Predictive Velocity Modelling and Control in Autonomous Vehicles

[redacted]

Contact:

[redacted]

I. ABSTRACT

In this paper, we discuss utilizing predictive models of acceleration and deceleration to influence the velocity of an autonomous vehicle. This is accomplished utilizing a Simulink controller, which parses and processes data such that it may be given to the autonomous vehicle to be acted on. We found that predictive modeling is relatively successful; however, it encounters difficulties when faced with externalities such as dynamics, drag, and the power of the vehicle. It particularly encountered problems when accelerating from higher starting velocities. Potential solutions for these problems are addressed in section VI of the paper.

II. DESIGN APPROACH

The design philosophy for the controller was to create a smooth model of acceleration or deceleration between arbitrary speeds over an arbitrary time. As such, we decided to take input in the form of current velocity (V_0), a novel input of set velocity (V_s), and use a constant for time (t). From these inputs, we wanted to generate a means of smoothly transitioning the velocity from V_0 to V_s, and as such, implemented a mesh population algorithm to generate discrete data resembling a Sigmoid-style curve over t modeling velocity. To make this continuous, we decided to employ Natural Cubic Splines in order to obtain C^2 differentiability, allowing us to obtain a numerical derivative for acceleration values to give to the vehicle as well as jerk, if a scenario arose in which jerk became an issue. From this point, we utilized a closed-loop system to iterate through the resulting acceleration values and input them to the car such that the vehicle's velocity would approximate the predictive model generated from the algorithm.

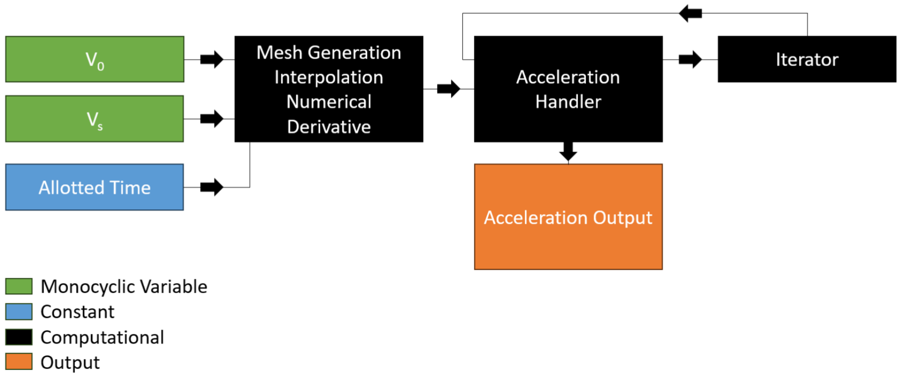


Figure 1. A diagram of the controller, with directional arrows to indicate the flow of data.

III. ALGORITHM AND VELOCITY MODEL

Given the nature of the problem, we sought to create a model that represented a smooth acceleration or deceleration within a fixed time frame. We defined smoothness as a lack of "jumping" behavior; that is, changes in velocity should be gradual so as not to induce excessive jerk on the vehicle, as well as hopefully compensate for some degree of momentum.

A. Initial Approach

Initially, we sought an analytic solution to model velocity, such that it would be natively adaptable. Our original thoughts were to utilize Sigmoid curves, as they are demonstrably smooth and have a peak derivative at their midpoint, denoting that peak acceleration would be obtained halfway through. Thinking on our original conditions from the design philosophy, we decided it would likely be optimal to adapt the population growth logistic function, as it allows for initial and final conditions, with the equation adapting to meet the desired final condition over a specified amount of time.

$$P(t) = \frac{K}{1 + \frac{K - P_0}{P_0} e^{-rt}}$$

Figure 2: The logistic population growth equation, where K is the maximum capacity, t is time, r is the growth rate, and P_0 is the initial population.

To adapt this equation, we substituted K for V_S, V_S for P_0 , and utilized t for t. This left us with the following equation:

$$V_1(t) = \frac{V_s}{1 + \frac{V_s - V_0}{V_0} * e^{-t\alpha_1}}$$

Figure 3: The population growth equation modified to suit our context.

Thus requiring us to symbolically solve for α_1, the growth rate for our specific equation given our various inputs. This resulted in:

$$\alpha_1 = -\frac{1}{t} ln((\frac{V_s}{V_s - \varepsilon} - 1)(\frac{V_0}{V_s - V_0}))$$

Figure 4: The solution to α_1 after symbolically solving the above equation.

