Stabilization and Control of a 2D Quadcopter System

Nathan Isaman
University of Washington
A A 548: Multivariable Control
Winter 2018

I. Introduction

This paper covers the simulation, linearization, and stabilization of a simplified Quadcopter system. Stabilizing Feedback via the Lyapunov equation and Linear Quadratic Regulator (LQR) schemes are investigated with respect to the inherently unstable system. The final step in the analysis involves implementing a Kalman Filter to noisy system measurements.

II. Simulation of the 2D Quadcopter System

A. System Model

System dynamics were modeled using Forward Euler numerical integration over the desired time span. The state of interest was $[q\ \dot{q}]^T$ where q consists of the position values of $[h v \theta]^T$ and $\dot{q}$ are their derivatives. The state components q and $\dot{q}$ were solved for using Forward Euler and a valid time-step, h:

$$q_+ = q + h\dot{q}_- + h^2\ddot{q}_-$$
$$\dot{q}_+ = \dot{q} + h\ddot{q}$$

where

$$\ddot{q} = \begin{bmatrix} \frac{u_1}{m} sin(\theta) \\ \frac{u_1}{m} cos(\theta) - g \\ \frac{u_2}{I} \end{bmatrix}$$

B. Equilibrium Point

The equilibrium when $u = [mg\ 0]^T$ is $[q\ \dot{q}]^T = [h_0\ v_0\ \theta_0\ 0\ 0\ 0]^T$ due to the mg thrust vector countering the gravitational force acting on the quadcopter. This equilibrium state represents the quadcopter in a "hover" mode at its initial location.

C. Sinusoidal Input

The system's dynamics were simulated for a sinusoidal input

$$u(t) = \begin{bmatrix} mg + sin(2t\pi\omega) \\ 0 \end{bmatrix}$$

Simulation results for $\omega = [\frac{\pi}{2}\ \pi\ 2\pi]$ can be seen in Figures 1-3. The horizontal and rotational positions do not deviate from the initial condition, but the vertical position climbs in an oscillatory manner with respect to the sinusoidal input. The frequency of the input has an interesting impact on the ascent behavior of the quadcopter, making the climb in either a smoother trajectory or a more "wavy" one depending on ω. All final times were kept at a constant 2π so that the comparison of the behavior can be done over the same time interval.

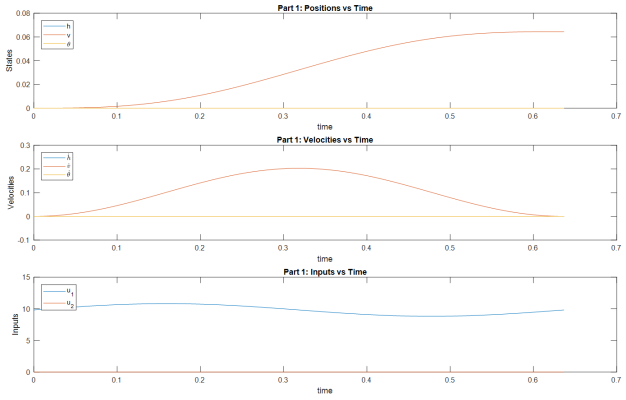


Fig. 1. System Dynamics with an input frequency of $\omega = \frac{\pi}{2}$

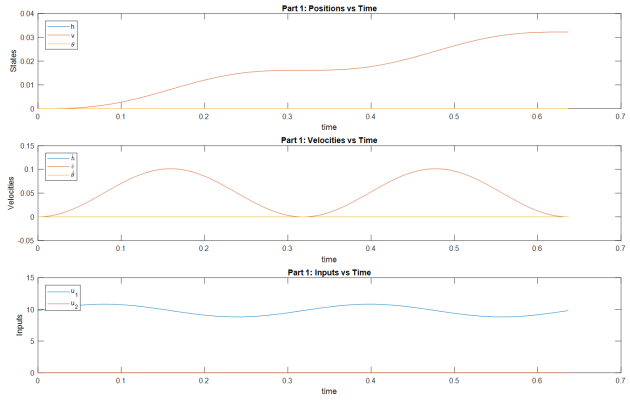


Fig. 2. System Dynamics with an input frequency of $\omega = \pi$

III. Stabilization of the System

A. Linearization

The quadcopter system was linearized around the equilibrium state and input seen below prior to stabilization.

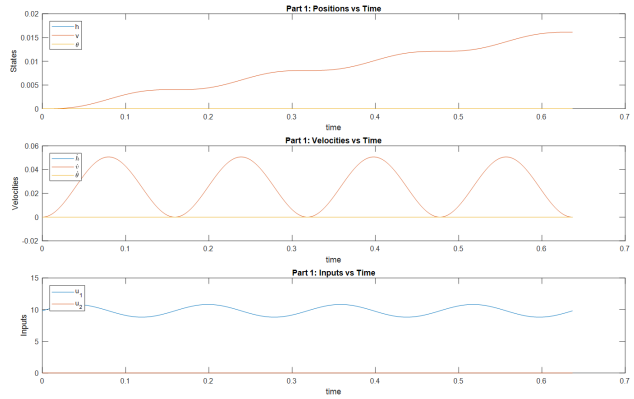


Fig. 3. System Dynamics with an input frequency of $\omega = 2\pi$

$$\begin{bmatrix} q \\ \dot{q} \end{bmatrix} = \begin{bmatrix} 0 \\ 0.1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad u = \begin{bmatrix} mg \\ 0 \end{bmatrix}$$

Linearizing the system was done as follows [1]:

$$\tilde{A} = \frac{\partial f((\tilde{q},\tilde{\dot{q}}),\tilde{u})}{\partial x}$$

and

$$\tilde{B} = \frac{\partial f((\tilde{q},\tilde{\dot{q}}),\tilde{u})}{\partial u}$$

where

$$f((q,\dot{q}),u)) = \frac{d}{dt}\begin{bmatrix} q \\ \dot{q} \end{bmatrix} = \begin{bmatrix} \dot{h} \\ \dot{v} \\ \dot{\theta} \\ \frac{u_1}{m} sin(\theta) \\ \frac{u_1}{m} cos(\theta) - g \\ \frac{u_2}{I} \end{bmatrix}$$

The result of of taking the partial derivatives of $f((q,\dot{q}),u)$ results in the following Jacobian matrices.

$$\tilde{A} = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & \frac{u_1}{m} cos(\theta) & 0 & 0 \\ 0 & 0 & 0 & \frac{-u_1}{m} sin(\theta) & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\tilde{B} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ \frac{sin(\theta)}{m} & 0 \\ \frac{cos(\theta)}{m} & 0 \\ 0 & \frac{1}{I} \end{bmatrix}$$

These Jacobian matrices are then evaluated at the equilibrium point $((\tilde{q},\tilde{\dot{q}}),\tilde{u})$ to arrive at the following linearized matrices.

$$\tilde{A} = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & g & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\tilde{B} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ \frac{1}{m} & 0 \\ 0 & \frac{1}{I} \end{bmatrix}$$

B. Controllability and Stabilization of the System

It is reasonable to linearize about the specified point because it is a stationary point. The quadcopter will maintain its initial condition at these state and input values unless perturbed. This linearized system is controllable as a result of the Controllability Matrix being full rank, therefore satisfying the following theorem.

$$rank(\mathscr{C}) = dim(\mathscr{C}) \iff System\ is\ Controllable$$

where

$$\mathscr{C} = \begin{bmatrix} \tilde{B} & \tilde{A}\tilde{B} & \tilde{A}^2\tilde{B} & \dots & \tilde{A}^5\tilde{B} \end{bmatrix}$$

A stabilizing gain matrix is required for this system as all of the eigenvalues of the $\tilde{A}$ matrix do not have negative real components[See MATLAB Appendix]. In general, the eigenvalues are pure imaginary numbers. A stabilizing gain matrix was created using a modified version of the Lyapunov Stability equation seen below. We must first verify that stabilizing the system via pole placement will not make the system uncontrollable.

$$PA + A^T P + Q = 0$$

Claim: Since $(\tilde{A},\tilde{B})$ is controllable, then so is $(-\lambda I - A, B)$ for every $\lambda \in \mathbb{R}$.

