

NAVIGATION SOLUTIONS FOR AN UNMANNED SURFACE VEHICLE

A THESIS SUBMITTED TO THE GRADUATE DIVISION OF THE UNIVERSITY OF
HAWAI'I AT MANOA IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE
DEGREE OF

MASTER OF SCIENCE

IN

MECHANICAL ENGINEERING

NOVEMBER 2022

By

Kelsey T. Kim

Thesis Committee:

A Zachary Trimble, Chairperson
Michael Krieg
Peter Berkelman

ACKNOWLEDGMENTS

I would like to thank my advisor, Dr. A Zachary Trimble, for providing me with so much guidance throughout my time as a Master's student. You have not only provided me your technical expertise, but have also taught me how to approach problems and think like an engineer. I am so grateful for all of the time you have spent setting up and guiding field tests, asking me difficult questions about my work, editing my work, and patiently explaining concepts that should already be second nature. Thank you so much for your mentoring, and again, patience, over the past couple of years.

Thank you to my thesis committee, Professor Michael Krieg and Professor Peter Berkelman, for your flexibility, time, and support. Your words of encouragement and interest in my work have been very much appreciated.

Thank you to the Office of Naval Research for funding my research for the past few years.

Thank you to my lab-mates, Kai Jones and Kainani Santos, for acting as a sounding board and always taking the time to help me think out any problems I have. Without you two, so much of what I have produced and learned would not have been possible.

Lastly, I would like to thank my family for providing me with the opportunity to continue my studies and pursue a field that I enjoy. You have done everything possible to set me up for success and I am so grateful for all of your love, support, and patience.

ABSTRACT

In order to begin developing autonomy on an unmanned surface vehicle (USV), an accurate, precise, and robust navigation solution must first be implemented, as it is the basis for control and guidance algorithms. Without a direct velocity measurement, it is difficult to obtain an estimate of the USV's surge and sway velocities, as these are the least observable states. In this work, two navigation methods are compared to determine the effects of incorporating the vehicle dynamics of a USV into an Extended Kalman Filter (EKF). The first navigation solution utilizes an EKF with a state propagation model that incorporates a linear three degree-of-freedom dynamic positioning model for USVs. This model characterizes the USV's dynamics and environmental disturbances. The second method is the open-source Robot Operating System (ROS) *robot_localization* package, which utilizes an EKF with a state propagation model that utilizes a generic omnidirectional robot model. The same sensor data is fed through each navigation solution and results are compared to a ground truth position, orientation, and velocity in simulation. The resulting estimates from both navigation solutions are also compared using real world sensor data from a USV. The noise and deviation from the ground truth for each navigation output are compared. Results indicate that incorporating vehicle dynamics into the EKF significantly reduces error in the less-observable states, the surge and sway velocities. For the two most observable states, the heading and angular velocity, the two filters have similar low errors. Incorporating vehicle dynamics into the EKF increases the noise in the estimates when compared to the generic omnidirectional EKF, but allows finer changes in the actual state to be captured more accurately and with less latency.

CONTENTS

ACKNOWLEDGMENTS	i
ABSTRACT.....	ii
CONTENTS.....	iii
LIST OF TABLES	vi
LIST OF FIGURES	vii
CHAPTER 1: INTRODUCTION.....	1
1.1 Objective	1
1.2 Significance.....	1
1.3 Literature Review.....	2
1.3.1 Doppler Velocity Log	2
1.3.2 Kalman Filter	3
1.3.3 Particle Filter.....	3
1.3.4 Extended Kalman Filter with Nonlinear Vehicle Model	4
1.3.5 Interpolation Based Filters	4
1.4 Thesis Overview.....	4
CHAPTER 2: VEHICLE DYNAMICS.....	6
2.1 Coordinate Frames	6
2.2 Linear Three Degree of Freedom Model	7
2.2.1 Simplifications	7
2.2.2 Vector Representation.....	8
2.2.3 Mass Matrix	9
2.2.4 Damping.....	10
2.2.5 Thruster Model.....	11
CHAPTER 3: EXTENDED KALMAN FILTER.....	12
3.1 Extended Kalman Filter Algorithm	12
3.2 State Propagation Model.....	13
3.3 Measurement Model	17

3.4 Covariance Tuning.....	20
3.5 Observability.....	21
CHAPTER 4: IMPLEMENTATION	22
4.1 Physical Components.....	22
4.1.1 Wave Adaptive Modular Vessel	22
4.1.2 Thrusters	22
4.1.3 Sensors	23
4.1.3.1 RTK GNSS	23
4.1.3.2 Inertial Measurement Unit	29
4.1.3.3 Sensor Specifications	31
4.2 Software	31
4.2.1 Differential GPS Bearing Calculation	32
4.2.2 Transformation from Latitude/Longitude to NED.....	33
4.2.3 Wrench Calculation	33
4.2.4 Parameter Files.....	35
4.2.5 Extended Kalman Filter Algorithm and ROS Wrapper.....	37
CHAPTER 5: ROBOT_LOCALIZATION	39
5.1 Background.....	39
5.2 Configuration	39
5.2.1 EKF Localization Node	39
5.2.2 Navsat Transform Node.....	41
CHAPTER 6: PERFORMANCE COMPARISON	43
6.1 Virtual RobotX Background.....	43
6.2 Results.....	43
6.2.1 Straight Line Maneuver Simulation.....	43
6.2.2 Wave-Like Maneuver Simulation.....	47
6.2.3 Random Maneuver Simulation	51
6.2.4 Real World Straight Line Maneuver.....	54

6.2.5 Real World Curve Maneuver	59
6.3 Discussion	62
CHAPTER 7: CONCLUSION	66
REFERENCES	67
Appendix A: robot_localization Parameters	70

LIST OF TABLES

Table 1: Sensor Specifications.....	31
Table 2: Parameters defined in baseParameters.yaml.....	35
Table 3: Parameters defined in kalmanParameters.yaml.....	36
Table 4: EKF covariances defined in kalmanParameters.yaml.....	37
Table 5: Observability Ranking.....	63
Table 6: Average filter covariance.....	64
Table 7: Root-mean-squared error for simulated maneuvers.....	65
Table 8: Mean distance error for simulated maneuver in meters.....	65

LIST OF FIGURES

Figure 1: NED/map frame and body-fixed coordinate system	7
Figure 2: WAMV collecting data at Sand Island, Hawaii	22
Figure 3: Rear view of WAMV with Minn Kota motors.....	23
Figure 4: SparkFun ZED-F9P RTK GPS board (left) and Taoglas antenna (right)	24
Figure 5: GNSS rover antennae and IMU onboard WAMV	25
Figure 6: Histogram of latitude measurements with RTK corrections	26
Figure 7: Histogram of longitude measurements with RTK corrections	26
Figure 8: Histogram of latitude measurements without RTK corrections.....	27
Figure 9: Zoomed in histogram of latitude measurements without RTK corrections	27
Figure 10: Histogram of longitude measurements without RTK corrections.....	28
Figure 11: Zoomed in histogram of longitude measurements without RTK corrections	28
Figure 12: ADIS16470 IMU with breakout board.....	29
Figure 13: Histogram of stationary angular velocity measurements	30
Figure 14: Histogram of stationary x-acceleration measurements	30
Figure 15: Histogram of stationary y-acceleration measurements	30
Figure 16: Software flow chart for WAMV Extended Kalman Filter implementation.....	32
Figure 17: Transformation tree for robot_localization	40
Figure 18: robot_localization software flow chart with sensor editor script	41
Figure 19: Straight line simulation trajectory	44
Figure 20: Straight line simulation yaw orientation	45
Figure 21: Straight line simulation surge velocity	45
Figure 22: Straight line simulation sway velocity	46
Figure 23: Straight line simulation angular yaw velocity	47
Figure 24: Wave maneuver simulation trajectory.....	48
Figure 25: Wave maneuver simulation yaw orientation.	48
Figure 26: Wave maneuver simulation surge velocity	49
Figure 27: Wave maneuver simulation sway velocity.....	50
Figure 28: Wave maneuver simulation angular yaw velocity	50
Figure 29: Random maneuver simulation trajectory	51

Figure 30: Random driving simulation yaw orientation	52
Figure 31: Random driving simulation surge velocity	53
Figure 32: Random driving simulation sway velocity	53
Figure 33: Random driving simulation angular yaw velocity	54
Figure 34: Straight line trajectory for real world	55
Figure 35: Straight line yaw orientation for real world	56
Figure 36: Straight line surge velocity for real world	57
Figure 37: Straight line sway velocity for real world	58
Figure 38: Straight line angular yaw velocity for real world	58
Figure 39: Curve maneuver trajectory for real world	59
Figure 40: Curve maneuver yaw orientation for real world	60
Figure 41: Curve maneuver surge velocity for real world	61
Figure 42: Curve maneuver sway velocity for real world	61
Figure 43: Curve maneuver angular yaw velocity for real world	62

CHAPTER 1: INTRODUCTION

1.1 Objective

The purpose of this work is to compare the performance of the established *robot_localization* state estimation software package that does not utilize the unmanned surface vehicle's (USV) dynamics to a real-time state estimator that does incorporate the USV's dynamics. The goal of this state estimator is to improve the robustness, precision, and accuracy of the USV's estimated position, orientation, linear velocity, and angular velocity by incorporating the USV's dynamics, which are not subject to kinematic constraints. The most important components of the state vector are the linear velocities and the orientation, as many proven controllers and guidance algorithms are designed around these components and a direct velocity measurement can be difficult to obtain. A precise and accurate state estimate is crucial for a completed guidance, navigation, and control (GNC) scheme, as poor estimates will propagate into the guidance and control algorithms.

1.2 Significance

Advancements in the autonomous capabilities of USVs have the potential to aid environmental efforts addressing ocean pollution through scanning for trash detection and clean-up. Plastic ocean pollution exacerbates biotic mixing, allowing alien species to invade natural marine ecosystems. A 58% decrease in global marine species diversity is estimated, should this biotic mixing continue[1][2]. Plastic ocean pollution further harms marine life through ingestion and entanglement. In the years 1982-1998, 173 cases of Hawaiian monk seal entanglement were documented on the Hawaiian Islands [3]. In 2017, six beaches on the windward coast of O'ahu, Hawai'i were surveyed and found to contain microplastic densities ranging between 700-1700 particles per square meter[4]. As of 2014, there was an estimated minimum of 5.25 trillion plastic particles, or 268,940 tons of plastic pollution in the ocean globally[5]. USVs have been used in several instances to successfully scan and characterize water quality and ocean pollution, as well as autonomously clean the surface of the water[6]–[9]. In one application, an

autonomous surface vehicle (ASV) was used to quantify the abundance and detect the behavior of zooplankton[10]. When the same characterization was attempted with other sampling platforms, the introduction of artificial light made sampling impossible for depths less than 100m.

In addition to aiding with environmental efforts, USVs can be used for munitions detection, a task that could otherwise put human operators at risk. Roughly 32,000 tons of munitions were dumped at sea in the years 1919-1970[11]. Of the 32,000 tons, roughly 2,700 tons of munitions are located off the shores of the Hawaiian Islands[12]. As shorelines are extended farther into the ocean, human injury due to munitions have occurred during the dredging process of development. USVs could automate ocean floor scanning and munitions detection, reducing risks to human workers. Improving the accuracy of the USV's state estimates will improve the accuracy of the locations of the detected ocean pollution and munitions. With reduced uncertainty in the USV's state estimate, the data analysis process will be expedited, as less effort is needed to reduce noise in location estimates. Additionally, guidance and control algorithms will have more accurate and precise inputs, allowing more accurate maneuvering and potentially decreasing the USV's energy consumption during ocean operations.

1.3 Literature Review

1.3.1 Doppler Velocity Log

One of the most straightforward methods of achieving a velocity estimate is a velocity sensor. A popular sensor used with unmanned underwater vehicles (UUV) and USVs is the Doppler Velocity Log (DVL). This sensor provides a velocity estimate of the vehicle with respect to the ocean floor by emitting acoustic beams towards the ocean floor. The beams are echoed back to the DVL, with a change in frequency due to the sensor's motion. The changes in pitch are used to determine the sensor's velocity. DVLs have been shown to provide velocity measurements with variances on the scale of less than 0.008 m/s[13]. They have also been shown to improve velocity and localization estimates when combined with both dead reckoning techniques and filtering methods[14]–[18]. While the DVL is a simple solution to obtaining an estimate of the USV's velocity, DVL sensors are expensive and less costly methods of determining the USV's state exist.

1.3.2 Kalman Filter

The Kalman Filter is a popular algorithm used for state estimation that utilizes a prediction and measurement update step[19][20]. During the prediction step, a state propagation model is defined to predict how the state changes between each time step. This prediction is updated using sensor measurements. The error between the final estimate and the predicted estimate, as well as propagation model covariance and measurement covariance are used to determine the noise associated with each estimate. The Kalman Filter's key assumption is that the noise associated with the measurements are Gaussian. While this assumption applies to the system in this work, the Kalman Filter also requires a linear state propagation model. In this work, a nonlinear state propagation model is used with a variant of the Kalman Filter, the Extended Kalman Filter (EKF). The EKF is discussed in the following sections.

1.3.3 Particle Filter

Similar to Kalman Filters, the particle filter achieves accurate state estimation through a prediction and update step[21]–[23]. A specified number of particles taken at the previous estimation step for each element in the state vector are propagated through a prediction model. Each particle is associated with a weight to specify its likelihood of being the true state. When a measurement is received, the particles are updated with new weights. Particles with low weights are removed and replaced by those with higher weights. The Particle Filter is not limited to sensors and models with Gaussian noise distributions. Because each element in the state vector must be recalculated for each particle associated with it, the Particle Filter is more computationally expensive than the Kalman Filter and takes more time to produce a valid estimate due to its need for extensive resampling. This heavy computation is exacerbated by larger state vectors. In an experiment comparing the performance of an EKF to the performance of a Particle Filter with 15,000 particles for localization of an autonomous underwater vehicle (AUV), the Particle Filter provided a smoother trajectory estimate[24]. The pitfall to the Particle Filter was that it required a mean of 0.04445s to produce an estimate. The EKF required a mean of 0.00007956s to produce an estimate. For non-real time position estimation, the Particle Filter is a valid option, as it provides improved accuracy and precision in its estimates, but for real-time state estimation, better estimation options are available.

1.3.4 Extended Kalman Filter with Nonlinear Vehicle Model

As seen in Section 1.3.3, the EKF provides comparable accuracy and precision in its produced estimates with less computational time requirements when compared to the Particle Filter. The EKF algorithm is the same as the Kalman Filter algorithm and is restricted to the same assumption of a Gaussian noise distribution, but allows the use of nonlinear models[25]. The nonlinear model is linearized by finding the Jacobian of the nonlinear model and is then included as the state propagation model of the Kalman Filter. In this case, the benefit of using a nonlinear model is the inclusion of higher order damping terms, cross-coupling terms, and sinusoidal relationships. These added details result in a more accurate model that is applicable for most maneuvers. For this work, a simplified linear model is used, as higher order damping terms approach zero at speeds less than 2m/s, which is roughly the maximum speed of the USV used in this thesis[26].

1.3.5 Interpolation Based Filters

Filters for state estimation of nonlinear systems based on polynomial approximations obtained through interpolation are a popular alternative to the EKF. These filters follow a similar prediction and update algorithm as the Kalman Filter, but differ in how they handle the nonlinearity of the system and its associated covariance. Rather than approximating a nonlinear system using the Taylor series, as with the EKF, the system is linearized using interpolation. This is to avoid singularities associated with derivatives and for simpler implementation by avoiding derivations of the Jacobian. This method has been shown to provide covariance outputs of higher accuracy and state estimates of similar accuracy when compared to the EKF[27][28]. Despite simpler implementation, practical computational requirements are similar to those of the EKF.

1.4 Thesis Overview

In chapter two the mathematical hydrodynamic model based on the linear three degree of freedom (3DOF) dynamic positioning model by Fossen is defined for the USV. Chapter three describes the EKF algorithm and the state propagation model derived from the hydrodynamic model of chapter two, as well as the measurement model used in this application. Chapter four

discusses the EKF implementation including the USV used in testing, the sensors used, and the software structure. In chapter five a brief overview of the *robot_localization* package is given with its configuration for the sensor set and vehicle used in testing. The performance of each navigation solution is summarized and compared in chapter six. Finally, concluding remarks and future work is discussed in chapter seven.

CHAPTER 2: VEHICLE DYNAMICS

In this chapter the mathematical USV model is defined. Adjustments are made to the linear 3DOF DP model by Fossen to create a relationship between the vessel's velocity components, acceleration components, and control inputs[26]. This model takes the vessel's physical characteristics into account using hydrodynamic damping and added mass terms.

2.1 Coordinate Frames

The model of the USV's vehicle dynamics utilizes three coordinate frames. The first is the inertial NED frame, located on the surface of the Earth with its positive x-axis pointing North, positive y-axis pointing East, and positive z-axis pointing downwards towards the center of the Earth. The NED frame can be located at an arbitrary point, as long as it is defined. For this work, the NED frame is located at the ground station. The NED frame is also known as the map frame.

The second coordinate frame used in this model is the body-fixed coordinate frame, also known as the body frame. The location of this coordinate frame moves with the vehicle and can be defined anywhere on the body of the vehicle that is convenient for GNC implementation. For this work, the body frame is located at the aft edge of the vehicle's platform and centered between the port and starboard platform edges. The body frame moves with the vehicle so that its location on the vehicle remains fixed with its positive x-axis pointing towards the front of the vehicle, positive y-axis pointing starboard, and positive z-axis pointing downwards. The two reference frames are depicted in Figure 1.

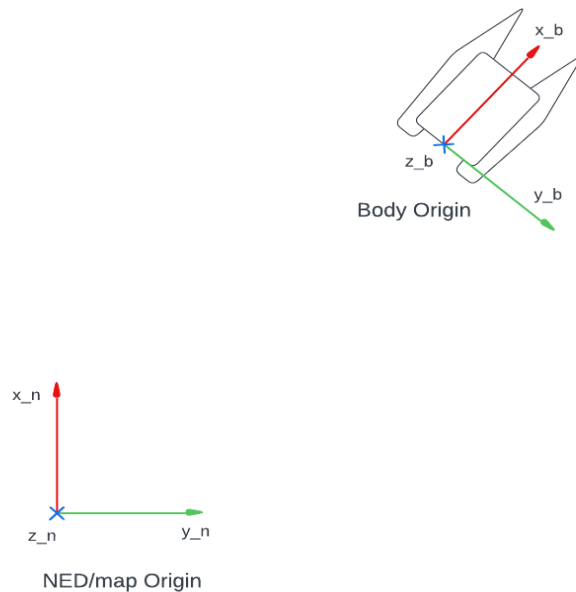


Figure 1: NED/map frame and body-fixed coordinate system

2.2 Linear Three Degree of Freedom Model

2.2.1 Simplifications

The model used in this work is ideal for slow speed and stationkeeping applications in which the vessel's objective is to maintain a desired position and orientation. It was selected for its simplified linearity and three degrees of freedom, as opposed to other models incorporating the full six degrees of freedom (6DOF) with nonlinear terms. Under the slow speed assumption, the vessel's dynamics can be simplified to produce a linear model, as higher order terms approach zero at speeds less than 2m/s. Because the vessel used in this application operates on the surface of the ocean in environments with glassy to rippling waters (Douglas sea state 0-1), linear motion in the heave direction and angular motion in the roll and pitch directions are minimal. With minimal pitching, rolling, and heave motion, the vessel's dynamics can be simplified to only address angular motion in the yaw direction and linear motion in the surge and sway directions.

