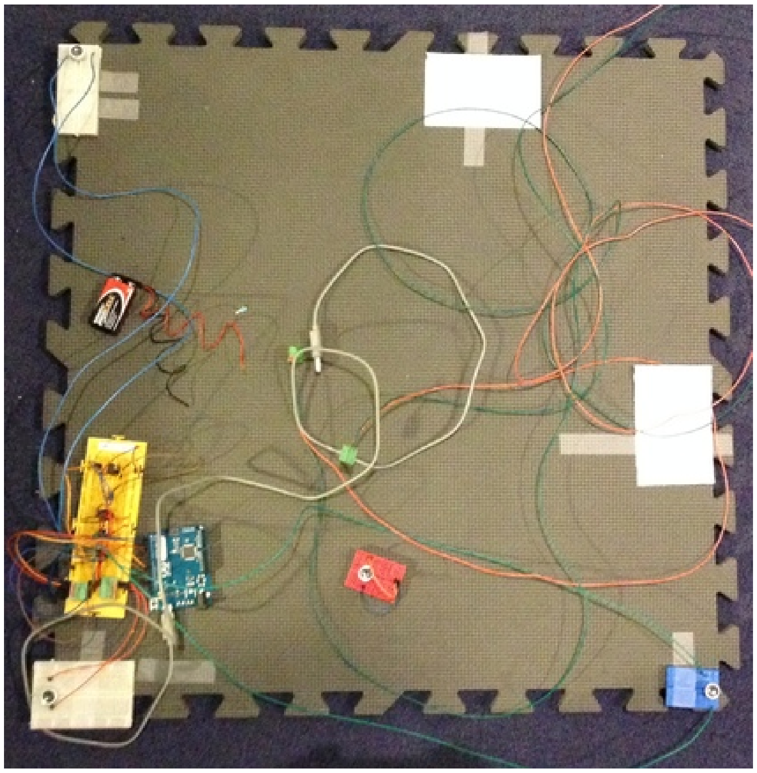
Ultrasound Positioning System using the Kalman Filter

16.322 Stochastic Estimation and Control Final Project

Massachusetts Institute of Technology



Enrique Fernandez

[redacted]@mit.edu

December 8, 2013

Contents

1 Introduction

In my everyday research I normally don't encounter estimation problems. However, as I think filtering is an essential tool for an engineer, and I wasn't familiar with it prior to taking this class, I decided that I would build a physical system and design a Kalman Filter for it.

I have always been fascinated by how GPS localization works and how, by using time-of-flight measurements, receivers can estimate position with very high accuracy. Because of that, I decided that I would try to build my own indoor localization system using ultrasound trilateration. I could have simulated the system for this project, but I thought it would be much more interesting to build the physical system for several reasons. First, in class we've only dealt with simulated systems and I wanted to experience how Kalman filtering performs in real systems with non-linear equations and in which not all of our common assumptions hold. Also, I was very interested in seeing whether ultrasound positioning would actually work fine, and for that I needed to build the real system. Finally, I wanted to take this opportunity to gain experience in building custom designed hardware and data capture software, which I normally don't do in my everyday research, and was eager to learn.

That given said, this project aims to demonstrate how Kalman filtering can help us obtain accurate position estimates using ultrasound trilateration. The project structure is as follows. First, I describe the physical setup that I built for the project and the equations needed to calculate the position from the sensor measurements. I continue describing the filters used in this project (EKF and UKF). Finally, I present my results and conclusions.

2 Position Estimation through Ultrasound Trilateration

The main idea behind ultrasound trilateration is that we can calculate the position of the intended object relative to known locations by measuring the distance to those locations. In this project, I chose to use three known locations named *Sensor 1 (S1)*, *Sensor 2 (S2)* and *Sensor 3 (S3)*. The three sensors are ultrasound receivers, while the object to be tracked is an ultrasound emitter. Given that speed of sound is constant for a given temperature, we can calculate the distance from the object to the sensors by measuring the time it takes for each of the sensors to detect an ultrasound pulse sent by the emitter.

The distance from the object to each sensor can be calculated as :

$$d_i(\mathrm{m}) = t_i(\mathrm{\mu s}) \cdot speed_{sound} \tag{2.1}$$

$$speed_{sound} = 340.29 \times 10^{-6}\,\mathrm{m/\mu s} \tag{2.2}$$

Given that the sensors are fixed in know locations and placed in the $z = 0$ plane according to figure 1, the measured distances are related to the object's position in the following way

$$d_1 = \sqrt{x^2 + y^2 + z^2} \tag{2.3}$$

$$d_2 = \sqrt{(x - l_2)^2 + y^2 + z^2} \tag{2.4}$$

$$d_3 = \sqrt{x^2 + (y - l_3)^2 + z^2} \tag{2.5}$$

with $l_2 = 0.567\,\mathrm{m}$ and $l_3 = 0.560\,\mathrm{m}$.

Under the assumption that $z \geq 0$, we can solve the previous system of equations to find x ,y and z as:

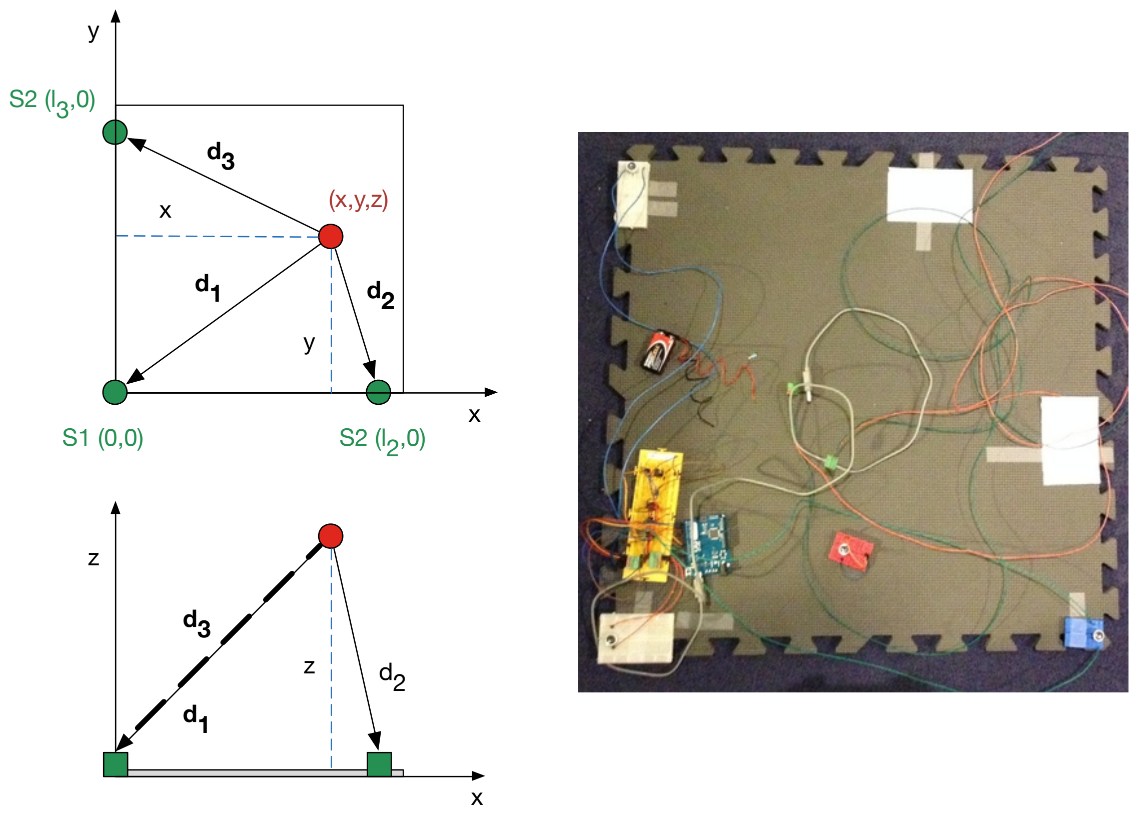


Figure 1: Diagram and photo of localization area and sensors.

$$x = \frac{d_1^2 - d_2^2 + l_2^2}{2l_2} \tag{2.6}$$

$$y = \frac{d_1^2 - d_3^2 + l_3^2}{2l_3} \tag{2.7}$$

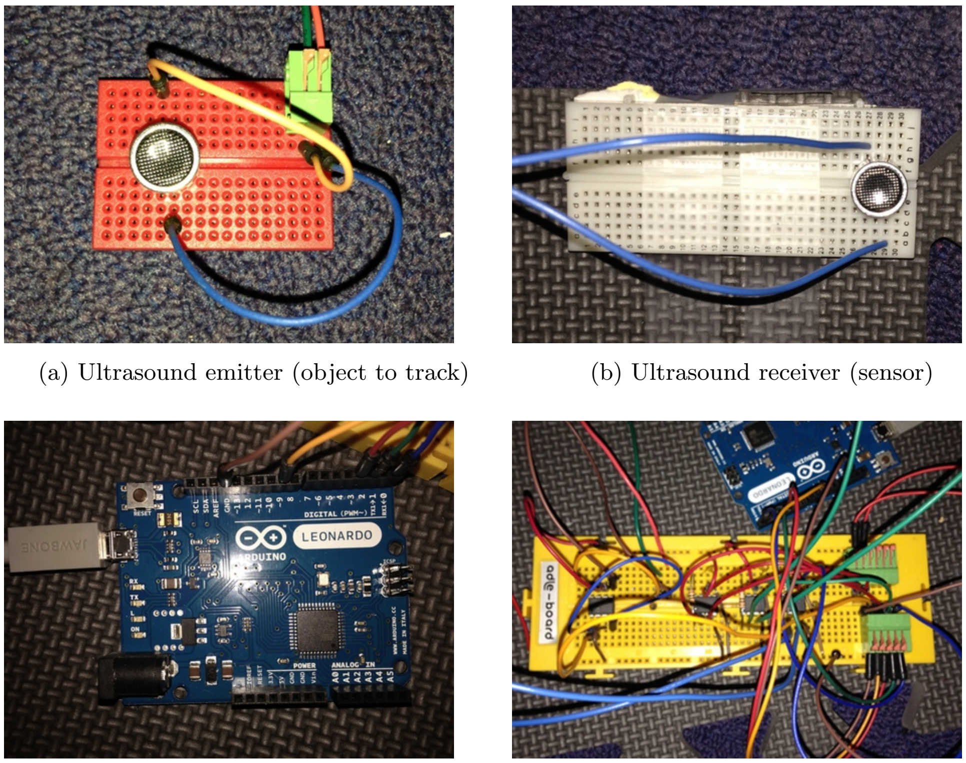
$$z = \Re\left(\sqrt{d_1^2 - x^2 - y^2)}\right) \tag{2.8}$$

We can use equations (2.6)–(2.8) to directly estimate the position of the object using the sensor measurements. However, because the measurements are noisy, the estimate will be poor. In the next sections of this project, I will explain how to design non-linear Kalman filters to improve this simple estimate.

3 Hardware Setup

As described in section 2, the purpose of the hardware is to measure the time it takes each of the sensors to detect the ultrasound pulse originated at the object we want to track.

