AE353: Design Problem 04

Emilio Gordon

May 3, 2017

1 Goal

`DesignProblem04` simulates a two-wheeled robot that is similar to the Segway robotic mobility platform[1], which has been considered for use with the NASA robonaut [1]. The robot is equipped with actuators that enable control over the torque applied to each wheel. In addition to the actuators, the robot is equipped with sensors which provide information regarding the forward speed, the turning rate, and the pitch angle of the chassis. The sensors also provide information about the robots behavior on the road such as the turning radius of the road along which the robot can drive, the error in lateral position and the error in heading relative to the road.

The goal is to make the robot race along a randomly generated road at the fastest possible speed without crashing while experiencing a critical time delay. In order to verify the success of this goal, the following requirements are to be established.

- The robot must be able to complete at least five randomly generated roads to competition.
- The robot must be able to complete any road in less than 100 seconds.

With this requirement in mind, a procedure for verification is established. Verification ensures that the system meets the established requirements. Verification will be conducted through several methodologies and are as follows

- A random road will be generated and the robot must complete the track at least five consecutive times.
- The time for a robot will be recorded and validated with the requirment

2 Model

`DesignProblem04` consist of two major players that create the model, the robot segway and the road. The following section will breakdown the control systems that the controller and observer base around.

[1] See also http://www.segwayrobotics.com.

2.1 Variables

This section provides extra information about how variables are named in `DesignProblem04`. The equations of motion are used to establish the motion of the segway. The variables to the equations of motion are as follows: [2]

- ϕ, the chassis angle, called `phi` in MATLAB
- $\dot{\phi}$, the time derivative of the chassis angle, called `phidot` in MATLAB
- v, the forward speed, called `v` in MATLAB
- w, the turning rate, called `w` in MATLAB
- τ_R and τ_L, the right and left motor torques, called `tauR` and `tauL` in MATLAB

These variables can be initialized in Matlab as symbolic variables as demonstrated:

```
1 syms phi phidot v w tauR tauL real
```

There are two extra equations of motion not defined by the MATLAB code. The extra variables to these equations are as follows:

- e_{lateral}, the lateral error in following the road, called `e_lateral` in MATLAB
- e_{heading}, the heading error in following the road, called `e_heading` in MATLAB
- w_{road} the turning rate of a trajectory that would follow the road centerline, called `w_road` in MATLAB
- v_{road} the forward speed of a trajectory that would follow the road centerline, `v_road` in MATLAB

These additional variables can be initialized as follows:

```
1 syms e_lateral e_heading v_road w_road real
```

It should be noted that v_{road} and w_{road} are not states. They are however, parameters that the model depends on.

2.2 Robot

The motion of the robot is governed by ordinary differential equations with the form

$$\begin{bmatrix} \ddot{\phi} \\ \dot{v} \\ \dot{w} \end{bmatrix} = f(\phi, \dot{\phi}, v, w, \tau_R, \tau_L) \tag{1}$$

where ϕ is the pitch angle of the chassis, $\dot{\phi}$ is the pitch angular velocity, v is the forward speed, w is the turning rate, and τ_R and τ_L are the torques applied by the chassis to the right and left wheel, respectively [3].

For the purpose of control design to follow a road, it is useful to keep track of the position and orientation relative to the road as opposed to an absolute reference frame. The sensors can report the radius of curvature r_{road} of the road along which the robot is currently moving (where $r_{\text{road}} > 0$ means turning left, $r_{\text{road}} < 0$ means turning right, $r_{\text{road}} = 0$ means turning in place, and $r_{\text{road}} = \infty$ means going straight). Keeping this in mind and knowing a speed v_{road} at which the robot can travel along the road, then the turning rate w_{road} necessary to follow its centerline can be computed as

$$w_{\text{road}} = \frac{v_{\text{road}}}{r_{\text{road}}}. \tag{2}$$

Keeping this in mind, e_{lateral} can be defined as the perpendicular distance from the centerline of the road to the position (x, y) of the robot (where $e_{\text{lateral}} > 0$ means being too far to the right and $e_{\text{lateral}} < 0$ means being too far to the left), and define e_{heading} as the difference between the orientation θ of the robot and the direction of the road. It is possible to show that

$$\begin{aligned} \dot{e}_{\text{lateral}} &= -v \sin\left(e_{\text{heading}}\right) \\ \dot{e}_{\text{heading}} &= w - \left(\frac{v \cos\left(e_{\text{heading}}\right)}{v_{\text{road}} + w_{\text{road}} e_{\text{lateral}}}\right) w_{\text{road}}. \end{aligned} \tag{3}$$

A complete set of nonlinear equations of motion for the purpose of control design can be obtained from (1) and (3)—with states $e_{\text{lateral}}, e_{\text{heading}}, \phi, \dot{\phi}, v, w$ and inputs τ_R, τ_L—augmented as usual with a differential equation that describes $d(\phi)/dt$.