The epsilon here, in particular, refers to the single precision machine epsilon, and was created to ensure that the leftmost component of the product in the natural log was never equal to zero. However, we neglected the case that V_0 may be equal to zero, which ensures that we are taking a natural log of 0. To remedy this, it is possible to add an epsilon to V_0 in the numerator of the rightmost component of the product, however, this will also likely result in some degree of instability in the case of $V_0 = 0$. This is because we would be dividing ϵ by V_S, which would cause an underflow in single precision. As such, a scaling epsilon may be necessary, or perhaps expanding to double precision while retaining a single precision epsilon would suffice. A similar issue can be seen in the deceleration model, shown below.

$$V_2(t) = \frac{V_s}{1 + \frac{V_s - V_0}{V_0} * e^{-t\alpha_2}}$$

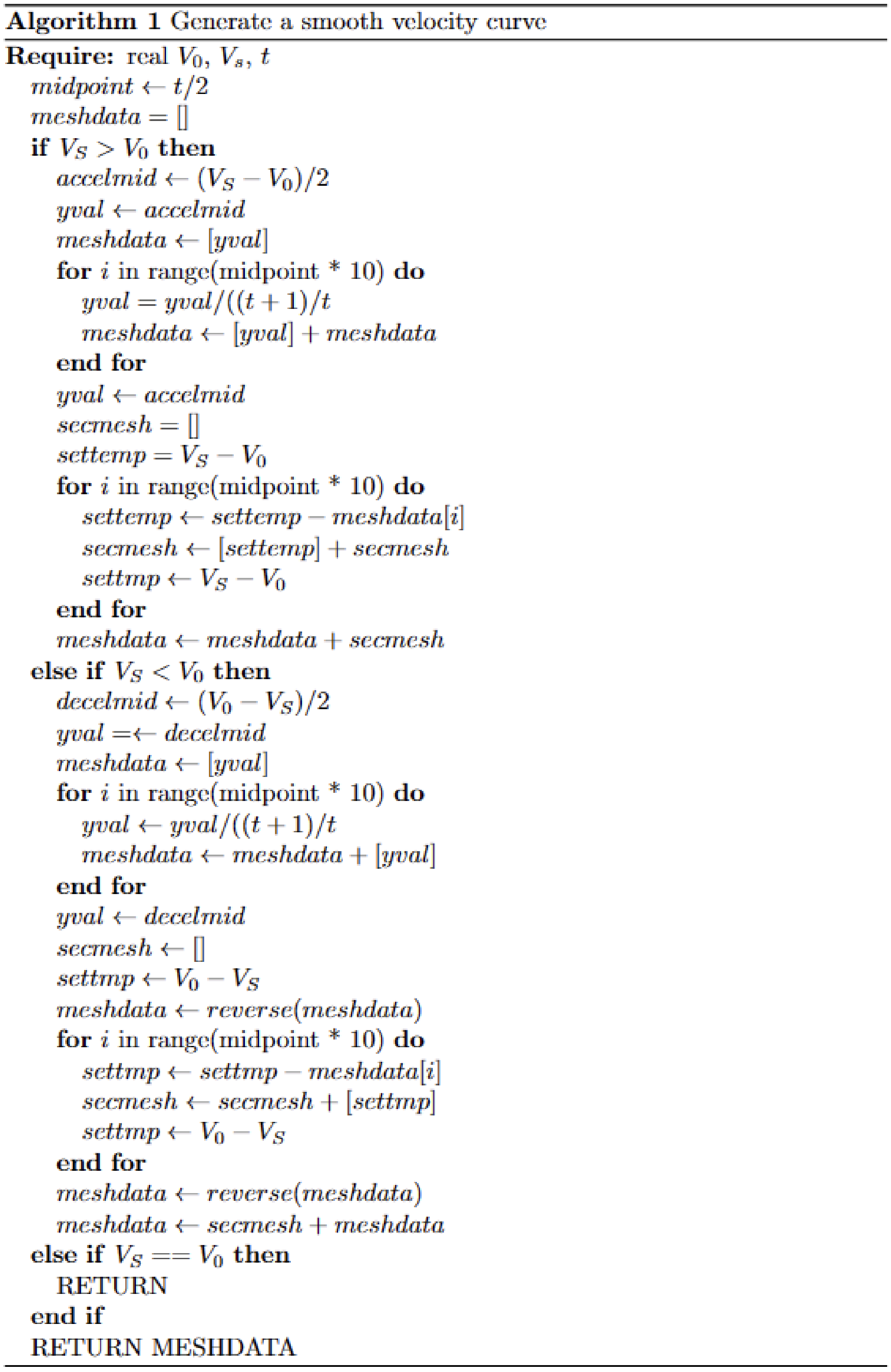
$$\alpha_2 = -\frac{1}{t} ln((\frac{V_s}{V_s + \varepsilon} - 1)(\frac{V_0}{V_s - V_0}))$$

Figure 5: The deceleration model and corresponding rate equation.

This model also had an additional problem of requiring a translation, as changes would occur in negative time and at negative velocity in the context of the problem. Given the problematic nature of the model, we elected to switch to a generative model instead. However, given more time, the original model may still prove suitable.

