Design Project 2 - Differential Drive Robot

Rohen A. Agarwal*
University of Illinois at Urbana-Champaign, Champaign-Urbana, Illinois, 61801

This report outlines my process of designing, implementing, and testing a controller I designed for a differential drive robot. The aim of the controller is to be able to move the robot around a circular path from start to finish without toppling over the edge as well as meeting certain requirements. In my process of designing a controller for this purpose, I found that there were certain limitations that were imposed on factors such as the forward speed, lateral error and others that caused me to try multiple iterations until I could find a suitable range of values for my controller to work properly. In this report I go through my process from start to finish including my explanation of the limits of my controller.

I. Nomenclature

$e_{lateral}$	=	lateral error (m), or the distance from the center of the robot to the center line of the track
$e_{heading}$	=	heading error (rad), or the difference between the orientation of the robot and the direction of the road
v	=	forward speed of the robot (m/s)
w	=	turn rate of the robot (rad/s)
θ	=	pitch angle (rad)
$\dot{\theta}$	=	pitch change rate (rad/s)
τ_L	=	left wheel torque (N.m)
τ_R	=	right wheel torque (N.m)
u	=	input in state space model
x	=	state in state space model
A	=	state matrix in state space model
B	=	input matrix in state space model
C	=	output matrix in state space model
D	=	feedthrough matrix in state space model
K	=	gain matrix

II. Introduction

The aim of my controller is to make the robot go around the circular track as close to the centre lane as possible. My robot is made up of a chassis and two wheels on the left and right. The left and the right wheels are controlled by torques applied to them. When a torque is applied to the left wheel, the left wheel starts rotating. Similarly when a torque is applied to the right wheel, the right wheel start rotating. Therefore, in the case where the left torque and the right torque are the same, the robot moves straight ahead. If the left torque is more than the right torque, the robot starts turning towards the right and vice-versa. I will leverage these properties in my controller and make it so that the left and right wheels get just enough torque so as to continue along the circular path as close to the centre line as possible. One limitation is that the maximum torque that can be applied in a either wheel is 5 (N.m) in either direction. The robot has sensors that give back data on factors such as the torque on each wheel, the forward speed, the lateral error, and the heading error while it is in motion.

*Junior, Aerospace Engineering

III. Requirements and Verification

A. Requirements

My robot must:

1) have a maximum lateral error of .5m in either direction
2) move with an average forward velocity of 2.3 m/s
3) satisfy requirements 1 and 2 for any random initial condition

I chose my maximum lateral error to be .5m because the distance between the two wheels is .7m, the half of which is .35m and so accounting for a little bit of error I found that .5m is a reasonable lateral error limit. As for the forward speed of 2.3 m/s, I found 2.3 m/s to be a suitable forward velocity for my robot so that it is stable and doesn't topple over the edge.

B. Verification

I will be running a simulation in Pybullet to implement and test my controller. With the help of the sensors on the robot, I will be getting data of the lateral error, forward speed, turning rate and other important factors, which I will plot against time. In my case, the plots that I am most concerned with are the ones for lateral error and forward speed. If the maximum lateral error is less than or equal to .5m in magnitude, I will have satisfied that requirement 1. Additionally, if after a short span of time the plot for forward speed averages out at 2.3 m/s, requirement 2 will also be met. The reason that mention "after a short span of time" is because the robot goes from rest to max forward speed in the beginning but I am concerned with the average over the time the entire simulation is run. For requirement 3, I will test this for different initial conditions to verify that my controller works for all cases and not just a particular "lucky" case.

IV. System

Figure 1 and 2 show the system we start out with. The AE353 course website provides us with this system initialised. The light blue circular path is the path on which the robot must travel while also meeting the pre-set requirements. The dark blue body in figure 2 is the chassis and the two orange wheels are the left and right wheels.