2.2.2 Vector Representation

The 3DOF DP Model can be represented in vector form as:

$$\dot{\bar{\eta}} = \bar{R}(\psi)\bar{v} \quad (1)$$

$$\bar{M}\dot{\bar{v}} + \bar{D}\bar{v} = \bar{R}^T(\psi)\bar{b} + \bar{\tau} + \bar{\tau}_{wind} + \bar{\tau}_{wave} \quad (2)$$

where $\bar{\eta}$ of Equation (1) represents a vector containing the position and orientation of the origin of the body frame with respect to NED:

$$\bar{\eta} = \{x_n, y_n, \psi\}^T \quad (3)$$

in which x_n and y_n represent the position of the body frame, with respect to the NED frame and expressed in NED coordinates. ψ represents the heading angle of the vehicle with respect to North.

The vector, \bar{v} represents a vector containing the velocity of the origin of the body frame with respect to NED, expressed in body-fixed coordinates:

$$\bar{v} = \{u, v, r\}^T \quad (4)$$

in which u and v represent the linear velocities of the vehicle in the surge and sway directions, respectively. The last term, r , represents the vehicle's angular yaw velocity about its vertical body-fixed z-axis.

From Equation (1), $\bar{R}(\psi)$ represents a rotation matrix representing transformations between the NED coordinate frame and the body-fixed coordinate frame and is shown below:

$$\bar{R}(\psi) = \begin{bmatrix} \cos(\psi) & -\sin(\psi) & 0 \\ \sin(\psi) & \cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (5)$$

It is important to note that Equation (5) is used to convert coordinates represented in the body frame to coordinates in the NED frame. The transpose of this matrix can be used to convert coordinates from the NED frame to the body frame.

From Equation (2), \bar{M} and \bar{D} represent the mass and damping matrices, respectively. They will be discussed in further detail in the following sections. Slowly changing ocean current is represented by \bar{b} . Finally, τ , τ_{wind} , and τ_{wave} represent the control, environmental wind, and environmental wave forces and torques, respectively. For this application, the vehicle operates in very calm waters with minimal environmental disturbances, so \bar{b} , τ_{wind} , and τ_{wave} are reduced to zero. Additionally, for the low-cost system used in this work, there are no direct measurements available for these environmental disturbances. They are excluded from this EKF application. Equation (2) is simplified to:

$$\bar{M}\dot{\bar{v}} + \bar{D}\bar{v} = \bar{\tau} \quad (6)$$

Equation (6) is the dynamical model used in the state propagation model of EKF and Equation (1) is used to relate inertial position measurements to the vehicle's body-fixed velocity in the measurement update portion of the EKF.

2.2.3 Mass Matrix

The mass matrix of the 3DOF DP Model includes rigid body mass and rotational inertia terms, as well as added mass terms, which scale with the vehicle's acceleration. The added mass terms are used to model forces and torques that act on the vehicle as it accelerates and displaces the water around it. The mass matrix is written:

$$\bar{M} = \begin{bmatrix} m - X_{\dot{u}} & 0 & 0 \\ 0 & m - Y_{\dot{v}} & mx_g - Y_{\dot{r}} \\ 0 & mx_g - Y_{\dot{r}} & I_z - N_{\dot{r}} \end{bmatrix} \quad (7)$$

in which m represents the total mass of the vehicle, I_z represents the inertia of the vehicle about the vertical downwards axis located at its center of gravity, and x_g represents the offset of the origin of the body frame from the vessel's center of gravity and center of rotation in the body-fixed y-direction. The vessel's center of gravity and center of rotation are assumed to be coincident. $X_{\dot{u}}$ represents the added mass coefficient that scales forces acting on the vehicle in the body-fixed x-direction, caused by acceleration in the surge direction. Similarly, $Y_{\dot{v}}$ and $Y_{\dot{r}}$ are coefficients for forces acting on the vehicle in the body-fixed y-direction caused by acceleration in the sway direction and angular yaw acceleration, respectively. Finally, $N_{\dot{r}}$ is a coefficient that characterizes the torque about the body-fixed z-axis caused by angular acceleration in yaw.

Because the origin of the body-fixed coordinate frame is chosen along the length-wise midline of the vehicle, and symmetry in the port and starboard directions about the length-wise midline are assumed, the body frame lies along the x-axis of the origin located at the vessel's center of gravity. There is no offset in the body-fixed y-direction, but an offset in the body-fixed x-direction remains and x_g is non-zero. Equation (7) is the mass matrix used in the EKF models described in the following chapter.

2.2.4 Damping

The damping matrix of the 3DOF DP Model represents forces acting on the vehicle that scale with the vehicle's velocity. The damping matrix is shown below:

$$\bar{D} = - \begin{bmatrix} X_u & 0 & 0 \\ 0 & Y_v & Y_r \\ 0 & N_v & N_r \end{bmatrix} \quad (8)$$

in which X , Y , and N represent forces acting along the body-fixed x-axis and y-axis and torque acting about the body-fixed z-axis. The subscripts u , v , and r indicate the velocities in the body-fixed x-direction, y-direction, and angular z-direction that scale with the term.

2.2.5 Thruster Model

The control vector $\bar{\tau}$ of Equation (2) consists of two forces and one torque. The vector can be written:

$$\bar{\tau} = \{X, Y, N\}^T \quad (9)$$

in which X and Y represent forces in the body-fixed x-direction and y-direction, respectively and N represents a torque about the body-fixed z-axis.

$\bar{\tau}$ can be broken down further to map thruster commands to forces and torque in SI units. This is shown in the equation below:

$$\bar{\tau} = \bar{B}\bar{u} \quad (10)$$

where \bar{B} represents the thruster configuration matrix and \bar{u} represents a vector of thrust commands. The thruster matrix is shown below:

$$\bar{B} = a \begin{bmatrix} 1 & 1 \\ 0 & 0 \\ y_p & -y_s \end{bmatrix} \quad (11)$$

in which a is a constant mapping thrust commands to forces and torque. The variables y_p and y_s represent the port and starboard thruster offsets in the body-fixed y-direction from the vehicle's center of gravity. For actual implementation, the thrusters are capable of a higher forward thrust than reverse thrust. The magnitude of a depends on the sign of the control input, discussed in more detail in Section 4.2.3.

The control vector, \bar{u} , consists of commands sent to the port and starboard thrusters and is shown below:

$$\bar{u} = [T_p, T_s]^T \quad (12)$$

where T_p and T_s represent the port and starboard thrust commands, respectively. For the vehicle used in this work, thrust commands range between -1000 and 1000, with -1000 representing a full reverse thrust and 1000 representing a full forward thrust.

CHAPTER 3: EXTENDED KALMAN FILTER

3.1 Extended Kalman Filter Algorithm

The state estimation method used is a EKF algorithm. This method was chosen for its simplicity and proven success as a standard in state estimation. The EKF begins with an initial state and initial error covariance and loops through the steps shown in Equations (13-17) below:

State projection to the next time step:

$$\hat{\bar{x}}_k^- = \bar{A}\hat{\bar{x}}_{k-1} + \bar{B}\bar{u}_{k-1} \quad (13)$$

Covariance projection to the next time step:

$$\bar{P}_k^- = \bar{A}\bar{P}_{k-1}\bar{A}^T + \bar{Q} \quad (14)$$

Kalman gain calculation:

$$\bar{K}_k = \bar{P}_k^- \bar{H}^T (\bar{H}\bar{P}_k^- \bar{H}^T + \bar{R})^{-1} \quad (15)$$

State update incorporating new measurements:

$$\hat{\bar{x}}_k = \hat{\bar{x}}_k^- + \bar{K}_k(\bar{z}_k - \bar{H}\hat{\bar{x}}_k^-) \quad (16)$$

Covariance update incorporating new measurements:

$$\bar{P}_k = (\bar{I} - \bar{K}_k \bar{H})\bar{P}_k^- \quad (17)$$

in which \bar{x} represents the state vector, \bar{A} represents the state propagation model, and \bar{B} represents the relationship between the control input \bar{u} and the state. The symbol \bar{P} represents the error covariance in the estimated state. The symbols \bar{Q} and \bar{R} represent the covariance associated with the state propagation model and the sensor measurements, respectively. The Kalman gain is

represented by \bar{K} . Finally, \bar{z} represents the sensor measurements and \bar{H} represents the relationship between the sensor measurements and the state vector. The subscripts k and $k - 1$ indicate the current and previous loop iterations. The negative sign superscript indicates a state estimate or error covariance created before incorporation of new sensor measurements.

Estimates are denoted by a hat accent.

For this application, the state vector includes $\bar{\eta}$, \bar{v} , and the linear components of $\dot{\bar{v}}$. The state vector is shown below:

$$\bar{x} = \{x_n, y_n, \psi, u, v, r, \dot{u}, \dot{v}\}^T \quad (18)$$

It is important to note that u in the state vector is not the same as the control vector represented by \bar{u} .

The EKF can be split into two steps: the prediction step, consisting of Equations (13-14), and the measurement update step, consisting of Equations (15-17). These steps and their associated components are discussed in more detail in the following sections.

3.2 State Propagation Model

The state propagation model used in this EKF is derived from the vehicle dynamics in Equation (6). By isolating the $\dot{\bar{v}}$ term, Equation (6) can be rewritten:

$$\dot{\bar{v}} = -\bar{M}^{-1}\bar{D}\bar{v} + \bar{M}^{-1}\bar{\tau} \quad (19)$$

Equation (19) is used to propagate the accelerations forward to the next time step through their relationships to the velocities of the previous step. The first three terms in the state vector are estimated in the prediction step using a first order Taylor series approximation with Equation (1). The linear velocity terms are estimated in the prediction step using a first order Taylor series approximation with the previous state's acceleration. The vehicle dynamics are included in the position and linear velocity estimates through the incorporation of acceleration estimates. Similarly, the angular vehicle dynamics are propagated into the orientation estimate through its relationship with angular velocity and time.

From Equation (1) and Equation (19), the $-\bar{M}^{-1}\bar{D}$ matrix can be expanded to form the last three rows of the state propagation matrix, \bar{A} :

$$\bar{A} = \begin{bmatrix} 1 & 0 & 0 & \cos(\psi_{k-1}) dt & -\sin(\psi_{k-1}) dt & 0 & 0 & 0 \\ 0 & 1 & 0 & \sin(\psi_{k-1}) dt & \cos(\psi_{k-1}) dt & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & dt & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & dt & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & dt \\ 0 & 0 & 0 & -dt[M^{-1}D](3,1) & -dt[M^{-1}D](3,2) & 1 - dt[M^{-1}D](3,3) & 0 & 0 \\ 0 & 0 & 0 & -[M^{-1}D](1,1) & -[M^{-1}D](1,2) & -[M^{-1}D](1,3) & 0 & 0 \\ 0 & 0 & 0 & -[M^{-1}D](2,1) & -[M^{-1}D](2,2) & -[M^{-1}D](2,3) & 0 & 0 \end{bmatrix} \quad (20)$$

in which dt represents the time elapsed between the current and previous update. The first two rows involve the transformation matrix from Equation (5) relating the position estimates in NED to the velocity estimates in body-fixed coordinates. The last three rows include the mass matrix and the damping matrix of the vehicle from Equation (7). Because the relationship between the position estimates in NED and the velocity estimates in body-fixed coordinates are nonlinear, the Jacobian matrix of Equation (19) must be determined and used as the state propagation model of the EKF. The Jacobian matrix is derived by calculating the partial derivatives of each function, with respect to each state vector component. This is represented in symbolic form below:

$$\bar{A} = \begin{bmatrix} \frac{\partial x_n}{\partial x_n} & \frac{\partial x_n}{\partial y_n} & \frac{\partial x_n}{\partial \psi} & \frac{\partial x_n}{\partial u} & \frac{\partial x_n}{\partial v} & \frac{\partial x_n}{\partial r} & \frac{\partial x_n}{\partial \dot{u}} & \frac{\partial x_n}{\partial \dot{v}} \\ \frac{\partial y_n}{\partial x_n} & \frac{\partial y_n}{\partial y_n} & \frac{\partial y_n}{\partial \psi} & \frac{\partial y_n}{\partial u} & \frac{\partial y_n}{\partial v} & \frac{\partial y_n}{\partial r} & \frac{\partial y_n}{\partial \dot{u}} & \frac{\partial y_n}{\partial \dot{v}} \\ \frac{\partial \psi}{\partial x_n} & \frac{\partial \psi}{\partial y_n} & \frac{\partial \psi}{\partial \psi} & \frac{\partial \psi}{\partial u} & \frac{\partial \psi}{\partial v} & \frac{\partial \psi}{\partial r} & \frac{\partial \psi}{\partial \dot{u}} & \frac{\partial \psi}{\partial \dot{v}} \\ \frac{\partial u}{\partial x_n} & \frac{\partial u}{\partial y_n} & \frac{\partial u}{\partial \psi} & \frac{\partial u}{\partial u} & \frac{\partial u}{\partial v} & \frac{\partial u}{\partial r} & \frac{\partial u}{\partial \dot{u}} & \frac{\partial u}{\partial \dot{v}} \\ \frac{\partial v}{\partial x_n} & \frac{\partial v}{\partial y_n} & \frac{\partial v}{\partial \psi} & \frac{\partial v}{\partial u} & \frac{\partial v}{\partial v} & \frac{\partial v}{\partial r} & \frac{\partial v}{\partial \dot{u}} & \frac{\partial v}{\partial \dot{v}} \\ \frac{\partial r}{\partial x_n} & \frac{\partial r}{\partial y_n} & \frac{\partial r}{\partial \psi} & \frac{\partial r}{\partial u} & \frac{\partial r}{\partial v} & \frac{\partial r}{\partial r} & \frac{\partial r}{\partial \dot{u}} & \frac{\partial r}{\partial \dot{v}} \\ \frac{\partial \dot{u}}{\partial x_n} & \frac{\partial \dot{u}}{\partial y_n} & \frac{\partial \dot{u}}{\partial \psi} & \frac{\partial \dot{u}}{\partial u} & \frac{\partial \dot{u}}{\partial v} & \frac{\partial \dot{u}}{\partial r} & \frac{\partial \dot{u}}{\partial \dot{u}} & \frac{\partial \dot{u}}{\partial \dot{v}} \\ \frac{\partial \dot{v}}{\partial x_n} & \frac{\partial \dot{v}}{\partial y_n} & \frac{\partial \dot{v}}{\partial \psi} & \frac{\partial \dot{v}}{\partial u} & \frac{\partial \dot{v}}{\partial v} & \frac{\partial \dot{v}}{\partial r} & \frac{\partial \dot{v}}{\partial \dot{u}} & \frac{\partial \dot{v}}{\partial \dot{v}} \end{bmatrix} \quad (21)$$

As an example, if Equation (13), the first equation of the EKF prediction step, is multiplied out using Equation (20), the first row of the resulting equation is found:

$$x_k = x_{k-1} + \cos(\psi_{k-1}) u_{k-1} dt - \sin(\psi_{k-1}) v_{k-1} dt + X \quad (22)$$

In which X represents the control force in the x-direction. If the first-order partial derivative of Equation (22) is taken with respect to each state member according to the first row of Equation (21), the resulting equation is found:

$$\bar{A} = [1 \quad 0 \quad -(u_{k-1} \sin(\psi_{k-1}) + v_{k-1} \cos(\psi_{k-1})) dt \quad \cos(\psi_{k-1}) dt \quad -\sin(\psi_{k-1}) dt \quad 0 \quad 0 \quad 0] \quad (23)$$

When Equation (23) is multiplied by the state vector, it can be used to find the predicted estimate of the x-position in the form of Equation (13):

$$x_k = [1 \quad 0 \quad -(u_{k-1} \sin(\psi_{k-1}) + v_{k-1} \cos(\psi_{k-1})) dt \quad \cos(\psi_{k-1}) dt \quad -\sin(\psi_{k-1}) dt \quad 0 \quad 0 \quad 0] \\ \times [x_{k-1} \quad y_{k-1} \quad \psi_{k-1} \quad u_{k-1} \quad v_{k-1} \quad r_{k-1} \quad \dot{u}_{k-1} \quad \dot{v}_{k-1}]^T \quad (24)$$

Equation (23) corresponds to the first row of the state propagation matrix, \bar{A} . Following the same procedure for the remaining seven states, the resulting Jacobian matrix is found:

$$\bar{A} = \begin{bmatrix} 1 & 0 & -dt(u_{k-1} \sin(\psi_{k-1}) + v_{k-1} \cos(\psi_{k-1})) & \cos(\psi_{k-1})dt & -\sin(\psi_{k-1})dt & 0 & 0 & 0 \\ 0 & 1 & dt(u_{k-1} \cos(\psi_{k-1}) - v_{k-1} \sin(\psi_{k-1})) & \sin(\psi_{k-1})dt & \cos(\psi_{k-1})dt & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & dt & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & dt & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & dt \\ 0 & 0 & 0 & -dt[M^{-1}D](3,1) & -dt[M^{-1}D](3,2) & 1 - dt[M^{-1}D](3,3) & 0 & 0 \\ 0 & 0 & 0 & -[M^{-1}D](1,1) & -[M^{-1}D](1,2) & -[M^{-1}D](1,3) & 0 & 0 \\ 0 & 0 & 0 & -[M^{-1}D](2,1) & -[M^{-1}D](2,2) & -[M^{-1}D](2,3) & 0 & 0 \end{bmatrix} \quad (25)$$

The \bar{A} matrix of Equation (25) is used as the state propagation model of the EKF. As seen above, this linearization is done about the previous state estimate¹.

The control vector \bar{u} is identical to $\bar{\tau}$ of Equation (19) and consists of a force along the body-fixed x-axis, a force along the body-fixed y-axis, and a torque about the body-fixed z-axis:

$$\bar{u} = \{X, Y, N\}^T \quad (26)$$

Because the vessel used in this thesis utilizes a differential thruster layout, it is underactuated and there is currently no control in the body-fixed y-direction. The Y term is included to maintain matrix dimension compatibility with \bar{M}^{-1} and to allow the option of using control in the body-fixed y-direction, should the vehicle change to a different thruster layout that allows full actuation.

Utilizing Equation(26), \bar{B} is derived:

¹ This linearization step may not be necessary. It is possible to use Equation (20) as the state propagation matrix instead of Equation (25), however this has not been implemented and tested in this work.

$$\bar{B} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ dt * M^{-1}(3,1) & dt * M^{-1}(3,2) & dt * M^{-1}(3,3) \\ M^{-1}(1,1) & M^{-1}(1,2) & M^{-1}(1,3) \\ M^{-1}(2,1) & M^{-1}(2,2) & M^{-1}(2,3) \end{bmatrix} \quad (27)$$

The control inputs are again related to the linear velocity, position, and orientation components through their relationships with the linear acceleration and angular velocity.