Proof. Suppose (A,B) is controllable. Let ϕ be the eigenvector associated with x such that $Ax = \phi x$. Then

$$(-\lambda I - A)x = -\lambda x - \phi x = (-\lambda - \phi)x$$

This means x is an eigenvector pair for ϕ as well as for $(-\lambda - \phi)$. Since x is an eigenvector for the controllable (A,B), it must be that $(-\lambda I - A$,B) is also controllable by the eigenvector test [1]. □

Since, by the proof above, $(-I - A, B)$ is controllable, we can choose a λ sufficiently large such that our system will be stable as well as controllable. At the end of the stabilization process, we will arrive at a closed-loop system with state feedback: $\dot{x} = (A - BK)x$ where K is the stabilizing gain matrix we are about to find via the Lyapunov equation. Assuming we can find such a K, the Lyapunov equation can be modified as follow:

$$P(A - BK) + (A - BK)^T P + 2\lambda P = 0$$

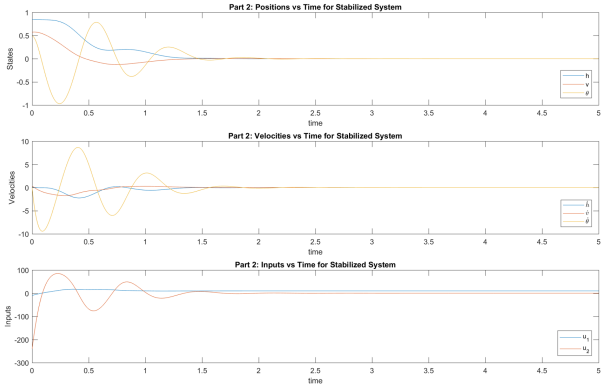


Fig. 4. Feedback Response with Custom Lyapunov Solver

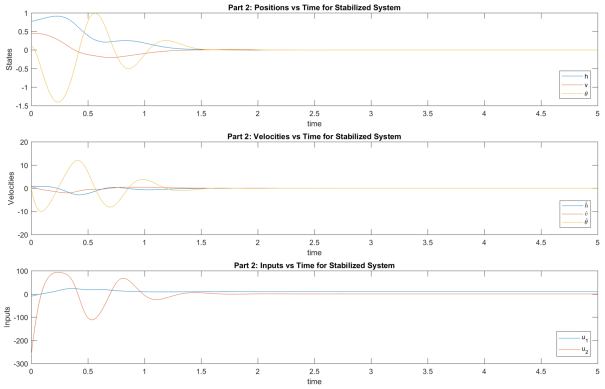


Fig. 5. Feedback Response with Built-in Lyapunov Solver

where $K = \frac{1}{2}B^T P$

$$PA + A^T P - PBK - K^T B^T P + 2\lambda P = 0$$
$$PA + A^T P - PBB^T P + 2\lambda P = 0$$
$$P(I + A) + (I + A)^T P - PBB^T P = 0$$
$$P(-\lambda I - A) + (-\lambda I - A)^T P + PBB^T P = 0$$

multiplying by P^{-1} on the left and right yields the following

$$(-\lambda I - A)P^{-1} + P^{-1}(-\lambda I - A)^T + BB^T = 0$$
$$(-\lambda I - A)W + W(-\lambda I - A)^T + BB^T = 0$$

The final result above is a modified version of the Lyapunov equation and solving for W will lead to the suitable stabilizing gain matrix for the chosen λ such that $K = \frac{1}{2}B^T W^{-1}$. A custom function was created to numerically solve the Lyapunov equation and, despite the poor convergence rate, a stabilizing gain matrix was found. See Figure 4 for stabilizing feedback using the custom Lyapunov solver and Figure 5 for the built-in solver. Both cases provide stabilizing feedback in the presence of randomized initial state perturbations. Use of built-in MATLAB functions, such as gram() or lyap() result in shorter settling times and less overshoot in the system response compared to the custom solver used for the stated Figures. Generally, it is recommended to use these built-in functions for more ideal results, as can be seen when comparing Figures 4 and 5.

The graphs clearly illustrate the system returning to equilibrium conditions of $[q \;\; \dot{q}]^T = 0$ and inspection of the eigenvalues of the closed loop system show that $\forall \lambda_i \in \sigma(A - BK)$, $\mathscr{R}\{\lambda_i\} < 0$ as desired for a stable system [See MATLAB Appendix]. Several randomized perturbations of the initial states were tested and the same results were achieved in all cases.

IV. Continuous Time LQR Design

A. LQR Solution Foundations

The stabilizing feedback gain matrix found in the previous section does a decent job of stabilizing the system, but more can be done. In order to achieve optimal control, we now move onto LQR design. A function to generate random Positive Semi-Definite (PSD) matrices was created to facilitate solving the LQR problem. The code for PSD_RandMat can be found in the MATLAB portion of the Appendix. The LQR gain matrix was solved for using Forward Euler to numerically backward-in-time integrate the Riccati Differential Equation found below. This numerical integration was implemented by the function CT_Riccati_FwdEuler2 [See MATLAB Appendix].

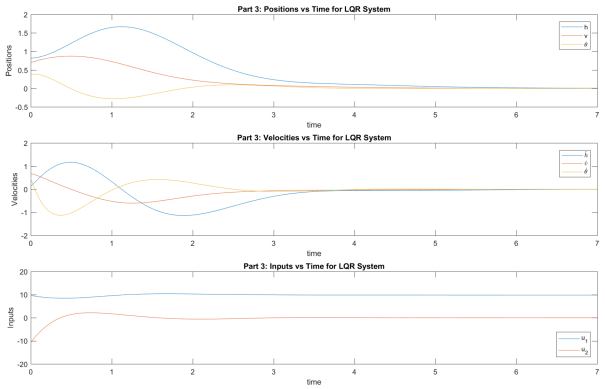


Fig. 6. System Response under LQR

$$-\dot{P} = A^T P_t + P_t A - P_t B R^{-1} B^T P_t + Q$$

The above differential equation rapidly converges to a steady-state value P_{ss} when the iteration reaches a sufficient distance from the time horizon. This steady-state P_{ss} was subsequently used to determine the optimal LQR gain matrix via the equation below.

$$K_{LQR} = -R^{-1} B^T P_{ss}$$

B. LQR Applied to Quadcopter System

The tools developed in the preceding section were applied to the Quadcopter system in order to determine the LQR gain matrix that will optimally stabilize the system.

C. Comparison of LQR with Stabilizing Feedback

The system response under the current LQR implementation does not converge to steady-state equilibrium as rapidly as it

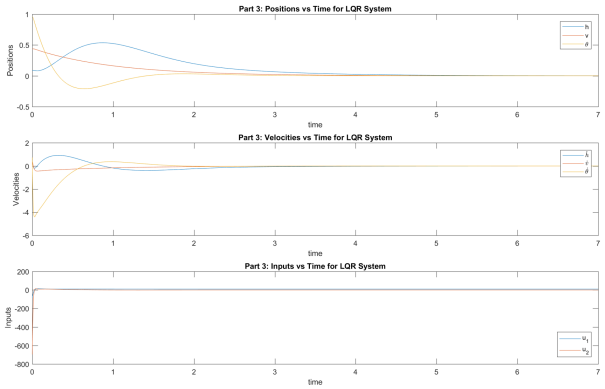


Fig. 7. System Response under Modified LQR

does with stabilizing feedback. Stabilizing feedback reaches steady-state within 1.5-2.0 seconds, whereas the current LQR reaches steady-state around 5-6 seconds. Quick comparison of the control effort reveals the source of this difference. The stabilizing feedback requires -300 to 200 units of torque and LQR only requires -10 to 2 units of torque to stabilize the system in their respective timelines.

D. Adjusting Q and R

Adjusting the Q and R matrices can bring the performance of LQR closer to the stabilizing feedback. Since Q represents the weight of the state, increasing the magnitude of Q will make the state a more important cost consideration. The opposite can be done for R, where decreasing its magnitude will lower the cost associated with the control input, thereby allowing a much higher control input in order to drive the system towards steady-state.

