AE 353: Design Problem 04

Parthiv V Kukadia

6th December, 2020

This paper will describe the design, implementation, and testing of a controller for a 3D model of one of NASA's Robonauts; similar to a segway robotic mobility platform [1]. The Robonaut will attempt to race along randomly generated path in the least amount of time without crashing using the controller.

1 Goal

DesignProblem04 simulates a two-wheeled robot that is similar to the segway robotic mobility platform1, which has been considered for use with the NASA robonaut. The robonaut is equipped with actuators which allow me to specify the torque applied to each wheel. It also has sensors which allow me to measure the angular velocity of each wheel, the turning radius of a road along which the robot can drive, the error in lateral position, and the error in heading relative to this road. Multiple controllers will be designed to test out which controller is most effective in driving along a 120m long randomly generated road. The goal is to design a controller that will allow the robonaut to race along this road as fast as possible without crashing.

2 Requirements and Verification

Requirement for a successful controller: 80% finish rate racing on 5 randomly generated roads in under 60s.

Verification will be done via the MATLAB DesignProblem04 simulator. The controllers will be run through 5 randomly generated roads and they will have to complete the track successfully. The results will be tabulated and displayed. If the controller has 4 finishes out of the 5 races in under 60 s, the controller will have met the requirement.

3 Model

3.1 Variables

This section provides information about how variables are named in `DesignProblem04`. The equations of motion are used to establish the motion of the robonaut. The variables to the equations of motion are:

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

These variables were initialized in Matlab as symbolic variables using `syms`.
There are two extra equations of motion, and the variables to these equations are:

- e_{lateral}, the lateral error in following the road, elateral

- e_{heading}, the heading error in following the road, eheading
- w_{road} the turning rate of a trajectory that would follow the road centerline, wroad
- v_{road} the forward speed of a trajectory that would follow the road centerline, vroad

These variables were also initialized using `syms`. It should be brought to readers attention that v_{road} and w_{road} are not states, they are parameters that the model depends on.

3.2 Robonaut

The motion of the robonaut is governed by the second order 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 $_R$ and $_L$ are the torques applied by the chassis to the right and left wheel, respectively.

For the purpose of my 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 $r_{\text{road}} < 0$ instructs the robonaut to turn right, $r_{\text{road}} < 0$ instructs the robonaut to turn left, $r_{\text{road}} = 0$ instructs the robonaut to turn in its place to re-orient itself, and $r_{\text{road}} = \infty$ instructs the robonaut to go straight). Knowing a speed, v_{road}, at which the robot can travel along the road, the turning rate w_{road} necessary to follow the centerline can be computed as

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

Two more equations are useful to me:

$$\dot{e}_{\text{lateral}} = -v \sin(e_{\text{heading}}) \tag{3}$$

$$\dot{e}_{\text{heading}} = w - \left(\frac{v \cos(e_{\text{heading}})}{v_{\text{road}} + w_{\text{road}} e_{\text{lateral}}} \right) w_{\text{road}} \tag{4}$$

where $e_{lateral}$ is the perpendicular distance from the centerline of the road to the position $(x; y)$ of the robot (where $e_{lateral} > 0$ means being too far to the right and $e_{lateral} < 0$ means being too far to the left), and $e_{heading}$ is the difference between the orientation θ of the robot and the direction of the road.

The linear state space model is represented by two equations:

$$\dot{x} = Ax + Bu \tag{5}$$

$$y = Cx + Du \tag{6}$$

In this design problem, there is no feed-forward so the D matrix in Eq.(6) is $D = [0]$. The EOM's are used to define the state, x, and input, u. The robonaut's sensors provide the information to form the output, y, with the following derived relationships between the angular velocity of the robonaut's wheels, speed, and turning rate:

$$w_R = \frac{1}{r}v + \frac{b}{2r}w \qquad\qquad w_L = \frac{1}{r}v - \frac{b}{2r}w \tag{7}$$

Therefore to begin linearizing the non-linear equations of motion, we chose an equilibrium point. The function f in Eq. (1) was run through DesignProblem04 and stored in a data.mat file symbolically. The state variables of interest, input vector, and output vector are defined from the variables of the parsed EOMs, and an initial guess was made about the values of these variables to determine the equilibrium points.