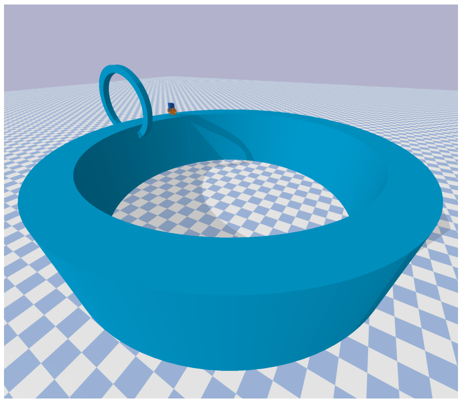

Fig. 1 System in pybullet

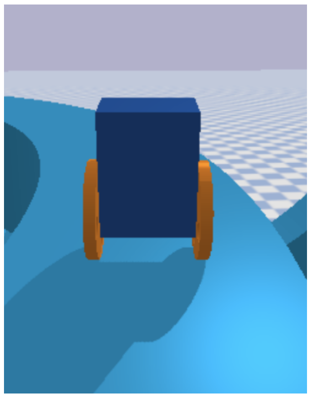

Fig. 2 Differential drive robot

V. System

A. Equations of Motion

The governing equations of motion for this robot is a system of ODEs (ordinary differential equations) and can be represented as

$$\begin{bmatrix} \dot{e}_{lateral} \\ \dot{e}_{heading} \\ \dot{v} \\ \dot{w} \\ \ddot{\theta} \end{bmatrix} = f(e_{lateral}, e_{heading}, v, w, \theta, \dot{\theta}, \tau_L, \tau_R) \tag{1}$$

where f, the second order ODE is defined as

$$f = \begin{bmatrix} vsin(e_h) \\ w \\ -\frac{2400\tau_L+2400\tau_R+2808\left(\dot{\theta}^2+w^2\right)sin(\theta)+13\left(250\tau_L+250\tau_R-195w^2\sin(2\theta)-8829\sin(\theta)\right)\cos(\theta)}{11700\cos^2(\theta)-12168} \\ \frac{32\left(-875\tau_L+875\tau_R-1443\dot{\theta}w\sin(2\theta)-2925vw\sin(\theta)\right)}{13\left(3120\sin^2(\theta)+2051\right)} \\ \frac{42250\tau_L+42250\tau_R-32955w^2\sin(2\theta)+300\left(100\tau_L+100\tau_R+117\left(\dot{\theta}^2+w^2\right)\sin(\theta)\right)\cos(\theta)-1492101\sin(\theta)}{1404\left(25\cos^2(\theta)-26\right)} \end{bmatrix} \tag{2}$$

The first step would be to linear this and bring it into a state space model such as

$$\dot{x} = Ax + Bu \tag{3}$$

$$y = Cx + Du \tag{4}$$

The feedthrough matrix D = [0] in this case, and we obtain output matrix y through the data we receive from the sensors on the robot.

B. Equilibrium values

Chosen equilibrium values should be such that

$$f(e_{lateral}, e_{heading}, v, w, \theta, \dot{\theta}, \tau_L, \tau_R) = [0] \tag{5}$$

One fitting choice from looking at the EOMs is

1) $e_{lateral_e} = 0$
2) $e_{heading_e} = 0$
3) $w_e = 0$
4) $\theta_e = 0$
5) $\dot{\theta} = 0$
6) $v_e = 2.3$
7) $\tau_{L_e} = 0$
8) $\tau_{R_e} = 0$

I chose the input variables of interest to be τ_L and τ_R while the state variables of interest were $e_{lateral}$, $e_{heading}$, v, w, θ, $\dot{\theta}$. Therefore, the state and input matrices with the equilibrium values becomes

$$x = \begin{bmatrix} e_{lateral} - e_{lateral_e} \\ e_{heading} - e_{heading_e} \\ v - v_e \\ w - w_e \\ \theta - \theta_e \\ \dot{\theta} - \dot{\theta}_e \end{bmatrix} \quad u = \begin{bmatrix} \tau_L - \tau_{L_e} \\ \tau_R - \tau_{R_e} \end{bmatrix} \tag{6}$$

and at equilibrium

$$x_{equilibrium} = \begin{bmatrix} 0 \\ 0 \\ 2.3 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad u_{equilibrium} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \tag{7}$$

While there could be other possible equilibrium values, I chose these because they made my system much simpler. For example, the 0 lateral error helped me so the segbot would stay as close to the center line as possible. Similarly, a 0 initial left and right wheel torque made it easier for me to determine the effectiveness of my torque exerted just by my controller. Coming to forward speed, I chose a value of 2.3 $\frac{m}{s}$ because after testing a range of values, I found that my controller functioned well only when the forward speed was between 1.5 to 2.6 $\frac{m}{s}$.

