Australian National University

Implementation of Pose Estimation Algorithms on the Clearpath Jackal UGV

Alexander Ollman

Supervised by
A/Prof Jochen Trumpf
Mr Jack Henderson

A thesis submitted in part fulfilment of the degree

Bachelor of Engineering (Honours)

College of Engineering and Computer Science
The Australian National University

November 2019

This thesis contains no material which has been accepted for the award of any other degree or diploma in any university. To the best of the author's knowledge, it contains no material previously published or written by another person, except where due reference is made in the text.

Alexander Ollman
8th November 2019

Acknowledgements

I'd like to thank my primary supervisor, Jochen Trumpf, for approaching me with a project that allowed me to dive head-first into the wonderful world of robotics research. His guidance, experience and encouragement throughout this project has made it an enjoyable, challenging and rewarding one. I am incredibly appreciative for the opportunity to have undertaken a project that has given me such a passion for a potential career path. I'd also like to thank my secondary supervisor, Jack Henderson, for without whom several hurdles of this project would have taken far longer to overcome. Your insight, advice and willingness to indulge me in the many questions I had throughout this project has been incredibly appreciated. I wish you all the best for the remainder of your doctorate and the Jackal Project.

I would also like to acknowledge all my family and close friends for their patience, love and support throughout my time at university. I feel privileged to have got to know so many wonderful people throughout my time at ANU and I thank you for all the support over the years.

To my parents, Paulina, David, Stephen and Maria, thank you for your unwavering love, wisdom and encouragement and for instilling in me the importance of integrity, hard work and a love for what you do. I will be forever grateful that your support was able to provide me the platform to complete a project such as this.

Lastly, I would like to acknowledge my grandmother, Nona. I don't think I can ever express the required gratitude and appreciation for all you have done for me throughout my life that has led to the completion of my degree. Your continual reminder of how I am forever "burning the candle at both ends" has been well encapsulated by this paper.

Abstract

An attitude estimation filter, known as the Geometric Approximate Minimum-Energy (GAME) filter was proposed by Zamani, Trumpf and Mahony of the ANU in 2012. The GAME Filter has been shown to have a lower root-mean-square estimation error than the industry standard attitude filter, the Multiplicative Extended Kalman Filter (MEKF), in a variety of simulated applications. The next step in developing the GAME Filter is to validate its simulation performance on a physical platform. Having an implementation of the GAME Filter on a physical platform such as the Jackal UGV will allow researchers to observe measurement noises that may not have been accounted for in simulation and begin work on the application of the filter to perform collaborative localization.

The Defence, Science and Technology Group (DTSG) have provided the ANU with a mobile robot development platform: a Clearpath Jackal UGV. The primary goal of this individual project was to scope how the GAME Filter would be implemented into ROS and subsequently onto the Jackal UGV. This report provides a breakdown of how filtering algorithms are currently implemented on the Jackal UGV and the methodology for implementing the GAME Filter onto the Jackal UGV.

Table of Contents

List of Figures

Glossary of Terms

ROS - Robot Operating System
EKF - Extended Kalman Filter
UKF - Unscented Kalman Filter
MEKF - Multiplicative Extended Kalman Filter
GAME - Geometric Approximate Minimum-Energy
DTSG - The Defence, Science and Technology Group
ANU - The Australian National University
UGV - Unmanned Ground Vehicle
UAV - Unmanned Aerial Vehicle
IMU - Inertial Measurement Unit
URDF - Unified Robot Description Language. This is an XML dialect for representing a robot model.
Jackal - Shorthand reference to a Clearpath Jackal UGV
Jackal Project - In reference to the partnership between the DTSG and the ANU, in which the ANU has been lent a Clearpath Jackal UGV to aid in the research and development of the GAME Filter.

Chapter 1 Introduction

This project primarily focussed on the theory of mobile robot localisation. Localisation is the process by which a robot determines its location with respect to a known environment. A lack of sufficient localization ability will likely result in the robot not being able to accurately estimate its position or orientation, known as its pose, with respect to an environment or task-specific coordinate frame. Most if not all applications requiring mobile robots will rely on a robot's ability to estimate its pose, and it is for this reason that localisation and pose estimation algorithms have been at the forefront of robotics research for several decades [1].

Difficulty arises in pose estimation when several stochastic processes add to the modelled kinematic motion of the robot. As a result, the modelled motion of the robot based on a known input and known dimensions becomes a random noise process. To correct for this, measurements are taken by the robot of environmental landmarks with known positions in a defined coordinate space. These measurements are also susceptible to noise (interference, latency, etc) and are thereby treated with a degree of uncertainty. This uncertainty is generally modelled as a noise process. As such, the assumption can be made that the only information available to any mobile robot are stochastic random variables.

Currently, the most established way to estimate pose from modelled random variables is using a stochastic filter. In robotics, the objective of stochastic filtering is reducing the uncertainty of an estimation of the state using partial proprioceptive and exteroceptive observations. The development of filters aims to improve the accuracy of estimation in specific mobile robotics applications, such as 3D attitude estimation and environmental mapping. More specifically, having more robust filtering algorithms allow systems to achieve the desired localization accuracy even with cheaper, noisier sensor measurements. Naturally this is a sought outcome, as having cheaper systems that still perform to the system requirements is highly desired for any physical engineering project.

The research aim of this project is to determine how to implement a second-order minimum-energy filter proposed by ANU researchers, known as the Geometric Approximate Minimum-Energy (GAME) Filter onto a physical robot. Currently, the GAME Filter has only been tested and validated in simulation with specific parameters and applications and as such there are no published works related to a physical implementation of GAME Filter. The next step in the filter's development is to therefore verify that the performance achieved in simulations can be replicated on a physical robot. ANU has been provided with a Clearpath Jackal UGV by the DTSG to aid in both the validation and development of the GAME Filter.

The primary stakeholders of this project are the partnership representatives of both the DTSG and the ANU: Dr Behzad "Mohammad" Zamani and A/Prof Jochen Trumpf, respectively. The research partnership is a government-funded initiative to further develop the GAME Filter. The respective research groups of each organisation, including ANU PhD candidate Jack Henderson, will benefit from the outputs of this report, which directly aid their ability to implement the GAME Filter onto physical platforms for testing and development.

Chapter 1 introduces the project by outlining the project context, research aims, stakeholders, and provides an overview of the report. Chapter 2 introduces the higher-level theory of this project, providing a conceptual foundation for the reviewed literature presented in Chapter 3, which summarizes the key underlying concepts necessary to complete this project. The concepts are accompanied by the supporting literature that aided in the author's understanding of them. This chapter also introduces the various filters and filtering theory that have led to the development of the GAME Filter. Chapter 4 describes the methodology of the project that lead to the project outputs, including the derivation of the necessary equations for an Extended Kalman Filter implementation specifically for the Jackal UGV. Chapter 5 summarizes the outcomes of the project, including the performance results of a custom filter implementation on the Jackal and the proposed methodology for implementing the GAME Filter into ROS for the Jackal. Chapter 6 concludes the main body of the report and outlines recommendations for future works.

Chapter 2 Background

The following chapter provides the theoretical context and motivation for this project. The information presented in this chapter is a higher-level introduction to the concepts presented in Chapter 3.

2.1 Robot Awareness

When solving problems in the field of robotics, one of the primary complicating factors is that robots do not sense the world as we do. As such, seemingly simple hurdles often require significant amounts of research in order to comprehend the root of these issues and resolve them. Robotic vision is the study of how to translate the physical world into a form that can be computationally processed in such a way that robotic systems can "see". One of the simpler cases involves a camera mounted onto a robot and a "vision" algorithm which processes the received images in such a way that is specific to a task at hand. Other forms of "vision" include a variety of sensors, such as laser (LiDAR), ultrasonic distance sensors or an inertial measurement unit (IMU), that can be used to detect nearby objects, surrounding environments and/or the movement of the robot; all of which aid the robot to better understand the world around it.

2.2 Pose Estimation

Pose estimation is the determination of the position and orientation of an object (known as its "pose") with respect to a reference frame. A robot's pose is most commonly estimated using proprioceptive motion sensors which capture the change in its translation and rotation. A common way of estimating the pose of a mobile robot is to model a robot's *odometry* over time [1]. Odometry is the use of data from sensors that capture the motion of the robot, such as motor encoders and inertial measurement units, to estimate the change in position over time. The odometry can be integrated over time using the forward Euler method to determine the pose of a robot.

2.3 Localization

Localization is the process by which intrinsic and extrinsic information is used to determine where a mobile robot is located with respect to a known environment. Huang and Dissanayake [1] describe localization as "one of the most fundamental competencies required by an autonomous robot." The knowledge of the robot's own location is an essential precursor to making decisions about future actions. In a typical robot localization scenario, the

environment is known, and the robot is equipped with sensors that observe the environment as well as its own motion. The localization problem then becomes one of estimating the robot position and orientation within the environment using information gathered from these sensors.

2.4 Noise and Uncertainty

Any robot is a system that can be described as one that combines sensing, actuation, computation and communication. As stated by Correll [2], every subsystem of a robot is subject to some degree of uncertainty, even if those uncertainties can be considered negligible. Some examples of these uncertainties include gyroscope "drift" and bias, electrical noise interference, communication disturbances and dropouts, as well as sensor interference and miscalibration. Succinctly, no measurement can ever be considered to represent the true state of the variable it is measuring: there will always be a degree of uncertainty.

The problem introduced by noise into systems is that with each measurement, more uncertainty is introduced into the system. Without effective modelling and filtering of these noisy processes, any estimation of a system's state is likely to continually diverge further from its true state over time. Using statistical estimation algorithms, such as *filters*, the sensor can then be modelled as a noisy process of the true value and the true value can be more accurately estimated. These algorithms more commonly take the form of some derivation of the *Bayes Filter* or the *Kalman Filter* [3].

2.5 Filter Application

Filters are prominent in most electronic systems and can be applied in countless ways and configurations. Their primary application is to reduce the noise and uncertainty introduced into a system by sensor measurements through recursive statistical methods to reduce the covariance between the variables that represent the state of a system.
In the case of robotic pose estimation and localization, these state variables represent the coordinates and orientation in space relative to a known reference frame. Therefore, it is highly desired to have filtering algorithms that perform this reduction accurately and with computational efficiency, particularly when it comes to the field of mobile robotics. Filtering algorithms with greater accuracy and efficiency performance allow for lower end systems with cheaper sensors and processors to meet more real-world requirements of mobile robotic applications. For this reason, improving the performance filtering algorithms has been at the forefront of robotics theory research for several decades [1][2].

Chapter 3 Literature Review

This chapter summarizes the literature that has been most relevant to the author's understanding of the required theory and introduces the scope of this project with respect to the greater Jackal Project. Section 3.1 summarizes the literature that aided in the author's understanding of necessary filter and kinematics theory as well as an introduction to ROS. Section 3.2 describes the greater picture of the Jackal Project and introduces the scope of this individual project, including how this project fits into the future development of the Jackal Project. This section includes summarizes the previous work on the simulated applications of the GAME Filter, performed by Zamani, Trumpf and Mahony, as well as describing the motivation and importance for this project to aid in the eventual outcome of testing the GAME Filter on a physical platform.

3.1 Reference Literature

The primary reference literature came in the form of online documents and research papers related to the Kalman Filter and the Extended Kalman Filter. These included the derivation of these algorithms such that the author could create their own implementation and further cement their own understanding. Additionally, the reference literature presented in this section describes the theory of mobile robot motion without considering the forces involved (kinematics), as well as the literature which introduces the software environment on the Jackal: Robot Operating System (ROS).

3.1.1 Kalman Filter

The Kalman filter is an algorithm first derived by Rudolf E. Kálmán in 1960, and is used to estimate state variables of a system based on noisy measurement information. The filter computes this estimation by taking the known prior system state, performing a prediction of the next state of the system and its covariance (Eq 1.1, 1.2), and updates this prediction using incoming measurements (Eq 2.1, 2.2). Depending on "noisiness" (the quality and variance) of the measurement data, the Kalman filter will weigh how much this new incoming data will influence the state prediction. The weighting, known as the *Kalman Gain*, converges as the filter models the variance of incoming measurements. As the algorithm does not need to have knowledge of any states except for the one before the current time step, it is recursive and very memory efficient [3]. The predict and update step equations [1][2] for the Kalman Filter are:

Predict Step

$$\hat{\mathbf{x}}_{k|k-1} = \mathbf{F}_k \hat{\mathbf{x}}_{k-1|k-1} + \mathbf{B}_k \mathbf{u}_{k-1} \tag{1a}$$

$$\mathbf{P}_{k|k-1} = \mathbf{F}_k \mathbf{P}_{k-1|k-1} \mathbf{F}_k^\mathsf{T} + \mathbf{Q}_k \tag{1b}$$

Update Step

$$\tilde{\mathbf{y}}_k = \mathbf{z}_k - \mathbf{H}_k \hat{\mathbf{x}}_{k|k-1} \tag{2a}$$

$$\mathbf{S}_k = \mathbf{H}_k \mathbf{P}_{k|k-1} \mathbf{H}_k^\mathsf{T} + \mathbf{R}_k \tag{2a}$$

$$\mathbf{K}_k = \mathbf{P}_{k|k-1} \mathbf{H}_k^\mathsf{T} \mathbf{S}_k^{-1} \tag{2c}$$

$$\hat{\mathbf{x}}_{k|k} = \hat{\mathbf{x}}_{k|k-1} + \mathbf{K}_k \tilde{\mathbf{y}}_k \tag{2d}$$

$$\mathbf{P}_{k|k} = \left(\mathbf{I} - \mathbf{K}_k \mathbf{H}_k\right) \mathbf{P}_{k|k-1} \tag{2e}$$

The standard Kalman filter is a linear, discrete-time, finite-dimensional system. The state of the system is represented by the vector at the current time step *k*, $\boldsymbol{x_k}$. The state transition matrix, which is the model for how the state will change with each time step, is given by $\boldsymbol{F_k}$. The influence the system inputs $\boldsymbol{u_k}$ have on each of the state variables is defined by the input control matrix $\boldsymbol{B_k}$. The incoming measurement/observation $\boldsymbol{z_k}$ is then weighted against the estimated model $\hat{\mathbf{x}}_{\mathbf{k|k-1}}$ multiplied by the matrix $\boldsymbol{H_k}$, which describes the mapping between the measurement $\boldsymbol{z_k}$ and the state $\boldsymbol{x_k}$. The difference between the measurement and the predicted state estimate (2a) is known as the *innovation*.

The amount of influence either the predicted step or the external measurement has on the system state estimate is determined by the Kalman Gain, $\boldsymbol{K_k}$. The uncertainty, or *noise*, associated with both the state transition model and the observed measurement are represented by the zero-mean Gaussian processes $\boldsymbol{w_t}$ and $\boldsymbol{v_t}$, respectively:

$$\mathbf{v}_k \sim \mathcal{N}\left(0, \mathbf{R}_k\right) \tag{3a}$$

$$\mathbf{w}_k \sim \mathcal{N}\left(0, \mathbf{Q}_k\right) \tag{3b}$$

Where $\boldsymbol{Q_t}$ and $\boldsymbol{R_t}$ are the process and measurement noise covariance matrices, respectively.

3.1.2 Non-Linear (Extended) Kalman Filter

State transitions and measurements are rarely linear in practice. For example, one could not apply the linear Kalman Filter to a system measuring the horizontal displacement of a pendulum, as its motion is harmonic and thereby nonlinear. In order to use a Kalman Filter to estimate the state of a nonlinear system, the state equations must be linearized at each time step. This is performed by the Extended Kalman Filter (EKF), which calculates a Gaussian approximation of the state's probability function by linearizing the system about its current mean (state estimate) using a Taylor series expansion at the mean.

Extended Kalman Filter (equivalent to 1a, 2a)

$$\hat{\mathbf{x}}_{k|k-1} = f(\hat{\mathbf{x}}_{k-1|k-1}, \mathbf{u}_k) \tag{4}$$

$$\tilde{\mathbf{y}}_k = \mathbf{z}_k - h(\hat{\mathbf{x}}_{k|k-1}) \tag{5}$$

Unlike the linear case, the state estimate $\hat{\mathbf{x}}_{\mathbf{k|k-1}}$ is given by the non-linear function $f(x_k, u_k)$ and depends on the current state and the inputs to the system. Whilst the state prediction and output can be calculated using the non-linear model, the system error covariances require the state equations to be linearized. The linearized state and output functions are given by their Jacobians, $\boldsymbol{F_k}$ and $\boldsymbol{H_k}$, which are linearized using a first-order Taylor expansion around the current state mean (*k-1*).

$$\boldsymbol{F}_k = \left.\frac{\partial f}{\partial \boldsymbol{x}}\right|_{\hat{\boldsymbol{x}}_{k-1|k-1}, \boldsymbol{u}_k} \tag{6a}$$

$$\boldsymbol{H}_k = \left.\frac{\partial h}{\partial \boldsymbol{x}}\right|_{\hat{\boldsymbol{x}}_{k|k-1}} \tag{6b}$$

The Taylor expansion linearization around the state mean within these Jacobians is only an approximation of the state. This approximation, while small in variance from the true state estimate, introduces another error to the system. The severity of this linearization error will differ depending on the implementation of the model and how accurately the model can be linearized. In the extreme cases where the process and measurement models have high non-linearities and/or high amounts of process noise, the EKF performs poorly and tends to diverge from the true state [5]. For this reason, there is an entire field of research dedicated to developing extensions of the EKF which better account for linearization error and other more

specific errors. Examples of these extensions are the Unscented Kalman Filter and the Multiplicative EKF (MEKF) [5].