$$x = \begin{bmatrix} \phi - \phi_{\mathrm{e}} \\ \dot{\phi} - \dot{\phi}_{\mathrm{e}} \\ v - v_{\mathrm{e}} \\ w - w_{\mathrm{e}} \\ e_{\mathrm{lateral}} - e_{\mathrm{lateral\text{-}e}} \\ e_{\mathrm{heading}} - e_{\mathrm{heading\text{-}e}} \end{bmatrix} \qquad u = \begin{bmatrix} \tau_R - \tau_{R-\mathrm{e}} \\ \tau_L - \tau_{L-\mathrm{e}} \end{bmatrix} \qquad y = \begin{bmatrix} w_R - w_{R-\mathrm{e}} \\ w_L - w_{L-\mathrm{e}} \\ e_{\mathrm{lateral}} - e_{\mathrm{lateral\text{-}e}} \\ e_{\mathrm{heading}} - e_{\mathrm{heading\text{-}e}} \end{bmatrix} \tag{8}$$

$$x_{equilibrium} = \begin{bmatrix} 0 \\ 0 \\ 6 \\ 0 \\ 0 \\ 0 \end{bmatrix} \qquad u_{equilibrium} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

The chosen values for equilibrium in Eq.((8)) were such that there would be no movement away from the center line, so that the robonaut would move in a straight line whilst remaining upright. As the robonaut is moving forward, v_{e} can be a varying positive number, so I chose an arbitrary value. All other values were set to 0 to allow a forward moving straight line motion.

The equilibrium values were then used in the Jacobian function to produce state space matrices A, B, and C. These matrices were computed as partial derivatives of f and y with respect to x and u evaluated around the equilibrium points. The matrices are represented by

$$A = \left.\frac{\partial f}{\partial x}\right|_{(x_{\mathrm{e}}, u_{\mathrm{e}})} \qquad B = \left.\frac{\partial f}{\partial u}\right|_{(x_{\mathrm{e}}, u_{\mathrm{e}})} \qquad C = \left.\frac{\partial y}{\partial x}\right|_{(x_{\mathrm{e}}, u_{\mathrm{e}})} \tag{9}$$

Providing me with the following state space matrices:

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 27.4427 & 0 & 0 & 0 & 0 & 0 \\ -3.4540 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -6 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix} \qquad B = \begin{bmatrix} 0 & 0 \\ -2.2255 & -2.2255 \\ 0.5874 & 0.5874 \\ 2.4814 & -2.4814 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}$$

$$C = \begin{bmatrix} 0 & 0 & 5 & 1 & 0 & 0 \\ 0 & 0 & 5 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \qquad D = \begin{bmatrix} 0 \end{bmatrix}$$

This provides us with the following state space equations:

$$\begin{bmatrix} \dot{phi} \\ \ddot{\phi} \\ \dot{v} \\ \dot{w} \\ \dot{e}_{\mathrm{lateral}} \\ \dot{e}_{\mathrm{heading}} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 27.4427 & 0 & 0 & 0 & 0 & 0 \\ -3.4540 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -6 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} \phi \\ \dot{\phi} \\ v \\ w \\ e_{\mathrm{lateral}} \\ e_{\mathrm{heading}} \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ -2.2255 & -2.2255 \\ 0.5874 & 0.5874 \\ 2.4814 & -2.4814 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \tau_R \\ \tau_L \end{bmatrix}$$

and output

$$y = \begin{bmatrix} w_R \\ w_L \\ e_{\mathrm{lateral}} \\ e_{\mathrm{heading}} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 5 & 1 & 0 & 0 \\ 0 & 0 & 5 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \phi \\ \dot{\phi} \\ v \\ w \\ e_{\mathrm{lateral}} \\ e_{\mathrm{heading}} \end{bmatrix} + \begin{bmatrix} 0 \end{bmatrix} \begin{bmatrix} \tau_R \\ \tau_L \end{bmatrix}$$

Now that we have the state space matrices, we can check for controllability, which is given from the matrix W. A system is controllable when the rank of matrix W = the length of matrix A, which in this case, it does, and it = 6. Length(A) = rank(W) = 6.
Despite the system being controllable, it's asymptotically unstable. This means that the real eigenvalues of A are not all negative. Since that is the case, it is highly likely that the robonaut will either crash or go off. the road. Therefore, we must design a controller that will allow us to improve our controller.

$$eig(A) = \begin{bmatrix} 0 \\ 5.2386 \\ -5.2386 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

We can go on to check if our system is observable or not now. This is done using the MATLAB function obsv(A,C), and we compare the rank of this O matrix with length of the A matrix, which are both 6. Therefore we can say that our system is observable. Length(A) = rank(O) = 6.

