Final Project Report

Peter Kohler
Robotics Department
Colorado School of Mines
Golden, Colorado, USA
pmkohler@mines.edu

***Abstract*—This document is a report on the final project implemented by Peter Kohler for the class Estimation Theory and Kalman Filtering. The project involves the implementation of an Extended Kalman filter and an Unscented Kalman filter for the purposes of state estimation of a two wheeled robot in a simulated space. The implementation methodology is discussed, the results are presented, and conclusions are drawn about the efficacy of the implementation as well as comments are made on potential future additions to the project.**

***Index Terms*—State Estimation, Kalman Filtering, Robotics, Control Theory, Computer Vision**

I. Introduction

This report is a discussion as well as an opportunity to present the final project implemented by Peter Kohler for the class Estimation Theory and Kalman Filtering. An Extended Kalman filter and an Unscented Kalman filter were implemented for the purpose of state estimation of a differential drive robot. The state dynamic equations for this system are shown in equations 1, 2, and 3.

$$\dot{x} = cos(\phi)\frac{(v_l + v_r)}{2} \tag{1}$$

$$\dot{y} = sin(\phi)\frac{(v_l + v_r)}{2} \tag{2}$$

$$\dot{\phi} = \frac{(v_r - v_l)}{b} \tag{3}$$

Where the wheel separation b = 0.287 meters.

This system was simulated in a simulation framework popular in the Robotics community called Gazebo. Gazebo has an integrated version of the ODE physics engine, the OpenGL rendering engine, and has support for sensor simulation and actuator control. The OpenGL rendering engine allows for the system to be visualized in real time, an example of which can be seen for this system in Figure 1.

Gazebo is an open source project. This means that all of the source code is freely available online, and libraries and packages can be freely added to suit the users needs. Gazebo is, unless specified otherwise, a perfect simulation. In the context of the simulated differential drive robot, the values being reported from the wheel encoders, namely the wheel velocities, are exactly correct. For the purposes of the project and to better simulate a real world system, a library "plugin" was written and integrated into the author's local build of the Gazebo simulator to add uncertainty to

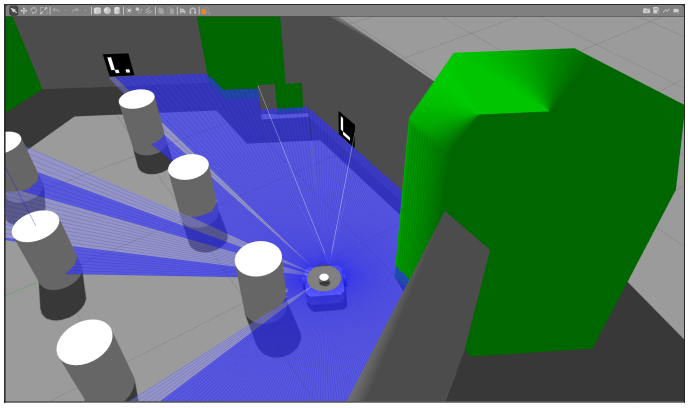

Fig. 1. Overhead view of Robot system in Gazebo

this simulated data. The library pulls the wheel encoder data from the simulation and via random number generation adds gaussian noise $\sim\mathcal{N}(0, 0.01)$ to the wheel velocity values being reported by the simulator. These wheel velocity values, with the added gaussian noise, serve as the input to the state dynamics equations used by both the Extended and Unscented Kalman filters. Using the known uncertainty of the wheel velocity inputs in conjunction with the defined state dynamics equations allows us to solve for the variance of each state and in turn populate the Q matrix used in the Kalman filter equations. This matrix was analytically determined to be

$$Q = \begin{bmatrix} 0.001 & 0 & 0 \\ 0 & 0.001 & 0 \\ 0 & 0 & 2.178E-5 \end{bmatrix}$$

and is used in both the Extended and Unscented Kalman Filter time update equations.

A goal from the outset of this project was to get both Kalman Filters running in real time alongside the active simulation. The author was interested in achieving an implementation that, as closely as possible, resembled something currently in use in industry to solve estimation problems such as robot localization. The Gazebo simulator provides a metapackage that allows for integration with a meta-operating system called ROS, which stands for the Robot Operating System. Among other things, ROS provides a very clean and easy to use interface that facilitates communication between different software processes. ROS is leveraged for this project to achieve fast data communication between the simulation

and the multiple different processes that are running which comprise the state estimation implementation for this system.

