Automated Intersection Management For Self-Driving Vehicles With *Q-Learning*

Introduction

In an age of rapid urbanization, traffic and the speed of transportation becomes a growing issue around the world. However, traffic lights are manually programmed, are inefficient and cannot respond to dynamic traffic patterns. Alternatives with reinforcement learning have been tested, but models do not have access to the exact number of cars in each lane and their destinations. As a result, automated intersection management(AIM) systems, designed to direct vehicles integrated into the traffic system, are not currently feasible.

Proposed Solution

- An AIM could improve the effectiveness and safety of self-driving vehicles. The cars could provide data to the AIM to improve traffic direction.
- Q-learning trains based on arrays of data and accounts for temporal differences, making it a suitable form of reinforcement learning for intersection management..

Criteria

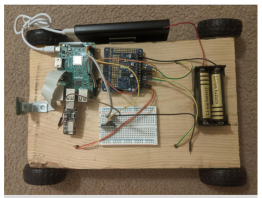
- A traffic simulation with vehicles providing positional and destination data to the AIM.
- A physical self-driving car driving on a to-scale model intersection evaluated qualitatively for testing the AIM.
- Train multiple Q-learning models with different training variables optimizing for training speed and model performance.
- Models will be tested in traffic patterns with different major lanes and compared to a control based on manually programmed intersections.
- All threaded processes should run smoothly on a 2.9 GHz processor.

Design

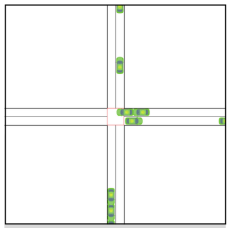
Self-Driving Car

1. The first prototype(pictured right) had power issues which were solved by the final design(pictured below).
2. Gyroscope sensor and Bluetooth Low Energy(BLE) sniffer added for indoor positioning system(IPS) purposes. The camera was upgraded as well.
3. The wiring uses a breadboard and a detached motor HAT which solved an I2C connection error.

First prototype of vehicle; 4 DC motors and wheels, USB webcam, portable charger, 6V battery pack, Raspberry Pi 3, and Adafruit motor HAT.
Dimensions: 11in x 7in



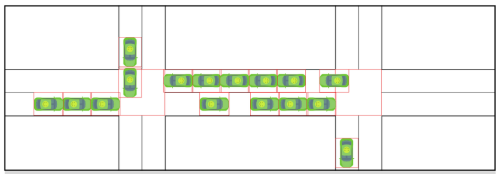
Final vehicle prototype; Arducam, Raspberry Pi 3, Adafruit motor HAT, USB BLE sniffer, 4 DC motors and wheels, gyroscope sensor, 6V battery pack, 5V power supply.



First traffic simulation used to train models.

AIM

1. The AIM uses Pygame to draw a 2 lane, 4-way intersection with lines and hitboxes(pictured top right) to train the first set of Q-tables.
2. The final simulation(pictured below) uses two intersections and trained the second set of Q-tables.
3. Car objects spawn randomly in lanes and navigate to a random destination.
4. The destination and position data of cars is available to the traffic direction program in arrays.



Final traffic simulation; one Q-table trained for each intersection in the second set of models.

Self-Driving

- Lane finding converts an image to grayscale, applies a Gaussian blur, and uses Canny from the OpenCV library to find gradients.
- A bitwise_and function is applied on a mask of the region of interest, outlining gradients.
- The HoughLinesP function identifies lines in the new image. The two longest lines of opposite slopes form a quadrilateral as the lane.
- Curve finding uses pixel summation, the sum of nonblack pixels in each column, on the lane finding image to calculate how much the lane curves towards a side.
- The curve value generated is applied to motor speeds, subtracting from one side and adding to the other.
- Object detection uses a convolutional neural network(CNN) to detect common objects that appear while driving that are too large, and therefore too close, in the video.