C. Finding matrices A and B

This system must now be linearised to find definite matrices A and B. I found the Jacobians

$$A = \left.\frac{\partial f}{\partial x}\right|_{x_{equi}, u_{equi}} \quad B = \left.\frac{\partial f}{\partial u}\right|_{x_{equi}, u_{equi}} \tag{8}$$

using the the Jacobian function in python which I used to evaluate my system at my chosen equilibrium point, giving me the matrices

$$A = \begin{bmatrix} 0. & 2.3 & 0. & 0. & 0. & 0. \\ 0. & 0. & 0. & 1. & 0. & 0. \\ 0. & 0. & 0. & -0. & -245.25 & 0. \\ 0. & 0. & -0. & -0. & -0. & -0. \\ 0. & 0. & 0. & 0. & 0. & 1 \\ 0. & 0. & 0. & -0. & 1062.75 & -0. \end{bmatrix}$$

$$B = \begin{bmatrix} 0. & 0. \\ 0. & 0. \\ 12.07264957 & 12.07264957 \\ -1.05014439 & 1.05014439 \\ 0. & 0. \\ -51.46011396 & -51.46011396 \end{bmatrix}$$

which can be plugged into equation (3) to get the definite state space model.

D. Controllability

Now that we have defined out state space model, It is critical to determine if the system is even controllable or not. To do this, we need to find the controllability matrix W that is defined as

$$W = \begin{bmatrix} B & AB & A^2B & A^3B & \dots & A^{n-1}B \end{bmatrix} \tag{9}$$

A system is controllable is the rank of the controllability matrix W is equal to the number of states in the system. In other words, the rank of matrix W must equal the number of rows in matrix A. From above, we can see that A is a (6x6) matrix and therefore has 6 states. Computing the rank of the controllability matrix B using python's np.linalg.matrix rank function, I found the rank of W to equal 6. The system is controllable!

E. Stability

We ascertained that the system we have is controllable but we still need to test stability. A stable system is one that, when given a definite, bounded input, returns a bounded output. Such a system is called "asymptotically stable". Mathematically, if all the real parts of the eigenvalues of a matrix are negative, the system is asymptotically stable. While the imaginary parts may be whatever they may be, in some cases it is preferable to be 0 while in some a non zero number is just what we need. In my case, I did not concern myself much with the imaginary part but was sure to to look out for negative real parts. In python I used a for loop that computed and checked the eigenvalues of matrix A and since at least on of them have non negative real parts, the current system is unstable.

F. Using lQR to find stable gain matrix

We can however work around this by forming a closed-loop system $F = A - BK$ and using LQR (Linear Quadratic Regulator). The reason for choosing LQR as the method to find suitable gain matrix K is because not only is it easy to implement, but is also efficient since the the entire idea of LQR is to reduce the "cost" and yet make the system attain equilibrium the fastest. The LQR cost function is

$$cost = \int_0 x(t)^T Qx(t) + u(t)^T Ru(t)\, dt \tag{10}$$

where Q and R are the weights associated with x(t) and u(t) respectively. Since x(t) is a square matrix of dimension 6, Q must also be a square matrix of dimension 6. Since u(t) is (2x1) and $u(t)^T$ is (1x2), R must be (2x2). Each entry in x(t) and u(t) are weighted by their corresponding entry in Q and R.

Using linalg's "solvecontinuousare" function and a for loop I found a list of gain matrices K that could make my system stable. My final choice of K, after trying multiple gain matrices was

$$K = \begin{bmatrix} -1 & -2.9849339 & -1.15470054 & -1.82822411 & -21.45902858 & -1.68170601 \\ 1 & 2.98493398 & -1.15470054 & 1.82822411 & -21.45902858 & -1.68170601 \end{bmatrix} \tag{11}$$

I know this is stable because the eigenvalue matrix is

$$eigvals = \begin{bmatrix} -1.519 + 0j & -0.813 + 0.845j & -0.813 - 0.845j & -51.446 + 0j & -20.789 + 0j & -0.277 + 0j \end{bmatrix} \tag{12}$$

Now I have a stable control in theory, but I must implement it and analyse the results to see if my controller met the requirements or not.