Increasing the diagonal of Q by 5000 and reducing R to 0.5I results in a system similar to the stabilizing feedback where the system converges to steady-state in roughly 3 seconds and utilizes a significantly higher control input magnitude. As seen in Figure 7, the proposed changes to Q and R have resulted in the desired outcome. If input cost were more important than speedy convergence, the opposite operation could be preformed.

V. Implementation of a Kalman Filter on the Closed Loop System

A. Discretization of the System

Discretization of the system was performed after closing the loop as the open-loop A matrix is singular; discretization of the system requires a non-singular A. The LQR gain matrix was utilized for feedback in closing the loop and the following system was used in the discretization.

$$A_{CL} = (A - BK_{LQR})$$
$$\dot{x} = A_{CL}x + Bu_{eq}$$

where u_{eq} is the equilibrium input $[mg\ 0]^T$.
This closed-loop system was discretized using the following equations.

$$A_d = e^{A_{cl}\tau}$$
$$B_d = A_{cl}^{-1}(A_d - I)B$$

where τ is the discretization time-step or desired sample time. These discretized matrices were then used to represent the system in discrete time with the addition of process and measurement noise.

$$x_{k+1} = A_d x_k + F_k w_k$$
$$y_k = C_d x_k + H_k v_k$$

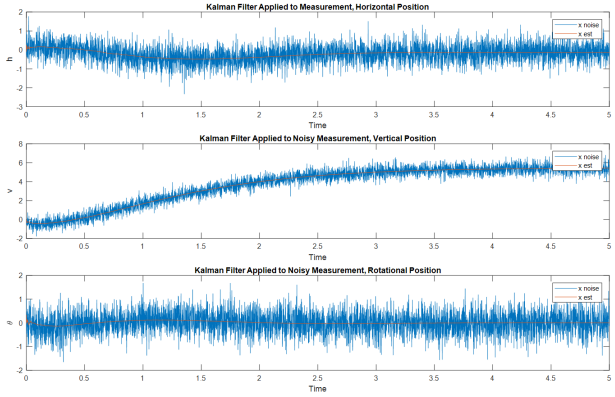


Fig. 8. Kalman Filter Implemented on Noisy, Discretized State and Measurement

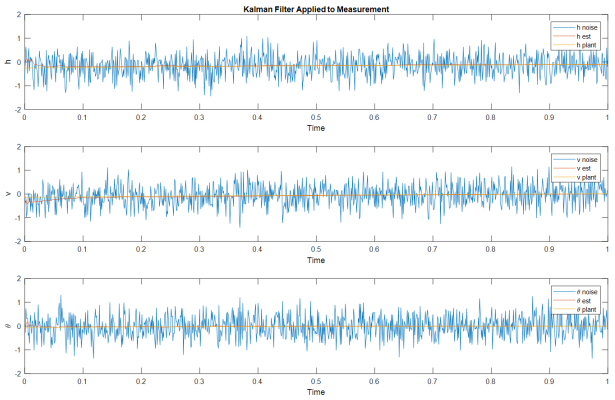


Fig. 9. Comparison of Measurement, Noisy Measurement, and Estimated Measurement

The F_k and H_k matrices represent the noise dynamics and w_k, v_k represent the random process and measurement noise respectively at time-step k. The latter two terms are random variables whereas the former two matrices are assumed constant and at set values. F_k is assumed equal to B_d and H_k is assumed to be an identity matrix. The following are the other initial assumptions regarding the mean and covariance of the state and noise components.

$$\mathbb{E}[w_t w_t^T] = W$$
$$\mathbb{E}[v_t v_t^T] = V$$
$$\mathbb{E}[x_0] = \bar{x}_0$$
$$\mathbb{E}[(x_0 - \bar{x}_0)(x_0 - \bar{x}_0)^T] = \Sigma_0$$

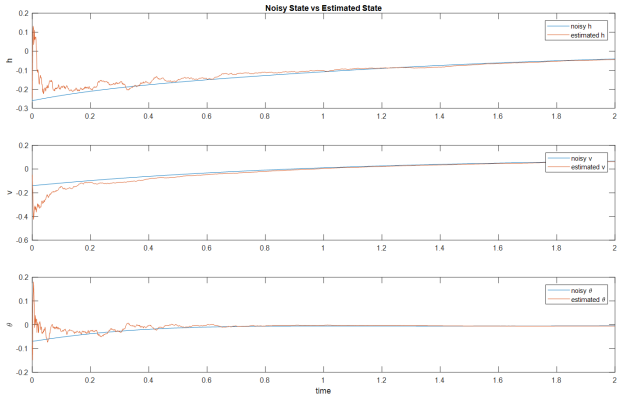


Fig. 10. Comparison of Noisy State and Estimated State

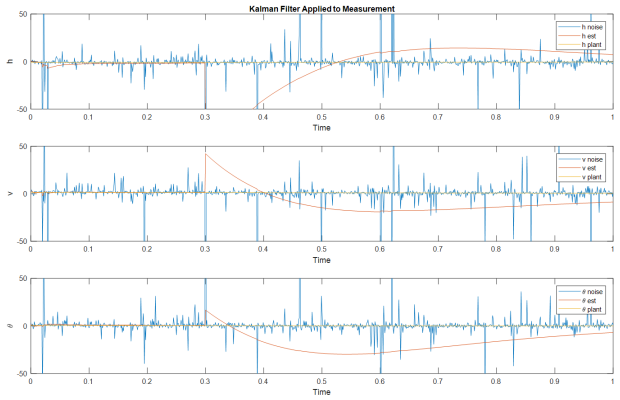


Fig. 11. Comparison of Measurement, Noisy Measurement, and Estimated Measurement; Alternate Distributions

$$v_t \sim \mathcal{N}(0, 0.2I_{2x2})$$
$$w_t \sim \mathcal{N}(0, 0.1I_{2x2})$$
$$\Sigma_0 = 0.1I_{6x6}$$
$$F_t = B_d$$
$$H_t = I$$

Application of the Kalman Filter on normally distributed noise and the purely diagonal covariances results in reliable and rapid convergence of the estimated state to the actual state, as seen in Figures 8 and 10. The implemented Kalman Filter was then tested on alternative noise distributions with covariances that are not strictly diagonal. The following adjustments were made for the results seen in Figures 11 and 12.

$$\Sigma_0 \succeq 0$$
$$v_t \sim \mathcal{T}(\nu = 1)$$
$$w_t \sim \mathcal{T}(\nu = 1)$$

Using the Student t-distribution with $\nu = 1$ (degrees of freedom) to generate the process and measurement noise had a clear effect on the resulting noisy measurement and the Kalman Filter's ability to generate good estimates of the state. The estimate converges to the actual state until an outlier noisy measurement is received. This outlier drastically shifts the estimated measurement and estimated state away from the actual measurement and state. The filter then proceeds to return the estimates back to the actual values as expected. This specific filter was not designed to work on this type of noise distribution, but it still does a decent job returning to the actual state. Other tests with normally distributed noise and covariances resulted in similar results to the initial test, which inspired this alternative (and more interesting) test. A different filter formulation is probably required to tackle t-distributed noise, perhaps one that is less sensitive to outliers.

VI. Conclusion

The simplified 2D Quadcopter system used in this project presented an exciting opportunity to implement controllability, stabilization feedback, LQR, and Kalman Filtering concepts and processes. Creating a basic simulation of the non-linear system allowed for valuable insight later in the project where feedback effectiveness needed to be analyzed. Linearization of the non-linear system about a "hovering" equilibrium state allowed for investigation of the stability of the system. After seeing that the system was not inherently stable, stabilization schemes were investigated and implemented on the system. Results from these implementations were verified using the model simulation developed at the beginning of the project. The stabilization feedback gain calculated using a Lyapunov equation resulted in rapid convergence to steady-state equilibrium, but at a large input cost. The LQR method resulted in an initially slower steady-state convergence rate, but modifying the underlying Q and R matrices allowed for some tunability of the settling time and allowed input effort. Finally, measurement and process noise were introduced to a discretized version of the linearized system model and a Kalman Filter was applied to verify that the actual state could be reliably estimated. Results from this section showed that normally distributed noise can be reliably filtered out to arrive at a good estimation of the state, whereas alternative distributions result in less reliable results.

