00
QoS profile:
  Reliability: BEST_EFFORT
  Durability: VOLATILE
  Lifespan: 9223372036854775807 nanoseconds
  Deadline: 9223372036854775807 nanoseconds
  Liveliness: AUTOMATIC
  Liveliness lease duration: 9223372036854775807 nanoseconds

Subscription count: 1

Node name: ackerman_odometry_plugin
Node namespace: /
Topic type: sensor_msgs/msg/Imu
Endpoint type: SUBSCRIPTION
GUID: 6b.29.10.01.cd.1e.8f.28.21.79.70.f5.00.00.71.04.00.00.00.00.00.00.00
QoS profile:
  Reliability: BEST_EFFORT
  Durability: VOLATILE
  Lifespan: 9223372036854775807 nanoseconds
  Deadline: 9223372036854775807 nanoseconds
  Liveliness: AUTOMATIC
  Liveliness lease duration: 9223372036854775807 nanoseconds

```

Figure 4.40: Qos parameter debugging

Chapter 5

Discussion

This chapter contains discussion regarding comparisons and what claims are supported in the literary survey in chapter 1.3 and otherwise in the theory section(chapter 2). The validity of the findings are discussed, possible interpretations, possible shortcomings of the methods that were used, and what could have been done differently.

5.1 Sensors

5.1.1 Wheel Odometry

Wheel odometry can be seen to be sensitive to errors in initialization, and systematic calibration[7] was attempted to compensate for errors in position. This was difficult, although further careful tuning can possibly be done.

5.1.2 Inertial Measurement Unit

The use of accelerometers, especially in low velocity driving has been implemented before[7][4], which resulted in relatively poor performance. It is shown in figure 4.10 that position estimates with only IMU sensor measurements diverges relatively quickly, however, the IMU is the most accurate in the first seconds as shown in table 4.1.

5.1.3 Sensor Fusion

Wheel odometry and directional gyroscope

The combination of wheel odometry and a directional gyroscope(heading) proved to be much more accurate than either of them alone. Everett B. Borenstein[7] achieved a drift of about 8 inches(20cm) over the span of with a following encoder. This hints towards the fact that slip is a big contributing factor for error in wheel odometry. It makes sense that the Ackerman model(figure 3.1) has some slip because of the fact that the wheels are wide, and load bearing(as discussed in chapter 2.1.5). This is anyways a representation of expected wheel odometry performance without slip detection for the specific vehicle model that Redrock supplied.

Wheel odometry and IMU

Wheel odometry, and IMU fusion showed no significant improvements over the combination of wheel odometry and directional gyroscope. This could be because of the fact that the vehicle is inherently slow moving, and as a result accelerometers cant contribute as much as they are better for rapidly moving agents. Possible improvements that could be made is increasing the kalman filter frequency rate. The filter only ran at about 30hz, which was the highest that was achievable on the hardware that was used.

5.2 Unscented Kalman Filter

Figure 4.14 shows that the ukf provides a better estimate of position in comparison to the ekf filter. Therefore, the Unscented Kalman Filter also has less linear velocity error. This supports the claim from *Vadim Bistrov and Ansis Kluga*[6] stating that the UKF has lower velocity estimation error in comparison to the EKF, although this claim is only supported by a small sample.

5.3 Process Noise Covariance

Tuning the process noise covariance matrix was extremely important, as vast improvements were made when it was tuned. This was true for all of the localizations, as every localization was

significantly worse, when this matrix was not tuned. An example of the improvements made is shown in chapter 4.3.4, where the kalman filter was shown to highly disregard wheel odometry readings when it was wrongly configured.

5.4 Steering Input

The use of steering input did not provide any significant advantages, but it seemed that the constraints were very difficult to tune. It is also possible that the method of steering the vehicle isn't representative of the steering input that would be used in the real world, as the steering input that was applied was highly discrete, whereas in a real use case the steering input would most likely be more continuous.

5.5 Dynamic Process Noise Covariance

Dynamic process noise covariance was shown in figure 4.24 and 4.25. This improvement has no impact of the localization performance, but increases the accuracy of the covariance, especially when the localization is later combined with position fixing measurements.

5.6 Shortcomings

Although tuning the state estimation filters for the same driving route was possible, while changing the physical parameters, the vehicle had to be recorded once again. This makes it harder to compare localization with changes in physical parameters for the same driving. If it is desirable to be able to compare changes in physical, possible changes could be made to implement wheel odometry in a ROS node instead of a Gazebo plugin, although this would certainly also cause some delay in odometry output.

Another shortcoming is the lack of samples. If further samples were recorded for different driving patterns (such as standstill, straight line, and multiple arbitrary driving patterns), the robustness and performance metrics of each localization would have more credibility.

5.7 Objectives