VI. Results

A. Implementation

While implementing my controller, I varied the equilibrium forward speed values and found that my segbot would topple off if I went higher that 2.6 m/s and so I chose my equilibrium forward speed to be 2.3 m/s. My initial conditions for all the parameters were set to 0. I tested my controller three times so as to make sure I get consistent results. As seen

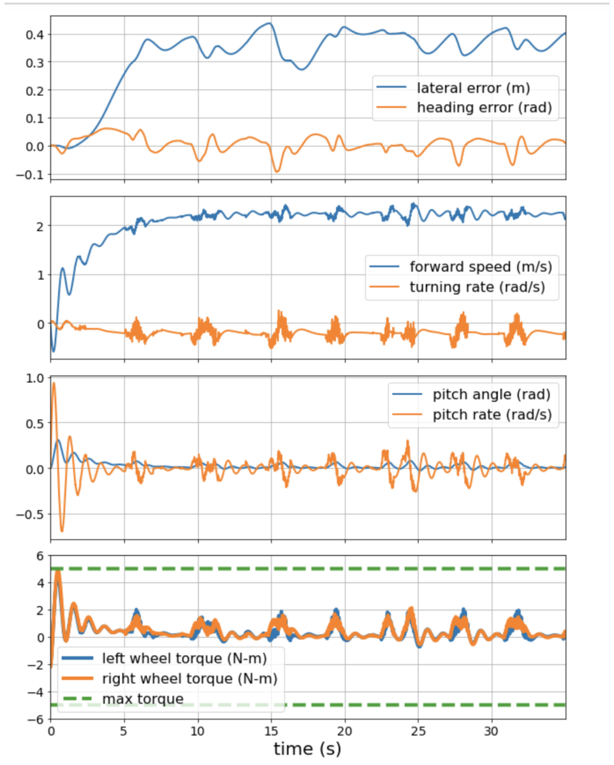

Fig. 3 Initial cond 0, forward speed = 2.3 m/s

in the first plot, the lateral error fluctuates, but never goes above 0.5m as required by requirement 1. Additionally, after a short time after starting, the forward velocity curve seems to average out at 2.3 m/s which was the desired equilibrium forward velocity, thus satisfying requirement 2. Now I must also test the same with random initial conditions. I chose initial forward speed to be 2 m/s and the resulting plot was

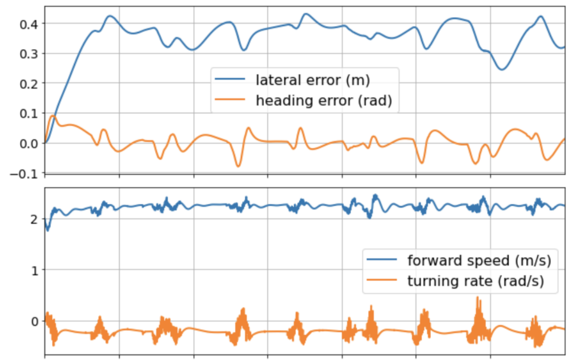

Fig. 4 Initial forward speed = 2 m/s

As seen, requirements 1 and 2 are still satisfied, but this time, since my initial forward speed was 2 m/s, my segbot attains it's equilibrium 2.3 m/s much faster. This intrigued me and made me think what would happen if my initial forward speed was higher that my equilibrium forward speed. I tested my controller with initial forward speed 3 m/s and get

