Intersection of Autonomous Vehicles

Shushmita Bhattacharya **Rachel Dedinsky** **Mohammad Farazi**

Mohammad Khayatian **Skyler Rothman**

Abstract

Intelligent multi-agent systems have the potential to change the world's way of vehicle travel, making it safer and more efficient. The state of the art approach to managing connected autonomous vehicle traffic through intersections is Robust Intersection Manager(RIM)[1]. RIM uses a centralized intersection manager to assign velocity of arrival and time of arrival to vehicles requesting to safely cross the intersection. This method is robust against external disturbances and model mismatches[1], and requires lower network bandwidth compared to many other centralized intersection management approaches. RIM implemented not only a software simulation of its approach, but additionally produced a hardware simulation. The perception mechanism for the hardware simulation utilized GPS and wheelbase encoders, which wasn't accurate enough to determine the positioning of vehicles and so a demonstration could not be clearly illustrated. The Perception in Robotics class project gave the team an opportunity to showcase the ability of perception in determining more accurate positioning data for vehicles, resulting in a verifiable demonstration of RIM. This hardware simulation used the OptiTrack Motion Capture system to provide perception for the positioning of the vehicles so the IM could make more informed decisions for scheduling and vehicles could more easily follow the road. Robot Operating System (ROS) and Transmission Control Protocol were used to implement the communication between OptiTrack, the IM, and the vehicles. From this hardware simulation, the vehicle throughput observed was 1.875s and no accidents occurred from the 20 hardware simulation runs.

1 Introduction

In the future, traffic intersection designed to accommodate Connected Autonomous Vehicles (CAVs) should implement an intelligent multi-agent system to achieve higher throughput and safety of vehicles. Currently, intersections spend much of the time with travel lanes stopped using red lights. To increase throughput, vehicles may be allowed to go through the intersection without being required to stop, while only needing its speed adjusted to avoid hitting other vehicles simultaneously traveling through the intersection. Currently, CAVs are not deployed on the street but researchers have shown the superiority of intelligent multi-agent systems over a traffic light using simulation and scaled models. However, current implementations are not very effective because of the lack of accurate localization devices. Two prominent approaches to intersection management are cooperative and centralized, where cooperative is a many-to-many communication between cars in order to create a scheduling policy and centralized is a one-to-many communication between intersection infrastructure and vehicles. Cooperative approaches are relatively less popular because of security issues associated with solely relying on connected vehicle data. Centralized approaches use an in-

[1]The IM calculates the assigned value of velocity based on the vehicle's dynamics model, meaning if this model isn't able to abide by the perceived model it is vulnerable to a model mismatch.

tersection manager(IM) to communicate with vehicles interacting with the intersection in order to create an efficient and safe schedule for vehicles.

The state of the art IM is called RIM[1]. In RIM, vehicles send current velocity, position, and timestamp and receive from the IM a velocity of arrival(VOA) and time of arrival(TOA). This provides CAVs a flexible trajectory since it is robust to external disturbances and doesn't assume positioning constraints, instead only assuming velocity and timing constraints. The calculated VOA and TOA is the only hard-constraint on the system since scheduling is calculated based on the assigned VOAs and TOAs of other cars in the intersection, ensuring that no two vehicles have overlapping trajectories in the time-space domain. RIM, like many other systems, is a cyber-physical system and relies on hard-timing constraints. This means to best test the system works as expected, it is best to run the system in a physical environment in addition to software simulation.

In order to test this research in the real-world, a 1/10 scale model intersection was created using TRAXXAS RC cars as CAVs, a motion capture system for positioning, and Robot Operating System(ROS) as the communication link. The IM is implemented using MATLAB to communicate to the cars over ROS topics. The OptiTrack Motion Capture System is used to get an accurate position of all cars used within our work area. The cars communicate with the IM over Wi-Fi and the OptiTrack Motion Capture System communicates with the intersection manager using ROS. We observed that the RIM algorithm leads to zero accidents given 20 runs with six cars.

2 Approach

Vehicles need to know their position for navigation and when they want to communicate with the intersection manager. The intersection manager node reads the real-time position data of all vehicle using the mocap_OptiTrack library and broadcasts it to vehicles. The data has Pose2D format consisting of position (x,y) in meters and heading in Radians. Figure 1 shows an overview of our intersection management system.