3.3 Measurement Model

The measurement update steps of the EKF involve Equations (15-17). For this work, the vehicle's position, yaw orientation, angular yaw rate, and linear accelerations are measured using an inertial measurement unit (IMU) and two RTK GNSS sensors. The two RTK GNSS sensors give redundant position information. The measurements are related to the state vector through the equation:

$$\bar{z}_k = \bar{H}\bar{x}_k \quad (28)$$

in which \bar{z}_k represents a vector of the values being measured and \bar{H} represents a measurement model that relates the values being measured to the state vector. For this application, the measurement vector is:

$$\bar{z}_k = \{x_n, y_n, x_n, y_n, \psi, r, \dot{u}, \dot{v}\}^T \quad (29)$$

For each position measurement, the offset of the antenna from the origin of the body frame needs to be taken into account. The relationship between the position of the origin of the body frame and the position of the GNSS antenna is written:

$$x_{gps/n}^n = x_{b/n}^n + x_{gps/b}^n \quad (30)$$

$$y_{gps/n}^n = y_{b/n}^n + y_{gps/b}^n \quad (31)$$

in which x and y represent the x-position and y-position. The subscripts indicate the frame whose position is being measured and the reference frame to which it is measured from. The superscript represents the coordinate frame that the measurements are expressed in. As an example, the first term of Equation (30) is $x_{gps/n}^n$ and can be read as “the x-position of the GPS with respect to the origin of the NED frame, expressed in NED coordinates”.

Equation (30) is a function of the vessel’s yaw orientation, ψ , as the GPS offsets $x_{gps/b}^n$ and $y_{gps/b}^n$ are measured in body coordinates: $(x_{gps/b}^b, y_{gps/b}^b)$ and must be transformed using the rotation matrix of Equation (5) into NED coordinates:

$$\bar{x}_{gps/b}^n = \bar{R}(\psi)\bar{x}_{gps/b}^b \quad (32)$$

in which \bar{x} represents a reduced state vector consisting of position coordinates. Expanded, Equation (29) can be written:

$$\begin{bmatrix} x_{gps/b}^n \\ y_{gps/b}^n \end{bmatrix} = \begin{bmatrix} \cos(\psi) & -\sin(\psi) \\ \sin(\psi) & \cos(\psi) \end{bmatrix} \begin{bmatrix} x_{gps/b}^b \\ y_{gps/b}^b \end{bmatrix} \quad (33)$$

Multiplying out Equation (30), the coordinates of the GPS antenna with respect to the origin of the body-fixed coordinate frame and expressed in NED coordinates is found:

$$x_{gps/b}^n = \cos(\psi) x_{gps/b}^b - \sin(\psi) y_{gps/b}^b \quad (34)$$

$$y_{gps/b}^n = \sin(\psi) x_{gps/b}^b + \cos(\psi) y_{gps/b}^b \quad (35)$$

Equation (30) can be rewritten using Equations (34-35):

$$\begin{bmatrix} x_{gps/n}^n \\ y_{gps/n}^n \end{bmatrix} = \begin{bmatrix} x_{b/n}^n + \cos(\psi) x_{gps/b}^b - \sin(\psi) y_{gps/b}^b \\ y_{b/n}^n + \sin(\psi) x_{gps/b}^b + \cos(\psi) y_{gps/b}^b \end{bmatrix} \quad (36)$$

The relationship between the measurements and the state vector are nonlinear, so the Jacobian matrix relating the measurement vector to the state vector is derived from Equation (36) and is used to write the measurement model matrix of Equation (28). Expanded, Equation (28) can be written:

$$\begin{bmatrix} x_{gps,p} \\ y_{gps,p} \\ x_{gps,s} \\ y_{gps,s} \\ \psi \\ r \\ \dot{u} \\ \dot{v} \end{bmatrix} = \begin{bmatrix} 1 & 0 & -x_p \sin(\psi) - y_p \cos(\psi) & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & x_p \cos(\psi) - y_p \sin(\psi) & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & -x_s \sin(\psi) - y_s \cos(\psi) & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & x_s \cos(\psi) - y_s \sin(\psi) & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_n \\ y_n \\ \psi \\ u \\ v \\ r \\ \dot{u} \\ \dot{v} \end{bmatrix} \quad (37)$$

in which $x_{gps,p}$, $y_{gps,p}$, $x_{gps,s}$, and $y_{gps,s}$ represent the x-position and y-positions of the port and starboard GNSS antennae respectively, with reference to the NED frame and expressed in NED coordinates. The position offsets of the port and starboard antennae from the origin of the body-fixed coordinate frame, $x_{gps/b}^b$ and $y_{gps/b}^b$ of Equation (33) are represented by x_p , y_p , x_s , and y_s . The final measurement matrix used in the EKF is written:

$$\bar{H} = \begin{bmatrix} 1 & 0 & -x_p \sin(\psi) - y_p \cos(\psi) & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & x_p \cos(\psi) - y_p \sin(\psi) & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & -x_s \sin(\psi) - y_s \cos(\psi) & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & x_s \cos(\psi) - y_s \sin(\psi) & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \quad (38)$$

It is evident that the only states not being measured are the linear velocities. The measurement matrix is used to calculate the Kalman gain, as seen in Equation (15). The Kalman gain, measurement matrix, and measurement vector are used to determine the state estimate after the

measurement update, as seen in Equation (16). Finally, the Kalman gain and the measurement matrix are used to update the covariance associated with the estimate, as seen in Equation (17). While the matrices used to calculate the Kalman gain does not change in simulation due to the unchanging state propagation model, state propagation model covariance, measurement model, and measurement model covariance, the Kalman gain calculated at each time step does change in the real-world application. The covariances associated with the RTK GNSS sensors are updated at each time step in real-time by the sensors.

3.4 Covariance Tuning

The two covariance matrices, Q and R , are parameters have a great impact on how the EKF performs. These parameters determine how much noise, error, and latency result in the state estimates. Tuning the measurement covariance matrix, R , is straightforward. Covariances associated with each measurement can be calculated using data sheets or by finding the standard deviation of a set of data. Tuning the model covariance matrix, Q , is much more abstract and is often done by “feel”. The method used for this application involves an optimizer to minimize the error between the state estimates and the ground truth data. This is done to create a fair comparison of the performance of the two navigation solutions. Because it requires knowledge of the ground truth state, this method can only be utilized in simulation.

The equation used to tune the Q matrix is written:

$$F(\vec{x}) = \sqrt{\sum_{n=1}^8 \left(\sqrt{\frac{\sum_{m=1}^k (\hat{\vec{x}}_t - \vec{x}_t)^2}{k}} * \vec{w} \right)^2} \quad (39)$$

In which $\hat{\vec{x}}_t$ represents the vector of state estimates at each time step, \vec{x}_t represents the ground truth state vector at each time step, k represents the number of time steps, and \vec{w} represents a vector of weights to emphasize various state members for various maneuvers. The cost function utilizes the Euclidean norm of the root-mean-squared error between the state estimate and the ground truth, multiplied by the weight vector. The diagonal members of the Q matrix are utilized by the optimizer as parameters to change.

For this work, the EKF was tuned using simulation data from the Virtual RobotX (VRX) simulation in the Gazebo software. This simulation software is discussed further in Chapter 6. The “measured” sensor data and ground truth data were recorded and used to tune the EKF in Matlab. The obtained values were then used directly on the real-world EKF implementation.

3.5 Observability

The covariance associated with the estimates produced by the EKF is used to determine a ranking of observability for each state. Because the system is nonlinear, standard calculations for the observability Gramian cannot be done with this application. While there are methods for determining the observability Gramian of nonlinear systems involving Lie brackets, the sinusoidal terms in the state propagation matrix makes these methods very complicated.

The method used to determine a ranking of observability involves normalizing the covariances associated with the state estimates with respect to each other and determining the eigenvalues and eigenvectors of the normalized covariance matrix[29], [30]. Each eigenvector indicates which state member is associated with the corresponding eigenvalue. Eigenvalues with lower values indicate a high observability. The ranking produced is in relation to the other state members. The degree of observability between two separate filters cannot be compared using this method. This method is used in Chapter 6 to verify that without a direct velocity measurement, the surge and sway velocities are the least observable state members in the state vector. These state members, with heading, are the most important state members for the guidance and control algorithms to be used on the vehicle.

CHAPTER 4: IMPLEMENTATION

4.1 Physical Components

4.1.1 Wave Adaptive Modular Vessel

The USV used to test the navigation solutions is the 16-foot Wave Adaptive Modular Vessel (WAMV)[31]. It is a catamaran comprised of two pontoon hulls and a platform for loading sensors, computers, batteries, and other components. The WAMV is ideal for marine robotics because it features a suspension system with articulating hulls, giving the platform extra stability. The inflatable hulls also absorb vibrations caused by waves.



Figure 2: WAMV collecting data at Sand Island, Hawaii

4.1.2 Thrusters

The thrusters used on the WAMV are the 80lb Riptide Terrova Bow-Mount Trolling Motors by Minn Kota. While documentation indicates that they are capable of a maximum forward thrust of 80lb[32], the actual forward thrust achieved by the motors is likely lower. For this work, the documented 80lb capability, while inaccurate, will be used. Four motors are used in total on the

WAMV: two attached to the right pontoon and two attached to the left pontoon, as seen in Figure 3.



Figure 3: Rear view of WAMV with Minn Kota motors

The motors are laid out in a differential thrust configuration. This is important to keep in mind for GNC development because this prohibits control forces in the vehicle's sway direction. This also affects the control matrix for mapping thruster commands to the control vector, discussed in more detail in Section 4.2.3.

4.1.3 Sensors

4.1.3.1 RTK GNSS

To obtain position and heading measurements, two RTK GNSS modules and antennae are used onboard the WAMV. The GNSS board used is the SparkFun GPS-RTK2 Board-ZED-F9P, seen in Figure 4. This board is capable of 1cm of accuracy when in RTK-fixed mode, but typically reports covariances of 3-5 cm when out on the water[33].



Figure 4: SparkFun ZED-F9P RTK GPS board (left) and Taoglas antenna (right)

The antenna used to collect satellite data is the Taoglas AA.175.301111, depicted in Figure 4[34]. The two rover antennae are placed parallel to each other off the port and starboard edges of the WAMV platform in an effort to minimize noise coming from the WAMV's electrical system. The rover antennae and the IMU onboard the WAMV are depicted in Figure 5.

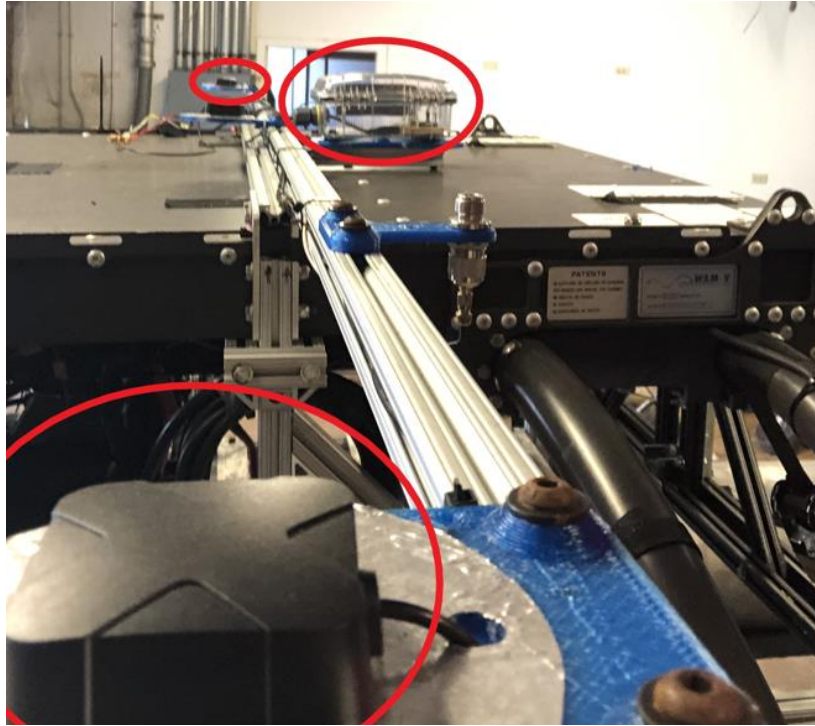


Figure 5: GNSS rover antennae and IMU onboard WAMV

A third SparkFun and antenna set-up is used at the base station to pass RTK corrections to the two rovers onboard the WAMV and to act as a reference point for the origin of the NED coordinate frame.

An important assumption made regarding the Kalman Filter algorithm is that errors have a Gaussian distribution. To prove this true for the GNSS sensors, stationary GNSS data was recorded and plotted with real-time kinematic (RTK) corrections and without RTK corrections[35].

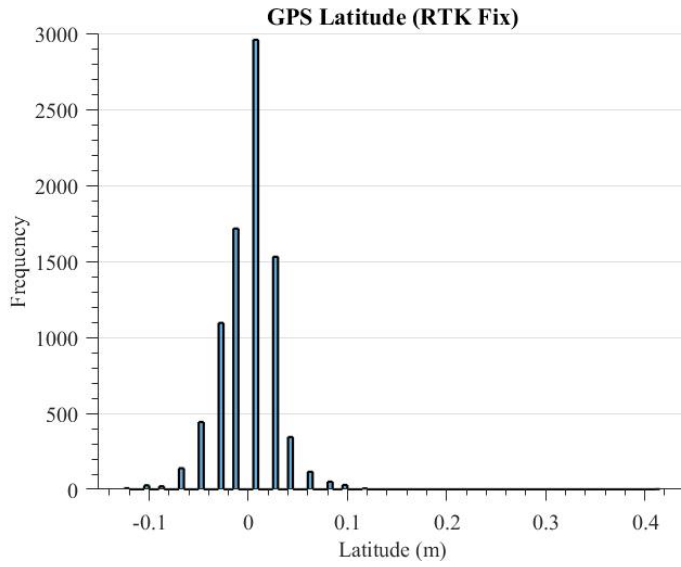


Figure 6: Histogram of stationary latitude measurements with RTK corrections

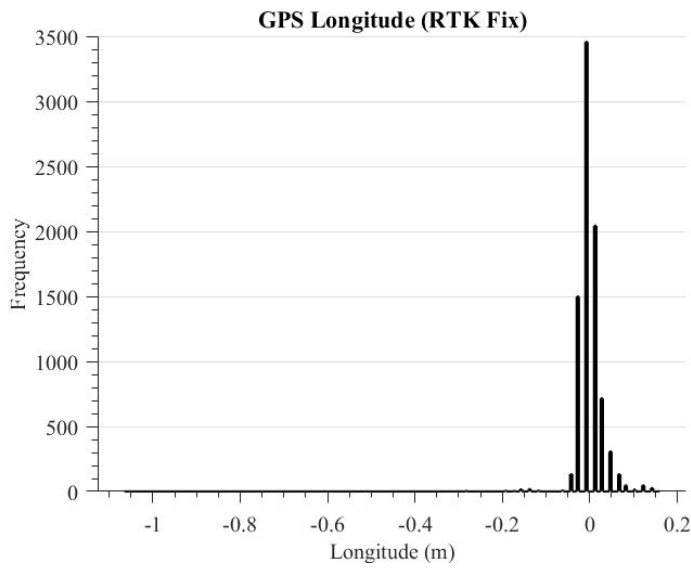


Figure 7: Histogram of stationary longitude measurements with RTK corrections

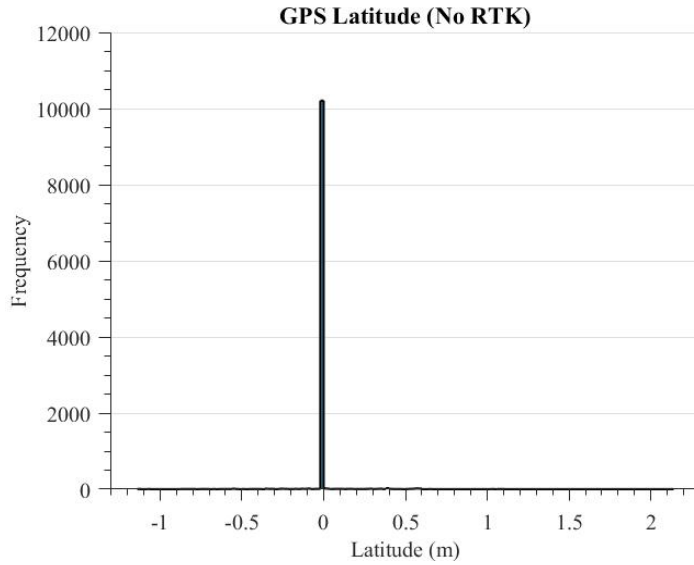


Figure 8: Histogram of stationary latitude measurements without RTK corrections. It is difficult to detect smaller bins located near the largest bin, but the distribution is Gaussian when zoomed in.

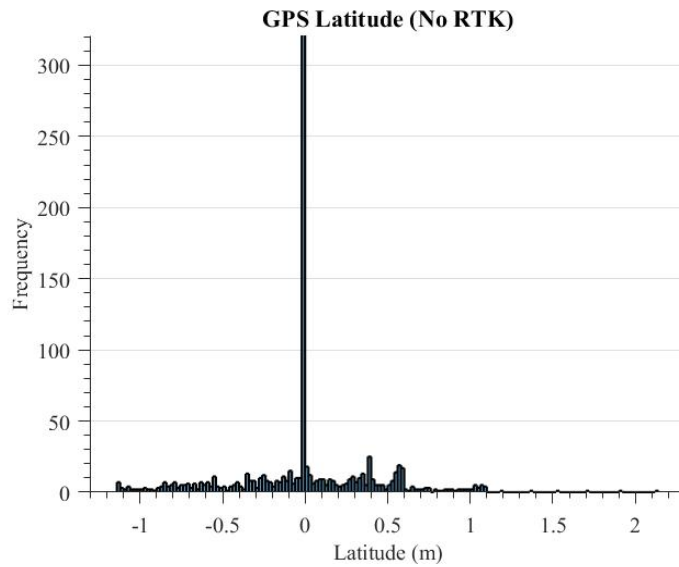


Figure 9: Zoomed in histogram of stationary latitude measurements without RTK corrections

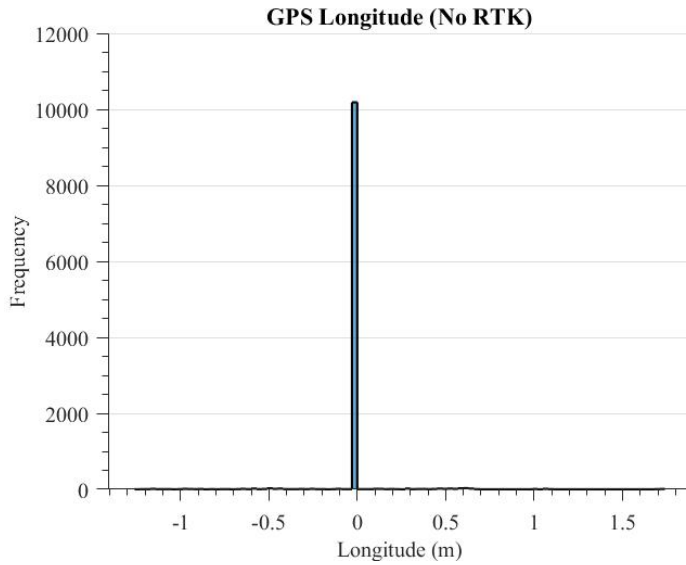


Figure 10: Histogram of stationary longitude measurements without RTK corrections. It is difficult to detect smaller bins located near the largest bin, but the distribution is Gaussian when zoomed in

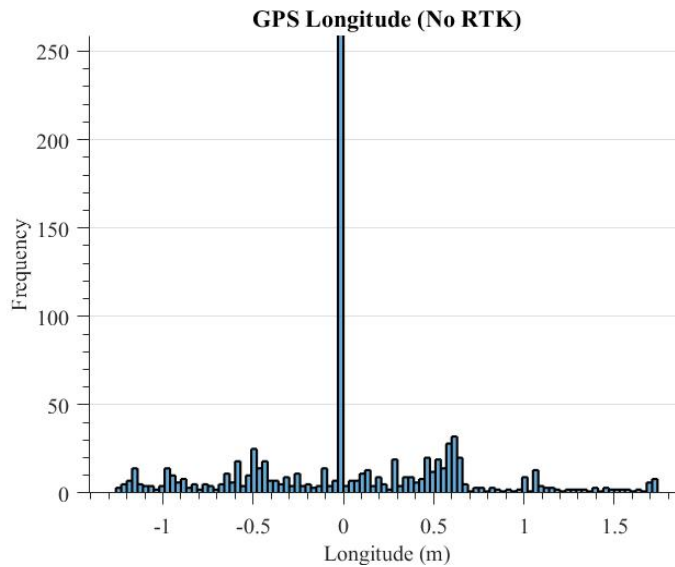


Figure 11: Zoomed in histogram of stationary longitude measurements without RTK corrections

As seen in Figures (6-11), the assumption of Gaussian error holds true for the GNSS sensors both with and without RTK corrections.