II. Methods

A. The Robot

The robot used for this simulation is called "Turtlebot3". Figure 2 depicts the Turtlebot3.

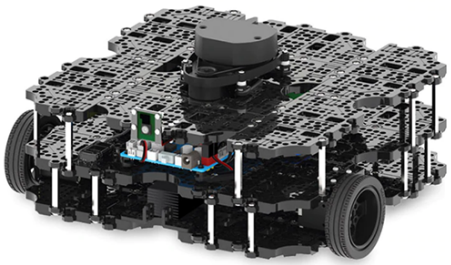

Fig. 2. The Turtlebot3 Robot, "Waffle Pi" variant

TurtleBot3 is a collaboration project among companies Open Robotics, ROBOTIS, and more. The physical version of this robot sells for approximately $1700. Fortunately, the simulated version of this robot in Gazebo is open source and available for free. This robot comes equipped with a omnidirectional LIDAR sensor, as well as a camera placed at the front of the robot. This camera is used in the process to take measurements from the simulated world that provide estimation corrections in the Kalman filtering process.

B. The Measurement

This implementation's measurement makes use of objects termed "fiducial markers" in the field of Computer Vision. Fiducial markers are objects that are easily and reliably detectable by computer vision detection algorithms in a digital image. A subset of these fiducial markers are used in this implementation called "ArUco" markers. Figure 3 shows an example of an ArUco marker.

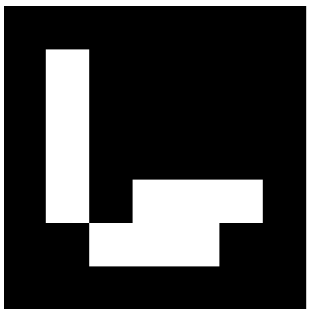

Fig. 3. An Example ArUco Marker (Dictionary: Original, ID: 7)

ArUco markers are assigned an ID number that corresponds to the unique pattern of the squares in the center of the object. ArUco markers that have the same ID number, and belong in the same dictionary subset, all have the same pattern in the center. In the simulation space, 6 ArUco markers are placed on the outer walls in a circle. Figure 4 depicts one of these simulated ArUco markers in the simulation space for this system.

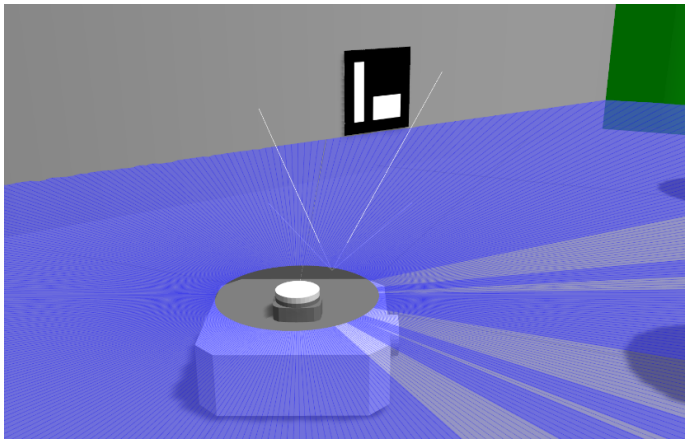

Fig. 4. A Simulated ArUco Marker in Gazebo

In this simulation, each of these placed ArUco markers have a unique ID and pattern. There are no duplicates. Each digital image generated from the simulated camera on the Turtlebot3 robot is run through an ArUco marker detection algorithm, making use of the camera's intrinsic parameters to account for simulated lens distortion. If an ArUco marker is detected, the pixel coordinates of the four corners of the detected marker are returned, as well as the ID number of the ArUco marker. After this detection algorithm has run, and if an ArUco marker is detected, a second computer vision algorithm is executed called Perspective-n-Point(PnP). The PnP algorithm provides an estimate of the information needed to construct a homogenous transform[1] from the center point of the camera's lens to the center point of the detected ArUco marker. Figure 5 depicts a visualization of a detected ArUco marker and subsequent pose estimation.

