Feedback Control Design in Simulink for Trajectory Tracking of Autonomous Vehicles

Bryce C. Mack, *Student, Bradley University*

***Abstract*—This paper discusses the results produced when creating a feedback system from the control process discussed in [1]. Simulink was used to produce models for a differential drive wheeled mobile robot.**

In [1], there is a discussion about how a feedback control system can be used for autonomous vehicle control. A dynamic feedback linearization model is used to be accurate for trajectory tracking and set point regulation simultaneously.

I. Introduction

AUTONOMOUS vehicle control is on the verge of becoming one of the 21st centuries greatest technologies. Once cars can safely and reliably get from point A to point B, the world will be changed forever. However, working behind the scenes on these self-driving cars are dozens of engineers working on how to allow computers to govern the controls and guide the cars to their destinations. This project discusses how a control system would be able to follow a given path with minimal errors and travel safely. (For specifics on trajectory tracking for other autonomous vehicles see Appendix A.)

II. Problem

For this problem to be solved, several differential equations will need to be solved to be able to formulate the equations that govern the motion of the vehicle.

A. WMR Model

The vehicle discussed in [1] was a differential drive wheeled mobile robot (WMR). Such a vehicle is driven by two wheels, each with a linear and angular velocity. [1] discusses how the kinematic equations that govern a unicycle can be used to model the WMR.

$$\dot{x} = v\cos\theta \tag{1}$$
$$\dot{y} = v\sin\theta \tag{2}$$
$$\dot{\theta} = \omega \tag{3}$$

These differential equations were solved in Simulink using integrator blocks for each state variable.

B. Linear Trajectory

I needed a subsystem in Simulink that would be able to produce the PD control equation values. This Simulink subsystem takes the desired position measurements as well as the current velocity and positions and uses them to calculate the values u_1 and u_2.

$$u_1 = \ddot{x}_d + k_{p1}(x_d - x) + k_{d1}(\dot{x}_d - \dot{x}) \tag{4}$$

$$u_2 = \ddot{y}_d + k_{p2}(y_d - y) + k_{d2}(\dot{y}_d - \dot{y}) \tag{5}$$

The question was how to get values for $\ddot{x}_d, \ddot{y}_d, \dot{x}_d, \dot{y}_d, \dot{x}, \dot{y}, x$ and y. This will be discussed in III.

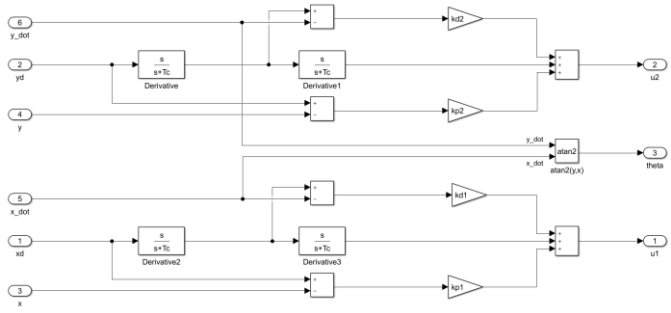

Figure 1: Linear Trajectory Model

C. WMR Control

I created another Simulink subsystem to determine the control variable for the WMR. This subsystem uses the linear trajectory subsystem to generate variables for the dynamic compensator.

From [1], values u_1 and u_2 can be used to create the equations of the dynamic compensator.

$$\dot{\xi} = u_1\cos\theta + u_2\sin\theta \tag{6}$$
$$v = \xi \tag{7}$$
$$\omega = \frac{u_2\cos\theta - u_1\sin\theta}{\xi} \tag{8}$$

Within this subsystem, I implemented (6) and (7). The only variable that needs to be integrated is ξ, so that will be found using another integrator block. Then once ξ is found, ω can be solved for from (8).