In the process of solving this problem, a state, input and output are defined. Shown in Equation (4), the system has six states, two inputs and five outputs. For the case of state, $\dot{\phi}$ is not included since it does not appear on the right hand side of the equation of motion in Equation (1).

$$\text{state} = \begin{bmatrix} \dot{\phi} \\ v \\ w \\ \phi \\ e_{\text{lateral}} \\ e_{\text{heading}} \end{bmatrix} \quad \text{input} = \begin{bmatrix} \tau_R \\ \tau_L \end{bmatrix} \quad \text{output} = \begin{bmatrix} v \\ w \\ \phi \\ e_{\text{lateral}} \\ e_{\text{heading}} \end{bmatrix} \tag{4}$$

Note that the output does not provide all of the state elements. As a result, an observer must be implemented to estimate the missing state value. To begin the linearization of the system, an equilibrium for the input and state must be determined. The script in Figure 1 initializes equilibrium values for θ, ϕ, $\dot{x}$ and $\dot{z}$. The equilibrium state and equilibrium input

```
eqstate = [0; .1; .1; 0; 0; 0];
eqinput = [0; 0];
state = [phidot; v; w; phi; elateral;eheading];
input = [tauR; tauL];
```

Figure 1: Equilibrium Calculation Script

for the system are defined in Equation (5).

$$eq_{\text{state}} = \begin{bmatrix} \dot{\phi} \\ v \\ w \\ \phi \\ e_{\text{lateral}} \\ e_{\text{heading}} \end{bmatrix} = \begin{bmatrix} 0 \\ 0.1 \\ 0.1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \qquad eq_{\text{input}} = \begin{bmatrix} \tau_R \\ \tau_L \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \tag{5}$$

Having acquired the equilibrium points and defined a state, input and output for the system, a linear model can now be produced. These calculations were conducted utilizing Matlab's symbolic toolbox.

```
%Define State and Input
state = [phidot; v; w; phi; elateral;eheading];
input = [tauR; tauL];
%Linearization
A = double(vpa(subs(jacobian(fsym,state),[state; input],[eqstate; eqinput
    ])));
B = double(vpa(subs(jacobian(fsym,input),[state; input],[eqstate; eqinput
    ])));
C = [0 1 0 0 0 0;
    0 0 1 0 0 0;
    0 0 0 1 0 0;
    0 0 0 0 1 0;
    0 0 0 0 0 1];
```

Figure 2: Symbolic Linearization Script

The results to the section of code in Figure 2 are shown in Equation Set (6) which contain the matrices A, B and C for the state-space model. The D matrix is simply 0 since the output does not rely on the input of the system.

$$A = \begin{bmatrix} 0 & 0 & 0 & 27.4517870326773 & 0 & 0 \\ 0 & 0 & 0 & -3.45386373246069 & 0 & 0 \\ 0 & 0 & 0 - 0.0508186860006732 & 0 & 0 & \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -0.1 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}$$

$$B = \begin{bmatrix} -2.22551849308104 & -2.22551849308104 \\ 0.587386021655837 & 0.587386021655837 \\ 2.48138115237662 & -2.48138115237662 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \tag{6}$$

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

The resulting state-space model is

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

such that the behavior of this linear system and of the original, nonlinear system will be approximately the same so long as the system is close to equilibrium.

3 Controller and Observer

The system is of a two wheeled robot model that can drive on a randomly generated path without tipping or going off track. In the previous section, a space-state model was derived by computing linear model around a set of equilibrium points for the equations of motion described in equation (1). To obtain the desired performance, it is required to test for controllability and observability. This is critical for the analysis of a system before deciding the best control strategy to be applied, or whether it is even possible to control or stabilize the system.

3.1 Controller

Controllability refers to a systems ability to reach a particular state through an appropriate control input. In designing a controller, the system must be verified for controllability and

stability. If a state is controllable, then the system can be told to reach a certain state and is considered stable. However, if the system is not controllable, no input will be able to control the state. Even if the state is not controllable, the dynamics may still be stable, resulting in a stable state. The MATLAB code in Figure 3 completes these objectives.

```
1  %Verfies System is Controllable
2  abs(rank(ctrb(A,B))-length(A))
3
4  %Define Gains Controller
5  Qc = 500*[1 0 0 0 0 0;0 15 0 0 0 0;0 0 1 0 0 0;0 0 0 1 0 0;0 0 0 0 10 0;0
       0 0 0 0 10];
6  Rc = [1 0;0 1];
7  K = lqr(A,B,Qc,Rc);
8
9  %Verifies if Controller is Asymptoticly Stable
10 eig(A-B*K)
```

Figure 3: Controller

The code insures the system is controllable by validating controllability and stability. Controllability is verified in the second line of code in Figure 3. The output of this line results can be either

- 0: The rank is equivalent to the size of the A matrix and results in 0, confirming controllability.
- 1: The rank is not equivalent to the size of the A matrix and results in 1, disproving controllability.

After proving controllability, a linear feedback control design based on Quadratic Regulators (LQR) is implemented. Implementing LQR introduces a Q and R matrix. The Q matrix represents the cost on the state while the R matrix is the cost on the action. The selected Q and R matrices are represented in Equation (7). Through LQR, a K matrix can be found as shown in line seven of Figure 3.

$$Q_c = \begin{bmatrix} 500 & 0 & 0 & 0 & 0 & 0 \\ 0 & 7500 & 0 & 0 & 0 & 0 \\ 0 & 0 & 500 & 0 & 0 & 0 \\ 0 & 0 & 0 & 500 & 0 & 0 \\ 0 & 0 & 0 & 0 & 50000 & 0 \\ 0 & 0 & 0 & 0 & 0 & 50000 \end{bmatrix} \qquad R_c = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{7}$$

Finally, with the controllability verified and gain defined, controller stability must now be computed. Stability is verified in line 10 of code in Figure 3. The eigenvectors of the matrix

computation, $A - BK$, must have negative real parts.

$$\text{eig(A-BK)} = \begin{bmatrix} -100.84 + 0.0000i \\ -2.99 + 1.15i \\ -2.99 - 1.15i \\ -77.82 + 0.0000i \\ -9.97 + 0.0000i \\ -1.52 + 0.0000i \end{bmatrix} \tag{8}$$

Since Equation (8) has negative real parts, stability is confirmed and the controlled is complete.