B. Generative Model

Given the problems of the analytic model, we elected to instead utilize a mesh of algorithmically generated data points to approximate a Sigmoid curve, such that it would be plotted over t with a range of $abs(V_S - V_0)$. We elected to utilize a step size $h = 0.1$, as we believed a finer mesh would not be of particular use given momentum and other externalities. From this, we decided an approach of calculating decay and inverting would likely be the most viable means of efficiently calculating the model, as it not only makes sense to ensure symmetry, but also because it ensures that the loops utilize only simple operations that should be fast to compute. The algorithm also functions to help serve as cruise control about V_S, as it will sample the current speed every period t and generate a curve to cause the vehicle to accelerate or decelerate towards that V_S. As as result, the algorithm not only finds use in calculating velocity changes between a V_S and V_0 of a large magnitude, but also that of a small magnitude, already demonstrating that it serves a wide basis of cases. It is also scalable to time, allowing for varying times to be passed. The pseudocode for the algorithm is on the following page.



Algorithm 1 Generate a smooth velocity curve

```
Require: real $V_0$, $V_s$, $t$
  $midpoint \leftarrow t/2$
  $meshdata = []$
  if $V_S > V_0$ then
    $accelmid \leftarrow (V_S - V_0)/2$
    $yval \leftarrow accelmid$
    $meshdata \leftarrow [yval]$
    for $i$ in range(midpoint * 10) do
      $yval = yval/((t+1)/t$
      $meshdata \leftarrow [yval] + meshdata$
    end for
    $yval \leftarrow accelmid$
    $secmesh = []$
    $settemp = V_S - V_0$
    for $i$ in range(midpoint * 10) do
      $settemp \leftarrow settemp - meshdata[i]$
      $secmesh \leftarrow [settemp] + secmesh$
      $settmp \leftarrow V_S - V_0$
    end for
    $meshdata \leftarrow meshdata + secmesh$
  else if $V_S < V_0$ then
    $decelmid \leftarrow (V_0 - V_S)/2$
    $yval =\leftarrow decelmid$
    $meshdata \leftarrow [yval]$
    for $i$ in range(midpoint * 10) do
      $yval \leftarrow yval/((t+1)/t$
      $meshdata \leftarrow meshdata + [yval]$
    end for
    $yval \leftarrow decelmid$
    $secmesh \leftarrow []$
    $settmp \leftarrow V_0 - V_S$
    $meshdata \leftarrow reverse(meshdata)$
    for $i$ in range(midpoint * 10) do
      $settmp \leftarrow settmp - meshdata[i]$
      $secmesh \leftarrow secmesh + [settmp]$
      $settmp \leftarrow V_0 - V_S$
    end for
    $meshdata \leftarrow reverse(meshdata)$
    $meshdata \leftarrow secmesh + meshdata$
  else if $V_S == V_0$ then
    RETURN
  end if
  RETURN MESHDATA
```

Figure 6: The pseudocode for the mesh generation algorithm.

The resulting data, contained in an array, is then paired with corresponding x values such that when plotted, each y is paired with the corresponding step of its calculation. This yields curve approximations such as the following.

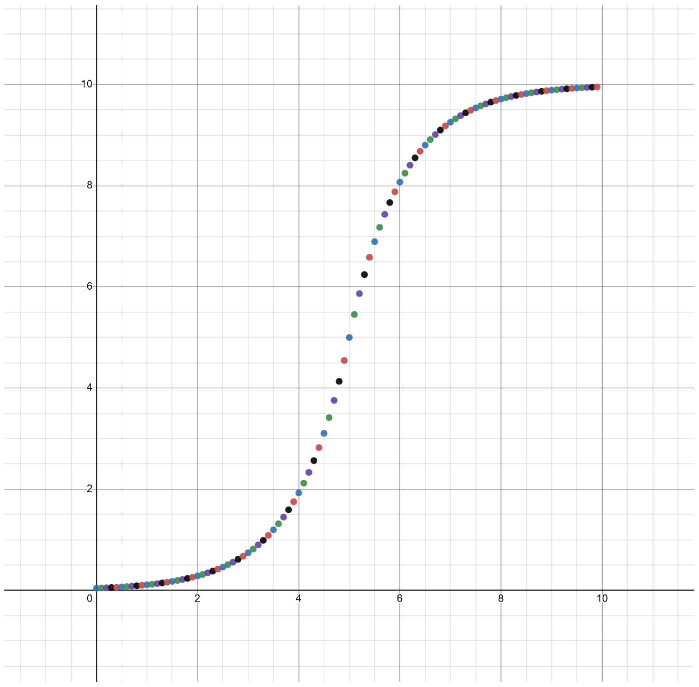


Figure 7: An acceleration mesh over 10 seconds, with a change in velocity of 10 m/s.

