LEHIGH UNIVERSITY
COMPUTER SCIENCE

Final Report December 9, 2023

Final Report on Forward Sensing Kalman Filter

by

Edward Jeffs

Abstract:

For my project, I want to create a Kalman estimator which can predict the state of non-holonomic drone with a forward facing distance sensor. I decide to use an Adaptive Extended Kalman Filter (AEKF). The assumptions I made for this assignment to simplify the model which may give problems in a real world robot are as follows; access to an accurate map, a smooth environment, constant wind across the environment, constant drift of the robot, velocity control instead of acceleration or jerk, a consistent distance sensor, and an accurate compass onboard. I will discuss the results of my simulations, and the challenges I faced in order to achieve these results. In the future works section I will go over some of the methods I will use to overcome the flaws in my assumptions in order to expand my work in the future.

Adaptive Extended Kalman Filter Equations

In this section, we present the fundamental equations that constitute the core of the Adaptive Extended Kalman Filter [1], a sophisticated algorithm used for the precise localization of mobile robots. The AEKF intelligently combines predictions based on a model of robot movement with sensor data, continuously adjusting its estimates to account for real-world uncertainties. The presented equations detail the mathematical framework underlying this process, including state prediction, uncertainty estimation, and the crucial adaptive mechanisms that enable the AEKF to dynamically respond to changing environmental conditions and sensor feedback.

1. State Prediction Equation

$$\hat{x}_{k|k-1} = f(\hat{x}_{k-1}, u_{k-1}) \tag{1}$$

Projects the state estimate $\hat{x}_{k-1}$, based on the current state and control input u_{k-1}, into the next time step using the state transition function f. f is usually a nonlinear function in robotics, accounting for motion dynamics. Changing f affects how the system predicts its future state based on current actions.

2. Covariance Prediction Equation

$$P_{k|k-1} = A_{k-1} P_{k-1} A_{k-1}^T + Q_{k-1} \tag{2}$$

Predicts the next state covariance $P_{k|k-1}$, incorporating the effect of process noise represented by Q_{k-1} and the Jacobian of f, A_{k-1}. Increasing Q_{k-1} can make the filter more responsive to changes, but may also introduce more noise.

3. Measurement Update Equation

$$\hat{x}_k = \hat{x}_{k|k-1} + K_k(z_k - h(\hat{x}_{k|k-1})) \tag{3}$$

Updates the state estimate $\hat{x}_k$ using the measurement z_k and the measurement function h, adjusted by the Kalman gain K_k. The function h maps the predicted state to the measurement space. It's crucial for interpreting sensor data. Modifying h can significantly affect how sensor readings are used to update the state.

4. Kalman Gain Equation

$$K_k = P_{k|k-1}H_k^T(H_kP_{k|k-1}H_k^T + R_k)^{-1} \tag{4}$$

Calculates the Kalman gain K_k, determining the trust level in the measurements relative to the prediction, with R_k representing measurement noise. The Jacobian of h, H_k tells us what the measurement is updating in the state.

5. Covariance Update Equation

$$P_k = (I - K_kH_k)P_{k|k-1} \tag{5}$$

Updates the state covariance P_k after incorporating the measurements, accounting for K_k and H_k. This step recalculates the state's uncertainty after the measurement update. The identity matrix I ensures that the update is scaled correctly. Changes here directly affect the filter's estimation of its own accuracy.

1 Implementation

My goal with my implementation was to enable the Kalman filter to accurately estimate the XY coordinates of the robot, given the assumptions of access to an accurate map, a smooth environment, constant wind across the environment, constant drift of the robot, velocity control instead of acceleration or jerk, a consistent distance sensor, and an accurate compass onboard. I am able to predict the x and y coords and vx and vz of the wind affect on the robot with the distance sensor, and the orientation of the robot with the compass using the Kalman filter. The biggest feature (and challenge) of my implementation is the prediction of the constant drift of the robot relative to the heading of the robot by using both sensors together. As the direction in XY of this drift changed with the orientation of the robot, the normal matrices would need some rotation component for both measurement and movement in order to compensate.