3.1.3 The Multiplicative Extended Kalman Filter

The MEKF is considered the industry standard filter for the majority of modern robotics applications, in particular for aerial and spacecraft attitude estimation (systems that are represented in $\mathbf{R^3}$) [10]. In such an application, the attitude (the relative orientation of a vehicle with respect to some reference, usually the ground) of an aerial vehicle is represented by unit quaternions. Quaternions are a mathematical notation used to represent orientations and rotations of objects in three dimensions. The primary advantages to using quaternions over Euler angles to represent attitude is that they can be composed without much difficulty and avoid the problem of gimbal lock[1]. [7]

The derivation of the MEKF resulted from the fact that the "linearization" approach of the EKF does not respect the geometry of the quaternion space. According to Widodo and Wada [8], the standard linear correction terms do not preserve the norm of the quaternion and the derivation of a linear innovation term does not make sense to calculate for quaternions. As such, modifications were made to the EKF to respect the geometry of the quaternion space, by using the quaternion estimation for a multiplicative correction term which preserves the unit norm and changes the way the innovation term was derived. [8]. As a result, the MEKF effectively addresses the quaternion and attitude estimation limitations of the EKF.

[1]Gimbal lock is the loss of one degree of freedom in a three-dimensional, three-plane Euler angle space that occurs when the axis of two of the three gimbals are driven into a parallel configuration, that is "locked" to the same plane. When the system is in this state, the two gimbals represent the same axis rotation, "locking" the system into rotation in a two-dimensional space [5].

3.1.4 Minimum-Energy Filtering

The "energy" of a system refers to the total magnitude of variation between an estimated state trajectory and the measurements made over a time interval of interest. The *cost* function *J(t; ...)* is a measure of the aggregate energy stored in the unknown initialised state estimate and measurement signals [9]. Given measurements and velocity inputs made over all time, the goal of minimum-energy filtering is to obtain an estimate ($\hat{X}(t)$) of the true state ($X(t)$) by minimizing the cost function. This is achieved by seeking a combination of an initial state and a set of velocity inputs (with their respective modelled noise processes) over all time that are compatible with the measurements and velocity inputs, such that the state estimate $\hat{X}(t)$ still

fulfils the system's kinematic constraints. By minimizing the cost function, we produce an initial state and noise models that contain minimum collective energy. In an attitude minimum-energy filter, such a cost function is defined for attitude estimation error [9]. A Value function (7) is also defined [10], which is the minimum cost function given the control input at a given time.

$$V(R,t) := \min_{\delta|_{[0,t]}} J(t;R_0,\delta|_{[0,t]}) \tag{7}$$

Where *J(t; ...)* is the cost function, *R* is an SO(3) state signal and δ is the input measurement error. When the derivative of the Value function equals zero, we have located a local minima amongst the energy of the state. At this point, the state estimate produced by the filter is the closest representation of the true state.

3.1.5 The GAME Filter

The Geometric Approximate Minimum Energy (GAME) filter is a second order attitude filter for estimation on the special orthogonal rotation group in three dimensions (SO(3)). It was first proposed by Zamani [10] and has been further tested, developed and modified in conjunction with Jochen Trumpf and Robert Mahony [10, 11, 23-26].

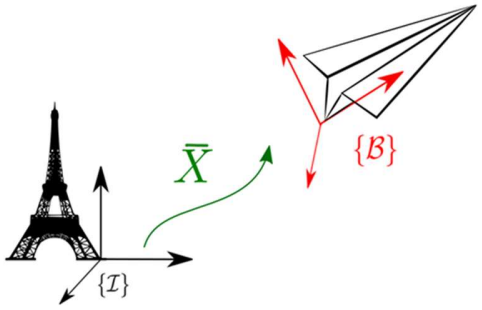


Figure 1: Graphical description of the trajectory $\bar{X}$ of a rigid body fixed frame {B} with respect to a fixed frame {I} [11].

Under the kinematic constraints of an aerial mobile robot $\dot{\mathrm{X}}(\mathrm{t}) = \hat{\mathrm{X}}(t) \cdot \mathrm{U}(t)$ (where X is the attitude rotation matrix and U is the angular velocity), there exists a trajectory $\bar{X}$ (Figure 1) in which the landmark measurement and velocity input noise processes are at a minima. This trajectory is considered the best estimate of the SO(3) motion of the state. This is determined by the cost function, which represents the state transition trajectory with the minimum noise

processes (or minimum-energy). We find this minimum-energy cost function trajectory by calculating the derivative of the value function V(X,t), where $\frac{dV}{dt} = 0$. When solving for the covariance of the state following the state transition, P^{-1}, it is assumed that the matrix P is quadratic. Under this assumption, a closed form solution to P can be calculated through a Taylor series expansion to the second derivative. As such, the covariance estimate calculated by the filter is only an approximation, although it is a higher order approximation than the first-order linearization of other nonlinear filters. As demonstrated by Zamani [9], the approximation order of the proposed method has the potential to be extended to arbitrary higher orders.

The quaternion state observers and Riccati equations for the GAME Filter and MEKF [11] are:

GAME Filter

$$\dot{\hat{q}} = \frac{1}{2}\hat{q} \otimes (\text{vex}(A) + \sum_i P(y_i \times \hat{y}_i^T)),\ \hat{q}(0) = [\,1\,0\,0\,0\,]^T \tag{7}$$

$$\dot{P} = \mathbb{P}_s(2PA - \sum_i P\left(P(y_i \times \hat{y}_i^T)\right)_\times) + I - P(\sum_i \text{trace}(\mathbb{P}_s(\hat{y}_i y_i^T))I - \sum_i \mathbb{P}_s(\hat{y}_i y_i^T))P \tag{8}$$

MEKF

$$\dot{\hat{q}}_m = \frac{1}{2}\hat{q}_m \otimes (\text{vex}(A) + \sum_i P_m(y_i \times \hat{y}_i^T)),\ \hat{q}_m(0) = [1\,0\,0\,0]^T \tag{9}$$

$$\dot{P_m} = \mathbb{P}_s(2P_m A) + I - P_m(\sum_i \text{trace}(\mathbb{P}_s(\hat{y}_i \hat{y}_i^T))I - \sum_i \mathbb{P}_s(\hat{y}_i \hat{y}_i^T))P_m \tag{10}$$

Where P is given by the Riccati equation of the filters, equivalent to the inverse of the weighting matrix which represents the state covariance, and A is the state transition matrix. The MEKF and the GAME Filter share the same observer equations. However, the term -$\mathbb{P}_s$ $\sum_i (P(y_i \times \hat{y}_i^T))_\times$ that appears in the Riccati equation of the GAME Filter (8) does not appear in the MEKF's Riccati equation (10). This term favours the quadratic term in GAME's Riccati equation over the MEKF's by utilizing not only the information contained in the estimates ($\hat{y}_i$) but also the information in the measurements (y_i). This term will also be small once the filter has converged and $\hat{y}_I$ is close to y_i. The current belief is that this term assists with curvature correction when traversing in SE(3) [9]. As a result, several simulations have indicated better transient and asymptotic behaviour of the GAME Filter compared to the MEKF [11].

The GAME Filter has also been shown to be near-optimal in performance, meaning that the cost function attained by the GAME Filter is close to the minimum-energy cost [10].

3.1.6 Mobile Robot Kinematics

Kinematics refers to the study of the motion of points, objects, and groups of objects with respect to a reference frame or space without considering the causes of its motion (such as forces). [12] As described by Siegwart and Nourbakhsh [13], the modelling of a mobile robot's kinematics depends on the properties of the robot and what points on the robot are of interest to the application. Such properties include the type of wheel, the type of steering and the types of actuation. These properties not only provide the context for modelling the kinematics of the robot but also determine the degrees of freedom and actuation, which will also influence the kinematic model of the robot in the form of motion constraints.

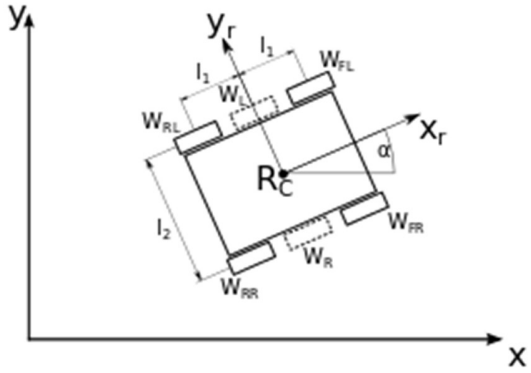


Figure 2: Modelling kinematic equivalence between front and rear pairs of wheels and left and right centre wheels. [14]

The Clearpath Jackal UGV is a skid-steer differential drive robot, in that it has four wheels but each pair of wheels on either side are actuated by a single motor. For this reason, the Jackal can be modelled as a standard two-wheel differential drive system (Figure 2) with additional terms to account for the individual motion and slip of each wheel. As discussed in Section 4.2, based on the future testing surfaces, these additional terms for slip can be ignored for when modelling the Jackal for a GAME Filter implementation.

3.1.7 Robot Operating System (ROS)

The operating system on the Clearpath Jackal UGV used for the Jackal Project is Ubuntu 16.04. Subsequently, all software packages for the Jackal are written for and executed within the environment known as Robot Operating System (ROS), an open-source, meta-operating system for robots [15]. The primary advantage to ROS is both its modularity with executable code, commonly in the form of *packages*, and its easy integration of and communication between pieces of software and robot hardware.

Given that the author had no previous experience with ROS prior to embarking on this project, extensive research into literature, resources and tutorials, in addition to the hours of trial and error, were required to confidently navigate ROS. The most straightforward way to learn the fundamentals of ROS was to follow the tutorial series on the ROS wiki page [16]. The YouTube series by Tiziano Fiorenzani [17] aided with the setup of the ROS environment in Ubuntu, as well as providing a visual tutorial into the basics of ROS that supplemented the ROS wiki page. A guide published by Griffith University [18] aided in the learning of commands needed to navigate and manipulate the ROS file system structure. Documentation and instructions on how to simulate and manipulate the Jackal in ROS were provided by the Jackal's manufacturer, Clearpath Robotics [19]. The Jackal simulation packages and installation instructions for ROS Kinetic are publicly accessible and open-source [20]. The current Extended Kalman Filter used to fuse odometry and measurement information on the Jackal was also developed by Clearpath [21] and is embedded into the package used for controlling the Jackal's motion [22].

3.2 Project Scope

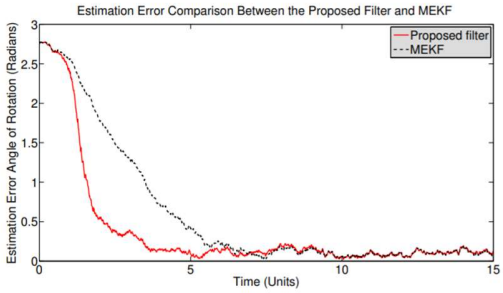
This section provides a summary of the prior research progress of the GAME Filter and how the goals of this research formulated the scope for this individual project. This section also presents the commercial advantages and applications of improving filtering methods, specifically the GAME Filter.

3.2.1 Summary of Prior Research

The GAME Filter was first proposed by Zamani in 2012 [10, 25] and formed the main result of his PhD dissertation [9]. In his dissertation, Zamani introduces the attitude estimation problem on the 3D rotation and special Euclidean groups along with nonlinear vectorial measurement models. The theory of minimum-energy filtering (Section 3.1.4) is adapted to respect the geometry of the problem and reduce state linearization error. This approach results in the geometric approximate minimum-energy (GAME) filter which, in a range of Monte

Carlo simulations, outperforms all current state-of-the-art attitude filters including the MEKF [26]. Zamani [9] also demonstrates that the proposed filter is shown to be near-optimal by deriving a bound on the optimality error of the filter that is proven to be small in simulations. The superior performance of the proposed GAME Filter in simulations was further supported with a least-squares analysis.

Throughout the research period of Zamani's dissertation, several journal and conference papers were published. Each of these papers focussed on aspects of the development of the algorithm, including validating the near-optimal cost when attitude filtering [23] and observing the filter's performance when performing a higher order differentiation of the value function [24]. Concluding the dissertation research, Zamani, Trumpf and Mahony derive the GAME Filter equations based on a cost function dependent on the Hessian (second-order approximation) of the Value function. [10]
In the specific simulations performed for each of these papers, the GAME Filter is compared to and outperforms industry standard quaternion version derivations of the EKF, including the MEKF.



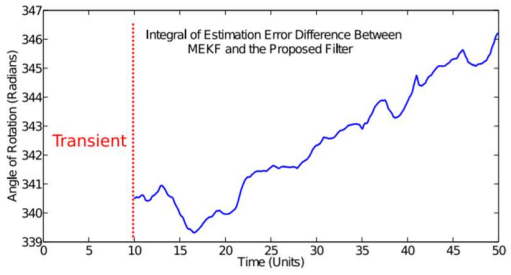
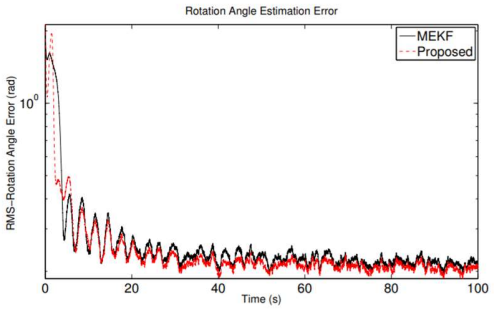


Figure 3 (left): Comparing the estimation errors of the proposed (GAME) filter and the MEKF. [10]

Figure 4 (right): The integral of the estimation error of the MEKF minus the estimation error of the proposed (GAME) filter. [10].

Figure 3 shows that the GAME Filter has equivalent performance to the MEKF but with faster estimation convergence. Figure 4 shows the integral is slowly growing in time, which indicates the proposed filter also has a slight advantage in asymptotic behaviour compared to the MEKF. This behaviour difference is observed in most of the simulation tests with differing initial conditions and measurement errors.

Zamani, Trumpf and Mahony [25] then applied the GAME Filter to filter attitude kinematics and introduced an extension of the GAME Filter for slowly time-varying bias in gyroscope measurements. As shown in Figure 5, this derived extension achieved a lower root-mean-square rotation angle estimation error than the MEKF.



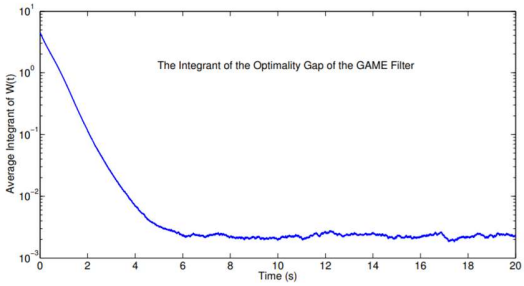


Figure 5 (left): The rotation angle estimation error performance of the proposed filter compared against the MEKF. [25]

Figure 6 (right): The average of the integrand of W(t) over 100 repeats plotted against time [11].

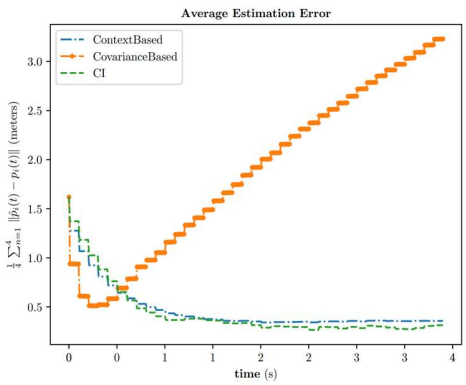
Zamani, Trumpf and Mahony [11] derived the equation for the optimality gap, *W(t),* which quantifies the difference between the numerical optimal minimum-energy cost and the cost attained by the GAME Filter when approximating the expansion of the Value function to the second-order derivative.

Figure 6 demonstrates that the GAME Filter achieves a cost that is close to the minimum-energy cost, with an average integrand of $W(t) < 0.01$.

A comparison of nonlinear filtering methods applied to attitude estimation was performed by Zamani, Trumpf and Mahony [26] to compare the GAME Filter to several widely implemented, well-performing filters. The filters were compared for their performance in reducing root-mean-square rotation estimation error and handling gyroscope bias over time. The GAME Filter was shown to have the best performance of all the presented filters.

