Aaron Pollon
Matt Scaperoth

Smart Car Simulator

Abstract - The use of technology is increasing in modern day infrastructure. Because of this, there is a need for study in learning algorithms for reactive decision making in traffic simulations. These algorithms not only make the car drive safely on the road, but can potentially solve problems caused by other vehicles that may not be following the same sort of rules.

The traffic simulator simulates an environment with vehicles that can react to human error or mechanical malfunctions in traffic flow. The goal of the simulation is for a vehicle, that we call “law-abiders” that is staying under a certain velocity, the speed limit, to be able to automatically make decisions that will result in a vehicle, which we will refer to as a “speeder”, from traveling faster the speed limit.

For consistency in terminology, the main variables in the simulator that are expressed in the animation and calculations are the number of lanes, the wait time -- time between adding a new speeder or non-speeder vehicle , lane width, and number of vehicles that are considered “smart cars”.

I. Literature review of related work

A. Graphical models for driver behavior recognition in a SmartCar
In the traffic simulator, decisions are made to strategically decide the best next move for the vehicle to stop a speeder in the traffic. IEEE published a report for the IEEE Intelligent Vehicles Symposium 2000 titled “Graphical models for driver behavior recognition in a SmartCar” (http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=898310). The concept behind this research was that data models could be built off of real-time data resulting in a set of tactical maneuvers for urban driving. The data acquisition system, called SmartCar learns the maneuvers from human drivers and reportedly, after a learning session, on average can predict the next required action 1 second before the human driver. This report is similar to the traffic simulator in that it is trying to predict the best action in a given driving scenario. The learning algorithm proposed in the report, however, is based on processing real-time environments and applying processed tactics to the current situation and the traffic simulated mentioned in this report is a simulation built on the idea of reactive maneuvers and actions. The system created with SmartCar applies machine learning with real life situation analysis. Since real driving behavior is being observed and learned, a more realistic simulation and decision making process is created. A goal for future expansion of our project would be to look at real driving behavior analysis such as SmartCar, which would optimize autonomous vehicles by beginning with existing driving behavior and looking for improvement.

B.SITRAS and Lane Changing and Merging:

One component of our simulator is to model a realistic lane changing movement and a decision making process. Since amongst our many assumptions is having a straight road, and our goal when changing lanes is to block a speeder, our lane changing algorithms are much simpler when compared to lane changing simulation as part of a simulation at the University of New South Wales (http://www.sciencedirect.com/science/article/pii/S0968090X02000268). The lane changing algorithms are one component of the much larger "Simulation of Intelligent TRAnsport Systems" (SITRAS), which models autonomous vehicles. It can be used to test Intelligent Transportation Systems (ITS) which are implemented with the goal of lowering congestion using existing infrastructure. SITRAS has been in developed at the University of South Wales since 1995. Peter Hidas' paper focus specifically on lane changing and merging situations, as contrasted with a simple lane change with no competing car, therefore no merge needed. A common situation is in highway congestion. SITRAS distinguishes between two types of lane changes: forced and cooperative. The general procedure of the SITRAS lane changing model shares some characteristics with our model. Like in our model, movement in SITRAS is step based and at each step a car determines if a lane change is needed. SITRAS adds a more pricise element by specifying if a lane change is either "desirable" or "essential." Next, just as in our model, a feasibility check is done to determine to see if there is a car in the desired lane that prevents the lane change from occurring. If feasible the lane change occurs, but if not then only if the move is "essential" will a "driver courtesy" be simulated in order to change lanes. The simulation includes an interesting algorithm that models driver courtesy alone, which determines whether the requested is granted or denied. This algorithm takes into factors such as speed and acceleration as well "driver type", which suggest the aggressiveness of the driver. The SITRAS is also similar to our simulation in that each step produces a series of "if" statement conditions that if met, alter the acceleration of the car.

C. FreeSim - a free real-time freeway traffic simulator

The traffic simulator uses data provided by the cars in the simulator to calculate certain decisions while maneuvering along the highway. A system called FreeSim a traffic simulator that simulates large scale traffic systems is described in a report published by IEEE for the Intelligent Transportation Systems Conference in 2007 (http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4357627). FreeSim is a robust Java application with a MySQL backing for persistent data used to accumulate and process large quantities of observed results. FreeSim reads the data from the cars found in the traffic system and calculates optimization options including shortest paths and fastest paths . The traffic simulator mentioned in this report implements a similar method to gain metadata about the traffic from the sum of the individual active parts. In FreeSim, each car is an individual part that sends its data, such as location and speed, to a central server. The simulator from this report gathers data from each car object once the SmartCarSimulator class updates its current information. Unlike FreeSim, though, our traffic simulator's primary focus is the logic behind the proper reaction to a given scenario rather than only collecting and displaying data.