1.1 Setup

The environment as shown in Figure 1 is the simple environment that was tested during the construction of the Kalman filter. The environment is a rectangular room with four smooth walls. The true robot would have a simple motion control on board that was left unchanged while I created the filter. It would simply turn

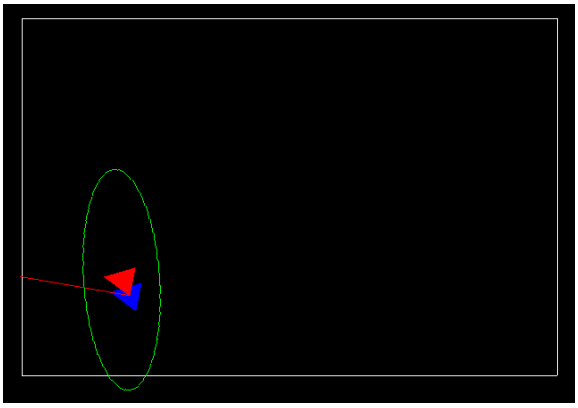


Figure 1: Robot in the environment surrounded by walls denoted by the white rectangle. The red triangle is the estimated state, the blue triangle is the true state, the red line coming from the true state is the distance sensor, and the green ellipse is the covariance ellipse of the Kalman filter.

to the right whenever it saw a wall at a certain distance for a random amount of time then move forward at a constant control velocity. If the robot runs into a wall it can not pass it, although it may become stuck. In the future I may want to make the environment more complex, or represented with more uncertainties. The state of my robot has 8 variables; $[x, y, \theta, v_{x_{wind}}, v_{y_{wind}}, v_\theta, v_{x_{drift}}, v_{y_{drift}}]$. There is a constant wind in the environment which affects the movement of the robot at a constant velocity. The robot does not need to decelerate or accelerate, but it will drift according to its internal bias in drift, and the external wind factor.

1.2 Predict Step

The predict step's primary goal is to evolve the state of the robot using Equation 1 and update the covariance using equation 2. Therefore we need to define $f(\hat{x}_k, u_k)$, A, P, and Q. I implemented $f(\hat{x}_k, u_k)$ with

$$\hat{x}_{k|k-1} = A_{k-1}\hat{x}_{k-1} + B_{k-1}u_{k-1} \tag{6}$$

where B_{k-1} affects how the control input u_{k-1} affects the state.

The A-matrix is the Jacobian of f and the model I use is:

$$A = \begin{bmatrix} 1 & 0 & 0 & dt & 0 & 0 & cos(\theta) & -sin(\theta) \\ 0 & 1 & 0 & 0 & dt & 0 & sin(\theta) & cos(\theta) \\ 0 & 0 & 1 & 0 & 0 & dt & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{7}$$

The diagonal being 1 means that the system does not grow or decay over time. The dt's in the first three rows are the time step, as the position in x, y and θ evolve with the velocity of the drone. The top right of my A matrix is the effect of the drift of the drone on the x and y coordinates. It is a rotation matrix on θ. Since θ changes dynamically, I need to update my A matrix on every time step.

The B-matrix has 3 inputs, the x velocity, y velocity, and angular velocity. These inputs are processed by the state using the θ in order to generate the correct x and y velocity which depends on the direction the robot is facing.

$$A = \begin{bmatrix} dt & 0 & 0 \\ 0 & dt & 0 \\ 0 & 0 & dt \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{8}$$

the dt's add the velocity of the drone into the state x.

The P matrix stores the error covariance, which I initialize as an 8x8 identity matrix. The Q matrix determines how fast my uncertainty expands, and some of my state variables needed different values as they become uncertain faster. I also updated the top right corner of my Q matrix with a rotation matrix biased in the direction the robot was moving in order to represent how the robot has more uncertainty in the forward direction.

These matrices placed into Equation 1 and 2 are what makes up the core of my predict step.

1.3 Update Step

The update step's primary goal is to update the state and uncertainty based on measurements using equations 3, 4, and 5. The measurement update equation 3 uses the measurement to directly update the state by the Kalman gain. The Kalman gain equation updates the trust level of the measurements through H and R.