4 Control Design

This section has been written out, but matrices are not confirmed as of now. They will change depending on how the controller works.

4.1 Controllability

To design the controller, the system needs to be controllable. We need to check for controllability as it allows system to move via the input from an intial state to the final state. We found controllability above. Therefore, we know go on to find the controller gain matrix, K, which can be found using the lqr function in MATLAB, using state cost weighted matrix and input cost weighted matrix.

$$K = lqr(A, B, Q_c, R_c) \tag{10}$$

The cost weighted matrices used were;

$$Q_c = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 450000 & 0 \\ 0 & 0 & 0 & 0 & 0 & 450000 \end{bmatrix}, R_c = \begin{bmatrix} 10 \end{bmatrix}$$

This outputted a K matrix of;

$$K = \begin{bmatrix} -12.6635 & -2.4556 & -0.2236 & 8.8959 & -150.0000 & 196.2465 \\ -12.6635 & -2.4556 & -0.2236 & -8.8959 & 150.0000 & -196.2465 \end{bmatrix} \tag{11}$$

4.2 Observability

In my case, as I don't have complete knowledge of the state, a new ODE is formed, where I require L as shown below:

$$\dot{\hat{x}} = A\hat{x} + Bu - L(C\hat{x} - y) \tag{12}$$

[2] Where L, the observer gain matrix is formed using:

$$L = lqr(A', C', inv(R_o), inv(Q_o))' \tag{13}$$

With the observer matrices being:

$$R_o = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 5 & 0 \\ 0 & 0 & 0 & 0 & 0 & 5 \end{bmatrix}, Q_o = \begin{bmatrix} 30 \end{bmatrix}$$

Outputting a L matrix of:

$$L = \begin{bmatrix} -13.9784 & -13.9784 & -0.0000 & 0.0000 \\ -70.4667 & -70.4667 & -0.0000 & 0.0000 \\ 4.9655 & 4.9655 & 0.0000 & -0.0000 \\ 3.8626 & -3.8626 & -0.1297 & 0.3786 \\ -0.1297 & 0.1297 & 4.8402 & -1.6943 \\ 0.3786 & -0.3786 & -1.6943 & 1.8973 \end{bmatrix} \tag{14}$$

4.3 Stability

We then go on to make 2 final checks regarding the asymptotic stability of the system. If they have all negative real parts, then the system is stable.

$$eig(A - BK) = \begin{bmatrix} -0.1372 + 0.0000i \\ -4.7723 + 0.0000i \\ -5.7577 + 0.0000i \\ -19.0776 + 19.5268i \\ -19.0776 - 19.5268i \\ -5.9933 + 0.0000i \end{bmatrix} \qquad eig(A - CL) = \begin{bmatrix} -38.5684 + 0.0000i \\ -7.2619 + 0.0000i \\ -3.8248 + 0.0000i \\ -7.7349 + 0.0000i \\ -3.3639 + 2.2870i \\ -3.3639 - 2.2870i \end{bmatrix}$$

The results that we obtain show that our system is now asymptotically stable; will tend to zero, allowing me to implement and test my controller.

5 Testing

Having successfully verified controllability and observability, we can now implement our controller into the system. I made 3 controllers, with varying gains. The first controller has high speed, as well as a high turning rate, and a high chassis angle. The second one is adjusted and has controller gain matrices that have the same values. The last controller has a slower speed and gain matrices where R_c is much smaller than Q_c. The three controllers and the reasoning behind the changes will be spoken about in this section. When creating the controllers, many errors came into play. This meant that the controller crashed, and there were only 2 reasons why the controller would crash, either it drove off the road, or it tipped over. The issue of crashing could be de-bugged by changing w (if robot was turning into a crash) or ϕ (if robot was tipping). $e_{lateral}$ and $e_{heading}$ were focused on in the gain matrix if the robot was veering off the road.
Regardless of these tweaks, I should mention that failure occured most often on roads that had consecutive sharp turns, and the controller was unable to recover and stabilize in time. The controller was able to stabilize in time if it was one sharp turn, but did not have enough time to stabilize with them being back to back.