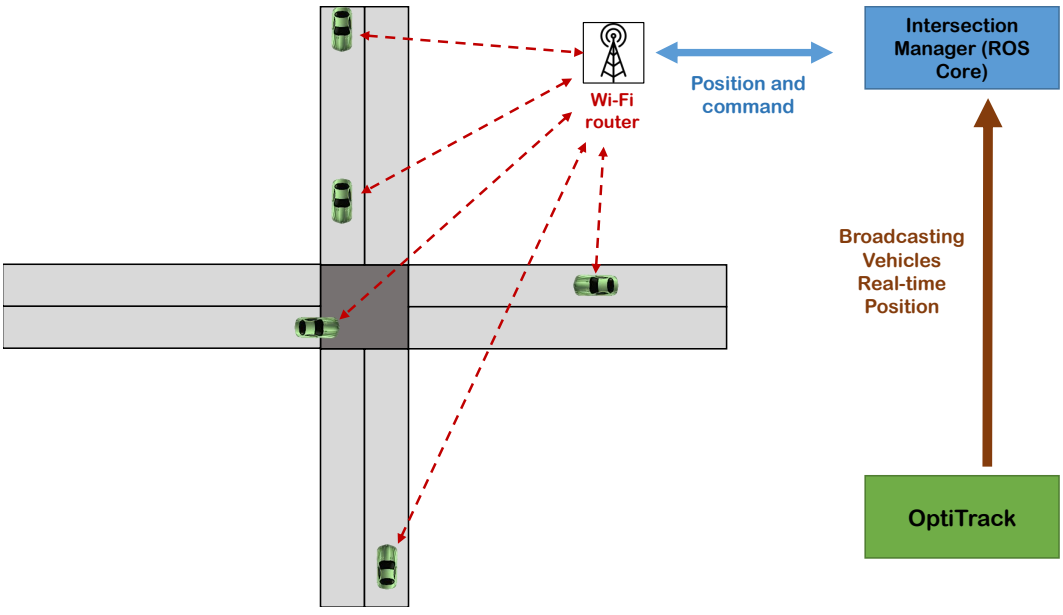

Figure 1: An overview of how the interaction between the different components in the intersection simulation will work together to efficiently and safely manage traffic.

Figure 2: An photograph of one of the several cameras used in the OptiTrack camera array.

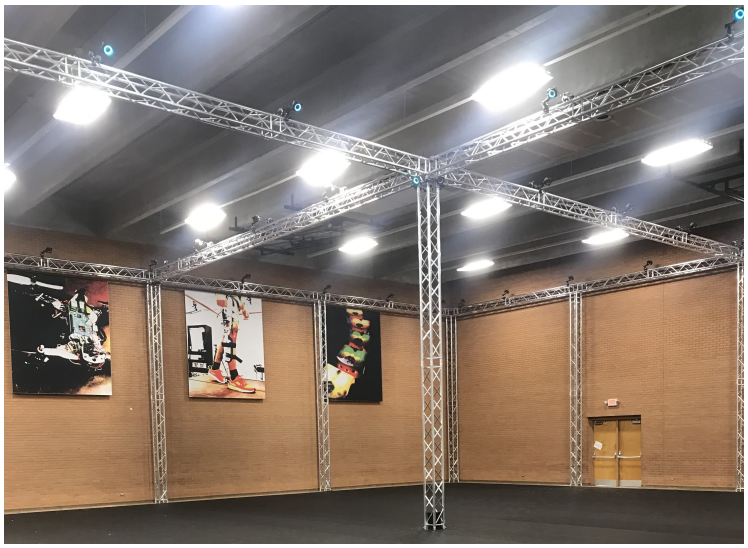

Figure 3: The array of OptiTrack cameras in the Gym Lab.

2.1 Perception Module

OptiTrack[2] is a motion capture system that uses IR cameras as well as reflective trackers to track motion. The system combines the data of several 2D "images" of tracker locations to "reconstruct" the 3D location of each of the trackers. To do this, the system uses several triangulation calculations with its *Point Cloud Reconstruction Engine* to estimate the 3D coordinate for each tracker with respect to an arbitrarily set origin. Our setup uses an array of 17 of these IR cameras set up as an array of a roughly square area on a steel scaffolding within a large room. Figure 2 shows a photograph of what one of the cameras look like and Figure 3 shows the structure with installed cameras. The data of these cameras routed through a network switch and ultimately end up processed on a dedicated computer connected to the same switch. The *Motive* software does the bulk of the processing and provides as the interface for us to visually see the tracking of known objects in real-time, as well as export the position data for use outside of Motive.