3.2 Observer

Observability relates to the possibility of observing the state of a system, via output sensor measurements. In designing an observer, the system must be verified for observability and stability, similar to designing a controller. If a state is not observable, the controller will never be able to predict the behavior of an unobservable state and therefore, is unable to be stabilized. However, a system that is not observable may still be stable. The code insures the

```
%Verifies System is Observable
rank(obsv(A,C))-length(A)

%Define Gains Observer
Ro = 10*[1 0 0 0 0 0;0 1 0 0 0 0;0 0 10 0 0 0;0 0 0 1 0 0;0 0 0 0 1 0;0 0
    0 0 0 1];
Qo = [1 0 0 0 0;0 1 0 0 0;0 0 10 0 0;0 0 0 10 0;0 0 0 0 10];
L = lqr(A',C',inv(Ro),inv(Qo))';

%Verifies if Observer is Asymptoticly Stable
eig(A-L*C)
```

Figure 4: Observer

possibility of an observer by validating observability and stability. Observability is verified in the second line of code in Figure 4. The output of this line results can be either

- 0: The rank is equivalent to the size of the A matrix and results in 0, confirming observability.
- 1: The rank is not equivalent to the size of the A matrix and results in 1, disproving observability.

After proving observability, a linear feedback control design based on Quadratic Regulators (LQR) is implemented much like what was done for the controller. Implementing LQR introduces a Q and R matrix where the Q matrix represents the cost on the state and the R matrix represents the cost on the action. The selected Q and R matrices are represented in Equation (9). Through LQR, an L matrix can be found as shown in line seven of Figure 4.

$$R_o = \begin{bmatrix} 100 & 0 & 0 & 0 & 0 & 0 \\ 0 & 10 & 0 & 0 & 0 & 0 \\ 0 & 0 & 100 & 0 & 0 & 0 \\ 0 & 0 & 0 & 10 & 0 & 0 \\ 0 & 0 & 0 & 0 & 10 & 0 \\ 0 & 0 & 0 & 0 & 0 & 10 \end{bmatrix} \qquad Q_o = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 10 & 0 & 0 \\ 0 & 0 & 0 & 10 & 0 \\ 0 & 0 & 0 & 0 & 10 \end{bmatrix} \tag{9}$$

Finally, with the observability verified and a gain defined, controller stability must now be computed. Stability is verified in line 10 of code in Figure 4. The eigenvectors of the matrix computation, $A - LC$, must have negative real parts.

$$\text{eig(A-LC)} = \begin{bmatrix} -5.7516 + 0.0000i \\ -4.7754 + 0.0000i \\ -0.3162 + 0.0000i \\ -0.3366 + 0.0000i \\ -1.1672 + 0.6434i \\ -1.1672 - 0.6434i \end{bmatrix} \tag{10}$$

Since Equation (10) has negative real parts, stability is confirmed and the observer is complete.

4 Implementation

With controllability and observability successfully verified, a controller can be implemented into the system. This section will go into detail the setup of the controller. First, a Zero-Input controller will be demonstrated to serve as control data set, a comparison for future test. Second, the controller will be demonstrated and explained. Finally, a comparison between the two will be made.

4.1 Zero-Input System

A zero-input system has no inputs. This means the value being put into the actuators is simply zero. No force or actuation is occurring so the system will behave in its most natural sense. This is an important observation to keep in mind, especially after the controller is applied. It is expected that the controller will operate more efficiently than the zero-input

