# CMPE 245 Project 2

Leo Chen

June 12, 2018

**Abstract**

The second project is focused on a miniature quadrotor. Through the dynamical model, the task is to design an extended Kalman Filter (EKF) for the model.

## 1 Introduction

For the second project of the course, the task is to design an EKF for the movement of a quadrotor. We are given the dynamical system and measurements to design a Kalman filter for the model in the first part. The second part requires us to include c in our Kalman filter and find the best fit for the parameter by considering the residuals. Lastly, the third part asks only for the x-y dynamics. This part however has corrupted measurements that the Kalman filter should be able to notice.

## 2 Methodology

### 2.1 Provided Data

For this project we are given the data struct of the x-,y-, and z-coordinates, as well as the pitch, roll and, yaw angles. Additionally the thrust and the change of time (dtrec) are all provided in the same 'data3D.mat'-file. The manual provides the dynamical model of the quadrotor which is

$$\ddot{x} = c * g(sin\psi sin\phi + cos\psi cos\phi sin\theta)$$

$$\ddot{y} = c * g(sin\psi cos\phi sin\theta - cos\psi sin\phi)$$

$$\ddot{z} = c * gcos\phi cos\theta - g)$$

while g is the gravitational acceleration (=9.81kg/$s^2$) and $\theta$, $\phi$, $\psi$ are the pitch, roll, and yaw angles respectively. For the first part, c=1. The noisy measurements are provided in form of

$$x_m(t) = x(t) + n_x(t)$$

$$y_m(t) = y(t) + n_y(t)$$

$$z_m(t) = z(t) + n_z(t)$$

$$\psi_m(t) = \psi(t) + n_\psi(t)$$

The yaw angle measurement is modeled as

$$\psi = \sigma_\psi$$

For part 2, c is changed to

$$c = (0.000409s^2 + 0.1405s - 0.099)/c_0$$

with s=255/6000*(thrust). For the third part of the project, there are new sets of data that are recorded with a sample time of $T_s = 0.1s$.

There are also models and matrices provided for us in the 'parameters' document. The discrete model for part 1 and 2 is

$$x_{k+1} = x_k + v_k^x \Delta T_k$$

$$y_{k+1} = y_k + v_k^y \Delta T_k$$

$$z_{k+1} = z_k + v_k^z \Delta T_k$$
$$v_{k+1}^x = v_k^x + c * g(sin\psi_k sin\phi_k + cos\psi_k cos\phi_k sin\theta_k)\Delta T_k + w_k^x$$
$$v_{k+1}^y = v_k^y + c * g(sin\psi_k cos\phi_k sin\theta_k - cos\psi_k sin\phi_k)\Delta T_k + w_k^x$$
$$v_{k+1}^z = v_k^z + (c * gcos\phi_k cos\theta_k - g)\Delta T_k + w_k^x$$

which ultimately is my x(t). Note that the third and sixth equation with respect to the z-coordinate can be ignored in the third part of the project since we only care about the x-y coordinates for that part of the project. The Q matrix for part 1 and 2 is

$$Q = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0.01 \end{bmatrix} \Delta T_k$$

and

$$Q = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0.01 \end{bmatrix} \Delta T_k$$

in part 3. The R matrix in part 1 and 2 is given

$$R = \begin{bmatrix} 0.01^2 & 0 & 0 & 0 \\ 0 & 0.01^2 & 0 & 0 \\ 0 & 0 & 0.01^2 & 0 \\ 0 & 0 & 0 & (\pi/180)^2 \end{bmatrix}$$

and is

$$R = \begin{bmatrix} 0.01^2 & 0 & 0 \\ 0 & 0.01^2 & 0 \\ 0 & 0 & 0.01^2 \\ 0 & 0 & (\pi/180)^2 \end{bmatrix}$$

in part 3. The matrix H in part 3 is

$$H = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Note that, when the measurements are corrupted in part 3, the matrices R and H change to

$$R = \begin{bmatrix} 0.01^2 & 0 \\ 0 & 0.01^2 \end{bmatrix}, H = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{bmatrix}$$

Now that all the data is given, the next step would be to calculate the derivatives in order to find the matrices.