The electronics design for the ultrasonic receivers and transmitter is based on the design showcased in a final project of a Cornell microcontrollers course [1]. The ultrasound receivers and the emitter resonate at 40 kHz. The whole system is controlled by an *Arduino Leonardo* platform (figure 2c), that I programmed from scratch. Every 50 ms, the Arduino generates a 40 kHz square signal that triggers a MOSFET driver that is connected to the ultrasound emitter (the object we want to track, figure 2a). At that moment, the microcontroller sets the timer to zero and starts counting. The generated ultrasound wave travels at



(a) Ultrasound emitter (object to track)

(b) Ultrasound receiver (sensor)

(c) Arduino (microcontroller)

(d) Sensor board

Figure 2: Hardware setup

the speed of sound until it reaches the three sensors (figure 2b). On the receiving end, the output of each of the ultrasound receivers is amplified with an op-amp and then connected to one of the interrupt pins of the Arduino. When the traveling wave reaches one of these sensors, the associated interrupt code is triggered and the Arduino stores the value of the timer at that point. When all three sensors are triggered, the microcontroller sends a message containing the measured time values for each sensor to a MATLAB program running in parallel that stores the data. The previously described system stores samples at 20 Hz. Although this sampling rate can be adjusted, I found it to be a good choice for this problem.

4 Filter Design

In this project, I have developed three position estimators, and I have compared their performance. The following subsections describe how those estimators were designed and how the sensor data is processed from the time is acquired until the estimate is calculated.

4.1 System Modeling

The objective of this project is to track the position of an object using the measured distances to three sensors whose positions are known. A priori, we don't know anything about the dynamics of the object to track: it could move in any way. Because of that, I chose to represent the dynamics of the system with a simple linear model consisting of six states: x, y, z and their velocities.

The model can be expressed with the following equation

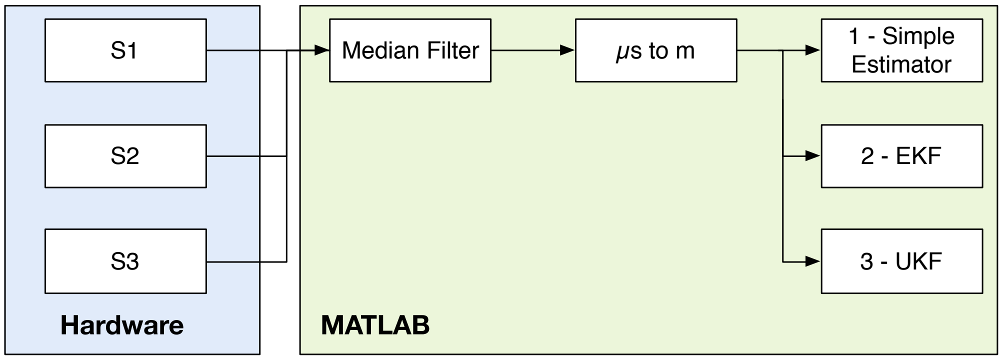


Figure 3: Diagram of filters

$$\begin{pmatrix}\dot{x}\\ \dot{y}\\ \dot{z}\\ \ddot{x}\\ \ddot{y}\\ \ddot{z}\end{pmatrix} = \begin{pmatrix}0&0&0&1&0&0\\0&0&0&0&1&0\\0&0&0&0&0&1\\0&0&0&0&0&0\\0&0&0&0&0&0\\0&0&0&0&0&0\end{pmatrix}\begin{pmatrix}x\\ y\\ z\\ \dot{x}\\ \dot{y}\\ \dot{z}\end{pmatrix} + \begin{pmatrix}0&0&0\\0&0&0\\0&0&0\\1&0&0\\0&1&0\\0&0&1\end{pmatrix} w \tag{4.1}$$

, being w white Gaussian noise with intensity W. Note that we consider the *process noise*, w, the unknown factor that drives the change in the system. The effect of the agent that changes the position of the object is then modeled as white noise acting on the acceleration of the system.

In this project, I implemented the filters in discrete time. Given that the *sampling rate* of the system is 20 Hz, we can use the *conversion algorithm* to find A_d as:

$$A_d = \begin{pmatrix}1&0&0&0.5&0&0\\0&1&0&0&0.5&0\\0&0&1&0&0&0.5\\0&0&0&1&0&0\\0&0&0&0&1&0\\0&0&0&0&0&1\end{pmatrix} \tag{4.2}$$

The process noise and its intensity (W and W_d) are discussed later in section 4.6.

The measurement equation, as discussed in section 2 is **non-linear**, and can be expressed as:

$$\mathbf{h}(\bar{x}) = \begin{pmatrix}\sqrt{x^2+y^2+z^2}\\ \sqrt{(x-l_2)^2+y^2+z^2}\\ \sqrt{x^2+(y-l_3)^2+z^2}\end{pmatrix} \tag{4.3}$$

4.2 Simple Estimator

This estimator is the simplest of all three, as it only estimates the position based on the current measurement by solving the equations (2.3)–(2.4). The estimator is calculated as described in section 2 and its equations are reproduced here again.

$$x = \frac{d_1^2 - d_2^2 + l_2^2}{2l_2} \tag{4.4}$$

$$y = \frac{d_1^2 - d_3^2 + l_3^2}{2l_3} \tag{4.5}$$

$$z = \Re\left(\sqrt{d_1^2 - x^2 - y^2}\right)) \tag{4.6}$$

Note that this estimator doesn't consider the history of the object, or any statistics about the sensor or process noise of the system and, therefore, we can expect it to perform poorly.

