16-299 Assignment: Controlling a TWIP

Deriving the Dynamics

How could you check my work? One good way is to create a TWIP model in a simulation package like ODE, Gazebo, or Bullet and see if the resulting simulations matched. Only do this if you think it will be fun.

It was fun for me. Ode version of dynamics I checked against: `http://www.cs.cmu.edu/~cga/controls-intro/twip-ode.zip`

I also computed the energy of the system (which I had already generated using the Lagrangian approach), and checked if that was constant when zero torque is applied. The following is inserted into main.m after the command uu is computed (around line 52):

```
% Energy check (in Matlab)
 th = xx(2);
 dx = xx(3);
 dth = xx(4);
 energy = (I_p*dth^2)/2 + (dx^2*m_p)/2 ...
  + (dx^2*(m_w*r_w^2 + I_w))/(2*r_w^2) ...
  + (dth^2*l_p^2*m_p)/2 ...
  + g*l_p*m_p*cos(th) + dth*dx*l_p*m_p*cos(th);
 e_array(i,1) = energy;
```

All this checking led me to find a bug in the dynamics I gave you! See below.

Extra credit/a project is available if you augment the model to include varying ground contact force, slip, deformable tires, and liftoff, and your controller can handle these issues. ...

I haven't done this.

Extra credit/a project is available to make a 3D model and control and the lab robot to drive along paths and turn.

I haven't done this.

Other projects involve 1) driving over hilly terrain, curbs, and potholes, and 2) getting the robot to dance to music.

Haven't done this either.

TWIP dynamics

The dynamics I gave you had an error in it. Here is what I gave you:

$$\begin{pmatrix} \frac{I_w}{r_w^2} + m_p + m_w & l_p * m_p * \cos(\theta) \\ l_p * m_p * \cos(\theta) & I_p + l_p^2 * m_p \end{pmatrix} \begin{pmatrix} \ddot{x} \\ \ddot{\theta} \end{pmatrix} = \begin{pmatrix} r_w * (-\tau + \dot{\theta}^2 * l_p * m_p * r_w * \sin(\theta)) \\ \tau + \mathrm{G} * l_p * m_p * \sin(\theta) \end{pmatrix} \quad (1)$$

Here is the correct dynamics. The top element on the right hand side needed to be divided by r_w^2.

$$\begin{pmatrix} \frac{I_w}{r_w^2} + m_p + m_w & l_p * m_p * \cos(\theta) \\ l_p * m_p * \cos(\theta) & I_p + l_p^2 * m_p \end{pmatrix} \begin{pmatrix} \ddot{x} \\ \ddot{\theta} \end{pmatrix} = \begin{pmatrix} (-\tau/r_w + \dot{\theta}^2 * l_p * m_p * \sin(\theta)) \\ \tau + \mathrm{G} * l_p * m_p * \sin(\theta) \end{pmatrix} \quad (2)$$

`http://www.cs.cmu.edu/~cga/controls-intro/ass1/twip-corrected.m`

I will give answers below assuming the model I gave you is correct (which it is not). All the code I mention is in

`http://www.cs.cmu.edu/~cga/controls-intro/ass1/twip-manual-ass1.zip`

The Actual Assignment

A. Try to design a controller manually

A1) Given the example Matlab program in **twip-manual.zip**, *try to design satisfactory manual gains.*

Many possible answers here. You can check them by running them in a simulation using main.m. An example answer is

```
k = [ 1.0 60.0 10.0 10.0 ];
```

A2) Set the duration to 100.0 (100s). Try to minimize the one step criterion $dt * (x^2 + \theta^2 + \tau^2)$ *by manually choosing gains. What are your best gains?*

I am using the linearized dynamics and LQR.

```
>> A = [1  0         0.01 0
        0  1         0    0.01
        0 -0.066508 1    0
        0  0.149644 0    1 ];
>>
>> B = [ 0
         0
        -0.008271
         0.016610 ];
>>
>> Qc = [ 1 0 0          0
          0 1 0          0
          0 0 0.00000001 0
          0 0 0          0.00000001 ];
>>
>>
```

```
>> Rc = [ 1 ];
>>
>> [Kc Sc Ec] = dlqr( A, B, Qc, Rc )

Kc =

    0.9606   20.2454    5.1969    7.4260


Sc =

   1.0e+03 *

    0.5410    0.7730    1.4607    0.7900
    0.7730    5.9836    3.3843    2.9580
    1.4607    3.3843    6.9223    3.7854
    0.7900    2.9580    3.7854    2.3567


Ec =

   0.9978 + 0.0020i
   0.9978 - 0.0020i
   0.9620 + 0.0021i
   0.9620 - 0.0021i
```

B. Linearize the dynamics

Linearize the TWIP dynamics in the straight up configuration. What are $\mathbf{A}$ *and* $\mathbf{B}$*? Which of the symbolic terms in the dynamics in the* $\mathbf{M}(\mathbf{x})\mathbf{a} = \mathbf{v}(\mathbf{x})$ *equation above could you have ignored in the first place? Why?*

Using `http://www.cs.cmu.edu/~cga/controls-intro/ass1/linearize.m`, I get the following $\mathbf{A}$ and $\mathbf{B}$ matrices for the twip.m code I gave you (0.01 is the sampling interval, which multiplies each of the continuous time linearization numbers).

$$\mathbf{A} = \begin{pmatrix} 1 & 0 & 0.01 & 0 \\ 0 & 1 & 0 & 0.01 \\ 0 & -0.066508 & 1 & 0 \\ 0 & 0.149644 & 0 & 1 \end{pmatrix} \tag{3}$$

$$\mathbf{B} = \begin{pmatrix} 0 \\ 0 \\ -0.008271 \\ 0.016610 \end{pmatrix} \tag{4}$$

In linearization, $\cos(\theta) \rightarrow 1$ and $\sin(\theta) \rightarrow \theta$, for $\theta \leq 30°$. and terms including products of velocity (centripetal and Coriolis terms) are zero, since they are quadratic in small values.

C. Design an LQR controller

C1) Choose an optimization criterion ($\mathbf{Q}$ *and* $\mathbf{R}$*) and design a corresponding LQR controller. Implement it for the simulation (and later for the lab). What is your* $\mathbf{Q}$*,* $\mathbf{R}$*, and corresponding* $\mathbf{K}$*? Evaluate the resulting control in simulation, and explain why you chose the optimization criterion you did.*

Many possible answers, depending on your linear model and optimization criterion. I initially made Qc and Rc identity matrices:

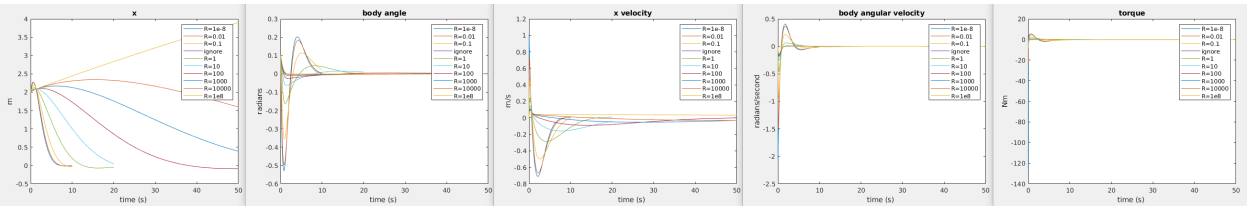
$$\mathbf{Q}_c = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{5}$$

$$\mathbf{R}_c = \begin{pmatrix} 1 \end{pmatrix} \tag{6}$$

```
A = [1  0         0.01 0
     0  1         0    0.01
     0 -0.066508 1    0
     0  0.149644 0    1 ];

B = [ 0
      0
     -0.008271
      0.016610 ];

Qc = [ 1 0 0 0
       0 1 0 0
       0 0 1 0
       0 0 0 1 ];

Rc = [ 1 ];
```

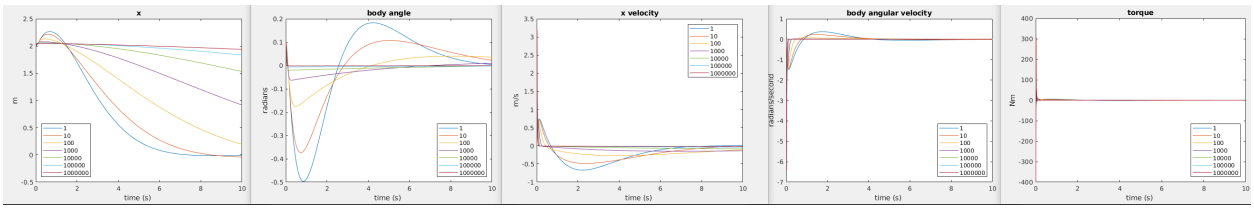
```
[Kc Sc Ec] = dlqr( A, B, Qc, Rc )

Kc =

    0.9595   20.3625    5.3084    7.6125


Sc =

   1.0e+03 *

    0.5532    0.7934    1.4776    0.7985
    0.7934    6.2228    3.5829    3.0662
    1.4776    3.5829    7.1695    3.9166
    0.7985    3.0662    3.9166    2.4353


Ec =

   0.9978 + 0.0020i
   0.9978 - 0.0020i
   0.9694 + 0.0000i
   0.9524 + 0.0000i
```

Here are plots of movements to zero starting from [2,0.1,0,0] where R was varied from 1e-8 (cheap control) to 1e8 (expensive control) while Q was the identity matrix:



One thing to note is that there is little difference between R=1e-8 and R=1e-2, so I am not plotting everything in between. The TWIP control problem limits the fastest speed the robot can move, because torques are limited or the pendulum will fall over. Similarly there is a minimum initial x velocity at the very beginning of the x velocity plot to move the body vertical, or it will fall over. The maximum body angle, x velocity, body angular velocity, and torque rapidly decrease as R increases.

In the following plots $\mathbf{Q}$ was the identity matrix, and $\mathbf{R}$ was 0.01. $\mathbf{Q}(2,2)$, the penalty on body angle, was varied from 1 to 1000000. The maximum body angle was reduced as $\mathbf{Q}(2,2)$ increased:



C2) Vary the diagonal components of $\mathbf{Q}$ *and* $\mathbf{R}$ *over several orders of magnitude (..., 1e-2, 1e-1, 1, 10, 100, ...) and generate a root locus plot for each component. When do which components of* $\mathbf{Q}$ *and* $\mathbf{R}$ *matter? What is the minimal set of* $\mathbf{Q}$ *and* $\mathbf{R}$ *components that give you good control of pole locations?*

The root locus plots depend on what $\mathbf{Q}$ and $\mathbf{R}$ you start with, and what parameter you change. If you change multiple parameters along a one-dimensional path, the root locus that results depends on the direction or path in parameter space you take. I will look at root locus plots for two starting points. One is $\mathbf{Q}$ and $\mathbf{R}$ are identity matrices.

There are 5 parameters (the non-zero elements of $\mathbf{Q}$ and $\mathbf{R}$). However, since scaling the parameters by the same scale factor has no effect on the gains chosen by LQR design, there are only 4 dimensions to explore. I don't have a good way to explore 4 dimensions, so I will only search a single direction for each of the 5 parameters by scaling one parameter at at time up and down. I indicate the root locus point corresponding to the lowest value of the parameter with an open circle, and the root locus point corresponding to the highest value of the parameter with an open square. To figure out how roots move, start at open circles and follow the paths to open squares.

Here are the root locus plots for starting from $\mathbf{Q}$ and $\mathbf{R}$ are identity matrices, and varying each parameter (Figures 1, 2, 3, 4, and 5). I used a program I wrote to generate these figures: `http://www.cs.cmu.edu/~cga/controls-intro/ass1/my_root_locus.m`

Now let's take a different $\mathbf{Q}$ and $\mathbf{R}$ as the starting point for root locus plots:

$$\mathbf{Q}_c = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 10 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{7}$$

$$\mathbf{R}_c = \begin{pmatrix} 0.01 \end{pmatrix} \tag{8}$$

Look at Figures 6, 7, 8, 9, and 10).

None of these searches found a $\mathbf{Q}$ and $\mathbf{R}$ that moved all the poles away from (1,0). This meant some modes were always going to be slow. To explore the space more, I then used a program I wrote to try out random Q matrices, `http://www.cs.cmu.edu/~cga/controls-intro/ass1/random_Q.m`. (Why didn't I have to also vary $\mathbf{R}$?) I only paid attention to control designs that were critically or overdamped in all modes, and had the minimum maximal root magnitude, to get all poles away from (1,0). The logic here is to find controllers that don't oscillate and reduce errors as fast as possible. Some of these designs were nice. Try out the program and see what you get.