To answer the first research question in chapter 1.4, when does the position drift, and how fast does it drift? For wheel odometry, position primarily starts to drift when the vehicle is turning. In the arbitrary driving pattern that was used, the vehicle drifted on the order of a meter per second of turning, although this was dependant on the sharpness of the turn. For IMU localization, the position started to drift after a few seconds, and then drifted rapidly. The combination of Wheel odometry and gyroscope also drifted primarily while turning, but less than wheel odometry on its own. This application drifted on the order of a decimeter per second of turning, also being dependant on the sharpness of the turn. Wheel odometry and IMU together performed similarly or worse than wheel odometry and gyroscope depending on the tuning parameters of the estimation filter. The other research questions were, how does dead-reckoning affect localization in absence of GNSS, how few, and which, sensors are needed to maintain satisfactory performance. To answer the first question, dead-reckoning was shown to always cause a drift in position, where periodic absolute position measurements are necessary. This is also supported by existing literature [7]. The only thing that changes is the required frequency of absolute position measurements. As for how few and which sensors are needed, it is suggested that wheel odometry is important for mobile robot navigation which also supports existing literature on odometry[7](chapter 1.3), and that a heading measurement greatly improves localization accuracy. Therefore, wheel odometry and heading measurement are suggested to be the fewest possible for some applications, and the most needed.

Chapter 6

Conclusions and further work

The goal of this report was to evaluate the performance characteristics of different sensors on an ackerman based vehicle model, as well as finding out how few could be used, and which needed to be used to give satisfactory localization performance during dead-reckoning. This was without knowing exactly what "satisfactory" entailed. Using the context for the use cases that Re-dRock AI has, certain requirements were discussed. The results of the simulations show that wheel odometry is overwhelmingly necessary for accurate localization, and combined wheel odometry and heading measurements improve localization further. This was contrary to the performance of an IMU which quickly diverges, although it was discussed that this may be a result of the slow nature of the simulated vehicle or the slow frequency of the kalman filter that was used. Therefore, the combination of wheel odometry and a directional gyroscope was proposed as the best possible solution, while still giving relatively good performance, although the localization was not as good as what was desired. This is because centimeter level precision was desired, but decimeter level precision was achieved.

Further work could include some of the following. The wheel odometry in this project is implemented as if the radius of the wheels are constant. In the real world, this is often not true. Over the lifespan of a rubber wheel, the radius can decrease as much as 3%[\[10\]](#). This error in radius will make the odometry diverge more quickly, as small errors in the odometry are integrated. A Possible solution to this is to implement what is called an error-state Kalman Filter[\[10\]](#). This is a type of filter that models the errors of the state, instead of the actual states of the system. This

type of implementation can often be better, as the dynamics of the errors of a system can more accurately be modeled. The errors in a system are also more often than not, simpler than the actual states. Further systematic calibration can also be done with the method described[7].

The wheels of any vehicle can naturally also slip. Implementing a detection algorithm for slip, could improve the performance of wheel odometry. Seyr, Martin and Jakubek, Stefan[22], and Paul D. Groves[10] shows examples of slip detection. Another alternative is to implement encoders on the rear wheels of the vehicle[7]. This implementation would in theory be as easy as implementing the kinematics of the rear wheels, and then comparing their encoder values with the front wheels.

Bibliography

- [1] Ieee standard specification format guide and test procedure for single-axis laser gyros. *IEEE Std 647-2006 (Revision of IEEE Std 647-1995)*, pages 1–96, 2006. doi: 10.1109/IEEESTD.2006.246241.
- [2] Et al. Aggarwal, P. *MEMS based Integrated Navigation*. Artech House, 2010. ISBN 978-1-60807-043-5.
- [3] Antonio Angrisano. *GNSS/INS Integration Methods*. PhD thesis, Schulich School of Engineering, 01 2010.
- [4] B. Barshan and H.F. Durrant-Whyte. Inertial navigation systems for mobile robots. *IEEE Transactions on Robotics and Automation*, 11(3):328–342, 1995. doi: 10.1109/70.388775.
- [5] Ali Barzegar, Oualid Doukhi, and Deok-Jin Lee. Design and implementation of an autonomous electric vehicle for self-driving control under gnss-denied environments. *Applied Sciences*, 11(8), 2021. ISSN 2076-3417. doi: 10.3390/app11083688. URL <https://www.mdpi.com/2076-3417/11/8/3688>.
- [6] Vadim Bistrov and Ansis Kluga. The analysis of the ukf-based navigation algorithm during gps outage. *Electronics and Electrical Engineering*, 19, 12 2013. doi: 10.5755/j01.eee.19.10.5886.
- [7] J. Borenstein, H. R. Everett, and L. Feng. "where am i?" sensors and methods for mobile robot positioning, 1996.
- [8] Ingemar J. Cox. Blanche-an experiment in guidance and navigation of an autonomous robot vehicle. *IEEE Transactions Robotics and Automation*, 7:193–204, 1991.

- [9] Trimmer W Gabriel K, Jarvis J. A report on the emerging field of microdynamics: report of the workshop on microelectromechanical systems research. *ATT Bell Laboratories*, 1988.
- [10] Paul D. Groves. *Principles of GNSS, Inertial, and Multisensor Integrated Navigation Systems*, volume 2. Artech House, 2013. ISBN-13: 978-1608070053.
- [11] Fredrik Gustafsson and Gustaf Hendeby. Some relations between extended and unscented kalman filters. *IEEE Transactions on Signal Processing*, 60(2):545–555, 2012. doi: 10.1109/tsp.2011.2172431.