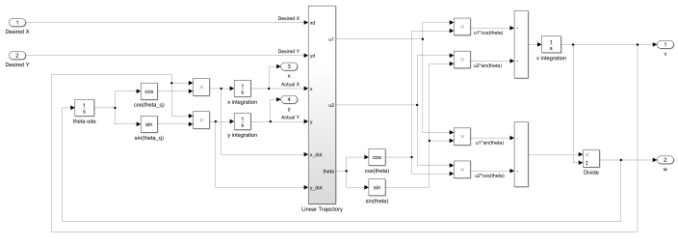

Figure 2: WMR Control Model

D. Trajectory Tracking Model

Figure 1 shows how the linear trajectory model that I created in Simulink. It takes inputs from the WMR control model in figure 2 and converts them into new values that the WMR control model needs to complete calculations for linear and angular speed.

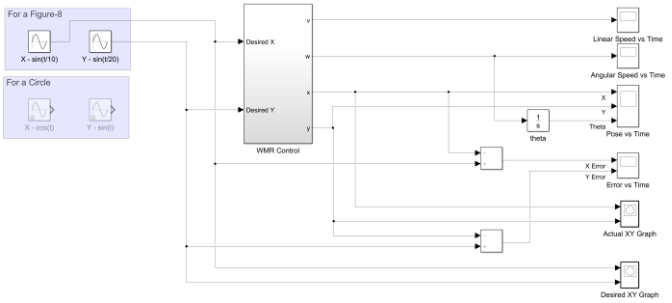

Figure 3: Trajectory Tracking Model

Figure 3 shows the complete model for the trajectory tracking system. This is where the desired inputs are input into the WMR control model shown in figure 2. The outputs of the WMR control model are then fed into Simulink scopes to see the position and speed of the robot over the duration of each simulation.

III. Solution

The solution of this problem required a desired path from x_d and y_d for the robot model to track to. I chose to use the same equations as were chosen in [1] to see if I could reproduce the results in [1].

A. Procedure

Using Simulink to solve this problem required creating the subsystems mentioned in II. I had to be sure that each block was outputting the correct signal for the next block to use.

B. Compilation

For the simulation to operate correctly, the correct values needed to be found. Initially, I used the differentiator block in Simulink to find $\ddot{x}_d, \ddot{y}_d, \dot{x}_d$ and $\dot{y}_d$. I assumed that it would work great for this project since the input path was twice differentiable and continuous for all time. However, the first simulation resulted in huge errors with proper tracking.

My next option was to use a simple transfer function block that would be able to act as a differentiator for my system. The differentiator block failed because the pole was much smaller than the time constant for my system's inputs. The transfer function would be the same as the differentiator block, but the pole would be further away from the origin, so that the system inputs wouldn't be cut out.

Variables $\dot{x}, \dot{y}, x$ and y were found before and after integrating velocity and integrating angular velocity to find the steering angle.

C. Control Parameters

In equations (4) and (5), there are the four control values k_{d1}, k_{d2}, k_{p1} and k_{p2}. In [1], the simulation results were produced using values $k_{d1} = k_{d2} = 0.7$ and $k_{p1} = k_{p2} = 1$.

There were also initial conditions for where the WMR would start. Instead of starting at the origin, it would be starting at [0.2, -0.3] with a steering angle of $\frac{\pi}{3}$.

For the differentiator I created in the linear trajectory model, I made the pole at 1. This was because the time constant of the system was 0.1 from the sine wave input function. If the time constant of the system were to change, the differentiator pole should be moved to a decade above the time constant of the system.

These values were given in [1] for the simulations. I chose to use the same values to reproduce the results in IV.

IV. Simulations

As stated previously, I used the values given in [1] to produce the following simulations. The chosen desired trajectory was a continuous figure-8 path described by $x = \sin(\frac{t}{10})$ and y= $\sin(\frac{t}{20})$.

