# Two-Wheel Nonholonomic Robot Trajectory Control

ES 158 Final Project Report, Fall 2016

Joanna Chung, Liam Mulshine

**ABSTRACT:** In this final project, we seek to model a system for a two-wheeled nonholonomic robot that follows a given reference trajectory path through feedback control. Three control systems have been developed and presented here. The first, proportional-integral control, was the naive first-pass implementation of a controller that used constant control matrices on measurements of error to track the trajectory. A more advanced controller, the Linear-Quadratic-Regulator, provided far better tracking performance and eliminated steady state error. Adding a Kalman filter to the LQR, we modeled a Linear-Quadratic-Gaussian controller as our third system, which was highly robust to noise.

**Figure 1.** Photograph of a two-wheeled nonholonomic robot

## INTRODUCTION & BACKGROUND INFORMATION

Our project was inspired by a challenge that is currently facing the undergraduate robotics club on campus, *RFC Cambridge*. This club, which is less formally known as RoboCup, specializes in designing autonomous soccer playing robots. In order to enable the robots to play soccer, the club needs to develop a control system capable of guiding the robots along predetermined trajectories. Our goal for this project is to develop a trajectory tracking controller for a simplified version of RoboCup's robots that is robust to input noise and unexpected deviation in environment variables.

Our initial approach to this control problem was to use a simple PI controller to track the reference path. We quickly realized, however, that this simple controller was not nearly robust enough for our purposes, due to the time-varying nature of the system. Our next approach used the time-

varying Linear-Quadratic-Regulator (LQR), which allowed us to determine the optimal feedback gain along each point in our reference path. The results of this controller were strong. However, LQR does not account for system disturbances and noise. To improve the performance of our controller in the presence of noise and disturbance, we introduced a Kalman filter, the optimal observer. Combining the optimal feedback gain from LQR with the Kalman Filter made our controller a Linear-Quadratic-Gaussian (LQG) controller. This controller not only tracked our reference input, but was robust even in more life-like simulations with noise and disturbance.

## CONTROL METHODS

### PI Control Model

We created a first iteration model using basic PID control to simulate a minimum viable product that performs satisfactorily. The model uses a combination of feedforward control, based on the reference trajectory, and feedback control to provide robustness to error.

Below are the parameters of the system we intend to model:

$x$ - State Space - position, tangential velocity, and angular velocity: $x = [x, y, \theta]^T$
$u$ - Controller outputs - tangential velocity ($v_t$), angular velocity ($\omega$): $u = [v_t, \omega]^T$
$x_r$ - Reference Inputs - $x_r = [x_r, y_r, \theta_r]^T$
$u_r$ - Feedforward Inputs - $[v_{t_r}, \omega_r]^T$

A visualization of the system states and reference inputs are shown below, in a diagram from Klancar, et al.:

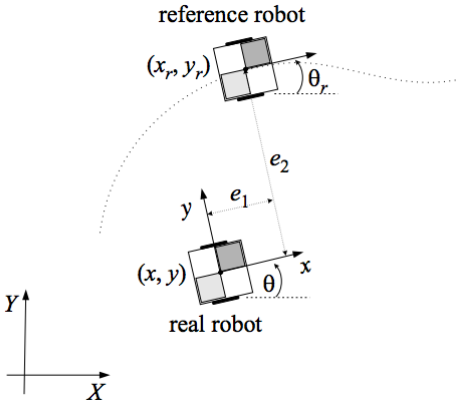



**Figure 2.** Diagram of system parameters and states, the reference positions, actual positions, and error

Our model is that of a two-wheeled nonholonomic square robot with length L that is controlled with tangential velocity and angular velocity inputs, translated into velocity inputs to each wheel with the formula:

$$v_R = v + \frac{\omega L}{2} \; v_L = v - \frac{\omega L}{2}$$

In the block diagram below, we show the robot dynamics as the plant of the feedback control system, and u as the controller, which we design as proportional-integral control.

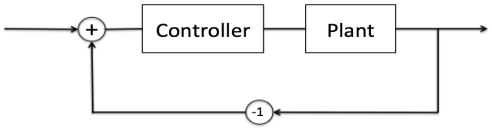



**Figure 3.** Block diagram of feedback system, with plant as the robot dynamics and the controller to be designed

Based on the block diagram above, we first defined the robot's system dynamics:

$$\dot{x} = Ax + Bu$$

$$x = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} \quad A = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad B = \begin{bmatrix} cos(\theta) & 0 \\ sin(\theta) & 0 \\ 0 & 1 \end{bmatrix} \quad u = \begin{bmatrix} v \\ w \end{bmatrix}$$

We then calculated the feedforward control inputs, and reference angle based on the reference path:

$$v_r(t) = \pm\sqrt{\dot{x}_r^2(t) + \dot{y}_r^2(t)}$$

$$\omega_r(t) = \frac{\dot{x}_r(t)\ddot{y}_r(t) - \dot{y}_r(t)\ddot{x}_r(t)}{\dot{x}_r^2(t) + \dot{y}_r^2(t)}$$

$$\theta_r(t) = \arctan2(\dot{y}_r(t),\ \dot{x}_r(t))$$

In order to account for error, defined as $e = x - x_r$, we added a PI controller, with the controller in Figure 1 defined as:

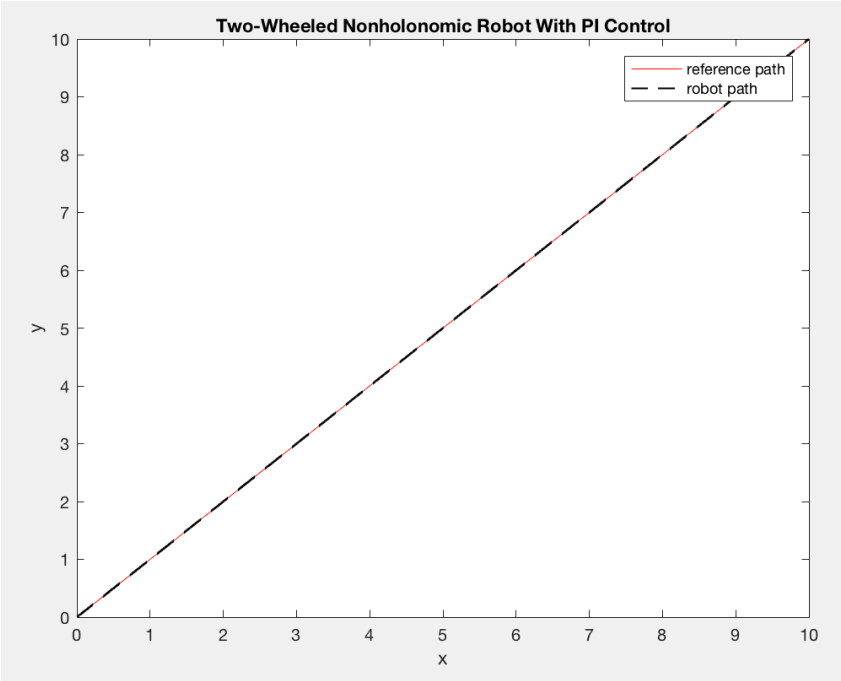
$$u = K_p e - K_i \int e * dt$$

And gain matrices

$$K_p = \begin{vmatrix} -kp1 & -kp2 & 0 \\ 0 & 0 & -kp3 \end{vmatrix} \qquad K_i = \begin{vmatrix} -ki1 & -ki2 & 0 \\ 0 & 0 & -ki3 \end{vmatrix}$$

In the gain matrices, the top row of each matrix multiplies with the error in order to compose the feedback tangential velocity, and the bottom row composes the angular velocity.