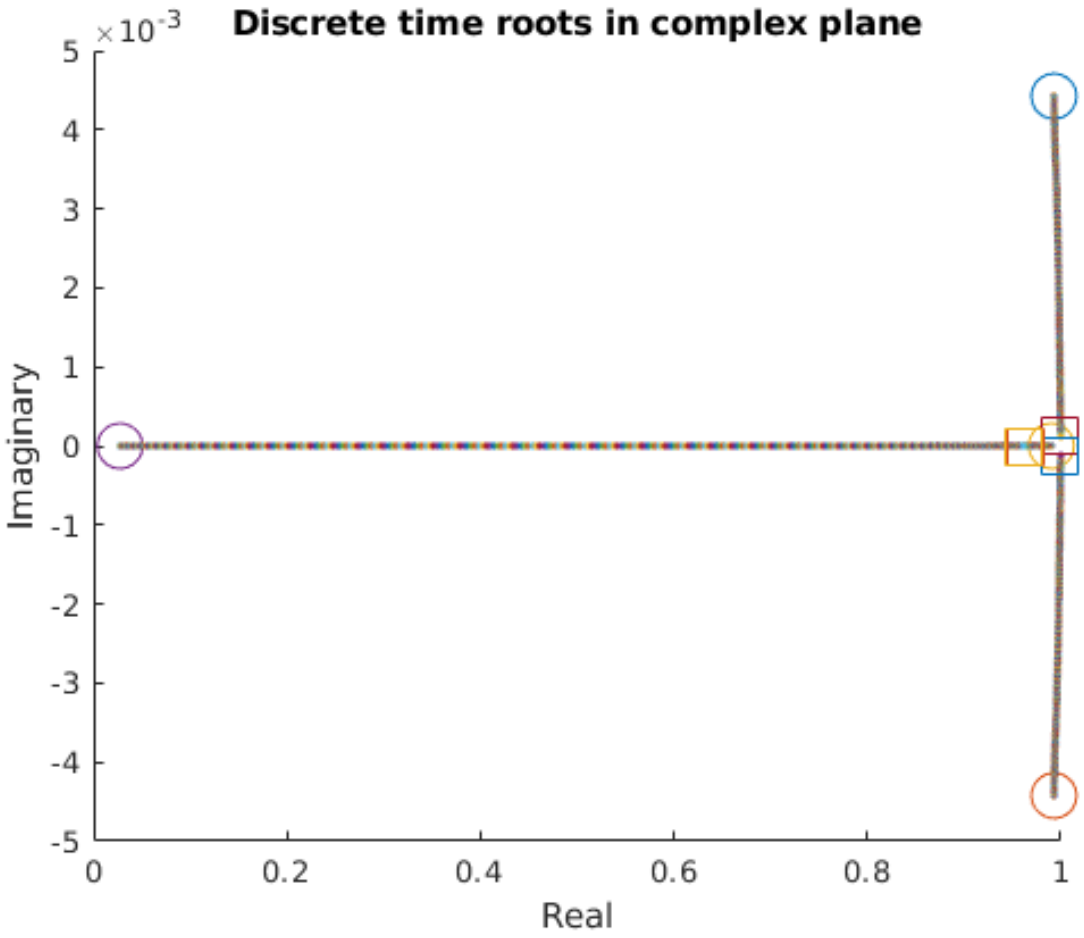


Figure 1: Varying R(1,1), starting from Q=diag(1,1,1,1), R=1. I want all the roots to be near the X axis so they don't cause oscillations, which is achievable. However, no matter what R(1,1) I choose, there will be some poles with magnitude close to 1, which will decay slowly.

D. Design a Kalman filter

D1) Choose a noise model, design a Kalman filter based on measurement of only x and θ, and implement it for the simulation (and later for the lab). What is your $\mathbf{Q}_{\mathrm{KF}}$, $\mathbf{R}_{\mathrm{KF}}$, and corresponding $\mathbf{K}_{\mathrm{KF}}$? Evaluate the resulting control in simulation with added noise, and explain why you chose the optimization criterion you did.

I think the encoder is not noisy, so let's say the encoder could be off by a count, plus or minus. One count is pi/780 = 0.004 radians. To put this estimate in the measurement noise matrix $\mathbf{R}$, I would square 0.004 to get 0.000016.

The accelerometer is very noisy, so let's say its noise is 0.1 radian. We square that to get 0.01 to put in the measurement noise matrix.

The question should have said you also have a gyro, which turns out to be pretty clean. But we

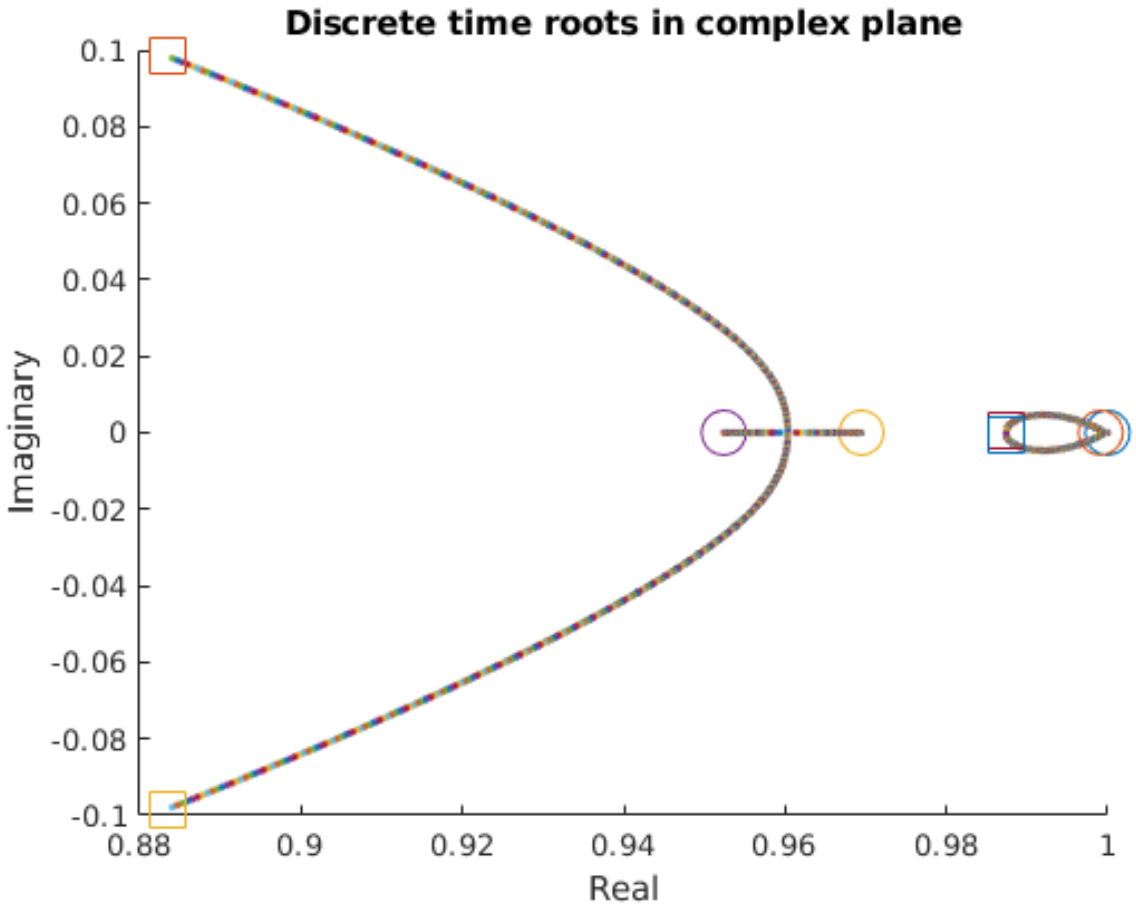


Figure 2: Varying Q(1,1), starting from Q=diag(1,1,1,1), R=1. There is not a lot I can do about those two roots that are close to (1,0).

will answer the question assuming no gyro.

In terms of the process noise matrix $\mathbf{Q}$, I think the "noise" should be low on the encoder and body angle dynamics, but high on the encoder velocity and body angular velocity dynamics.