## 3 Mathematical Approach

### 3.1 Formulas for EKF

The equations that are important to remember are inside the prediction and update step in the discrete time model. The predicted state estimate inside the prediction step is defined by

$$\hat{x}_{k+1}(-) = f_k(\hat{x}_k, 0)$$

while the predicted covariance estimate is

$$P_{k+1}(-) = \Phi_k P_k \Phi_k^T + \Gamma_k Q_k \Gamma_k^T$$

The update step consists of the Kalman gain and the updated state and covariance estimate. The Kalman gain is defined by

$$K_{k+1} = P_{k+1}(-)H_{k+1}^T(H_{k+1}P_{k+1}(-)H_{k+1}^T + R_{k+1})^{-1}$$

while the updated state estimate and the updated covariance estimate are respectively defined by

$$\hat{x}_{k+1} = \hat{x}_{k+1}(-) + K_{k+1}(y_{k+1} - h_{k+1}(\hat{x}_{k+1}(-)))$$

$$P_{k+1} = (I - K_{k+1}H_{k+1})P_{k+1}(-)$$

The state transition matrix $\Phi$ is defined by

$$\Phi_k = \frac{\delta f_k(\hat{x}_k, 0)}{\delta x_k}$$

$\Gamma$ is defined by

$$\Gamma_k = \frac{\delta f_k(\hat{x}_k, 0)}{\delta w_k}$$

and the observation matrix H is defined by

$$H_k = \frac{\delta h_k(\hat{x}_k(-), 0)}{\delta x_k}$$

The equations will help me to take the derivatives of $\dot{X}(t)$ with respect to the corresponding parameter to plug it into the prediction and update step of my Matlab code.

## 3.2 Calculations for EKF design

To find $\Phi_k$ I took the partial derivative of $f_k$ (rows) with respect to $x_k$ (columns) and ultimately gives me a matrix of

$$\Phi_k = \Phi_k = \frac{\delta f_k(\hat{x}_k, 0)}{\delta x_k} = \begin{bmatrix} 1 & 0 & 0 & \Delta T & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & \Delta T & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & \Delta T & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & \Delta T * g * (cos\psi_k sin\phi_k - sin\psi_k cos\phi_k sin\theta_k) \\ 0 & 0 & 0 & 0 & 1 & 0 & \Delta T * g * (cos\psi_k cos\phi_k sin\theta k + sin\psi_k sin\phi_k) \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

The $\Phi$ in part 2 however is different since now $c \neq 0$ and I would have to consider that for the derivative. The result for $\Phi$ in the second part is computed similarly and ultimately is

$$\Phi_k = \Phi_k = \frac{\delta f_k(\hat{x}_k, 0)}{\delta x_k} = \begin{bmatrix} 1 & 0 & 0 & \Delta T & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & \Delta T & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & \Delta T & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & c * \Delta T * g * (cos\psi_k sin\phi_k - sin\psi_k cos\phi_k sin\theta_k) \\ 0 & 0 & 0 & 0 & 1 & 0 & c * \Delta T * g * (cos\psi_k cos\phi_k sin\theta k + sin\psi_k sin\phi_k) \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

In the third part of the project, the z does not have to be considered which therefore is a different $\Phi$, as well

$$\Phi_k = \Phi_k = \frac{\delta f_k(\hat{x}_k, 0)}{\delta x_k} = \begin{bmatrix} 1 & 0 & \Delta T & 0 & 0 \\ 0 & 1 & 0 & \Delta T & 0 \\ 0 & 0 & 1 & 0 & \Delta T * g * (cos\psi_k sin\phi_k - sin\psi_k cos\phi_k sin\theta_k) \\ 0 & 0 & 0 & 1 & \Delta T * g * (cos\psi_k cos\phi_k sin\theta k + sin\psi_k sin\phi_k) \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Finding $\Gamma_k$ is similar in which I need to take the partial derivative of $f_k$ (rows) with respect to $w_k$ (columns) and gave me following matrix