Earlier this year, Zamani and Hunjet [27] present a collaborative localization algorithm, aimed at achieving accurate localization for all robots in a swarm with minimal landmark measurements. The aim of the paper was to demonstrate the accuracy of the algorithm even when measurements are noisy or non-existent (due to interference, poor-quality sensors, dropouts, etc). This was achieved by deriving the algorithm such that the covariance matrices could be flexibly tuned based on contextual and/or environmental knowledge, the covariance of the other robots in the swarm, and covariance intersection. The GAME Filter is used as the pose estimation algorithm on all robots. Even with no external landmarks, the four simulated robots were able to achieve a localization error of less than half a meter for a short period of

time becoming overconfident (Figure 7). With limited external landmarks, the proposed algorithm matched the estimation error performance of established collaborative localization algorithms whilst reducing the number of required communications and the distributed processing load of the swarm (Figure 8). All three plots are of the proposed algorithm with minor variations in their implementation.



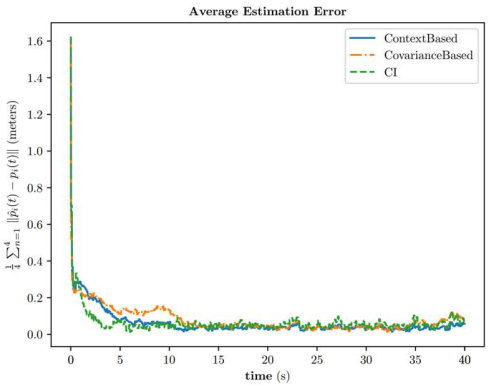


Figure 7 (left): Average estimation error of the proposed collaborative localization method with no external landmarks.

Figure 8 (right): Average estimation error of the proposed collaborative with limited external landmark measurements.

Zamani and Hunjet [27] have demonstrated the potential performance of their proposed collaborative localization algorithm, using the GAME Filter for pose estimation, with noisy and/or minimal landmark measurements available. Their results demonstrate that, in a collaborative swarm, robots carrying **noisier** sensor hardware can still accurately locate themselves within unknown environments. A potential application is one of the robots in a swarm carries far more sophisticated measurement hardware than the others, but the pose estimation error of each robot in the swarm still meets the requirements for the application.

Overall, the performance of the GAME Filter in a variety of applications is promising. However, in order to validate the real-world performance of the GAME Filter, it will need to be implemented onto a physical robot to be tested under real-world conditions. Having this implementation will allow researchers to observe how the filter performs processing variances in the noise processes that may not have been accounted for in simulation. A hardware implementation will also provide a baseline for future physical application of the filter and allow the algorithm to be developed such that it can be used on a variety of mobile robots for a wide array of applications.

Following this validation, the next goal of the Jackal Project is to implement the collaborative localization algorithm proposed by Zamani and Hunjet [27] onto a Jackal. If the filter can be shown to replicate a similar performance to the simulation, it can be applied to multiple mobile robots (such as more Jackal) to improve the overall localization performance of the system. As demonstrated by shown by Roumelitis and Bekey [28], the localization ability of a swarm of mobile robots is far greater than that of a single robot.

3.2.2 Development of Project Scope

To aid the validation and development of the GAME Filter on physical platforms, the DSTG obtained several Clearpath Jackal UGV robots. The research partnership between the DSTG and the ANU resulted in the university being provided a Jackal to be used as a physical development platform for validation and research. The Jackal is an ideal initial hardware platform for validating the GAME Filter. As an UGV, it has fewer degrees of actuation and freedom than a UAV, making it easier the kinematic motion and estimate process noises. The Jackal itself is a thoroughly pre-configured development platform with tuned measurement covariances for its proprioceptive sensors.

The original scope of this project was to develop a general implementation of the GAME Filter as a ROS package. This implementation would then be installed onto the provided Jackal to compare the performance of the GAME Filter to other state-of-the-art filters in a variety of single and multi-robot situations. As this project developed, it was determined that this was a far larger task than initially anticipated. Achieving the original goals set for the project requires an understanding of the theory behind what is trying to be achieved and sufficient background knowledge of the software platform to implement the filter. This included developing a thorough understanding of stochastic filters and how to implement them and becoming proficient enough in ROS to develop nodes and packages that integrate with the Jackal. Gaining this required background knowledge took up a large portion of this project. The scope was then altered to focus on laying the groundwork for others to work towards achieving the original scope.

3.2.3 Final Project Scope

The scope of this individual project is to develop a method for implementing the GAME Filter onto the Clearpath Jackal UGV. Primarily, this involved working through available literature to gain an understanding of how filtering algorithms are currently implemented in Robot Operating System (ROS) and on the Jackal.

The first milestone of this project was to develop an Extended Kalman Filter (EKF) for the Jackal. The EKF is arguably the most renowned filtering algorithm for nonlinear dynamical systems, as most industry-leading filtering algorithms some extension on the EKF. To achieve this required learning the theory behind both linear and nonlinear (extended) Kalman filtering and mobile robot kinematics. This understanding of filtering theory was both tested and expanded by developing my own Kalman filter scripts for a variety of dynamic systems in MATLAB and Python. The project also involved learning how to translate robot actuation to the motion of a mobile robot and extend previously acquired knowledge of coordinate frames and dynamic systems to transform a robot's motion between arbitrary coordinate frames. Using this acquired knowledge, an equivalent kinematic model for the Jackal would be developed to then fuse with simulated noisy sensor measurements to estimate a Jackal's pose using an EKF in Python.

The second milestone was to understand how to implement a filter into ROS (specifically, for the Clearpath Jackal UGV) and then develop a custom filter package to be implemented onto the Jackal. This involved learning about how filtering algorithms are currently implemented into ROS and reviewing the literature related to the packages that run these filters. The next step was to review literature related to the Jackal and how the robot initializes and publishes relevant information (such as sensor measurements) to the onboard filtering package. The outcome was an EKF implementation in ROS which relied on the sensor information published from the Jackal.

The final milestone and major goal of the project was to develop an implementation of the GAME Filter using the acquired knowledge and methods from the first two milestones. This milestone was considered a stretch goal and was not able to be achieved within the project time. The final project output was rescoped to use the knowledge acquired throughout the project to develop a method for a future implementation of the GAME Filter onto the Jackal. The outputs of this project will aid the eventual implementation of a GAME Filter onto the Jackal, which will subsequently assist the ANU's future research and development of the filter.

Chapter 4 Methodology

This chapter outlines the method undertaken to establish an understanding of the required background theory and construct the equations needed for an EKF implementation to be used for estimating the pose of a Jackal.

4.1 Understanding Filtering Theory

An established understanding of the underlying filtering theory was required in order to gain an understanding of the project's proposed scope. Chapters 1-3 of *Optimal Filtering* by Anderson and Moore [3] established a basis of knowledge of linear Kalman filters such that, whilst not entirely understood initially, was the primary reference for other explanations and derivations of the linear Kalman filter [29][30][31][32].

To apply the knowledge gained from the literature, a linear Kalman filter model was created to solve a basic estimation problem: modelling the displacement of a ball rolling down a slope using noisy ultrasonic sensor measurements. Figure 9 shows the position as estimated by the filter.

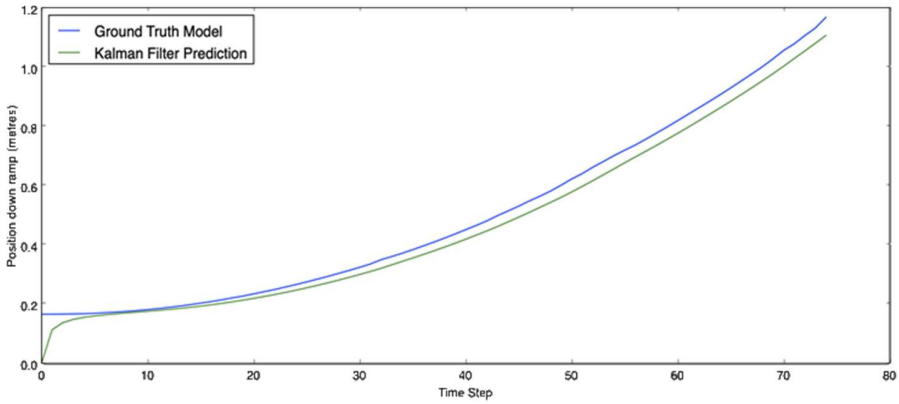


Figure 9: First implementation of a linear Kalman filter in Python, estimating the position of a ball rolling down a hill.

Having a fundamental understanding of linear Kalman filters allowed the author to delve into resources that described how to filter nonlinear systems. Out of many available papers and online resources describing nonlinear filters, there were a handful of resources that provided an effective introduction to the most popular and standardized non-linear implementation of the Kalman Filter, the Extended Kalman Filter (EKF) [33][34][35][36].

To cement the author's understanding of a nonlinear filter, an EKF was applied to estimate the position of an inverted pendulum using noisy horizontal displacement measurements. Figure 10 shows the position of the pendulum as estimated by the filter.

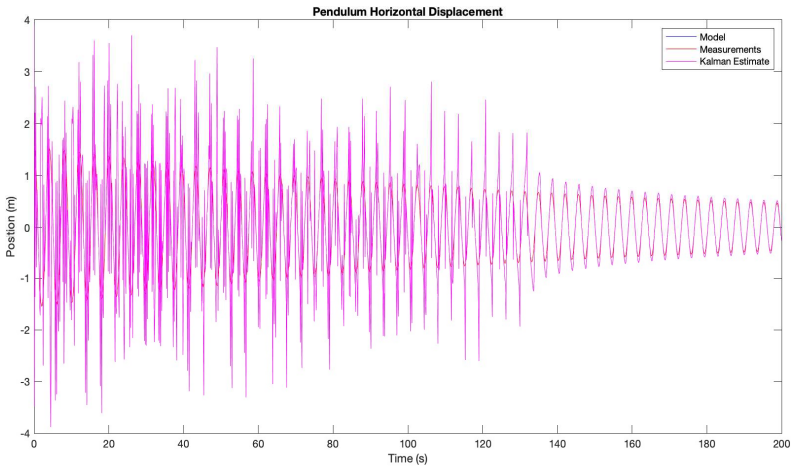


Figure 10: First implementation of an Extended Kalman Filter in MATLAB, estimating the horizontal position of a pendulum.

One key conclusion that came from creating the inverted pendulum filter was the importance of tuning the noise covariance matrices. In most Kalman filter implementations, noise is modelled as either process noise or measurement noise. Process noise is the possible variance the observation model has from the true state of the system, usually given by stochastic disturbance to the system. In the example of the inverted pendulum, should this have been a system implemented in the real world, the process noise covariance matrix would account for physical properties that would make the model differ from the real system (such as the effect of air resistance and string tension). Measurement noise is an estimated uncertainty of the measurements made by sensors. These uncertainties are often given by the manufacturers of higher-end sensors to be modelled and accounted for. Each of these noise errors are described by their potential variance from the expected values. Figure 11 shows the same filter with slightly higher initial state and noise covariance values. The observation is made that even very slight modifications to these noises covariance values results in a large variance in the behaviour of the system.

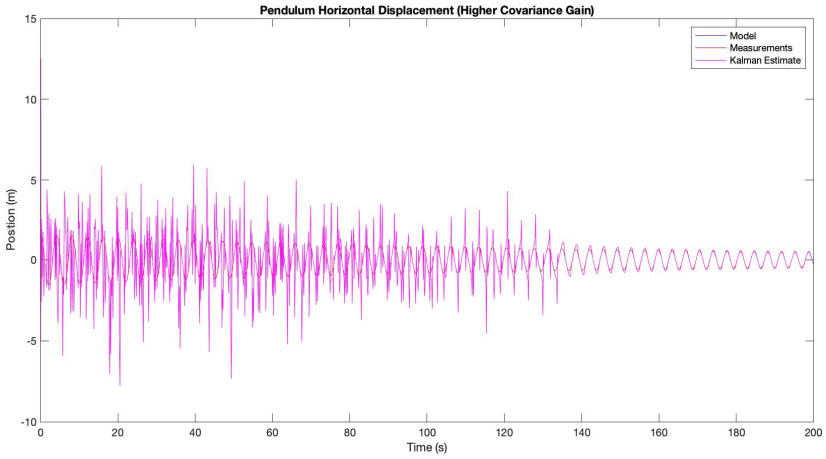


Figure 11: EKF estimating the horizontal position of a pendulum with slight variances in the initial state and measurement covariances.

These experiments demonstrated that it is vital to the performance of a filter that these noise covariance matrices are set or "tuned" as accurately as they can be. Even slight derivations in the values had a great impact on the accuracy and robustness of the system, both immediately and over time.

4.2 Kinematic Modelling of the Clearpath Jackal UGV

The Clearpath Jackal UGV is a four wheeled, skid-steer differential drive robot. Both pairs of wheels on each side are actuated by single motors. In a skid-steer robot, there is no explicit steering mechanism. Steering is accomplished by actuating the wheels on each side at a different rate or in a different direction, causing the wheels or tracks to slip, or "skid" on the ground. [37]

The kinematic description for a skid-steer robot are given by:

$$\begin{bmatrix} v_x \\ v_y \\ \omega_z \end{bmatrix} = \frac{r}{B\chi} \begin{bmatrix} \frac{B\chi}{2} & \frac{B\chi}{2} \\ 0 & 0 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} \omega_l \\ \omega_r \end{bmatrix} \tag{11}$$

Where v_x, v_y and ω_z are the linear and angular velocities of the robot with respect to its body fixed coordinate frame, ω_l and ω_r are the angular velocities of the left and right wheel, r is the radius of each wheel and B is the distance between adjacent wheels and χ is a surface coefficient [38].

Equation (11) describes the relationship between the angular velocity of each wheel to the linear and angular velocity of the Jackal's body fixed frame [38]. Equation (12) performs the transform between the velocity of the robot's body fixed frame to the global coordinate frame.

$$\begin{bmatrix} \dot{X} \\ \dot{Y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} cos\theta & -sin\theta & 0 \\ sin\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} v_x \\ v_y \\ \omega_z \end{bmatrix} \tag{12}$$

On ideal surface conditions (where the $\chi \approx 1$), the skid steer differential drive robot model is kinematically equivalent to the standard differential drive robot model, as illustrated in Figure 12. The difference between the models is that the two-wheel differential drive model assumes constant and coincident wheel contact points with the ground, whilst the contact points of a skid-steer robot are dynamics-dependent and always lie outside of the tread centrelines because of slippage [39]. Depending on the linear and angular velocities of the robot, in addition to the surface conditions, the contact area on the wheels of the skid-steer robot also changes. This results in a small difference in motion from the two-wheel model. On ideal surfaces, this difference is very small and can be ignored. The two-wheel model also holds true for the rolling non-slipping constraints, meaning that the linear motion of the robot is always in the X direction of the robot and therefore ignoring slip.

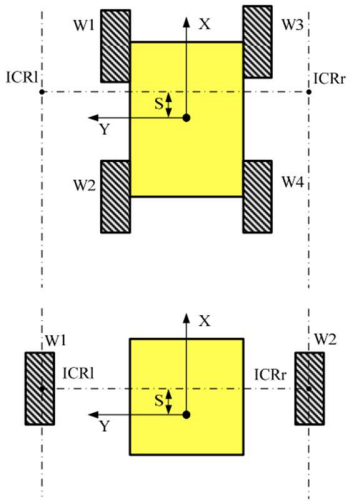


Figure 12: Geometric equivalence between skid-steer model and two-wheel differential drive model under rolling non-slipping assumptions [13].

Future experiments involving the Jackal will likely be performed on carpet and on concrete which can be assumed to be ideal surface conditions with minimal slip. As a result, it can be stated that the Jackal can be considered kinematically equivalent to a two-wheel differential drive robot, including the non-Holonomic rolling non-slipping constraints of the model.