- [12] T. Moore Hide, C. and M. Smith. Adaptive kalman filtering for low cost ins/gps navigation applications. *Journal of Navigation*, 56:143–152, 2003.
- [13] J. Hollingum. Caterpillar make the earth move: automatically.
- [14] Guoquan P. Huang, Anastasios I. Mourikis, and Stergios I. Roumeliotis. Analysis and improvement of the consistency of extended kalman filter based slam. *2008 IEEE International Conference on Robotics and Automation*, page 473–479, 2008. doi: 10.1109/robot.2008.4543252.
- [15] Simon J. Julier and Jeffrey K. Uhlmann. New extension of the kalman filter to nonlinear systems. *SPIE Proceedings*, 3:182–193, 1997. doi: 10.1117/12.280797.
- [16] S.J. Julier and J.K. Uhlmann. Unscented filtering and nonlinear estimation. *Proceedings of the IEEE*, 92(3):401–422, 2004. doi: 10.1109/jproc.2003.823141.
- [17] I. Erkmen Kaygisiz, B. H. and A. M Erkmen. Gps/ins enhancement for land navigation using neural network. *Journal of Navigation*, 57:297–310, 2004.
- [18] J.B. Pletta, W.A. Amai, P. Klarer, D. Frank, Jeffrey Carlson, and Raymond Byrne. The remote security station (rss) final report. 10 1992.
- [19] B. Goldsmith R. Dimitrid. *Math. intelligencer*, 11, 1989.
- [20] Alessandro Ridolfi, B. Allotta, Luigi Chisci, Riccardo Costanzi, Francesco Fanelli, Enrico Meli, Claudio Fantacci, Andrea Caiti, Francesco Di Corato, and Davide Fenucci. A com-

- parison between ekf-based and ukf-based navigation algorithms for auvs localization. 05 2015. doi: 10.1109/OCEANS-Genova.2015.7271681.
- [21] Mohammad Taghi Sabet, Hamidreza Mohammadi Daniali, Alireza Fathi, and Ebrahim Alizadeh. A low-cost dead reckoning navigation system for an auv using a robust ahrs: Design and experimental analysis. *IEEE Journal of Oceanic Engineering*, 43(4):927–939, 2018. doi: 10.1109/JOE.2017.2769838.
- [22] Martin Seyr and Stefan Jakubek. Proprioceptive navigation, slip estimation and slip control for autonomous wheeled mobile robots. In *2006 IEEE Conference on Robotics, Automation and Mechatronics*, pages 1–6, 2006. doi: 10.1109/RAMECH.2006.252627.
- [23] E.A. Wan and R. Van Der Merwe. The unscented kalman filter for nonlinear estimation. In *Proceedings of the IEEE 2000 Adaptive Systems for Signal Processing, Communications, and Control Symposium (Cat. No.00EX373)*, pages 153–158, 2000. doi: 10.1109/ASSPCC.2000.882463.
- [24] Cheng Yang, Wenzhong Shi, and Wu Chen. Comparison of unscented and extended kalman filters with application in vehicle navigation. *Journal of Navigation*, 70(2):411–431, 2017. doi: 10.1017/S0373463316000655.
- [25] Yueming Zhao. Gps/imu integrated system for land vehicle navigation based on mems. 2011.

Appendices

A Preproject report(Norwegian)

FORPROSJEKT - RAPPORT

FOR BACHELOROPPGAVE

TITTEL:

Robot Localization

KANDIDATNUMMER(E):

DATO:

21.01.2022

EMNEKODE:

IE303612

EMNE:

Bacheloroppgave

DOKUMENT TILGANG:

- Åpen

STUDIUM:

ELEKTROINGENIØR - AUTOMASJON

ANT SIDER/VEDLEGG:

8/0

BIBL. NR:

- Ikke i bruk -

OPPDRAKSGIVER(E)/VEILEDER(E):

Nikolai Persen, Andreas Fjetland(RedRock.AI)/Erlend Coates, Eirik Fagerhaug

OPPGAVE/SAMMENDRAG:

Denne rapporten er skrevet av en bachelor student som går 3. året på Elektroingeniør med hovedretning Automasjon på NTNU i Ålesund. Oppdragsgiveren til oppgaven er RedRock.AI, ett firma i Kristiansand som jobber med autonom lasthåndtering med mål om å minke driftskostnadene til kundene. Bacheloroppgaven handler om å utforme og iverksette tilfredsstillende lokalisering av en UGV med den mest økonomiske løsningen, samtidig som bærekraft og etikk tas i betraktning. Forprosjektrapporten definerer hvordan bacheloroppgaven skal gjennomføres og hvilke problemstillinger som skal løses. Denne rapporten er ett arbeidskrav som skrives i begynnelsen av semesteret og som danner grunnlaget til en omfattende bacheloroppgave.

Denne oppgaven er en eksamensbesvarelse utført av student(er) ved NTNU i Ålesund.

Postadresse
Høgskolen i Ålesund
N-6025 Ålesund
Norway

Besøksadresse
Larsgårdsvegen 2
Internett
www.hials.no

Telefon
70 16 12 00
Epostadresse
postmottak@hials.no

Telefax
70 16 13 00

Bankkonto
7694 05 00636
Foretaksregisteret
NO 971 572 140

INNHold

1 INNLEDNING	3
2 BEGREPER	3
3 PROSJEKTORGANISASJON	3
3.1 PROSJEKTGRUPPE.....	3
3.2 STYRINGSGRUPPE (VEILEDER OG KONTAKTPERSON OPPDRAGSGIVER).....	4
4 AVTALER	4
4.1 AVTALE MED OPPDRAGSGIVER.....	4
4.2 ARBEIDSSTED OG RESSURSER.....	4
4.3 GRUPPENORMER – SAMARBEIDSREGLER – HOLDNINGER.....	4
5 PROSJEKTBEKRIVELSE	4
5.1 PROBLEMSTILLING - MÅLSETTING - HENSIKT.....	4
5.2 KRAV TIL LØSNING ELLER PROSJEKTRESULTAT – SPESIFIKASJON.....	4
5.3 PLANLAGT FRAMGANGSMÅTE(R) FOR UTVIKLINGSARBEIDET – METODE(R).....	4
5.4 INFORMASJONSINNSAMLING – UTFØRT OG PLANLAGT.....	5
5.5 VURDERING – ANALYSE AV RISIKO.....	5
5.6 HOVEDAKTIVITETER I VIDERE ARBEID.....	5
5.7 FRAMDRIFTSPLAN – STYRING AV PROSJEKTET.....	5
5.8 BESLUTNINGER – BESLUTNINGSPROSESS.....	6
6 DOKUMENTASJON	6
6.1 RAPPORTER OG TEKNISKE DOKUMENTER.....	6
7 PLANLAGTE MØTER OG RAPPORTER	6
7.1 MØTER.....	6
7.2 PERIODISKE RAPPORTER.....	6
8 PLANLAGT AVVIKSBEHANDLING	6
9 UTSTYRSBEHOV/FORUTSETNINGER FOR GJENNOMFØRING	7
10 REFERANSER	7
VEDLEGG	7

1 INNLEDNING

Denne rapporten er skrevet av en bachelor student som går 3. året på Elektroingenør med hovedretning Automasjon på NTNU i Ålesund. Oppdragsgiveren til oppgaven er RedRock.AI, ett firma i Kristiansand som jobber med autonom lasthåndtering med mål om å minke driftskostnadene til kundene. Oppgaven er å utforme ideer for, og teste forskjellige metoder for lokalisering av en UGV med minst antall sensorer. Hensikten er å skape økonomisk verdi for evt. Produkt eller tjeneste. Oppgaven skal til slutt vise til ett eller flere løsninger som oppfyller hensikten.

2 BEGREPER

GNSS – Global Navigation Satellite Systems

