The Thesis committee for Neda Shahidi Certifies that this is the approved version of the following thesis:

A Delayed Response Policy for Autonomous Intersection Management

APPROVED BY

SUPERVISING COMMITTEE:

Supervisor: _____
Peter Stone

_____
Christine Julien

A Delayed Response Policy for Autonomous Intersection Management

by

Neda Shahidi, B.Sc. in Engineering

THESIS

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

Master of Science in Engineering

THE UNIVERSITY OF TEXAS AT AUSTIN

August 2010

Dedicated to my people who stood up for freedom and democracy while I was writing this thesis.

Acknowledgments

This work has taken place in the Learning Agents Research Group (LARG) at the Artificial Intelligence Laboratory, The University of Texas at Austin. LARG research is supported in part by grants from the National Science Foundation (CNS-0615104 and IIS-0917122), ONR (N00014-09-1-0658), DARPA (FA8650-08-C-7812), and the Federal Highway Administration (DTFH61-07-H-00030).

I also appreciate all the helps of my colleagues in LARG especially Dr. Tsz-Chiu Au.

A Delayed Response Policy for Autonomous Intersection Management

Neda Shahidi, M.S.E.
The University of Texas at Austin, 2010

Supervisor: Peter Stone

The DARPA Urban Challenge in 2007 showed that fully autonomous vehicles, driven by computers without human intervention on public roads, are technologically feasible with current intelligent vehicle technology [6]. Some researchers predict that within 5-20 years there will be autonomous vehicles for sale on the automobile market. Therefore, the time is right to rethink our current transportation infrastructure, which is primarily designed for human drivers, not autonomous vehicles. The *Autonomous Intersection Management* (AIM) project at UT Austin aims to propose a large-scale, real-time framework to be a substitute for current traffic light and stop signs.

Automobiles in modern urban settings spend a lot of time idling at intersections. In 2007, US drivers wasted 4.16 billion hours of their time and 2.81 billion gallons of gas in congestion, costing a total of 87.2 billion dollars nationwide [18]. A big portion of this waste takes place at intersections. The

AIM project is able to utilize the capacity of intersections to minimize time waste and fuel consumption.

The fundamental idea of *Autonomous Intersection management (AIM)* [13] is a reservation system in which cells in space-time will be reserved by the autonomous vehicles based on their trajectories. An intersection manager takes care of the reservation as well as communication with the vehicles. This mechanism tries to maximize the usage of the intersection area. It ensures a collision free intersection as well.

The main question of this project is what intersection control mechanism is appropriate for reducing an autonomous vehicle's waiting time and improving the throughput of the intersection. Previous work proposed the *first-come-first-served* (FCFS) policy in which the reservation requests are served as soon as they are received. The results of simulation show that FCFS outperforms the current traffic systems, traffic light and stop sign, by orders of magnitude.

We, however, observe that FCFS performs suboptimal in certain traffic patterns that are pretty common in urban settings. In this project, first we study the limitations of FCFS, then develop a more efficient policy to alleviate these limitations. The idea that we examined is a systematic policy of granting reservations that have the objective of minimizing the cost of delaying vehicles.

In an attempt to build the system in reality, we used miniature robots called Eco-be. Due to their cost and size, Eco-bes are good candidates for

testing a multi-agent system with a large number of agents. In spite of the fact that the physical challenges of Eco-bes do not perfectly match those of full size autonomous vehicles, they are still useful for demonstration and education purposes as well as for the study of collisions for which experiments with full size vehicles are costly and dangerous.

Table of Contents

List of Tables

Chapter 1

Introduction

Thousands of people around the world are in danger of death or serious injury caused by traffic collisions. In the United States, over 37,000 motor vehicle deaths have been reported during 2008 by the National Highway Traffic Safety Administration (NHTSA) [16]. But "Fatalities are just the tip of the iceberg" [15].For each death, 18 people are hospitalized and 400 are medically attended for injuries, NHTSA reports [15]. They predict that with increases in travel and no improvement over the current safety performance, fatalities and injuries could increase 50% by 2020 [15]. The waste of money and time is no less frustrating. During 2007 US citizens wasted 4.16 billion hours of their time and 2.81 billion gallons of gas in congestion. This cost a total of 87.2 billion dollars nationwide [18].

Intelligent Transportation Systems (ITS) integrate information technology with transportation systems to improve safety and efficiency of transportation [2]. Such systems include efforts to add autonomous features to vehicles to ease the burden of driving for humans as well as development of tools and algorithms for dynamic traffic lights to decrease the waiting time.