4.3 Extended Kalman Filter (EKF)

The Extended Kalman Filter (EKF) was implemented in discrete time considering that the state equation is linear and that we can linearize the measurement as:

$$\frac{\partial \mathbf{h}}{\partial \mathbf{x}}(\mathbf{x}) = \begin{pmatrix} \frac{x}{\sqrt{x^2+y^2+z^2}} & \frac{y}{\sqrt{x^2+y^2+z^2}} & \frac{z}{\sqrt{x^2+y^2+z^2}} & 0 & 0 & 0 \\ \frac{x-l_2}{\sqrt{(x-l_2)^2+y^2+z^2}} & \frac{y}{\sqrt{(x-l_2)^2+y^2+z^2}} & \frac{z}{\sqrt{(x-l_2)^2+y^2+z^2}} & 0 & 0 & 0 \\ \frac{x}{\sqrt{x^2+(y-l_3)^2+z^2}} & \frac{y-l_3}{\sqrt{x^2+(y-l_3)^2+z^2}} & \frac{z}{\sqrt{x^2+(y-l_3)^2+z^2}} & 0 & 0 & 0 \end{pmatrix} \tag{4.7}$$

$$C_k(\hat{\mathbf{x}}_{k|k-1}) = \left.\frac{\partial \mathbf{h}}{\partial \mathbf{x}}\right|_{\mathbf{x}=\hat{\mathbf{x}}_{k|k-1}} \tag{4.8}$$

The EKF update equations are then:

$$\hat{\mathbf{x}}_{k|k-1} = Ad \cdot \hat{\mathbf{x}}_{k-1|k-1} \tag{4.9}$$

$$\mathbf{Q}_{k|k-1} = A_d \mathbf{Q}_{k-1|k-1} A_d^T + W_d \tag{4.10}$$

$$\mathbf{Q}_{k|k} = \mathbf{Q}_{k|k-1} - \mathbf{Q}_{k|k-1} C_k^T (C_k \mathbf{Q}_{k|k-1} C_k^T + R)^{-1} C_k \mathbf{Q}_{k|k-1} \tag{4.11}$$

$$L_k = \mathbf{Q}_{k|k} C_k^T R^{-1} \tag{4.12}$$

$$\hat{\mathbf{x}}_{k|k} = \hat{\mathbf{x}}_{k|k-1} + L_k(\mathbf{y_k} - \mathbf{h}(\hat{\mathbf{x}}_{k|k-1})) \tag{4.13}$$

4.4 Unscented Kalman Filter (UKF)

As in the EKF case, the Unscented Kalman Filter was implemented considering that the state dynamics are linear, and the measurement equations are non-linear.

The mean and covariance resulting from the linear state propagation can then be expressed by equations (4.9) and (4.10). The non-linear measurement update is calculated using the unscented transformation according to:

$$Q_{yy} = \frac{1}{2n}\sum_{i=1}^{2n}(\hat{\mathbf{y}}_k^{(i)} - \hat{\mathbf{y}}_k)(\hat{\mathbf{y}}_k^{(i)} - \hat{\mathbf{y}}_k)^T + R_k \tag{4.14}$$

$$Q_{xy} = \frac{1}{2n}\sum_{i=1}^{2n}(\hat{\mathbf{x}}_k^{(i)} - \hat{\mathbf{x}}_{k|k-1})(\hat{\mathbf{y}}_k^{(i)} - \hat{\mathbf{y}}_k)^T \tag{4.15}$$

$$L_k = Q_{xy}Q_{yy}^{-1} \tag{4.16}$$

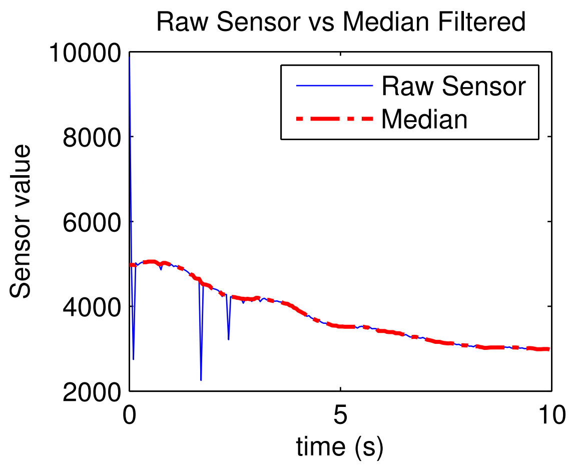
$$\hat{\mathbf{x}}_{k|k} = \hat{\mathbf{x}}_{k|k-1} + L_k(\mathbf{y}_k - \hat{\mathbf{y}}_k) \tag{4.17}$$

$$Q_{k|k} = Q_{k_k-1} - L_kQ_{yy}L_k^T \tag{4.18}$$

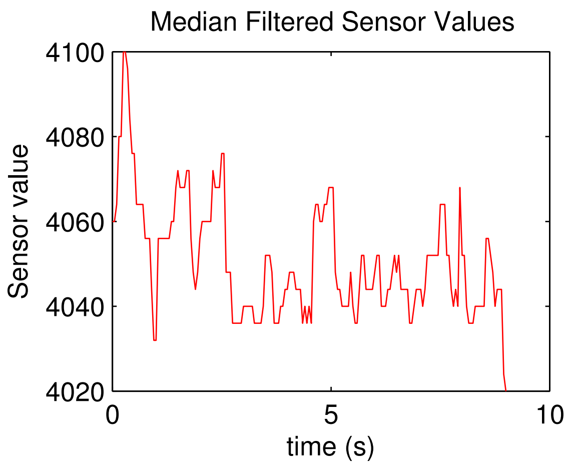
being $\hat{\mathbf{x}}_k^{(i)}$ the *sigma points* around $\hat{\mathbf{x}}_{k|k-1}$ and $\hat{\mathbf{y}}_k^{(i)} = \mathbf{h}(\hat{\mathbf{x}}_k^{(i)})$ those transformed sigma points through the non-linear measurement equations. I chose to use the *Wan and Van der Merwe* sigma points with parameters $\alpha = 1 \times 10^{-3}$, $\kappa = 3 - n = -3$ and $\beta = 2$.