For tracking the cars, retro-reflective markers are attached to each of the cars which stay in a fixed formation. Motive software allows us to assign these formations as "rigid bodies" which is then recognized as a discrete object for tracking. These rigid bodies have their own approximated 3D location, which is broadcast as several ROS topics, one for each individual rigid body. Figure 4 shows what real-time tracking looks like inside of Motive while we have several cars in the work area at the same time.

2.2 Control Module

The Intersection Manager (IM) creates a ROS node which subscribes to OptiTrack ground poses to get location information of each car. The information contains the x,y position of each car. The IM

[2]Link to Optitrack Documentation Wiki:
https://v21.wiki.optitrack.com/index.php?title=OptiTrack_Documentation_Wiki

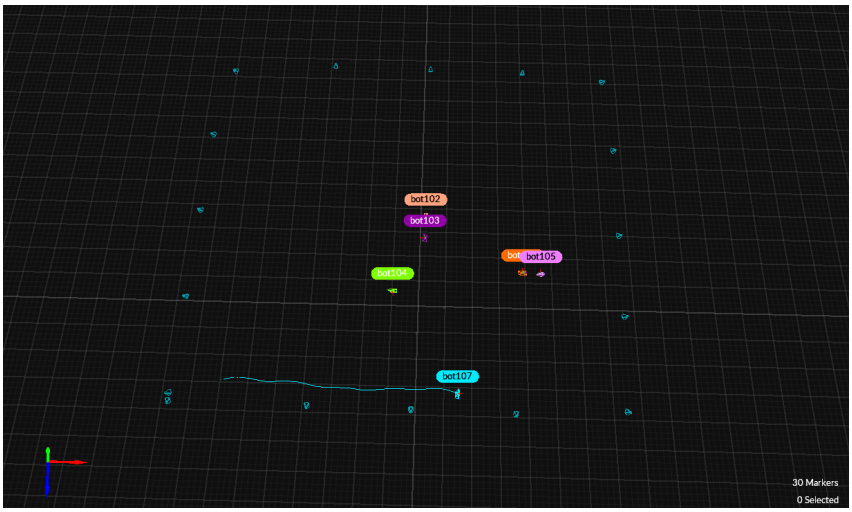

Figure 4: A cropped screenshot from the Motive software showing several of our cars being tracked in the work area. The blue line trailing the bot107 indicates the trajectory of recent movement of the rigid body.

module decides the start and stop time of a car after it has reached a predefined distance of 8 meters (this can be tuned for different experiment setup). The IM instructs a car to move with a particular velocity if it has not reached the finish line. The IM creates TCP sockets to communicate with the cars separately by writing the command output to the respective sockets.

Each car reads and parses the socket in order to identify the command sent by the IM. The car writes the velocity information to its D6 pin and rotation information to its D4 pin. In our experiments we want our cars to move straight. However due to servo motor issues all the cars are not exactly aligned. Due to this reason, we send the direction information to the servo pin. We also implemented p-control for keeping the car direction straight.

We used Arduino C programming language for writing code for each of the cars' control boards and the intersection manager is written in MATLAB programming language. The computer running Matlab is on a physically separate machine from cars and is connected to the same network via Wi-Fi similar to the rest of the cars.

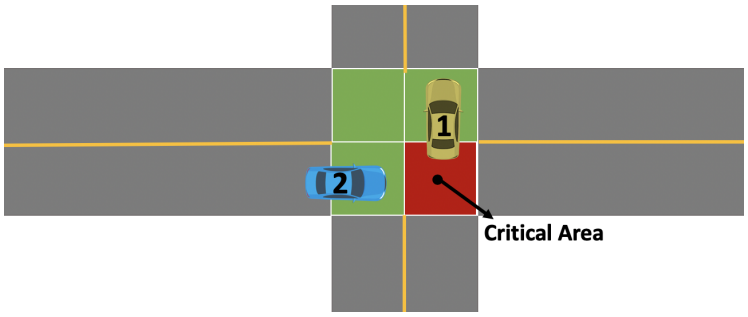

Figure 5: As vehicle 1 exits, vehicle 2 enters.