Specifically for my application my measurements are made using a compass and a time of flight (tof) distance sensor with limited range and slight Gaussian uncertainty in distance. When a wall is sensed by the tof sensor, I use the estimated state to predict which wall the tof sensor is facing, then use the tof sensor and estimated θ to find normal distance to that wall. Since I know where that wall is in the environment I can update my x, y state as I know that I probably exist within a certain distance of that wall in the environment. This causes the uncertainty to flatten to the parallel of the wall. The compass measurements directly measure the θ of the robot with a slight Gaussian uncertainty.

2 Challenges

The initial challenge with this project was getting a working understanding of how Kalman filters work, and creating a simulation which could demonstrate my understanding and control over the details of the algorithm. I wanted to achieve a robust system which could be transferred to real world robots once fully developed, so I started out with a full state robot which was controlled by accelerations. However, I found that I did not understand the Kalman filter well enough to understand my coding errors, nor did I want to use my time to tune a PID controller for my accelerations in a simulation. Therefore for this project I decided to directly control velocities. The second major challenge was using wall measurements instead of direct x,y coordinates to estimate state in the Kalman filter. I eventually found that the wall measurements use a one dimensional input along with a rotation of the wall in the H to measure the position. I had not needed to use this rotation of the wall relative to space, as I simplified the problem by making each wall parallel to the x or y axis. The final major challenge was understanding how to integrate the

natural robot drift relative to the local frame into the dynamics of the robot. I spent a great deal of time on this issue as I had a fundamental bug (degrees vs radians) in my rotation matrix calculations, which caused me to misunderstand why it was not working.

3 Results

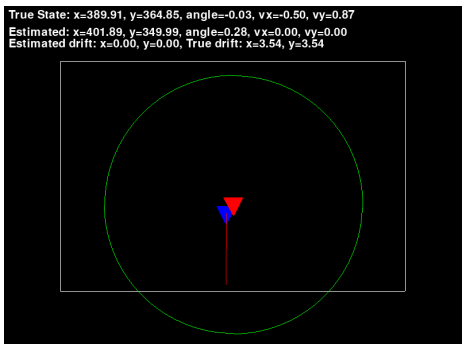
From my simulation in pygame, I found I had made a robust Kalman filter that gives a good estimate of the state. Figure 2 shows a few frames from one such simulation. The covariance ellipse starts extremely large, but collapses to surround just the robot. The wind estimation vx and vy and robot drift very closely approach the true values. When these values are excluded from the model, the robot has a difficult time tracking the true state when a measurement is not being taken, as the uncontrolled movements are not compensated for.

Figure 3 shows the plots of the positional error in pixels over time, and the uncertainty of the covariance matrix over time. Each drop in the graph shows when a wall measurement is taken, and it seems to take 6 or 7 wall bounces to converge. Comparing it to the video, and it reaches convergence once it observes a full circuit in the environment. While there is still some small amount of error in the Kalman estimated state, this is a result of the random variables in the environment from the distance measurement, the angle measurement, and the robot motion in velocity and angle.

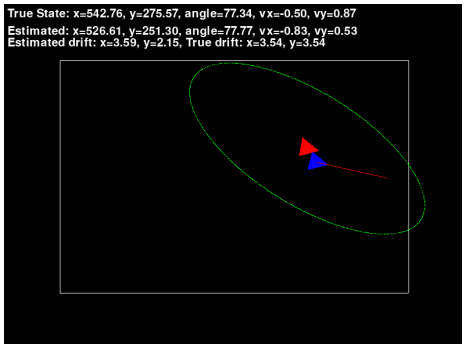
4 Future Work

The Kalman also required that the estimated state started very close to the true state, as if the estimated robot state leaves the walls, its internal map breaks. If I wanted to create a more complex environment, this problem would need to be resolved. I might need to make the robot better able to predict its own state, maybe by attempting to add MDP to the Kalman state for direct coordinates. I might also want to dive into uncertainties in wind velocity, like turbulence, however I do not think that the Kalman model would be able to handle such uncertainties.