So ....

$$\mathbf{A} = \begin{pmatrix} 1 & 0 & 0.01 & 0 \\ 0 & 1 & 0 & 0.01 \\ 0 & -0.066508 & 1 & 0 \\ 0 & 0.149644 & 0 & 1 \end{pmatrix} \tag{9}$$

$$\mathbf{C} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} \tag{10}$$

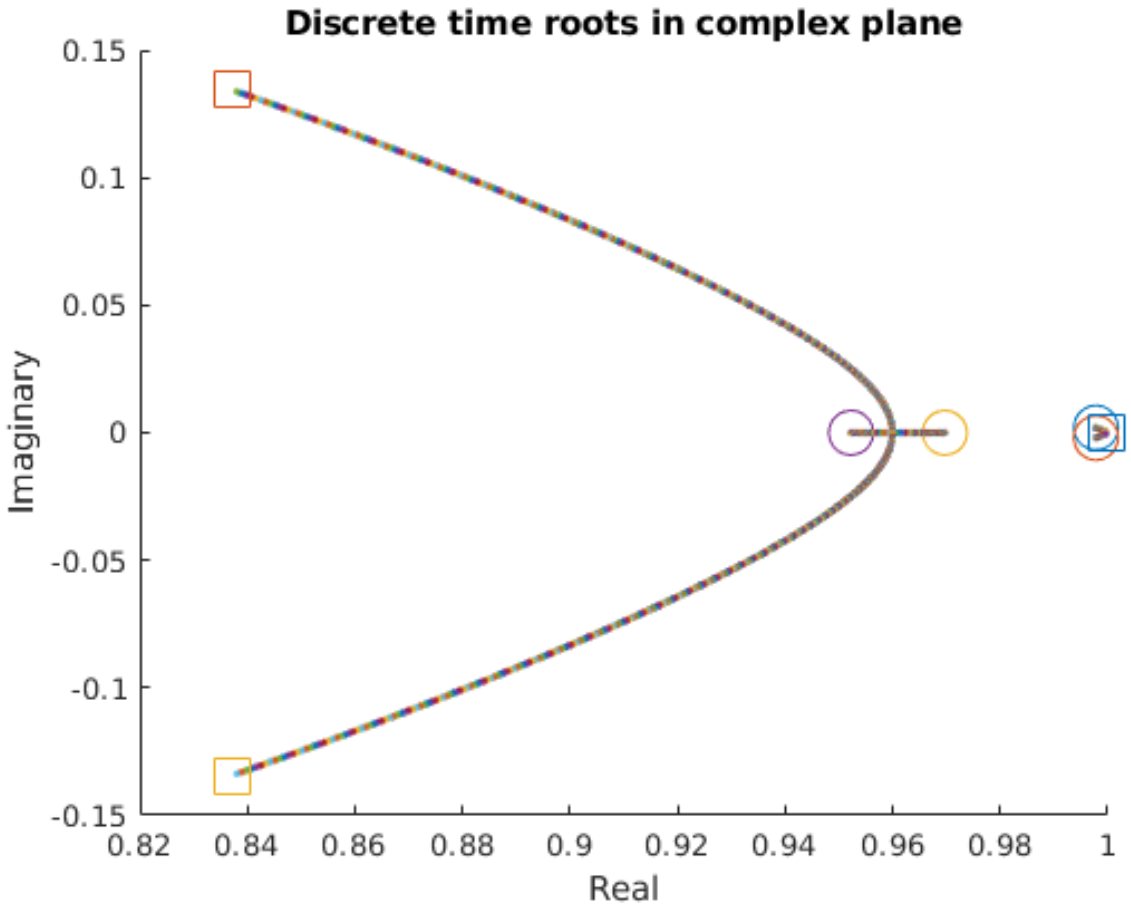


Figure 3: Varying Q(2,2), starting from Q=diag(1,1,1,1), R=1. There is not a lot I can do about those two roots that are close to (1,0).

$$\mathbf{Q}_f = \begin{pmatrix} 0.000016 & 0 & 0 & 0 \\ 0 & 0.000016 & 0 & 0 \\ 0 & 0 & 0.01 & 0 \\ 0 & 0 & 0 & 0.01 \end{pmatrix} \tag{11}$$

$$\mathbf{R}_f = \begin{pmatrix} 0.000016 & 0 \\ 0 & 0.01 \end{pmatrix} \tag{12}$$

```
[Kf Sf Ef] = dlqr( A', C', Qf, Rf )
Kc = [ 3.6004   31.2493   16.6414   19.0620 ];
```

This set of numbers worked in simulation, but the noise (especially the body process and measurement noise) is not aesthetically pleasing (Figure 11).

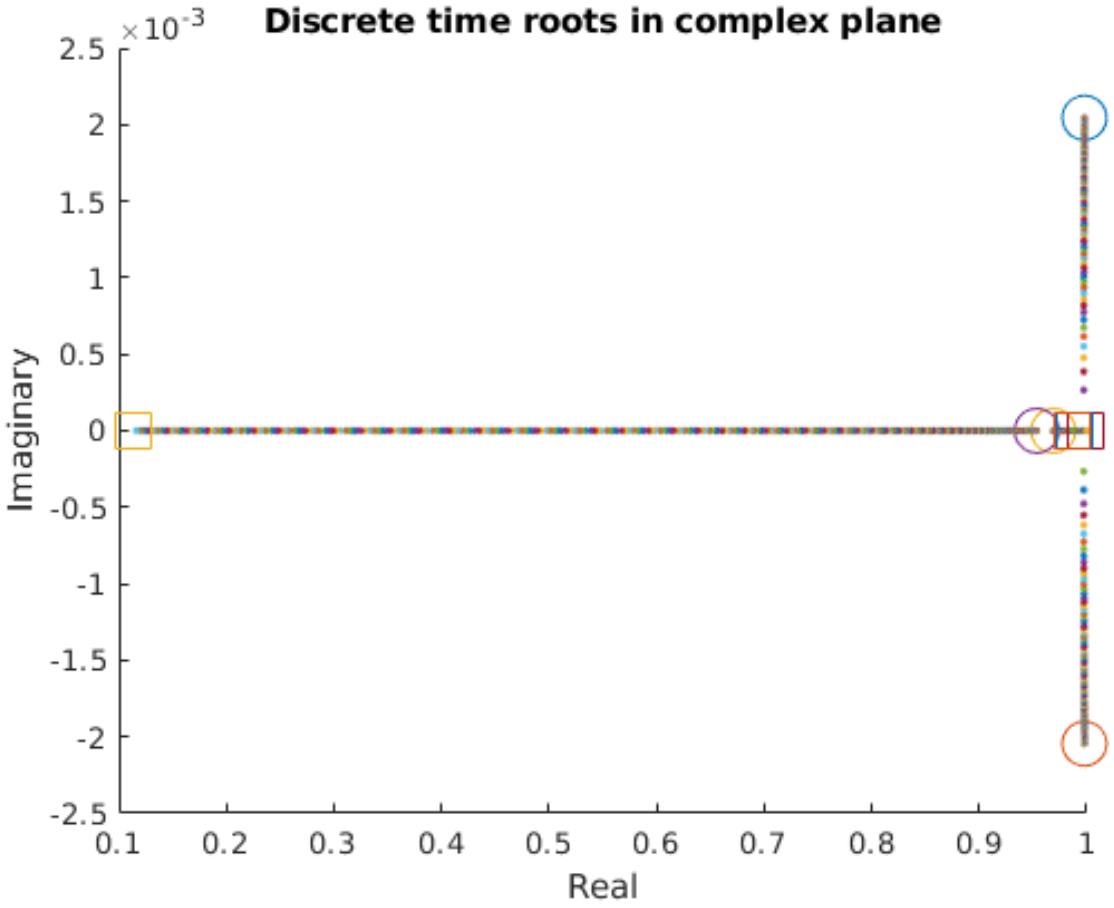


Figure 4: Varying Q(3,3), starting from Q=diag(1,1,1,1), R=1. There is not a lot I can do about those two roots that are close to (1,0).