INS – Inertial Navigation System

Dead Reckoning – Klassisk navigasjonmetode for beregning av posisjon ut ifra en kjent posisjon i fortiden [7]

Gazebo – Robot simulator [6]

ROS2 – Robot Operating System 2 []

3 PROSJEKTORGANISASJON

3.1 *Prosjektgruppe*

Navn: Simen Bjerkestrand

Tlf: 90095184

Mail: Simenkbj@stud.ntnu.no

Studentnummer: 522470

3.1.1 Oppgaver for prosjektgruppen - organisering

Prosjektet gjøres av en enkeltperson, dermed har den person ansvar for alle arbeidsoppgaver og organisering av arbeidet.

3.2 *Styringsgruppe (veileder og kontaktperson oppdragsgiver)*

Veiledere

Navn: Erlend Coates

Mail: erlend.coates@ntnu.no

Navn: Eirik Fagerhaug

Mail: eirik.fagerhaug@ntnu.no

Kontaktperson(er) oppdragsgiver

Navn: Nikolai Persen

Mail: nikolai.persen@redrock.no

Navn: Andreas Fjetland

Mail: andreas.fjetland@redrock.no

4 AVTALER

4.1 *Avtale med oppdragsgiver*

Avtaler er vedlagt

4.2 *Arbeidssted og ressurser*

Arbeidsplass på L167 er tildelt studenter for å gjennomføre bacheloroppgave, i tillegg har studenten tilgang til Manulab og AGV 60/90 robot ved avtale. Det kreves ingen spesiell tilgang til arbeidssted eller ressurser utover dette.

4.3 *Gruppenormer – samarbeidsregler – holdninger*

Studenten skal være motivert og villig til å jobbe strukturert og jevnt for å oppnå det ønskede målet. En skal også ha ett åpent sinn, en positiv holdning og optimisme når en møter på problemstillinger. Å holde en tanke på etiske utfordringer og ta forsvarlige beslutninger basert på dette er også svært viktig,

5 PROSJEKTBEKRIVELSE

5.1 *Problemstilling - målsetting – hensikt*

Effekt mål

RedRock.AI begynte hovedsakelig med å jobbe med kraner. Prinsippet var da å hjelpe kundene sine oppnå mindre kostnader med kranføring ved hjelp av automatisering. RedRock.AI har også begynt å jobbe mere med løsninger på kai, og trenger automatiseringsløsninger som skaper økonomisk verdi. Denne oppgaven handler om lokalisering av en UGV som også håndterer tap av GNSS signal ved bruk av «Dead Reckoning». Lokaliseringen kan brukes videre i kombinasjon med andre lokaliseringmetoder for å løse problemet om styring av UGV roboten. Ved å minimere kostnader ved Lokaliseringen, kan da den komplette løsningen bidra til lønnsomhet ved produktet eller tjenesten. Effekt målet er da å komme med en eller flere aktuelle løsninger som både utfyller kravene og som er økonomisk.

Resultat mål

Resultat målet er å komme frem til et resultat som er realistisk, på både teoretisk og praktisk vis. Ved å sammenligne resultatene må rapporten synliggjøre ett resultat som består kravene som er satt og som oppnår den ønskede hensikten. Rapporten skal inneholde godt faglig grunnlag, med en god struktur som gjør det enkelt å forstå innholdet.

Prosess mål

Forfatteren skal være flink til å planlegge og dokumentere framdrift, og dermed raskt oppdage avvik i opprinnelig plan. God kommunikasjonsteknikk med veiledere og oppdragsgiver er også viktig.

5.2 *Krav til løsning eller prosjektresultat – spesifikasjon*

En del av oppgaven er å oppfylle visse krav til ytelsen til lokaliseringen, detaljene rundt disse er enda ikke bestemt av oppdragsgiver. Disse vil komme frem til en senere tid.

5.3 Planlagt framgangsmåte(r) for utviklingsarbeidet – metode(r)

Siden denne oppgaven skal gjøres av ett enkeltindivid er det spesielt viktig å ha en god definert framgangsmetode. Den planlagte prosjektstyringsmetoden er hovedsakelig basert på «kritisk bane» metoden [1]. Med denne metoden er den kronologiske sekvenseringen av oppgaver og den estimerte varigheten for hver oppgave svært viktig. Her er det lov å prioritere oppgaver slik at de oppgavene som er viktigst for oppgavens gjennomføring gjøres først.

Hovedtrekkene til framgangsmåten består av problemanalyse, Kravarbeid, Utforming, Implementasjon og Utpøving. Det er viktig å ikke utsette viktige oppgaver som problemanalyse og kravarbeid da disse skaper grunnlaget for hele oppgaven.

5.4 Informasjonsinnsamling – utført og planlagt

Planlagt informasjonsinnsamling

- ROS2 for beginners [3]
- Principles of GNSS, Inertial, and Multisensor Integrated Navigation systems [2]
- Robot Localization package ROS2 [4]
- Gazebo Tutorials [5]

5.5 Vurdering – analyse av risiko

Muligheten for å realisere ett tilfredsstillende resultat er ikke usannsynlig. Uansett så kan det komme utfordringer ved framgang, som gjør det vanskelig å vise til ett resultat. Dette er bla. noe av det som avgjorde prosjektstyringsmetoden som ble brukt. Det kan sies at det er en viss risiko knyttet den praktiske implementasjonen, hvor utfordringer knyttet dette kan påvirke resultatet i stor grad.

5.6 Hovedaktiviteter i videre arbeid

Week	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19		
Starting	jan 10	jan 17	jan 24	jan 31	feb 7	feb 14	feb 21	feb 28	mar 7	mar 14	mar 21	mar 28	apr 4	apr 11	apr 18	apr 25	mai 2	mai 9	mai 16		
Bacheloroppgave varighet																					
Forprosjekt																					
Innlevering Forprosjektsrapport																					
Avtale med eksternt virksomhet																					
Utarbeiding Bachelor rapport																					
skrive teori																					
litteraturstudium																					
metodologi																					
resultater																					
Diskusjon og konklusjon																					
Implementasjon																					
Simulering																					
Validering av resultater																					
Finskrive rapport																					
levere bacheloroppgave																					

5.7 Framdriftsplan – styring av prosjektet

5.7.1 Hovedplan

- Implementasjon skal gjøres i C++ med ROS2. Testing/Simulering skal i utgangspunktet gjøres ved simulasjon, ved bruk av Gazebo.
- Implementasjonen skal begynnes tidlig (31. jan – 7. feb).