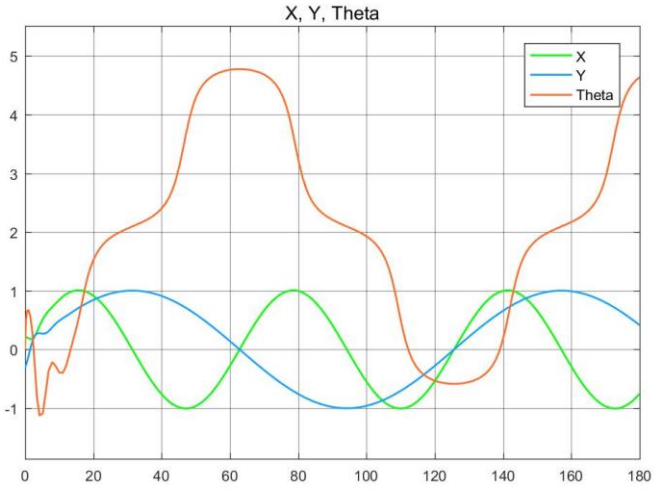

Figure 4: X, Y, θ vs. time

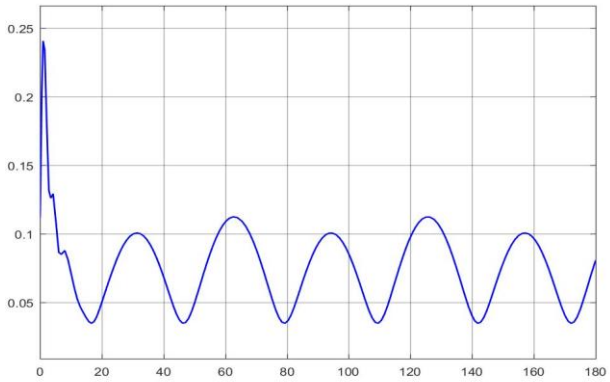

Figure 5: Linear Speed vs. time

Figures 4-6 match figures 6-8 in [1]. The only difference is that Simulink took more time to stabilize than the system that