Methodology

Q-learning

- Training was conducted for 10,000 episodes on randomized traffic patterns.
- Rewards: -20 penalty if 4 vehicles in a lane, -10 penalty if no vehicles moved between observations; -15 penalty in any car does not move for too long also tested in first set of models trained.
- Variables Tested: double vehicle speed, no simulation, double Q-learning, and using a second type of Q-table; attempted to optimize training speed and performance.
- Double Q-table learning performed best; a second set of Q-learning models was trained testing double Q-table learning and double Q-learning, which prevents overoptimization of future values.
- New models were tested for 500 episodes against the control based on various intersections' traffic light patterns.

Double Q-table Learning

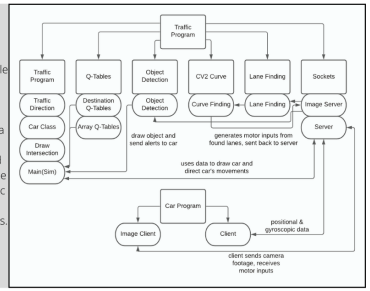
- Novel learning method designed for the AIM, uses one Q-table for the number of cars in each lane and one for the destination of the frontmost cars.
- The Q-values for both observations are added together to find the optimal action to take.
- Destination Q-table was not trained; a weight may also be applied to affect destination table's influence on actions.

$$Q_{new}(s_t, a_t) = Q(s_t, a_t) + lr(r_t + \gamma(maxQ(s_{t+1}, a_t) - Q(s_t, a_t))$$

old Q-value · learning rate · discount · max new Q-value · old Q-value

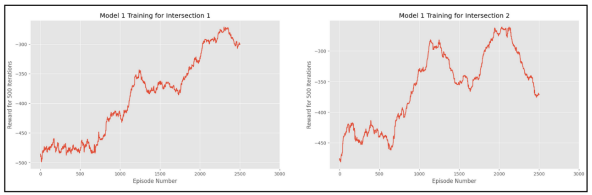
Q-Learning Equation

Full code and file hierarchy; connections between programs, data sent between programs and between vehicle and main traffic program through sockets.

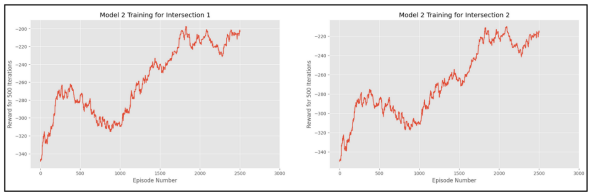


Results

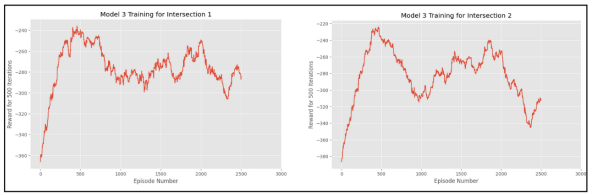
Model 1 Training Results



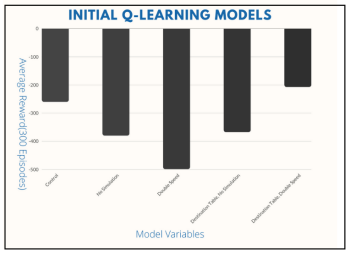
Model 2 Training Results



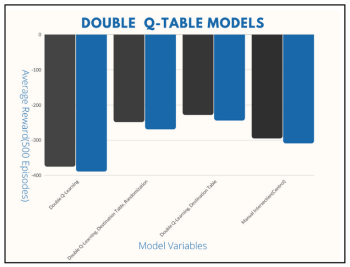
Model 3 Training Results



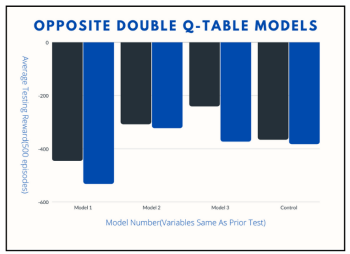
Original Q-Tables



Final Q-Tables Normal Vehicle Spawn Rate



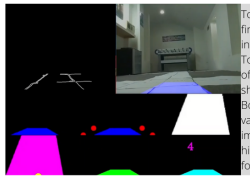
Final Q-Tables Inverse Vehicle Spawn Rate



Object Detection

Uses a CNN to detect common objects while driving, can detect cars, traffic lights, and pedestrians; boxes objects and shows level of certainty; will also show warnings when an object is too close.