- Om noe skal gjennomføres i praksis/virkeligheten, skal dette avgjøres underveis. Dette skal også bestemmes relativt tidlig (14.feb) slik at eventuelle ressurser kan skaffes tidsnok.

5.7.2 Styringshjelpemidler

Både Framdriftsrapportene og Gantt diagrammet er viktige hjelpemidler for å styre prosjektet, og Planlegge videre arbeid.

5.7.3 Utviklingshjelpemidler

Det er så langt ikke noen planlagte utviklingshjelpemidler.

5.7.4 Intern kontroll – evaluering

Ethvert mål og delmål skal ha ett dokumentert resultat før sin frist. Om kriteriene til resultatene er oppnådd skal vurderes i framdriftsrapporten.

5.8 Beslutninger – beslutningsprosess

Beslutninger skal hovedsakelig handle om avgrensninger i prosjektet. Dersom viktige beslutninger tas skal de dokumenteres i framdriftsrapporten.

6 DOKUMENTASJON

6.1 *Rapporter og tekniske dokumenter*

Framdriftsrapport, Bachelorrapport, og Forprosjektrapport skal gjøres som en del av bacheloroppgaven. Planlagte rutiner skal sørge for at disse gjennomføres tidsnok og med god kvalitet. Det skal også utarbeides avtaler med oppdragsgiver.

7 PLANLAGTE MØTER OG RAPPORTER

7.1 *Møter*

7.1.1 Møter med veiledere

Møte med veiledere gjennomføres hver 14. dag fom. 17.01.2022. Datoene og tidspunktene kan endres underveis. Oppdragsgiver inviteres også til eventuelle møter. Hensikten er å vise fremgang og plan fremover vha. framdriftsrapport.

7.2 *Periodiske rapporter*

7.2.1 Framdriftsrapporter (inkl. milepæl)

Framdriftsrapporter skrives hver 14 dag og fullføres før møtene.

8 PLANLAGT AVVIKSBEHANDLING

Dersom prosjektet ikke går som planlagt mtp. framdrift eller resultater leveres oppgaven som den er. Grundig drøfting og dokumentasjon av evt feilkilder må da inkluderes i oppgaven.

9 UTSTYRSBEHOV/FORUTSETNINGER FOR GJENNOMFØRING

- Prosjektet har ingen spesielle krav til utstyr

10 REFERANSER

[1] Nybegynner tips for prosjektstyring, <https://www.microsoft.com/nb-no/microsoft-365/business-insights-ideas/resources/guide-for-project-management>, [17.01.2022]

[2] Principles of GNSS, Inertial, and Multisensor Integrated Navigation Systems, <https://www.amazon.com/Principles-Multisensor-Integrated-Navigation-Applications/dp/1608070050>

[3] ROS2 for beginners, <https://www.udemy.com/course/ros2-for-beginners/> [19.01.2022]

[4] Robot_localization wiki, http://docs.ros.org/en/melodic/api/robot_localization/html/index.html [19.01.2022]

[5] Gazebo Tutorials, <https://gazebosim.org/tutorials> [19.01.2022]

[6] Gazebo, <http://gazebosim.org/> [23.01.2022]

[7] Dead Reckoning, https://snl.no/dead_reckoning [23.01.2022]

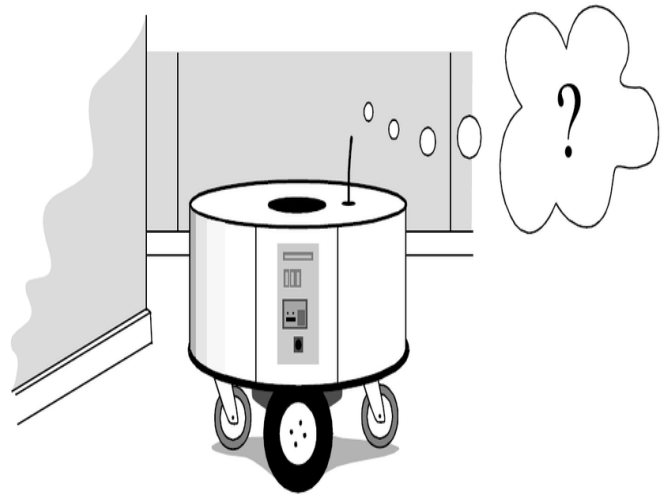
B Robot Localization proposed bachelor or master thesis

Short description

An important aspect within autonomy is localization. Localization deals with finding the position of an agent (e.g., mobile robot) within the environment it is in.

There exists a large number of sensors that can be used to improve the overall performance of localization. However, a common issue with localization is temporary or permanent loss of sensor(s), for example in tunnels. In such cases, the position can drift over time.

Dead Reckoning (DR) is a popular tool for compensating loss of sensors such as GNSS to estimate the position. This approach can use sensors such as gyroscopes, accelerometers and wheel encoders to estimate position.



Keywords

- Cybernetics
- Mechatronics
- ROS (ROS2 preferred)
- Odometry
- Sensor Fusion
- Programming (C++, Python)
- Simulation
- State estimation
- Localization
- Robotics

Project Description

The aim of this thesis is to evaluate the performance of localization using different sensors on an acherman based vehicle. The student(s) shall:

Perform a literary review of the state-of-the-art localization models. This includes different sensors and filters that are used to estimate position and orientation.

Perform tests on a simulated mobile robot and demonstrate its localization capabilities. This includes adding individual sensors to the robot, and evaluating the performance of that specific sensor. When does the position start to drift? How much and how fast does the position drift? Evaluate how sensor fusion affects the overall performance. Which sensors must be used to maintain acceptable performance?

Evaluate how dead reckoning affects localization in the absence of GNSS. How many sensors (e.g., gyroscope, accelerator, encoder) are needed to maintain satisfactory performance during DR? Using the most amount of sensors to achieve the most robust and accurate localization model is *not* the focus in this thesis. It is more interesting to know how *few* sensors are needed, while still maintaining acceptable performance. It is recommended that the students use ROS2 on Ubuntu 20.04.

The use of any visual based localization is beyond the scope of this thesis.

RedRock.AI is built on more than ten years of expertise in lifting & handling by Red Rock and is now a part of the Ocean Infinity Group, a world-leading marine robotics company.

We are driven by creating an impact on the future of autonomous solutions, our mission is building autonomous environments that enable any equipment, regardless of brand and function to cooperate seamlessly. In RedRock.AI we are working on state-of-the-art control algorithms, sensor processing, object detection, navigation and advanced simulations for autonomous vehicles. We are a group of devoted R&D engineers working closely together to solve the most challenging task related to autonomy. We care about the sustainability of our projects and we carry our production out with the lowest environmental impact.

Company Contact Information

Full name	E-mail address	Phone number
Andreas K Fjetland	andreas.fjetland@redrock.no	934 25 966
Nikolai Olav Persen	nikolai.persen@redrock.no	915 95 171

C Gazebo URDF