So I tried $\mathbf{Q}$ is 0.000001*identity-matrix and $\mathbf{R}$ is 0.0001*identity-matrix, and got a more visually pleasing result (Figure 12).

The code to generate these plots is `http://www.cs.cmu.edu/~cga/controls-intro/ass1/sim_kf.m`

D2) Explore two Kalman filter designs. D2a) Generate a Kalman filter design with time constants much faster than your LQR controller. Plot the pole locations for the combined system. D2b) Generate a Kalman filter design with time constants about the same as your LQR controller. Plot the pole locations for the combined system. What is the difference in command response between this KF+LQR combination, and the same LQR design with full state feedback.

It turns out that if the model is perfect the LQR controller and the Kalman filter do not interact, and the poles of the combined system are the combination of the poles of each individual system. This is known as the Separation Principle. It can be best understood by formulating the combined system in terms of the dynamics of the actual state and the estimation error, the difference between

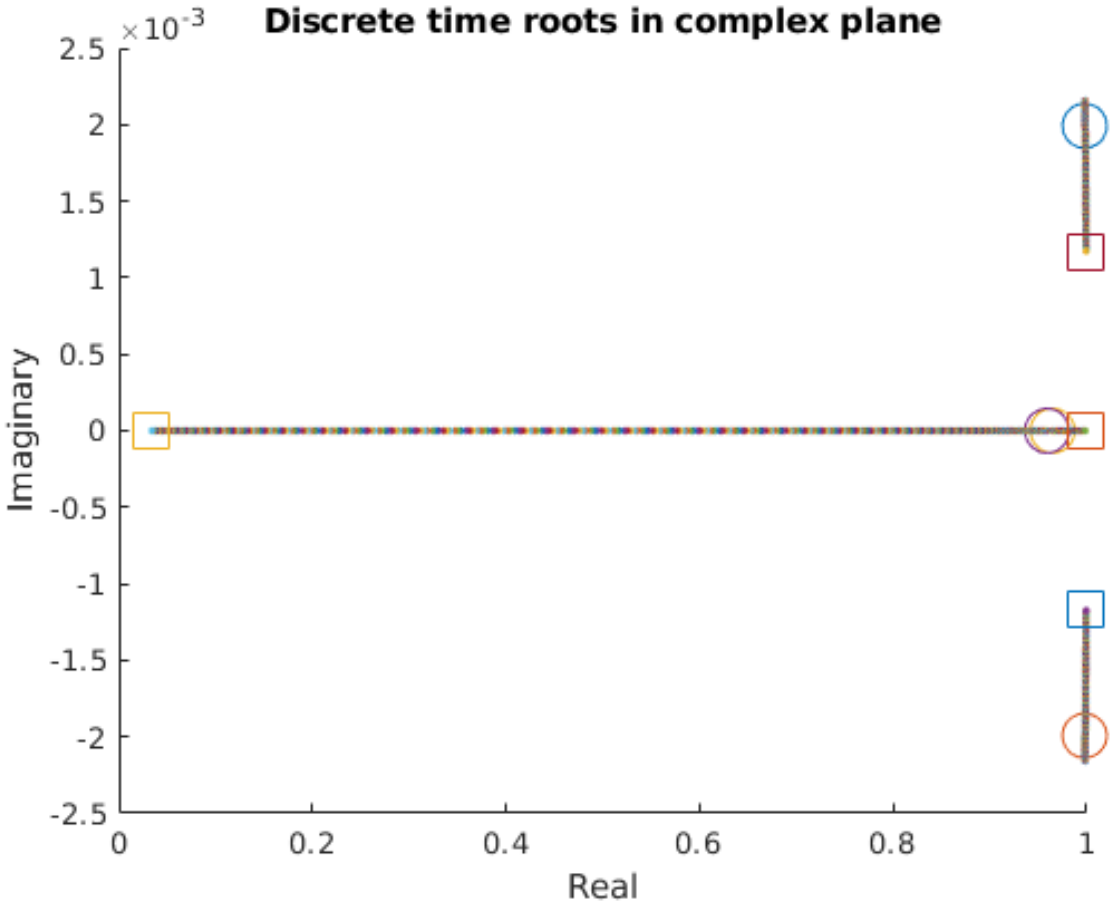


Figure 5: Varying Q(4,4), starting from Q=diag(1,1,1,1), R=1. There is not a lot I can do about those two roots that are close to (1,0).

the belief state and the actual state ($\mathbf{e} = \mathbf{b} - \mathbf{x}$).

$$\begin{aligned} \mathbf{x}_{\text{next}} &= \mathbf{A}\mathbf{x} - \mathbf{B}\mathbf{K}_c\mathbf{b} \\ &= \mathbf{A}\mathbf{x} - \mathbf{B}\mathbf{K}_c(\mathbf{e} + \mathbf{x}) \\ &= (\mathbf{A} - \mathbf{B}\mathbf{K}_c)\mathbf{x} - \mathbf{B}\mathbf{K}_c\mathbf{e} \end{aligned} \tag{13}$$

$$\mathbf{b}_{\text{next}} = (\mathbf{A} - \mathbf{B}\mathbf{K}_c)\mathbf{b} - \mathbf{K}_f(\mathbf{C}\mathbf{b} - \mathbf{C}\mathbf{x}) \tag{14}$$

Subtract $\mathbf{x}_{\text{next}}$ from both sides of the above equation.

$$\begin{aligned} \mathbf{e}_{\text{next}} &= (\mathbf{A} - \mathbf{B}\mathbf{K}_c)\mathbf{e} + \mathbf{B}\mathbf{K}_c\mathbf{e} - \mathbf{K}_f\mathbf{C}\mathbf{e} \\ &= \mathbf{A}\mathbf{e} - \mathbf{K}_f\mathbf{C}\mathbf{e} \end{aligned} \tag{15}$$

Let's rewrite this as a matrix equation.

$$\begin{pmatrix} \mathbf{x}_{\text{next}} \\ \mathbf{e}_{\text{next}} \end{pmatrix} = \begin{pmatrix} \mathbf{A} - \mathbf{B}\mathbf{K}_c & -\mathbf{B}\mathbf{K}_c \\ 0 & \mathbf{A} - \mathbf{K}_f\mathbf{C} \end{pmatrix} \begin{pmatrix} \mathbf{x} \\ \mathbf{e} \end{pmatrix} \tag{16}$$