Since each of the placed ArUco markers in the simulation space are unique, and each of the markers positions and orientations were defined by the author in the simulation's configuration file, given the detected ArUco marker's ID number a second homogenous transform going from the simulation space's world frame origin to the detected ArUco marker can be easily constructed. Two properties of homogenous transforms are used here. First, the inverse of a homogeneous transform matrix defines a transform in the reverse direction. Second, the multiplication of homogenous transform matrices allows one to "chain" transforms together. The transform from the origin to the detected ArUco marker is multiplied with the inverse of the PnP algorithm's generated transform from the camera's lens to the ArUco marker. This in effect provides a direct transform from the world frame origin of

[1]If the reader is unfamiliar with homogenous transforms, they are 4x4 matrices that encode the translation and rotation of an object in 3 dimensional space with respect to a fixed coordinate system.

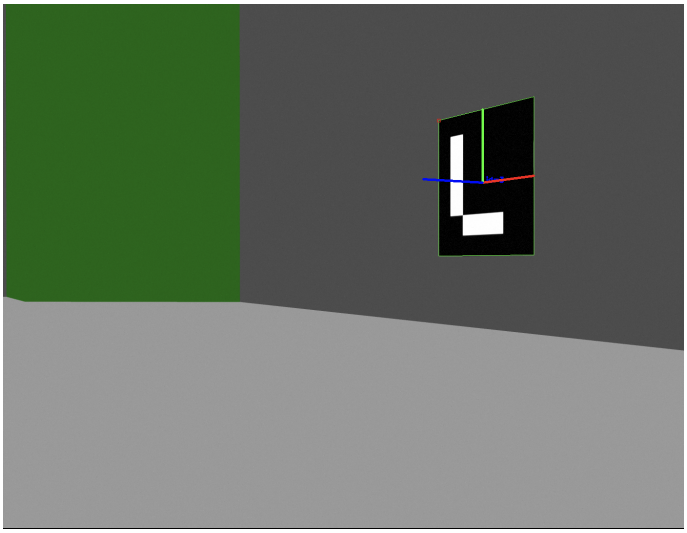

Fig. 5. A Detected ArUco Marker and Successful Pose Estimation

the simulation to the camera lens of the robot. After this, a static transform(given that the camera never moves relative to the robot) from the camera lens to the robot's base footprint is applied to get a transform that defines the position and orientation of the robot in the world frame of the simulation space. This process provides a very accurate observation of the robot's position and orientation in the world frame with a very small observed error. Unfortunately, this obviates the need for state estimation entirely. So in order to make use of Kalman filtering and state estimation techniques, only the X and Y coordinates of this calculated transform are used in conjunction with the known X and Y coordinates of the detected ArUco marker. The difference between these two coordinates is calculated and then fed into the four quadrant inverse tangent function to generate a single angle calculation dubbed "sigma". Equation 4 defines this function, and Figure 6 shows a diagram of this angle calculation.

$$g(x_k, u_k) = \sigma = atan2(y_k - y_A, x_k - x_A) \tag{4}$$

Where x_A and y_A are the known X and Y coordinates respectively of the detected ArUco marker.

This measurement contains a somewhat "undefined" uncertainty from the perspective of the author. Defined in the simulation parameters is a gaussian noise $\sim\mathcal{N}(0,\ 0.000049)$ that is applied to each of the pixel color channel values (these values range from 0 to 1) in the digital image which is used detect the ArUco marker. The author at the moment is unable to confidently define how this uncertainty propagates through the measurement process defined above and impacts the resulting sigma calculation. Put simply, the author is unsure of how to analytically calculate the value of R in the Kalman filtering equations. To this end, R was used as a tuning parameter and chosen by the value that provided the best performance.

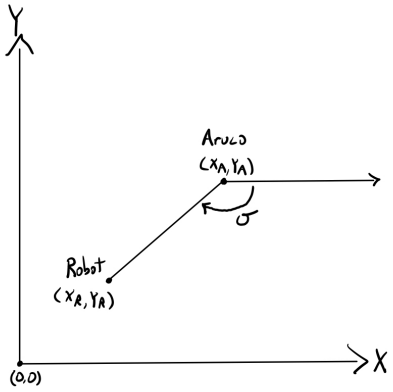

Fig. 6. An Example ArUco Marker (Dictionary: Original, ID: 7)