$$
\begin{aligned}
t_{cross} &= \frac{\text{critical area size}}{VOA1} \\
TOA1 &= TOA2 + t_{cross} \\
t_{out1} &= t_{in2}
\end{aligned}
\tag{1}
$$

The hardware setup used the parameters described in Table 4.1. The tightest schedule results in a scenario as depicted by Figure 2.2. The figure shows that as one vehicle exits the intersection, the next scheduled vehicle enters. The critical area is outlined in red, emphasizing the area where the trajectories of the vehicles may overlap. As vehicle 1 exits at time t_{out1}, vehicle 2 is entering at time t_{in2}. The timing between the cars is related to the time it takes the first vehicle to cross the critical area. The assigned time of arrivals between vehicles in the tightest case is shown in Equation Set 1.

3 Vehicles

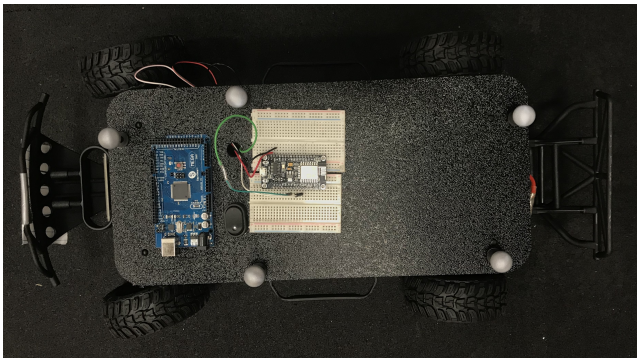

Figure 6: An overview of one the connected autonomous cars. Installed trackers help to detect and track the car using motion capture system.

We built 6 cars from scratch where each is a 1/10 scale model RC car that has two motors one for controlling the speed and one for the steering angle. An ESP8266 Wi-Fi transceiver was used to communicate with the intersection manager and control the steering angle and speed of the vehicle. ESP8266 generates a PWM (Pulse Width Modulation) signal that is fed to a motor driver which delivers the required current to the main DC motor. The speed of the motor is dependent on the duty cycle of the PWM signal. The servo motor for controlling the steering receives a periodic pulse with a width between 1 to 2 milliseconds. The duration of the pulse corresponds to the angle of the servo motor which is between 0 to 180 i.e. 90 degree is equal to going straight. A PID controller was implemented for lane keeping. We installed 5 trackers on each car and saved it as a rigid body in the motion capture system. Figure 6 shows an overview of one of the vehicles. Each vehicle is 0.7 m long and 0.25 m wide and has the max speed of 3.5 m/s.

4 Experiments

Figure 7 shows an overview of our testbed consisting of 6 connected autonomous vehicles. Intersection lines are depicted in yellow.

4.1 Data Collection

Data collection is based on the system input parameters defined by Table 4.1. The data to collect involved correctness testing and throughput. Correctness testing was limited, as there were no observed accidents however there is no counter-example to show that the cyber-physical setup was flawed. This is a primary issue with hardware simulation since a finite testing set is assumed to prove correctness but the infinitely many possibilities cannot be feasibly tested. To calculate throughput, average wait time per test iteration had to be calculated. Table 4.1 shows the average wait time per vehicle over all the 20 simulations. This table was not used to calculate throughput but is used just for reference and brevity of describing average wait time.

4.2 ROS System Setup/Communication Setup

In the project, ROS acted as an adapter to stream the position data from Motive software to the Intersection Manager. Motive provides a setting to stream ”frames” of motion data at a specified

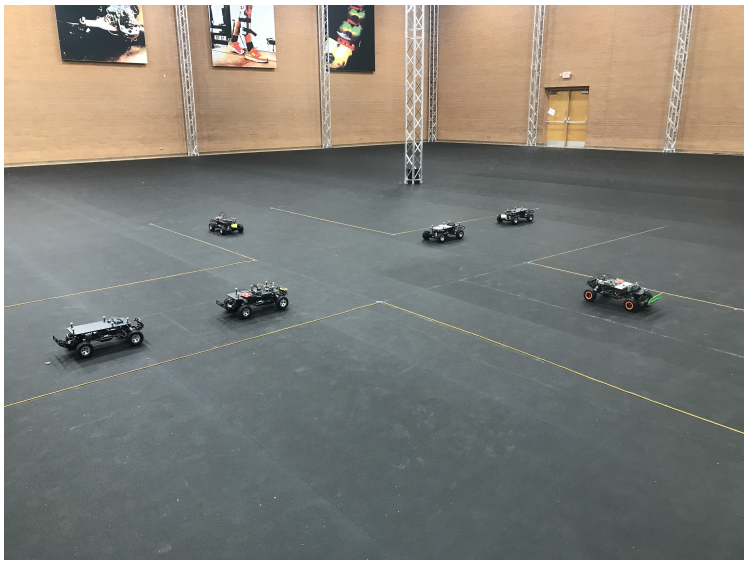

Figure 7: Our testbed consisting of 6 connected autonomous vehicles.