5.1 Controller 1

This controller is the original controller, with high velocity with equilibrium velocity being 6 m/s and gains as shown below:

$$Q_c = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 450000 & 0 \\ 0 & 0 & 0 & 0 & 0 & 450000 \end{bmatrix}, R_c = [10]$$

and observer gain being:

$$R_o = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 5 & 0 \\ 0 & 0 & 0 & 0 & 0 & 5 \end{bmatrix}, Q_o = [30]$$

This controller was able to operate well, but had a success rate of only 60 percent, as it was unsuccessful in making consecutive sharp turns.

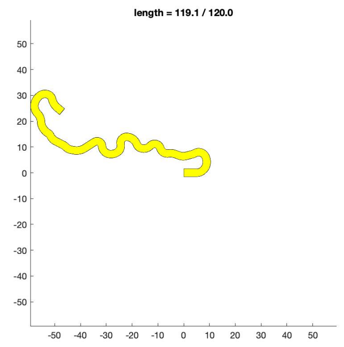

Figure 1: Failure Track

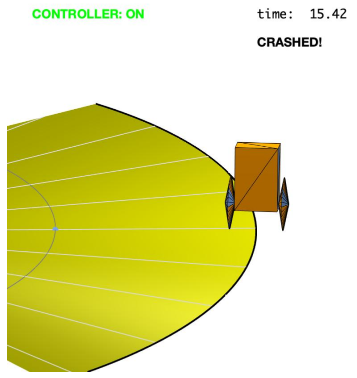

Figure 2: Robonaut Crash

As we can see in the figure above, the robonaut veers off the track where there are multiple turns. The robonaut went off track as too much leeway was given to it in the observer matrix, allowing it to move away from the centerline by a larger amount. This means that R_o needs to reduce to make sure that the turn isn't as loose. Another reason that this controller didn't work was because of the velocity. The robonaut was travelling too fast for it to stabilize in time during these turns and re-align itself with the center-line, causing it to veer off track, and crashing. Therefore, to eliminate this kind of failure, I reduced the speed of the robonaut, and further reduced the leeway allowed in my observer matrices. As the robot veered off the track, that means that I need to change my $e_{lateral}$ and $e_{heading}$ in my controller matrix to get my robonaut to stay closer to the center-line.

5.2 Controller 2

This controller was adjusted from the original controller, with a lower velocity being 5.8 m/s, and a change in the controller and observer matrices at $e_lateral$ and $e_{heading}$, to adjust for the veering off, and allowing it time to re-stabilize itself. The gains are shown below:

$$Q_c = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 470000 & 0 \\ 0 & 0 & 0 & 0 & 0 & 470000 \end{bmatrix}, R_c = [10]$$

and observer gain being:

$$R_o = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 4.8 & 0 \\ 0 & 0 & 0 & 0 & 0 & 4.8 \end{bmatrix}, Q_o = \begin{bmatrix} 30 \end{bmatrix}$$

Therefore testing this controller on a track with sharp turns, we see that the controller is able to make those turns, whilst being able to re-stabilize itself and keep itself inbounds.