A caveat that should be mentioned is after this measurement process was implemented initial performance of both Kalman filters were extremely poor, with deviations occurring at seemingly random times during testing. After rather extensive debugging, the source of these deviations was later determined to be an instability in the pose calculation of the detected ArUco marker at distances around or larger than 3 meters. At this distance, the majority of the estimated poses via the PnP algorithm in a given span of time were accurate and in line with expectations. A small subset of them however, at seemingly random times, contained a very inaccurate pose estimation. This inaccuracy would provide a large difference in estimated location of the robot relative to the previous estimate and throw the Kalman filter's state estimate wildly off track. To correct for this, a simple fix was applied where measurements were only taken if the detected ArUco marker was estimated to have a euclidean distance from the camera lens of 2 meters or less.

One large flaw of this measurement methodology is that it does not incorporate the state ϕ in any way. This means that the measurement update process in the Kalman filter implementation will not provide any correction to the estimated state of ϕ, and the accuracy of ϕ is entirely reliant on the state dynamic equations. This in turn means that any error in the estimated initial condition of ϕ will persist for the lifetime of the Kalman Filter and it's state estimations.

C. The Extended Kalman Filter

The extended Kalman filter consists of a time update, and then a measurement update. Entering with $\hat{x}_k^{(+)}$ and $P_k^{(+)}$, our time update makes use of the gradient of the state dynamics equations defined in the matrix below:

$$\nabla_x f_c(x_k, u_k) = \begin{bmatrix} 0 & 0 & \frac{-sin(\phi_k)(v_l+v_r)}{2} \\ 0 & 0 & \frac{cos(\phi_k)(v_l+v_r)}{2} \\ 0 & 0 & 0 \end{bmatrix}$$

Since our state dynamics equations are continuous time functions, we use the euler approximations defined in equa-

tions 5 and 6 to perform our time update.

$$f(x_k, u_k) \approx x_k + f_c(x_k, u_k) * T_s \tag{5}$$

$$\nabla_x f(x_k, u_k) \approx I + \nabla_x f_c(x_k, u_k) * T_s \tag{6}$$

Both the extended and unscented Kalman filters are run at 30 Hz, so our sampling time T_s is 0.033 seconds.

The measurement update follows the standard update equations used for an extended Kalman filter. The gradient of the output function $g(x_k, u_k)$ defined in Equation 4 is

$$\nabla_x g(x_k, u_k) =$$

$$\begin{bmatrix} \frac{(y_A - y_k)}{(x_k - x_A)^2 + (y_k - y_A)^2} & \frac{(x_k - x_A)}{(x_k - x_A)^2 + (y_k - y_A)^2} & 0 \end{bmatrix}$$

Once a time and measurement update have occurred using the standard equations, new input data is pulled in from the real time simulator and, if available, a new measurement is taken using the process described above.

As mentioned previously, the uncertainty at the output dubbed "R" in the Kalman filter equations was used as a tuning parameter. For the extended Kalman filter, the value of this parameter was experimentally determined to be 15.

D. The Unscented Kalman Filter

The Unscented Kalman Filter is implemented using six sampling points which corresponds to double the number of states to be estimated by the filter. The square root factorization of the covariance matrix P is implemented by computing the Cholesky decomposition. The rest of the time update and measurement update is implemented using the equations $f(x_k, u_k)$ and $g(x_k, u_k)$ as defined in the previous sections and the standard equations for an unscented Kalman filter. This filter uses the same values of input uncertainty, or "Q" as the extended Kalman filter. The value of the uncertainty at the output termed "R" in the Kalman filter update equations was again used as a tuning parameter and chosen based on what value experimentally provided the best performance. This value through trial and error was determined to be 0.05.

III. Results

A primary metric used in this section to determine efficacy of an individual filter is mean squared error(MSE) with respect to the recorded ground truth value for each individual state. Both Kalman Filters were provided with an initial condition of

$$\begin{bmatrix} x_0 \\ y_0 \\ \phi_0 \end{bmatrix} = \begin{bmatrix} -2.2 \\ -0.55 \\ 0 \end{bmatrix}$$

This compares to the true initial condition of the robot which is $\{-2.0, -0.5, 0\}$ for x, y, and ϕ respectively. No error in the initial condition of ϕ_0 was provided since ϕ is not being updated by the measurement process, and so no correction is being applied to the estimate of ϕ_k. This means any error that is present in the initial condition at the start of the Kalman Filter will exist for the lifetime of the filter. This is, to be sure, a design flaw in the measurement process.