$$\Gamma_k = \Gamma_k = \frac{\delta f_k(\hat{x}_k, 0)}{\delta w_k} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ \sigma & 0 & 0 & 0 \\ 0 & \sigma & 0 & 0 \\ 0 & 0 & \sigma & 0 \\ 0 & 0 & 0 & \sigma \end{bmatrix}$$

and since we are not considering the z-coordinates in part 3, the $\Gamma$ matrix changes to

$$\Gamma_k = \Gamma_k = \frac{\delta f_k(\hat{x}_k, 0)}{\delta w_k} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ \sigma & 0 & 0 \\ 0 & \sigma & 0 \\ 0 & 0 & \sigma \end{bmatrix}$$

$\sigma$ has to be adjusted at the end to find the best value for a solid filter. The matrix H is derived similarly to the others. I used the derivative of the observation model (rows) with respect to $x_k$

$$H_k = \frac{\delta h_k(\hat{x}_k(-))}{\delta x_k} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

while, for the same reasons as previous, changes for part 3 to

$$H_k = \frac{\delta h_k(\hat{x}_k(-))}{\delta x_k} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Provided with all the calculated matrices, the next step is to code the Matlab file.

## 3.3 Matlab Implementation

Note: The Matlab files are included in the submission e-mail.

### 3.3.1 Part 1

I first did all the declarations of variables and assignments. $\sigma$ can be defined at the top of the code so that way I will be able to change the noise for a better result. Since R is already given I left it as it is. For the initial p, I used the similar method as for the previous project in which I defined P as

$$P = \begin{bmatrix} 0.01^2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.01^2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.01^2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{0.01^2}{\Delta T} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{0.01^2}{\Delta T} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{0.01^2}{\Delta T} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & (\frac{\pi}{180})^2 \end{bmatrix}$$

I took the noise that was suggested for P11-P33 and used the same noise divided by $\Delta T^2$ for the variance. To find the initial values for my functions x, I just used the initial (44th) value of the data provided for us. For the velocity I took the first two data points and subtracted them with each other and then divided by the time difference. Inside the for-loop I still have my Q and $\Phi$ matrix declared since those change due to the changing $\Delta T$. Then I have my prediction and update step written down, as I mentioned earlier on how to do that and plotted all the data I had.

## 3.4 Part 2

For this part I need to use a nested for-loop. Most of the things stay the same, except that this time there is a 'c' to consider. First of all, it is important that there is a c0 for our outer for-loop to find the perfect value for c0. At the end we should be able to see a plot in which the c0 is the most optimal while running the loop. Inside the beginning of the EKF, I added the formulas provided for us

$$s = 255/6000 * (thrust), c = \frac{0.000409 * s^2 + 0.1405 * s - 0.099}{c0}$$

This goes into the inner for-loop which means that it will be at the top of the EKF. The inner for-loop stays similar for most of the part. The matrix $\Phi$ as well as my functions change as mentioned earlier, since 'c' has to be considered now. At the end of the update step, I sum up the residuals and store it in another variable. The outer for-loop iterates through c0. At the beginning it declares my P matrix and my functions. Then, it clears the saved sum value and runs the EKF once more. The sum is then divided by N and is plotted against c0 to find the perfect value.

### 3.5 Part 3

The third part of the project is similar to the first part of the project. Some matrices are different that I mentioned earlier. The only difference in this part is the corrupted data that is happening. To deal with it, I tried to calculate the residual of the yaw by subtracting the provided data, with my predicted data and squaring it. Then I used an if-else-statement after my prediction step and before my update step in which I say if the residual is less than $\pi/5$ then the data is not corrupted and we can use

$$H = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}, R = \begin{bmatrix} 0.01^2 & 0 & 0 \\ 0 & 0.01^2 & 0 \\ 0 & 0 & (\frac{\pi}{180})^2 \end{bmatrix}$$

else the matrices would look as following

$$H = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{bmatrix}, R = \begin{bmatrix} 0.01^2 & 0 \\ 0 & 0.01^2 \end{bmatrix}$$

and we could set K=0. Besides that, everything stays the same in the implementation