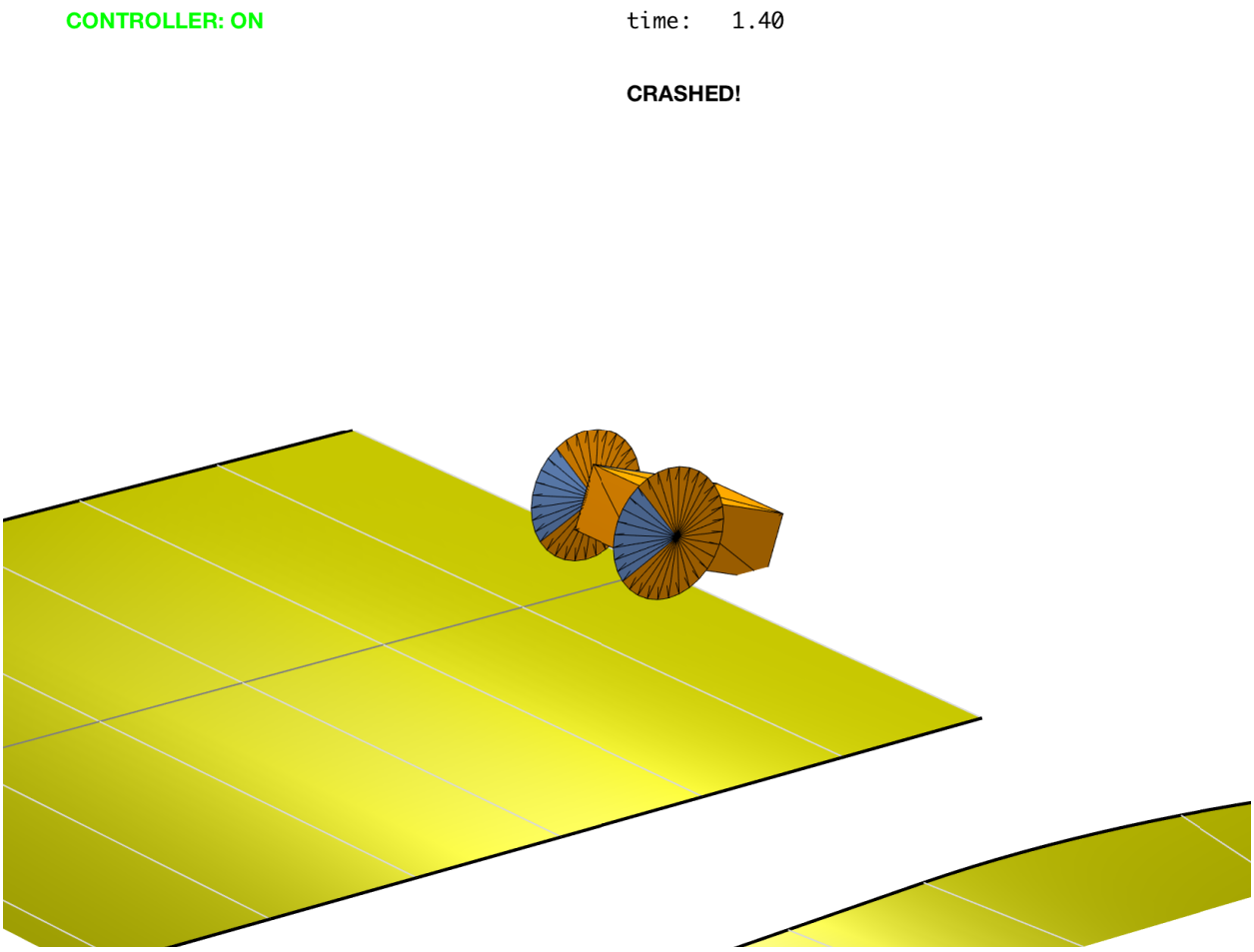

Figure 5: Flight Path and Simulated Model with Zero Input

system and therefore, makes a good basis for how well the controller behaves.

```
actuators.tauR = 0;
actuators.tauL = 0;
```

As can be seen with Figure 5, the glider carried out with the initial random velocity it was assigned and proceeded to plummet to a vertical height of zero.

4.2 Closed Loop System

It was seen that through a zero-input system, the model will behave under no forces or dynamic changes. In the following section, a controller using state-feedback and state-estimation is implemented. The sensors that are accessible are as follows:

- v, the forward speed, called `v` in MATLAB
- w, the turning rate, called `w` in MATLAB

- ϕ, the chassis angle, called `phi` in MATLAB
- e_{lateral}, the lateral error in following the road, called `e_lateral` in MATLAB
- e_{heading}, the heading error in following the road, called `e_heading` in MATLAB

Note that the defined state in Equation (4) has a total of six states however, only five states can be collected from the robot's sensors. To resolve this, an observer is created in order to generate a state estimate that would closely represent the missing data.

Since C was defined as a $5x6$ matrix, it is important to define the output as the five states the system can contribute.

```
%Define Sensor Values
v = sensors.v;
w = sensors.w;
phi = sensors.phi;
elateral = sensors.e_lateral;
eheading = sensors.e_heading;
%Establish Output
output = [v; w; phi; elateral; eheading];
```

Using the sensor values, the output of the system must be corrected in terms of the equilibrium values.

```
%WHAT?
y = output - C*[eqstate];
```

Now that the output of the system has been defined, focus is placed on the input. The input, as for any closed-loop system is simply

```
%Input using estimated state
u =-K*xhat;
```

$\hat{x}$ which represents the state estimation, was previously defined as $[0.015; 2; 0; 0; 0; 0]$ but is subject to change after the first iteration. To accomplish this the change in state estimation, $\dot{\hat{x}}$ is computed.

```
%Next step for dxhat
dxhat = A*xhat+B*u-L*(C*xhat-y);
```

With $\dot{\hat{x}}$ now known, Euler's method can be used to find $\hat{x}$, the estimate for the state. This is later applied in the next iteration affecting the input.

```
%Euler's Method for finding xhat
data.xhat = xhat + h*dxhat;
```

With all these calculation made, an input can confidently be implemented into the actuator. The actuators must be corrected by adding the equilibrium input. For the case of this system, the correction factor is simply zero.

```
%Applying controller to the input
actuators.tauR = u(1);
actuators.tauL = u(2);
```

The gains for this controller are the same as the gains provided in the Controller and Observer section of this report. Through this, the robot was simulated on five different roads which produced the results shown in Table 1 With these results, both of the requirements set in

Road	Time (sec)
Road 1	91.14
Road 2	90.10
Road 3	92.98
Road 4	90.78
Road 5	90.48

Table 1: Road completion times for closed loop.

the Goal section have been satisfied. This controller works on any road but has limitations. The closed loop controller is not capable of reaching speeds of higher than 1.5. Any speed exceeding 1.5 will result in the robot immediately crashing. This is possibly an issue with the gains that can be remedied but has not been proven. Despite the incompetence with higher speeds, the robot is capable of completing any road configuration. The full code for this controller can be found in the Listing 1.