4.1.3.2 Inertial Measurement Unit

The IMU used for this experiment is the Analog Devices ADIS16470 IMU[36]. This IMU consists of a gyroscope and an accelerometer, but does not have a magnetometer. The IMU's orientation estimates derived from the gyroscope are not used. Only the angular velocity and linear acceleration measurements are included in the EKF.

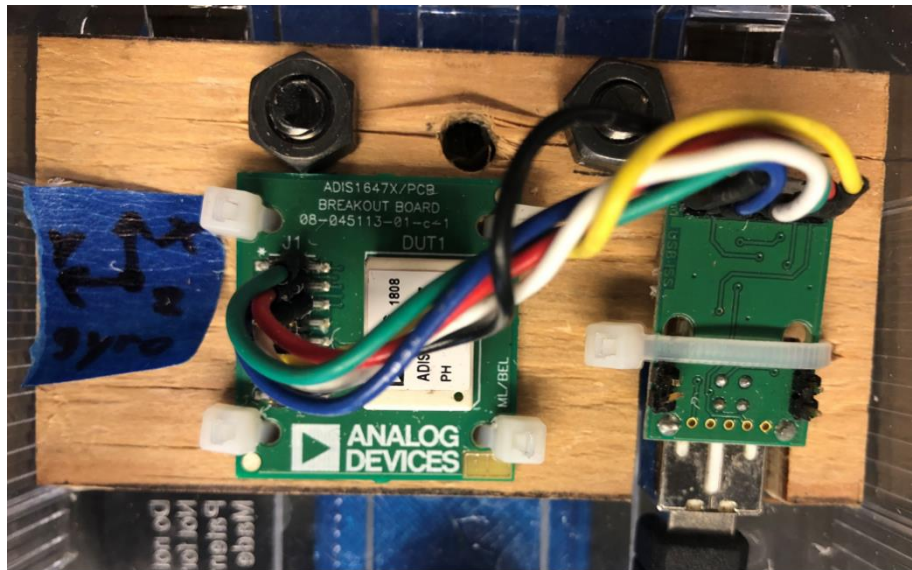


Figure 12: ADIS16470 IMU with breakout board

The driver used with the ADIS16470 is the *adi_driver* ROS package maintained by Tokyo Open-source Robotics Kyokai (TORK)[37]. The package allows users to configure IMU settings like the update rate and the ROS frame ID. It also allows live data visualization using RVIZ and graphical plots. Upon starting the ROS launch file from TORK, the IMU begins collecting and publishing orientation with respect to its starting measurements, angular velocity, and linear acceleration data to ROS. It also begins an algorithm to determine the bias of the gyroscope sensor. After 40s at rest, the averages of the gyroscope sensors are stored as the bias value on the IMU chip and are subtracted from the angular velocity measurements.

Similar to the GNSS sensors, it is important that the measurements used by the IMU have a Gaussian error distribution. This is demonstrated with stationary IMU data and plotted below.

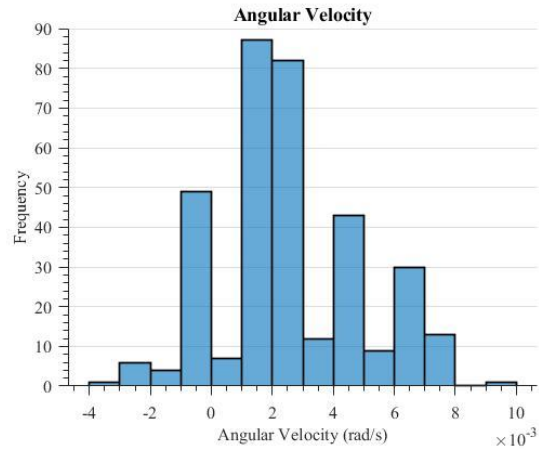


Figure 13: Histogram of stationary angular velocity measurements

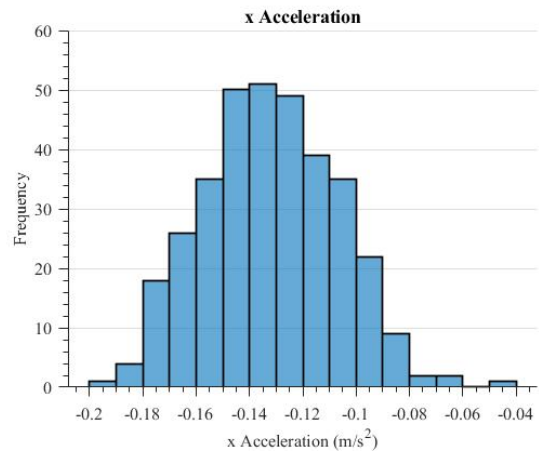


Figure 14: Histogram of stationary x-acceleration measurements

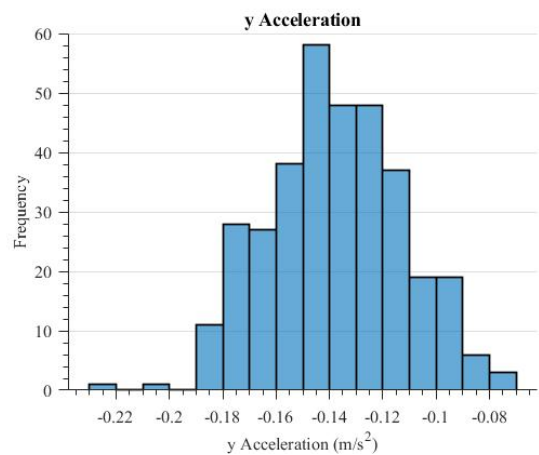


Figure 15: Histogram of stationary y-acceleration measurements

As seen in Figures (13-15), the angular velocity, x-acceleration, and y-accelerations have Gaussian error distributions.

4.1.3.3 Sensor Specifications

For this work, all sensors were updated at a rate of 10Hz. A table of sensor specifications is shown below. The covariance values listed are the same values used in the EKF with vehicle dynamics.

Table 1: Sensor Specifications

	Update Rate (Hz)	Covariance	Resolution
Latitude (RTK)	10	0.0025 m ²	1.700*10 ⁻⁷ decimal degrees
Longitude (RTK)	10	0.002 5m ²	1.700*10 ⁻⁷ decimal degrees
Latitude (No RTK)	10	1 m ²	1.700*10 ⁻⁷ decimal degrees
Longitude (No RTK)	10	1 m ²	1.700*10 ⁻⁷ decimal degrees
Angular Velocity	10	0.2 (rad/s) ²	1.297*10 ⁻⁵ rad/s
x-Acceleration	10	0.1 (rad/s) ²	2.673*10 ⁻⁴ m/s ²
y-Acceleration	10	0.1 (rad/s) ²	2.673*10 ⁻⁴ m/s ²

4.2 Software

In order to implement the state estimation technique described in the previous chapter, programming was done using Python 2. Robot Operating System (ROS) was used as the framework to pass sensor data, updated states, control inputs, and other calculations between various pieces of code, called nodes. Communication between nodes is achieved when a node publishes information to a topic, which is then received by another node through subscription to that topic. This allows real-time data processing needed for live sensor fusion. A flowchart of the various nodes and topics communicated between them for this implementation is shown in Figure 16.

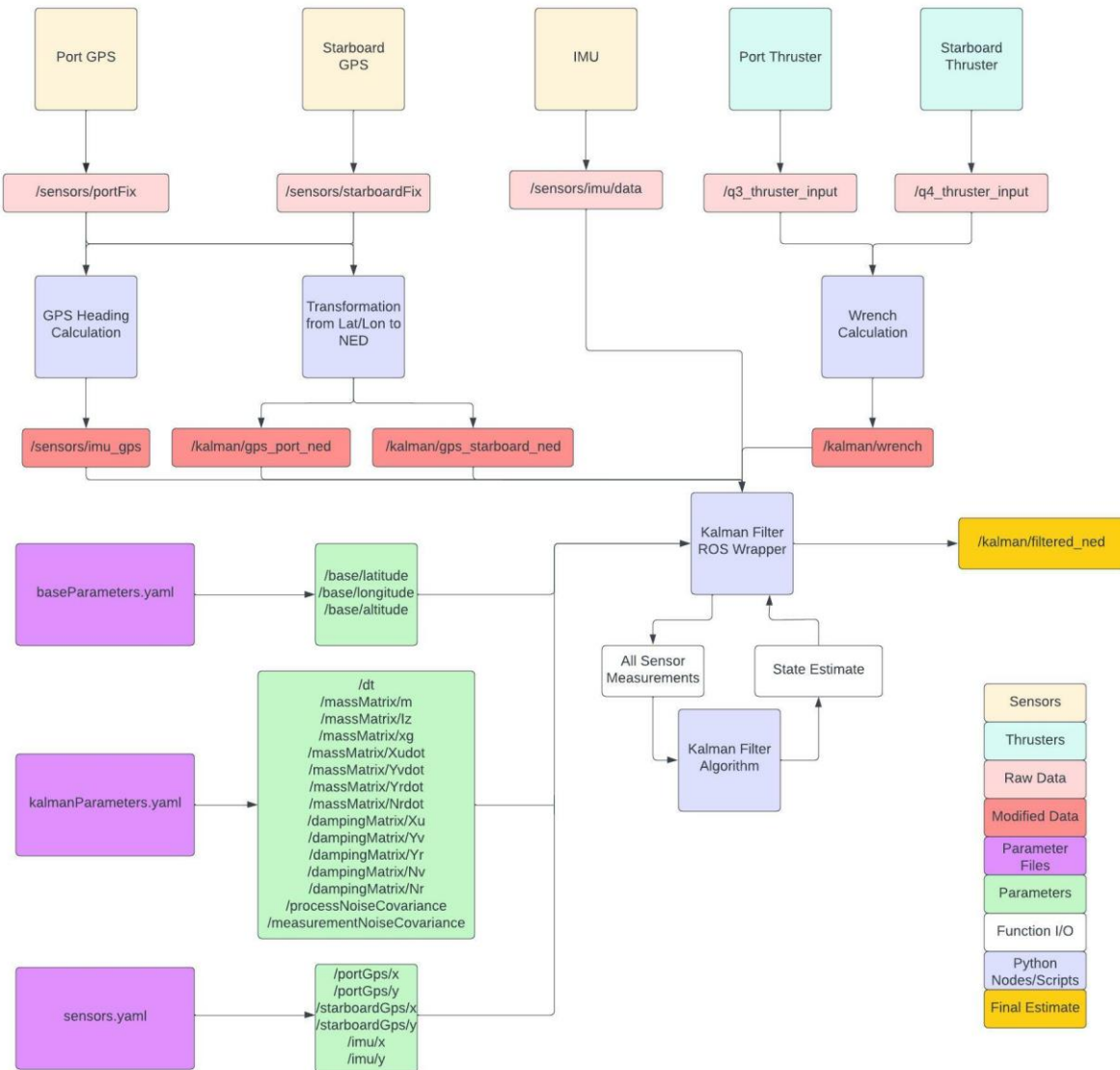


Figure 16: Software flow chart for WAMV Extended Kalman Filter implementation

4.2.1 Differential GPS Bearing Calculation

The IMU used in this application has no magnetometer and is unable to produce a heading measurement. Because two RTK GNSS are available, the bearing of the vehicle can be calculated from the two sets of latitude and longitude measurements using:

$$X = \cos(\mu_b) \sin(l_b - l_a) \quad (40)$$

$$Y = \cos(\mu_a) \sin(\mu_b) - \sin(\mu_a) \cos(\mu_b) \cos(l_b - l_a) \quad (41)$$

$$\beta = \text{atan2}(X, Y) \quad (42)$$

which μ_a and μ_b represent the latitude measurements of each GNSS and l_a and l_b represent their corresponding longitude measurements. The variable β represents the bearing between the two points, with respect to NED, in which North is equivalent to true North[38][39]. The heading of the vehicle is found by adding 90 degrees to β .

4.2.2 Transformation from Latitude/Longitude to NED

Before incorporating position measurements from each RTK GNSS directly into the EKF, the data must be converted into x_n and y_n coordinates. This is done using AlvinXY coordinate transformations:

$$x_n = (\mu - \mu_0)(111132.09 - 566.05 \cos(2\mu_{0,rad}) + 1.20 \cos(4\mu_{0,rad}) - 0.002 \cos(6\mu_{0,rad})) \quad (43)$$

$$y_n = (l - l_0)(111415.13 \cos(\mu_{0,rad}) - 94.55 \cos(3\mu_{0,rad}) - 0.12 \cos(5\mu_{0,rad})) \quad (44)$$

in which μ and l represent the GNSS latitude and longitude measurements, respectively. The subscripted μ_0 and l_0 represent an arbitrary reference point. In this case, it is chosen to be coincident with the local map frame at a point near the base station[21][22].

4.2.3 Wrench Calculation

The wrench vector consists of linear forces in the body-fixed x-direction and y-direction in units of N and a torque about the body-fixed z-axis in units of N*m. It is calculated from input thruster commands as described in Section 2.2.5. For this work, thruster commands range from a value of -1000, corresponding to a maximum reverse thrust, to 1000, corresponding to a maximum forward thrust. From Section 4.1.2, each Minn Kota motor is capable of a maximum

forward thrust of 80lb. The maximum reverse thrust is not listed. From previous testing with the motors, it was found that the amount of thrust force provided by the thrusters when given a command of maximum reverse is roughly half of that at maximum forward thrust. Based on this information, a maximum reverse thrust of 40lb for each motor is used.

As seen in section 4.1.2, two motors are attached to each pontoon for a total of four motors. Motors attached to the same pontoon are treated as one large thruster with a maximum forward thrust of 80lb, or 355.86N and maximum reverse thrust of 40lb, or 177.93N. To map thruster commands from [-1000, 1000] to [-177.93, 355.86], the following piecewise function is used:

$$X = \begin{cases} \frac{177.93}{1000} \tau, & x < 0 \\ \frac{355.86}{1000} \tau, & x \geq 0 \end{cases} \quad (45)$$

where X represents the control force in the linear x-direction and τ represents the motor commands in the range of [-1000, 1000]. This conversion is done for each thruster pairing whenever a new command is published and the results are summed to find the total force acting on the WAMV by all four thrusters. Because the motors are attached to the WAMV in a differential thruster layout, there is no control force in the body-fixed y-direction and a constant value of 0N is published to the wrench topic.

The torque about the body-fixed z-axis is found by multiplying each thruster's offset by the control force of Equation (44):

$$N = y_{thruster} * X \quad (46)$$

in which N represents the torque caused by the specific thruster pair and $y_{thruster}$ represents the offset in the body-fixed y-direction from the WAMV's center of gravity. This results in a positive torque corresponding to clockwise direction, consistent with the body-fixed axes and right hand rule. It is assumed that the center of gravity lies along the WAMV's fore to aft centerline due to symmetry. Similar to the surge forces, the resulting

torques from the left and right thruster pairings are summed to find the total control torque acting on the WAMV.

4.2.4 Parameter Files

One advantage of using ROS is its parameter server feature. The parameter server allows constants called parameters to be defined in a parameter file and uploaded where nodes can access their values. This is convenient for defining and adjusting characteristics specific to each vehicle, as variables are all located in one place. The same code can be used for different vehicles, as long as their characteristics are changed in the parameter files.

For this work, three parameter files are used: *baseParameters.yaml*, *kalmanParameters.yaml*, and *sensors.yaml*. The first file, *baseParameters.yaml*, is used to hold the latitude and longitude coordinates of the base station. Its contents are shown in Table 2.

Table 2: Parameters defined in *baseParameters.yaml*

Parameter Name	Value (degrees)	Description
/base/latitude	21.311773N	Base station latitude
/base/longitude	-157.889303E	Base station longitude

The second file, *kalmanParameters.yaml*, is used to define any variables needed to create the EKF. This includes the WAMV's physical characteristics, and the added mass and hydrodynamic damping coefficients of the dynamic model.

Table 3: Parameters defined in *kalmanParameters.yaml*

Parameter Name	Value	Units	Description
dt	0.1	s	EKF update rate
/massMatrix/m	368	kg	Total mass of WAMV with all components
/massMatrix/xg	0	m	Offset of body frame from COG in y-direction
/massMatrix/Iz	512	kg*m ²	WAMV's inertia about z-axis
/massMatrix/Xudot	0	kg	$X_{\dot{u}}$
/massMatrix/Yvdot	0	kg	$Y_{\dot{v}}$
/massMatrix/Yrdot	0	kg	$Y_{\dot{r}}$
/massMatrix/Nrdot	0	kg*m ²	$N_{\dot{r}}$
/dampingMatrix/Xu	51.3	kg/s	X_u
/dampingMatrix/Yv	40.0	kg/s	Y_v
/dampingMatrix/Yr	0	kg/s	Y_r
/dampingMatrix/Nv	0	kg/s	N_v
/dampingMatrix/Nr	400.0	kg*m ² /s	N_r

Because the actual added mass and hydrodynamic damping coefficients are unknown for the WAMV, these terms were taken from the Virtual RobotX (VRX) simulation and then tuned to base off of certain maneuvers that would give information about these parameters[42]. The maneuvers used include a full forward thrust acceleration from rest to find the $X_{\dot{u}}$ term and constant forward velocity to find the X_u term. To find the Y_v term, a lateral thruster was placed in the center of the WAMV and a strafing maneuver was used. Lastly, to find the N_r term, the WAMV was driven in a circular motion with constant half thrust sent to one thruster and full thrust sent to the other. All maneuvers and model tuning were done in simulation. The same values obtained were used in the real-world application. The process noise covariance matrix associated with the EKF's state propagation model and measurement noise covariance matrix associated with each sensor are also included in *kalmanParameters.yaml*. They are shown below:

Table 4: EKF covariances defined in *kalmanParameters.yaml*

Parameter Name	Value
/processNoiseCovariance	[2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.5236, 0, 0, 0, 0, 0, 0, 0, 0, 0, 4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.5236, 0, 0, 0, 0, 0, 0, 0, 0, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 3]
/measurementNoiseCovariance	[0.0025, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.0025, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.0025, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.0025, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.1745, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.1]

It is important to note that the process noise covariance is not tuned optimally for this system. The covariance associated with each sensor measurement could be found through each sensor’s documentation. This is used to construct the measurement noise covariance matrix. Because the added mass coefficients and hydrodynamic damping coefficients are borrowed and not specific to the vehicle being used, a covariance of much higher scale than those of the sensors is assigned to the process noise covariance matrix so that the filter will put more faith into the sensor measurements.