4.5 Signal preprocessing: rejecting outliers

The blue plot in figure 4a shows the measured times by one of the sensors in one of the recorded runs. As the plot shows, the sensor data often includes large spikes or outliers. These outliers don't follow a statistical distribution that we can model, and are hard to deal with in our standard filter formulations. Because of that, I decided to pass the sensor readings through a median filter using MATLAB's *medfilt1* function. This function computes a moving window median filter of 3 elements, selecting the median. The red plot in figure 4a shows how the median filter effectively eliminates most of the spikes or outliers in the sensor values. The plot in figure 4b shows a sample of the sensor values recorded in a case in which the tracked object was in a relatively fixed position, and therefore gives us a rough idea on what the sensor noise looks like.



(a) Raw sensor values vs median filtered



(b) Stationary median filtered sensor output

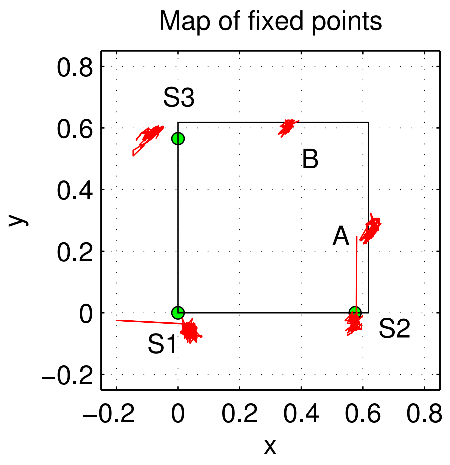
Figure 4: Plots of sensor data before and after applying a median filter.

All three estimators use the median filtered data in order to ensure that most of the outliers have been eliminated.

4.6 Sensor and process noise

Apart from the equations discussed in sections 4.3 and 4.4, we still need to choose two important design parameters: the sensor noise $\mathbf{R}$, and the process noise intensity $\mathbf{W}$.

In order to estimate the sensor noise, I performed a simple test consisting in positioning the sensor in several fixed locations and recording the sensor values. As discussed in section 4.5, the sensor values analyzed were first passed through a median filter in order to eliminate the outliers due to ultrasound echoes and bad electrical connections.



(a) The red plot shows the five locations at which I took the noise measurements (calculated with the simple direct equations (4.4)–(4.6))

Location	Sensor 1 noise (m)	Sensor 2 noise (m)	Sensor 3 noise (m)
A	0.0052	0.0037	0.0069
B	0.0022	0.0024	0.0031
S2	0.0045	0.0049	0.0068
S3	0.0061	0.0046	0.0023
S1	0.0031	0.0058	0.0046
Average (std)	**0.0042**	**0.0043**	**0.0047**

(b) This table shows the standard deviations of the measurements of each sensor at each location and the average of those standard deviations across all fixed locations.

Figure 5: Sensor noise at different fixed locations.

Figure 5a shows approximately the five different locations at which I took the noise measurements. Figure 5b shows the standard deviation of those measurements for each sensor at each location. It is important to consider that, due to the electrical setup that I had, it was very hard to keep the object still at a fixed position, as I had to do it manually. Therefore, these noise measurements could be different than the real values due to the movement of my hand while holding the object. The final sensor noise intensity that I used is:

$$\mathbf{R} = \begin{pmatrix} 0.1764 \times 10^{-4} & 0 & 0 \\ 0 & 0.1849 \times 10^{-4} & 0 \\ 0 & 0 & 0.2209 \times 10^{-4} \end{pmatrix} \mathrm{m}^2 \tag{4.19}$$

Choosing an appropriate value for the process noise was much harder, since this noise is fictitious and its purpose is to represent the change in acceleration due to the object moving along an a priori unknown trajectory. After testing with multiple values, I decided to use $w = 0.05\,\mathrm{m/s}^2$. Discretizing $\mathbf{W}$ as in section 4.1, I obtained the following final process noise intensity:

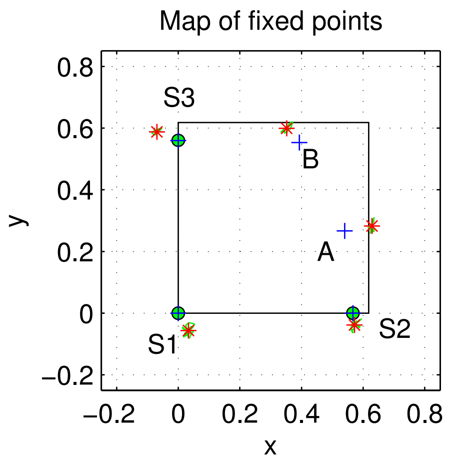
$$\mathbf{W_d} = \begin{pmatrix} 0 & 0 & 0 & 0.0001 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.0001 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.0001 \\ 0.0001 & 0 & 0 & 0.0025 & 0 & 0 \\ 0 & 0.0001 & 0 & 0 & 0.0025 & 0 \\ 0 & 0 & 0.0001 & 0 & 0 & 0.0025 \end{pmatrix} \tag{4.20}$$

5 Results

In order to assess the performance of the filters, I ran several tests consisting on leaving the object in a fixed location and on moving the object along a trajectory.

5.1 Static position error

The first thing I wanted to test is the accuracy of this ultrasound positioning system. To that end, I marked and measured several points in the localization region (x and y coordinates only) and I took sensor measurements while leaving the object still on top of those markers. I then run the UKF filter on those measurements in order to get an estimate of the x and y position. The results of this test are summarized in figure 6. In figure 6a, the blue markers represent the 'known' positions in the map. The green plots show the output of the UKF filter (without the initial transient region). Finally, the red star markers sow the average estimate of the UKF filter. The accuracy errors, displayed in figure 6b, show that the system has an average error of 6.5 cm. Although this error seems very big, there are a couple of important factors to be considered. First, I manually position the object on top of the markers and tried to leave it still as best as I could. As a result, I may not have been able to place the object exactly on the designated position and the object wasn't exactly still (my hands shook a little). In any case, it is clear that there is some bias in the measurements.