II. Project Description

A. Java Classes Overview

Written in Java, the simulator uses a Graphical User Interface (GUI) to display the results of each step in the calculation of the scenario. The traffic simulator graphically demonstrates that a vehicle's intelligent reactions to speeders can possibly slow down, or prevent, a non-malicious speeder from driving at a velocity above the speed limit for an extended period of time.

There are four classes that make up the simulator with another, fifth class, that acts as the random number generator. The four classes draw the main environment and controls in TrafficSim, draw the inner environment in Road, create each smart car object in SmartCar, and simulate the animation of the smart car(s) in SmartCarSimulator. The TrafficSim class creates an initial JFrame with the controls "Reset", "Go", "Pause", and "Quit". Once this frame is created, the Road class is initialized and passed variables for the speed limit, number of lane, lane width, debug, and the traffic simulator. Using these variables, in the TrafficSim's paintComponent method, the road draws itself in the middle of the screen. After the road is drawn and the "Reset" button is clicked, the first smart car is drawn off screen using a random car image from the "images/", the simulator is initialized, and the statistics are reset. Once "Go" is selected, the simulation begins. During the simulation, TrafficSim generates cars in random lanes with a random chance that it is a speeder.

The Road class acts as a data bank for all cars. Using the data stored in Road as sensors, a smart car can query about the metadata for all other smart cars on the road. This data includes which cars are speeders, the cars' x and y values, the number of lanes, and various statuses for each car.

The SmartCar is an instance of a single car on the road in the simulator. This object contains the decision logic for the vehicles displayed in the simulator. Since the car is passed the road, which is passed the simulator, the car is privy to information such as the speed limit, number of lanes, lane width, and various other environmental data that one may expect a car to know about its surroundings. The car acts in the move() function where all of the primary decision making is made and calls various a number of functions. These logic of these functions will be covered further in the " Program Logic and Algorithm" section.

The SmartCarSimulator animates each SmartCar within the main panel. SmartCarSimulator calculates, based on the most recently assigned velocity and angle, the car's next position on the road. Along with animation, the simulator is also in control of tagging which car should be removed from the animation once it reaches a certain threshold as well as drawing each SmartCar.

B. Program Logic and Algorithm

Each of the two types of cars ("Speeder" and "Law-Abiders") behaves slightly differently. Upon no interaction with each other, all cars seek to stay in their lane and travel and their starting speed. Every move is checked to see if the car is getting too close to another. If too close, a

law-abider will simply slow down, while a speeder will attempt to change lanes if possible. A car will only change lanes if there there is sufficient space in the next lane.
The law-abider's goal is to try to catch speeders by forcing them to slow down. Every move a car is looking find the closest speeder that is behind them in any lane and within a pre-determined threshold distance. If a speeder is detected, the car will then see if there is another car in the speeder's lane to block them. If not, the law-abider will change lanes towards the speeder if possible. If there is a car in front of the speeder, if changing lanes is not possible, or if the law-abider is in the same lane as the speeder, the car will slow down until directly in front of the speeder. The car slows even if it is not in the same lane as the speeder in order to block the speeder in and prevent it from going around a block. Once the law-abiders are in front, they will maintain its speed. Law-abiders also look to see if slowing down will cause a car to become directly behind them. If so, and that car is not the speeder, then the law-abider will change lanes toward the speeder, if possible. The simulation models acceleration of cars when changing to its desired speed.

Once a speeder is unable to change lanes and is forced to slow down, it has been "caught." The simulator tracks how long a speeder has been caught and after a pre-determined amount of time caught, the speeder is changed to a law-abider and the other cars will resume their original speeds and default behavior

III. Results

Currently, the visible success of the simulator is dependent on not only the number of lanes involved, but the ratio of speeders to law-abiders. As a visual result it was noticed that If there are a large number of speeders, larger than a percentage equivalent to 10*number of lanes being generated, then the likelihood of law-abiders catching these speeders goes down. For example, if there are 3 lanes and the percentage of speeders being generated is above 30%, then there may not be enough law-abiders to catch the speeders. Further data gathering is required to find the optimum ratio of cars to speeders for the simulation.

The algorithms used to execute a proper "catching" of a speeder can be improved upon. There are certain cases in which the speeder is not caught due either to a miscalculation or noisy data from other cars including multiple speeders around a similar y point.

In future versions, a rapid simulation will be developed to run the scenario for a set amount of time in order to capture data points such as the number of speeders caught and the average percentage of speeders caught over x iterations of the full simulation.