## 4 Results

### 4.1 Part 1

The results for part 1 looks as following As you can see the results look really satisfying. The data

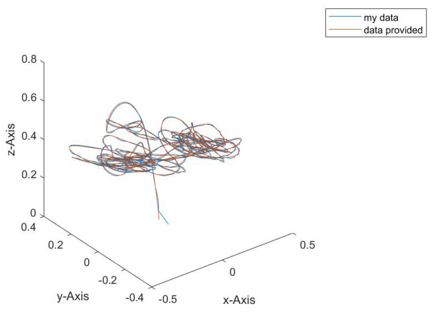

Figure 1: XYZ-Diagram of EKF result

almost aligns with my predictions and therefore this can be called a successful filter. I was varying the noise $\sigma$, as well and turned out to leave it at around 3. If it would be any different, such as for example =1, then this would result in figure 2 and not be as good as a result anymore. You can also see that the predicted $\dot{x}$ and $\dot{y}$, as well as the yaw angle are very similar to the data that was provided in the following figures below.

### 4.2 Part 2

For this part of the lab the result of the data is very similar to the data from part 1 as you can see. The $\dot{x}$, $\dot{y}$, and yaw angle are similar, as well. Now this part of the project focuses on the graph of the sum vs c0. I was expecting a parabola to come out. However, there is some error in my code in which the graph is not the parabola. The result I am getting is in figure 7. Since this is not the expected result that I wanted I will evaluate on what was expected and what the next step would be. The task is to find the best value for the parameter c0 which means that idealistically, the graph should have been a parabola in which I could be able to find the lowest point of the parabola. Once I have a certain area of the lowest point, I would redefine the for-loop to a smaller

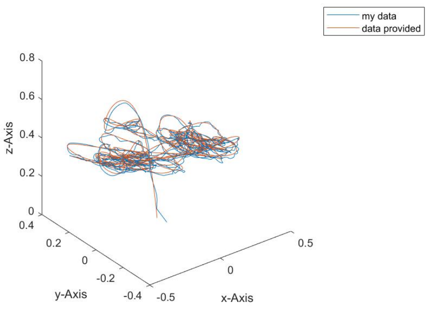

Figure 2: Part1: EKF with bad sigma value

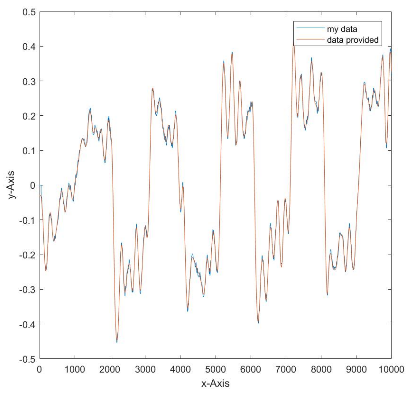

Figure 3: Part1: xdot predicted vs provided

scale and redo this step until I can see what exactly the c0 value is. This value would then be the ultimate value to plug into c which could get rid of the residuals as much as possible.

### 4.3 Part 3

For the last part of the project, I am only looking for the x-y data. My predicted data should be similar to the provided data. However, something in my predict or update state is incorrect and causes a lot of wrong data. My implementation seems right to me for the most part, however I cannot find where the bug is. Conceptually, I understand that I am supposed to write an if-else-statement that can detect the corrupted measurement and handle it accordingly. However, I am still not very familiar with Matlab to fully understand where my error is at. The faulty data is provided below

## 5 Conclusion

Overall the second project gave a better insight on applied estimated Kalman Filters. The task was related to real-life engineering applications of a quadrotor and involved designing a Kalman filter for it. The first part was another regular Kalman filter which was a good exercise to get used to design Kalman filters. The second part included using the average sum of square of the Kalman filter residuals and gave me a good insight on how to handle the residuals. Lastly, the third part had corrupted data in the x-y dynamics and challenged me to design a robust filter that can deal with the corrupted data. Even though this project was not completely successful, I still gained a lot of knowledge on different aspects of the Kalman filter which was really helpful and interesting for future applications.