```
<robot name=placeholder>

...
<!-- This part of the code is protected by non-disclosure agreement
--!>
<!-- attach the imu_link to a ackerman based vehicle with a joint --!>
...

<link name="$(arg link_namespace)imu_link">
  <visual>
    <geometry>
      <box size="0.1 0.1 0.1"/>
    </geometry>
  </visual>
</link>

  <!-- This plugin drives the ackerman vehicle with the commands
  from cmd_vel topic-->
<gazebo>
  <plugin name="gazebo_ros_ackermann_drive" filename="
  libgazebo_ros_ackermann_drive.so">
    <ros>
      <namespace>simulation</namespace>
      <remapping>cmd_vel:=cmd_vel</remapping>
      <remapping>distance:=distance</remapping>
    </ros>
    <update_rate>100.0</update_rate>
    <!-- wheels -->
```

```

<front_left_joint>$(arg link_namespace) front_left_joint </
  front_left_joint >
<front_right_joint>$(arg link_namespace) front_right_joint </
  front_right_joint >
<rear_left_joint>$(arg link_namespace) rear_left_joint </
  rear_left_joint >
<rear_right_joint>$(arg link_namespace) rear_right_joint </
  rear_right_joint >
<left_steering_joint>$(arg link_namespace) steer_left_joint </
  left_steering_joint >
<right_steering_joint>$(arg link_namespace) steer_right_joint </
  right_steering_joint >
<steering_wheel_joint>$(arg link_namespace) steering_joint </
  steering_wheel_joint >
<!--      steer and steering angle are in rad      -->
<max_steer>0.785398</max_steer>
<max_steering_angle>7.854</max_steering_angle>
<!--      speed is in m/s      -->
<max_speed>17.8816</max_speed>

<!-- Feel free to tune these values -->
<left_steering_pid_gain>955000 65 5</left_steering_pid_gain>
<left_steering_i_range>0 0</left_steering_i_range>
<right_steering_pid_gain>955000 65 5</right_steering_pid_gain>
<right_steering_i_range>0 0</right_steering_i_range>
<linear_velocity_pid_gain>5000 45 3</linear_velocity_pid_gain>
<linear_velocity_i_range>0 0</linear_velocity_i_range>
<!-- output -->
<publish_odom>>false</publish_odom> <!-- this is the odometry
  that is too accurate --!>

```

```

    <publish_odom_tf>false </publish_odom_tf>
    <publish_wheel_tf>false </publish_wheel_tf>
    <publish_distance>false </publish_distance>
    <odometry_frame>$(arg link_namespace)odom</odometry_frame>
    <robot_base_frame>simulation_base_link</robot_base_frame>
  </plugin>
</gazebo>

<!-- This plugin publishes wheel odometry -->
<gazebo>
  <plugin name="ackerman_odometry_plugin" filename="libackerman_odom
    .so">
    <ros>
      <remapping>odom:= / odometry / wheel</remapping>
    </ros>
    <left_joint_name>$(arg link_namespace) rear_left_joint </
      left_joint_name >
    <right_joint_name>$(arg link_namespace) front_right_joint </
      right_joint_name >
    <use_imu_heading>true</use_imu_heading>
    <wheels_radius>0.2615475</wheels_radius>
    <wheels_separation>1.990400</wheels_separation>