The last parameter file, *sensors.yaml* contains the linear offsets between each GNSS antenna and the body-fixed coordinate frame origin. The magnitude of the measured offset is 1.1557m in the positive and negative body-fixed y-direction for both the port and starboard GNSS antennae.

4.2.5 Extended Kalman Filter Algorithm and ROS Wrapper

The EKF is implemented in two parts: the algorithm and the ROS wrapper. The first part, the EKF algorithm, is a Python script that implements the actual mathematics of the filter. The second part, the ROS wrapper, is a node that handles all functions associated with ROS necessary for the EKF algorithm. This includes subscribing to sensor messages and wrench topics and loading parameters from the various parameter files. These values are passed to the

EKF algorithm and the final state estimate is returned back to the ROS wrapper. The ROS wrapper is responsible for publishing the state estimate to a topic. ROS functions are separated from the actual algorithm implementation so that adjustments to the filter can be made without affecting the rest of the code. Similarly, adjustments to ROS topics and messages will not affect the actual EKF algorithm as long as the correct variables are passed from the wrapper.

CHAPTER 5: ROBOT_LOCALIZATION

5.1 Background

The *robot_localization* package estimates a vehicle's linear position and angular orientation, velocity, and acceleration about all three of its body-fixed axes. The package is capable of fusing data from an arbitrary number of odometry, IMU, pose, and velocity sensors using two nonlinear filtering options: the Extended Kalman Filter and the Unscented Kalman Filter (UKF)[43]. For this thesis, the EKF option is used. The state propagation model used in the *robot_localization* package's EKF is a 3D kinematic model based on Newtonian mechanics. This is ideal for unmanned ground vehicles (UGV) subject to kinematic constraints, a characteristic that the USV lacks. Testing on a UGV has resulted in standard deviations of 0.54m in the x-position estimate and 0.34m in the y-position estimate when fusing measurements from an odometer, two IMUs, and two GPSs. After driving through a loop trajectory, the positional loop closure error was reported to be 0.79m and 0.58m in the map frame x-direction and y-direction, respectively[44]. While the *robot_localization* package is capable of providing state estimates in six degrees of freedom, a parameter called "2d_mode" is marked true for this work so that the package only provides estimates in three degrees of freedom.

5.2 Configuration

5.2.1 EKF Localization Node

The EKF localization node is responsible for the actual sensor fusion. Configuring the EKF localization node requires users to edit the parameter file associated with the *robot_localization* package. Parameters for the EKF include frequency, process noise covariance, initial estimate covariance, sensor timeout, 2D mode, transform time offset, transform timeout, and debugging. The name of the coordinate frames must also be specified. These include the name of the world frame, map frame, odom frame, and base link frame. For this application, the world frame and the map frame are chosen to coincide with the NED frame at base station and the base link frame coincides with the body fixed frame of the WAMV. The odom frame is a drifting frame that accounts for bias caused by incremental sensors. This frame is only specified to provide a

transformation between the base link and the map frame. Finally, each sensor has an associated coordinate frame to represent its location and orientation with respect to the body fixed frame. The transformation tree depicting relationships between each coordinate frame is shown in Figure 17.

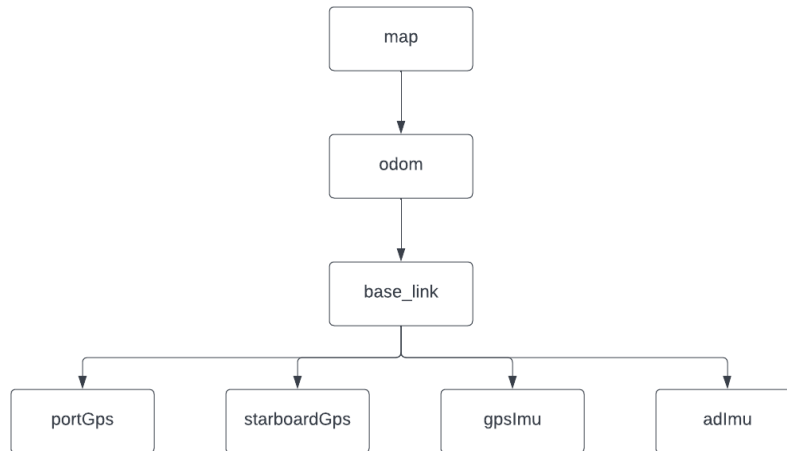


Figure 17: Transformation tree for *robot_localization*

When using incremental sensors like the IMU, two instances of the EKF localization node are used to create a state estimate. The first node creates a state estimate in the map frame by fusing all sources of sensor data. This estimate is the final navigation solution to be used with guidance and control algorithms. The second EKF localization node fuses data coming from incremental sensors only. This node is used to produce the odom frame transformation between the map frame and base link frame. This characterizes the drift in the estimates caused by sensor bias. The estimate produced by the second node is not used.

In addition to parameters directly related to the EKF, parameters related to each source of sensor information must also be defined. For this work, two sets of parameters for the positioning data from the Navsat transform node and two sets of IMU are defined. Important parameters include a configuration matrix that specifies what data is being fused from the sensor and the frame ID used in the sensor’s published message. The parameter file used is shown in Appendix A.

The coordinate frames used by *robot_localization* align with ROS REP-105 standards. The default map frame is defined as an East-North-Up (ENU) coordinate frame located at the

vehicle’s starting position, but can be specified using the datum parameter to coincide with any position. In accordance with REP-103 standards, angular velocity and linear acceleration measurements must be defined with positive x-axes forward, positive y-axis left, and positive z-axis up. Yaw orientation measurements must also be defined with respect to East and with a positive z-axis upwards. Because the measurements produced by the two IMU sensors do not align with these standards, a separate script is used to transform and publish measurements with the correct axes. The script is also used to change the frame ID of each sensor in their published messages in order to create a transformation tree, as seen in Figure 17. The software structure is depicted in Figure 18.

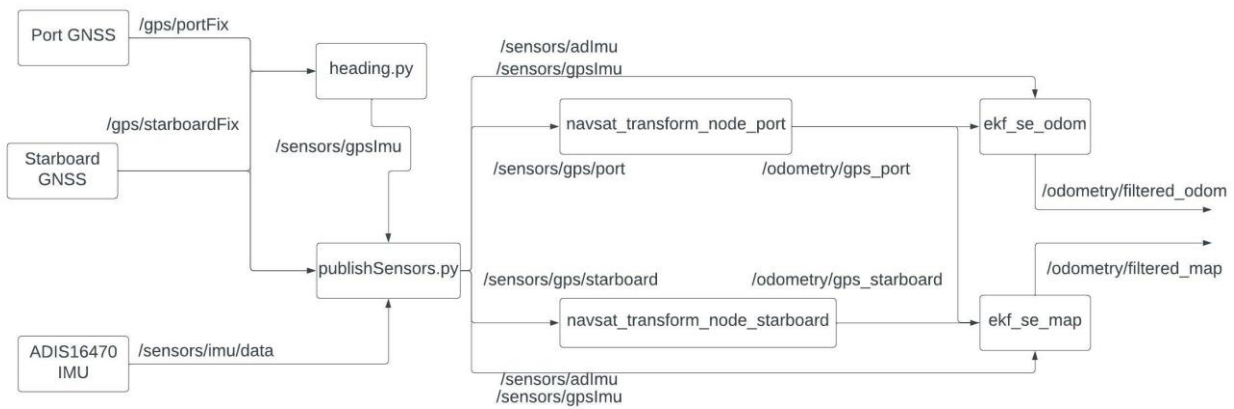


Figure 18: *robot_localization* software flow chart with sensor editor script

5.2.2 Navsat Transform Node

The Navsat transform node transforms GPS decimal degree position measurements into the *robot_localization* map frame coordinates and publishes the data to a ROS topic with the *nav_msgs/Odometry* message type. Important parameters to be specified for the Navsat transform node include the frequency, magnetic declination, yaw offset, and datum. For this work, a frequency of 10Hz is chosen. The magnetic declination is specified as 0.1654572rad, corresponding to the test location of Sand Island, Hawaii. For each instance of the Navsat transform node, an associated heading measurement must be specified. The sensor message used is the heading calculation from the two GPS measurements, as described in Section 4.2.1. This heading calculation is converted to be with respect to an ENU coordinate frame, so that the yaw

offset can be specified as 0rad. Lastly, the datum parameter indicates the location of the map frame. By default, the map frame is defined at the vehicle's starting position. For this work, the map frame is chosen to coincide with the base station RTK GNSS antenna's coordinates: (21.311773, -157.889303).

CHAPTER 6: PERFORMANCE COMPARISON

6.1 Virtual RobotX Background

The simulation environment used to test and compare the *robot_localization* package and the EKF with vehicle dynamics is the Virtual RobotX (VRX) simulation. The VRX software simulates a WAMV operating out of the Sand Island beach park. It includes the vehicle dynamics of the WAMV, along with the environmental wind, wave, and current characteristics of Sand Island beach park[42]. Conveniently, the same vehicle and beach are used to collect data for the real-world comparison discussed in Section 6.2. This simulation is useful because it allows comparisons between the two filters to be made with respect to a ground truth state. Using this software also provides information on how the EKF with vehicle dynamics will perform when the vehicle parameters used are approximated and do not exactly match the USV's actual parameters.

6.2 Results

The two navigation solutions are compared for three maneuvers in the simulation environment: a straight line maneuver, a wave-like maneuver, and a random driving maneuver. The same sensor datasets are used for both the *robot_localization* package and the EKF with vehicle dynamics for consistency.

6.2.1 Straight Line Maneuver Simulation

For the straight line maneuver, full forward thrust is produced by both thrusters in differential configuration. Sensor data collection begins after the vehicle reaches a constant velocity in order to create a simple expected behavior for the first test. By beginning recording after a steady-state velocity has been reached, turning motion caused by uneven thruster forces upon startup is minimized. Data continues to be collected for roughly two minutes at steady state.

For this maneuver, the vehicle begins data collection at a Northeast position and travels in the Southwest direction. As seen in Figure 19, the position estimate produced by both the EKF with vehicle dynamics is nearly identical to the ground truth position. The original position estimate produced by *robot_localization*, labelled “*robot_localization (Raw)*”, is initialized at the

same location as the ground truth, but immediately begins deviating North of the ground truth trajectory, increasing as the maneuver continues. This error is caused by an intermediary transformation used by *robot_localization* to Universe Transverse Mercator (UTM) coordinates when converting from GPS coordinates to map frame coordinates. This transformation utilizes raw IMU heading data, incorporating unaccounted for error into the position estimate after filtering. To avoid incorporating this error, the filtered position estimates can be recorded in GPS coordinates and transformed into map frame coordinates using a separate script. This drastically improves the position estimates, labelled “*robot_localization*” in Figure 19, in which the three trajectories are coincident.

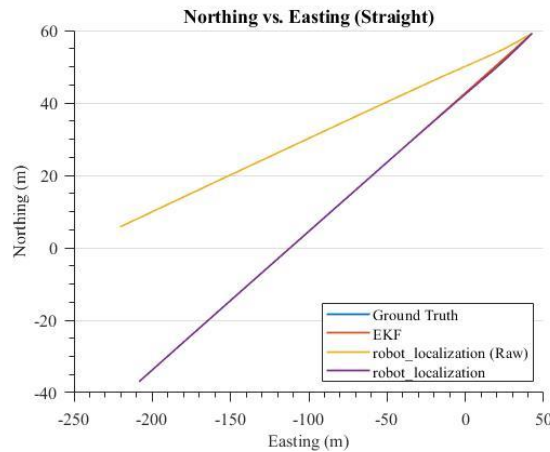


Figure 19: Straight line simulation trajectory. The ground truth curve and EKF curves lie beneath the *robot_localization* curve.

The EKF with vehicle dynamics produces a heading plot that lies on top of the ground truth heading curve, as seen in Figure 20. The heading curve produced by the *robot_localization* package is initially noisier and offset from the ground truth curve, but settles to lie coincident with the ground truth after roughly 30s.

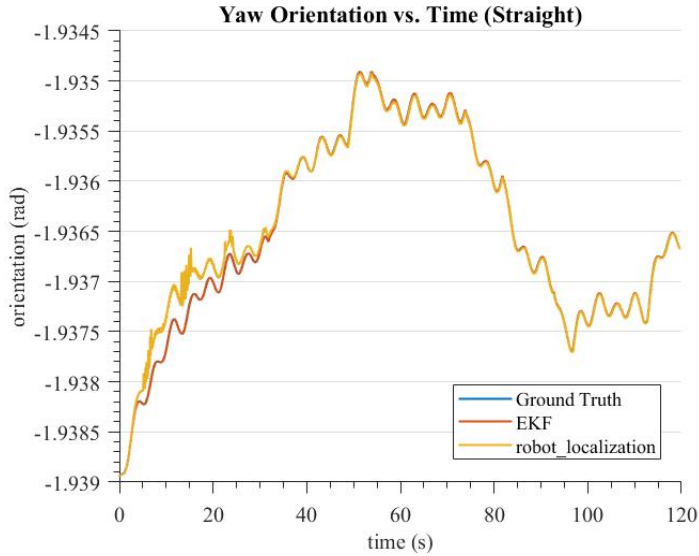


Figure 20: Straight line simulation yaw orientation. The EKF and ground truth curves lie beneath the *robot_localization* curve.

As seen in Figure 21, the EKF with vehicle dynamics produces an initial surge velocity estimate of -2.8m/s, but immediately converges to a steady state velocity with oscillations of roughly 0.1m/s magnitude about the ground truth surge velocity. The *robot_localization* package produces an initial estimate of 0m/s, but converges to lie coincident with the ground truth surge velocity after roughly 15s.

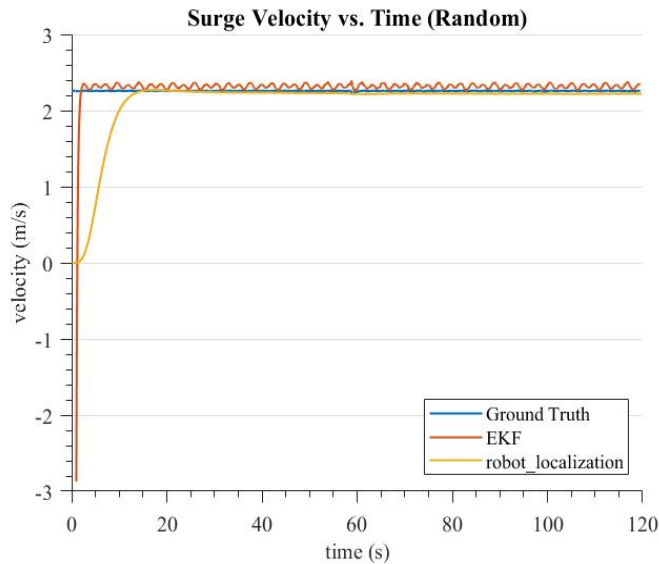


Figure 21: Straight line simulation surge velocity

The EKF with vehicle dynamics produces an initial sway velocity estimate of -22m/s, but immediately converges to the ground truth velocity of 0m/s with slight noise, as seen in Figure 22. The sway velocity estimate produced by *robot_localization* begins at 0m/s, but slowly diverges from the constant ground truth velocity of 0m/s.

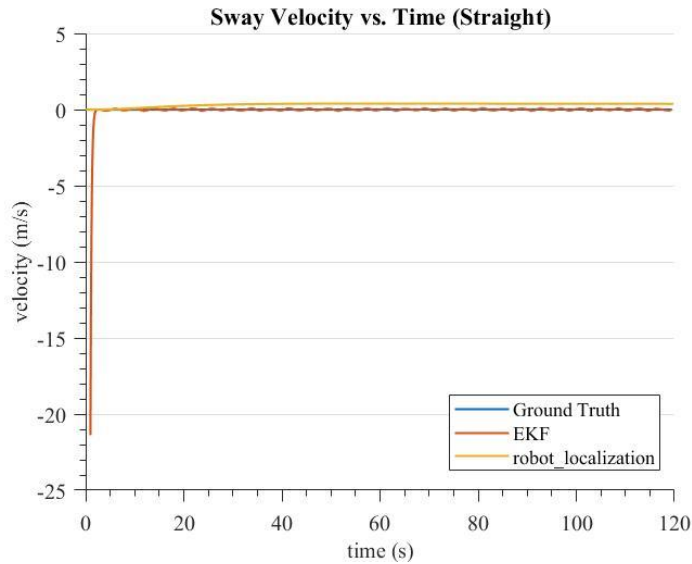


Figure 22: Straight line simulation sway velocity

The angular velocity estimates produced by both navigation solutions have the same magnitudes, both slightly smaller than the ground truth angular velocity, shown in Figure 23. Both filters detect changes almost immediately, with their curves lying on top of the ground truth curve.

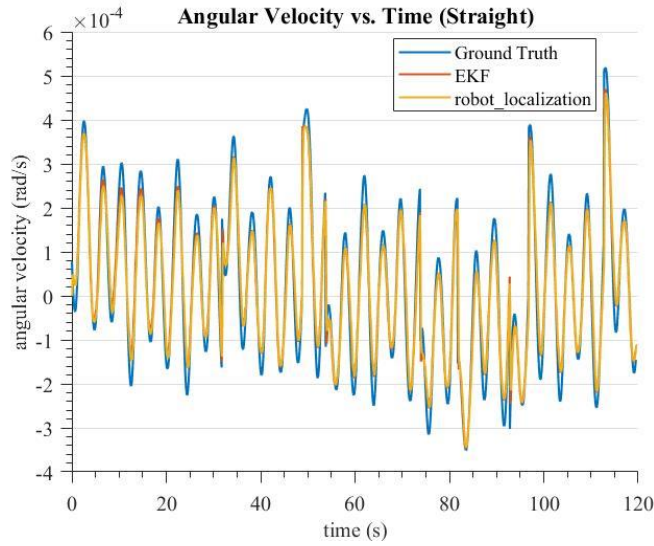


Figure 23: Straight line simulation angular yaw velocity

6.2.2 Wave-Like Maneuver Simulation

For this maneuver the WAMV begins at rest and full forward thrust is produced by both thrusters until the WAMV reaches a steady state velocity. After reaching steady state velocity, the starboard thruster is sent a command of zero thrust while the port thruster continues to produce full thrust for 10s. After the 10s period, the port thruster is sent a command of zero thrust and the starboard thruster is sent a command of full forward thrust. The thruster commands switch every 10s. Data collection begins after the periodic turning.

The WAMV is initially at a Northeast location and moves in the Southwest direction, as seen in Figure 24. Using the raw position estimates by *robot_localization* results in a similar trajectory to the ground truth that again deviates to the North. Curves evident in the ground truth trajectory are also smoothed out in the *robot_localization* estimate. After manual conversion between GPS coordinates and NED coordinates, the trajectory estimate produced by both filters lie nearly coincident with the ground truth trajectory.