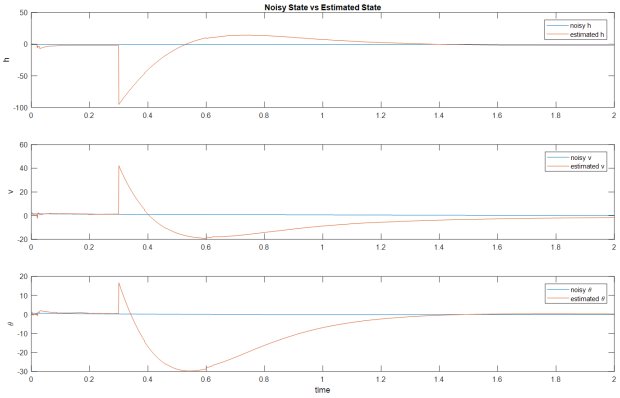


Fig. 12. Comparison of Noisy State and Estimated State; Alternate Distributions

REFERENCES

[1] Hespanha, J. (2009). Linear systems theory. Princeton University Press.

AA548 Project - Nathan Isaman (20180309)

```
clear all; close all; clc

% ========================================================================
% ================ Part 1: Modeling and Simulation ======================
% ========================================================================

% Subpart a: Implement ODE control system model
m = 1;                                              %kg
g = 9.81;                                           %m/s^2
I = 1;                                              %kg/m^2

t = 0.0001;                                         %time step param
timespan = 0:t:(2/pi);                              %time horizon vector
u = [m*g, 0];
omega = 2*pi;
 for o = 1:length(timespan)
     ut(:,o) = [m*g + sin(2*timespan(o)*pi*omega); 0];
 end

% for r = 1:length(timespan)
%     ut(:,r) = [-m*g;0];
% end

%Simulating the system dynamics
q(:,1) = [0;0;0];                                   %initial condition (origin)
qdot(:,1) = [0;0;0];                                %and at rest
qdot(:,2) = f(t,ut(:,1),q(:,1),qdot(:,1),m,I,g);
q(:,2) = h(t,ut(:,2),[0;0;0],[0;0;0],m,I,g);

for j = 3:length(timespan)
    qdot(:,j) = f(t,ut(:,j),q(:,j-1),qdot(:,j-1),m,I,g);

    q_dotdot = qdotdot(ut(:,j),q(:,j-2),m,I,g);
    q(:,j) = h(t,ut(:,j),q(:,j-1),qdot(:,j-2),q_dotdot,m,I,g);

end

figure
subplot(3,1,1)
plot(timespan,q(1,:),timespan,q(2,:),timespan,q(3,:))
xlabel('time')
ylabel('States')
legend('h','v','\theta','Location','NorthWest')
title('Part 1: Positions vs Time')

subplot(3,1,2)
plot(timespan,qdot(1,:),timespan,qdot(2,:),timespan,qdot(3,:))
xlabel('time')
ylabel('Velocities')
leg2 =legend('$\dot{h}$','$\dot{v}$','$\dot{\theta}$','Location','NorthWest');
set(leg2, 'Interpreter', 'latex');
title('Part 1: Velocities vs Time')
```

```
subplot(3,1,3)
plot(timespan,ut(1,:),timespan,ut(2,:))
xlabel('time')
ylabel('Inputs')
legend('u_1','u_2','Location','NorthWest')
title('Part 1: Inputs vs Time')

% ========================================================================
% ================== Part 2: Stabilizing the System ======================
% ========================================================================

%Equilibrium point to stabilize about
eqQ = [0; 0.1; 0; 0; 0; 0];
eqU = [m*g; 0];

%Jacobian wrt state
A =     [0 0 0 1 0 0;
         0 0 0 0 1 0;
         0 0 0 0 0 1;
         0 0 eqU(1)*(1/m)*cos(eqQ(3)) 0 0 0;
         0 0 -eqU(1)*(1/m)*sin(eqQ(3)) 0 0 0;
         0 0 0 0 0 0];

%Jacobian wrt input
B     = [0 0;
         0 0;
         0 0;
         sin(eqQ(3))/m 0;
         cos(eqQ(3))/m 0;
         0 1/I];

% Part 2b -- Testing controllability
ctrlMat = [B, A*B, A*A*B, A*A*A*B, A*A*A*A*B, A*A*A*A*A*B, A*A*A*A*A*B];
fprintf('\n\nPart 2b: Testing controllability of linearized system')
if rank(ctrlMat) == size(ctrlMat,1)
    fprintf('\n\tControllability Matrix is full rank => System is Controllable')
else
    fprintf('\n\tControllability Matrix is not full rank => System is not Controllable')
end

% Part 2c -- Design a stabilizing feedback control law using Lyapunov
lambda = 4;
Ac = (-lambda*eye(size(A)) - A);
fprintf('\n\nProblem 2c: Stabilizing Feedback Control')
fprintf('\n\tTesting Stability of -lambda*I - A matrix')
eig_of_Astab = eig(Ac)

Bc = B;
Cc = [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 0 0 0];
% Cc = [1 0 0 0 0 0;
%       0 1 0 0 0 0];
```

```
Dc = zeros(size(Bc));

ctrlSys = ss(Ac,Bc,Cc,0);
Wc = gram(ctrlSys,'c');
%Wc = teddyGram(Ac,Bc,0.01);
%Wcf - Wc;

K = 0.5*transpose(Bc)*inv(Wc);
Acl = (Ac - Bc*K);

fprintf('\n\nPart 2d: Stabilizing the System with a Feedback gain matrix')
fprintf('\n\tClosed Loop system is stable if all eigenvalues have negative real components
')
eig_of_Acl = eig(Acl)

% Part D: Simulate the Nonlinear CL System using the K matrix found above
%         for a random initialization in the neighborhood of eqQ.

t = 0.0001;                                         %time step param
timespan = 0:t:5;                                   %time horizon vector

%Simulating the system dynamics
q_del = rand(3,1);                                  %Random perturbations
qdot_del = rand(3,1);

q2(:,1) = [0;0.1;0];                                %IC for lin pt
q2dot(:,1) = [0;0;0];                               %and at rest

q2(:,1) = q2(:,1) + q_del;                          %Perturbing the IC
q2dot(:,1) = q2dot(:,1) + qdot_del;                 %Perturbing the IC

utStab(:,1) = eqU + -K*vertcat(q2(:,1),q2dot(:,1));

q2dot(:,2) = f(t,utStab(:,1),q2(:,1),q2dot(:,1),m,I,g);
q2(:,2) = h(t,utStab(:,1),q2(:,1),q2dot(:,1),m,I,g);

utStab(:,2) = eqU + -K*vertcat(q2(:,2),q2dot(:,2));

for j = 3:length(timespan)
    q2dot(:,j) = f(t,utStab(:,j-1),q2(:,j-1),q2dot(:,j-1),m,I,g);
    q2_dotdot = qdotdot(utStab(:,j-1),q2(:,j-2),m,I,g);
    q2(:,j) = h(t,utStab(:,j-1),q2(:,j-1),q2dot(:,j-2),q2_dotdot,m,I,g);

    utStab(:,j) = eqU + -K*vertcat(q2(:,j),q2dot(:,j));
end

figure
subplot(3,1,1)
plot(timespan,q2(1,:),timespan,q2(2,:),timespan,q2(3,:))
xlabel('time')
ylabel('States')
legend('h','v','\theta','Location','SouthEast')
title('Part 2: Positions vs Time for Stabilized System')

subplot(3,1,2)
plot(timespan,q2dot(1,:),timespan,q2dot(2,:),timespan,q2dot(3,:))
xlabel('time')
```

```
ylabel('Velocities')
leg2 =legend('$\dot{h}$','$\dot{v}$','$\dot{\theta}$','Location','SouthEast');
set(leg2, 'Interpreter', 'latex');
title('Part 2: Velocities vs Time for Stabilized System')

subplot(3,1,3)
plot(timespan,utStab(1,:),timespan,utStab(2,:))
xlabel('time')
ylabel('Inputs')
legend('u_1','u_2','Location','SouthEast')
title('Part 2: Inputs vs Time for Stabilized System')

% ========================================================================
% ====================== Part 3: CT LQR Design  ==========================
% ========================================================================

% Part A1: Write a function to create a random PSD matrix
%               --> See PSD_RandMat function for this

% Part A2: Write a numerical scheme to solve LQR.
%               --> See CT_Riccati_FwdEuler2 function for this

T = 7;                          %time horizon
dt = 0.0001;                    %time step
Qalt = transpose(Cc)*Cc;        %alternate Q formulation
Q = PSD_RandMat(size(A,1));     %random Q matrix for cost function
R = PD_RandMat(size(B,2));      %random R matrix for cost function
R = R/norm(R,2);
R = 0.5*eye(size(R));
Q = Q + 5000*eye(size(Q));
%multi-dim array with Pt's
Plqr(:,:,:) = CT_Riccati_FwdEuler2(Q,R,A,B,Q,dt,T);

%Constructing K matrices from the computed P values
time = 0:dt:T;
Rinv = inv(R);
Btr = transpose(B);
for t = 1:length(time)
   Klqr(:,:,t+1) = -Rinv*Btr*Plqr(:,:,t);
end
Klqr_final = Klqr(:,:,end);            %Last value is SS val


%Simulating the LQR Dynamics
t = 0.0001;                                     %time step param
timespan = 0:t:T;                               %time horizon vector

%q_del = rand(3,1);                             %Random perturbations
%qdot_del = rand(3,1);

q3(:,1) = [0;0.1;0];                            %IC for linearized sys
q3dot(:,1) = [0;0;0];                           %and at rest

q3(:,1) = q3(:,1) + q_del;                      %Using previous q_del
q3dot(:,1) = q3dot(:,1) + qdot_del;             %Using previous qdot_del

utLQR(:,1) = eqU + Klqr_final*vertcat(q3(:,1),q3dot(:,1));
```

```
q3dot(:,2) = f(t,utLQR(:,1),q3(:,1),q3dot(:,1),m,I,g);
q3(:,2) = h(t,utLQR(:,1),q3(:,1),q3dot(:,1),m,I,g);

utLQR(:,2) = eqU + Klqr_final*vertcat(q3(:,2),q3dot(:,2));

for j = 3:length(timespan)
    q3dot(:,j) = f(t,utLQR(:,j-1),q3(:,j-1),q3dot(:,j-1),m,I,g);
    q3_dotdot = qdotdot(utLQR(:,j-1),q3(:,j-2),m,I,g);
    q3(:,j) = h(t,utLQR(:,j-1),q3(:,j-1),q3dot(:,j-2),q3_dotdot,m,I,g);

    utLQR(:,j) = eqU + Klqr_final*vertcat(q3(:,j),q3dot(:,j));
end

% MATLAB LQR Soln-----------------------------------------------------------
[Ksolv,Psolv,Esolv] = lqr(A,B,Q,R,0);

qLQR(:,1) = [0;0.1;0];                                  %IC for lin pt
qLQRdot(:,1) = [0;0;0];                                 %and at rest

qLQR(:,1) = qLQR(:,1) + q_del;                          %Perturbing the IC
qLQRdot(:,1) = qLQRdot(:,1) + qdot_del;                 %Perturbing the IC

utLQR_M(:,1) = eqU + -Ksolv*vertcat(qLQR(:,1),qLQRdot(:,1));

qLQRdot(:,2) = f(t,utLQR_M(:,1),qLQR(:,1),qLQRdot(:,1),m,I,g);
qLQR(:,2) = h(t,utLQR_M(:,1),qLQR(:,1),qLQRdot(:,1),m,I,g);

utLQR_M(:,2) = eqU + -Ksolv*vertcat(qLQR(:,2),qLQRdot(:,2));

for j = 3:length(timespan)
    qLQRdot(:,j) = f(t,utLQR_M(:,j-1),qLQR(:,j-1),qLQRdot(:,j-1),m,I,g);
    qLQR_dotdot = qdotdot(utLQR_M(:,j-1),qLQR(:,j-2),m,I,g);
    qLQR(:,j) = h(t,utLQR_M(:,j-1),qLQR(:,j-1),qLQRdot(:,j-2),qLQR_dotdot,m,I,g);

    utLQR_M(:,j) = eqU + -Ksolv*vertcat(qLQR(:,j),qLQRdot(:,j));
end
%-------------------------END of MATLAB LQR Soln Code --------------------

figure
subplot(3,1,1)
plot(timespan,q3(1,:),timespan,q3(2,:),timespan,q3(3,:))
xlabel('time')
ylabel('Positions')
legend('h','v','\theta','Location','NorthEast')
title('Part 3: Positions vs Time for LQR System')
% subplot(2,1,2)
% plot(timespan,qLQR(1,:),timespan,qLQR(2,:),timespan,qLQR(3,:))
% xlabel('time')
% ylabel('States')
% legend('h','v','\theta','Location','NorthEast')
% title('Part 3: Positions vs Time for LQR System (Matlab)')

%figure
subplot(3,1,2)
plot(timespan,q3dot(1,:),timespan,q3dot(2,:),timespan,q3dot(3,:))
xlabel('time')
```