  </plugin>
</gazebo>

<!-- This plugin publishes inertial odometry -->
<gazebo reference="$(arg link_namespace)imu_link">
  <gravity>true</gravity>
  <sensor name="imu_sensor" type="imu">

```

```

<always_on>true</always_on>
<update_rate>100</update_rate>
<visualize>true</visualize>
<topic>__default_topic__</topic>
<plugin filename="libgazebo_ros_imu_sensor.so" name="
  imu_plugin">
  <ros>
    <remapping>~/out:=imu</remapping>
  </ros>
  <topicName>imu</topicName>
  <bodyName>$(arg link_namespace) imu_link</bodyName>
  <updateRateHZ>100</updateRateHZ>
  <gaussianNoise>0.01</gaussianNoise>
  <xyzOffset>0 0 0</xyzOffset>
  <rpyOffset>0 0 0</rpyOffset>
  <frameName>$(arg link_namespace) base_link</frameName>
  <initial_orientation_as_reference>>false</
    initial_orientation_as_reference>
</plugin>
<pose>0 0 0 0 0 0</pose>
</sensor>
</gazebo>

<!-- This plugin publishes true world position -->
<gazebo>
  <plugin name="gazebo_ros_p3d" filename="libgazebo_ros_p3d.so">
    <ros>
      <namespace>simulation</namespace>
      <remapping>odom:=p3d</remapping>
    </ros>

```

```
<body_name>$(arg link_namespace) base_footprint </body_name>
<frame_name>world</frame_name>
<update_rate>60</update_rate>
<xyz_offset>0 0 0</xyz_offset>
<rpy_offset>0 0 0</rpy_offset>
<gaussian_noise>0.0</gaussian_noise>
</plugin>
</gazebo>
</robot>
```

D Ackerman Odometry Plugin Source Code

```
#include "ackerman_odom.h"
#include <math.h>
#include <gazebo_ros/conversions/builtin_interfaces.hpp>
#include <gazebo/common/Time.hh>
#include <rclcpp/qos_overriding_options.hpp>
#include <mw/types.h>

#define tau 6.28318530718

void AckermanOdom::Load(gazebo::physics::ModelPtr _model, sdf::ElementPtr
    _sdf)
{
    // Here I extract the wheel joint names, and other parameters that I
    // received from urdf file
    std::string left_joint_name = _sdf->GetElement("left_joint_name")->Get<
        std::string>();
    std::string right_joint_name = _sdf->GetElement("right_joint_name")->Get<
        std::string>();
    use_imu_heading = _sdf->GetElement("use_imu_heading")->Get<bool>();
    wheels_separation = _sdf->GetElement("wheels_separation")->Get<double>()
        ;
    wheels_radius = _sdf->GetElement("wheels_radius")->Get<double>();

    // Here, I get the link defined as "link_name" from the actual robot
    // model
    left_joint_ = _model->GetJoint(left_joint_name);
    right_joint_ = _model->GetJoint(right_joint_name);
```



```
// Initialize values
time = _model->GetWorld()->SimTime();
model_ = _model;
x = 0.0;
y = 0.0;
dx = 0.0;
dy = 0.0;
theta = 0.0;
dtheta = 0.0;
theta_prev = 0;

odom_.twist.covariance = {0.01, 0, 0, 0, 0, 0,
                          0, 0.01, 0, 0, 0, 0,
                          0, 0, 0.01, 0, 0, 0,
                          0, 0, 0, 0.01, 0, 0,
                          0, 0, 0, 0, 0.01, 0,
                          0, 0, 0, 0, 0, 0.05};

// Here, I will get all the ros related stuff from the urdf file that is
// within the tags <ros> such as namespace and remapping of topic name
node_ = gazebo_ros::Node::Get(_sdf);

if (use_imu_heading)
{
    // since the imu topic has non-matching QoS settings, we have to
    // change them manually
    auto qos_profile = rclcpp::QoS(rclcpp::KeepLast(10),
        rmw_qos_profile_sensor_data);

    // Here a subscriber is created to be used to fetch imu heading
```

```

heading_sub_ = node_->create_subscription<sensor_msgs::msg::Imu>("imu
    ", qos_profile,

                                                                    std::
                                                                    bind
                                                                    (&
                                                                    AckermanOdom
                                                                    ::
                                                                    Callback
                                                                    ,
                                                                    this
                                                                    ,
                                                                    std
                                                                    ::
                                                                    placeholders
                                                                    ::
                                                                    _1
                                                                    ))
                                                                    ;
}
else
{
    use_imu_heading = false;
}

// Setting up the publisher with default topic name "odom"
odom_pub_ = node_->create_publisher<nav_msgs::msg::Odometry>("odom", 10)
    ;

// Bind the OnUpdate function to be called on every gazebo simulation
iteration

```

```

update_connection_ = gazebo::event::Events::ConnectWorldUpdateBegin( std
    ::bind(&AckermanOdom::OnUpdate,
                                                    this
                                                    )
                                                    )
;

}

// This will run for every simulation iteration
void AckermanOdom::OnUpdate()
{
    std::lock_guard<std::mutex> scoped_lock(lock_);

    // I get the x-position of the link here
    left_vel = left_joint_->GetVelocity(1);
    right_vel = right_joint_->GetVelocity(1);

    // OBS! kanskje gazbo tid for rsaker hopp i signalet
    time = model_->GetWorld()->SimTime();
    dt = time.Double() - prev_time.Double();
    prev_time = model_->GetWorld()->SimTime();

    // Calculate the arc lengths of the left and right hand side
    respectively
    arc_length_right = 2 * wheels_radius * right_vel * dt; // 2*pi*r*
    rotations right
    arc_length_left = 2 * wheels_radius * left_vel * dt; // 2*pi*r*
    rotations left

```