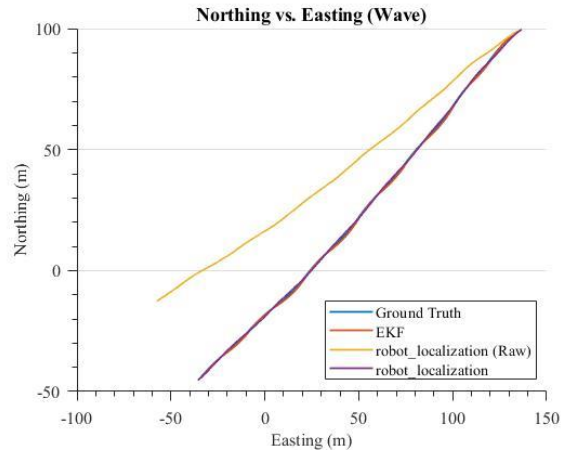


Figure 24: Wave maneuver simulation trajectory. The ground truth curve and the EKF curve lie beneath the *robot_localization* curve with fine differences.

The heading estimates produced by the both the EKF with vehicle dynamics and the *robot_localization* package are identical to the ground truth. Their curves lie on top of the ground truth curve, as seen in Figure 25.

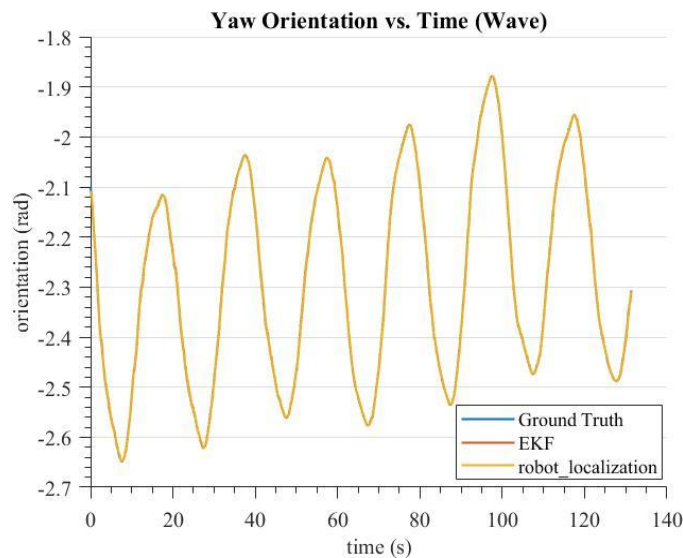


Figure 25: Wave maneuver simulation yaw orientation. The EKF and ground truth curves lie beneath the *robot_localization* curve.

The surge velocity estimate produced by the EKF with vehicle dynamics is initially -1.9m/s and contains noise that lasts for the first 50s of the maneuver, shown in Figure 26. After the noise settles, the surge velocity estimate captures oscillations present in the ground truth. Compared to the ground truth velocity, the magnitude the fluctuations in surge velocity is larger and contains noise. The surge velocity estimate produced by *robot_localization* is initialized at 0m/s and slowly climbs to roughly 1.7m/s after 15s. The estimate struggles to capture the oscillatory changes in the surge velocity that are present in the ground truth. Its steady state value is also lower than the mean ground truth velocity.

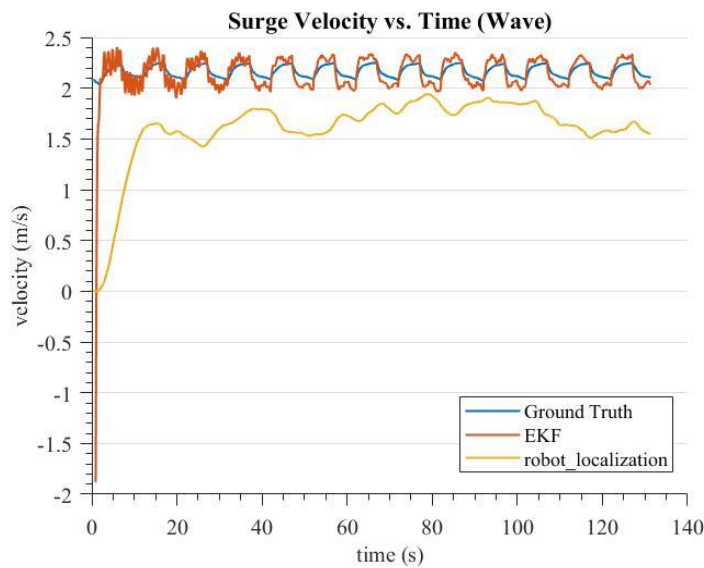


Figure 26: Wave maneuver simulation surge velocity

The sway velocity estimate produced by the EKF with vehicle dynamics is initially 27m/s, but immediately converges to oscillate about 0m/s, depicted in Figure 27. The sway velocity estimate is identical to the ground truth, but contains noise. The estimate produced by the *robot_localization* package is initially 0m/s, where it remains constant throughout the entire maneuver. Oscillations present in the ground truth are not captured.

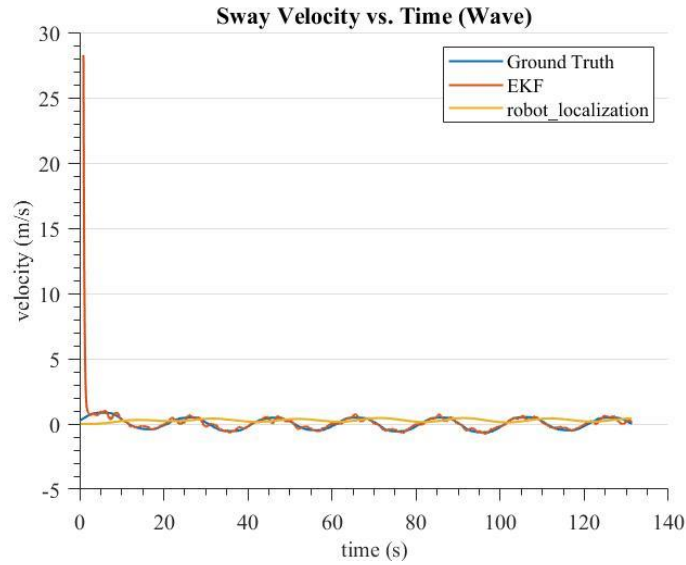


Figure 27: Wave maneuver simulation sway velocity

Similar to the heading estimates, the EKF with vehicle dynamics and *robot_localization* package both produce angular velocity estimates identical to the ground truth, as seen in Figure 28. The ground truth and EKF curves lie beneath the *robot_localization* curve.

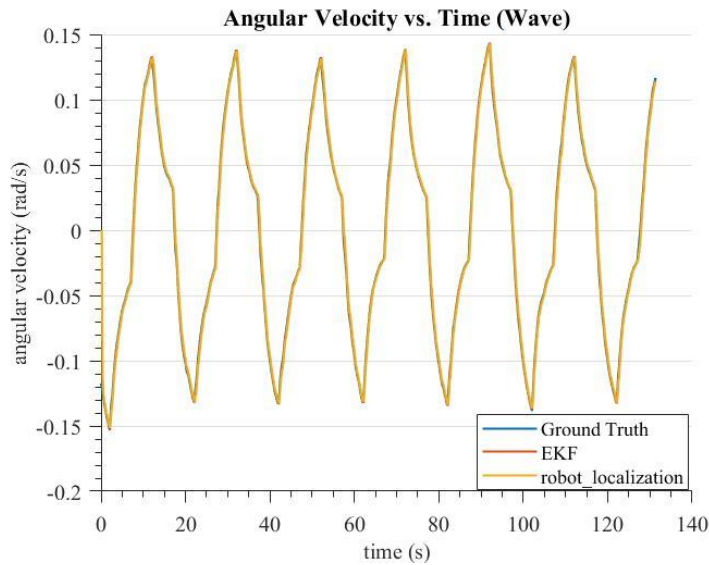


Figure 28: Wave maneuver simulation angular yaw velocity. The EKF and ground truth curves lie beneath the *robot_localization* curve.

6.2.3 Random Maneuver Simulation

The random driving maneuver is completed using a handheld controller with a joystick. The vehicle begins from rest and is driven for roughly 3min, with the maneuver capturing both forward and reverse motion.

The vehicle begins from rest and is driven in the Southeast direction, before turning to move North. Its beginning position is depicted as the center of Figure 29 where the three curves coincide. After travelling North, the vehicle moves in a curve headed West. The EKF with vehicle dynamics produces a position estimate nearly identical to that of the ground truth. Its trajectory lies on top of the ground truth trajectory. Without a manual transformation between GPS coordinates and map NED coordinates, the position estimate produced by *robot_localization* captures the shape of the trajectory, but is skewed West of the ground truth curve, indicated by the curve labelled “*robot_localization (Raw)*”. After a manual conversion between GPS coordinates and NED coordinates, the *robot_localization* position estimates capture the shape of the ground truth trajectory for the majority of the maneuver, but deviates from the ground truth curve for sharper turns.

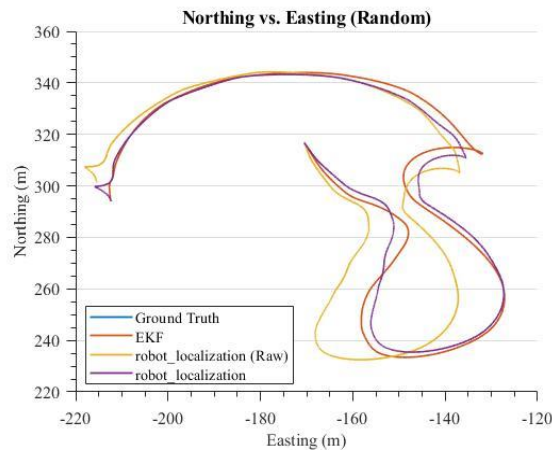


Figure 29: Random maneuver simulation trajectory. The ground truth curve lies beneath the EKF curve

Both *robot_localization* and the EKF with vehicle dynamics successfully capture the changes in heading, as depicted in Figure 30. The estimates produced by both filters are coincident with the ground truth when plotted.

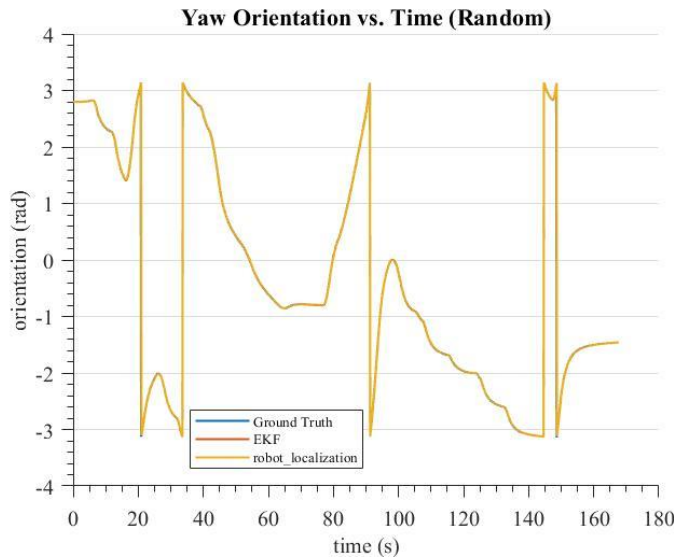


Figure 30: Random driving simulation yaw orientation. The EKF and ground truth curves lie beneath the *robot_localization* curve.

The EKF with vehicle dynamics also captures the changes in surge velocity, as depicted in Figure 31. Throughout the maneuver, the estimated surge velocity is of a larger magnitude than the ground truth for smaller fluctuations and of a smaller magnitude for larger changes. The *robot_localization* package fails to capture much of the finer changes in surge velocity. Its estimates capture larger, slower changes, but filters out smaller fluctuations. The changes that are captured are of a smaller magnitude than the ground truth by roughly 5s.

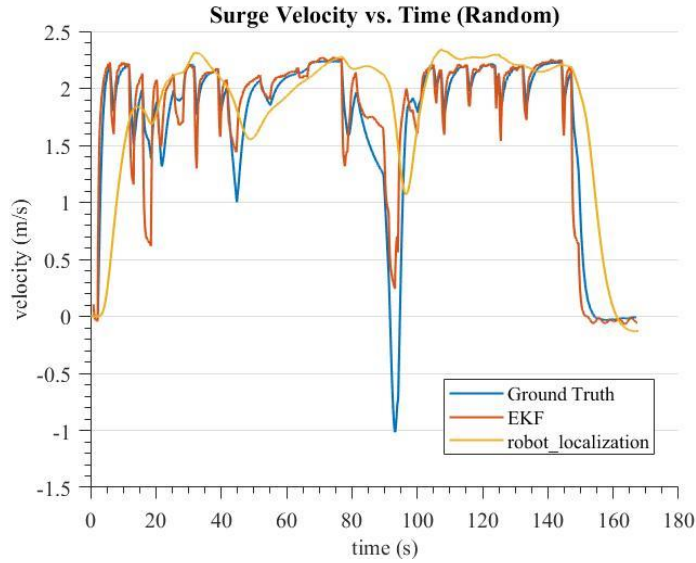


Figure 31: Random driving simulation surge velocity

The sway velocity estimates produced by the EKF with vehicle dynamics successfully captures changes in sway velocity, as seen in Figure 32. The estimate lies on top of the ground truth, but contains noise. The sway velocity estimate produced by *robot_localization* again captures larger, slower changes in velocity, but struggles to capture finer fluctuations. There is a delay of roughly 5s in the sway velocity estimates as well.

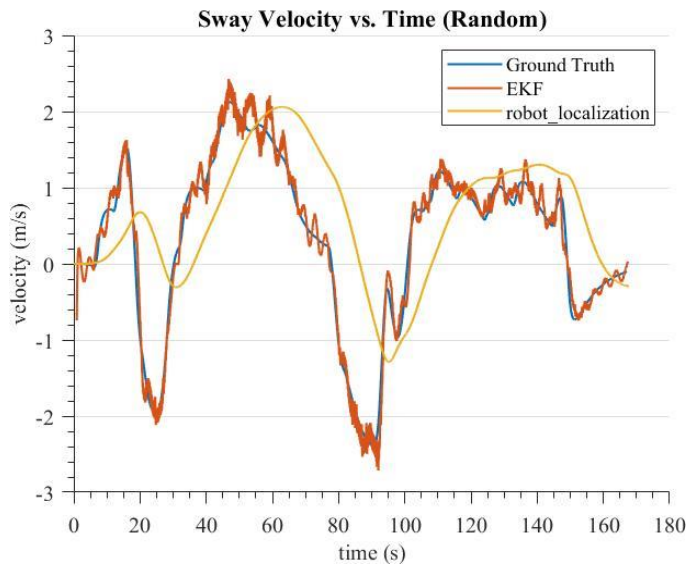


Figure 32: Random driving simulation sway velocity

Similar to the heading estimates, both the *robot_localization* package and the EKF with vehicle dynamics successfully capture changes in angular velocity. Both estimates lie coincident with the ground truth when plotted, as seen in Figure 33.

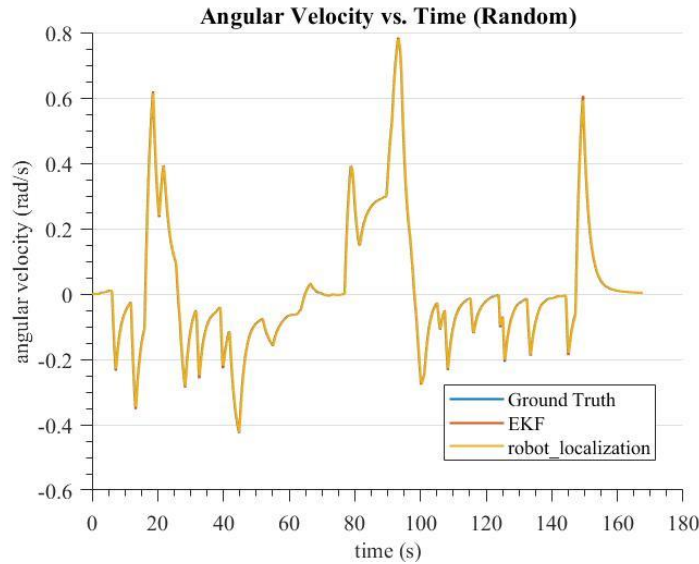


Figure 33: Random driving simulation angular yaw velocity. The EKF and ground truth curves lie beneath the *robot_localization* curve.

6.2.4 Real World Straight Line Maneuver

The straight line maneuver is implemented similarly in the real-world as in simulation. Thrust commands of full forward thrust are sent to the WAMV. Like the simulation maneuver, data collection begins after the USV seems to have reached a steady state forward velocity.

For this maneuver, the WAMV begins at a Northeast position and travels in the Southwest direction. The initial position estimate by the *robot_localization* package begins at (0,0) in the map frame, as seen in Figure 34. This is because the *robot_localization* package initializes its estimate at the datum specified by the user. It immediately updates to match the initial position estimate by the EKF with vehicle dynamics. Without a manual conversion between GPS coordinates and map frame coordinates, the trajectory estimated by the *robot_localization* package deviates to the North of the trajectory estimate produced by the EKF with vehicle dynamics. When the position estimates are manually transformed, the *robot_localization* estimates and the EKF estimates are coincident. The same behavior is seen repeatedly in all three simulated maneuvers.

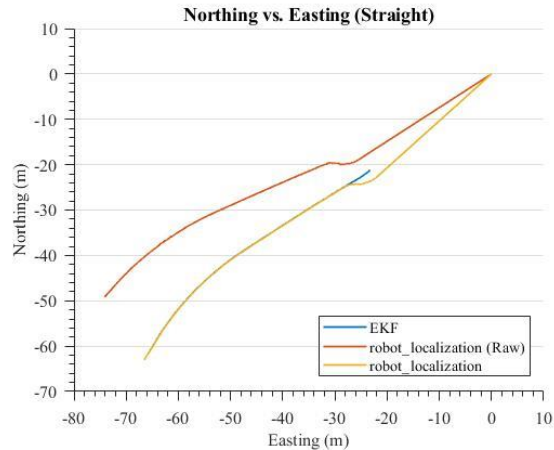


Figure 34: Straight line trajectory for real world. The EKF curve lies beneath the *robot_localization* curve.

The heading estimate produced by the EKF with vehicle dynamics is initially -1.2rad and gradually rotates to -2.8rad , as seen in Figure 35. The initial heading estimate by *robot_localization* is 1.6rad and increases before wrapping to match the downwards trend of the EKF with vehicle dynamics after eight seconds have passed. This behavior is similar to the simulated straight line maneuver, in which the *robot_localization* package produced an estimate that was initially noisy and offset from the EKF with vehicle dynamics, but eventually converged to a similar shape. Unlike the simulated comparison, the two curves do not lie coincident after the *robot_localization* estimates have converged. Based on the simulated straight line maneuver, in which the EKF produced an orientation curve that was immediately and constantly coincident with the ground truth, the EKF with vehicle dynamics is likely more accurate in this real-world maneuver.