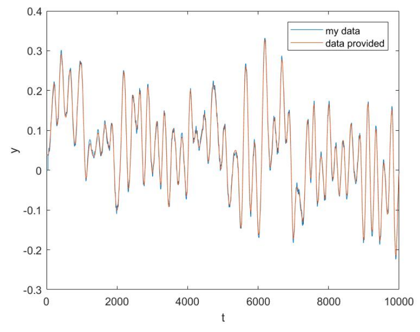

Figure 4: Part1: ydot predicted vs provided

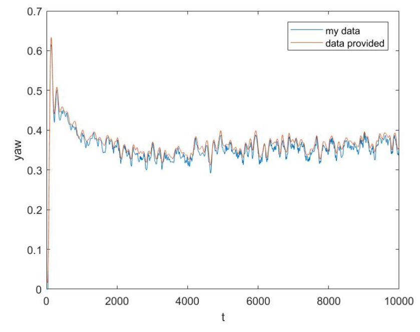

Figure 5: Part1: yaw predicted vs provided

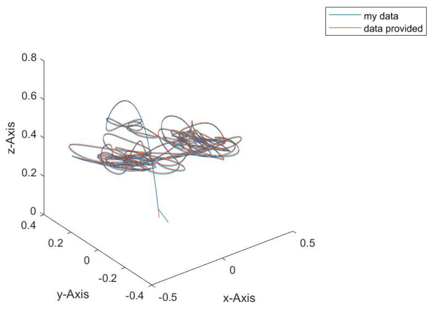

Figure 6: Part 2: quadrotor dynamics

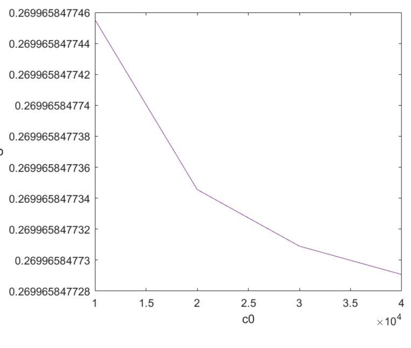

Figure 7: Part 2: Finding c0

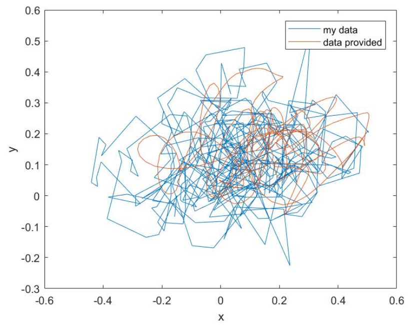

Figure 8: Part 3: X-Y Dynamics

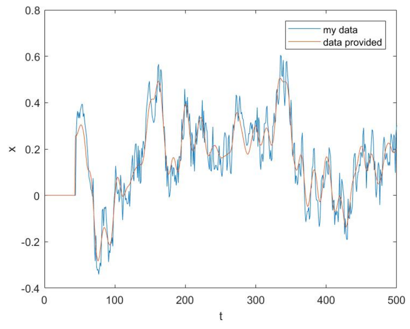

Figure 9: Part 3: X Dynamics

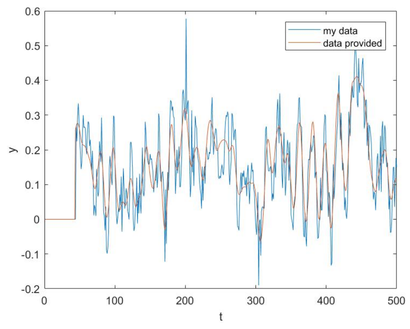

Figure 10: Part 3: Y Dynamics

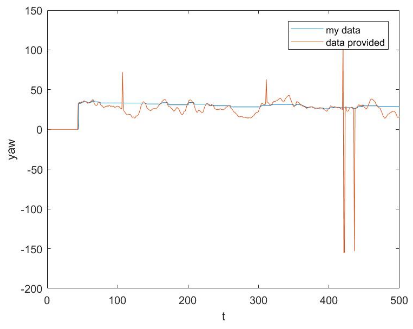

Figure 11: Part 3: Yaw