4.2.1 EKF Model for the Jackal UGV

Let $\dot{\boldsymbol{x}}_{\boldsymbol{k}} = [\dot{\mathrm{X}}_{\mathrm{k}}, \dot{\mathrm{Y}}_{\mathrm{k}}, \dot{\theta}_{\mathrm{k}}]^{\boldsymbol{T}}$ be the velocity of the Jackal in a global coordinate frame and $\boldsymbol{u}_{\boldsymbol{k}} = [\mathrm{v}_{\mathrm{x}_{\mathrm{k}}}, \omega_{\mathrm{z}_{\mathrm{k}}}]^{\boldsymbol{T}}$ ($\mathrm{v}_{\mathrm{y}_{\mathrm{k}}} = 0$) represent the control input matrix, using the variables from (12). Performing a forward Euler integration on $\dot{\boldsymbol{x}}_{\boldsymbol{k}}$ and giving equivalence to (4) yields:

$$\hat{\mathbf{x}}_{\mathbf{k}|\mathbf{k}-\mathbf{1}} = \mathrm{x}_{\mathrm{k}-1|\mathrm{k}-1} + (\dot{\boldsymbol{x}}_{\boldsymbol{k}} * \partial \mathrm{t}) = \boldsymbol{f}\left(\mathrm{x}_{k-1|k-1}, u_k\right) \tag{13}$$

$$\boldsymbol{f}(\mathrm{x}_{k-1|k-1}, u_k) = \begin{bmatrix} \mathrm{X}_{k-1|k-1} + v_{x_k} * cos(\theta_{k-1|k-1}) * \partial t \\ \mathrm{Y}_{k-1|k-1} + v_{x_k} * \sin(\theta_{k-1|k-1}) * \partial t \\ \theta_{k-1|k-1} + \omega_{z_k} * \partial t \end{bmatrix} \tag{14}$$

Where $\hat{\mathbf{x}}_{\mathbf{k}|\mathbf{k}-\mathbf{1}}$ (13) is an estimate of the pose of the robot in the global coordinate frame at time *k* based on the state of the previous time step *k-1*. We linearize the current state estimate by solving for $\boldsymbol{F}$ (6a) and $\boldsymbol{B}$**,** which are the derivatives (Jacobians) of $\boldsymbol{f}(\mathrm{x}_{k-1|k-1}, u_k)$ with respect to the previous state estimate and the control input matrix, respectively. We solve for these derivatives by performing a first-order Taylor expansion at the previous state estimate and current control input.

$$\boldsymbol{F}_{\boldsymbol{k}} = \begin{bmatrix} \frac{\partial f_1}{\partial \hat{X}_{k-1|k-1}} & \frac{\partial f_1}{\partial \hat{Y}_{k-1|k-1}} & \frac{\partial f_1}{\partial \hat{\theta}_{k-1|k-1}} \\ \frac{\partial f_2}{\partial \hat{X}_{k-1|k-1}} & \frac{\partial f_2}{\partial \hat{Y}_{k-1|k-1}} & \frac{\partial f_2}{\partial \hat{\theta}_{k-1|k-1}} \\ \frac{\partial f_3}{\partial \hat{X}_{k-1|k-1}} & \frac{\partial f_3}{\partial \hat{Y}_{k-1|k-1}} & \frac{\partial f_3}{\partial \hat{\theta}_{k-1|k-1}} \end{bmatrix} = \begin{bmatrix} 1 & 0 & \partial t * v_{x_k} * -\sin(\hat{\theta}_{k-1|k-1}) \\ 0 & 1 & \partial t * v_{x_k} * \cos(\hat{\theta}_{k-1|k-1}) \\ 0 & 0 & 1 \end{bmatrix} \tag{15}$$

$$\boldsymbol{B}_{\boldsymbol{k}} = \frac{\partial f}{\partial u_k}\Big|_{\mathrm{x}_{k-1|k-1}, u_k} = \begin{bmatrix} \frac{\partial f_1}{\partial \mathrm{v}_{\mathrm{x}_{\mathrm{k}}}} & \frac{\partial f_1}{\partial \omega_{\mathrm{z}_{\mathrm{k}}}} \\ \frac{\partial f_2}{\partial \mathrm{v}_{\mathrm{x}_{\mathrm{k}}}} & \frac{\partial f_2}{\partial \omega_{\mathrm{z}_{\mathrm{k}}}} \\ \frac{\partial f_2}{\partial \mathrm{v}_{\mathrm{x}_{\mathrm{k}}}} & \frac{\partial f_2}{\partial \omega_{\mathrm{z}_{\mathrm{k}}}} \end{bmatrix} = \begin{bmatrix} \partial t * cos(\hat{\theta}_{k-1|k-1}) & 0 \\ \partial t * \sin(\hat{\theta}_{k-1|k-1}) & 0 \\ 0 & 1 \end{bmatrix} \tag{16}$$

Let $l_k = [l_x^1, l_y^1, l_x^2, l_y^2 \dots l_x^n, l_y^n]$ represent the positions of known *n* landmarks in the global coordinate frame and $z_k = [\hat{l}_x^1, \hat{l}_y^1, \hat{l}_y^2, \hat{l}_y^2 \dots \hat{l}_x^n, \hat{l}_y^n]$ be the measurements of each landmark made by the robot in the body fixed frame. The output or *innovation* of the system is then given by (5), where $h(\hat{x}_{k|k-1}, l_k^n)$ maps the known landmark positions to the variables of the state.

$$h(\hat{x}_{k|k-1}, l_k^n) = \begin{bmatrix} -\cos(\hat{\theta}_{k|k-1})(\hat{X}_{k|k-1} - l_{x,n}) - \sin(\hat{\theta}_{k|k-1})(\hat{Y}_{k|k-1} - l_{y,n}) \\ \sin(\hat{\theta}_{k|k-1})(\hat{X}_{k|k-1} - l_{x,n}) - \cos(\hat{\theta}_{k|k-1})(\hat{Y}_{k|k-1} - l_{y,n}) \end{bmatrix} \tag{17}$$

Solving for the Jacobian of $h(\hat{\boldsymbol{x}}_{\boldsymbol{k|k-1}})$ using (6b) gives:

$$\boldsymbol{H_k} = \begin{bmatrix} -\cos(\hat{\theta}_{k|k-1}) & \sin(\hat{\theta}_{k|k-1}) & (\sin(\hat{\theta}_{k|k-1})(\hat{x}_{k|k-1} - l_{x,n}) - \cos(\hat{\theta}_{k|k-1})(\hat{y}_{k|k-1} - l_{y,n}) \\ \sin(\hat{\theta}_{k|k-1}) & -\cos(\hat{\theta}_{k|k-1}) & (\cos(\hat{\theta}_{k|k-1})(\hat{x}_{k|k-1} - l_{x,n}) + \sin(\hat{\theta}_{k|k-1})(\hat{y}_{k|k-1} - l_{y,n}) \end{bmatrix} \tag{18}$$

Using these equations, we can then iteratively update the state by solving for (2a – 2e) after each exteroceptive measurement of a known landmark. These equations make up the EKF for estimating the pose of a differential drive robot with respect to a global coordinate frame, using known landmarks as update measurements. Equations (17) and (18) can be manipulated for other measurement devices, such as sensors, to provide additional information to the filter when updating the state.

4.3 The Clearpath Jackal UGV in ROS

The Clearpath Jackal UGV (Figure 13) was provided to the ANU by the DSTG as part of the collaboration between the two organisations, to aid in the research and development of the GAME Filter. The Jackal is, as described by the manufacturer, a “small, fast, entry-level field robotics research platform. It has “an on-board computer, GPS and IMU fully integrated with ROS for out-of-the-box autonomous capability” [40] and is compatible with a variety of sensors, vision systems and actuators. The Jackal is running Ubuntu 16.04 and is programmed using ROS.

Figure 13: The Clearpath Jackal Unmanned Ground Vehicle (UGV) [40]

4.3.1 Updating the software and firmware on the Clearpath Jackal

Upon its arrival to the ANU, the Jackal was found to have been running Ubuntu 14.04.5 and, subsequently, ROS Indigo. This was the configuration of the robot as sent to the DSTG by the manufacturer. The developers of the ROS platform state on their website that the Indigo distribution was to be considered "end-of-life" as of May 2019 [41]. The tutorial pages published by the manufacturer, Clearpath, for setting up and simulating the Jackal [42] referred only to their packages made for ROS Indigo. Clearpath updated their Jackal packages to include support for ROS Kinetic in June of 2018 [17]. As ROS Kinetic was only primarily targeted to run on Ubuntu 16.04 [43], this meant that our Jackal required a complete reinstallation of its operating system.

With the assistance of Glen Pearce from the DSTG, the Jackal was updated to Ubuntu 16.04 and new firmware was installed to allow for the onboard GPS module to function. The GPS module is believed to be functional and publishing to a topic (see Section 5.2) but does not appear to be producing valid location information due to not being able to find a satellite fix whilst inside its resident building.

4.3.2 Simulating the Jackal using ROS Gazebo

For the majority of this project, development in ROS was performed on a simulated Jackal on a virtual machine running Ubuntu 16.04 and ROS Kinetic. The packages, services, nodes and topics available in the provided Gazebo simulation packages were observed to be near-identical to the configuration of the actual Jackal. Installing the software packages and dependencies for the Jackal were done so following the instructions provided by Clearpath [42].

The ROS package *jackal_gazebo* aims to replicate of some of the packages on the physical Jackal in how it mimics the publication of sensor hardware data onto topics. The best method for understanding the ROS integration and packages for the Jackal, both in simulation and the physical robot, was to view each topic, node, and service individually. Any of those relevant to filtering were analysed more closely. This allowed for a thorough understanding of how the Jackal provides sensor information to its EKF node, to then aid my own filter implementation. The filter presented in Chapter 5 of this report was primarily developed in the simulated environment shown in Figure 14.

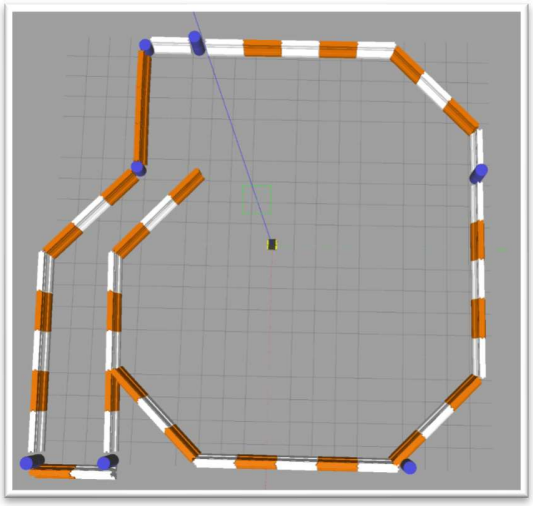


Figure 14: The default Gazebo simulation environment for the Jackal, with the two middle walls removed in order to allow the Jackal to traverse further distances from the origin.

4.3.3 Understanding Current Pose Filtering Implementations in ROS

The most popular open-source nonlinear filter ROS package currently available for ROS Kinetic is the *robot_localization* package, written and maintained by Clearpath Robotics [44]. The package contains a collection of state estimation nodes which allow a user to implement non-linear state estimators for mobile robot systems without the need to formulate the filter themselves. The state estimation nodes implement either an EKF or UKF to produce an estimate of the robot's pose with respect to a defined global coordinate frame. The nodes take measurements from an arbitrary number of defined sensors, including multiples of the same type of sensor. This allows the filter nodes to perform effective sensor fusion; taking full advantage of the standardized ROS message types for odometry and IMU data, such as *nav_msgs/Odometry* and *sensor_msgs/Imu*.

The parameter or "configuration" files allow for the flexible definition of several aspects of the filter, including the initial states, covariances and measurement data sources. The output of these nodes is an odometry message (of type *nav_msgs/Odometry*), which describes the state of the robot (as instantaneous pose, linear and angular velocity) and the covariance matrix of the state. The *robot_localization* package is integrated into the *jackal* ROS packages from Clearpath. It the default filtering package shipped with the Jackal and publishes the filtered state estimate to the */odometry/filtered* topic.

4.4 Developing the EKF for the Jackal UGV

Using the EKF equations derived in Section 4.2.1, a Python implementation of an EKF for a differential drive robot was developed. This was developed outside of the ROS environment at first to make the code easier to debug, but very similar functions were then used in a ROS Python node for a custom ROS EKF implementation.

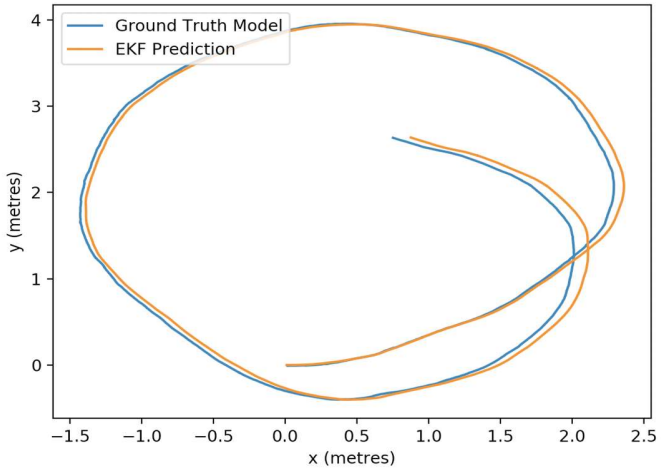


Figure 15: The state estimate of a modelled differential drive robot produced by an EKF in Python.

Figure 15 demonstrates the accuracy of the EKF model with Gaussian random linear and angular velocity inputs and noisy landmark measurements. Once the differential drive EKF model equations had been validated in a Python IDE, a ROS package was developed to implement the python script in a ROS node and to be simulated on the Jackal. The input velocities were taken from the */jackal_velocity_controller/odom* topic and represented not the control inputs to the Jackal, but the velocity of the robot at a given time. This was a choice made in order to replicate the information used by the EKF implementation already on the Jackal. The results and more detailed method of the development of the ROS package created to implement an EKF on the Jackal is discussed in Chapter 5.

Chapter 5 Results and Analysis

This chapter contains a summary of the information that is relevant to implementing a filter onto the Clearpath Jackal UGV in ROS, specifically the GAME Filter. Section 5.1 summarizes how the Jackal captures and relays both preconfigured proprioceptive and desired exteroceptive sensor information, and how to command the movement of the robot. Section 5.2 demonstrates how an EKF was implemented onto the Jackal using the information presented in Section 5.1. Section 5.3 summarizes how the GAME Filter would be implemented into ROS and onto one or more Jackals based on the information from Sections 5.1 and 5.2.

5.1 Summary of Available Information from the Jackal

The Jackal is configured by the manufacturer, Clearpath Robotics, such that the way with which the installed packages and scripts gather hardware-level information, such as from the sensor hardware, is to be of minimal concern to the user. These packages and scripts are made up of the *jackal* [45], *jackal_base* [46] and *jackal_description* [47] packages and subpackages; all of which are installed on the Jackal in the same layout they are presented within their respective github repositories. These packages produce relevant information about the intrinsic state of the robot and its sensors as standard message types. Such messages are then published by these lower level scripts (executed as ROS nodes to construct and publish these messages) onto topics that a user can then subscribe their own nodes to. An example of this process is presented in Section 5.2, which demonstrates how a custom node can subscribe to these topics and use the published information to perform localization and pose estimation using an Extended Kalman Filter.

This project focussed on the *base* configuration of the Jackal UGV; the *base* configuration referring to a Jackal without a declared payload (commonly, additional proprioceptive sensors, such as camera or front laser scanner). This is the default configuration. When installing a payload, the desired configuration of the robot can be declared in the *jackal_description* launch file (*description.launch*) with the parameters for each configuration presented in [47]. When simulating the Jackal, the parameter for enabling the front laser configuration is in the launch file that modifies the *description.launch* file when it creates an instance of the Jackal in Gazebo (*jackal_gazebo/launch/jackal_world.launch*). As such, it can be enabled by adding a *config* parameter when performing the *roslaunch* of the Jackal in Gazebo [48].

The most relevant information produced by the Jackal to a filter node are those that; contain information about the state; provide proprioceptive or exteroceptive measurements and; provide physical parameters about the robot such that it aids the robot's process model within the filter. The relevant topics on the *base* configuration of the Jackal that produce this information are presented in Table 1. These topics relate to information that is produced by the wheel encoders that reflect the odometry of the robot, the IMU and GPS modules and the velocity commands given to the robot.

The messages published to the topic */jackal_velocity_controller/odom* provide an estimate of the robot's pose (quaternion orientation), linear and angular velocity of the robot, based on its odometry. This information is gathered from the wheel encoders by the *diff_drive_controller* node; a member of the standardized robot controller package, *ros_controllers* [49]. The *diff_drive_controller* node performs an estimate of the odometry through forward Euler integration of the wheel (encoder) velocities [50, lines 140-169], and is believed to be the source of the robot's raw odometry values. The node is modelling the robot as a standard two-wheel differential drive robot. The odometry process covariance matrix diagonal elements for the internal EKF are declared in the *jackal_control/config/control.yaml* file and can be tuned accordingly.

The covariance diagonal values for the odometry and the IMU on the physical Jackal are very precise and differ from the Jackal model in simulation, which makes it highly likely that the covariance of these sensors have been finely tuned by the manufacturer. The IMU and GPS covariance diagonal values are declared in *jackal_base/launch/base.launch* file, which is launched on the robot's startup, and can be modified if needed. It is worth noting that the GPS module currently onboard the Jackal, the *Gtop GMM-U2P,* only has a position accuracy of 3 metres [51]. It is recommended that those involved with any future development of the Jackal seek a more accurate third-party module and have it publish to a new, separate topic.

	GPS (velocity)	GPS (position)	IMU	Odometry (filtered, EKF)	Odometry	Input Velocities
Topic	/navsat/vel	/navsat/fix	/imu/data	/odometry/filtered	/jackal_velocity_controller/odom	bluetooth_teleop/cmd_vel; c/md_vel
Message Type	geometry_msgs/Vector3Stamped	sensor_msgs/NavSatFix	sensor_msgs/Imu	nav_msgs/Odometry	nav_msgs/Odometry	geometry_msgs/Twist
Primary Publish Node	/nmea_topic_driver	/nmea_topic_driver	/imu_filter	/robot_localization	/jackal_base (diff_drive_controller)	N/A (user input)
Package	/nmea_navsat_driver	/nmea_navsat_driver	/imu_filter_madgwick	/robot_localization	/jackal_base	/moble_base_controller
Configuration Files	jackal_base/launch/base.launch	jackal_base/launch/base.launch	jackal_base/launch/base.launch	jackal_control/config/robot_localization.yaml	jackal_control/config/control.yaml	jackal_control/config/teleop(_ps4).yaml

Table 1: The sources of published sensor data by the Jackal relevant to filtering, base Jackal configuration.