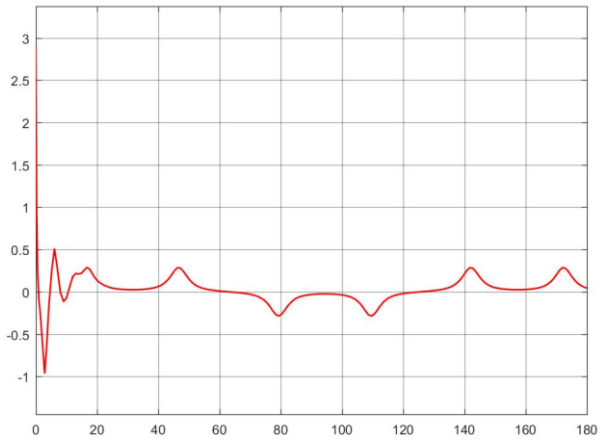

Figure 6: Angular Speed vs. time

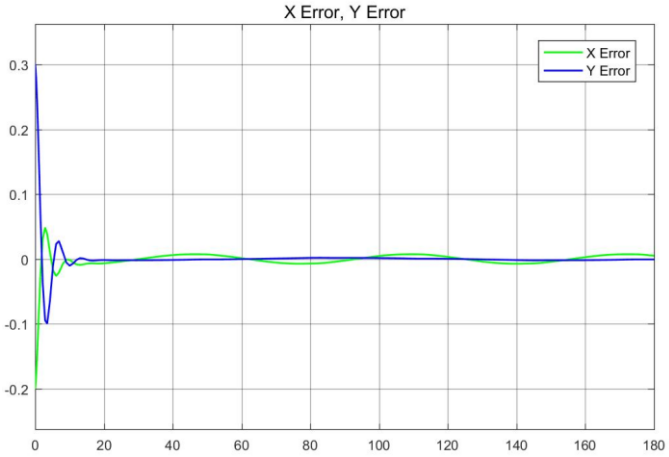

Figure 7: X, Y Position Error vs. time

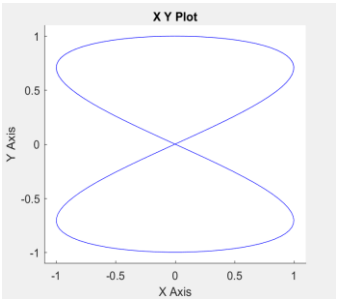

Figure 8: Desired Trajectory Figure-8

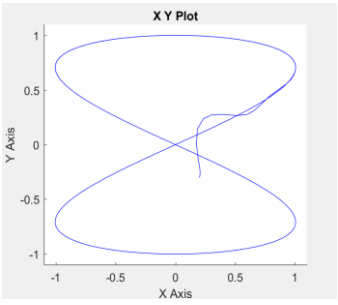

Figure 9: Actual Trajectory Figure-8

the authors of [1] used. In each figure, the first 10 seconds of the system running was used to get the WMR to the desired location with the correct orientation. From then on, the system tracks the desired input with extremely low error, as seen in figure 7.

Comparing figures 8 and 9 reveals how the WMR corrects itself to follow the desired path after starting off the path.

In figure 7, the positional deviations from the desired path are shown. The error follows the shape of a sine wave. This error occurs whenever the robot makes one of the turns in the trajectory. The robot slows down to make the turn and this creates a slight error. However, the positional deviations are extremely small and wouldn't be noticed in a large scale autonomous tracking system.

A. *Alternate Simulation*

After confirming the system's ability to track a figure-8 pattern, I wanted to test it with the desired path being a simple unit circle.

This simulation had greater error than the tracking shown in figure 9. Comparing figures 10 and 11 shows how the actual trajectory stabilized to a circle bigger than the desired circle. This can be corrected by introducing an integral controller to the system. Currently, the system uses only a PD controller, so the steady state error is not minimized. By upgrading to a PID controller, the tracking for this system would be nearly perfect.

V. Conclusion

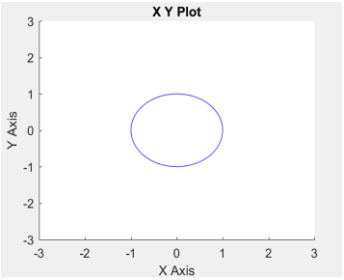

Figure 10: Desired Trajectory Circle

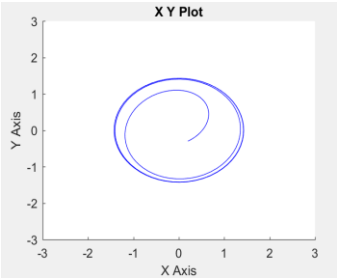

Figure 11: Actual Trajectory Circle

This report focused on how to implement dynamic feedback control with a Simulink model.

I picked out all the equations from [1] that would be needed for the model. Once I had the equations, I created a multi-layer model that would be able to take simple inputs and create the system needed for trajectory tracking.

After I had the model created, I tested the system for multiple inputs to see if it behaved correctly. My simulation results matched the simulations produced in [1] for a figure-8 input trajectory. The system was also able to track a circular input, albeit some error.

Overall, creating a Simulink for dynamic feedback control was a challenging, fun project.

A. *MATLAB Comparison*

Taking equations from a research paper and converting them into MATLAB code is challenging. Without comments and organization, MATLAB code quickly becomes confusing. In [1], there were many equations introduced, some needed for linear feedback control and others for dynamic feedback control. When I was working on this project using only MATLAB code, it was difficult to determine if my code was wrong or if I was using the wrong equation.

Although the same problem can happen with Simulink, there are no code typos that can create huge setbacks. With Simulink, it became a lot easier to visualize the entire system.

References

[1] G. Oriolo, A. De Luca, M. Vendittelli, "WMR Control Via Dynamic Feedback Linearization: Design, Implementation, and Experimental Validation," *IEEE Transactions on Control Systems Technology*, vol. 10, no. 6, pp. 835–852, Nov. 2002.