Table 1: This table shows the initial inputs and constraints.

PWM Range	[50,90]
Frequency of Detection	[30,60]Hz
Road Length	8.5m
Road Width	2m
Intersection Length	2m
Vehicle Length	51cm
No. Optitrack Cameras	17
No. Cars	6

rate, which is then broadcast on a chosen port in the local network. Since frames contained motion data about several discrete entities, separate rostopics were published simultaneously for simplicity of reading the position data. The main ROS program actively listened on that port for the rigid body[3] locations and published each individual rigid body's position by their ID set on Motive. The IM setup a subscriber to listen to messages published on *mocap*, using it to obtain the position of each of the vehicles. Using the position data, it was able to calculate a schedule for vehicles using the first-come first-serve (FCFS) approach. It then sent a calculated velocity of arrival and time of

[3]OptiTrack tracked the rigid body locations using passive trackers which were mounted to vehicles in a specific configuration which distinguished them from one another. A rigid body is a set of passive trackers which given a specific configuration identify a body interacting inside the OptiTrack space.

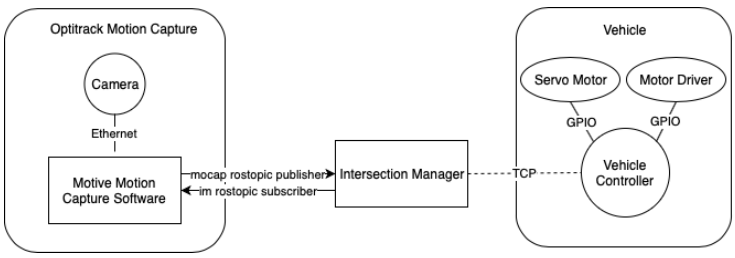

Figure 8: ROS is used to communicate vehicle position detected by the cameras to the IM so it can create a space-time schedule for the vehicles to traverse the intersection.

Table 2: This table shows the average wait time of each car after 20 iterations.

Vehicle No.	Average Wait
1	2.08s
2	2.07s
3	2.01s
4	2.02s
5	1.79s
6	1.46s

arrival to the vehicle and continuously monitored its trajectory through the intersection using TCP messages. See Figure 8 for an overall view of the communication structure.

4.3 Experimental Results

The throughput is calculated per test iteration using Equation 2. The average throughput over all 20 test iterations was 1.875s. This means between vehicles requesting a velocity of arrival and time of arrival from the IM to exiting the intersection the average wait time was 1.875s. Future test iterations should include multiple flow rates, or cars per second per lane which arrive at the intersection to achieve a more objective statistic for throughput. The Table 4.3 shows that in all 20 iterations, there were no trajectory conflicts between cars. This superficially proves correctness, since no counter-example showing failure was realized.

$$throughput = \frac{\sum \text{Wait Times}}{\text{No. Vehicles}} \tag{2}$$

Table 3: This table shows the average wait time of each car after 20 iterations.

Throughput	1.875
Failures	0

5 Discussion and Conclusion

Using a highly distributed, a cyber-physical system can be extremely difficult to debug because errors could propagate through the system from any point. For us, this meant that failures could happen from OptiTrack, the IM, the vehicles, communication, the network, etc. Unfortunately, the majority of development was not spent writing software but ensuring setup was correct. The results obtained show the correctness of the system to be maintained under tight scheduling constraints(one vehicle scheduled right after the other). The throughput obtained was 1.875s for our small sample size. In future iterations, the flow rate of vehicles entering the intersection (cars/lane/second)will be changed in order to have a more comprehensive idea of correctness and performance.

References

[1] M. Khayatian *et al.* RIM: Robust Intersection Management for Connected Autonomous Vehicles. In *2018 IEEE Real-Time Systems Symposium (RTSS)*, pages 35–44. IEEE, 2018.