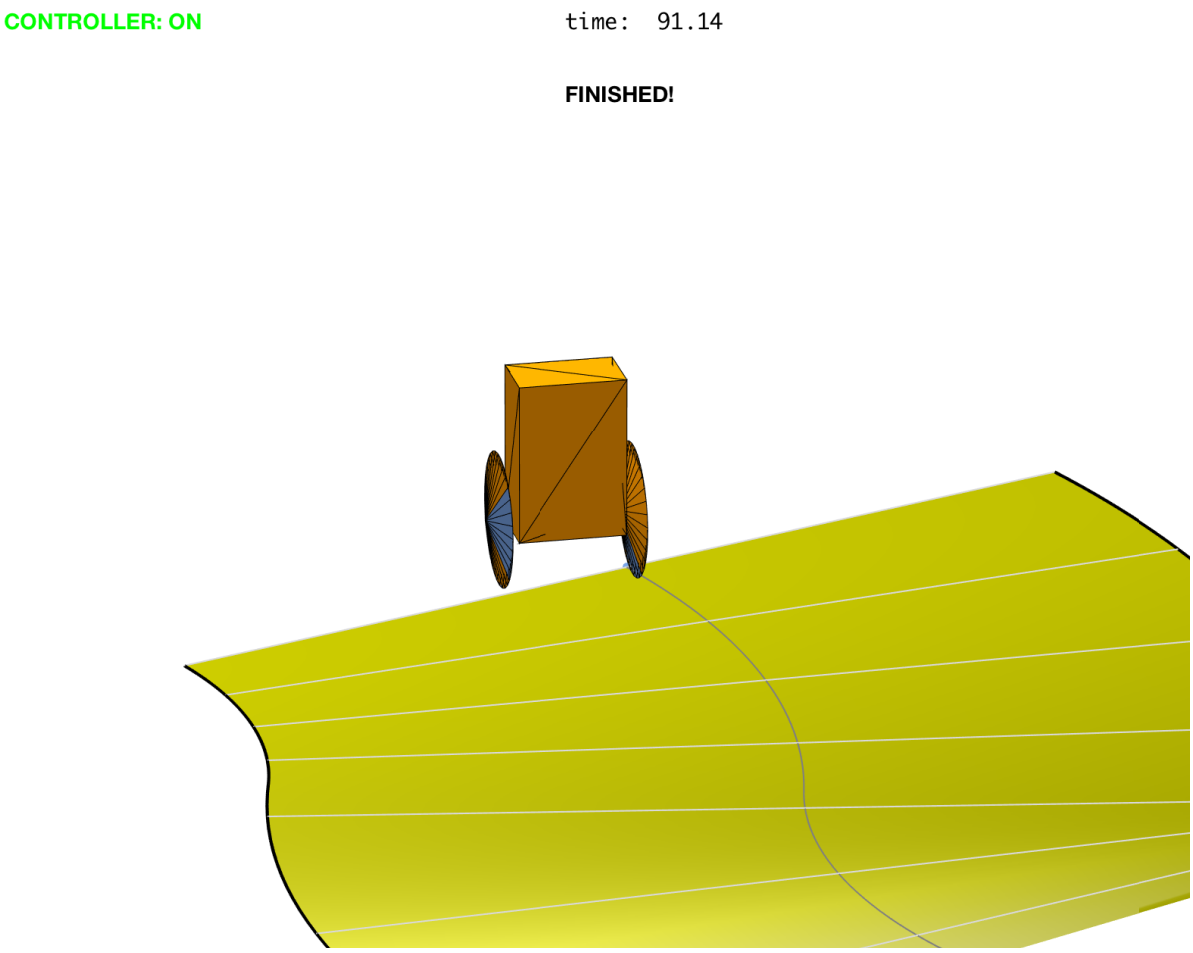

Figure 6: Robot finishing Road 1 in 91.14 seconds

4.3 Gain Scheduling

A gain-scheduled controller is a controller whose gains are automatically adjusted as a function of time. Gain scheduling is an efficient controller to implement for problems where the model is dynamic, such is the case for the segway robot. To implement gain scheduling there are two approaches. The first is to generate a library of gains corresponding to a range of equilibrium and using the equilibrium that corresponds most to the current state of the model. The second approach is to generate a new linearized model based on the current state model. The second approach will be implemented.

To begin this, the A and B matrix must be implemented as functions of whatever sensor variable the model should linearize about. For the purposes of this controller, the turning rate (w), was selected to be what the model linearizes about. In addition, was set to 0.015 and v was set to 2 while all other states were 0.

```
syms W real
%Equilibrium Points
xhat = [0; 0; 0; 0; 0; 0];
state = [phidot; v; w; phi; elateral; eheading];
eq_state = [0.015; 2; W; 0; 0; 0];
input = [tauR; tauL];
%Linearization
A = subs(jacobian(fsym,state),[state; input],[[0.015; 2; W; 0; 0; 0]; [0; 0]])
B = subs(jacobian(fsym,input),[state; input],[[0.015; 2; W; 0; 0; 0]; [0; 0]])
C = [0 1 0 0 0 0; 0 0 1 0 0 0; 0 0 0 1 0 0; 0 0 0 0 1 0; 0 0 0 0 0 1;]
```

Doing this establishes the A and B matrices as functions of W, the turning radius. It is discovered that B has no symbolic terms with respect to W and therefore is remains constant over time.

```
%Define Variable
W = sensors.w;
%Compute A with known W value
A = data.funcA(W);
B = data.B;
```

There is now a model for the system at the specific iteration. At every iteration a new model is generated linearized about the new state. Once this has been achieved, the controller is implemented the exact same way as it was for the closed loop controller. Through this, the robot was simulated on the same five different roads used for the earlier controller which produced the results shown in Table 2 As can be seen, this controller is much more optimal than the closed loop controller. It can complete each road approximately 20seconds faster than the closed loop controller, when the equilibrium velocity is set to two. With this model, the velocity can be set to even higher values. The tradeoff however is less stability in terms of e_{heading} and e_{lateral}. The robot was guaranteed to never tip but was vulnerable to veering off the road. The full code for this controller can be found in the Listing 1.