Constants for each of the matrices were found by guessing and checking, and yielded the results below.



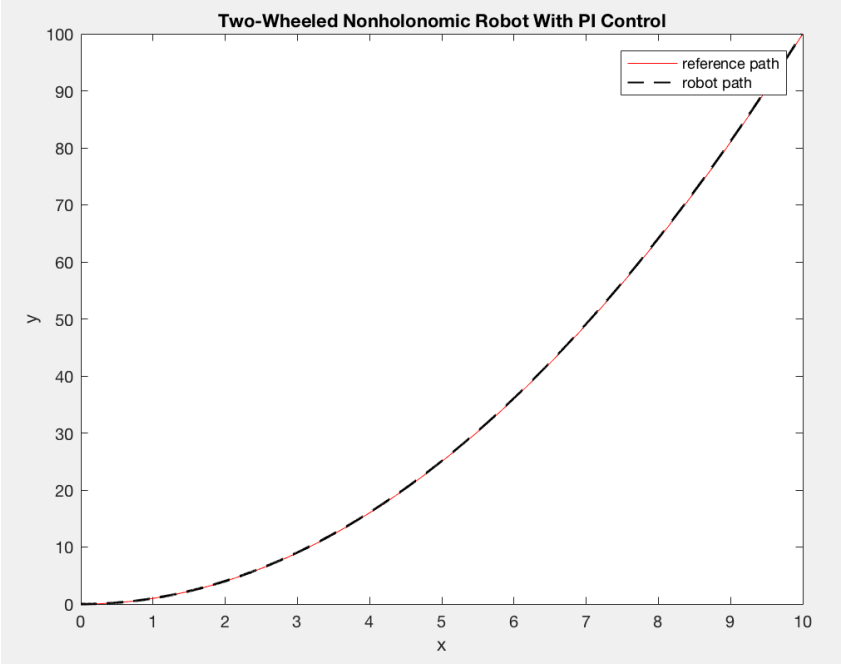




**Figure 4.** PI control tracking a linear reference path and a parabolic reference path.

Although these results are satisfactory for tracking simple reference trajectories, when more complex reference trajectories are tested such as paths with large curvature, the controller breaks down. This is because the optimal gains vary with time, but our PI controller uses constant K matrices. Also, since both x and y errors affect the tangential velocity, when they are equal but opposite, the feedback velocity will remain constant even though error exists. Thus, we needed to design a better controller.

## Linear-Quadratic-Regulator (LQR) Control

The system that we modeled was time variant in two ways. Firstly, our reference state inputs varied with time, since our robot's position changed as it moved along the reference path. Secondly, the system dynamics were time variant, since the robot we used was non-holonomic. Thus, determining a feedback gain matrix, K, that both produced accurate results and was time-invariant (as we attempted to do in the PI control implementation) was very difficult. In fact, for almost every combination of feedback gain terms that we used, the results were poor as time moved forward, particularly for reference paths with high curvature.

Thus, we set out to find the optimal feedback control at each point along our path, using a time varying linear quadratic regulator (LQR). Before doing so, we needed to linearize the nonholonomic robot system dynamics around each point along the reference path. Since we solved this control problem in the discrete domain, we also needed to discretize our system. The discrete-time nonlinear state space model for our system can be represented as

$$x(i+1) \;=\; A\,x(i) \;+\; B(x(i),i)\,u(i)$$

where A and B are defined in the previous section of this paper. We then define,

$$g(x(i),u(i),i) \;=\; B(x(i),i)\,u(i)$$

Where $g(x(i),u(i),i)$ is the only nonlinear term in the system dynamics. Linearizing the dynamics around the reference inputs, $x \;=\; x_r, u \;=\; u_r$, yields:

$$A_{lin} \;=\; A + \;(A + \frac{\delta g}{\delta x_{(x_r,u_r)}}\;\underline{x}) \;*\; dt$$

$$B_{lin} \;=\; \frac{\delta g}{\delta u_{(x_r,u_r)}}\;\underline{u} \;*\; dt$$

Where $\underline{x} = \;x - x_r, \underline{u} = \;u - u_r$, and $dt$ is the time interval between steps. Thus, our discretized linearized state space system is defined below:

$$\underline{x}\,(i+1) = \;A_{lin}(i)\,\underline{x}(i) \;+\; B_{lin}(i)\,\underline{u}(i)$$

$$\underline{y}\,(i+1) = C\,\underline{x}(i)$$
$$\underline{u}(i) = -K(i)\,\underline{x}(i)$$

After linearizing A and B, neither is dependent upon a state variable; instead, each is dependent only upon the current timestep, i.

Now we can calculate the optimal feedback control matrix at each point in our reference path by minimizing the following quadratic cost function.

$$J = E\left[\mathbf{x}_N^{\mathrm{T}} F \mathbf{x}_N + \sum_{i=0}^{N-1} \mathbf{x}_i^{\mathrm{T}} Q_i \mathbf{x}_i + \mathbf{u}_i^{\mathrm{T}} R_i \mathbf{u}_i\right],$$
$$F \geq 0, Q_i \geq 0, R_i > 0.$$

https://en.wikipedia.org/

The cost function J, is defined in terms of the system's state error, x, the control input error, u and the positive definite matrices, F, Q and R. Broadly speaking, F, Q and R are penalty matrices. One could increase the magnitude of Q to increase the penalty on deviation from the reference input. Similarly, one could increase the magnitude of R to increase the penalty on deviation from the reference control, and increase the magnitude of F in order to further penalize state error on the final time step. Varying the magnitude of F, Q, and R will affect the exact response of the final controller to certain errors.

The optimal feedback gain (the feedback gain that minimizes this cost function), K, can be determined a priori using the solution to a backwards pass of the differential Riccati Equation, S.

$$K_i = (B_i{}^T S_{i+1} B_i + R_i)^{-1} B_i{}^T S_{i+1} A_i$$
$$S_i = A_i{}^T (S_{i+1} - S_{i+1} B_i (B_i{}^T S_{i+1} B_i + R_i)^{-1} B_i{}^T S_{i+1}) A_i + Q_i, S_N = F$$

With the feedback gain matrix, K, and the linearized dynamics known, we were able to simulate our LQR system. See the results section for figures demonstrating the performance of our controller.

## Linear-Quadratic-Gaussian (LQG) Control

It is important to recognize the fact that disturbance and noise within our system is inevitable. The system plant could experience unexpected disturbances, and the sensor will undoubtedly contain some degree of noise. The LQG controller accounts for this by adding a state estimator that runs in parallel with the system plant to generate an accurate estimate of the system's next state despite disturbance and noise. This estimation loop, combined with the optimal feedback gain determined through the LQR method results in a controller robust to noise and initial state estimation error. Our discretized linear state space system with optimal feedback and estimator gain is defined below.