(a) Map of real points and estimated positions. This test only focuses on the x and y positions, and not on the height (z).

Loc	Real x	Real y	Est x	Est y	**Distance error (m)**
A	0.540	0.267	0.629	0.282	0.090
B	0.393	0.553	0.352	0.599	0.062
S2	0.567	-0.000	0.572	-0.038	0.039
S3	-0.000	0.560	-0.070	0.588	0.075
S1	-0.000	-0.000	0.033	-0.056	0.065
				Average	**0.065**

(b) Accuracy Error

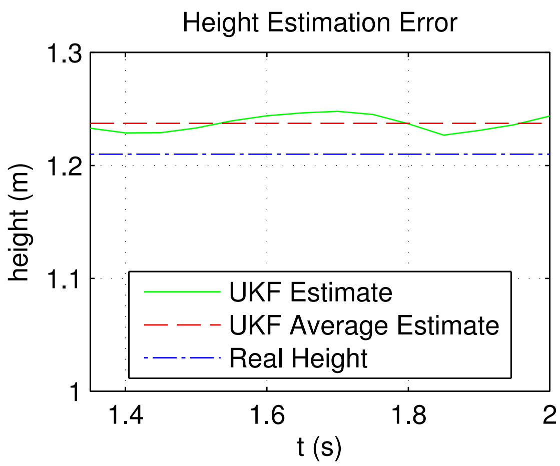
Figure 6: Accuracy tests

I repeated the same test in order to estimate the error in height (z coordinate). In this case I positioned the object at a fixed x, y position and at a known height of 1.2 m. Although figure 7b only shows the error on one case, the reported error of 2.73 cm, significant smaller than in the x, y case is consistent with my observations.

5.2 Position tracking along dynamic trajectories

In order to test the performance of the filters when tracking a moving object, I recorded a demonstration sequence in which I moved the object in a 'descending zig-zag' motion that lasted about 10 s. Figure 8 shows the estimate of the trajectory followed as calculated by the simple estimate, the EKF and the UKF. Although I wasn't able to record the true trajectory that I followed, I believe that the obtained results are very close and represent the real trajectory well.

The individual x, y and z plots are shown in figure figure 9. The left plots show the complete trajectory, while the right plots show a zoomed in region in which a confidence region of ± 2 standard deviations (as calculated with the EKF and UKF) is also shown. As I'll discuss in section 6, the results obtained with the EKF and UKF are similar, both in mean and covariance, once the filters reach steady state.



(a) Plot of height estimate

Real Height (m)	Estimated (m)	**Error (m)**
1.210	1.237	**0.0273**

(b) Height error

Figure 7: Height accuracy test

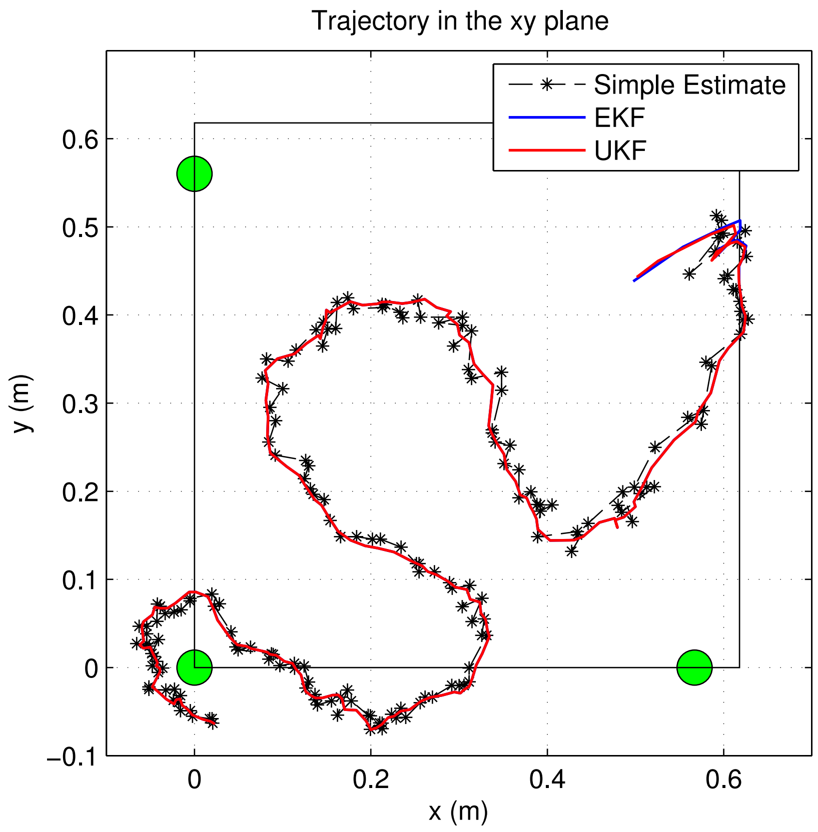


Figure 8: Estimate of the trajectory followed in the xy plane

6 Observations

The results presented in section 5 led me to draw some interesting observations about the ultrasonic positioning system and the designed Kalman filters. These observations are discussed next.