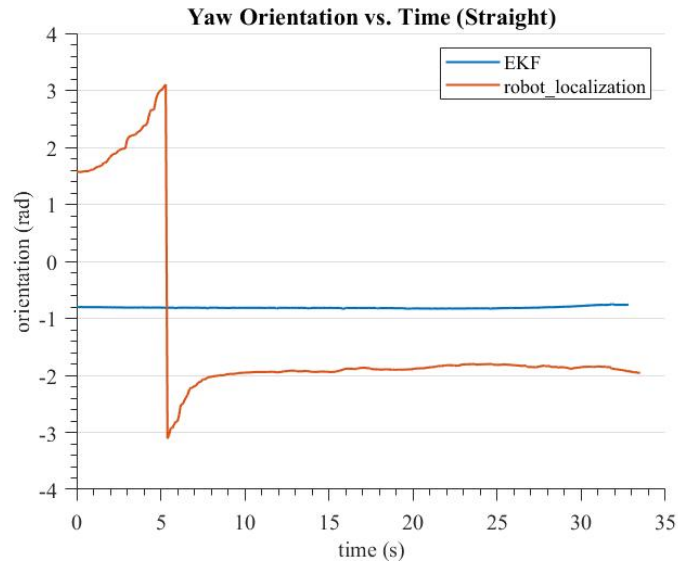


Figure 35: Straight line yaw orientation for real world

The surge velocity estimate produced by the EKF with vehicle dynamics is initially -1.5m/s, but quickly rises and settles to 2.9m/s after two seconds have passed, as seen in Figure 36. The surge velocity estimate produced by *robot_localization* is initially 0m/s and slowly rises to 3.3m/s after 12s have passed. These behaviors are both similar to the simulated straight line maneuver curves. After peaking, the *robot_localization* curve decreases to match the EKF with vehicle dynamics. The EKF with vehicle dynamics produces an estimate with less noise than that of *robot_localization*. This contrasts with the results of the simulated maneuvers, in which the EKF with vehicle dynamics produced surge velocity estimates that were much noisier than the *robot_localization* estimates.

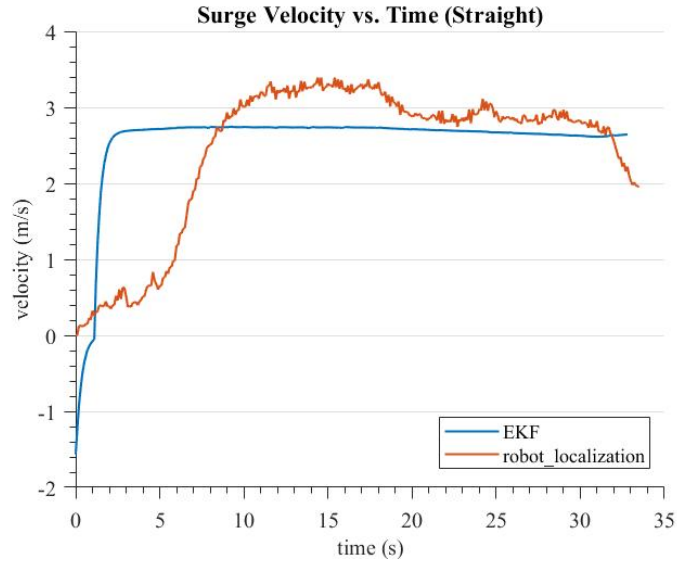


Figure 36: Straight line surge velocity for real world

Like with the surge velocity estimate, the EKF with vehicle dynamics produces an initial sway velocity estimate of -9m/s , underestimating by a large margin. This can be seen in Figure 37. The estimates produced immediately converge to zero, however, where it fluctuates with a maximum peak-to-peak noise of roughly 1m/s . The sway velocity estimates produced by *robot_localization* are again initially 0m/s . They then decrease and converge at roughly -0.5m/s with less noise than the estimates produced by the EKF with vehicle dynamics. This behavior is similar to the simulated behavior, in which the EKF with vehicle dynamics produced more noise in its sway velocity estimates than the *robot_localization* package. Additionally, the shapes of the two curves are similar to the simulated straight line maneuver of Figure 22.

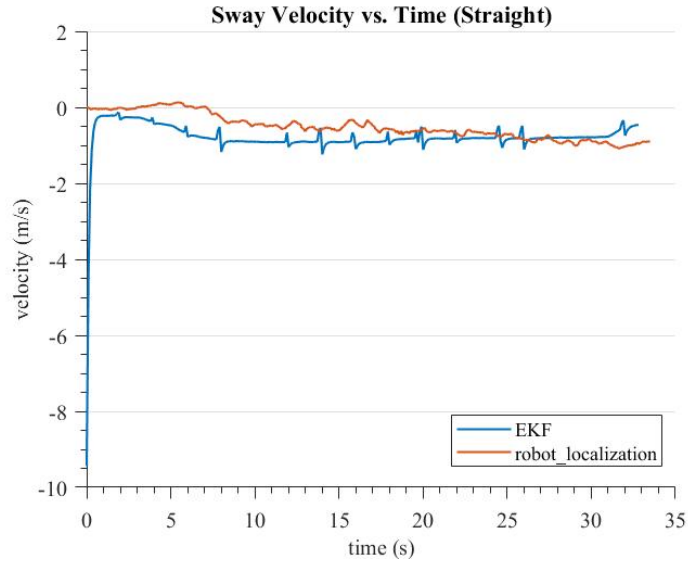


Figure 37: Straight line sway velocity for real world

The behavior of the angular velocity estimates for this maneuver differ from the behavior produced by filters in simulation. As seen in Figure 38, the EKF with vehicle dynamics produces a constant 0rad/s angular velocity estimate. The *robot_localization* package produces an estimate that fluctuates about 0.01rad/s for the first 18s of the maneuver. After 18s, the estimates fluctuate about 0.04rad/s. This is in contrast to the simulated maneuvers, in which the angular velocity estimates produced by the two filters lied coincident.

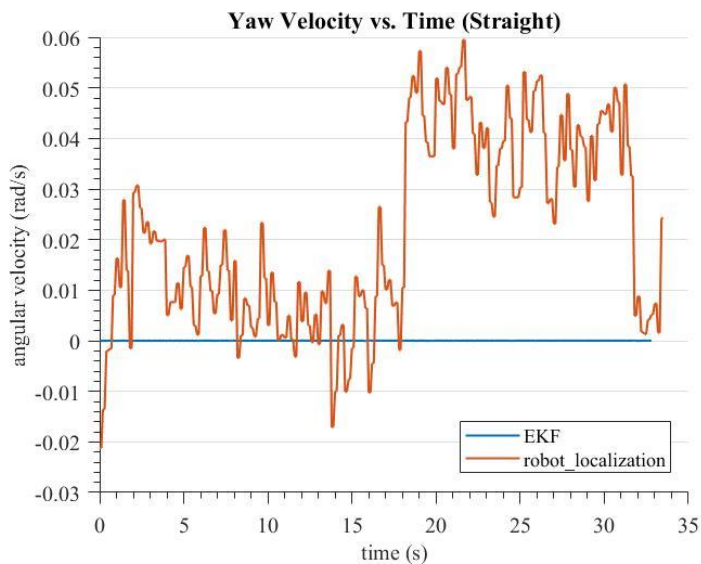


Figure 38: Straight line angular yaw velocity for real world

6.2.5 Real World Curve Maneuver

The curve maneuver is implemented similarly to the real-world straight line maneuver. For this maneuver, the WAMV begins at rest and is sent a command of half maximum forward thrust to the port thruster and full forward thrust to the starboard thruster. Data collection begins after the WAMV has reached a steady-state velocity.

For this maneuver, the WAMV begins at a location and moves in the Southwest direction initially. It continues its trajectory in the counter-clockwise direction when looking down at the ocean from above. As seen in Figure 39, the trajectory estimates produced by *robot_localization* again deviate from the EKF with vehicle dynamics' trajectory estimate when raw positioning estimates are used. When a manual conversion between GPS coordinates and map frame coordinates are performed on the *robot_localization* estimates, the *robot_localization* curve is coincident with the EKF curve. The *robot_localization* position estimates have initial estimate at (0,0) produced by *robot_localization* caused by its initialization at the user-specified datum parameter, which immediately converges to match the starting point of the EKF with vehicle dynamics. The behavior of the trajectory estimates is very similar to all three simulated maneuver trajectory estimates.

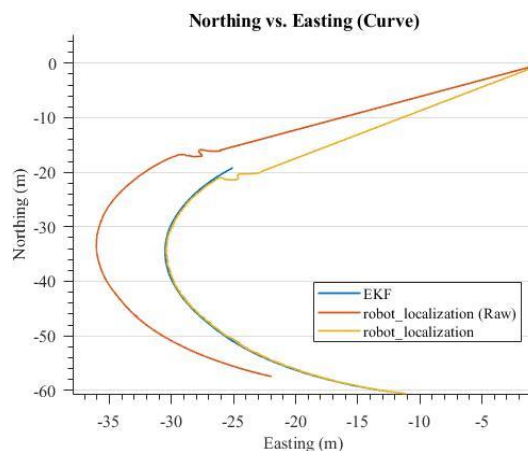


Figure 39: Curve maneuver trajectory for real world. The EKF curve lies beneath the *robot_localization* curve.

The heading estimate produced by the EKF with vehicle dynamics is initially -1rad. The heading estimate decreases rapidly for the first second, before decreasing at a constant negative slope before reaching a final value of 1.8rad, as seen in Figure 40. The heading estimate produced by *robot_localization* is initially 1.6rad. It fluctuates for the first two seconds of the maneuver before gradually decreasing from 3.1rad at a negative slope similar to that of the EKF with vehicle dynamics. Its final estimated heading is roughly 1.6rad. The behavior of the orientation estimates for this maneuver is very similar to the behavior of the orientation estimate of the simulated straight line maneuver and the real-world straight line maneuver.

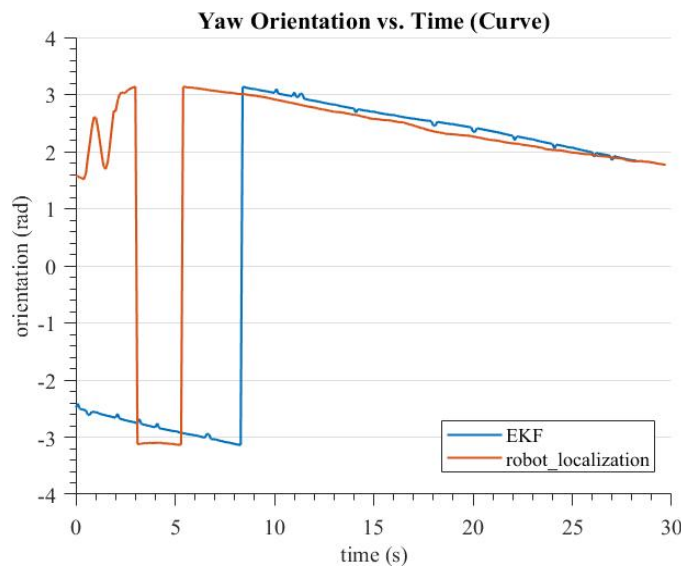


Figure 40: Curve maneuver yaw orientation for real world

For both the surge and sway velocities, the EKF with vehicle dynamics produces a highly negative estimate before immediately converging to its steady-state value, as seen in Figures 41-42. Like the previous subsection, the estimate produced by *robot_localization* for the surge velocity is initialized at 0m/s before gradually increasing to its steady-state value of 2.5-3m/s. The sway velocity estimate produced by *robot_localization* also begins at 0m/s before increasing to 1m/s within the first two seconds of the maneuver. After reaching 1m/s, it decreases very slowly to roughly 0.5m/s. The velocity estimates produced by the EKF with vehicle dynamics have much larger noise than the estimates produced by *robot_localization*. The estimates produced by *robot_localization* have slower fluctuations throughout the maneuver. These

behaviors are expected, as they are also seen in the simulated wave maneuver and the simulated random driving maneuver.

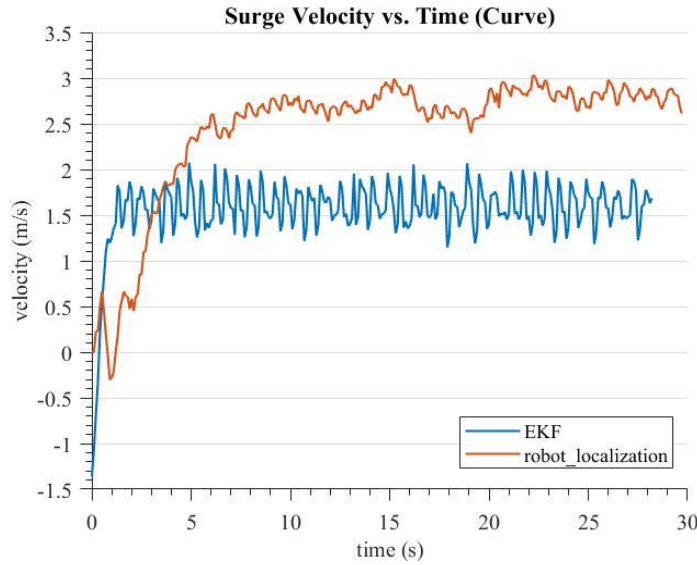


Figure 41: Curve maneuver surge velocity for real world

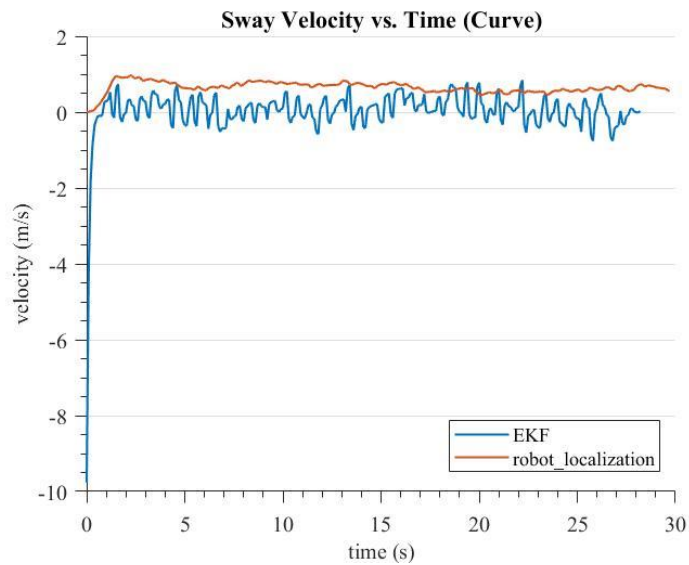


Figure 42: Curve maneuver sway velocity for real world

The angular velocity estimated by both packages have similar shapes, as seen in Figure 43. The only qualitative difference is the roughly 0.5s latency of the *robot_localization* estimate

behind the EKF with vehicle dynamics' estimate. This is in contrast to the simulated maneuvers, in which the angular velocity estimates from both filters were coincident.

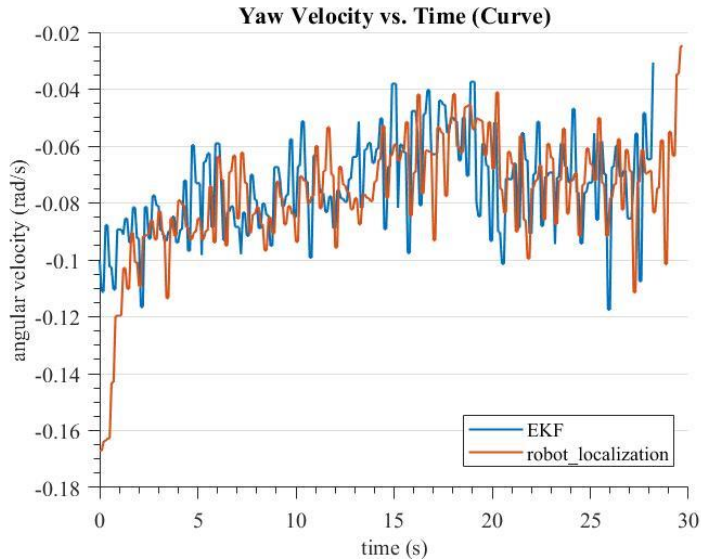


Figure 43: Curve maneuver angular yaw velocity for real world

6.3 Discussion

Both the EKF with vehicle dynamics and the *robot_localization* package successfully capture changes in heading and angular velocity. This is to be expected, as accurate measurements of these state members are available to the filters and the two state members are coupled. As seen in Table 5, these two state members are the most observable in the state vector.

When raw positioning estimates from *robot_localization* are compared, the EKF with vehicle dynamics produces significantly less root-mean-squared error for all three simulated maneuvers, as seen in Table 7. This is also reflected in the distance errors, as seen in Table 8. After manually converting positioning estimates from GPS coordinates to map frame coordinates, the *robot_localization* package produces 0.5334m less root-mean-squared error in its x-positioning estimates and 1.249m less root-mean-squared error in its y-positioning estimates than the EKF with vehicle dynamics for the straight line maneuver in simulation, as seen in Table 7. For the wave maneuver, this difference in error is less significant, with *robot_localization* producing x-position estimates with 0.4080m less error and y-position

estimates with 0.1653m less error than the EKF with vehicle dynamics. The differences in positioning error is most significant in the random driving maneuver with the EKF with vehicle dynamics producing 1.3171m less x-position error and 1.364m less y-position error than *robot_localization*. A similar trend can be seen in the mean distance error, in which the EKF with vehicle dynamics' position estimates have less mean distance offset from the ground truth than the *robot_localization* position estimates for the wave and random driving maneuvers. This is demonstrated in Table 8. A possible source of these differences in errors is the errors in estimates of other state members. Because the position and velocity state members are coupled in both filters, errors in the velocity estimates are propagated into the position estimates.

As seen in Table 7, the EKF with vehicle dynamics produces surge and sway velocity estimates with less error than the *robot_localization* package for all three simulated maneuvers, except the wave maneuver, in which *robot_localization* produces surge estimates with 0.2104m/s less error than the EKF with vehicle dynamics. This can be explained by the fact that the surge and sway velocities are the least observable members of the state vector, as there is no direct velocity measurement. Including vehicle dynamics into the state propagation matrix of the EKF allows the filter more information about the unmeasured states. This gives the EKF with vehicle dynamics the ability to detect finer changes in velocity and with less time delay, as seen in in the velocity plots of the wave maneuver and random driving maneuver (Figures 26, 27, 31, 32).

Table 5: Observability Ranking

Ranking	1	2	3	4	5	6
State Member	ψ	r	x	y	u	v

Errors in the WAMV's characteristic parameters can contribute to errors in surge velocity and sway velocity estimates. For both the wave maneuver and the random driving maneuver, surge and sway velocity estimates produced by the EKF with vehicle dynamics indicate fluctuations that are larger than the ground truth for smaller changes and smaller than the ground truth for larger changes. By contrast, the surge and sway velocity estimates produced by *robot_localization* have a different effect. The *robot_localization* package filters out faster and smaller changes in velocity. This heavier filtering effect demonstrates a tradeoff between accuracy and noise. As seen in Table 6, the *robot_localization* package produces surge and sway

velocity estimates with a much lower noise covariance than the EKF with vehicle dynamics. This is also demonstrated in Figures 19-43, where it can be seen that estimates produced by *robot_localization* are much smoother and have less noise than the EKF with vehicle dynamics. The EKF with vehicle dynamics produces estimates with a higher accuracy, as seen in Table 7.