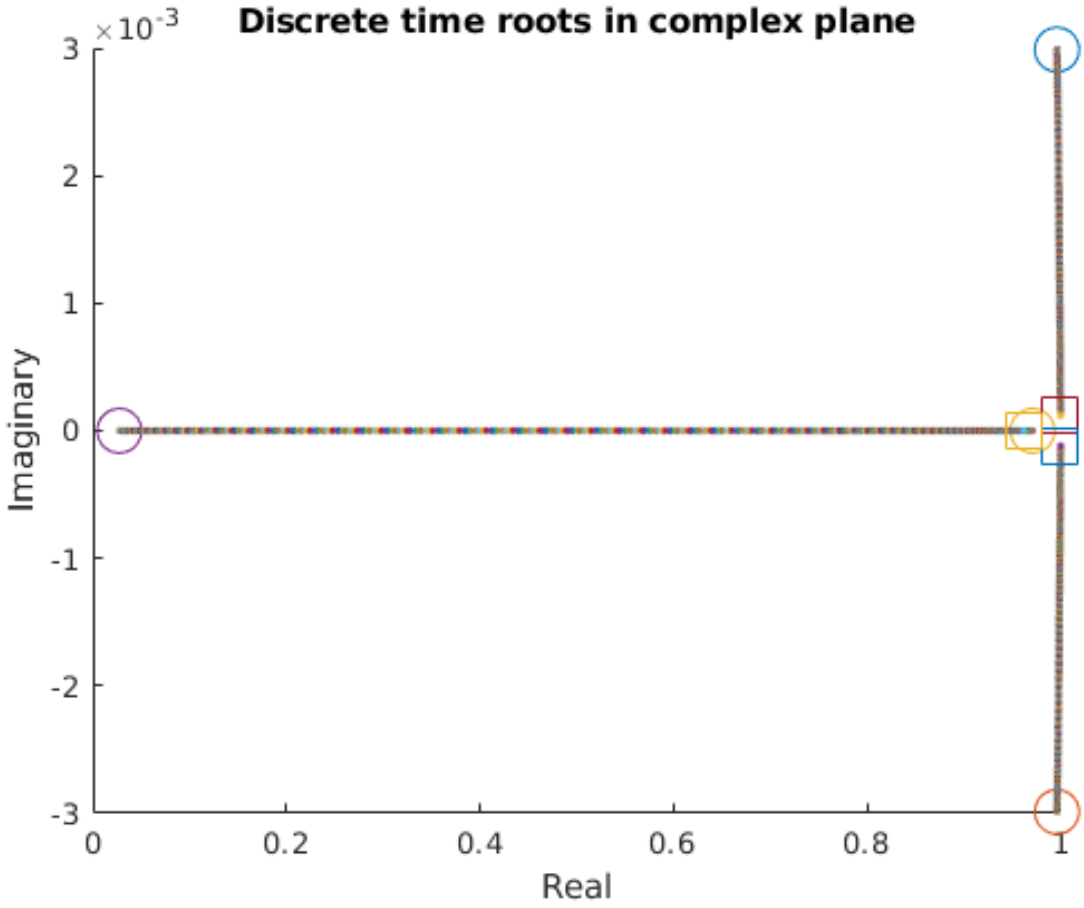


Figure 6: Varying R(1,1), starting from Q=diag(1,10,1,1), R=0.01. There is not a lot I can do about those two roots that are close to (1,0).

Because the left bottom block of the matrix is zero, the characteristic equation for this matrix is $(\mathbf{A} - \mathbf{B}\mathbf{K}_c)(\mathbf{A} - \mathbf{K}_f\mathbf{C})$, which has the combination of the roots from the LQR controller and the Kalman Filter. However, when the model is not perfect, the LQR controller and Kalman Filter can interact badly if the poles are nearby. One rule of thumb is to have estimator poles three or more times faster than controller poles.

`https://ethz.ch/content/dam/ethz/special-interest/mavt/dynamic-systems-n-idsc-dam/Lectures/Digital-Control-Systems/Slides_DigReg_2013.pdf` discusses this in more detail.

E. Robustness

I have not yet answered any of these questions.

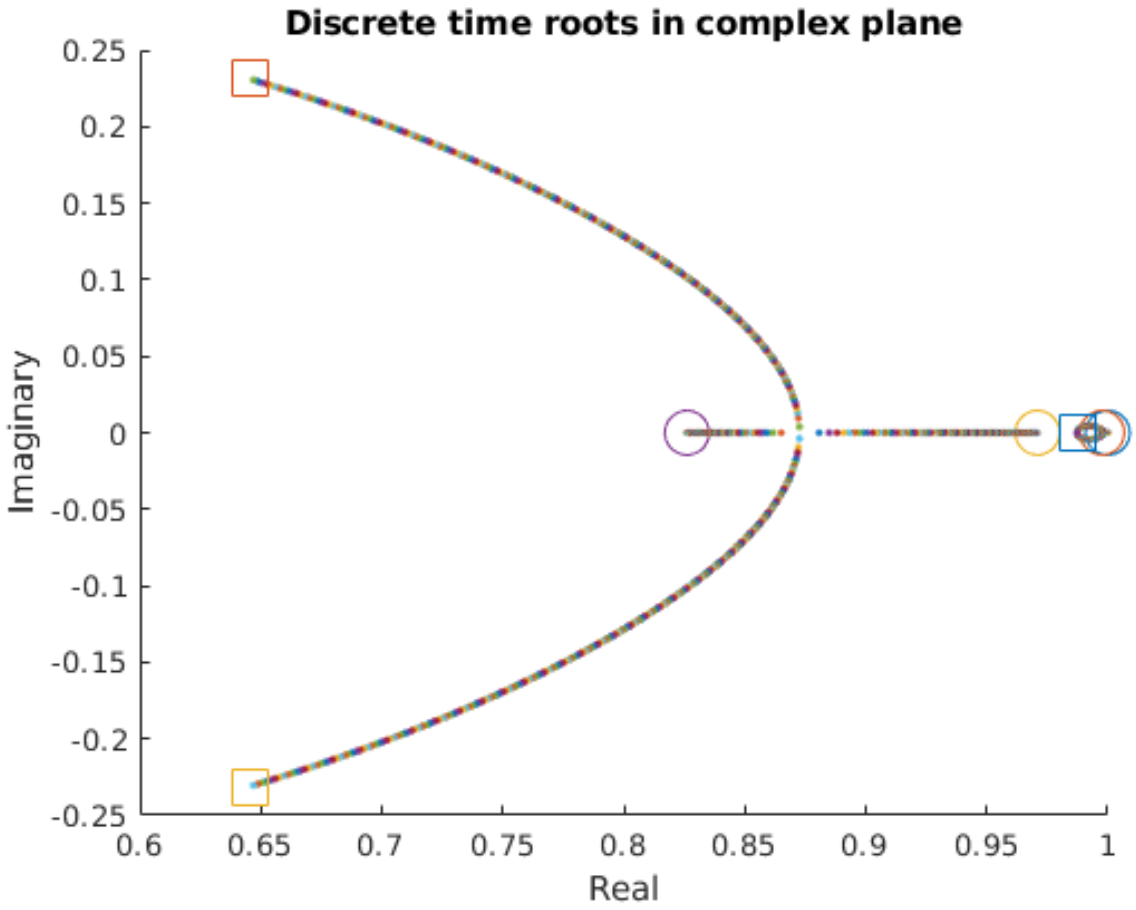


Figure 7: Varying Q(1,1), starting from Q=diag(1,10,1,1), R=0.01. There is not a lot I can do about those two roots that are close to (1,0).