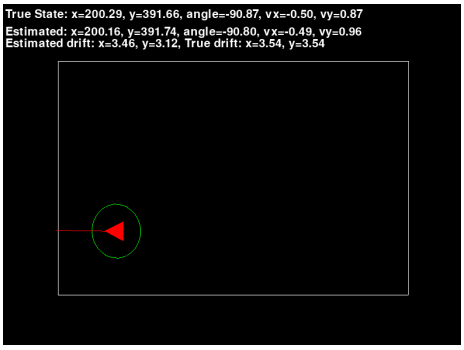
I would also want to create or use a more accurate simulator, like CoppeliaSim. This would allow me to reach a better step towards creating a model which would work on real world robots. I would attempt to collect data to find the maximum



(a) First Subfigure



(b) Second Subfigure



(c) Third Subfigure

Figure 2: These images show 3 figure of the robot Kalman filter over time. It lists the true state variables, and the predicted variables of the Kalman filter. The covariance ellipse shrinks over time with each measurement, the Kalman state accurately estimates the true state.

uncertainty allowed by the sensors and actuators which still allows the Kalman filter to converge.

Another part of the state I wanted to add to this system is an estimation of the control velocity, instead of directly using the desired velocity, of the agent using a Kalman filter. Given I was able to achieve control drift, I think that it would be a feasible project. However, I know there are other methods of model predictive control which can estimate this, I think it would be interesting to allow the robot to adjust its predictions online, for example if the robot runs slower on lower battery.

The compass might not always be reliable, and have some constant offset, so I could create an estimation of this offset using the Kalman filter. This would

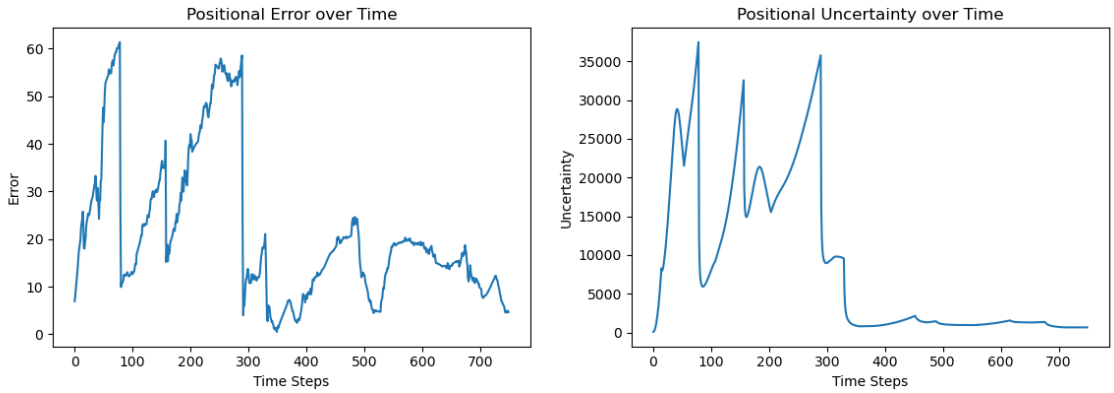


Figure 3: The first graph shows the positional error of the Kalman estimation in pixels, and the other shows the uncertainty of the covariance matrix in arbitrary units. It converges after 300 time steps.

be achievable with my current model, but may require some clever handling of the new state variable with more rotation matrices. In fact, a model whose only purpose is to estimate the compass offset using the wall measurements could be extremely valuable for our real world robots. I found recently that the initial offset of our IMUs can initialize at a random angle in some circumstances.

I would also need to see how efficiently I could run a full Kalman filter on a Arduino, to make sure that it would run online in our robots without slowing down the main control loop.

5 Conclusion

Through this work, I am now more confident in my Kalman filter skills, and may use this tool for Gaussian measurements, as in the past I would use a simple weighted average or complementary filter to remove noise in my experiments. I am confident that I would be able to use a Kalman filter for my robot applications in the future. One task I might try to do in the future would be to create a Kalman filter to estimate the position of an object relative to a camera attached to a blimp. The rocking motion and positional movements of the camera could make the problem solvable, as in the past I tuned weighted averages to solve these problems.

6 References

[1] L. Jetto, S. Longhi, and G. Venturini, “Development and experimental validation of an adaptive extended Kalman filter for the localization of mobile robots,” IEEE Trans. Robot. Autom., vol. 15, no. 2, pp. 219–229, Apr. 1999. This paper serves as the core of my model