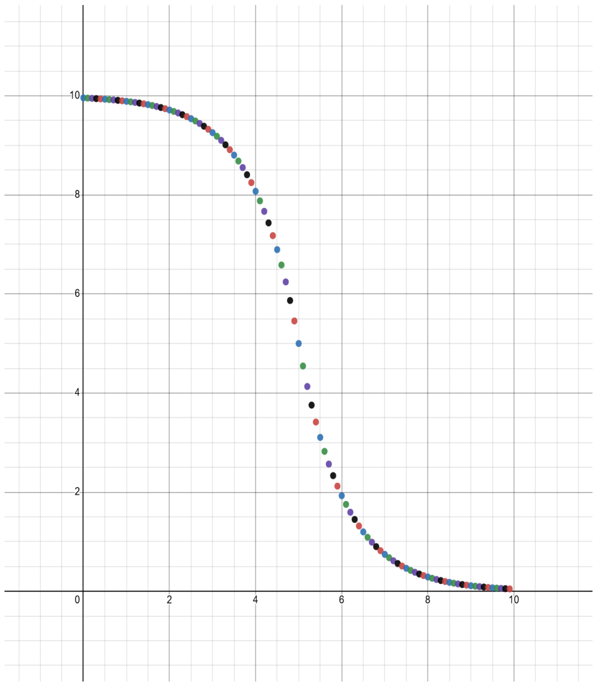


Figure 8: A deceleration mesh over 10 seconds, with a change in velocity of 10 m/s.

These meshes were then processed via interpolation using Natural Cubic Splines, as we wanted to ensure C^2 differentiability both to obtain a valid acceleration as well as to be able to evaluate jerk in the case of issues arising during simulation. After interpolation, we took the numerical derivative of the resulting splines in order to approximate the acceleration that the curve predicted, storing the values in another array. Graphically, these curves take the form of the following.