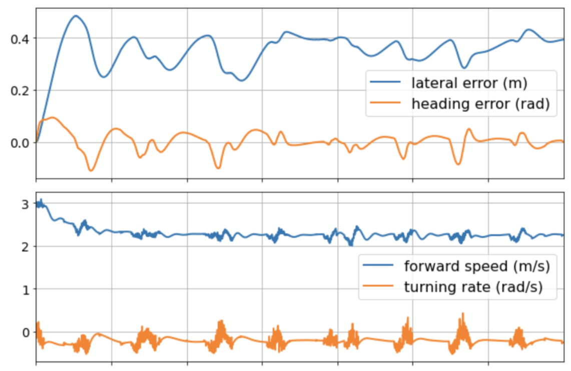

Fig. 5 Initial forward speed = 3 m/s

This proves that my controller is stable and regardless of what the initial forward speed is, the controller will make it so that the equilibrium forward velocity remain 2.3 m/s. I went ahead and explored other initial conditions. One of them was initial forward speed=3 m/s, ground pitch=0.5 rad, initial lateral error=0.01 m and the resulting plot was

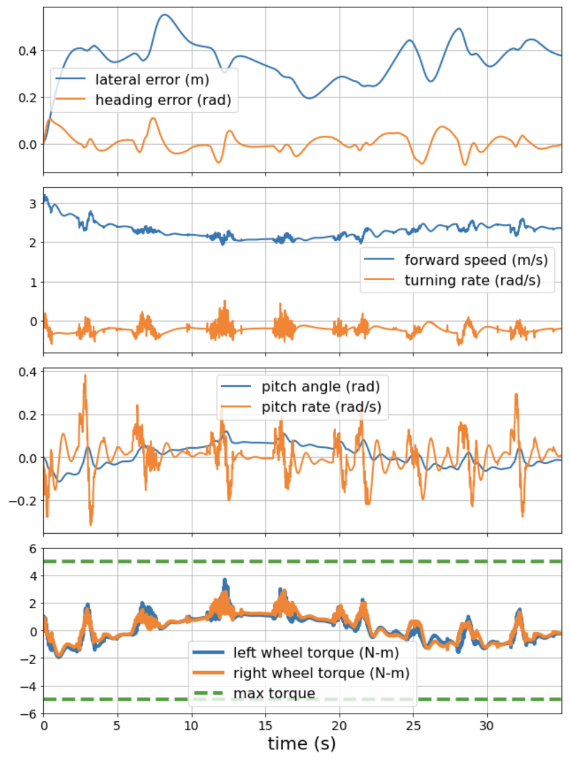

Fig. 6 Random initial conditions

This is quite an interesting result. The lateral error does overshoot the 0.5 m limit but that is because I introduce initial lateral error. The forward speed begins at the initial condition of 3 m/s, which is higher than the equilibrium forward speed, dips lower than 2.3 m/s and then then quickly attains equilibrium forward speed. This "dip" is caused by the up slope part of the track. The effect of the up slope and down slope can also be seen in the plot for the wheel torque where a higher torque was required for each wheel during the time the forward speed was dipping.

VII. Conclusion

I began my process by linearising my system to get a state space model, and found matrices A and B, after which I checked for controllability and stability. On finding that my state space model is controllable but not stable, I built a closed-loop form system that I could use to find suitable gain matrices K using a very efficient LQR method. Once I obtained a "good" K matrix, I implemented my control for all initial conditions as 0 and found that my controller satisfies all requirements. I also implemented the same controller with various random initial conditions (while still being in a realistic range) and yet my controller proved to be successful. I was able to verify my results by using the data received from the sensors on the robot which I converted into easily analyzable plots. I repeated the implementation for each case 3 times to make sure I had consistent results. In conclusion, I can say that my controller was successful because it satisfied requirements 1, 2, and 3.

Acknowledgments

I would like to thank professor Timothy Bretl for the very helpful lectures, example in-class codes as well as the homework that helped me in coding my controller. I also want to thank professor Timothy Bretl, for the AE353 course website off which I used the EOMs pre derived for me, as well as the images and description of the system. Additionally, his segbotDemo file as well as the DeriveEOM file was extremely helpful in my own implementation of my controller in I also want to credit Parthiv Kukadia because I used his for loop to find stable gain matrices K because I found that it is really efficient. I also want to thank all my peers for the continuous fruitful discussions we had on the topics as well as the project on Campuswire or Discord which helped me understand the content and do the project better.

References

Bretl, Timothy. https://tbretl.github.io/ae353-sp21/projectsdesign-project-2-differential-drive-robot