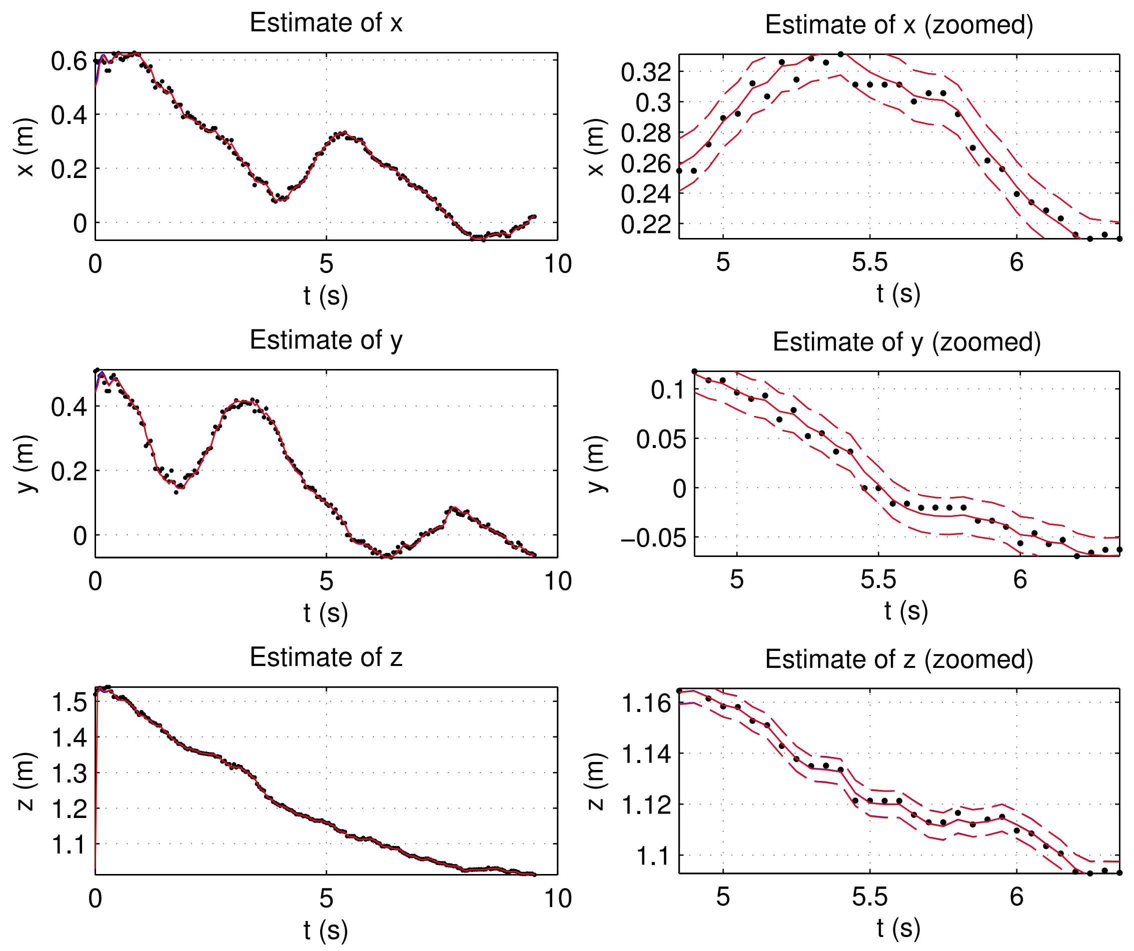


Figure 9: Estimate of x, y and z using the *simple estimator*, the *EKF* and the *UKF*. The black dots represent the results of the simple estimator applied to the captured sensor values. The EKF and UKF estimates are represented in blue and red respectively. The left plots show the complete data for the trajectory shown in figure 8. The right plots show a zoomed region. These plots also show the mean filter estimate ± 2 the standard deviation for the EKF and UKF

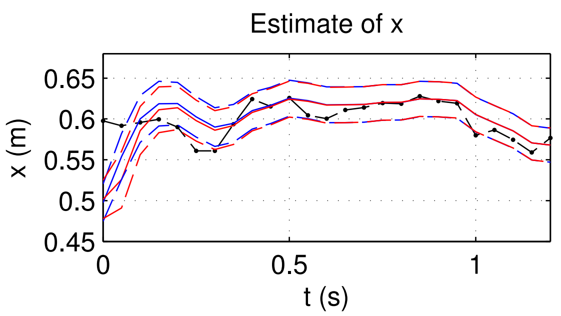
1. Steady state performance of UKF and EKF is similar

One of the most interesting surprises that I got out of this project is that both EKF and UKF performed similarly. After discussing all the problems associated to the EKF in class, I expected that the UKF would perform much better. However, we can see that both the mean estimate (figure 10a) and the covariance (figure 10b) converge to the same value after about 0.5 s (approximately 10 iterations).

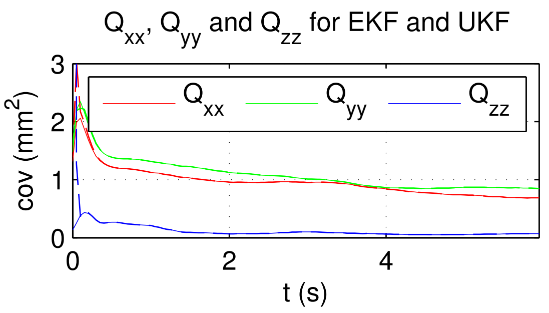
I believe the reason why EKF and UKF perform comparably well is that the non-linearities of the system are not very large. The dynamics of the system were modeled as a linear system and the measurement equation (equation (4.3)) only contains square roots and square functions. I would expect that the UKF filter would perform much better than the EKF in a system that presented more important non-linearities.

2. Height (z) is better estimated than x and y

During my tests with the system, I got the impression that the z position was being estimated more precisely than x and y. This suspicion was confirmed after analyzing the covariance obtained with the EKF and UKF filter. As we can see in figure 10b, the z covariance (Q_{zz}, blue line) is more than ten times smaller than the



(a) EKF and UKF estimates converge to the same value after 0.5 s



(b) x, y and z covariance in solid lines (EKF) and dashed lines (UKF). EKF and UKF covariances converge after 10 iterations.

Figure 10

x and y covariances. I believe this is due to the fact that all three sensors are placed in the $z = 0$ plane and, therefore, are capable of estimating z better than the other two coordinates.