Road	Time (sec)
Road 1	71.66
Road 2	71.16
Road 3	70.36
Road 4	71.50
Road 5	71.54

Table 2: Road completion times.

5 Testing

Through the process of creating a controller, many errors came about. Overall, there are two ways for the robot to crash, tipping over and going off the road.
If the robot is at fault for crash and is tipping or turning into a crash then the issue can be debugged by considering ϕ or w.

- If the robot is tipping, focus should be placed on ϕ (chassis angle) since the gains are not enforcing a desired reaction.
- If the robot is turning into a crash, focus should be placed on w (turning rate) since the gains are not enforcing a desired reaction.
- In either case, forward velocity (v) may also be the culprit. At faster speeds, the segway has less control over the ϕ and w.

If the robot is veering off road then the issue can be debugged by considering the gains placed on e_{heading} or e_{lateral}.

- If the robot veers off road but points in the direction of the road, focus should be placed on e_{lateral} since the robot would appear to have slid off the road.
- If the robot goes off the road but points in a direction other than the road, focus should be placed on e_{heading} since the robot would appear to have turned off the road.
- In either case, forward velocity may also be the culprit. At faster speeds, the segway will stray farther from e_{heading} or e_{lateral}.

With this said, failure occurs most often on roads with an excessive amounts of sharp turns. Often times, a sharp turn will need recovery time to stabilize the e_{heading} and e_{lateral} but if there are multiple sharp turns at once, this stabilization period is shortened and results in veering off the road/crashing.

5.1 Manipulating Gains

In the following section, three closed loop controllers will be compared. The controllers are exactly the same however each possess unique gains for the controller and observer. The controllers are named as follows:

- Controller One - The original Closed Loop Controller discussed in this report.
- Controller Two - A controller in which the gains are simply the corresponding identity matrix.
- Controller Three - A controller with gains slightly lowered from Controller One.

The gains defined have a significant impact on the system. By changing the gains, an optimal controller can be created so it is important to test a range of gains.

5.1.1 Controller One Gains

The gains set to the controller section of Controller One are as follows:

$$\mathrm{Q}_c = \begin{pmatrix} 500 & 0 & 0 & 0 & 0 & 0 \\ 0 & 7500 & 0 & 0 & 0 & 0 \\ 0 & 0 & 500 & 0 & 0 & 0 \\ 0 & 0 & 0 & 500 & 0 & 0 \\ 0 & 0 & 0 & 0 & 50000 & 0 \\ 0 & 0 & 0 & 0 & 0 & 50000 \end{pmatrix} \qquad \mathrm{R}_c = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

The gains set to the observer section of Controller One are as follows:

$$\mathrm{R}_o = \begin{pmatrix} 100 & 0 & 0 & 0 & 0 & 0 \\ 0 & 10 & 0 & 0 & 0 & 0 \\ 0 & 0 & 100 & 0 & 0 & 0 \\ 0 & 0 & 0 & 10 & 0 & 0 \\ 0 & 0 & 0 & 0 & 10 & 0 \\ 0 & 0 & 0 & 0 & 0 & 10 \end{pmatrix} \qquad \mathrm{Q}_o = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 10 & 0 & 0 \\ 0 & 0 & 0 & 10 & 0 \\ 0 & 0 & 0 & 0 & 10 \end{pmatrix}$$

5.1.2 Controller Two Gains

The gains set to the controller section of Controller Two are as follows:

$$\mathrm{Q}_c = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \qquad \mathrm{R}_c = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

The gains set to the observer section of Controller Two are as follows:

$$\mathrm{R}_o = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \qquad \mathrm{Q}_o = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

5.1.3 Controller Three Gains

The gains set to the controller section of Controller Three are as follows:

$$\mathrm{Q}_c = \begin{pmatrix} 500 & 0 & 0 & 0 & 0 & 0 \\ 0 & 5000 & 0 & 0 & 0 & 0 \\ 0 & 0 & 500 & 0 & 0 & 0 \\ 0 & 0 & 0 & 500 & 0 & 0 \\ 0 & 0 & 0 & 0 & 5000 & 0 \\ 0 & 0 & 0 & 0 & 0 & 5000 \end{pmatrix} \qquad \mathrm{R}_c = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

The gains set to the observer section of Controller Three are as follows:

$$\mathrm{R}_o = \begin{pmatrix} 1000 & 0 & 0 & 0 & 0 & 0 \\ 0 & 10 & 0 & 0 & 0 & 0 \\ 0 & 0 & 100 & 0 & 0 & 0 \\ 0 & 0 & 0 & 10 & 0 & 0 \\ 0 & 0 & 0 & 0 & 10 & 0 \\ 0 & 0 & 0 & 0 & 0 & 10 \end{pmatrix} \qquad \mathrm{Q}_o = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 10 & 0 & 0 \\ 0 & 0 & 0 & 10 & 0 \\ 0 & 0 & 0 & 0 & 10 \end{pmatrix}$$