The input velocity command topics differ dependent on application. The Jackal is pre-configured to be controlled using a Playstation 4 Bluetooth joystick controller, which publishes the controller's input values onto the */bluetooth_teleop/cmd_vel* topic. The standard */cmd_vel* topic also allows the user to publish messages onto it to control the robot. The open-source keyboard input command package, *teleop_twist_keyboard*, publishes to */cmd_vel* and was used during the simulated testing and development of the implementation presented in Section 5.2.1. The Jackal's preconfigured EKF (*/ekf_localization)* node does not subscribe to any direct input command topic and instead relies purely on the instantaneous velocity of the robot, provided by the odometry and IMU topics.

Another key ROS feature and source of information provided by the Jackal are the relationships between all known coordinate frames over time. ROS uses a package known as *tf* to maintain the relationship between known coordinate frames, including the frames of all payload accessories and even each of the wheels of the robot. This package provides the user with a tree structure overview of all system coordinate frames and allows for the transformation of points, vectors, etc between any two frames at a given time [52][53]. This will become increasingly more prevalent as the project moves into developing for movement around larger areas and for collaborative localization application.

Appendix A is the *tf* tree diagram (based on a generated tree diagram from ROS) that describes the relationships and hierarchy of all coordinate frames on the *base* Jackal. It is important to note that, as depicted, the EKF node is responsible for mapping the Jackal body fixed frame to the global (*odom*) reference frame. Aside from the transformation/updates between the body fixed frame and the frame link of each wheel, the no other transformations occur on the *base* Jackal.

The version of the Jackal's software; a disk image sourced from Clearpath [54]; can be found in the VERSION text file in the Jackal's home directory. The jackal_firmware package is where one will find the firmware version of the Jackal. It is not recommended to modify this package without prior consultation with the manufacturer, Clearpath Robotics.

5.2 EKF Implementation

Using the topics outlined in Table 1, an EKF was developed to estimate the Jackal's pose. The primary motivation for developing this custom implementation was ensure that the data published by the Jackal to the available topics were enough to create an accurate pose estimation filter. The implementation comes in the form of a package (*my_ekf*) which is made up in a file structure similar to other common ROS packages:

- A *launch* folder, which contains the "launch" file that calls an instance of the EKF node script (with declared parameters, if desired).
- A *config* folder, which contains the parameter (aka "param" or "YAML") file that declares the values of referenced parameters within the node. Having these parameters in a param file that allow for easier manipulation of variables, such as covariance values, subscribed data topics and the publishing frequency, without having to modify them in the node each time.
- A *src* folder, which contains the source files for the node. The primary node file, *ekf.py*, is called upon launch with the parameters of the declare param file.

This modular file structure is recommended for any future filter implementation. The primary node file receives messages from the declared topics and performs an EKF prediction and/or update, dependent on the information received. For proof of concept, three landmark locations are declared within the param file. With each time step, the filter node mimics a measurement from a landmark by calculating the distance and orientation from the body fixed frame to the known landmark coordinate and adds a small amount of Gaussian noise. These measurements are then used to update the current state estimate. The updated state is published to the topic */filter_output.*

In addition to the EKF node, a pose estimation comparison node was created to examine the performance of the internal pose estimation filter. The node subscribes to the raw, unfiltered odometry topic (*/jackal_velocity_controller/odom*), the filtered odometry topic (*/odometry/filtered*) and the topic produced by the custom EKF node *(/filter_output)* and graphs the pose estimate from each in real time. This comparison node also publishes the results to a text document that can be imported into Microsoft Excel for further analysis.

5.2.1 Gazebo Simulation

The following experiments were performed in a virtual machine Gazebo simulation of a Jackal. The Jackal was driven around in an approximate square, stopping and making 90 degree turns at each corner. The custom comparison node was used to record the pose estimation of the robot from the published topics of all the filtered and unfiltered odometry.

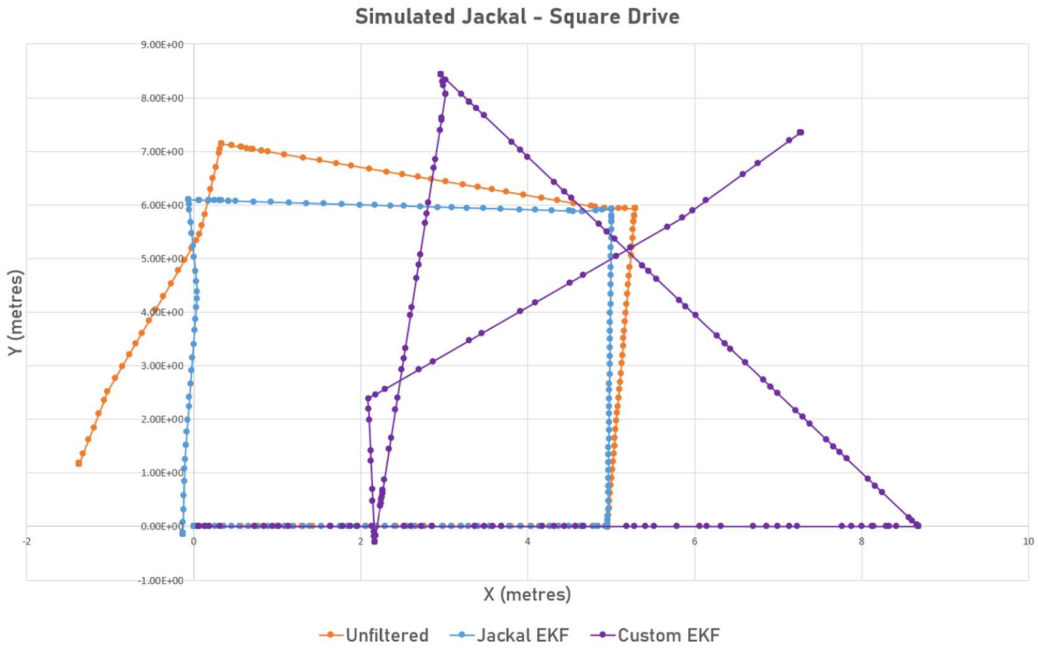


Figure 16: Plotting the estimated robot position of the unfiltered, Jackal EKF and the custom EKF outputs it travels in a square.

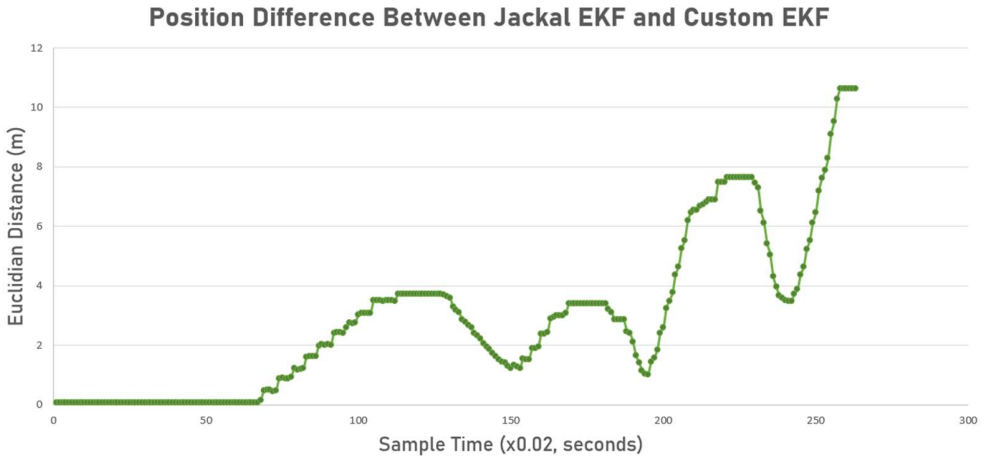


Figure 17: Euclidean distance between the estimated robot position produced by the custom EKF and the Jackal EKF.

As discussed in Section 5.1, the raw odometry node publishes a pose that is calculated using forward integration (Runge-Kutta method), while the robot's filter node takes the linear and angular velocities of the robot and estimates the pose using a polished EKF implementation. Figure 17 shows that, over time, the pose estimate of the custom EKF greatly diverges from the EKF pose estimate, which is considered to be a more accurate estimate of the true pose. What is immediately noticeable when comparing the integrated EKF and the custom model in is the difference in velocities. This is particularly notable during linear travel. The difference in position between time steps when comparing the custom EKF plot to the other plots is significant and shows that the amount of linear and angular velocity the robot is estimated to be experiencing is significantly higher than what the other two estimation methods are producing. Also observable in Figure 16 is the inconsistency in the difference in distance between points and the sample time is far more in the custom implementation than in either the unfiltered or Jackal EKF's pose estimation.

Given the process models of both the EKF and GAME Filters rely on a linear (x direction) and angular (yaw) velocity inputs, this variation between the expected velocity and the actual velocity will need to be considered when attempting to implement a filter. A way that was found to improve the performance of the custom filter and minimize the velocity difference was to apply a gain (0.5~0.7) to the velocity vector *u(t)*.

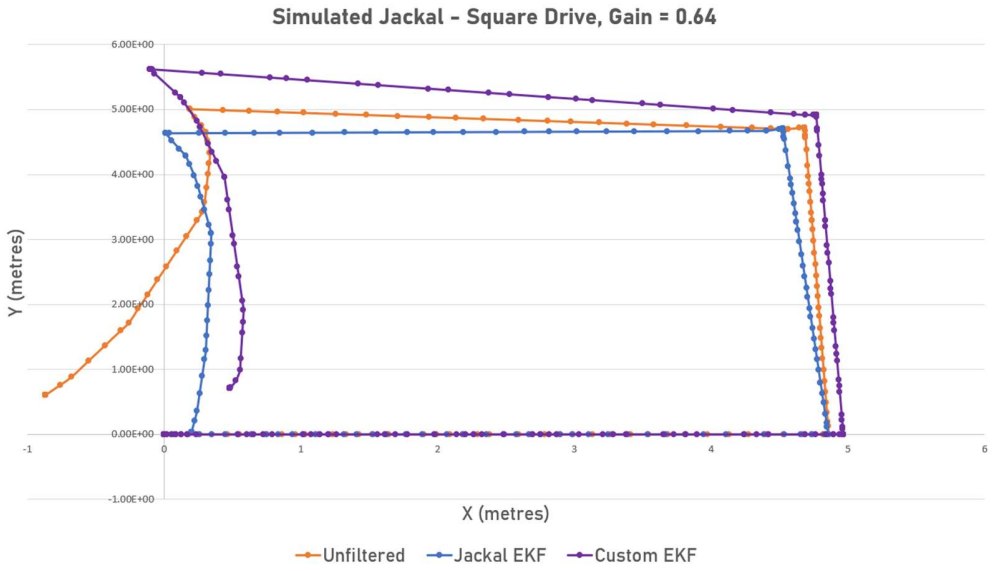


Figure 18: Plotting the estimated robot position of the unfiltered, Jackal EKF and the custom EKF outputs it travels in a square, with an applied custom EKF velocity gain of 0.64.

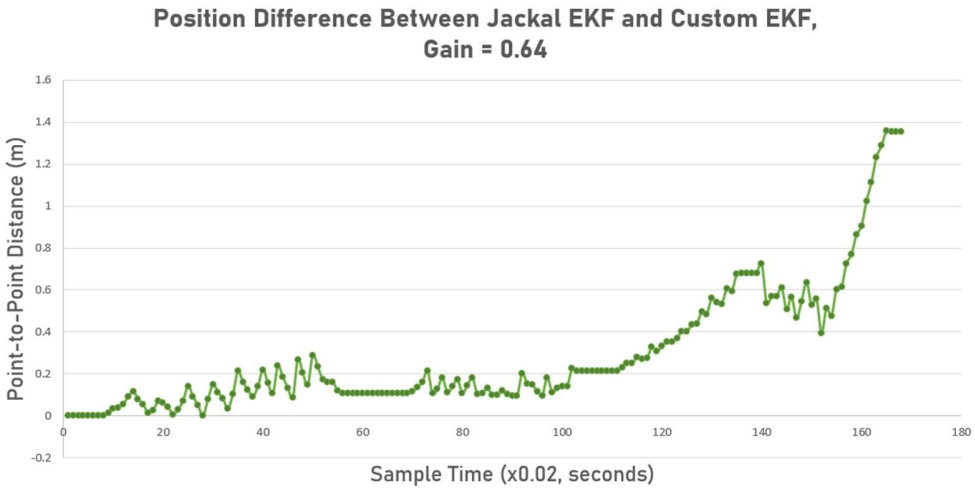


Figure 19: Euclidean distance between the estimated robot position produced by the custom EKF and the Jackal EKF, with an applied custom EKF velocity gain of 0.64.

Figure 18 and Figure 19 show the pose estimation performance of the custom EKF implementation. The velocity gain value of 0.64 was derived through trial and error, first starting at 0.5 and observing performance. When comparing Figures Figure 17 and Figure 19, the velocity gain appears to have greatly improved the pose estimation ability of the custom EKF. A potential reason for this is that the kinematic model and subsequent EKF model of the Jackal described in Section 4.2 may be incorrectly incorporating the velocity gain. The elements of the EKF process noise covariance matrix, Q, were experimentally tuned to observe their effects on the velocity discrepancy, but any change to the matrix made no impact. The exact cause of the velocity differences could not be determined.

5.2.2 On the Jackal UGV

The following experiments were performed on the Jackal UGV robot. Using a Playstation 4 controller as velocity input, the Jackal was driven around in an approximate square, stopping and making 90 degree turns at each corner. For the first three straights, the Jackal was given the normal speed value (no “boost” button (R1) press). Between the third and fourth corner, the Jackal was given the maximum throttle input (boost button press). After it had roughly returned to its origin, the robot would rotate to face the middle of the square, and again accelerate at full throttle to the middle of the square. The custom EKF update step was based on the input from simulated, noisy measurements to three landmarks in arbitrary positions (<10 metres in either axis direction). Measurements from the IMU were not integrated into the custom EKF implementation. Figure 15 describes the testing motion of the Jackal.

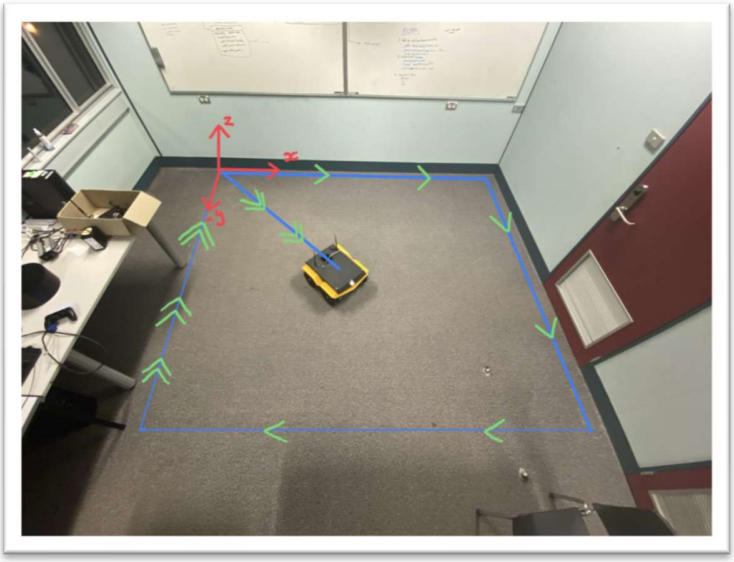


Figure 20: Experimental route performed by Jackal (with minor deviations due to manual control). Double chevrons indicate where maximum throttle input was applied. Note the y axis is pointed in the negative direction.

Following on from the results of the simulation experiments, the Jackal implementation was given the same velocity gain value of 0.64.

Figure 21 shows the Jackal's first completed run of the circuit and compares the pose estimation performance of the internal EKF to the custom EKF with the aforementioned velocity gain.

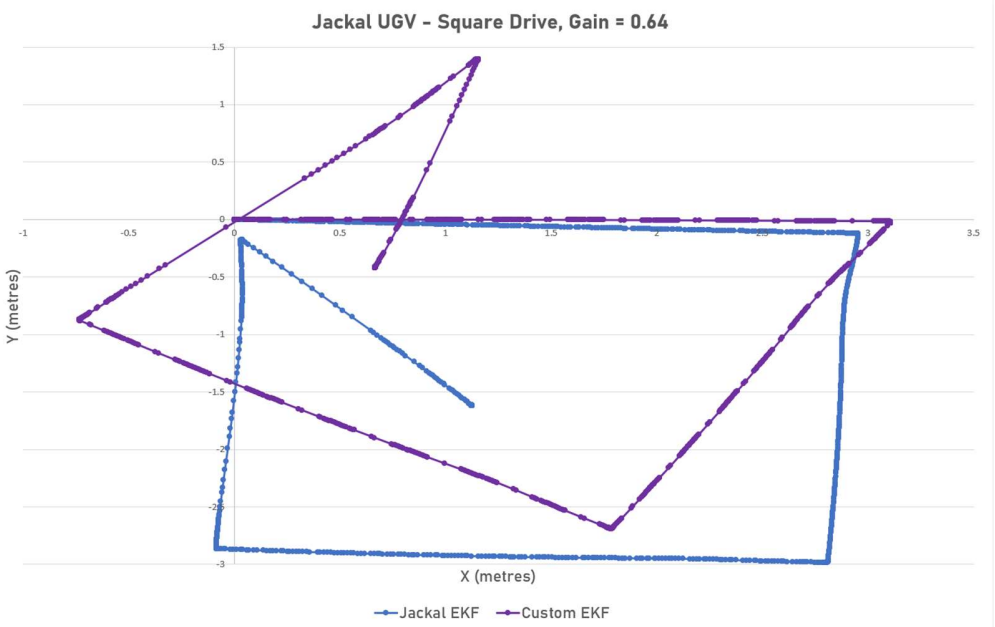


Figure 21: Plotting the estimated robot position of the Jackal EKF and the custom EKF outputs it travels in a rough square.