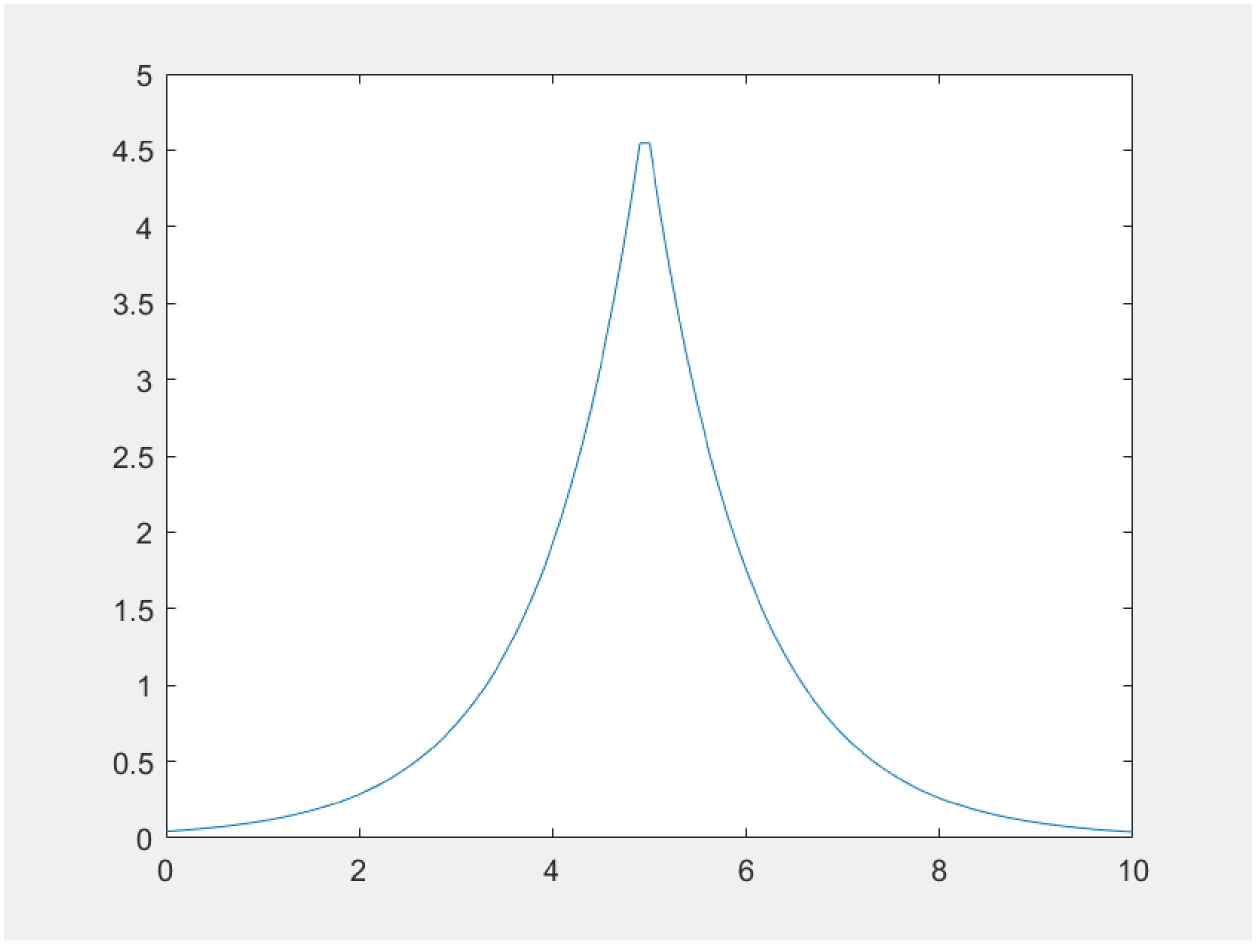


Figure 9: A typical acceleration graph resulting from the numerical derivative of the splines.

This array was then passed to an iterator loop to be processed for use by the vehicle.

IV. IMPLEMENTATION AND TESTING

A. Closed-Loop

Our controller was simulated in a closed-loop setting for both acceleration and deceleration. As mentioned before, the controller takes the car's velocity and the manually published set speed as input. Figure 10 shows an example of acceleration. The car's initial velocity is 20 m/s, and the set speed is 30 m/s. The car is able to steadily increase its speed to reach around 26 m/s within 10 seconds of the plot. It is not able to fully reach the desired 30 m/s because the acceleration our algorithm predicts is outside the safety acceleration range of -3 to 1.5 m/s^2 . However, it is a steady increase in velocity with minimal jerk.

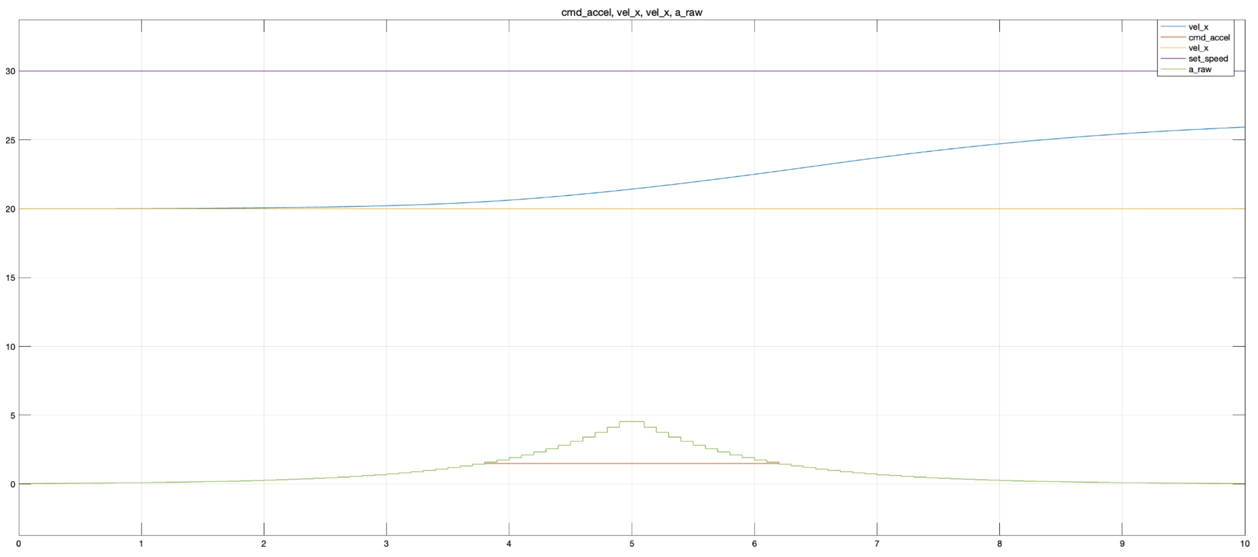


Figure 10: The closed-loop simulation for acceleration.

Similar to acceleration, the closed-loop simulation of decelerating has similar results in Figure 11. The car's initial velocity is 30 m/s, and the set speed is 20 m/s. The car steadily decreases its velocity to reach about 22 m/s. It does not fully reach the desired set speed because the safety range of acceleration previously mentioned prevents the car from fully decelerating to the desired speed. Overall, our controller seems to work as desired in a closed-loop simulation setting as the car nearly reaches its set speed based on its initial velocity.