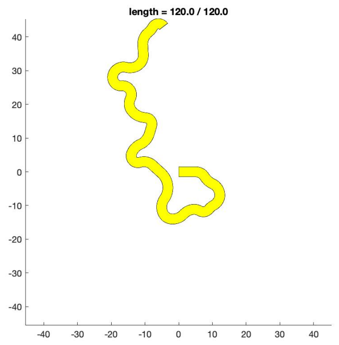

Figure 3: Failure Eliminated Track

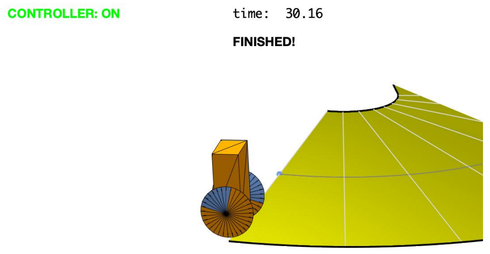

Figure 4: Robonaut Success

As we can see, the track had several consecutive sharp turns, but the robonaut successfully managed to complete it. This means that we managed to eliminate the issue of the robonaut veering off by changing $e_lateral$ and $e_{heading}$ as well as the velocity. However, the robonaut takes 30 seconds to complete the track, which completes the requirement set forward in terms of time, but I would like it to be quicker, so I changed the velocity back to 6 m/s to see if it possible for the robonaut to achieve success. The reason I want the robonaut to go faster is that with a faster robonaut, it would allow me to be a good racer in the competition, and so I increase the speed, and test the controller 5 times on 5 random roads to see if it has a 80% success rate.

Testing it on 5 random roads, we obtain the following: Therefore, we managed to achieve the 80 % success

Road	Time (sec)
Road 1	27.58
Road 2	28.28
Road 3	27.60
Road 4	28.02
Road 5	Crashed

Table 1: Road Completion Times

rate, but there was a failure in the last track, shown below:

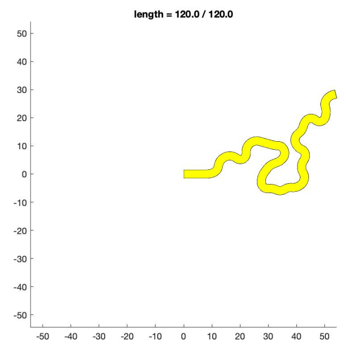

Figure 5: Failure Track

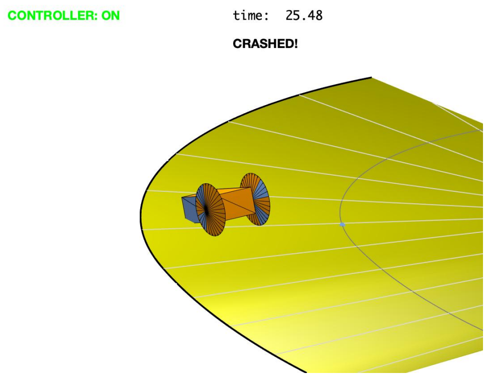

Figure 6: Robonaut Failure

We can see that the controller was working quite efficiently and was near the finish line when it crashed. This time, the crash wasn't it veering off the track, but it tipped over on the last turn. This means that I need to adjust my w and ϕ in my controller gains matrix to account for that crash. The reason this happened was because despite keeping my robonaut closer to the line, it tried to turn re-adjust too fast, causing it to loose balance and topple over. Therefore, I take that into account and increase my chassis angle and decrease my turning rate.

5.3 Controller 3

This controller was adjusted from controller 2, with a lower equilibrium velocity of 5.8 m/s, and a change in the controller and observer matrices at ϕ and w, to adjust for the tipping and crashing. This should allow my controller to re-stabilize itself after turns, and not crash. However, after nearly 100 different iterations, I changed a lot more than those 2 values, providing me with a fast, and efficient controller with the gains shown below:

$$Q_c = \begin{bmatrix} 65 & 0 & 0 & 0 & 0 & 0 \\ 0 & 8.5 & 0 & 0 & 0 & 0 \\ 0 & 0 & 8.5 & 0 & 0 & 0 \\ 0 & 0 & 0 & 9 & 0 & 0 \\ 0 & 0 & 0 & 0 & 490000 & 0 \\ 0 & 0 & 0 & 0 & 0 & 490000 \end{bmatrix}, R_c = \begin{bmatrix} 10 \end{bmatrix}$$

and observer gain being:

$$R_o = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 5 & 0 \\ 0 & 0 & 0 & 0 & 0 & 5 \end{bmatrix}, Q_o = \begin{bmatrix} 30 \end{bmatrix}$$

These gains were implemented into 5 consecutively randomly generated roads, and the results are shown below: When playing around with the values, there was a gains combination that allowed me to average 24s