F. Other approaches and designs

I have not yet answered any of these questions.

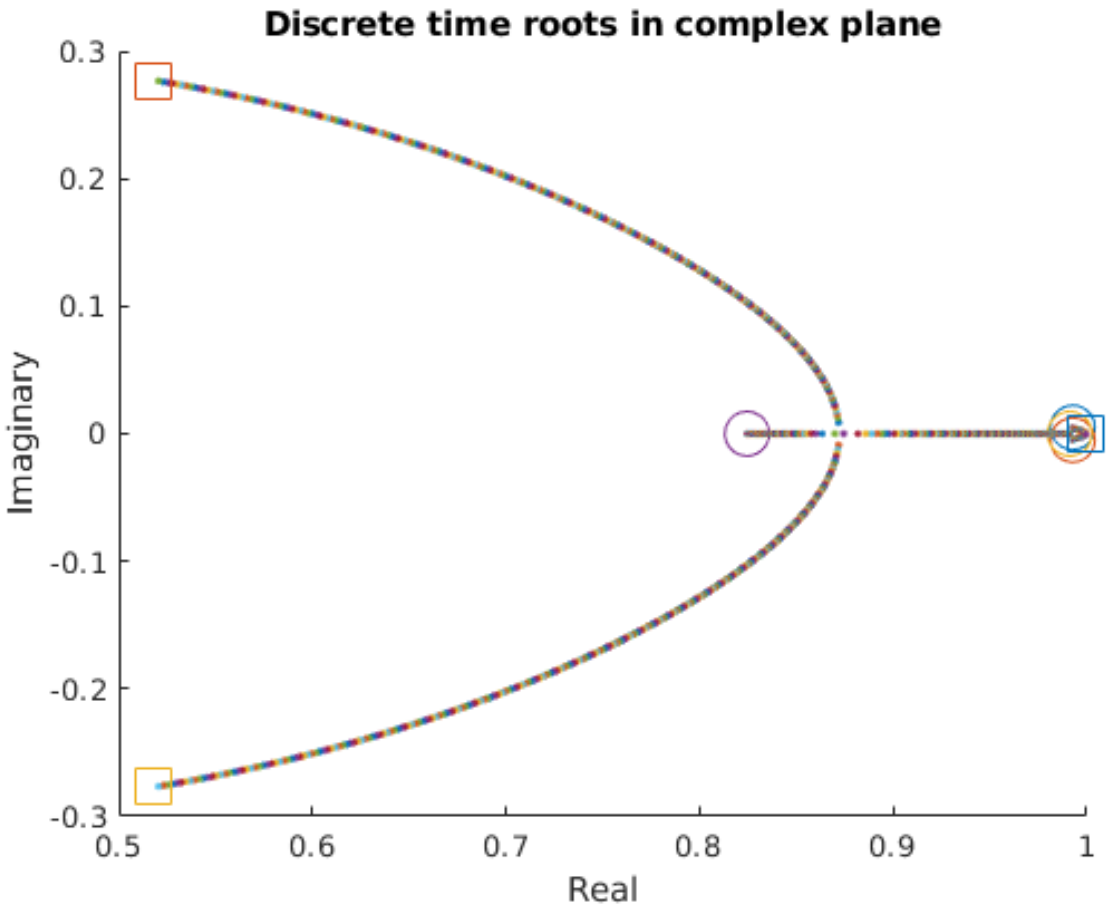


Figure 8: Varying Q(2,2), starting from Q=diag(1,10,1,1), R=0.01. There is not a lot I can do about those two roots that are close to (1,0).

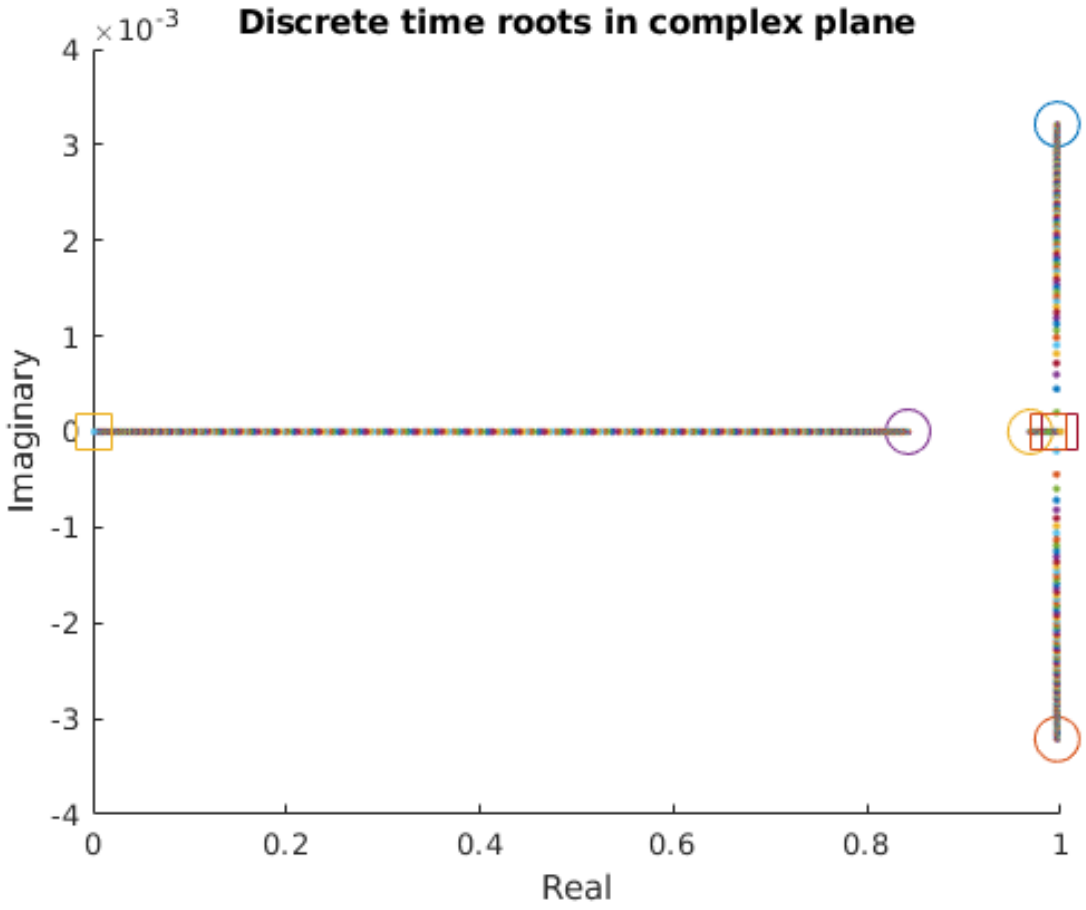


Figure 9: Varying Q(3,3), starting from Q=diag(1,10,1,1), R=0.01. There is not a lot I can do about those two roots that are close to (1,0).