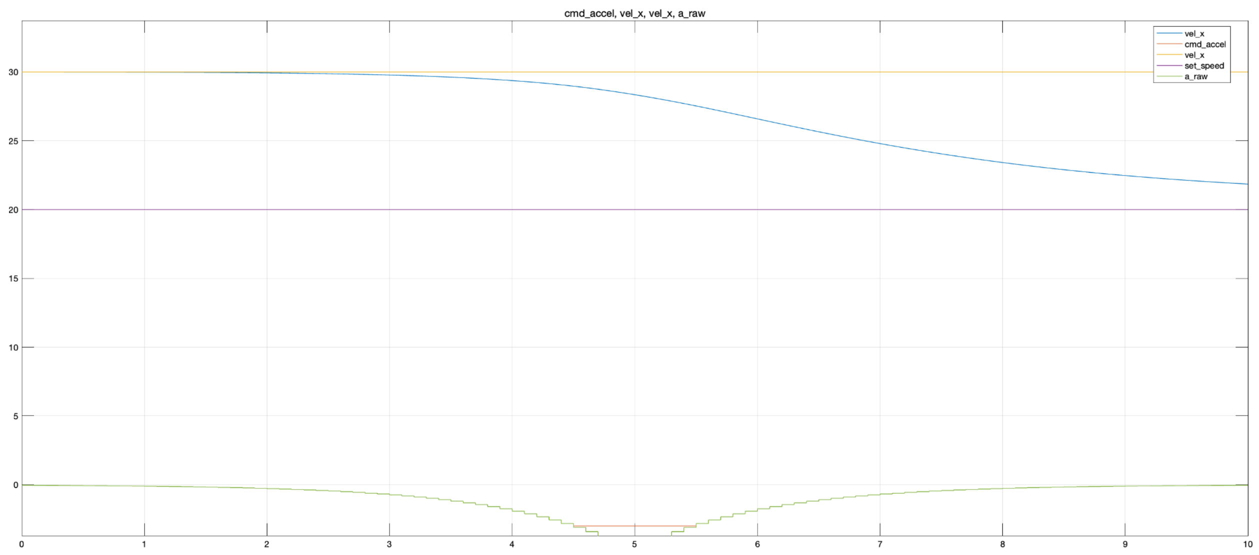


Figure 11: The closed-loop simulation for deceleration.

B. Open-Loop

Similar to the closed-loop simulations, the open-loop simulation works just as well. Because our algorithm creates an acceleration mesh with a 0.1 step size in a 10-second interval, our open-loop controller was only played for those 10 seconds. The car's velocity is initially at -2 m/s, and the set speed is at 2 m/s, as shown in Figure 12. The car reaches exactly the desired speed within 10 seconds, with the predicted acceleration staying within the safety range of -3 to 1.5 m/s^2. The acceleration in the plot shows similar results to the algorithm theory in Figure 9. Compared to the closed-loop in Figures 10 and 11, the car in this simulation is able to correctly reach its desired speed because the difference between the initial velocity and set speed is less than those performed in the closed-loop simulations. Thus, this open-loop simulation correctly shows how our controller can safely reach its target set speed if given enough time to do so with minimal jerk, as can be seen by the steady acceleration plot.

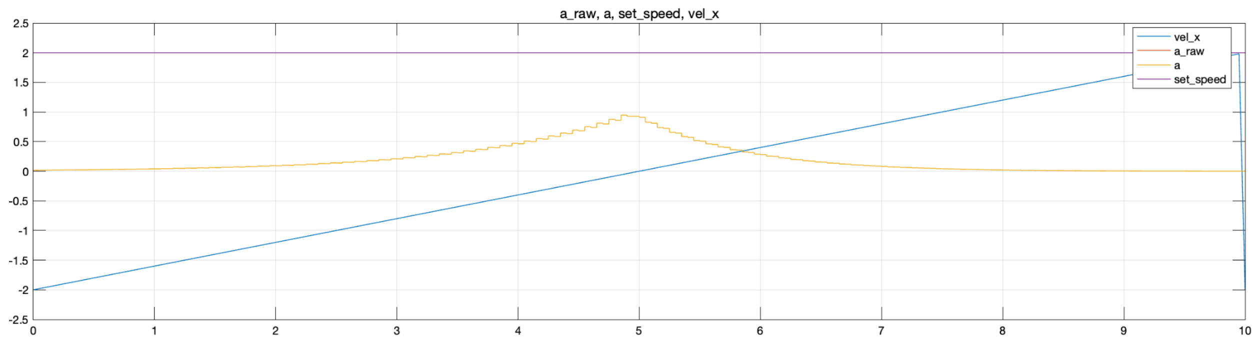


Figure 12: The open-loop simulation over a 10 second period.