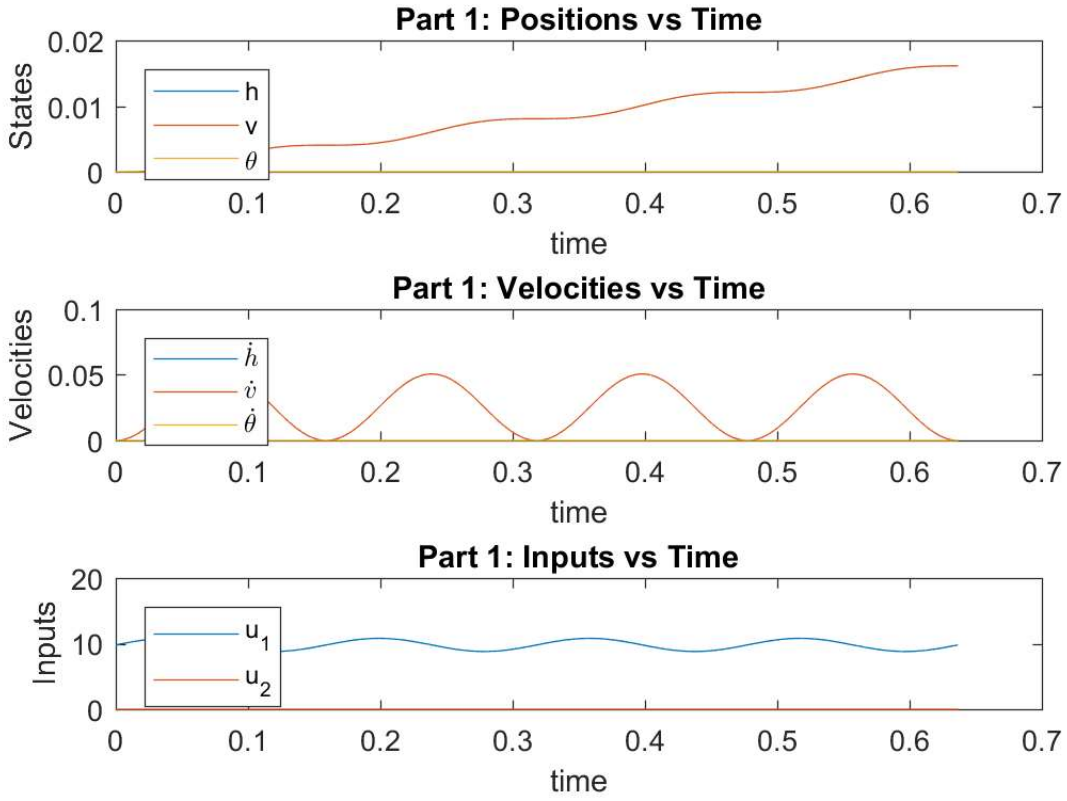
```
ylabel('Velocities')
leg2 =legend('$\dot{h}$','$\dot{v}$','$\dot{\theta}$','Location','NorthEast');
set(leg2, 'Interpreter', 'latex');
title('Part 3: Velocities vs Time for LQR System')
% subplot(2,1,2)
% plot(timespan,qLQRdot(1,:),timespan,qLQRdot(2,:),timespan,qLQRdot(3,:))
% xlabel('time')
% ylabel('Velocities')
% leg2 =legend('$\dot{h}$','$\dot{v}$','$\dot{\theta}$','Location','NorthEast');
% set(leg2, 'Interpreter', 'latex');
% title('Part 3: Velocities vs Time for LQR System (Matlab)')

%figure
subplot(3,1,3)
plot(timespan,utLQR(1,:),timespan,utLQR(2,:))
xlabel('time')
ylabel('Inputs')
legend('u_1','u_2','Location','SouthEast')
title('Part 3: Inputs vs Time for LQR System')
% subplot(2,1,2)
% plot(timespan,utLQR_M(1,:),timespan,utLQR_M(2,:))
% xlabel('time')
% ylabel('Inputs')
% legend('u_1','u_2','Location','SouthEast')
% title('Part 3: Inputs vs Time for LQR System (Matlab)')

% ADD COMPARISON BTW OPT CTRL AND STAB FEEDBACK CTRL HERE

% SHOW COMPARISON BTW VARYING Q AND R HERE

% Use C'C as Q and R = I to get similar perf as StabFeedback ctrl

% ========================================================================
% ===================== Part 4: Kalman Filtering =========================
% ========================================================================

% Part a: Discretization of the System (A-BK)x - Bw --------------------
Akf = (A + B*Klqr_final);                       %CL Sys using LQR gain mat K
del_t = 0.001;                                  %Discretization time step
endTime = 2;

Ad = expm(Akf*del_t);                           %-------------------------
Bd = inv(Akf)*(Ad - eye(size(Ad)))*B;           %|       Discretized     |
Cd = Cc;                                        %|         System        |
Dd = Dc;                                        %-------------------------
Fd = Bd;
Hd = eye(6);

% Part bcd: Noisy system ------------------------------------------------
kftime = 0:del_t:endTime;

%Generating the initial random state;
%sig_0 = 0.1*eye(6);
sig_0 = PSD_RandMat(6);
x0_mu = [0 0 0 0 0 0];
x0 = mvnrnd(x0_mu,sig_0,1)';
```