5.1.4 Analysis

Each controller behaves in its own way. Controller Two immediately crashes showing the importance of properly defining gains. Controller Two was an example of a poorly designed gain. On the other hand, Controller One and Controller Three are fairly similar so their results are expected to be similar. It can be seen in Figure 7 that Controller One and Controller Three are in fact very similar however, Controller One outperforms Controller Three. This should be expected. Controller One uses the gains established in the report and were the most effect Gain options when designing. Controller Three is a dampened version of Controller One meaning it penalizes its variables less severely. As a result, it took wider turns, traveled at a lower speed and was consistently at a large e_{heading} and e_{lateral} nearing the edge of the road. Despite this, the controllers were still fairly close in the end.

```
  one :   91.18 seconds :   120.0 meters : finished
three :   92.94 seconds :   120.0 meters : finished
  two :    0.46 seconds :     0.3 meters : crashed
```

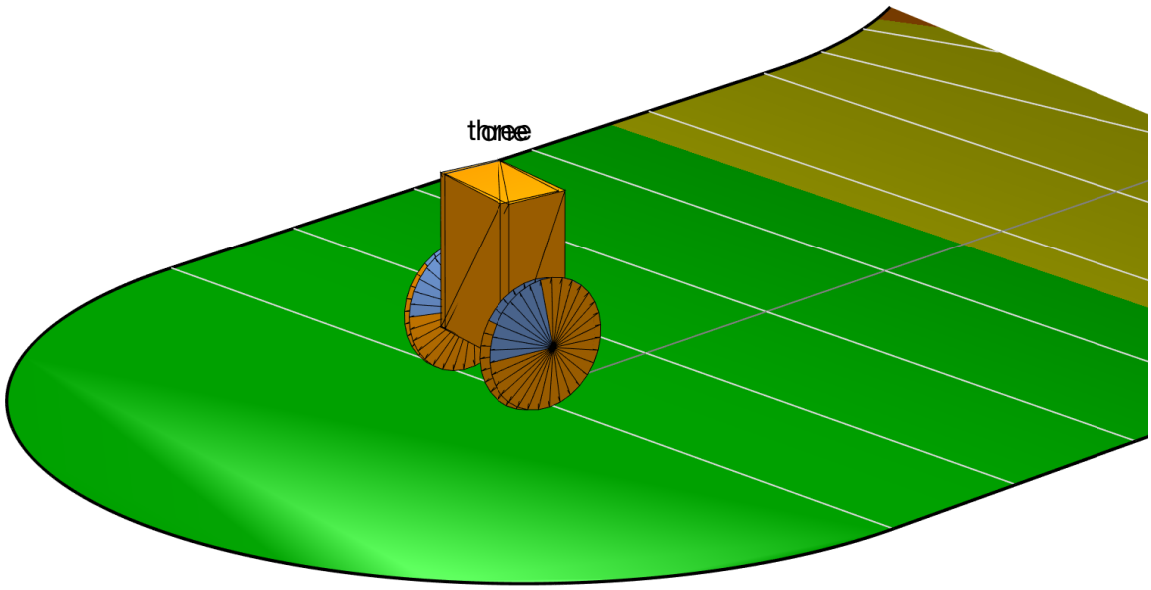

Figure 7: Three Controllers Racing on a road.

6 Code

The following section contains the complete code for the various controllers documented in this report.

6.1 Closed Loop

Listing 1: Closed Loop Code

```
function func = Controller

func.init = @initControlSystem;
func.run = @runControlSystem;
end

%
% STEP #1: Modify, but do NOT rename, this function. It is called once,
% before the simulation loop starts.
%

function [actuators,data] = initControlSystem(sensors,references,parameters,data)
% Load the equations of motion.
load('DesignProblem04_EOMs.mat');
% Parse the equations of motion.
f = symEOM.f;
% Define symbolic variables that appear in the equations of motion.
syms phi phidot v w tauR tauL elateral eheading vroad wroad

%Choose State
vroad = 10
wroad = vroad/(sensors.r_road)
fsym=[f;phidot; -v*sin(eheading); w-((v*cos(eheading))/(vroad+wroad*elateral))*wroad
    ];

%Equilibrium Points
xhat = [0; 0; 0; 0; 0; 0];
eqstate = [0; 1.5; 0; 0; 0; 0];
eqinput = [0; 0];
state = [phidot; v; w; phi; elateral;eheading];
input = [tauR; tauL];
%Linearization
A = double([vpa(subs(jacobian(fsym,state),[state; input],[eqstate; eqinput]))]);
B = double(vpa(subs(jacobian(fsym,input),[state; input],[eqstate; eqinput])));
C = [0 1 0 0 0 0;
    0 0 1 0 0 0;
    0 0 0 1 0 0;
    0 0 0 0 1 0;
    0 0 0 0 0 1];
```

```
39
40 % Design an optimal controller with reference tracking
41 %Define Gains Controller
42 Qc = 500*[1 0 0 0 0 0;0 15 0 0 0 0;0 0 1 0 0 0;0 0 0 1 0 0;0 0 0 0 100 0;0 0 0 0 0
       100];
43 Rc = [1 0;0 1];
44 K = lqr(A,B,Qc,Rc);
45 mat2str(K)
46 eig(A-B*K)
47 rank(ctrb(A,B))-length(A)
48
49 %Define Gains Observer
50 Ro = 10*[10 0 0 0 0 0;0 1 0 0 0 0;0 0 10 0 0 0;0 0 0 1 0 0;0 0 0 0 1 0;0 0 0 0 0 1];
51 Qo = [1 0 0 0 0;0 1 0 0 0;0 0 10 0 0;0 0 0 10 0;0 0 0 0 10];
52 L = lqr(A',C',inv(Ro),inv(Qo))';
53 eig(A-L*C)
54 rank(obsv(A,C))-length(A)
55
56 %Saving Variables
57 data.A = A;
58 data.B = B;
59 data.C = C;
60 data.K = K;
61 data.L = L;
62 data.h = parameters.tStep;
63 data.eqstate = eqstate;
64 data.eqinput = eqinput;
65 data.xhat = xhat;
66
67 [actuators,data] = runControlSystem(sensors,references,parameters,data);
68 end
69
70 %
71 % STEP #2: Modify, but do NOT rename, this function. It is called every
72 % time through the simulation loop.
73 %
74
75 function [actuators,data] = runControlSystem(sensors,references,parameters,data)
76 A = data.A;
77 B = data.B;
78 C = data.C;
79 K = data.K;
80 L = data.L;
81 h = data.h;
82 xhat = data.xhat;
83 eqstate = data.eqstate;
84 eqinput = data.eqinput;
```

```
85
86  output = [sensors.v; sensors.w; sensors.phi; sensors.e_lateral;sensors.e_heading];
87
88  y = output - C*[eqstate];%error
89  u =-K*xhat;%input
90  dxhat = A*xhat+B*u-L*(C*xhat-y);%this is obersever. State Estimate
91  data.xhat = xhat + h*dxhat; % Next step for state estimate.
92
93  actuators.tauR = u(1);
94  actuators.tauL = u(2);
95  end
```