C. Deployment Test Drive

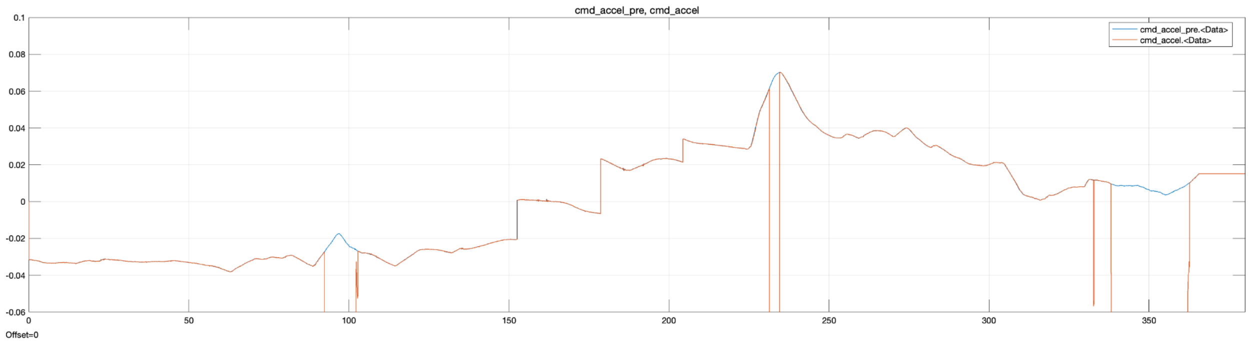


Figure 13: The acceleration topics plotted from the test drive.

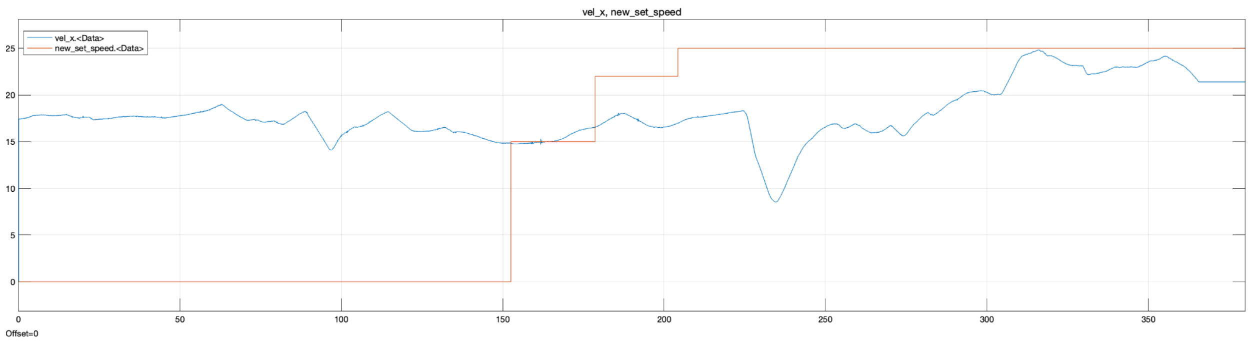


Figure 14: The velocity and set speed plotted from the test drive.

Our controller was deployed on the car as it was driving southbound on Franklin Road US 31, from Thompson Lane toward Old Hickory Boulevard. Our test drive assumed that there was no lead vehicle. Figure 13 shows the plot of the topics /cmd_accel_pre and /cmd_accel from the ROS bag. When these two match, our controller is running with the car's acceleration as /cmd_accel_pre. In Figure 14, the topics /vel_x and /new_set_speed are plotted. The /new_set_speed is a topic where the driver manually published a new plot via this command:

```
rostopic pub -r 10 /new_set_speed std_msgs/Float64 "data: 30"
```

The data entry is the desired speed to set in m/s.

The effects of our controller are starting to be shown when the /new_set_speed is set to 15 m/s at around 150 seconds in the plot. When this occurs, the car's velocity is already around that same value so no change to acceleration is really made, only small adjustments. At around 175 seconds, the /new_set_speed is now at 22 m/s. The resulting acceleration stays positive and allows the velocity to increase steadily. This set speed is not kept at a duration long enough to correctly adjust the velocity to match it with our controller. Finally, at around 210 seconds, the /new_set_speed is set to 25 m/s. The safety controller takes over for a few moments, resulting in the dip of the velocity at around 230 seconds as a lead vehicle appears. At around 245 seconds in the plot, you can see our controller take back over with our acceleration /cmd_accel_pre, as it steadily increases our velocity to hit our set speed at around 325 seconds. This proves that our controller is able to smoothly adjust our velocity with minimal jerk to get to the desired set speed. The safety controller takes over at around 330 seconds to the remainder of the plot. Therefore, our controller seems to correctly reach the desired /new_set_speed via manual publishing of this value if the set speed does not change very quickly.

V. CONTROLLER ANALYSIS

We believe that the idea behind this controller should be deployed in vehicles. While our current controller does not have any safety features to adapt if there is a lead vehicle, our controller shows proof that a vehicle can reach a new set speed steadily. As talked about in Section VI, instead of manually publishing a new set speed, future controllers should instead use their geographic position from the longitude and latitude topics to be able to change their set speed when coming across a new speed limit, different terrain, or a sharp curve for example. With the modifications to automatically adjust to the car's position and to be able to have the safety feature of adjusting to a lead vehicle, our controller would be able to have a positive impact on the driving community.