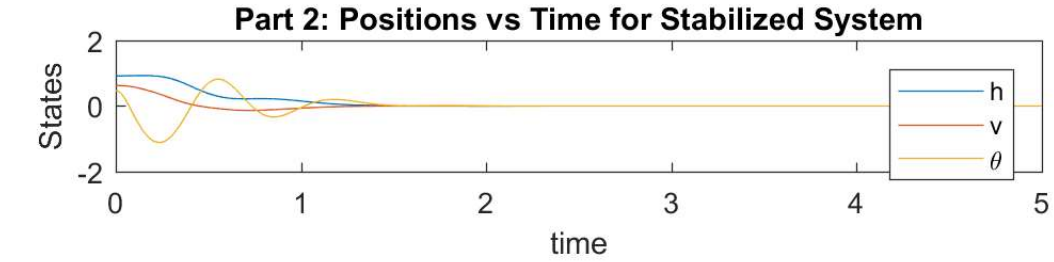
```
%Generating the signal noise
%sig_v = 0.2*eye(6);
sig_v = PD_RandMat(6);
v_mu = [0 0 0 0 0 0];
%vt = mvnrnd(v_mu,sig_v,length(kftime))';
vt = mvtrnd(sig_v,1,length(kftime))';

%Generating the state disturbance
%sig_w = 0.1*eye(2);
sig_w = PD_RandMat(2);
w_mu = [0 0];
%wt = mvnrnd(w_mu,sig_w,length(kftime))';
wt = mvtrnd(sig_w,2,length(kftime))';

%Initial Noisy State/Measurement Values
xnoise(:,1) = x0 + Fd*wt(:,1) + Bd*eqU;
ynoise(:,1) = Cd*xnoise(:,1) + Hd*vt(:,1);
yplot(:,1) = Cd*xnoise(:,1);

%Kalman Filter ICs
Kkf(:,:,1) = sig_0*Cd'*inv(Cd*sig_0*Cd' + Hd*sig_v*Hd');
x_hat(:,1) = Kkf(:,:,1)*ynoise(:,1) + Bd*eqU;
sig_hat(:,:,1) = sig_0;
y_hat(:,1) = Cd*x_hat(:,1);

for j = 2:length(kftime)
    %Noisy State Update
    xnoise(:,j) = Ad*xnoise(:,j-1) + Fd*wt(:,j) + Bd*eqU;
    ynoise(:,j) = Cd*xnoise(:,j) + Hd*vt(:,j);
    yplot(:,j) = Cd*xnoise(:,j);

    %Time Update
    x_hat(:,j) = Ad*x_hat(:,j-1) + Bd*eqU;
    sig_hat(:,:,j) = Ad*sig_hat(:,:,j-1)*Ad' + Fd*sig_w*Fd';

    %Measurement Update
    Kkf(:,:,j) = sig_hat(:,:,j)*Cd'*inv(Cd*sig_hat(:,:,j)*Cd' + Hd*sig_v*Hd');
    x_hat(:,j) = x_hat(:,j) + Kkf(:,:,j)*(ynoise(:,j) - Cd*x_hat(:,j));
    sig_hat(:,:,j) = (eye(6) - Kkf(:,:,j)*Cd)*sig_hat(:,:,j);

    %KF estimate of y
    y_hat(:,j) = Cd*x_hat(:,j);
end

figure
subplot(3,1,1)
plot(kftime(1:end/2 + 0.5),ynoise(1,1:end/2 + 0.5),kftime(1:end/2 + 0.5),y_hat(1,1:end/2 +
 0.5),kftime(1:end/2 + 0.5),yplot(1,1:end/2 + 0.5))
xlabel('Time')
ylabel('h')
title('Kalman Filter Applied to Measurement')
legend('h noise','h est','h plant')
ylim([-50,50])

subplot(3,1,2)
plot(kftime(1:end/2 + 0.5),ynoise(2,1:end/2 + 0.5),kftime(1:end/2 + 0.5),y_hat(2,1:end/2 +
 0.5),kftime(1:end/2 + 0.5),yplot(2,1:end/2 + 0.5))
```

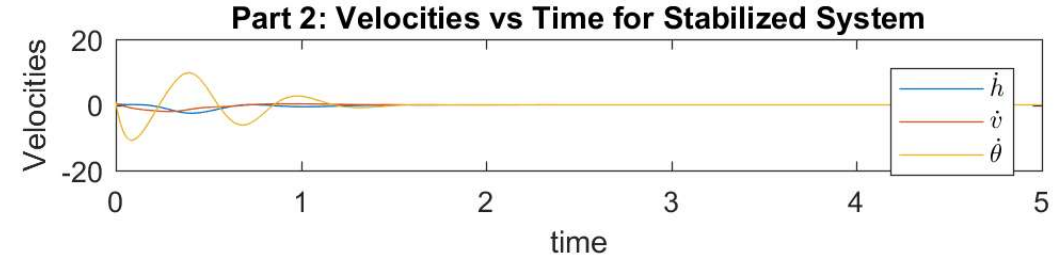
```
xlabel('Time')
ylabel('v')
%title('Kalman Filter Applied to Noisy Measurement, Vertical Position')
legend('v noise','v est','v plant')
ylim([-50,50])

subplot(3,1,3)
plot(kftime(1:end/2 + 0.5),ynoise(3,1:end/2 + 0.5),kftime(1:end/2 + 0.5),y_hat(3,1:end/2 +
 0.5),kftime(1:end/2 + 0.5),yplot(3,1:end/2 + 0.5))
xlabel('Time')
ylabel('\theta')
%title('Kalman Filter Applied to Noisy Measurement, Rotational Position')
legend('\theta noise','\theta est','\theta plant')
ylim([-50,50])

figure
subplot(3,1,1)
plot(kftime,xnoise(1,:),kftime,x_hat(1,:))
title('Noisy State vs Estimated State')
ylabel('h')
legend('noisy h','estimated h')

subplot(3,1,2)
plot(kftime,xnoise(2,:),kftime,x_hat(2,:))
ylabel('v')
legend('noisy v','estimated v')

subplot(3,1,3)
plot(kftime,xnoise(3,:),kftime,x_hat(3,:))
ylabel('\theta')
legend('noisy \theta','estimated \theta')
xlabel('time')
```