3. EKF and UKF can converge to wrong estimates

The EKF and UKF filters can converge to the wrong estimate in which $z_{est} = -z_{real}$. This can happen under multiple circumstances, such as when the filter is provided with the filters with a bad initial estimate (X_0) or due to spikes in the sensor data. Figure 11 shows a case in which z is wrongly estimated both by the EKF and UKF when the initial estimate is set to $z = -1$ m. Note that even when z is wrongly estimated, x and y are still properly estimated.

Of course, it is completely logical that this could happen, due to the sensor arrangement. The sensors would record the same exact values for an object placed in the same x, y position and inverse z position.

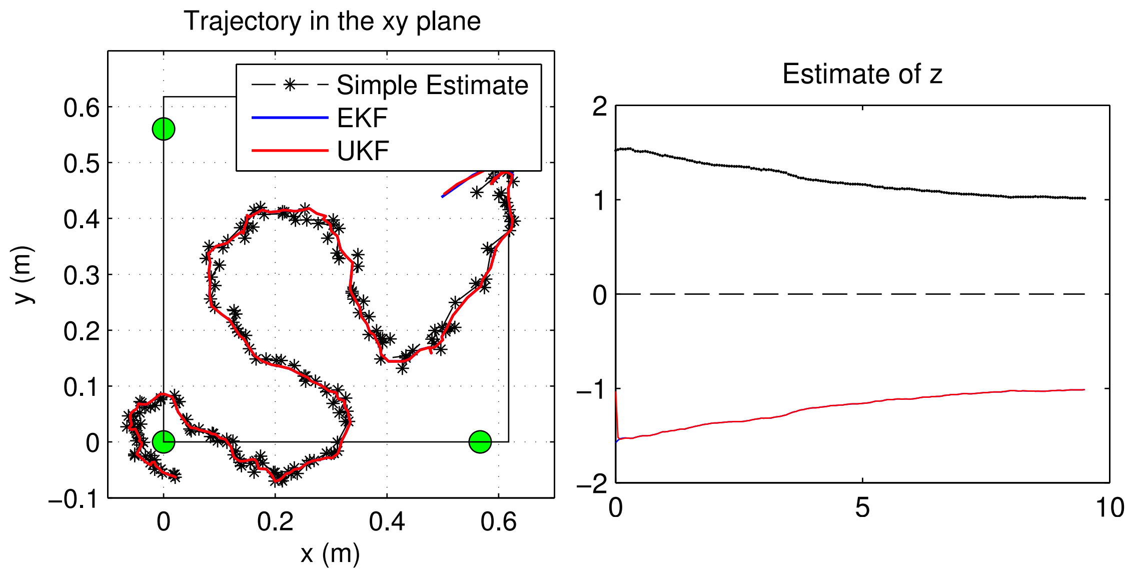


Figure 11: EKF and UKF can converge to the wrong estimate of inverse z position due to multiple factors such as a far initial estimate (X_0)

4. Choosing the noise parameters in real life is difficult

I found it very difficult to choose good values for the sensor (R) and process (W) noise intensity, and even now, at the end of the project, I'm not sure I chose the best values. There are several reasons why I think choosing those parameters was so difficult. In first place, the *process noise* is fictitious and it's only used as a tool to model the uncertainty of where the object is going to move next. Choosing the process noise becomes then a matter of tunning a parameter until the desired performance is achieved.

However, the sensor noise is also problematic. Due to the way the microprocessor was programmed, the sensors could only measure time in integer microseconds, which effectively discretizes the measurements. Moreover, it looks like the measurements I was obtaining were biased in some way (probably because of weak electrical connections, inaccurate distance measurements between sensors and the way the microprocessor was programmed). I also believe that the noise was different in different regions of the localization region, something that would be hard to model properly. Finally, I often obtained spikes in the measured data, maybe caused by ultrasound echoes. All these problems cause the assumption of Gaussian white noise not to hold anymore, and complicate the task of choosing appropriate noise parameters and checking that they are adequate. In the end, I decided to try with many different combinations of parameters until I reached one that met my performance requirements.

7 Conclussions and Lessons Learned

I think I learned a lot while working on this project and I am happy with how it turned out in the end. One of the lessons learned is that working with real hardware takes a lot of time, especially if one is not very familiar with these kind of projects. I think I was overly ambitious with this project and I had to leave out some things I wanted to try. Even then, this project ended up taking a large amount of time, although I believe it was worth it.

I think I could have obtained much better position estimates with this system if I had had more time or had worked with somebody else. Simple things such as making sure the electrical connections were properly secured, having the sensors completely fixed to their locations and accurately measuring the distances between them, and performing the tests in environments that were less prone to producing echo would have dramatically improved my results. I could also have programmed the microcontroller to measure sub microsecond time, which would have also helped improve the quality of the distance measurements.

I really enjoyed implementing the Extended and Unscented Kalman Filters and seeing for myself that they can be very useful in real life. Before taking this class, I already knew what the Kalman filter was for, but I had never had to use it (nor did I know how to). Learning to use the Kalman filters and getting good estimates was a great experience. I am happy I chose to implement the filters on real data, as I had to deal with real-life problems in order to get appropriate results. Finally, I could also see for myself that choosing the right parameters for noise is an art on itself and that designing good Kalman filters requires a lot of experience.

A Source Code and Data

All the source code and recorded data used in this project is available in the folder `project_code`, submitted with this report.

The instructions to run the MATLAB code are written in the file `README.md`. In order to generate the plots presented in this report, please run the MATLAB script `run_1633_Project.m` located in the `project_code/matlab` folder.

References

[1] William Myers and Guo Chin. 3D Paint, January 2012. URL http://people.ece.cornell.edu/land/courses/ece4760/FinalProjects/s2012/wgm37_gc348/wgm37_gc348/.