To provide a more confident assessment, the same trajectory was piloted in the simulation 5 times, with the estimated state from both the extended and unscented Kalman filters being recorded and the mean square error being calculated. Figure 7 shows the approximate trajectory taken by the robot in all 5 trials. The depicted trajectory was chosen with the goal to provide the Kalman filters with a relatively large number of measurements taken using the methodology described in section II.

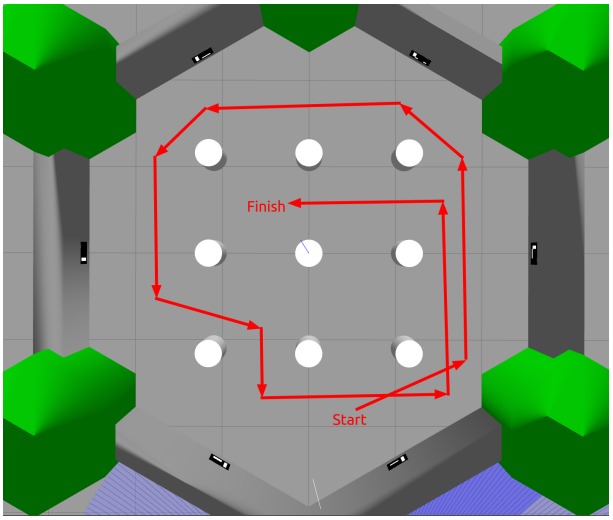

Fig. 7. Robot Trajectory for MSE Error Trials

The average of the MSE for all 5 trials is then calculated and shown below in table 1.

TABLE I
AVERAGE MSE COMPARISON OF EKF AND UKF

	Filter Implementation	
State	***EKF***	***UKF***
X	0.0396	0.0322
Y	0.0289	0.0258
Yaw	0.0380	0.0302

Figures 8, 9, and 10 depict MATLAB plots of the estimated states vs. the ground truth of one of these trials for the purposes of visualization. The "odom" values are the ground truth as dubbed by the Gazebo simulator.

The mean squared error values with respect to the ground truth coming from the simulator show that the unscented Kalman filter in this scenario provides a more accurate estimate of the true state of the robot system.

A metric of interest in the comparison of these two filters is to see how they handle inaccuracy of the initial condition. The aforementioned series of 5 trials run used an error of 10% in the estimated initial state of x_0 and y_0 but what happens if the initial estimate is off by a larger amount? How does this impact the efficacy of the two filters? Two trials were run, the first with an initial condition error of 50% in x_0 and y_0, and the second with an initial condition error of 100%. The MSE results are shown below in tables 2 and 3.