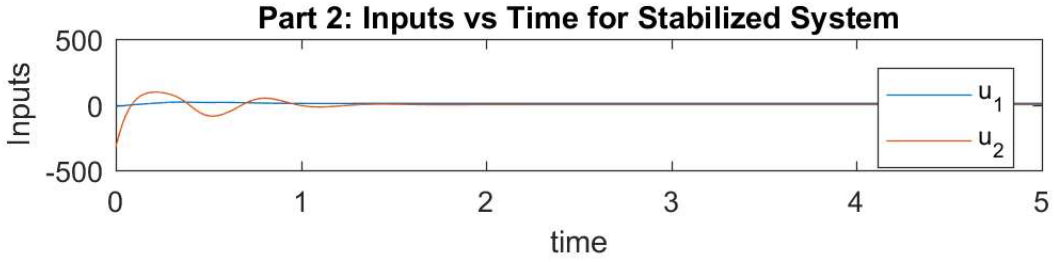
```
Part 2b: Testing controllability of linearized system
        Controllability Matrix is full rank => System is Controllable

Problem 2c: Stabilizing Feedback Control
        Testing Stability of -lambda*I - A matrix
eig_of_Astab =

    -4
    -4
    -4
    -4
    -4
    -4



Part 2d: Stabilizing the System with a Feedback gain matrix
        Closed Loop system is stable if all eigenvalues have negative real components
eig_of_Acl =

 -29.3105 + 0.0000i
```

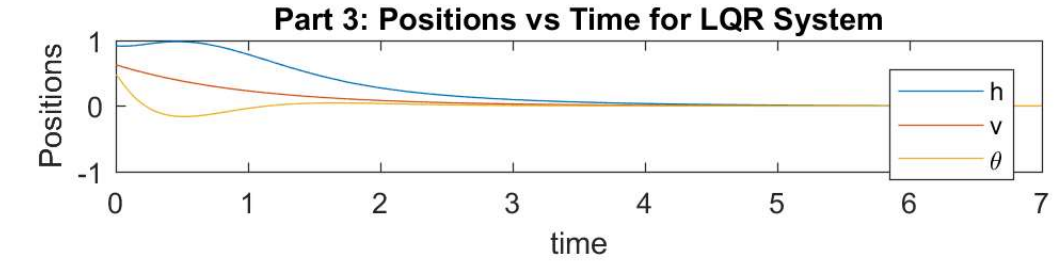
```
  -1.0718 + 3.8537i
  -1.0718 - 3.8537i
  -0.5459 + 0.0000i
 -14.9282 + 0.0000i
  -1.0718 + 0.0000i
```