Eight further runs of the square route were performed. Like the simulation experiments, the velocity gain was experimentally tuned throughout these runs. The observation was made that when the gain was set right for the linear velocity, the angular velocity was slightly off, and vice versa. Figure 22 demonstrates how the gain value, whilst providing a good estimation of the true linear velocity, is providing too much of an increase to the angular velocity.

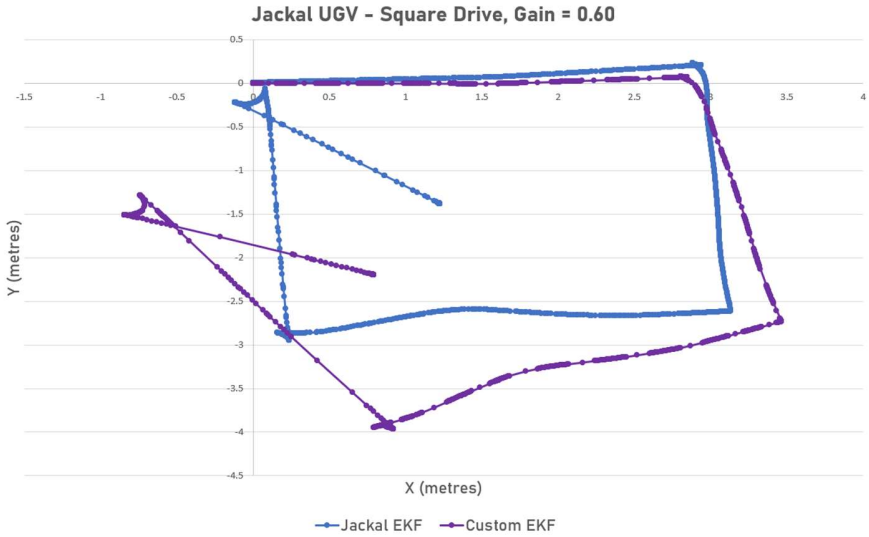


Figure 22: Plotting the estimated robot position of the Jackal EKF and the custom EKF with a tuned gain for the linear velocity only.

The single velocity gain was then substituted for individual gains applied to both the linear and angular velocities within the control input matrix *u(t)*. Additionally, the measurement and process noise covariances were tuned based on observed performances. These tunings primarily consisted of a slight increase in the robot's orientation process covariance value and decrease in initial position covariances.

Figure 23 shows the final run of the Jackal with tuned covariances and linear and angular velocity gains. Only the custom EKF output is shown as the internal EKF had, for an unknown reason, initialised its orientation to have changed, skewing the pose output. This may have occurred due to a measurement from the IMU when the robot was moved from the desk to its starting position, even though the robot had not been orientated around its Z axis.

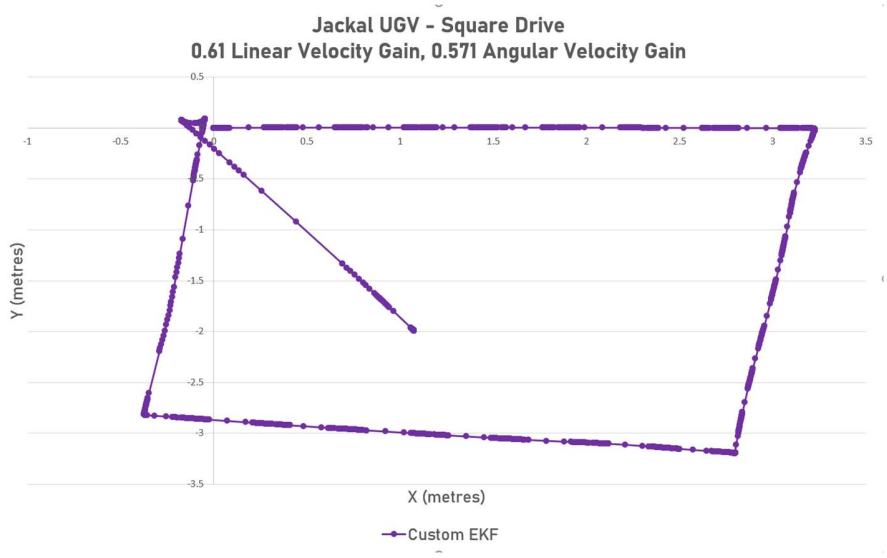


Figure 23: Plotting the estimated robot position of the custom EKF outputs with tuned velocity input gains and noise process covariances.

With the tuned velocity gains, the pose estimation accuracy of the Jackal greatly improves. As such, any future filter implementation on the Jackal UGV should ensure that the state transition model or process noise covariance matrix accounts for a potential difference in control input and actual actuated linear and angular velocity. It is recommended that the source of this velocity difference be determined and accounted for in simulation prior to any filter implementation being installed onto the Jackal UGV.

5.3 Preparing the Jackal

Future filter implementation experiments will likely involve additional sensors and other hardware. This section provides information and sources on how to configure the Jackal for additional hardware and ensure the Python version currently installed on the Jackal meets the requirements of a future GAME Filter implementation.

5.3.1 Configuring the Jackal

At boot, the Jackal runs the launch files declared in the file */opt/ros/kinetic/ros.d*. Currently, only two files are being executed on ROS startup: *base.launch* (from the *jackal_base* package) and *accessories.launch* (from the *jackal_bringup* package). These files are responsible for connecting launching the key Jackal software packages and hardware drivers, respectively. Given the necessary nodes and topics of the Jackal relevant to a filtering implementation are

not running until these scripts have launched, it is not recommended to add additional launch scripts to the *ros.d* file unless the custom package directly impacts or is in addition to the Jackal/ROS boot sequence. A more detailed outline of how the Jackal boots and how to modify this sequence is presented in [56]. The contained launch files, however, can be modified to run the drivers for added accessories, such as a camera, front laser or another accessory [40]. There are four configurations that are pre-configured for the more popular accessories. These configurations can be called by changing the *config* argument in the *base.launch* file from "base" to one of the configurations stated in Section 3 of [57].

The *base.launch* script launches the foundation packages for the Jackal. The first is the *description.launch* file (from *jackal_description)*, which is responsible for declaring to ROS the hardware configuration of the Jackal. This information is primarily stored in the */urdf* folder of this package, contained in .xacro files. The configurable information in these files includes individually enabling accessories and configuring their locations on the robot (with respect to the body fixed frame). The *jackal.urdf.xacro* file declares the locations and properties of all intrinsic properties of the Jackal, including the location and geometry of each wheel. This is the primary file to modify to reflect any physical changes made to the Jackal, should there be any. A more detailed explanation of how to configure the Jackal for additional accessories can be found at [58][59].

The second and third scripts that are launched from the *base.launch* file are the *control.launch* and *teleop.launch* scripts, both from the *jackal_control* package. These launch files are both run with configuration files, also located in the *jackal_control* package. The *control.launch* file first declares a control configuration file, *control.yaml.* This file contains the diagonal elements for the initial pose covariance and the process noise (*twist_covariance_diagonal)* covariance matrix. This file also contains other hardware parameters relating to maximum velocity and acceleration of the Jackal, and a boolean variable to allow for the active publishing of the frames of each of the wheel encoders. While most of these values are pre-configured by the manufacturer, they can be modified as is seen fit for purpose. The *control.launch* file also calls an instance of an EKF node from the *robot_localization* package. The input parameters for the EKF node, including what sensor data topics (same as declared in Table 1) to feed into the filter, are declared in the *robot_localization.yaml.* The method and considerations to add additional topics to this filter are outlined in [59]. The *teleop.launch* file launches the nodes relating to the input control of the robot; particularly, to the mapping of a joystick to velocity inputs for the Jackal. The launch file calls an instance of the *teleop_twist_joy* node [60], with the parameters for the node set by the *teleop_ps4.yaml* or

teleop.yaml file; depending on whether the user is controlling the robot with a Playstation 4 controller or a different controller, respectively.
Appendix B provides a graphical representation of the boot sequence of the Jackal and how each of the primary packages relevant to the filtering applications are launched. Michael Wiznitzer of Northwestern University [61] provides an example of how he modified all these configuration files for a desired SLAM application.

5.3.2 Updating Jackal Python Distribution

Several ROS dependencies currently installed on the Jackal, including several packages required to use the Jackal from Clearpath, contain scripts that have been developed using Python 2.7. This issue was faced in my Linux virtual machine running the same ROS version and packages, when it was advised that all development should be performed on Python 3.7.3. This was because the GAME Filter simulations are currently being developed using Python 3.7.3 and respective dependencies. For this reason, the virtual machine that was used for development was updated to accommodate this newer Python version. Upon installing the newer dependencies, a large majority of the *jackal* packages were automatically removed to accommodate the underlying update. They were then reinstalled on top of the Python 3.5.2 dependencies from their subsequent git repositories. Due to time constraints, the Jackal was not updated to the required Python version. The package was successfully loaded onto the Jackal and the primary EKF node script was rewritten with Python 2.7 syntax so that it could be compiled using the Python distribution installed on the Jackal. Instructions on how to update the ROS packages on the Jackal to Python 3.7.3 will be included in the handover repository.

5.4 Implementing a Filter onto the Jackal

This section outlines the requirements and method for a future GAME Filter implementation for both a physical and simulated Jackal.

5.4.1 GAME Filter Implementation Requirements

The system requirements for a future GAME Filter implementation were declared based on discussions with ANU PhD candidate Jack Henderson, who will be leading the future implementation and validation of the GAME Filter on the Jackal.

The requirements of a GAME Filter implementation are as follows:

- The implementation must be its own standalone package that can be installed onto an arbitrary number of robots.
- The package must have a param file for configuring the filter's static parameters, such as initial covariance values and process/noise covariance values can be modified without needing to modify the primary GAME Filter node script.
- The package must be able to subscribe to an arbitrary amount of topics for input information. By default, they must be configured to subscribe to the relevant information provided by the Jackal.
- The filter's process model takes an input vector u, a 6x1 vector which contains the 3-dimensional linear and angular velocity inputs of the robot as its entries.
- The filter processing and output must conform to the ROS coordinate frame standards as per *REP-105: Coordinate Frames for Mobile Platforms* [55]
- The package must be able to publish the state estimation output onto a topic.

Appendix B provides a graphical representation of the boot sequence of the Jackal and how each of the primary packages relevant to the filtering applications are launched. Michael Wiznitzer of Northwestern University [61] provides an example of how he modified all these configuration files for a desired SLAM application.

5.4.2 Creating a GAME Filter Package

When creating a GAME Filter package, it is recommended that the package be developed, troubleshooted and simulated on a Ubuntu 16.04 virtual machine running ROS Kinetic [43] with all of the relevant Jackal packages installed [62] and a *catkin* workspace initialized [63]. This section will detail the process what a GAME Filter package should look like and how to create one from scratch. A skeleton package will be provided in the handover repository for further development.

The first step is to create and configure a blank ROS *catkin* package [64]. Additional blank directories should be created within the package such that the package directory contains the same subdirectory layout as outlined at the beginning of Section 5.2. Once subsections 5.3.3a) - 5.3.3c) have been completed, it is important to run the compile the package in the *catkin* workspace using the *catkin_make* command and source the *catkin* setup file as instructed in [64].

a) Configuration (YAML) File

Create a file with the *.yaml* extension within the /*config* subdirectory. This configuration file should ideally contain the parameters and variables of filter that are most likely to be manipulated. This will allow the manipulation and editing of the script implementing the GAME Filter, which will be significantly larger than the configuration file, without having to edit the filter script itself. Suggested parameters are to be the input topics for measurement data, initial, process and measurement covariances and the refresh rate of the filter and/or the callback functions for the topics. It is recommended to include the topics listed in Table 1 as variables in the configuration file.

The configuration file that was used for the custom EKF can be found in Appendix C. More detailed examples of the format for a parameter file can be found within the *config* folder of the *jackal_control* package [64]. Parameters in the local configuration file are called within scripts between single quotes and with a tilde prior to the parameter name (e.g. '~pose_covariance_matrix'). The configuration file will need to be declared as a rosparam when launching the node script in order for the reference variables within the node script to relate to the configuration file. This process is explained for this application in more detail in Section 5.3.3c, and a more general explanation of the use of the YAML format in launch files is detailed in [65].

b) GAME Filter Node

Create a Python file in the */src* subdirectory. This script is where the class for the GAME Filter will need to be implemented. Like with many filter implementations, such as the GAME Filter simulation scripts currently being developed by Jack Henderson, this class will likely include *predict* and *update* functions. This script should also contain a function that subscribes to all topics outlined in Table 1, and another function to publish the state output from the filter onto a new topic.

Appendix D is an example of a skeleton for a node script as described above. The *start* function creates an instance of the GAME Filter; setting the initial state and covariance values and initializes a thread for the *testingThread* function to ensure continuous state publishing. The *start* function also initializes the subscriptions to the desired data topics, based on the parameters declared in the configuration file, and the topic on which to publish the state. The *callback* functions are required for the subscribed topics to execute the filter functions based on incoming topic information, such as performing a state prediction from a velocity input from the odometers, IMU or teleop controller, or a prediction update from an external

measurement topic such as a LiDAR accessory. On each iteration set by a refresh frequency, the *testingThread* function extracts the state from the filter object and compiles a *nav_msgs/Odometry* message, which is then published the declared topic. The *stop* function is triggered by a user input or when the node is killed and ceases the *testingThread* thread.

c) Launch File

Create a file with the *.launch* extension within the */launch* subdirectory. ROS launch files are written in XML format. The launch file should call an instance of the GAME Filter node using the *node* tag. The name of the node as it appears in ROS is declared in this tag. Within the node tag, the desired configuration file should be declared using the *rosparam* tag. Additionally, individual parameters can be declared for a node script using the *param* tag. Whether these individual parameters override any parameters a declared configuration file was not verified. Multiple instances of a filter node can be launched within the launch file. Clearpath and others [66][67] provide a more detailed explanation of launch files and the use of tags. Appendix E contains the launch file used for the custom EKF implementation.

5.4.3 Launching a GAME Filter Node

On a Ubuntu virtual machine running ROS with the configuration outlined in Section 5.3.3, an instance of the Jackal can be launched into a Gazebo simulation environment [42]. Once the Gazebo simulation has finished initializing, the GAME Filter package can be launched using the command:

```
roslaunch *name of package* *name of launch file (including .launch extension)*
```

To view the output of the filter in a terminal window, echo the declared topic that was being published:

```
rostopic echo *name of topic*
```

Instructions on how to install and run the package onto the Jackal will be outlined in more detail within the handover repository.

Chapter 6 Conclusions and Further Work

A second-order minimum-energy filter, referred to as the GAME Filter, has been proposed by Zamani, Trumpf and Mahony of the ANU. The filter has shown to perform promisingly in simulation, specifically when utilised for attitude estimation. In 2019, the ANU entered a research partnership with the Defence, Science and Technology Group (DSTG) to further develop the applicable capabilities of the filter. This partnership resulted in the ANU being lent a Clearpath Jackal UGV mobile robot development platform. The next step in the development of the GAME Filter was to validate the previously simulated performance on a physical platform and observe real-world variances that may not have been accounted for in the simulation.

This thesis outlines how to implement a nonlinear filter onto the Clearpath Jackal UGV. This includes providing a summary of the background knowledge in filtering theory, mobile robot kinematics and ROS that was obtained in order to implement a filter onto the Jackal. A method of how to develop and implement a filter in ROS based on the sensor and hardware information published by the Jackal. Using this method, an Extended Kalman Filter (EKF) was implemented onto the Jackal to estimate the pose of the robot in a local, arbitrary coordinate frame. The performance of this custom implementation was compared to the internal, more refined EKF implementation that was developed by the manufacturer of the Jackal. The performance differences provided considerations for a future GAME Filter implementation. A skeleton ROS package was developed for a future GAME Filter implementation for the Jackal and will be provided in the handover repository for future development.

The author highly recommended that those who wish to develop pose and attitude estimation filters for the Jackal UGV have an equivalent background related to the theory presented in this report.