Table 6: Average filter covariance

Maneuver	Filter	x	y	ψ	u	v	r
Straight (Simulation)	EKF	0.001576	0.02103	$1.000*10^{-7}$	7.119	146.8	$1.000*10^{-7}$
	<i>robot_localization</i>	0.04421	0.03926	0.005340	0.005883	0.001861	0.005343
Wave (Simulation)	EKF	0.001576	0.02103	$1.000*10^{-7}$	7.119	146.8	$1.000*10^{-7}$
	<i>robot_localization</i>	0.04342	0.04220	0.005647	0.005992	0.001834	0.005651
Random (Simulation)	EKF	0.001576	0.02103	$1.000*10^{-7}$	7.119	146.8	$1.000*10^{-7}$
	<i>robot_localization</i>	0.04271	0.04281	0.005329	0.006233	0.001987	0.005320
Straight (Real World)	EKF	0.0007880	0.01055	$9.999*10^{-8}$	8.985	101.6	$1.000*10^{-7}$
	<i>robot_localization</i>	0.02477	0.02479	0.5212	0.8309	5.353	0.01825
Curve (Real World)	EKF	0.0007880	0.01055	$9.999*10^{-8}$	8.982	101.7	$1.000*10^{-7}$
	<i>robot_localization</i>	0.04732	0.04737	0.4651	0.9665	4.155	0.01464

Despite producing estimates with much higher noise, the EKF with vehicle dynamics is able to detect finer changes in the state for less-observable state members. This is increasingly true for maneuvers that include turning. For the two most observable states, heading orientation and angular velocity, the two filters produce estimates that have root-mean-squared errors of the same magnitude. For the less observable states, the surge velocity and sway velocity, the EKF with vehicle dynamics produces estimates with smaller error. Additionally, the EKF with vehicle dynamics produces a smaller mean distance error in its trajectory estimates, as seen in Table 8.

Table 7: Root-mean-squared error for simulated maneuvers

Maneuver	Filter	x	y	ψ	u	v	r
Straight (Simulation)	EKF	0.8464	2.173	$1.346 \cdot 10^{-5}$	0.06602	0.04600	$4.950 \cdot 10^{-5}$
	<i>robot_localization</i> (raw)	24.91	6.926	$1.107 \cdot 10^{-4}$	0.3702	0.3464	$5.424 \cdot 10^{-5}$
	<i>robot_localization</i>	0.3110	0.9241	$1.107 \cdot 10^{-4}$	0.3702	0.3464	$5.424 \cdot 10^{-5}$
Wave (Simulation)	EKF	1.192	1.878	0.006726	0.08192	0.1378	0.007714
	<i>robot_localization</i> (raw)	18.98	13.02	0.005953	0.6088	0.5262	0.009739
	<i>robot_localization</i>	1.0267	1.470	0.005953	0.6088	0.5262	0.009739
Random (Simulation)	EKF	0.06790	0.05327	0.008603	0.2933	0.1747	0.07647
	<i>robot_localization</i> (raw)	4.938	6.352	0.007567	0.5988	1.001	0.01185
	<i>robot_localization</i>	1.385	1.417	0.007567	0.5988	1.001	0.01185

Table 8: Mean distance error for simulated maneuver in meters

	Straight Line	Wave	Random
EKF	2.308	1.688	0.0700
<i>robot_localization</i> (raw)	22.31	19.88	7.277
<i>robot_localization</i>	0.4530	0.7817	1.699

The *robot_localization* package produces estimates with less noise than the EKF with vehicle dynamics for less-observable state members, however its filtering effect can be too extreme, causing it to filter out actual changes in the state. In simulations, the EKF with vehicle dynamics produces noisier estimates, but captures changes in the vehicle’s state more quickly and more accurately than the *robot_localization* package. For real-world maneuvers, the EKF with vehicle dynamics is again noisier than the *robot_localization* package for less-observable state members. Assuming the filters behave similarly in the real-world as they do in the simulation, it is reasonable to conclude that the EKF with vehicle dynamics could have a lower steady-state error in the real-world application as well.

CHAPTER 7: CONCLUSION

In this work, a 3DOF mathematical model of 16ft WAMV's dynamics was created. This model was incorporated into the state propagation model of an Extended Kalman Filter, which was used to fuse position, heading, angular velocity, and acceleration data to create a state estimate of the USV's position, heading, surge velocity, sway velocity, and angular velocity. The resulting EKF was implemented using Python and ROS. Resulting estimates for three simulated maneuvers and two real-world maneuvers were compared to estimates produced by the *robot_localization* package, which utilized a generic omni-directional robot model in its state propagation.

Although including the vehicle's dynamics reduces error for less-observable state members, the vehicle's model parameters must be known to a reasonable accuracy. If the parameters used to in the filter differ severely from the vehicle's actual model parameters, the EKF can become unstable. In cases where finding a reasonable estimate of the vehicle's model parameters is not possible, using the *robot_localization* package could be a valid option.

Future improvements that can be made to the EKF with vehicle dynamics includes expanding the EKF to a nonlinear 6DOF model and including environmental forces like wind, wave, and current. This would allow the EKF implementation to be applied to other vehicles like unmanned ground vehicles, unmanned underwater vehicles, and unmanned aerial vehicles by simply changing the model parameters. This would also improve the quality of the estimates produced.

This work demonstrates that the effects of including vehicle dynamics includes comparable performance for highly observable states, a lower root-mean-square error for less observable states, which is exacerbated by turning, lower latency, and higher noise than the generic model. While a generic state propagation model produces estimates with less noise for less-observable state members, the simulation results indicate that smoothing effects can be too severe, ignoring actual changes in the vehicle's state.

REFERENCES

- [1] M. L. McKinney, “On Predicting Biotic Homogenization: Species-Area Patterns in Marine Biota,” *Source Glob. Ecol. Biogeogr. Lett.*, vol. 7, no. 4, pp. 297–301, 1998.
- [2] J. G. B. Derraik, “The pollution of the marine environment by plastic debris: a review,” *Mar. Pollut. Bull.*, vol. 44, no. 9, pp. 842–852, Sep. 2002, doi: 10.1016/S0025-326X(02)00220-5.
- [3] J. R. Henderson, “A Pre- and Post-MARPOL Annex V Summary of Hawaiian Monk Seal Entanglements and Marine Debris Accumulation in the Northwestern Hawaiian Islands, 1982–1998,” *Mar. Pollut. Bull.*, vol. 42, no. 7, pp. 584–589, Jul. 2001, doi: 10.1016/S0025-326X(00)00204-6.
- [4] S. F. Rey, J. Franklinid, and S. J. Rey, “Microplastic pollution on island beaches, Oahu, Hawai’i,” *PLoS One*, 2021, doi: 10.1371/journal.pone.0247224.
- [5] M. Eriksen *et al.*, “Plastic Pollution in the World’s Oceans: More than 5 Trillion Plastic Pieces Weighing over 250,000 Tons Afloat at Sea,” *PLoS One*, 2014, doi: 10.1371/journal.pone.0111913.
- [6] H. C. Chang, Y. L. Hsu, S. S. Hung, G. R. Ou, J. R. Wu, and C. Hsu, “Autonomous water quality monitoring and water surface cleaning for unmanned surface vehicle,” *Sensors*, vol. 21, no. 4, pp. 1–21, Feb. 2021, doi: 10.3390/s21041102.
- [7] F. P. Samaniego, D. G. Reina, S. L. T. Marin, M. Arzamendia, and D. O. Gregor, “A Bayesian Optimization Approach for Water Resources Monitoring through an Autonomous Surface Vehicle: The Ypacarai Lake Case Study,” *IEEE Access*, vol. 9, pp. 9163–9179, 2021, doi: 10.1109/ACCESS.2021.3050934.
- [8] H. Cao, Z. Guo, S. Wang, H. Cheng, and C. Zhan, “Intelligent wide-area water quality monitoring and analysis system exploiting unmanned surface vehicles and ensemble learning,” *Water*, vol. 12, no. 3, Mar. 2020, doi: 10.3390/w12030681.
- [9] F. Madricardo *et al.*, “How to Deal With Seafloor Marine Litter: An Overview of the State-of-the-Art and Future Perspectives,” *Front. Mar. Sci.*, vol. 7, Sep. 2020, doi: 10.3389/fmars.2020.505134.
- [10] M. Ludvigsen *et al.*, “Use of an Autonomous Surface Vehicle reveals small-scale diel vertical migrations of zooplankton and susceptibility to light pollution under low solar irradiance,” *Sci. Adv.*, vol. 4, no. 1, 2018, [Online]. Available: <https://www.science.org>.
- [11] Department of Defense, “Sea disposal of military munitions, defense environmental programs. Annual Report to Congress, Fiscal Year 2009,” 2009.
- [12] M. H. Edwards *et al.*, “The Hawaii Undersea Military Munitions Assessment,” *Deep Sea Res. Part II Top. Stud. Oceanogr.*, vol. 128, pp. 4–13, Jun. 2016, doi: 10.1016/J.DSR2.2016.04.011.
- [13] D. Rudolph and T. A. Wilson, “Doppler Velocity Log Theory and Preliminary Considerations for Design and Construction,” Mar. 2012, doi: 10.1109/SECon.2012.6196918.
- [14] P. Catania *et al.*, “Positioning accuracy comparison of GNSS receivers used for mapping and guidance of agricultural machines,” *Agronomy*, vol. 10, no. 7, 2020, doi: 10.3390/agronomy10070924.
- [15] A. Karmozdi, M. Hashemi, and H. Salarieh, “Design and practical implementation of kinematic constraints in Inertial Navigation System-Doppler Velocity Log (INS-DVL)-based navigation,” *Navig. J. Inst. Navig.*, vol. 65, no. 4, 2018, doi: 10.1002/navi.271.

- [16] P. M. Lee *et al.*, “An Integrated Navigation System for Autonomous Underwater Vehicles with Two Range Sonars, Inertial Sensors and Doppler Velocity Log,” 2004, doi: 10.1109/OCEANS.2004.1406359.
- [17] J. Snyder, “Doppler Velocity Log (DVL) Navigation for Observation-Class ROVs,” Sep. 2010, doi: 10.1109/OCEANS.2010.5664561.
- [18] H. C. Woithe, D. Boehm, and U. Kremer, “Improving Slocum Glider Dead Reckoning Using a Doppler Velocity Log,” 2011, doi: 10.23919/OCEANS.2011.6107296.
- [19] G. Welch and G. Bishop, “An Introduction to the Kalman Filter.” 1997, [Online]. Available: <http://www.cs.unc.edu/~gb>.
- [20] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics*. MIT Press, 2005.
- [21] M. S. Arulampalam, S. Maskell, N. Gordon, and T. Clapp, “A tutorial on particle filters for online nonlinear/non-Gaussian Bayesian tracking; A tutorial on particle filters for online nonlinear/non-Gaussian Bayesian tracking,” 2002. doi: 10.1109/78.978374.
- [22] F. Gustafsson and S. Member, “Particle Filter Theory and Practice with Positioning Applications,” 2010. doi: 10.1109/MAES.2010.5546308.
- [23] G. G. Rigatos, “Nonlinear Kalman Filters and Particle Filters for integrated navigation of unmanned aerial vehicles,” *Rob. Auton. Syst.*, vol. 60, no. 7, pp. 978–995, Jul. 2012, doi: 10.1016/J.ROBOT.2012.03.001.
- [24] N. Y. Ko and T. G. Kim, “Comparison of Kalman filter and particle filter used for localization of an underwater vehicle,” in *2012 9th International Conference on Ubiquitous Robots and Ambient Intelligence, URAI 2012*, 2012, pp. 350–352, doi: 10.1109/URAI.2012.6463013.
- [25] M. I. Ribeiro, “Kalman and Extended Kalman Filters: Concept, Derivation and Properties,” 2004.
- [26] T. I. Fossen, *Handbook of marine craft hydrodynamics and motion control*. Wiley, 2011.
- [27] T. S. Schei, “A finite-difference method for linearization in nonlinear estimation algorithms,” *Automatica*, vol. 33, no. 11, pp. 2053–2058, Nov. 1997, doi: 10.1016/S0005-1098(97)00127-1.
- [28] M. Nørsgaard, N. K. Poulsen, and O. Ravn, “New developments in state estimation for nonlinear systems,” *Automatica*, vol. 36, no. 11, pp. 1627–1638, Nov. 2000, doi: 10.1016/S0005-1098(00)00089-3.
- [29] F. M. Ham and R. Grover Brown, “Observability, Eigenvalues, and Kalman Filtering,” *IEEE Trans. Aerosp. Electron. Syst.*, vol. AES-19, no. 2, pp. 269–273, 1983, doi: 10.1109/TAES.1983.309446.
- [30] S. Han, S. Wales, and J. Wang, “Monitoring Degree of Observability in GPS/INS Integration.” [Online]. Available: <https://www.researchgate.net/publication/228997493>.
- [31] Marine Advanced Robotics, “Marine Advanced Robotics - WAM-V 16 ASV.” <https://www.wam-v.com/wam-v-16-asv> (accessed Apr. 07, 2022).
- [32] Johnson Outdoors Marine Electronics, “Riptide Terrova Bow-Mount Trolling Motor Owner’s Manual,” 2019.
- [33] “SparkFun GPS-RTK2 Board - ZED-F9P (Qwiic),” 2018. <https://www.sparkfun.com/products/15136> (accessed Apr. 14, 2022).
- [34] “Taoglas MagmaX2 AA.175 Antenna,” 2022. <https://www.mouser.com/new/taoglas/taoglas-aa175-antenna/> (accessed Apr. 14, 2022).
- [35] Kainani Santos and Brennan Yamamoto, “Stationary RTK GNSS Data.” Honolulu, 2020.
- [36] “ADIS16470,” 2022. <https://www.analog.com/en/products/adis16470.html#product->

- overview (accessed Apr. 14, 2022).
- [37] R. Tajima, “adi_driver.” 2022, Accessed: Apr. 18, 2022. [Online]. Available: https://github.com/tork-a/adi_driver.
 - [38] A. Patrik *et al.*, “GNSS-based navigation systems of autonomous drone for delivering items,” doi: 10.1186/s40537-019-0214-3.
 - [39] A. Upadhyay, “Formula to Find Bearing or Heading angle between two points: Latitude Longitude,” *GISMAP*. <https://www.igismap.com/formula-to-find-bearing-or-heading-angle-between-two-points-latitude-longitude/> (accessed Apr. 06, 2022).
 - [40] B. Bingham, “Local Coordinate Frames,” *NPS Wiki*, Mar. 30, 2017. <https://wiki.nps.edu/display/RC/Local+Coordinate+Frames> (accessed Apr. 07, 2022).
 - [41] C. Murphy and H. Singh, “Rectilinear Coordinate Frames for Deep Sea Navigation,” 2010. doi: 10.1109/AUV.2010.5779654.
 - [42] B. Bingham *et al.*, “Toward Maritime Robotic Simulation in Gazebo,” Oct. 2019, [Online]. Available: <https://bitbucket.org/osrf/vrx/>.
 - [43] T. Moore, “robot_localization.” 2014, Accessed: Apr. 19, 2022. [Online]. Available: [http://docs.ros.org/en/melodic/api/robot_localization/html/index.html#:~:text=robot_localization is a collection of,estimation nodes%20 ekf_localization_node and ukf_localization_node .](http://docs.ros.org/en/melodic/api/robot_localization/html/index.html#:~:text=robot_localization%20is%20a%20collection%20of,estimation%20nodes%20ekf_localization_node%20and%20ukf_localization_node)
 - [44] T. Moore and D. Stouch, “A Generalized Extended Kalman Filter Implementation for the Robot Operating System,” 2014.

Appendix A: robot_localization Parameters

ekf_se_odom:

frequency: 10

sensor_timeout: 2

two_d_mode: true

transform_time_offset: 0.0

transform_timeout: 0.0

print_diagnostics: true

debug: false

publish_tf: true

map_frame: map

odom_frame: odom

base_link_frame: base_link

world_frame: odom

imu0: sensors/adImu

imu0_config: [false, false, false,

 false, false, false,

 false, false, false,

 false, false, true,

 false, false, false]

imu0_nodelay: false

imu0_differential: false

imu0_relative: false

imu0_queue_size: 10

imu0_remove_gravitational_acceleration: true

imu1: sensors/gpsImu

imu1_config: [false, false, false,
false, false, true,
false, false, false,
false, false, false,
false, false, false]

imu1_nodelay: false

imu1_differential: false

imu1_relative: false

imu1_queue_size: 10

imu1_remove_gravitational_acceleration: true

imu2: sensors/adImu

imu2_config: [false, false, false,
false, false, false,
false, false, false,
true, true, false]

imu2_nodelay: false

imu2_differential: false

imu2_relative: false

imu2_queue_size: 10

imu2_remove_gravitational_acceleration: true

use_control: false

process_noise_covariance: [1e-3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 1e-3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 1e-3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0.3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0.3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

```
0, 0, 0, 0, 0, 0.01, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0.5, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0.5, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0.1, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0.3, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.3, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.3, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.3, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.3, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.3]
```

```
initial_estimate_covariance: [10000, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
```

```
0, 10000, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 1e-9, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 1.0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 1.0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 4, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 1.0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 1.0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 1.0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 1.0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1.0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1.0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1.0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1.0]
```

```
ekf_se_map:
```

```
frequency: 10
```

sensor_timeout: 2
two_d_mode: true
transform_time_offset: 0.0
transform_timeout: 0.0
print_diagnostics: true
debug: false

map_frame: map
odom_frame: odom
base_link_frame: base_link
world_frame: map

odom0: odometry/gps_port
odom0_config: [true, true, false,
 false, false, false,
 false, false, false,
 false, false, false,
 false, false, false]
odom0_queue_size: 10
odom0_nodelay: true
odom0_differential: false
odom0_relative: false

odom1: odometry/gps_starboard
odom1_config: [true, true, false,
 false, false, false,
 false, false, false,
 false, false, false,
 false, false, false]
odom1_queue_size: 10

odom1_nodelay: true
odom1_differential: false
odom1_relative: false

imu0: sensors/adImu
imu0_config: [false, false, false,
 false, false, false,
 false, false, false,
 false, false, true,
 false, false, false]

imu0_nodelay: true
imu0_differential: false
imu0_relative: false
imu0_queue_size: 10
imu0_remove_gravitational_acceleration: true

imu1: nav/imu_gps
imu1_config: [false, false, false,
 false, false, true,
 false, false, false,
 false, false, false,
 false, false, false]

imu1_nodelay: false
imu1_differential: false
imu1_relative: false
imu1_queue_size: 10
imu1_remove_gravitational_acceleration: true

imu2: sensors/adImu
imu2_config: [false, false, false,
 false, false, false,

```

    false, false, false,
    false, false, false,
    true, true, false]
imu2_nodelay: false
imu2_differential: false
imu2_relative: false
imu2_queue_size: 10
imu2_remove_gravitational_acceleration: true

use_control: false

process_noise_covariance: [1.0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 1.0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 1e-3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0.3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0.3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0.01, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0.5, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0.5, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0, 0.1, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0.3, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.3, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.3, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.3, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.3, 0.3]

initial_estimate_covariance: [10000, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
    0, 10000, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 1e-9, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

```

```
0, 0, 0, 1.0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 1.0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 4, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 1.0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 1.0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 1.0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 1.0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1.0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1.0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1.0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1.0]
```

navsat_transform_port:

frequency: 10

delay: 0

magnetic_declination_radians: 0.1654572131 # 9.48 degrees E For lat/long 21.2969N,
157.8171W (uh manoa coordinates)

yaw_offset: 0 #1.570796327 # IMU reads 0 facing magnetic north, not east

zero_altitude: true

broadcast_utm_transform: true

publish_filtered_gps: true

use_odometry_yaw: false

datum: [21.311773, -157.889303, 0, map, base_link]

wait_for_datum: true

navsat_transform_starboard:

frequency: 10

delay: 0

magnetic_declination_radians: 0.1654572131 # 9.48 degrees E For lat/long 21.2969N,
157.8171W (uh manoa coordinates)

yaw_offset: 0 #1.570796327 # IMU reads 0 facing magnetic north, not east
zero_altitude: true
broadcast_utm_transform: true
publish_filtered_gps: false
use_odometry_yaw: false
datum: [21.311773, -157.889303, 0, map, base_link]
wait_for_datum: true