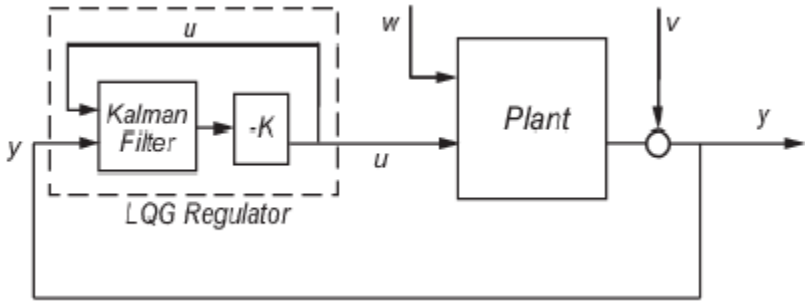
$$\hat{x}(i+1) = A_{lin}(t)\,\hat{x}(i) + B_{lin}(i)\,u(i) + L(i)(y(i+1) - C\hat{x}(i))$$
$$\hat{y}(i+1) = C\,\hat{x}(i)$$
$$u(i) = -K\,\hat{x}(i)$$

In this system, $\hat{x}$is defined as the estimation error (i.e. the difference between the estimated state and the actual state). It follows that $\hat{y}$ is the estimated error in the output term, which is fed back into the system dynamics. The gain matrix, L, is the optimal observer gain matrix calculated using the result of a forward pass of the differential Riccati Equation, P.

$$L_i = P_i C^T (C P_i C^T + W_i)^{-1}$$
$$P_{i+1} = A_i (P_i - P_i C^T (C P_i C^T + W_i)^{-1} C_i P_i)\, A_i^T + V_i, P_0 = E(x_0 - \hat{x})(x_0 - \hat{x})^T$$

Similarly, the gain matrix, K, is the optimal feedback gain, calculated as explained above in the section on LQR.

The optimal observer term in the system is known as the Kalman filter. The Kalman filter allows the LQG controller to generate accurate state estimates. Below is a high level block diagram of the entire LQG control system.




https://www.researchgate.net/

From this diagram, the effect of combining an optimal observer (the Kalman Filter) with optimal feedback from LQR (K) is clear. The two work in conjunction to generate an optimal control output which is fed into the true system plant (the physical robot) for robust robotic control.

We used our discretized linear state space model with optimal feedback gain and a Kalman filter to implement an LQG controller in MATLAB to control our nonholonomic, differential drive robot. Numerical results are presented and explained in the results section of this paper.

## RESULTS

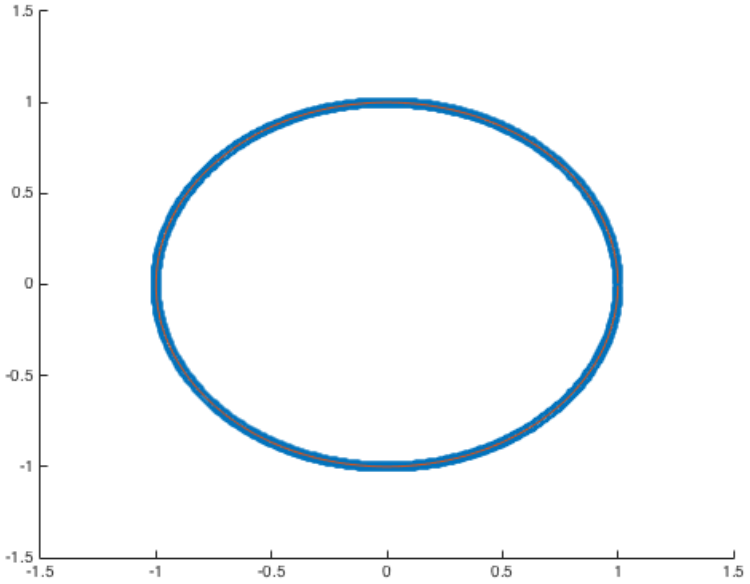
In this section, we present results indicating the performance of our LQR controller given a circular reference path and initial conditions unequal to the initial path position, for two different cost functions. We then test our LQG controller under similar conditions, except we introduce system

disturbances and noise. A discussion and comparison of the results can be found in the Discussion of Results section of the paper.

## LQR

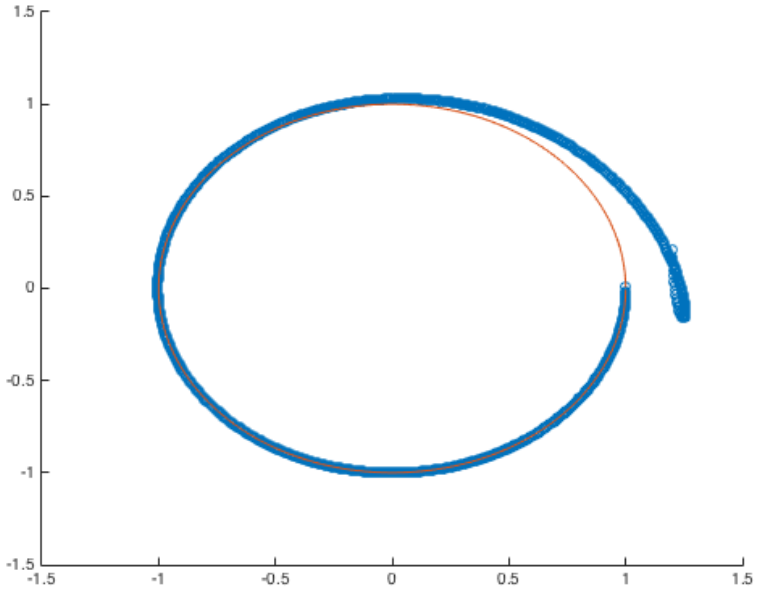
Ideal Conditions:

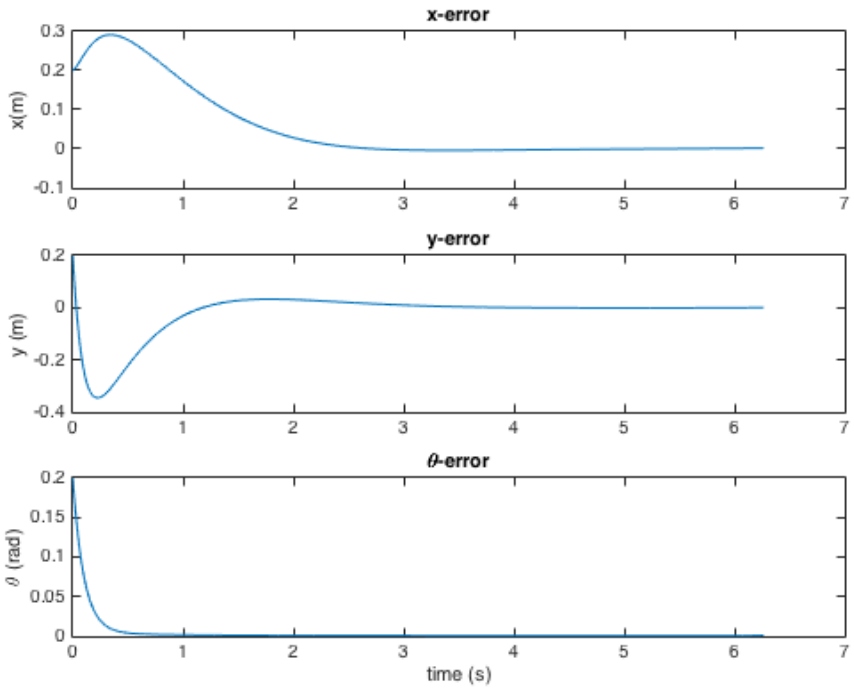
**Robot Trajectory**

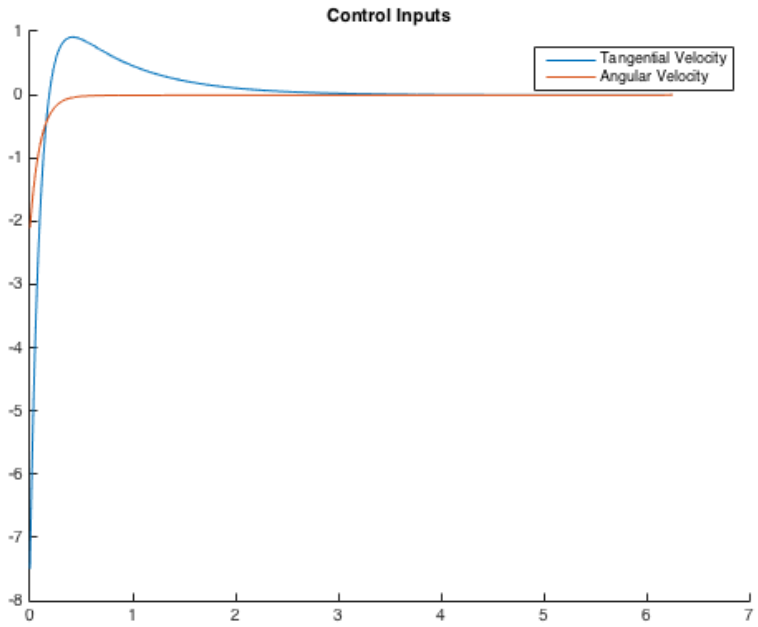



Inaccurate Starting Position:

**Robot Trajectory**

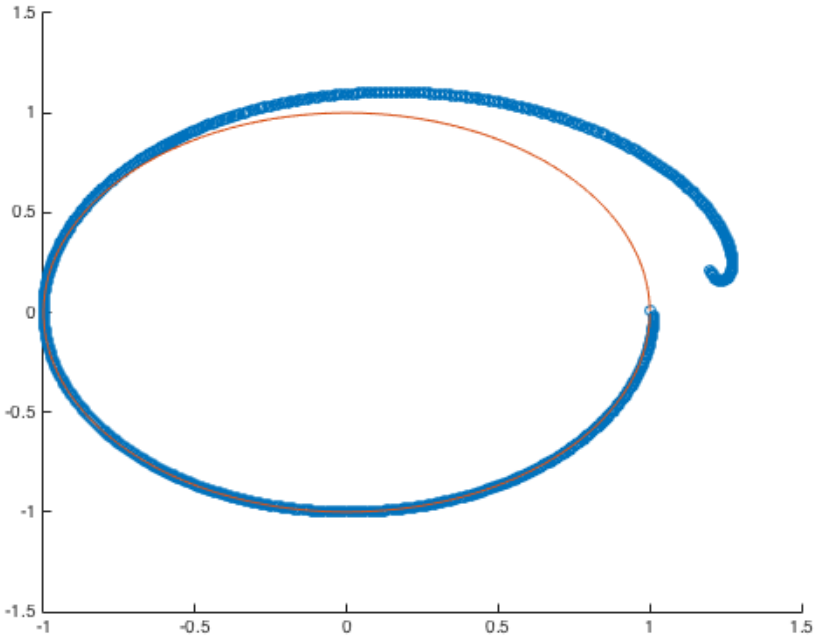

1.5
1
0.5
0
-0.5
-1
-1.5
-1.5
-1
-0.5
0
0.5
1
1.5

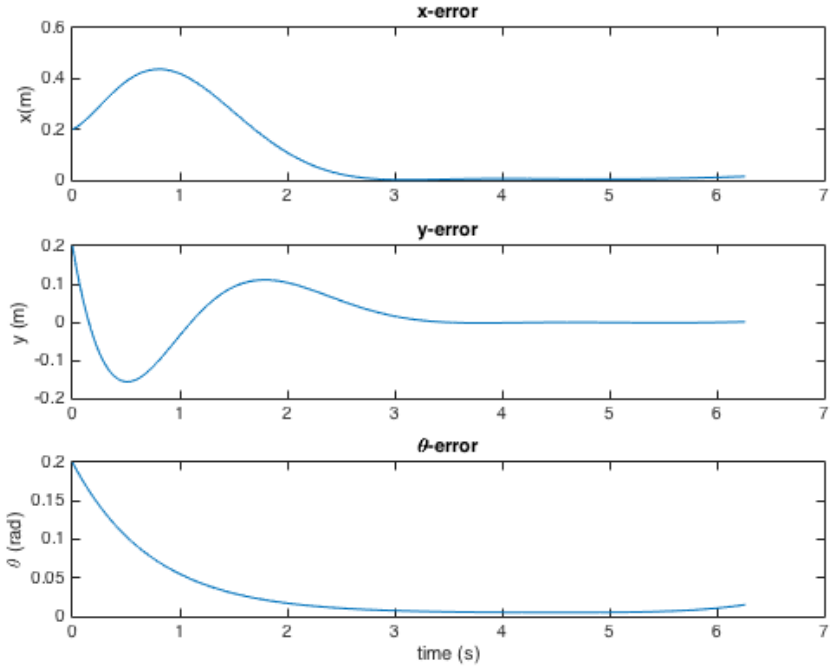

x-error
x(m)
0.3
0.2
0.1
0
-0.1
y-error
y (m)
0.2
0
-0.2
-0.4
θ-error
θ (rad)
0.2
0.15
0.1
0.05
0
0 1 2 3 4 5 6 7
time (s)

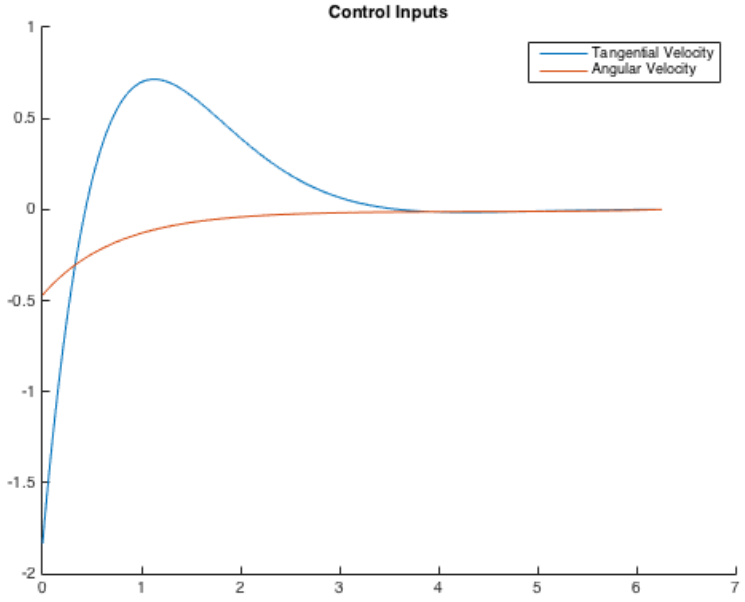



Lower Penalty on State Variable:

**Robot Trajectory**

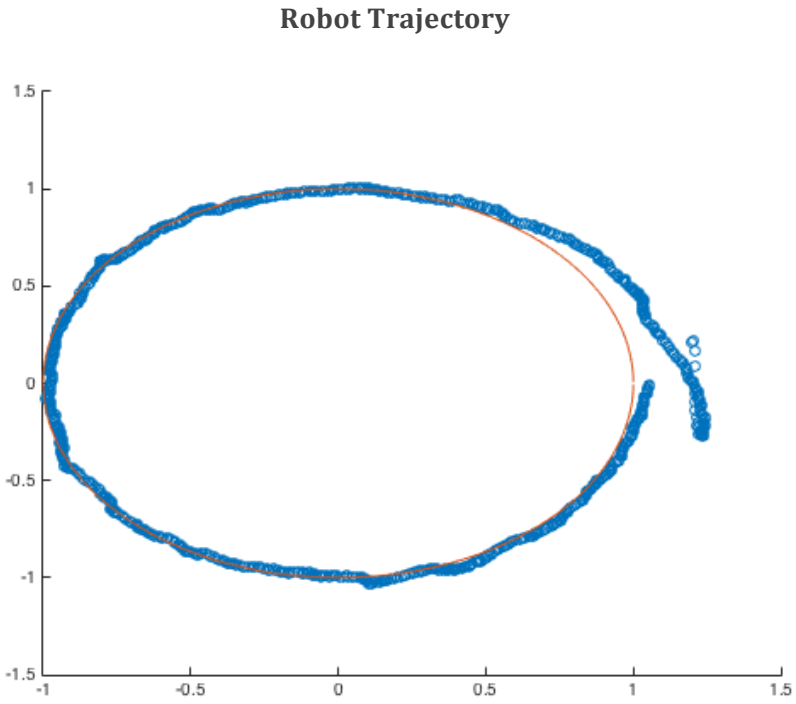


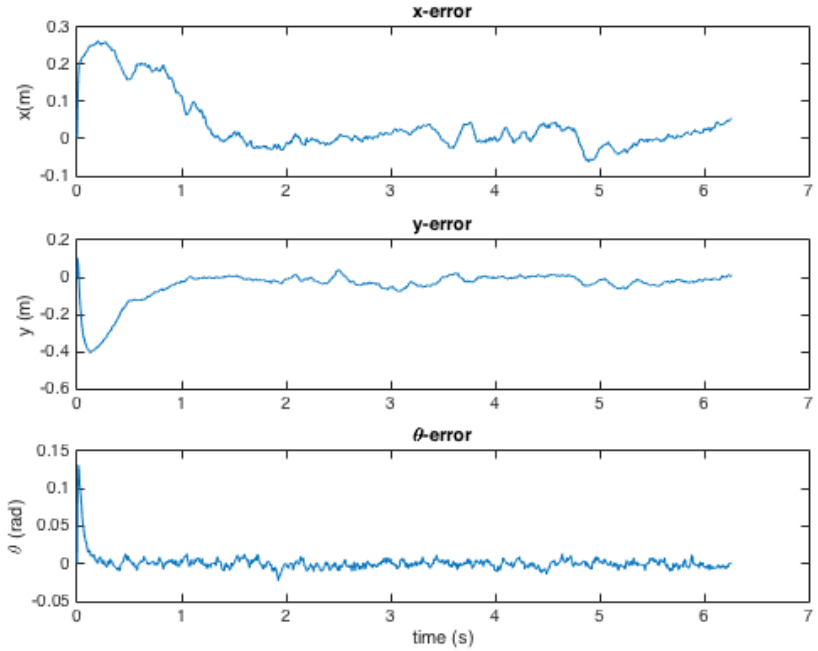

x-error
x(m)
y-error
y (m)
θ-error
θ (rad)
time (s)

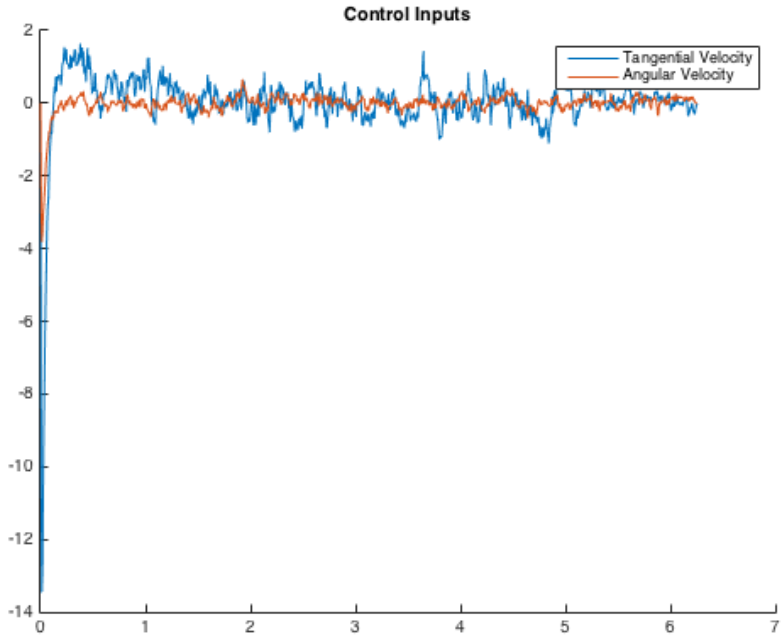

Control Inputs
Tangential Velocity
Angular Velocity

**LQG**

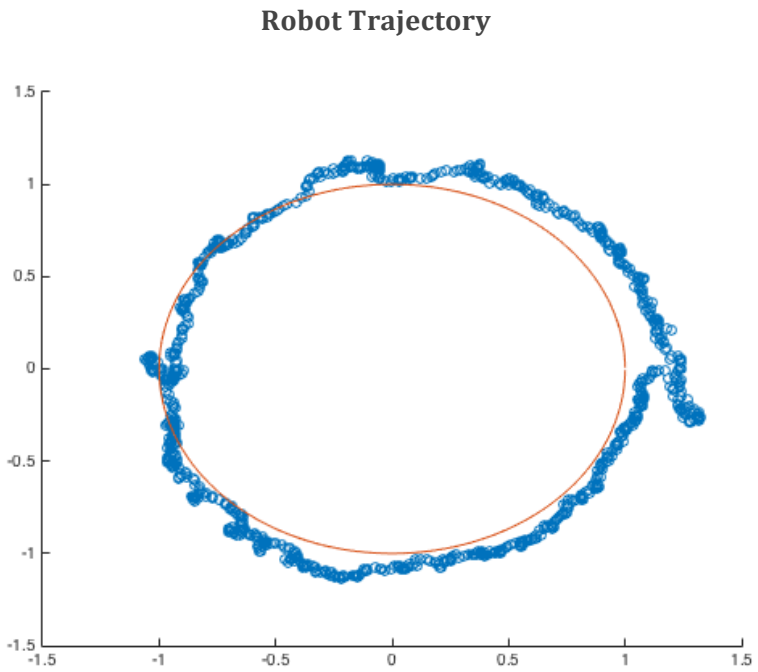
15% SNR, Small Disturbances and Inaccurate Initial State Estimate:

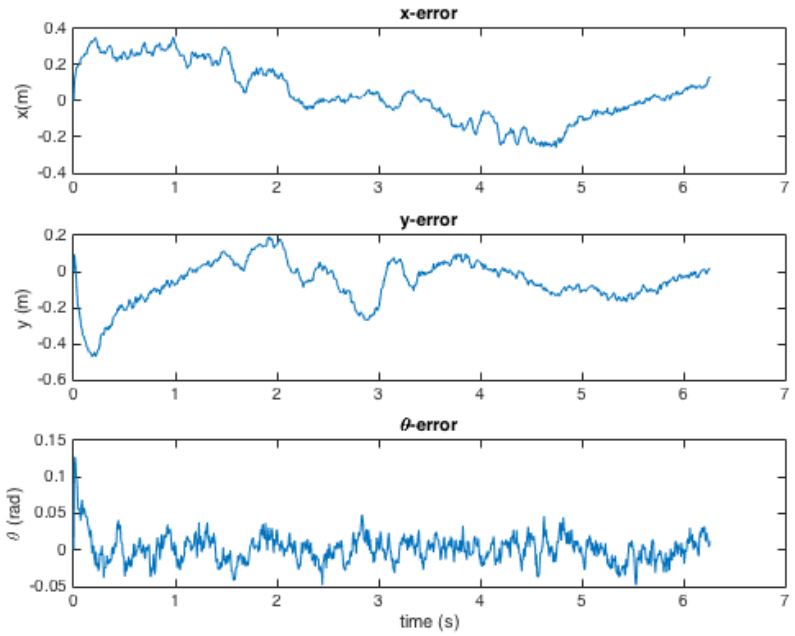


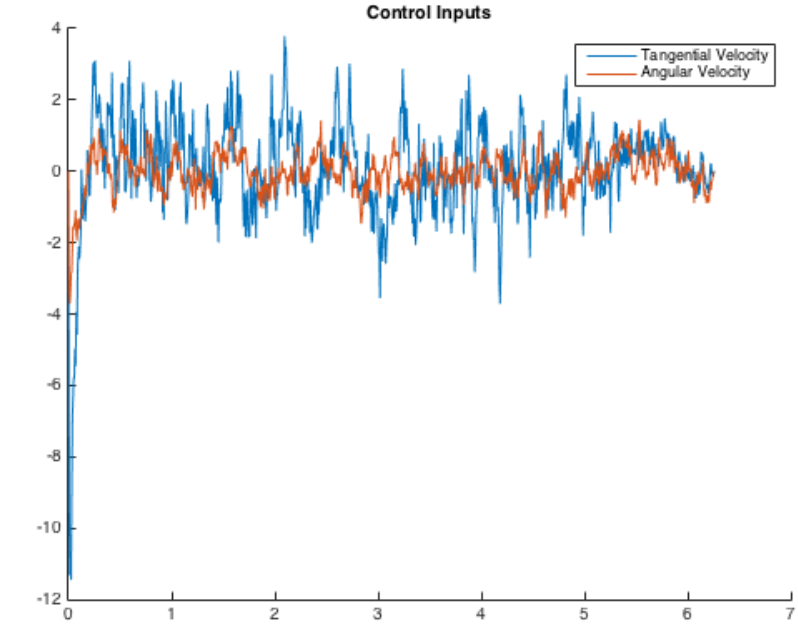






50% SNR, Small Disturbances and Inaccurate Initial State Estimate:




x-error
x(m)
y-error
y (m)
θ-error
θ (rad)
time (s)


Control Inputs
Tangential Velocity
Angular Velocity

# DISCUSSION OF RESULTS

## PI Control

The performance of our PI controller was not strong. Not only did steady state error fail to reach zero, but the system grew unstable for more complex reference paths. This can likely be attributed to the fact that we were attempting to generalize the feedback gain matrix for all points in our time-varying system. For paths that did not cause the dynamics to vary significantly, choosing a feedback gain matrix was not quite as difficult. However, for paths such as the circle, even after testing the system with a variety of different feedback gain matrices, the system failed to satisfactorily track the reference input.

## LQR Control

LQR solved many of the issues that we were seeing with the PI controller. Not only did the steady state error converge to zero, but also, it did so without introducing unusually large control inputs for both experiments.

It is interesting to note the differences between the two experiments with LQR. In the first case, when solving for the optimal controller, we penalized position error quite heavily, by increasing the magnitude of the penalization matrix, Q, in the cost function. Consequently, the controller generated larger control inputs to push the robot towards the reference path with haste. In contrast, in the second experiment, we reduced the magnitude of Q. The controller thus focused more heavily on reducing error in the reference input, slowing the response to the initial position error. In the control input plots for both controllers, we see that the magnitude of the control input is greater in the first experiment than it is in the second experiment as we would expect.

To increase our confidence in the robustness of our controller, we passed the control outputs that it generated through our nonlinear dynamics. The response was nearly identical, suggesting that, despite linearization, our controller would likely work in a physical two wheeled differential drive non-holonomic robot.

## LQG Control

The response of our LQG controller is quite similar in many respects to that of our LQR controller, apart from the presence of noise and disturbance in the former. The LQG controller tracks the reference path well, even in the presence of noise as high as 50% of the signal level. In order to generate our noise and disturbance terms, we used the MATLAB function, $mvnrnd(MU, SIGMA)$ which generates a vector of terms taken from a gaussian distribution with mean, $MU$, and covariance, $SIGMA$.

After some time, the steady state error terms for both experiments average close to 0. The deviation from 0 can be attributed to the gaussian noise and disturbance terms.

We passed our LQG controller - generated from the linearized robot dynamics - through the nonlinear dynamics, and saw a stable response that was nearly identical to the response that we saw when simulating with the linear dynamics. As in the LQR case, this suggests that, despite linearization, our controller would likely be able to control a physical two wheeled differential drive non-holonomic robot.

## CONCLUSION

This project pushed us to apply the knowledge that we had acquired throughout the semester on Linear Time-Invariant (LTV) systems to a more complex Nonlinear Time-Varying system. Although insufficient time kept us from implementing our controller on a physical system, we hope to continue with this project in upcoming semesters to test it on a physical two-wheeled non-holonomic robot using the test setup in lab. Future project groups can also explore modeling the same dynamical system, optimizing for parameters other than tracking accuracy and reaction time, by adding them to the cost function. These can include power consumption, G-Force, etc. Another control model that can be explored is the Model Reference Control (MRC) system, which would take future projections of trajectories into account and preemptively adjust the current system to optimize the trajectory tracking performance.

## REFERENCE SOURCES


Åström, Karl Johan, and Richard M. Murray. Feedback System: An Introduction for Scientists and Engineers. Princeton: Princeton UP, 2008. Print.