Using the method, skeleton package and implementation considerations presented by this report, the next step in the GAME Filter's development is to implement a GAME Filter class into a ROS node. This node can then be used to construct a subsequent package. This package should be tested and troubleshooted in simulations of the Jackal using Gazebo prior to implementation on the physical Jackal. The node should also aim to utilize other forms of measurement data published by the Jackal that were not utilized during this project. This includes the data published by the IMU and GPS, despite the poor position accuracy of the latter. Additional sensors and payloads should also be added to the Jackal to provide any future filter implementation with additional measurement information.

Once a GAME Filter package has been developed, the ANU's VICON testing space can be used to ensure the filter is functional and that the performance is not drastically dissimilar to the EKF currently implemented on the Jackal from the manufacturer. Having a functional implementation of a pose estimation GAME Filter on a Jackal will then allow for several possible experimental paths. One such path includes validating the collaborative localization algorithm proposed by Zamani and Hunjet; a stretch goal of the original scope of this project and a primary research topic of the Jackal Project.

Appendix A

Appendix A - *tf* Tree Diagram

odom (global coordinate frame, declared on start-up, default: origin at start-up pose.)

```
|
├── Broadcaster: /ekf_localization
├── Average rate: 49.969 Hz
├── Most recent transform: 1572579936.88 (0.016 sec old)
├── Buffer length: 4.903 sec
|
└──>base_link (body fixed frame (BFF))
   |
   ├── Broadcaster: /robot_state_publisher
   ├── Average rate: 10000.00 Hz
   ├── Most recent transform: 0.00 (no transform)
   ├── Buffer length: 0.000 sec
   |
   └──> chassis_link (BFF, default: middle of robot)
      |
      ├── Broadcaster: /robot_state_publisher
      ├── Average rate: 10000.00 Hz
      ├── Most recent transform: 0.00 (no transform)
      ├── Buffer length: 0.000 sec
      |
      └──> front_fender_link (front payload mount point, default: BFF)
      |
      ├── Broadcaster: /robot_state_publisher
      ├── Average rate: 10000.00 Hz
      ├── Most recent transform: 0.00 (no transform)
      ├── Buffer length: 0.000 sec
      |
      └──> mid_mount (middle of robot mount point, default: BFF)
         |
         ├── Broadcaster: /robot_state_publisher
         ├── Average rate: 10000.00 Hz
         ├── Most recent transform: 0.00 (no transform)
         ├── Buffer length: 0.000 sec
         |
         └──> front_mount(rear payload mount point, default: BFF)
```

```
    |
    |--- Broadcaster: /robot_state_publisher
    |--- Average rate: 10000.00 Hz
    |--- Most recent transform: 0.00 (no transform)
    |--- Buffer length: 0.000 sec
    |
    `---> rear_mount (rear payload mount point, default: BFF)
|
|--- Broadcaster: /robot_state_publisher
|--- Average rate: 10000.00 Hz
|--- Most recent transform: 0.00 (no transform)
|--- Buffer length: 0.000 sec
|
`---> imu_link (location of IMU module w.r.t BFF, default: BFF)
|
|--- Broadcaster: /robot_state_publisher
|--- Average rate: 10000.00 Hz
|--- Most recent transform: 0.00 (no transform)
|--- Buffer length: 0.000 sec
|
`---> navsat_link (location of GPS module w.r.t BFF, default: BFF)

|
|--- Broadcaster: /robot_state_publisher
|--- Average rate: 10000.00 Hz
|--- Most recent transform: 0.00 (no transform)
|--- Buffer length: 0.000 sec
|
`---> rear_fender_link (rear payload mount point, default: BFF)
|
|--- Broadcaster: /robot_state_publisher
|--- Average rate: 31.021 Hz
|--- Most recent transform: 1572579936.88 (0.002 sec old)
|--- Buffer length: 4.900 sec
|
`---> front_left_wheel_link (front left wheel w.r.t BFF)
|--- Broadcaster: /robot_state_publisher
|--- Average rate: 31.021 Hz
|--- Most recent transform: 1572579936.88 (0.002 sec old)
|--- Buffer length: 4.900 sec
|
`---> front_right_wheel_link (front right wheel w.r.t BFF)
```

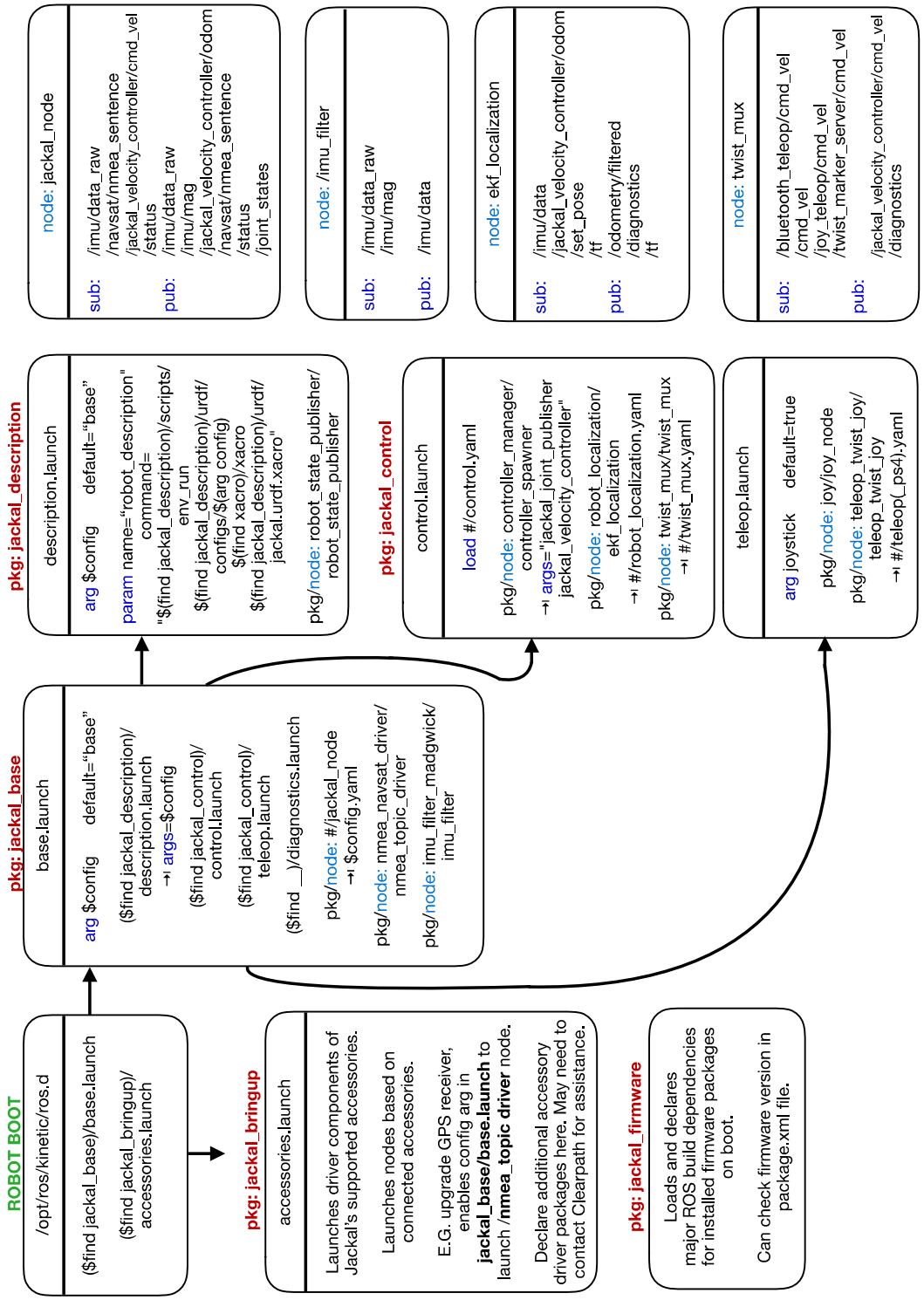
```
|
├── Broadcaster: /robot_state_publisher
├── Average rate: 31.021 Hz
├── Most recent transform: 1572579936.88 (0.002 sec old)
├── Buffer length: 4.900 sec
|
└──> rear_left_wheel_link (rear left wheel w.r.t BFF)
|
├── Broadcaster: /robot_state_publisher
├── Average rate: 31.021 Hz
├── Most recent transform: 1572579936.88 (0.002 sec old)
├── Buffer length: 4.900 sec
|
└──> rear_right_wheel_link (rear right wheel w.r.t BFF)
```

To adjust the values/connections of these frames, and for the wheel frame origin locations with respect to the BFF, see file /jackal/jackal_description/urdf/jackal.urdf.xacro

Appendix A: A transform or *tf* tree diagram describing the relationships between all declared coordinate frames running on the Jackal UGV.

Appendix B

Appendix B: Code Flow Diagram describing the nodes related to filtering that launch on ROS start-up, the launch arguments and their respective subscribed and published topics.



Appendix C

GAME Filter Node Skeleton Script (game.py)

```
1.  #!/usr/bin/env python3.5
2.
3.  __author__ = 'Alex Ollman'
4.
5.  import sys
6.  import math
7.  import time
8.  from threading import Thread
9.
10. import rospy
11. import numpy as np
12.
13. from geometry_msgs.msg import PoseWithCovarianceStamped, Vector3Stamped, Twist
14. from nav_msgs.msg import Odometry
15. from sensor_msgs.msg import Imu, NavSatFix
16.
17.
18. class GAMEFilter:
19.
20.     def __init__(self, x_0, P_0, Q, R):
21.     # Initial state
22.         self._x = x_0
23.         self._P = P_0
24.
25.         self.Q = Q
26.
27.         self.R = R
28.
29.     def predict(self, u, dt):
30.
31.         #GAME Predict
32.         return self._x, self._P
33.
34.
35.     def update(self, H, z):
36.
37.         #GAME Update
38.         return self._x, self._P
39.
40.
41. class quaternionConversion:
42.
43.     def euler_to_quaternion(self,roll, pitch, yaw):
44.
45.         qx = np.sin(roll/2) * np.cos(pitch/2) * np.cos(yaw/2) - np.cos(roll/2) * np.sin
    (pitch/2) * np.sin(yaw/2)
46.         qy = np.cos(roll/2) * np.sin(pitch/2) * np.cos(yaw/2) + np.sin(roll/2) * np.cos
    (pitch/2) * np.sin(yaw/2)
47.         qz = np.cos(roll/2) * np.cos(pitch/2) * np.sin(yaw/2) - np.sin(roll/2) * np.sin
    (pitch/2) * np.cos(yaw/2)
48.         qw = np.cos(roll/2) * np.cos(pitch/2) * np.cos(yaw/2) + np.sin(roll/2) * np.sin
    (pitch/2) * np.sin(yaw/2)
49.
50.         return [qx, qy, qz, qw]
51.
```

```
52.
53. class FilterNode:
54.
55.
56.     def cmdvelCallBack(self, cmd_vel):
57.         self.cmd_vel = cmd_vel
58.
59.     def odomCallBack(self, odom):
60.         self.odom = odom
61.
62.         #Odometry (message type: nav_msgs/Odometry)
63.         self.odomTimeStamp, self.odomLinVel, self.odomAngVel = self.odom.header.stamp,
    self.odom.twist.twist.linear, self.odom.twist.twist.angular
64.
65.         new_time = float('{}.{}'.format(self.odomTimeStamp.secs, self.odomTimeStamp.nse
    cs))
66.         self.dt = new_time - self.stateTime
67.         if self.dt > 0:
68.             #Declare filter input vector from current robot velocity.
69.             u = np.array([[velGain*self.odomLinVel.x], [angGain*self.odomAngVel.z]])
70.
71.             #Run Filter
72.             self.filter.predict(u,self.dt)
73.
74.         self.stateTime = new_time
75.
76.     def imuCallBack(self, imu):
77.         self.imu = imu
78.
79.         #IMU (message type: sensor_msgs/IMU)
80.         self.imuTimeStamp, self.imuQuatOrient,self.imuAngVel,self.imuLinVel = self.imu.
    header.stamp, self.imu.orientation, self.imu.angular_velocity, self.imu.linear_accelera
    tion
81.
82.     def GPSCallBack(self, gps):
83.         self.gps = gps
84.
85.         #GPS (message type: sensor_msgs/IMU)
86.         self.gpsTimeStamp,self.gpsLat,self.gpsLong,self.gpsPosCov = self.gps.header.sta
    mp, self.gps.latitude, self.gps.longitude, self.gps.position_covariance
87.
88.
89.
90.     def testingThread(self):
91.
92.         rate = rospy.Rate(self.frequency)
93.
94.         while not self.stopping:
95.
96.             #Extracting desired variables from each of the input data streams. View the
     resources to see how to extract the desired data.
97.
98.
99.             x_t = self.filter._x
100.            P_t = self.filter._P
101.
102.            #Construct the Odometry message to publish
103.            self.odom.header.stamp = self.odomTimeStamp
104.            self.odom.header.frame_id = "odom"
105.
106.            self.odom.pose.pose.position.x = x_t[0]
```

F

```
            self.odom.pose.pose.position.y = x_t[1]
            self.odom.pose.pose.position.z = 0

            #convert euler angles to quaternions (if required)
            quat = quaternionConversion()
            odom_quat = quat.euler_to_quaternion(0,0,float(x_t[2]))
            self.odom.pose.pose.orientation.x = odom_quat[0]
            self.odom.pose.pose.orientation.y = odom_quat[1]
            self.odom.pose.pose.orientation.z = odom_quat[2]
            self.odom.pose.pose.orientation.w = odom_quat[3]

            #Setting the velocity. Can use odom or imu for this.
            self.odom.child_frame_id = "base_link"
            self.odom.twist.twist.linear.x = self.odomLinVel.x
            self.odom.twist.twist.linear.y = self.odomLinVel.y
            self.odom.twist.twist.angular.z = self.odomAngVel.z

            #Publish the message and update time step variable.
            self.publisher.publish(od)

            #Wait until next cycle.
            rate.sleep()


    def start(self, s = 1.0):

        #Subscribing node to desired topics.

        self.cmd_vel = Twist()
        self.cmdSub = rospy.Subscriber('~teleop_vel',
        Twist,
        self.cmdvelCallBack,
        queue_size=1)

        #Odometry Topic
        self.odom = Odometry()
        self.odomSub = rospy.Subscriber('~odom0',
        Odometry,
        self.odomCallBack,
        queue_size=1)

        #IMU Topic
        self.imu = Imu()
        self.imuSub = rospy.Subscriber(rospy.get_param('~imu0'),
        Imu,
        self.imuCallBack,
        queue_size=1)

        #GPS Topic
        self.gps = NavSatFix()
        self.gpsSub = rospy.Subscriber(rospy.get_param('~gps0'),
        NavSatFix,
        self.GPSCallBack,
        queue_size=1)

        #Declaring Publisher and Output Topic
        self.od = Odometry()
        self.publisher = rospy.Publisher('filter_output',
        Odometry, queue_size=1)


```

```
        #Intialization of Global Variables and Filter Object

        x_0 = np.array(['~initial_state'])
        p_0 = np.array(['~initial_cov']
        Q = np.array(['~measurement_cov'])
        R = np.array(['~process_cov'])

        self.filter = GAMEFilter(x_0,P_0)

        #Declaring arbituary landmarks from configuration file
        self.landmarks = np.array(['~landmarks'])

        self.frequency = rospy.get_param('~frequency')

        self.u = np.array([[0],[0]])

        self.stopping = False
        self.thread = Thread(target = self.testingThread)
        self.thread.start()

        print ('GAME Filter - Started')

    def stop(self):
        self.stopping = True
        while self.thread.isAlive():
            time.sleep(0.1)

            self.cmdSub.unregister()
            self.odomSub.unregister()
            self.imuSub.unregister()
            self.gpsSub.unregister()

            print ('GAME Filter - Stopped')

if __name__ == "__main__":

    rospy.init_node('game_node');

    filterNode = FilterNode()
    filterNode.start()
    try:
        rospy.spin()
    except KeyboardInterrupt:
        print('GAME Filter Stopped: Keyboard Interrupt')
    finally:
        filterNode.stop()
```

Appendix C: GAME Filter node file skeleton, *game_node.py.* This file is the main node that will run the functions of the GAME Filter upon new published measurements on subscribed topics. The functions of the *GAMEFilter* class have been left intentionally blank.

Appendix D

GAME Filter Configuration File Example

```
#Configuation for GAME Filter node

#referesh frequency (Hz)
frequency: 50

#REFERENCE FRAMES
#https://www.ros.org/reps/rep-0105.html
odom_frame: odom
base_link_frame: base_link
world_frame: odom

#State and Covariances
initial_state: [0,0,0]
initial_cov: [0.01,0,0],[0,0.01,0],[0,0,0.01]

process_cov: [0.05,0,0],[0,0.05,0],[0,0,0.05]
measurement_cov: [0.1,0,0],[0,0.1,0],[0,0,0.1]

teleop_vel: /bluetooth_teleop/cmd_vel

#Robot odometry (linear and angular velocity) stream
odom0: /jackal_velocity_controller/odom

#IMU data steam
imu0: /imu/data_raw

#GPS data stream
gps0: /navsat/fix

#GPS velocity stream
gps_vel0: /navsat/fix

#Landmarks
landmarks: [1,1],[3,7]
```

Appendix D: A mock-up configuration file for a future GAME Filter package, based on the one used for the custom ROS EKF implementation.