A main advantage of this is that this controller could aid in traffic control. For example, consider a busy interstate where the vehicles are adhering to a speed limit of 60 mph. The interstate suddenly changes to 70 mph. Only some cars will adapt to this new speed change, while others will continue to adhere to the slower speed. This could potentially cause a decrease in traffic flow and capacity compared to if all cars sped up to adjust to the speed limit. This can lead to a traffic jam, which can be seen in a time-space diagram where multiple cars have minimal change in position over a span of time. The mixture of cars adhering to the different speed limits could also fluctuate the time gap between cars as faster cars following slower cars may decrease the time gap to an unsafe value.

The concepts behind our controller would allow all cars to be able to adjust to the new speed limit change automatically. This would increase the flow and capacity of the roadway as all cars would be going around the same speed without slower cars causing traffic jams. With our controller logic, we predict that the time-space diagram would show much fewer traffic jams with minimal car slow-downs. The adjustments to set speed also allow the time gaps between cars to stay consistent between the changes from 60 mph to 70 mph on an interstate. Overall, our controller would aid in traffic flow with modifications to adjust for geographic location to replace the manual publishing of set speeds and to correctly handle a lead vehicle.

VI. FUTURE USE AND APPLICATIONS

A. Road Use

In its current form, the controller is entirely unsuitable for use on the open road, as it has no safety mechanisms. While it could aid in traffic flow by ensuring an ideal speed is followed to maximize throughput, it is not of any particular use when doing so will result in a collision. However, another iteration of the controller could put us on the path to solving this problem, as the inclusion of a gain modifier to influence the acceleration value prior to it being processed by the vehicle could allow for safety measures to be implemented. If a gain block is added, then it is also possible to add an error detection and correction block, which could detect when the vehicle is not meeting its projected velocity. It would then issue acceleration modifiers to the gain block in order to get it to accelerate or decelerate at the proper, predicted value. It would also be possible to do collision prediction with this model, and using the gain block, induce deceleration to avoid a collision, thus making the model safer. At this point, the model would ideally be suitable for road use.

B. Further Applications

If the gain, safety, and error blocks were to be implemented, then the value of the controller could be particularly realized. By utilizing a graph representation of roadways with built-in GPS data, the vehicle could accept speed data according to geographic location. This, in turn, could then be used for routing and pathfinding via well-known algorithms, such as Simulated Annealing and Dijkstra's, respectively. However, the weights of the edges may be modified to reflect a desired outcome; for example, if the edge weights are time, then the shortest path or route could be found. If they represent availability, with lower-cost edges being more available, then maximizing theoretical throughput for roads is possible, thus reducing overall traffic congestion. The controller finds its application through its ability to accept novel input, which in turn allows for a basis to solve far more complex problems.

VII. CONCLUSION

This paper presents our research on predictive velocity modeling and control in autonomous vehicles through building a set speed controller. Our project centered around a Simulink controller designed to interpret and process data (ie GPS Data) for influencing the vehicle's velocity. Through our research, we found that while predictive modeling shows significant promise, it is not without challenges, especially when contending with external factors like dynamics, drag, and vehicle power limitations, particularly at higher velocities. Due to our initial goal of using GPS data to influence speed, our design philosophy focused on creating a smooth transition between different velocities over a set period. We initially explored an analytic solution using Sigmoid curves but later shifted to a generative model due to practical constraints. This shift proved fruitful, as the generative model, aided by Natural Cubic Splines for continuous differentiability, allowed us to create a more adaptable and efficient control system.

The results from our simulations in both closed-loop and open-loop settings were promising, demonstrating the controller's ability to manage velocity changes with minimal jerk, thus prioritizing safety and passenger comfort. The real-world application of our controller, as evidenced by the deployment test drive, further validated our approach, showing the controller's effective adaptation to varying speeds in a dynamic environment. However, our research also highlighted significant areas for future development. One such area is the integration of safety mechanisms into the controller to make it viable for real-world applications, particularly in open-road scenarios. Another exciting direction is exploring further applications of our controller in larger traffic management systems. By integrating geographic data for speed adjustments and leveraging routing algorithms like Dijkstra’s or Simulated Annealing, we can potentially contribute to reducing overall traffic congestion and enhancing road safety.

In conclusion, our project lays a foundation for future advancements in the field of autonomous vehicle technology and traffic control. By continuing to refine our controller, integrating safety features, and exploring broader applications, we believe we could achieve the same level of more advanced speed controllers already implemented in vehicles today.

VIII. Contributions

All members of the group contributed to presentations and reports.

[redacted]

[redacted]

IX. Relevant Links

Controller Analysis Video

[redacted]

GitHub Repository

[redacted]