Klancar, G., D. Matko, and S. Blazic. "Mobile Robot Control on a Reference Path."*Proceedings of the 2005 IEEE International Symposium On, Mediterrean Conference on Control and Automation Intelligent Control, 2005.* (2005): n. pag. Web. 12 Dec. 2016. <http://msc.fe.uni-lj.si/Papers/MED05_Klancar.pdf>.

"Linear–quadratic–Gaussian Control." *Wikipedia*. Wikimedia Foundation, 8 Nov. 2016. Web. 13 Dec. 2016. <https://en.wikipedia.org/wiki/Linear%E2%80%93quadratic%E2%80%93Gaussian_control#Mathematical_description_of_the_problem_and_solution>.

Tedrake, Russ, Ian Manchester, and Mark Tobenkin. "LQR-Trees: Feedback Motion Planning via Sums-of-Squares Verification." *Robotics: Science and Systems Conference* (2009): n. pag. MIT, 27 Feb. 2010. Web. 13 Dec. 2016. <http://groups.csail.mit.edu/robotics-center/public_papers/Tedrake10.pdf>.

# APPENDIX A

**PI Control**

```
clear
close all

% parameters
N = 10000;
t_tot = 10;
dt = t_tot / N;
t = linspace(0,t_tot,N);

r = 1;
% Define path
x_r = t;
y_r = t.^2;
figure();
plot(x_r,y_r);

% PI controller values
kp1 = 200;
kp2 = 200;
kp3 = 100;
ki1 = 500;
ki2 = 100;
ki3 = 300;

% controller K matrix
Kp =[ kp1   kp2   0;
       0     0   kp3];

Ki =  [ki1  ki2   0;
        0    0   ki3];

x_d = diff(x_r);
y_d = diff(y_r);
x_dd = diff(x_d);
y_dd = diff(y_d);
x_d = x_d(1:end-1);
y_d = y_d(1:end-1);

% reference input
v_r = sqrt(x_d.^2 + y_d.^2);
w_r = (x_d.*y_dd - y_d.*x_dd)./(x_d.^2 + y_d.^2);
feedforward = [v_r; w_r];

% reference angle calculation
theta_r = atan2(y_d,x_d);

% initial conditions = reference conditions
x(:,1) = [x_r(1);
          y_r(1);
          theta_r(1)];

e = zeros(3,N);
e(:,1) = x(:,1) - [x_r(1); y_r(1); theta_r(1)];
e_i = zeros(3,N);
v = zeros(2,N);
```

```
A = eye(3);

for i = 1:N-3

    % controller outputs based on error
    u(:,i) = -(Kp * e(:,i) + Ki*e_i(:,i));

    % Plant at time-step i
    B(:,:,i) = dt*[cos(x(3,i)), 0;
                   sin(x(3,i)), 0;
                             0, 1];
    % Pass state through dynamics
    x(:,i+1) = A*x(:,i) + B(:,:,i)*u(:,i);

    % calculate error
    e(:,i+1) = x(:,i+1) - [x_r(i+1);y_r(i+1);theta_r(i+1)];

    % calculate integral of error
    e_i(:,i+1) = e_i(:,i) + (e(:,i) + e(:,i+1))/2*dt;

end

figure()
p = plot(x_r,y_r,'r',x(1,:),x(2,:),'k--');
p(2).LineWidth = 1;
xlabel('x');
ylabel('y');
title('Two-Wheeled Nonholonomic Robot With PI Control');
legend('reference path', 'robot path');


figure()

subplot(3,1,1)
plot(t,e(1,:));
title('x-error');
ylabel('x(m)');

subplot(3,1,2)
plot(t,e(2,:));
title('y-error');
ylabel('y (m)');

subplot(3,1,3)
plot(t,e(3,:));
title('\theta-error');
ylabel('\theta (rad)');
xlabel('time (s)');

figure()
hold on
plot(t(1:end-3),u(1,:) + feedforward(1,1:end-1));
plot(t(1:end-3),u(2,:) + feedforward(2,1:end-1));
title('Control Inputs');
legend('Tangential Velocity','Angular Velocity');
hold off
```

# APPENDIX B

## LQR Control

```
clear
close all

% parameters
N = 1000;
t_tot = 2*pi;
dt = t_tot / N;
t = linspace(0,t_tot,N);

r = 1;
% Define path
x_r = r*cos(t);
y_r = r*sin(t);

% Generate path derivatives
x_d = diff(x_r);
y_d = diff(y_r);
x_dd = diff(x_d);
y_dd = diff(y_d);
x_d = (x_d(1:end-1) + x_d(2:end))./2;
y_d = (y_d(1:end-1) + y_d(2:end))./2;
x_r = (x_r(1:end-1) + x_r(2:end))./2;
x_r = (x_r(1:end-1) + x_r(2:end))./2;
y_r = (y_r(1:end-1) + y_r(2:end))./2;
y_r = (y_r(1:end-1) + y_r(2:end))./2;

% reference angle calculation
theta_r = atan2(y_d,x_d);
for i = 1:N-2
    if (theta_r(i) < 0)
            theta_r(i) = theta_r(i) + 2*pi;
    end
end
for i = 1:N-2
    if (theta_r(i) < pi/2)
        theta_r(i) = theta_r(i) + 2*pi;
    end
end

% Feedforward control calculation
v_r = sqrt(x_d.^2 + y_d.^2);
w_r = (x_d.*y_dd - y_d.*x_dd)./(x_d.^2 + y_d.^2);
feedforward = [v_r; w_r];

% initial conditions = reference conditions
x(:,1) = [x_r(1);
          y_r(1);
          theta_r(1)];

% State Space Linearization
A = zeros(3,3,N-4);
C = eye(3);
g1tp = -v_r.*sin(theta_r);
g2tp = v_r.*cos(theta_r);

for i = 1:N-4
```

```
    B(:,:,i) = dt*[cos(theta_r(i)), 0;
                   sin(theta_r(i)), 0;
                                   0, 1];

    A(:,:,i) = eye(3) + [0,0,g1tp(i);
                         0,0,g2tp(i);
                         0,0,     0];

    A(:,:,i)=dt*A(:,:,i)+eye(3);
end

% Feedback Gain Calculation
Q = 1*eye(3);
F = Q;
R = eye(2);
S = zeros(3,3,N-4);
S(:,:,N-4) = F;
K = zeros(2,3,N-4);

for i = 1:N-5
    S(:,:,N-4-i) = A(:,:,N-4-i)'*(S(:,:,N-3-i) - S(:,:,N-3-i)*B(:,:,N-4-i)*...
        inv(B(:,:,N-4-i)'*S(:,:,N-3-i)*B(:,:,N-4-i) + R)*B(:,:,N-4-i)'*...
        S(:,:,N-3-i))*A(:,:,N-4-i) + Q;
end

for i = 1:N-5
    K(:,:,i) = inv(B(:,:,i)'*S(:,:,i+1)*B(:,:,i) + R)*B(:,:,i)'*S(:,:,i+1)*A(:,:,i);
end

% Noise covariance
V = 10^-5*eye(3);
W = 10^-5*eye(3);

% Simulation
x_dot = zeros(3,N-4);
x_dot(:,1) = [.2;.2;.2]; % beginning error

for i = 1:N-4
    % control inputs
    u_bar(:,i) = -K(:,:,i)*x_dot(:,i);

    % Plant at time-step i
    B(:,:,i) = dt*[cos(x(3,i)), 0;
                   sin(x(3,i)), 0;
                               0, 1];

    % calculate error based on the linear system
    x_dot(:,i+1) = A(:,:,i)*x_dot(:,i) + B(:,:,i)*(u_bar(:,i) + feedforward(:,i));

    % current position
    x(:,i+1) = x_dot(:,i) + [x_r(i);y_r(i);theta_r(i)];
end

figure()
hold on
plot(x(1,:),x(2,:),'o');
plot(x_r,y_r);
hold off

figure()
subplot(3,1,1)
```

```
plot(t(1:end-3),x_dot(1,:));
title('x-error');
ylabel('x(m)');

subplot(3,1,2)
plot(t(1:end-3),x_dot(2,:));
title('y-error');
ylabel('y (m)');

subplot(3,1,3)
plot(t(1:end-3),x_dot(3,:));
title('\theta-error');
ylabel('\theta (rad)');
xlabel('time (s)');

figure()
hold on
plot(t(1:end-4),u_bar(1,:));
plot(t(1:end-4),u_bar(2,:));
title('Control Inputs');
legend('Tangential Velocity','Angular Velocity');
hold off
```