Road	Time (sec)
Road 1	25.70
Road 2	25.46
Road 3	25.84
Road 4	25.24
Road 5	25.58

Table 2: Road Completion Times

a track, but it also failed the 5th run, thus the gains was changed to the final controller 3 gains shown above. The randomly generated tracks and controller 3's successes are shown below:

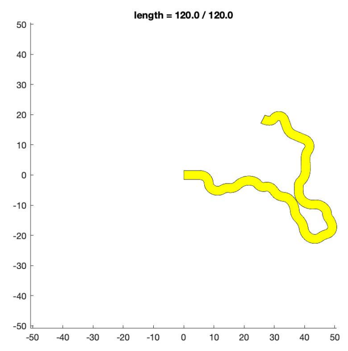

Figure 7: Track 1

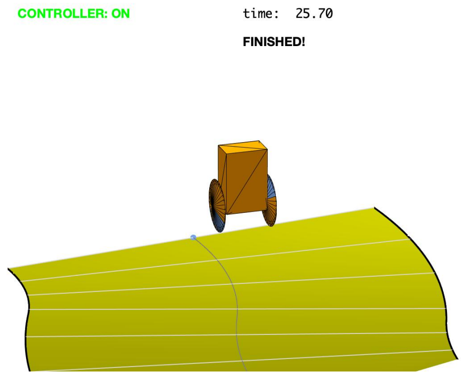

Figure 8: Success 1

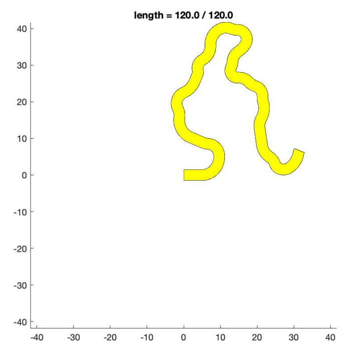

Figure 9: Track 2

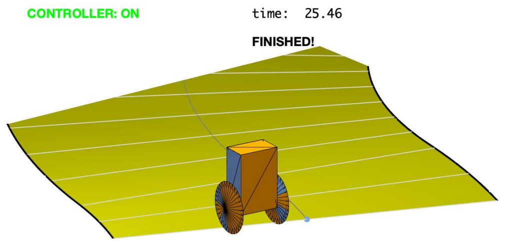

Figure 10: Success 2

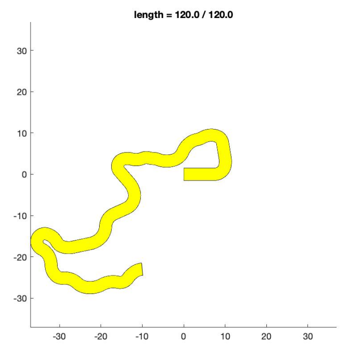

Figure 11: Track 3

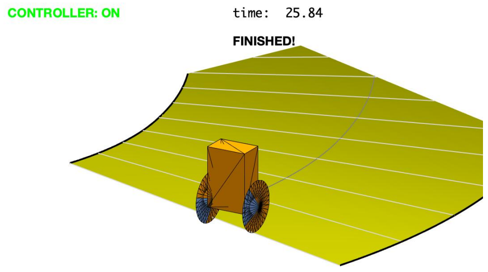

Figure 12: Success 3

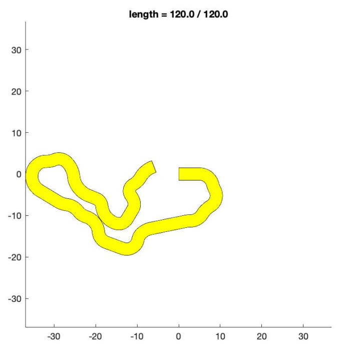

Figure 13: Track 4

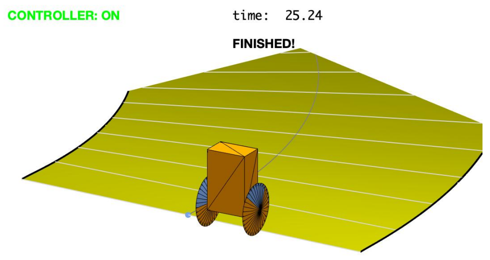

Figure 14: Success 4

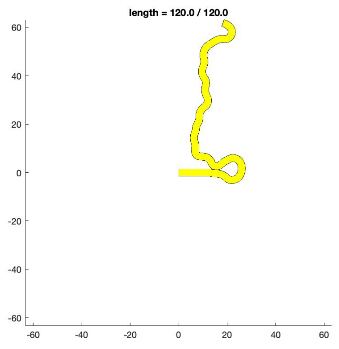

Figure 15: Track 5

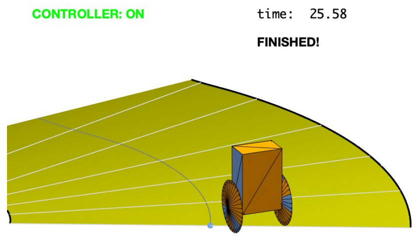

Figure 16: Success 5

Therefore, I managed to achieve a 100 % success rate, but there is obviously chance for failure in some tracks, which can not be accounted for, and that is shown below. We see that my controller crashed early on in this race, which was kind of a let down after 9 consecutive successful runs, but it happens. Not everything can be perfect, and that was shown to me in my 10th trial. Therefore, I can conclusively say that my requirements were fulfilled, where my controller successfully completed 5 consecutive random trials. We could have no failures with the controller present if we slowed it down, but then there would be no point in racing it, thus the controller speed is not reduced from 5.8 m/s. The controller was tested with an equilibrium velocity of 4.5 m/s and never failed, but it averaged 30s per track, which is too slow. The 10th track failure is shown below:

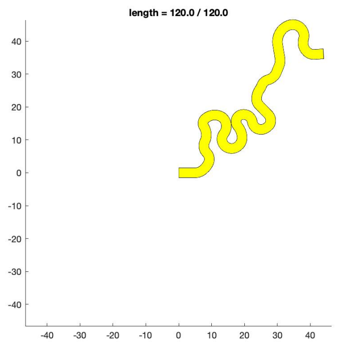

Figure 17: Track 10

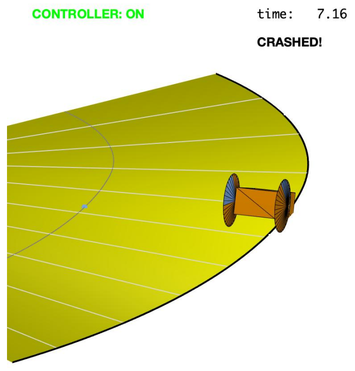

Figure 18: Failure

The robonaut crashed whilst taking sharp and then wide radius turns, showing us the limitations to the controller when increasing speed above a certain limit. Therefore, this is the final gains that I use.