Appendix E

ROS Launch File Example

```
<launch>


    <node pkg="my_ekf" type="ekf.py" name="custom_ekf_localization">
        <rosparam file="$(find my_ekf)/config/game.yaml" />

    </node>
    <!--
    <node pkg="robot_localization" type="navsat_transform_node" name="navsat_transform_node" respawn="true" output="screen">

        <param name="magnetic_declination_radians" value="0.244346"/>

        <param name="yaw_offset" value="0.01"/>
        <param name="publish_gps" value="true"/>

        <remap from="/imu/data" to="/imu_data" />
        <remap from="/navsat/fix" to="/tcpfix" />
        <remap from="/odometry/filtered" to="/odom" />
    </node>
    -->

</launch>
```

Appendix E: The launch file used to create an instance of the custom EKF node, declaring the parameters for the node to be from the *game.yaml* configuration file. This file also provides an example of how to launch other internal packages with parameters, such as the GPS node from *navsat_transform_node*.

Bibliography

[1] S. Thrun, "*Robotic Mapping: A Survey*", pp 2-3, 2002. [Online]. Available: http://robots.stanford.edu/papers/thrun.mapping-tr.pdf (accessed 12 October 2019)

[2] H. Ahmad and T. Namerikawa, "Extended Kalman filter-based mobile robot localization with intermittent measurements", Systems Science & Control Engineering, Volume 1 Issue 1, pp.113-126, 2013. [Online] Available: http://doi.org/10.1080/21642583.2013.864249 (accessed 12 October 2019)

[3] B. D. O. Anderson and J. B. Moore, *Optimal filtering / Brian D. O. Anderson, John B. Moore* Prentice-Hall Englewood Cliffs, N.J, 1979.

[4] Wikipedia. (2019). Extended Kalman Filter. [Online]. Available at: https://en.wikipedia.org/wiki/Extended_Kalman_filter (accessed 16 June 2019)

[5] B.Liu, Z. Chen, X. Liu, and F. Yang, "An Efficient Nonlinear Filter for Spacecraft Attitude Estimation," International Journal of Aerospace Engineering, vol. 2014, Article ID 540235, 11 pages, 2014. https://doi.org/10.1155/2014/540235

[6] T. Ainscough, R. Zanetti, J. Christian and P. D. Spanos. "Q-Method Extended Kalman Filter", Journal of Guidance, Control, and Dynamics, 38:4, 752-760, 2015. [Online]. Available: https://doi.org/10.2514/1.G000118

[7] Wikipedia. (2019). Quaternions and spatial rotation. [Online]. Available: https://en.wikipedia.org/wiki/Quaternions_and_spatial_rotation (accessed 7 October 2019)

[8] R. B. Widodo and C. Wada. "Attitude Estimation Using Kalman Filtering: External Acceleration Compensation Considerations," Journal of Sensors, vol. 2016, Article ID 6943040, 24 pages, 2016. https://doi.org/10.1155/2016/6943040

[9] M. Zamani, "Deterministic attitude and pose filtering, an Embedded Lie groups approach", 2013 [Online] Available: http://eprints.skums.ac.ir/5588/

[10] M. Zamani, J. Trumpf and R. Mahony, "A second order minimum-energy filter on the special orthogonal group," *2012 American Control Conference (ACC)*, Montreal, QC, 2012, pp. 1895-1900.
[Online]. Available: https://doi.org/10.1109/ACC.2012.6315025

[11] M. Zamani, J. Trumpf and R. Mahony, "On the distance to optimality of the geometric approximate minimum-energy attitude filter," *2014 American Control Conference*, Portland, OR, 2014, pp. 4943-4948.
[Online]. Available: https://doi.org/10.1109/ACC.2014.6858915

[12] lumencandela. “Basics of Kinematics”, 2017. [Online] Available: https://courses.lumenlearning.com/boundless-physics/chapter/basics-of-kinematics/ (accessed 3 October 2019)

[13] R. Siegwart and I.R. Nourbakhsh. (2004). Introduction to autonomous mobile robots. MIT, 2nd Ed, p. 181.

[14] L. Mitka, “Simple kinematics for mobile robot”, *Husarion*, 2019. [Online]. Available: https://husarion.com/tutorials/ros-tutorials/3-simple-kinematics-for-mobile-robot/ (accessed 6 October 2019)

[15] A. Dattalo, “ROS / Introduction”, *ROS.org Tutorials*, 2018. [Online]. Available: http://wiki.ros.org/ROS/Introduction (accessed 24 April 2019)

[16] A. Koubaa, “ROS Tutorials - ROS Wiki”, *ROS.org Tutorials*, 2018. [Online]. Available: http://wiki.ros.org/ROS/Tutorials (accessed 24 April 2019)

[17] T. Fiorenzani, “YouTube Playlist: ROS TUTORIALS”, 2018. [Online]. Available: https://www.youtube.com/playlist?list=PLuteWQUGtU9BU0sQIVqRQa24p-pSBCYNv (accessed 24 April 2019)

[18] Griffith University. “ROS Summary Overview”. [Online]. Available: http://www.ict.griffith.edu.au/~vlad/teaching/robotics.d/RESOURCES/ROS_Summary_Overview.pdf (accessed 26 April 2019)

[19] Clearpath Robotics Inc. “Jackal UGV Tutorials”, *Jackal Tutorials*, Clearpath Robotics Inc, 2015. [Online]. Available: http://www.clearpathrobotics.com/assets/guides/jackal/index.html (accessed 22 February 2019)

[20] T. Baltovski. (2018) *jackal_robot*, GitHub repository. [Online]. Available: https://github.com/jackal/jackal_robot/ (accessed 13 August 2019)

[21] T. Moore, “robot_localization - ROS Wiki”,*ROS.org,* 2018. [Online]. Available: http://wiki.ros.org/robot_localization (accessed 13 August 2019)

[22] M. Purbis, “jackal_control - ROS Wiki”,*ROS.org,* 2018. [Online]. Available: http://wiki.ros.org/jackal_control (accessed 13 August 2019)

[23] M. Zamani, J. Trumpf and R. Mahony, "Near-optimal deterministic attitude filtering," *49th IEEE Conference on Decision and Control (CDC)*, Atlanta, GA, 2010, pp. 6511-6516. [Online]. Available: https://doi.org/10.1109/CDC.2010.5717043

[24] M. Zamani, J. Trumpf and R. Mahony, "Minimum-energy filtering on the unit circle," *2011 Australian Control Conference*, Melbourne, VIC, 2011, pp. 236-241. [Online].

[25] M. Zamani, J. Trumpf and R. Mahony, "Minimum-Energy Filtering for Attitude Estimation," in *IEEE Transactions on Automatic Control*, vol. 58, no. 11, pp. 2917-2921, Nov. 2013. [Online]. Available: https://10.1109/TAC.2013.2259092

[26] Zamani, Mohammad & Trumpf, J. & Mahony, Robert. (2015). Nonlinear Attitude Filtering: A Comparison Study. [Online]. Available: https://arxiv.org/abs/1502.03990

[27] M. Zamani and R. Hunjet, "Collaborative Pose Filtering Using Relative Measurements and Communications," *2019 12th Asian Control Conference (ASCC)*, Kitakyushu-shi, Japan, 2019, pp. 919-924.

[28] S. I. Roumeliotis and G. A. Bekey, “Distributed multirobot localization” *IEEE Transactions on Robotics and Automation*, vol. 18, no. 5, pp. 781–795, 2002.

[29] T. Babb, “How a Kalman filter works, in pictures”, *bzarg.com*, 2015. [Online]. Available: https://www.bzarg.com/p/how-a-kalman-filter-works-in-pictures/ (accessed 15 April 2019)

[30] MathWorks. (2017). Video: *Understanding Kalman Filters, Part 1: Why Use Kalman Filters?* [Online]. Available: https://www.youtube.com/watch?v=mwn8xhgNpFY (accessed 25 March 2019)

[31] H. S. Chadha, “Kalman Filter Interview”, *Towards Data Science*, 2018. [Online]. Available: https://towardsdatascience.com/kalman-filter-interview-bdc39f3e6cf3 (accessed 30 May 2019)

[32] B. Esme, “Kalman Filter For Dummies”, *Bilgin’s Blog,* 2009. [Online]. Available: http://bilgin.esme.org/BitsAndBytes/KalmanFilterforDummies (accessed 28 May 2019)

[33] H. S. Chadha, “Extended Kalman Filter: Why do we need an extended one?”, *Towards Data Science*, 2018. [Online]. Available: https://towardsdatascience.com/extended-kalman-filter-43e52b16757d (accessed 30 May 2019)

[34] S. Särkkä, “Lecture 4: Extended Kalman filter and Statistically Linearized Filter”, Department of Biomedical Engineering and Computational Science, Aalto University, 2010. [Online]. Available: https://users.aalto.fi/~ssarkka/course_k2010/slides_4.pdf

[35] R. Serrano, “Extended Kalman Filters for Dummies”, *Medium,* 2017. [Online]. Available: https://medium.com/@serrano_223/extended-kalman-filters-for-dummies-4168c68e2117
(accessed 2 July 2019)

[36] B. Nam. (2019). “Video: Extended Kalman Filter Explained With Python Code”. [Online]. Available: https://www.youtube.com/watch?v=0M8R0IVdLOI (accessed 12 June 2019)

[37] MIT CSAIL. (2001) “Robo-Rats Locomotion: Skid-steer Drive”. [Online]. Available: https://groups.csail.mit.edu/drl/courses/cs54-2001s/skidsteer.html

[38] S. Rabiee and J. Biswas. (2019). “A Friction-Based Kinematic Model for Skid-Steer Wheeled Mobile Robots”. *2019 International Conference on Robotics and Automation (ICRA)*. [Online]. Available: https://doi.org/10.1109/ICRA.2019.8794216

[39] T. Wang, Y. Wu, J. Liang, C. Han, J. Chen, Q. Zhao. (2015). “Analysis and Experimental Kinematics of a Skid-Steering Wheeled Robot Based on a Laser Scanner Sensor”. *Sensors, 2015*, *15*, 9681-9702. [Online]. Available: https://doi.org/10.3390/s150509681

[40] Clearpath Robotics Inc. “Jackal UGV - Small Weatherproof Robot”, *Clearpath Robotics Inc*, 2018. [Online]. Available at: https://www.clearpathrobotics.com/jackal-small-unmanned-ground-vehicle/ (accessed 16 February 2019)

[41] M. Nascimento, “Distributions - ROS Wiki”, *ROS.org Tutorials*, 2018. [Online]. Available: http://wiki.ros.org/Distributions (accessed 31 May 2019)

[42] Clearpath Robotics Inc. “Simulating Jackal”, *Jackal Tutorials* by Clearpath Robotics Inc, 2015. [Online]. Available: http://www.clearpathrobotics.com/assets/guides/jackal/simulation.html (accessed 16 February 2019)

[43] T. Foote, “kinetic - ROS Wiki”, *ROS.org,* 2018. [Online]. Available: http://wiki.ros.org/kinetic (accessed 16 February 2019)

[44] T. Moore and D. Stouch. (2016) A Generalized Extended Kalman Filter Implementation for the Robot Operating System. Intelligent Autonomous Systems 13. Advances in Intelligent Systems and Computing, AISC, vol 302. [Online]. Available: https://10.1007/978-3-319-08338-4_25

[45] T. Baltovski. (2019) *jackal*, GitHub repository. [Online]. Available: https://github.com/jackal/jackal (accessed 13 August 2019)

[46] T. Baltovski. (2018) *jackal_robot/jackal_base*, GitHub repository. [Online]. Available: https://github.com/jackal/jackal_robot/tree/kinetic-devel/jackal_base (accessed 13 August 2019)

[47] T. Baltovski. (2018) *jackal/jackal_description*, GitHub repository. [Online]. Available: https://github.com/jackal/jackal/tree/kinetic-devel/jackal_description (accessed 13 August 2019)

[48] Clearpath Robotics Inc. "ROS Navigation Basics", *ROS Tutorials* by Clearpath Robotics Inc, 2015. [Online]. Available: https://www.clearpathrobotics.com/assets/guides/ros/ROS%20Navigation%20Basics.html (accessed 16 Feburary 2019)

[49] S. Chitta, E. Marder-Eppstein, W. Meeussen, V. Pradeep, A. Rodríguez Tsouroukdissian, J. Bohren, D. Coleman, B. Magyar, G. Raiola, M. Lüdtke and E. Fernandez Perdomo. (2017). *ros_control: A generic and simple control framework for ROS*, The Journal of Open Source Software. [Online]. Available: http://wiki.ros.org/ros_controllers

[50] L. Marchionni, B. Magyar, E. Fernandez, P. Mathieu. (2013) *ros_controllers/drive_controller/src/odometry.cpp,* lines 140-169, GitHub repository. [Online].
Available: https://github.com/ros-controls/ros_controllers/blob/kinetic-devel/diff_drive_controller/src/odometry.cpp (accessed 14 August 2019)

[51] GlobalTop Technology. (2019). Gmm-U2P - IVORY 3: High Sensitivity & Ultra Low Power GPS Module. [Online].
Available: http://www.alphamicrowireless.com/franchises/globaltop-technology/gmm-u2p.aspx (accessed 1 November 2019)

[52] J. Schultz. (2017). "tf - ROS Wiki", *ROS.org*. [Online]. Available: http://wiki.ros.org/tf (accessed 7 August 2019)

[53] H. Matic. "tf/Tutorials/Introduction to tf - ROS Wiki", *ROS.org,* 2015. [Online]. Available: http://wiki.ros.org/tf/Tutorials/Introduction%20to%20tf (accessed 7 August 2019)

[54] Clearpath Robotics Inc. (2019). Kinetic-jackal-amg64-0.3.7.iso. [Online]. Available: http://packages.clearpathrobotics.com/stable/images/latest/kinetic-jackal/amd64/ (accessed 22 October 2019)

[55] W. Meeussen. “Coordinate Frames for Mobile Platforms”, *Index of ROS Enhancement Proposals (REPs), REP-105, 2010.* [Online]. Available: https://www.ros.org/reps/rep-0105.html#relationship-between-frames (accessed 28 October 2019)

[56] Clearpath Robotics Inc. “Extending Jackal’s Startup”, *Jackal Tutorials*, Clearpath Robotics Inc, 2015. [Online]. Available: https://www.clearpathrobotics.com/assets/guides/jackal/startup.html (accessed 22 September 2019)

[57] T. Baltovski. “jackal_description - ROS Wiki”, *ROS.org,* 2015. [Online]. Available: http://wiki.ros.org/jackal_description (accessed 21 August 2019)

[58] T. Baltovski. “jackal_description - ROS Wiki”, *ROS.org,* 2015. [Online]. Available: http://wiki.ros.org/jackal_description

[59] T. Moore and D. Stouch. (2016) robot_localization wiki. [Online]. Available at:http://docs.ros.org/kinetic/api/robot_localization/html/index.html (accessed 13 August 2019)

[60] M. Wiznitzer. (2017) “Jackal Exploration (Northwestern University Winter Project), GitHub repository. [Online]. Available: http://wiki.ros.org/teleop_twist_joy (accessed 16 July 2019)

[61] T. Baltovski. (2018) *jackal_robot/jackal_base*, GitHub repository. [Online]. Available: https://github.com/mechwiz/jackal_exploration (accessed 24 September 2019)

[62] vfdev-5. (2019) *vfdev-5/INSTALL.md*, GitHub repository. [Online]. Available: https://gist.github.com/vfdev-5/57a0171d8f5697831dc8d374839bca12 (accessed 3 November 2019)

[63] W. Woodall. “catkin/Tutorials/create_a_workspace - ROS Wiki”, *ROS.org,* 2017. [Online]. Available: http://wiki.ros.org/catkin/Tutorials/create_a_workspace (accessed 24 October 2019)

[64] T. Baltovski. (2015) *jackal/jackal_control/config*, GitHub repository. [Online]. Available: https://github.com/jackal/jackal/tree/kinetic-devel/jackal_control/config (accessed 22 September 2019)

[65] The Robotics Back-End. (2019) ROS Param YAML Format. [Online]. Available: https://roboticsbackend.com/ros-param-yaml-format/ (accessed 3 November 2019)

[66] V. Ivan, Y. Yang, W. Merkt, M. Camilleri, S Vijayakumar. "Setting up ROSlaunch", *EXOTica*, 2019. [Online]. Available: https://ipab-slmc.github.io/exotica/Setting-up-ROSlaunch.html (accessed 3 November 2019)

[67] Clearpath Robotics Inc. "Launch Files", *ROS Tutorials* by Clearpath Robotics Inc, 2015. [Online]. Available: http://www.clearpathrobotics.com/assets/guides/ros/Launch%20Files.html (accessed 3 November 2019)