# APPENDIX C

## LQG Control

```
clear
close all

% parameters
N = 1000;
t_tot = 2*pi;
dt = t_tot / N;
t = linspace(0,t_tot,N);

r = 1;
% Define path
x_r = r*cos(t);
y_r = r*sin(t);

% Generate path derivatives
x_d = diff(x_r);
y_d = diff(y_r);
x_dd = diff(x_d);
y_dd = diff(y_d);
x_d = (x_d(1:end-1) + x_d(2:end))./2;
y_d = (y_d(1:end-1) + y_d(2:end))./2;
x_r = (x_r(1:end-1) + x_r(2:end))./2;
x_r = (x_r(1:end-1) + x_r(2:end))./2;
y_r = (y_r(1:end-1) + y_r(2:end))./2;
y_r = (y_r(1:end-1) + y_r(2:end))./2;

% Feedforward control calculation
v_r = sqrt(x_d.^2 + y_d.^2);
w_r = (x_d.*y_dd - y_d.*x_dd)./(x_d.^2 + y_d.^2);
feedforward = [v_r; w_r];

% reference angle calculation
theta_r = atan2(y_d,x_d);
for i = 1:N-2
    if (theta_r(i) < 0)
           theta_r(i) = theta_r(i) + 2*pi;
    end
end
for i = 1:N-2
    if (theta_r(i) < pi/2)
        theta_r(i) = theta_r(i) + 2*pi;
    end
end

% Linearizing the system
A = zeros(3,3,N-3);
C = eye(3);
g1tp = -v_r.*sin(theta_r);
g2tp = v_r.*cos(theta_r);

for i = 1:N-3
    B(:,:,i) = dt*[cos(theta_r(i)), 0;
                   sin(theta_r(i)), 0;
                                   0, 1];

    A(:,:,i) = eye(3) + dt*(eye(3) + [0,0,g1tp(i);
                                      0,0,g2tp(i);
                                      0,0,     0]);
end

% Observer Gain Calculation
P(:,:,:) = zeros(3,3,N-3);
L(:,:,:) = zeros(3,3,N-3);
V = 10^-4*eye(3);
W = 10^-4*eye(3);

for i = 1:N-4
    P(:,:,i+1) = A(:,:,i)*(P(:,:,i) - P(:,:,i)*C'*inv(C*P(:,:,i)*C' + W)*C*P(:,:,i))*A(:,:,i)'+ V;
```

```
    L(:,:,i) = P(:,:,i)*C'*inv(C*P(:,:,i)*C' + W
end

% Feedback Gain Calculation
Q = 1000*eye(3);
F = Q;
R = eye(2);
S = zeros(3,3,N-3);
S(:,:,N-4) = F;
K = zeros(2,3,N-3);

for i = 1:N-4
    S(:,:,N-3-i) = A(:,:,N-3-i)'*(S(:,:,N-2-i) - S(:,:,N-2-i)*B(:,:,N-3-i)*...
        inv(B(:,:,N-3-i)'*S(:,:,N-2-i)*B(:,:,N-3-i) + R)*B(:,:,N-3-i)'*...
        S(:,:,N-2-i))*A(:,:,N-3-i) + Q;
end

for i = 1:N-4
    K(:,:,i) = inv(B(:,:,i)'*S(:,:,i+1)*B(:,:,i) + R)*B(:,:,i)'*S(:,:,i+1)*A(:,:,i);
end

y_act = zeros(3,N-3);
x_est = zeros(3,N-3);
u_bar = zeros(2,N-4);
x_dot_est = zeros(3,N-3);
x_dot_act = zeros(3,N-3);
y_act(:,1) = eye(3) * x_dot_act(:,1);
y_est(:,1) = y_act(:,1);
x_dot_act(:,1) = [.2;.2;.2];
x = zeros(3,N-4);
x(:,1) = [x_r(1);y_r(1);theta_r(1)] + x_dot_act(:,1);


figure()
% Simulation Loop
for i = 1:N-4
    % control inputs based on estimator state
    u_bar(:,i) = -K(:,:,i)*x_dot_est(:,i);

    % Plant at time-step i
    B_non(:,:,i) = dt*[cos(x(3,i)), 0;
                       sin(x(3,i)), 0;
                                0, 1];
    A_non(:,:,i) = eye(3) + eye(3)*dt;

    % actual system
    x_dot_act(:,i+1) = A_non(:,:,i)*x_dot_act(:,i) + B_non(:,:,i)*(u_bar(:,i) + feedforward(:,i)) + mvnrnd([0;0;0],V)';
    y_act(:,i) = C*x_dot_act(:,i+1) + mvnrnd([0;0;0],W)';

    % estimator
    x_dot_est(:,i+1) = A(:,:,i)*x_dot_est(:,i) + B(:,:,i)*(u_bar(:,i) + feedforward(:,i))...
                          + L(:,:,i+1) * (y_act(:,i) - y_est(:,i));
    y_est(:,i+1) = C*x_dot_est(:,i+1);

    % calculate current position
    x(:,i+1) = x_dot_act(:,i) + ([x_r(i);y_r(i);theta_r(i)]);
end

figure()
hold on
plot(x(1,:),x(2,:),'o');
plot(x_r,y_r);
hold off

figure()
subplot(3,1,1)
plot(t(1:end-3),x_dot_est(1,:));
title('x-error');
ylabel('x(m)');

subplot(3,1,2)
plot(t(1:end-3),x_dot_est(2,:));
title('y-error');
```

```
ylabel('y (m)');

subplot(3,1,3)
plot(t(1:end-3),x_dot_est(3,:));
title('\theta-error');
ylabel('\theta (rad)');
xlabel('time (s)');

figure()
hold on
plot(t(1:end-4),(u_bar(1,:) + feedforward(1,1:end-2)));
plot(t(1:end-4),(u_bar(2,:) + feedforward(2,1:end-2)));
title('Control Inputs');
legend('Tangential Velocity','Angular Velocity');
hold off
```