From: Nicholas DePatie

CS1800

4/18/2021

Final Project – Simulink Model for Autonomous Car

Attachments: 7

For this project, I used a program based in MATLAB called Simulink to model the behavior and logic behind an autonomous, self-driving car approaching an intersection. Simulink allows the user to model dynamic or discrete systems using the integration or recursion of inputs.

Included with this report is (1) the actual Simulink model, (2) truth table used to write the logical expressions, (3) the logical expressions themselves, (4) the logic diagrams, (5) a picture of the Simulink model for viewing if MATLAB is not installed with a few notes about Simulink in general, (6) pictures of the models inside the subsystems, and (7) the data received from the model in the form of graphs over time.

The project was a success, and I was able to model the speed and distance traveled of a car going through a busy intersection. I am an Electrical and Computer Engineering major, so I was excited to use software I was already using and apply discrete mathematics to it, and I am satisfied with how it came out.

Brainstorming:

The general concept I was working with was that a car approaches an intersection and must speed up, slow down, stop, or do nothing based on a series of inputs. Finding an adequate number of inputs was a bit difficult, but I decided on considering:

1) the color of the traffic light (Red, Yellow, Green)
2) the speed of the car and comparing it to a 40mph speed limit (Greater than, Less than, Equal to)
3) the distance the car is from the intersection (Far, Approaching, Close)
4) pedestrians walking across the road (Walking, Standing)

With the 4 inputs, and 3 of them being tertiary inputs and 1 being binary, there are 54 possible situations that the car can face when approaching the intersection. From the 54 situations, I would need to decide what the car should do in each scenario and translate that into a behavior of the car:

1) car slows down
2) car speeds up
3) car comes to a complete stop
4) no change

Another important point is that I decided not to model the torques of any motors or engines in the car and use constant acceleration commands to model the behavior of the car, allowing me to focus more so on the logic behind everything. With Simulink, I could have done so, but it would have gotten even more complex.

Truth Table (see document (2)):

The first step of figuring out the logic behind the car is to create a truth table of all the possible situations and determine how the car should react. Since there was a lot of possible scenarios, I used Excel to create the table. Since the car would never need two behaviors at once, only 1 behavior was chosen for each situation and highlighted to help identify any patterns. The table was also divided into 3 sections based on the distance from intersection input which helped to write the logical expressions.

Logical Expressions and Simplification (see document (3)):

The next step was to create a series of expressions to define the logic. Using the sections created in the truth table based on distance, I created a small table that created sub expressions for each behavior and distance. These were then combined and simplified using logical laws of equivalence (see document above for what exactly was done). This process resulted in 1 unique expression to define the logic behind each behavior.

Interestingly, the yellow and red lights were found to cause the same behaviors in the car, meaning the traffic light input could be replaced with a semi binary output, green or not green. The stop command was also found to be the simplest expression, as well.

Logic Diagrams (see document (4)):

The logical expressions were then drawn out to make logic diagrams. In the document above, the logic diagrams were made using Simulink and are the actual diagrams that were implemented into the final product.

Model and Subsystems (see documents (5) and (6)):

To create the model, I had to lay out a few assumptions to simplify the model. For this model, the pedestrians are unruly, meaning they do not obey the traffic light when walking. Also, the car only travels straight in the positive x-direction.

I had to decide how to model the inputs of the model. For the traffic light, I decided on a step function that had 3 states: 0, 1, and 2. These were used to represent the lights (i.e. 0=yellow, 1=red, 2=green) by using a relational operator to see if the signal equaled 2, or the light was green. A binary pulse input was used to model the pedestrians (i.e. 0=not walking, 1=walking).

Modeling the speed and distance was a bit trickier since they were both inputs and outputs. I ended up creating a series of switches that integrated a certain acceleration value over time depending on what behavior the car should carry out, with an initial speed value being summed with the integral. The speed was then integrated to find the distance traveled.

These values for speed and distance were reduced to binary values through relational operators. For the speed comparison, the speed would never be exactly 40mph and testing for such a value would be unreliable. So, the value for "=40 mph" was determined using a small interval from 39.5 to 40.5 mph. For the distance comparison, the distance traveled was subtracted from the location of the intersection (i.e., intersection located at x=1000ft). This

value was compared to 150 ft and 360 ft. It is also important to note that early on when I was simplifying the logical expressions, I used the values 50 ft and 100 ft for distance comparisons, but I realized later that these distances are somewhat impractical while driving. I also added an additional comparison to 0 ft, so if the distance from the intersection was negative, or the car was past the intersection, then the value would be true. This removed any odd behavior after the car had gone through the intersection. The speed calculation and relational operations were made to be in subsystems to make the model easier to read. The logical expressions in document (4) were also put into subsystems and incorporated into the model using the values from the various inputs described above.

Lastly, a few scope blocks were added to be able to show the outputs of each signal and the speed and distance values. These graphs are shown in the next section about data.

Data from Model (see document (7)):

The data received from the model is actually very reliable. For each of the cases in the above document, the speed of the car (mph), distance the car had traveled (ft), traffic light signal, and pedestrian signals are shown.

For case 1, the initial speed was set to 0 mph and the intersection was located at x=1000 ft. From the data, the car does speed up to 40 mph and stop at the intersection, because it arrives at the time that there was not a green light, $t \approx 24$ seconds. However, the car does not move when the light turns green at t=120 seconds. Looking at the pedestrian input, however, pedestrians are still crossing when the light turns green, and the car starts moving when there are no pedestrians at $t \approx 135$ seconds. This means that the logic behind the car works!

To make sure the simulation works well, more cases must be tried. For case 2, the initial speed was set to 45 mph and the intersection was located at x=1000 ft. This time, the car slows down to 40 mph and stops again at the intersection. It exhibits the same behavior as in case 1, where it waits for pedestrians to cross again. In case 3, the initial speed was set to 20 mph and the intersection was located at x=11000 ft. I wanted to see how far the car could go. Turns out, the simulation works when the intersection is that far away, again! For case 4, I tried semi-random numbers with the initial speed being 29 mph and the intersection located at x=12345 ft. Again, the car stops at the intersection, waits for the light and pedestrians, then continues moving.

Overall, I am satisfied with the results of this simulation and consider it a success for what it was. Of course, this simulation was incredibly simple and autonomous cars need to consider many more inputs. If I were to continue to improve this model, I might have the car consider whether or not it was raining so it would stop sooner because of the slipping. I also might add in the ability for the vehicle to detect cars around it, duplicate the simulation for multiple cars, and simulate a traffic jam. However, like I said before, I am satisfied with what I did to make this simulation work and am excited to see what else I can apply discrete mathematics to in my future engineering endeavors.