6 Results

Each controller behaved differently, but we know that controller 1 was the least effective one. Controller 2 and 3 were similar in success rate, where both crashed in some instanced, but controller 3 was successful more frequently and averaged a lower time. This was to be expected as controller 2 was a dampened version, where it was penalizing its variables much less than controller 3; meaning that it took wider turns, was travelling at a lower speed, and was consistently near the edge of the road.
Therefore, from the testing shown above, we can conclude that controller 3 was the most efficient and effective controller produced, with an average of about 25.5 s per race, and a 80% success rate. What I realized with controller 3 was that it tipped if there were curves as soon as the road started because it was still stabilizing out, however, once it straightened out, regardless of the number of curves and sharp turns, the controller would successfully complete the race. If the speed of the controller was reduced, the success rate increased to nearly 100% success, however, it was slower than 25.5s, thus I did not reduce my v_{road} and left my controller at 80% success. I tested this out by running controller 3 through 100 randomly generated races, and the controller successfully completed 83 races out of those 100 races, successfully verifying my requirements. When I reduced equilibrium v_{road} to 4.5 m/s, the controller was able to complete 97 tracks successfully out of a 100, which shows how speed affects the success rate as well.

7 References

References

[1] J. Bibby and R. Necessary. (2020, 12) Robonaut 1. [Online]. Available: https://robonaut.jsc.nasa.gov/R1/sub/mobility.asp

[2] K. J. Aström and R. M. Murray, *Feedback Systems: An Introduction for Scientists and Engineers*, 1st ed. Princeton University Press, 2010. [Online]. Available: http://www.cds.caltech.edu/~murray/amwiki/index.php/First_Edition