```
// Calculate the radius of the curvature from the center of the vehicle.  
    This can be seen as constant each frame as the timestep is quite  
    small.  
curvature_radius = wheels_separation / 2 * (arc_length_right +  
    arc_length_left) / (arc_length_right - arc_length_left);  
  
// calculate the change in position of the vehicle  
dtheta = (arc_length_right - arc_length_left) / 1.990400;  
dx = curvature_radius * (cosf(theta) * sinf(dtheta) - (sinf(theta) * (1  
    - cosf(dtheta))));  
dy = curvature_radius * (sinf(theta) * sinf(dtheta) - (cosf(theta) * (1  
    - cosf(dtheta))));  
  
// integrate the change to find the total change in position  
if (!use_imu_heading)  
{  
    theta = theta + dtheta;  
}  
position.x = position.x + dx;  
position.y = position.y + dy;  
  
// calculate the linear and angular velocities needed  
linear_velocity.x = dx / dt;  
linear_velocity.y = dy / dt;  
angular_velocity.z = dtheta / dt;  
  
// reassign to odom container  
odom_.twist.twist.linear = linear_velocity;  
odom_.twist.twist.angular = angular_velocity;  
odom_.header.stamp = gazebo_ros::Convert<builtin_interfaces::msg::Time>(
```

```
    time);
    odom_.header.frame_id = "odom";
    odom_.child_frame_id = "simulation_base_footprint";

    // Publish the data to the "position" topic
    odom_pub_->publish(odom_);
}

void AckermanOdom::Callback(sensor_msgs::msg::Imu imu)
{
    // calculate and save the heading from imu quaternion
    auto q = imu.orientation;
    double siny_cosp = 2 * (q.w * q.z + q.x * q.y);
    double cosy_cosp = 1 - 2 * (q.y * q.y + q.z * q.z);
    theta = 0.5*std::atan2(siny_cosp, cosy_cosp);
    odom_.pose.pose.orientation.z = q.z;
    odom_.pose.pose.orientation.w = q.w;
    odom_.pose.pose.orientation.x = q.x;
    odom_.pose.pose.orientation.y = q.y;
}

// Remember to add this macro at the end of your plugins, and use the
    class name as input! Otherwise it will crash
GZ_REGISTER_MODEL_PLUGIN(AckermanOdom)
```

E Kalman Filter Parameter File(YAML)

```
### ekf config file ###
ekf_filter_node:
  ros__parameters:

    frequency: 30.0

    two_d_mode: false

    permit_corrected_publication: false

    publish_acceleration: false

    reset_on_time_jump: true

    dynamic_process_noise_covariance: true

    map_frame: map          # Defaults to "map" if unspecified
    odom_frame: odom        # Defaults to "odom" if unspecified
    base_link_frame: simulation_base_footprint # Defaults to "
      base_link" if unspecified
    world_frame: odom       # Defaults to the value of odom_frame
      if unspecified

    odom0: /odometry/wheel
    odom0_config: [false, false, false,
                  false, false, true,
                  true, true, true,
                  false, false, false,
```

```
        false , false , false ]
odom0_queue_size: 1
odom0_nodelay: false
odom0_differential: false
odom0_relative: true

imu0: /imu
imu0_config: [false , false , false ,
              true , true , true ,
              false , false , false ,
              false , false , false ,
              false , false , false ]
imu0_nodelay: false
imu0_differential: false
imu0_relative: false
imu0_queue_size: 5
imu0_remove_gravitational_acceleration: true

use_control: false

stamped_control: false

control_timeout: 0.2

control_config: [true , false , false , false , false , true]

acceleration_limits: [2.5, 0.0, 0.0, 0.0, 0.0, 3.4]

deceleration_limits: [2.5, 0.0, 0.0, 0.0, 0.0, 4.5]
```

acceleration_gains: [0.8, 0.0, 0.0, 0.0, 0.0, 0.9]

deceleration_gains: [1.0, 0.0, 0.0, 0.0, 0.0, 1.0]

process_noise_covariance: [0.05, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0,
0.0, 0.05, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0,
0.0,
0.0, 0.0, 0.06, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0,
0.0,
0.0, 0.0, 0.0, 0.03, 0.0,
0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0,
0.0,
0.0, 0.0, 0.0, 0.0, 0.03,
0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0,
0.0,
0.0, 0.0, 0.0, 0.0, 0.0,
0.06, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0,
0.0,
0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.005, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0,

0.0,
0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.005, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0,
0.0,
0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.04,
0.0, 0.0, 0.0, 0.0, 0.0,
0.0,
0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0,
0.01, 0.0, 0.0, 0.0,
0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0,
0.0, 0.01, 0.0, 0.0,
0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.005, 0.0,
0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.01,
0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0,
0.01, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0,

```

0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0,
0.0, 0.015]

```

```

initial_estimate_covariance: [1e-9, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0,

```

```

0.0, 1e-9, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0,
0.0, 0.0, 0.0,
0.0, 0.0, 1e-9, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0,
0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 1e-9, 0.0,
0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 1e
-9, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0,
0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0,
1e-9, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0,
0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 1e-9, 0.0, 0.0,
0.0, 0.0, 0.0,

```

0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 1e-9, 0.0,
0.0, 0.0, 0.0,
0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 1e-9,
0.0, 0.0, 0.0, 0.0,
0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0,
1e-9, 0.0, 0.0, 0.0,
0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0,
0.0, 1e-9, 0.0, 0.0,
0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 1e-9, 0.0,
0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 1e
-9, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0,
0.0, 1e-9, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0,

0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0,
0.0, 0.0, 1e-9]

initial_state: [138.76, -171.48, -0.14,
0.0, 0.0, 0.03,
0.0, 0.0, 0.0,
0.0, 0.0, 0.0,
0.0, 0.0, 0.0]