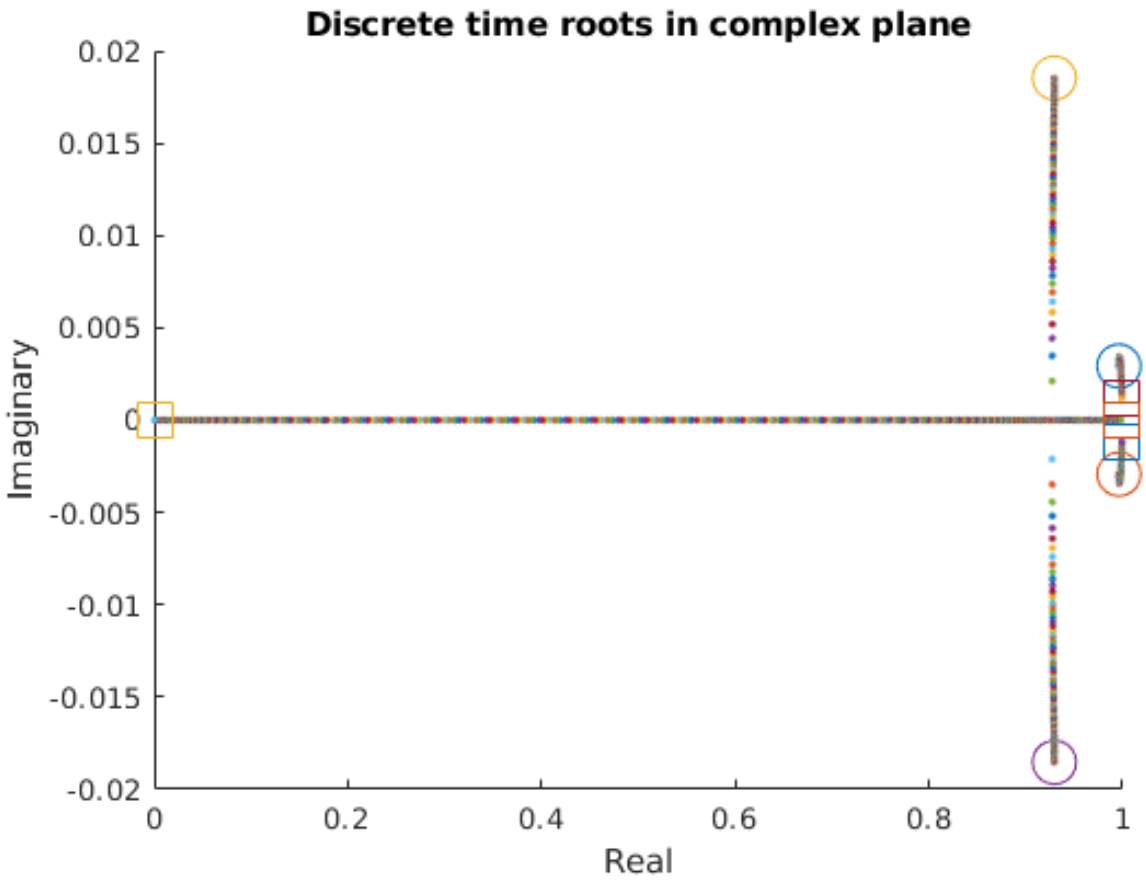


Figure 10: Varying Q(4,4), starting from Q=diag(1,10,1,1), R=0.01. There is not a lot I can do about those two roots that are close to (1,0).

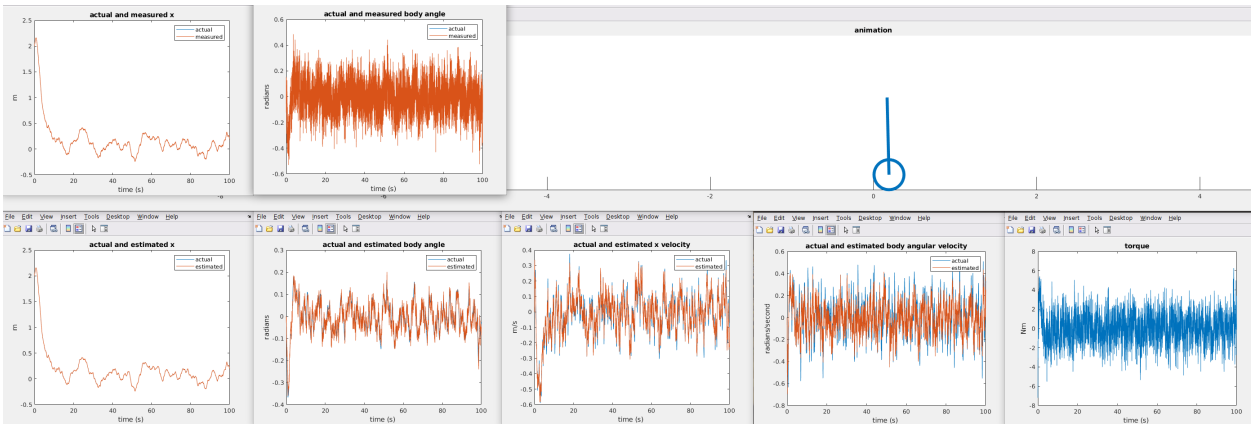


Figure 11: Kalman filter simulation 1

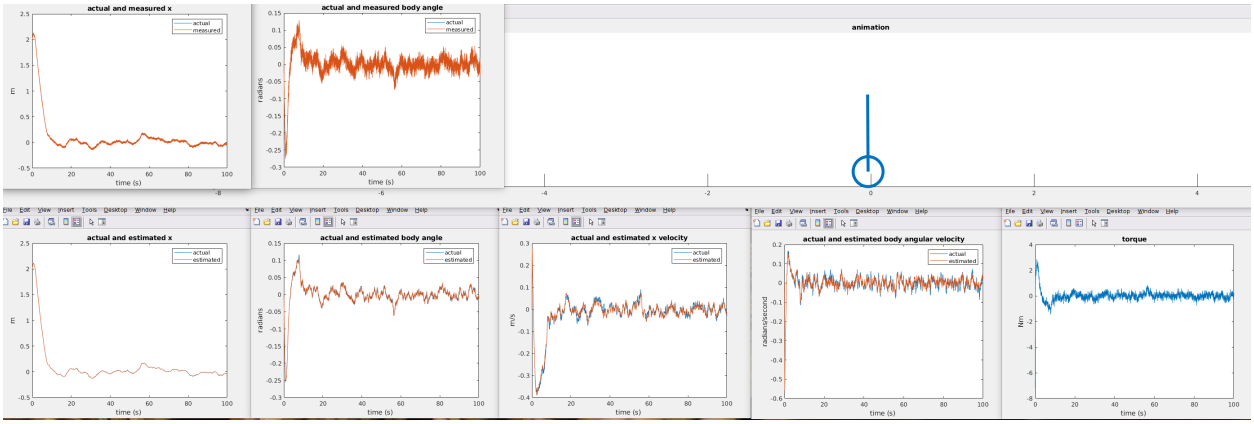


Figure 12: Kalman filter simulation 2