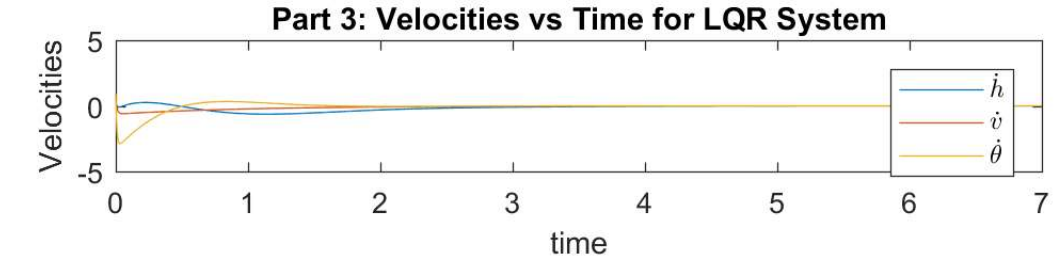


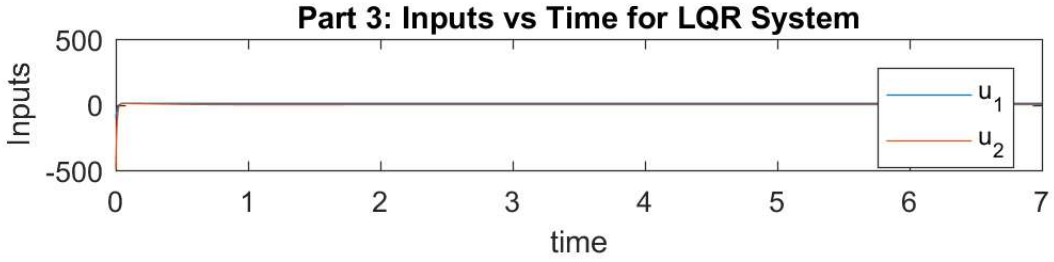

Part 2: Positions vs Time for Stabilized System
States
time
h
v
θ

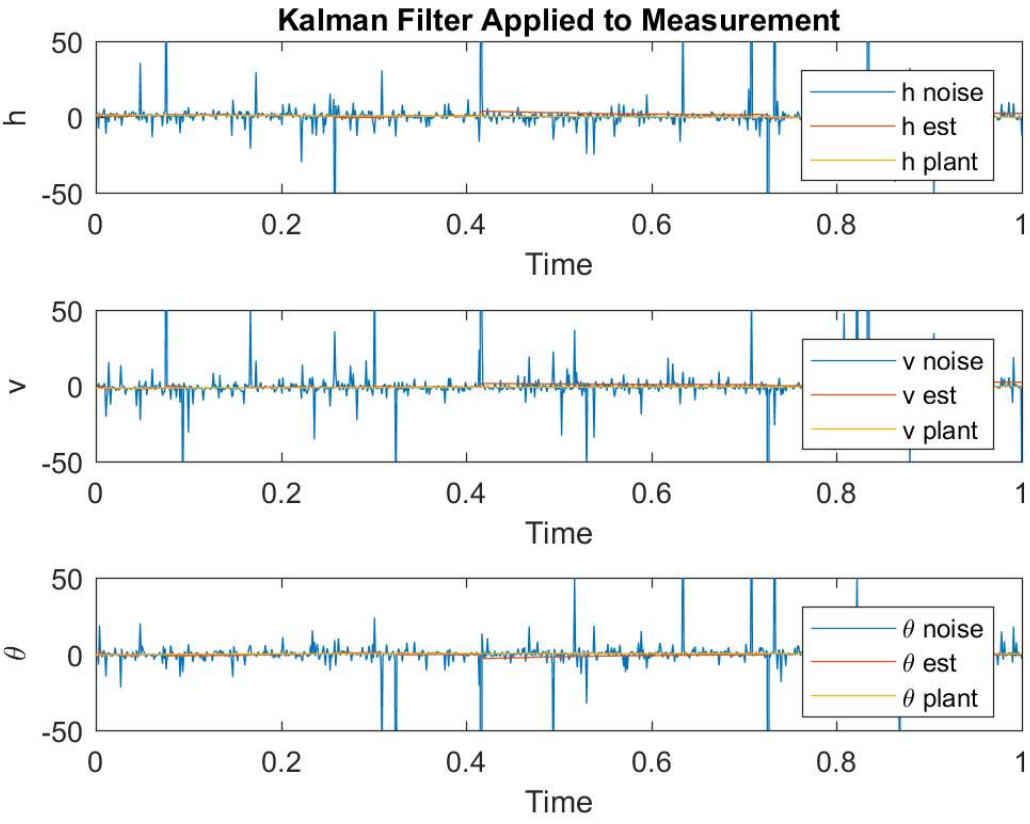

Part 2: Velocities vs Time for Stabilized System
Velocities
time
ḣ
v̇
θ̇

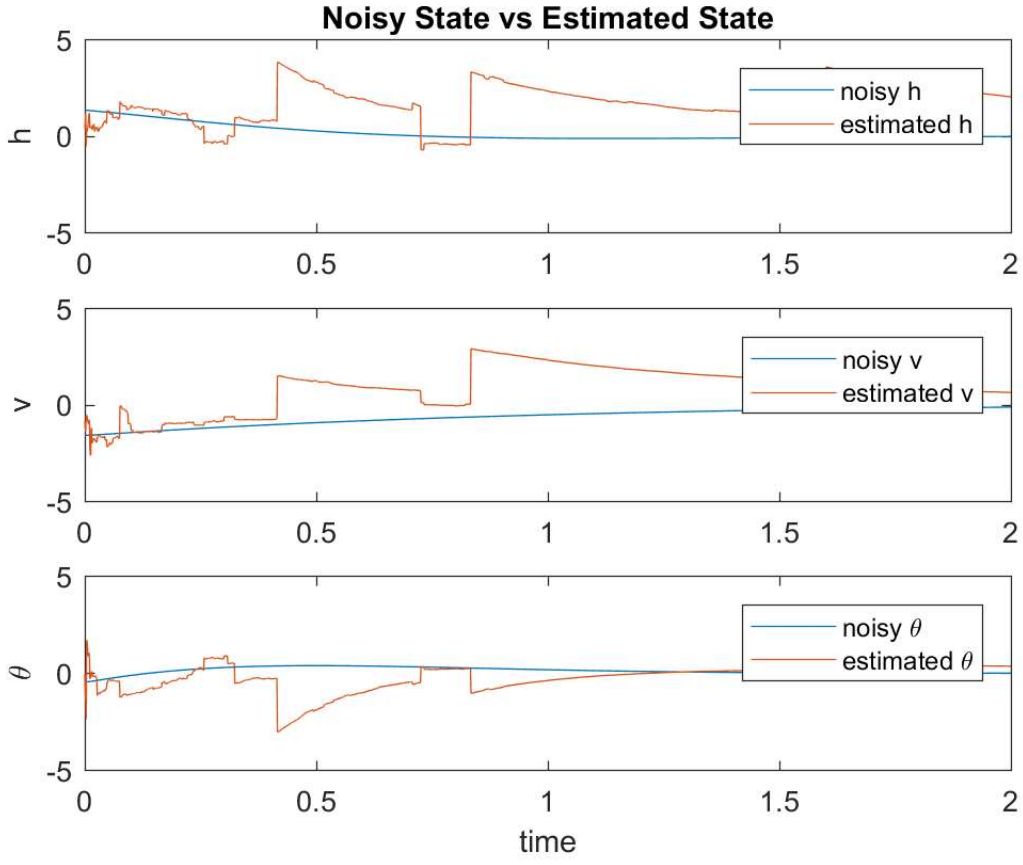

Part 2: Inputs vs Time for Stabilized System
Inputs
time
u1
u2


Part 3: Positions vs Time for LQR System
Positions
time
h
v
θ


Part 3: Velocities vs Time for LQR System
Velocities
time
ḣ
v̇
θ̇


Part 3: Inputs vs Time for LQR System
Inputs
time
u1
u2


Kalman Filter Applied to Measurement
h
v
θ
Time
h noise
h est
h plant
v noise
v est
v plant
θ noise
θ est
θ plant
50
0
-50
0
0.2
0.4
0.6
0.8
1


Noisy State vs Estimated State
h
v
θ
time
noisy h
estimated h
noisy v
estimated v
noisy θ
estimated θ
5
0
-5
0
0.5
1
1.5
2

Published with MATLAB® R2017a

```
function qdot_plus = f(t,u,q,qdot,m,I,g)
%Param t         : time step [s]
%Param u         : control input (thrust, torque) [N,Nm]
%Param q         : state (horizontal, vertical, roation) [m,m,rad]
%Param qdot      : state derivative [m/s,m/s,rad/s]
%Param m         : mass [kg]
%Param I         : moment of Inertia [kg/m^2]
%Param g         : gravitational accelleration [m/s^2]

qdot_plus = qdot + t*qdotdot(u,q,m,I,g);

end
```

```
Not enough input arguments.

Error in f (line 10)
qdot_plus = qdot + t*qdotdot(u,q,m,I,g);
```

Published with MATLAB® R2017a

```
function q_plus = h(t,u,q,qdot,qdotdot,m,I,g)
%Param t         : time step [s]
%Param u         : control input (thrust, torque) [N,Nm]
%Param q         : state (horizontal, vertical, roation) [m,m,rad]
%Param qdot      : state velocity [m/s,m/s,rad/s]
%Param qdotdot   : state acceleration [m/2^2, m/s^2, rad/s^2]
%Param m         : mass [kg]
%Param I         : moment of Inertia [kg/m^2]
%Param g         : gravitational accelleration [m/s^2]

q_plus = q + t*qdot + (t^2)*qdotdot;

end
```

```
Not enough input arguments.

Error in h (line 11)
q_plus = q + t*qdot + (t^2)*qdotdot;
```

Published with MATLAB® R2017a

```
function q_dotdot = qdotdot(u,q,m,I,g)
%Param u         : control input (thrust, torque) [N,Nm]
%Param q         : state (horizontal, vertical, roation) [m,m,rad]
%Param m         : mass [kg]
%Param I         : moment of Inertia [kg/m^2]
%Param g         : gravitational accelleration [m/s^2]

q_dotdot = [(u(1)/m)*sin(q(3));
            (u(1)/m)*cos(q(3)) - g;
            u(2)/I];

end
```

```
Not enough input arguments.

Error in qdotdot (line 8)
q_dotdot = [(u(1)/m)*sin(q(3));
```

Published with MATLAB® R2017a

```
function matPD = PD_RandMat(size)
% This function creates a Positive Definite matrix of a given size.

%Param size     : size of symmetric matrix to create

matPD = rand(size,size);

matPD = 0.5*(matPD + transpose(matPD)) + randi(size)*eye(size);

end
```

```
Not enough input arguments.

Error in PD_RandMat (line 6)
matPD = rand(size,size);
```

Published with MATLAB® R2017a

```
function matPSD = PSD_RandMat(size)
% This function creates a Positive Semi-Definite matrix of a given size.

%Param size     : size of symmetric matrix to create

matPSD = rand(size,size);

matPSD = 0.5*(matPSD + transpose(matPSD)) + randi(size)*eye(size);
matPSD(end,:) = zeros(1,length(matPSD(end,:)));
matPSD(:,end) = zeros(length(matPSD(end,:)),1);
end
```

```
Not enough input arguments.

Error in PSD_RandMat (line 6)
matPSD = rand(size,size);
```

Published with MATLAB® R2017a

```
function P = CT_Riccati_FwdEuler2(Q, R, A, B, Qf, dt, T)

P(:,:,1) = Qf;                          %Final value equal to Qf

time = 0:dt:T;

%Precomputing constant values here for efficiency.
Rinv = inv(R);
Btr = transpose(B);
Atr = transpose(A);

for i = 1:length(time)-1
    P(:,:,i+1) = P(:,:,i) - dt*(-Atr*P(:,:,i) - P(:,:,i)*A + P(:,:,i)*B*Rinv*Btr*P(:,:,i)
- Q);
end
end
```

```
Not enough input arguments.

Error in CT_Riccati_FwdEuler2 (line 3)
P(:,:,1) = Qf;                          %Final value equal to Qf
```

Published with MATLAB® R2017a

```
function cGram = teddyGram(A,B,dt)
%        _               _
%      ((`'-"``""-'`))
%        )    -     -   (
%       /     o _ o     \
%       \     ( 0 )     /    <I)
%     _, '-..__^__..-' ,_
%    (:'`'-/         \-'`':)
%     `'-.; .;;;;;. ;.-'``
%         ; ;;ctrl; ;
%        /  ';;;;;'  \
%       /  ;._____.;  \
%      _\  \         /  /_
%     (((___|      |___)))
% The purpose of this function is to solve the modified Lyapunov eqation
% for the Controllability Gramian. This method has non-ideal convergence
% rates and should only be used if lyap() and gram() are unavailable.

t = 0;
Q = B*transpose(B);

ctrlGram(:,:,1) = zeros(size(B*B'));
Q_ = A*ctrlGram(:,:,1) + ctrlGram(:,:,1)*A';

i=1;

while abs(norm(Q_ + Q,2)) > 0.006
   ctrlGram(:,:,i+1) =  ctrlGram(:,:,i) + dt*(expm(A*t)*B*B'*expm(A'*t));

   Q_ = A*ctrlGram(:,:,i+1) + ctrlGram(:,:,i+1)*A';

   t = t + dt;
   i = i + 1;

   if i > 3e3
       fprintf('\n\nfunction teddyGram reached max iterations');
       break
   end

end
cGram = ctrlGram(:,:,end);
end
```

```
Not enough input arguments.

Error in teddyGram (line 20)
Q = B*transpose(B);
```

Published with MATLAB® R2017a