Self-Driving



Top left: gradient finding in region of interest.
Top right: finding of lane lines and shape.
Bottom: curve value finding; image warping, histogram function for center.

Vehicle driving on intersection.

Analysis

AIM

- The AIM's 12 car objects, sockets, and lane finding processes ran smoothly on a 2.9 GHz processor, 8 GB RAM laptop together.
- All attempts to speed up training decreased model performance. The model with slight randomization applied to Q-values for solving issues with no vehicles moving did not perform better than others.
- The double Q-table models performed better than the control on several types of intersections besides the only double Q-learning model, likely because it was tested with a destination Q-table but wasn't trained with one.
- The second set of Q-tables trained faster overall due to double Q-learning minimizing overestimation of future Q-values and double Q-table learning's effective usage of information.

Self-Driving Car

- The vehicle could navigate across the roads end to end fairly accurately. The curve value finding was always accurate, and the lane finding was effective but would struggle with reading lines in an intersection. Navigation should be improved with the AIM's assistance after integration.
- The car's improved design solved issues with power supply, I2C communication, and added gyroscopic and positional data for the AIM's IPS.

Conclusion

The traffic program was able to provide data and conditions that were optimal for training an effective traffic-directing reinforcement learning model. The final Q-learning models performed better than the controls and were able to perform effectively in traffic patterns with different major lanes than trained on. The supplementary destination Q-table was effective in aiding the efficiency of traffic direction and produced great results without the use of complex calculations, making it perfect for application in an AIM. The vehicle's self-driving was both simple and relatively effective, being able to navigate the model intersection even without integration into the AIM. With the advantages of the AIM over traditional intersection management and the simplicity of the self-driving AI designed to integrate with the AIM, this client-server AIM and vehicle AI can soon be deployed feasibly and optimize current traffic management.

Application

- The AIM could apply to intersections as a hybrid system, aiding traffic direction using object detection on traffic cameras to count vehicles in lanes. Cars could provide GPS route data to show their destinations.
- A car object can represent multiple cars at once. Only a camera, GPS, and computer are needed for the self-driving, and both programs do not use much processing power, making it cost effective and feasible.
- The Q-learning model could continue to train to fit each unique intersection.
- The AIM would improve safety at night and in low-visibility conditions.
- If all vehicles were integrated, safety of self-driving vehicles, traffic management efficiency, and traffic management costs could be improved.

Future Works

- After finishing the IPS' triangulation, the car object will be adapted to the car, and driving will be assisted by hitboxes.
- Curve finding use a horizontal mask and circle drawing for turning at intersections. Object detection will be incorporated into the driving and simulation as well.
- In the future, a CNN could be tested for the car, and a CNN or DQN could be tested for the simulation by replacing the car sprite with its turn direction.
- Pedestrians and more complex intersections with more lanes will also be incorporated in the future.

References

Chen, C., Wei, H., Xu, N., Zheng, G., Yang, M., Xiong, Y., Xu, K., & Li, Z. (2020). Toward A Thousand Lights: Decentralized Deep Reinforcement Learning for Large-Scale Traffic Signal Control. https://chacha-chen.github.io/papers/chacha-AAAI2020.pdf.

Liu, M., Deng, J., Xu, M., Zhang, X., & Wang, W. (2017). Cooperative Deep Reinforcement Learning for Traffic Signal Control. https://urbcomp.ist.psu.edu/2017/papers/Cooperative.pdf.

Matiisen, T. (2015, December 19). *DEMYSTIFYING DEEP REINFORCEMENT LEARNING*. Computational Neuroscience Lab. https://neuro.cs.ut.ee/demystifying-deep-reinforcement-learning/.