6.2 Gain Scheduling - "In Loop Design"

Listing 2: Closed Loop Code

```
function func = Controller
func.init = @initControlSystem;
func.run = @runControlSystem;
end

%
% STEP #1: Modify, but do NOT rename, this function. It is called once,
% before the simulation loop starts.
%

function [actuators,data] = initControlSystem(sensors,references,parameters,data)
load('DesignProblem04_EOMs.mat');
syms phi phidot v w tauR tauL elateral eheading vroad wroad
f = symEOM.f;

vroad = 10
wroad = vroad/(sensors.r_road*0.5)
fsym=[f;phidot; -v*sin(eheading); w-((v*cos(eheading))/(vroad+wroad*elateral))*wroad
    ];

syms V W TAUR TAUL real
%Equilibrium Points
xhat = [0; 0; 0; 0; 0; 0];
state = [phidot; v; w; phi; elateral;eheading];
input = [tauR; tauL];
%Linearization
A = subs(jacobian(fsym,state),[state; input],[[0.015; 2; W; 0; 0; 0]; [0; 0]])
B = subs(jacobian(fsym,input),[state; input],[[0.015; 2; W; 0; 0; 0]; [0; 0]])
C = [0 1 0 0 0 0; 0 0 1 0 0 0; 0 0 0 1 0 0; 0 0 0 0 1 0; 0 0 0 0 0 1;]


%Saving Variables
data.funcA = matlabFunction(A)
data.funcB = matlabFunction(B);
data.C = C;
data.h = parameters.tStep;
data.xhat = xhat;

[actuators,data] = runControlSystem(sensors,references,parameters,data);
end

%
% STEP #2: Modify, but do NOT rename, this function. It is called every
% time through the simulation loop.
```

```
%

function [actuators,data] = runControlSystem(sensors,references,parameters,data)
V = sensors.v;
W = sensors.w;

eqstate = [0.015; 2; W; 0; 0; 0]

A = data.funcA(W);
B = data.funcB();

C = data.C;
h = data.h;
xhat = data.xhat;
%state = [phidot; v; w; phi; elateral;eheading];

% Design an optimal controller with reference tracking
%Define Gains Controller
Qc = 500*[1 0 0 0 0 0;0 15 0 0 0 0;0 0 1 0 0 0;0 0 0 1 0 0;0 0 0 0 1 0;0 0 0 0 0 1];
Rc = [1 0;0 1];
K = lqr(A,B,Qc,Rc);

%Define Gains Observer
Ro = 10*[100 0 0 0 0 0;0 1 0 0 0 0;0 0 1 0 0 0;0 0 0 1 0 0;0 0 0 0 1 0;0 0 0 0 0 1];
Qo = [1 0 0 0 0;0 1 0 0 0;0 0 100 0 0;0 0 0 100 0;0 0 0 0 100];
L = lqr(A',C',inv(Ro),inv(Qo))';

output = [sensors.v; sensors.w; sensors.phi; sensors.e_lateral;sensors.e_heading];

y = output - C*[eqstate];%error
u =-K*xhat;%input
dxhat = A*xhat+B*u-L*(C*xhat-y);%this is obersever. State Estimate
data.xhat = xhat + h*dxhat; % Next step for state estimate.

actuators.tauR = u(1);
actuators.tauL = u(2);

end
```

References

[1] K. J. Aström and R. M. Murray. *Feedback Systems: An Introduction for Scientists and Engineers* Princeton University Press, 2010
http://www.cds.caltech.edu/ murray/amwiki/index.php/First$_E$*dition*

[2] T. Bretl. *Design Problem 4 Description* UIUC, 2017
https://piazza-resources.s3.amazonaws.com/iy352lpu44t1zm/j1pjoc8j474q3/DesignProblem0

[3] Y. X. Mak *Realization of mini segway robot using NI MyRIO* University of Twente, 2015
http://purl.utwente.nl/essays/67004