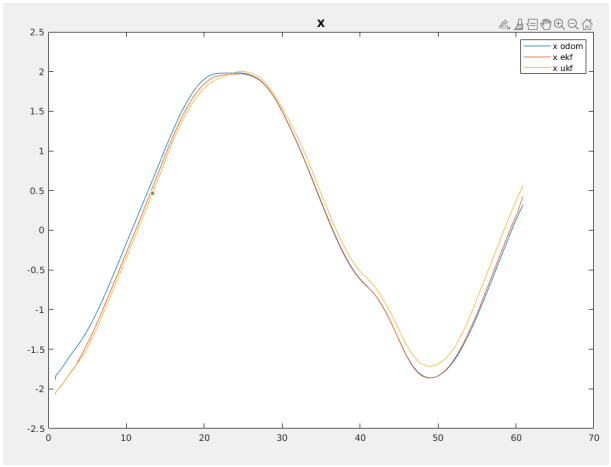

Fig. 8. Trial #5: x_k vs. Time

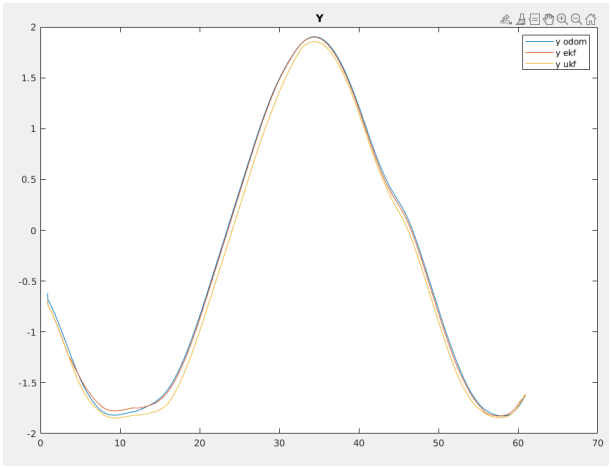

Fig. 9. Trial #5: y_k vs. Time

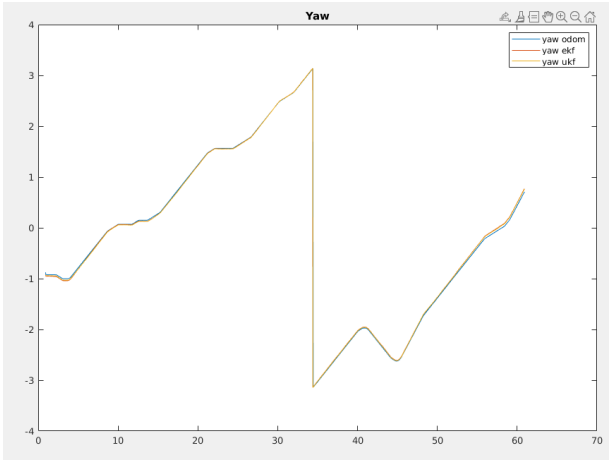

Fig. 10. Trial #5: ϕ_k vs. Time

TABLE II
50% INITIAL CONDITION ERROR

	Filter Implementation	
State	***EKF***	***UKF***
X	0.3385	1.0159
Y	0.0814	0.1193
Yaw	0.0817	0.0618

TABLE III
100% INITIAL CONDITION ERROR

	Filter Implementation	
State	***EKF***	***UKF***
X	1.6308	4.0002
Y	0.2427	0.3418
Yaw	0.0993	0.0499

These results show that as the error in the estimate of the initial condition increases with respect to the true initial condition of the system, the extended Kalman filter performs better than the unscented Kalman filter.

IV. CONCLUSIONS

This report details the methodology used to implement an extended and unscented Kalman filter both of which operate in real time alongside a simulated differential drive robot in the Gazebo simulation framework. The measurement or observation used for provide corrections to the estimated system state of both filters makes use of techniques and algorithms used commonly in the fields of robotics and computer vision. With an initial condition with a relatively high degree of accuracy, the unscented Kalman filter was through testing observed to have on average better performance than the extended Kalman filter. Should the initial state of the robot have a large amount of uncertainty to it the extended Kalman filter was demonstrated to perform better than the unscented counterpart. The degree to which the extended Kalman filter outperformed the unscented Kalman filter grew as the inaccuracy of the initial condition increased.

As mentioned in the explanation of the methodology used to implement the measurement for these two filters, one large flaw of the measurement is that it does not incorporate the orientation angle of the robot. The measurement update process in the Kalman filtering equations do not provide any correction to the filter's estimate of the state ϕ, and therefore any error that exists in the initial estimate of the robot's orientation on startup of the filter lasts throughout the entirety of the filter's lifetime. This creates a requirement that if any reasonable level of accurate state estimation is desired of this system, the initial estimate of the robot's orientation needs to have as little error as possible. If any work were to be added to this implementation, the first priority would be to redesign the measurement such that it incorporates the system state ϕ. This would then allow the measurement update process to provide a correction to ϕ, and in turn remove the constraint of a very high degree of accuracy in the estimate of the robot's

initial orientation. One possible solution would be to look for multiple ArUco markers in the detection process. The presence of multiple ArUco markers would allow for the "expected" orientation of the robot given multiple observed markers to be compared with the currently estimated orientation of the robot, and a correction could be provided to not only x_k and y_k, but also ϕ_k.

Another area of potential exploration of this implementation is the topic of optimization. Both Kalman filters benefit from running as fast as possible, such that method of numerial integration used (in this implementation, euler approximation) will generate as little error as possible with respect to the true system state. The simulator is set in the configuration file to run at a rate of 30 Hz, and so both Kalman filters were set to execute each respective time and measurement update at that same frequency. Testing could be done to see how fast both the simulator and the filters could run given the available computer hardware to try and produce the best possible representation of the true state of the simulated system.