A variety of autonomous features have already been added to the ve-

hicles in the market [5]. Fully autonomous vehicles are not yet in the market. However they are at the state of the art in several areas of research. The DARPA Urban Challenge in 2007 gathered teams from around the world to compete in building vehicles capable of driving in traffic, performing complex maneuvers such as merging, parking, passing and negotiating intersections. This program was an outgrowth of two previous DARPA grand challenges in 2004 and 2005 in which vehicles traveled across the desert for over one hundred miles. A team from the University of Texas at Austin collaborated with Austin Robot Technology (ART) to introduce Marvin, a self-driving SUV in 2007's Urban Challenge [6].

Robots can drive safer than humans because they are never sleepy or tired, never get drunk or stressed. They can process a huge amount of information and react quickly. They are more accurate and can follow a preplanned trajectory precisely. However urban traffic elements and protocols have been designed to work well with human drivers not robots. Traffic signs need to be detected everywhere in the streets, among hundreds of background objects, in day light or night and in extremely variant and noisy environments. Although detecting traffic signs using image processing and machine vision techniques is possible, this is not an efficient solution for autonomous drivers. The autonomous drivers still need to sense the world using cameras or radar systems in order to detect obstacles, pedestrians and other vehicles. But signaling them for traffic rules, e.g., stop and go, speed limits, no turn, no park etc. can be done much more efficiently using wireless communication.

On the other hand, the current traffic control systems do not use the full capacity of roads and intersections. In intersections that are the bottlenecks of urban traffic, large amounts of time and money are wasted behind red lights while the capacity of these intersections is much greater than what the traffic lights and stop signs use. In an extreme scenario, a vehicle stopped behind a red-light in empty streets in the middle of the night can avoid wasting time and energy if it can communicate its presence to the intersection. The improvement could be even more if the intersection be allowed to manage crossing vehicles knowing their trajectory. The intersection might allow two or more vehicles to pass simultaneously if their trajectories do not overlap. This is the main idea behind Autonomous Intersection Management.

Autonomous Intersection Management (AIM) has been developed by Dresner and Stone [9] to meet two major goals: to utilize the full capacity of intersections and to propose an intersection management framework customized for autonomous vehicles. AIM is a *multi-agent* system in which each vehicle is controlled by a driver agent and each intersection is managed by an intersection manager agent. The driver agents are responsible for communicating their information to the intersection managers and the intersection managers are in charge of managing vehicles to pass the intersection safely and efficiently. The fundamental idea of AIM is a reservation system which allows a vehicle to reserve cells in space-time according to its trajectory of movement. A vehicle fails to get a reservation if any part of the demanded cells in space-time have already been reserved for another vehicle. Then the vehicle has to

change its trajectory (e.g., reduce its speed) and keep requesting reservations until it succeed. The policy the intersection manager uses to decide which request gets the reservation is First Come, First Serve (FCFS). This policy is simple and fast, but it can not find the optimal solution for different types of intersections.

The contribution of this thesis consists of two separate parts: The first part is a new policy for intersection management that outperforms FCFS (Chapter 4 and 5). The second part (Chapter 6) discusses the attempts toward realization of AIM in the physical world using small size robots. Prior to those two chapters, I have reviewed the AIM concepts and implementation in Chapter 2. The technical details of the AIM simulator and the features and functionality of robots are explained in Chapter 3.

Chapter 2

Autonomous Intersection Management(AIM)

The Autonomous Intersection Management System (AIM) has been designed and simulated by Dresner and Stone [9] with the following features.

- It is a Multi-agent System with two types of agents: driver agents and intersection manager agents. Each vehicle needs to be an agent, working by itself because centralizing all the processes in a single agent is not practical and subject to failure. The interactions between driver agents and intersection manager agents are completely cooperative while interactions of driver agents are a mixture of cooperation and competition.

- Agents communicate. Driver agents communicate with intersection manager agents to receive grants for passing the intersection. Vehicle to vehicle communication is not allowed in Vehicle to Intersection (V2I) version of AIM. The number of messages and the information transmitted in each message are kept low for the sake of the privacy of the vehicles and the scalability of the system. The communication system has also been designed so that any failure in message delivery may prevent vehicles to enter the intersection, but it never causes a crash.

- It maintains the advantages of old systems: the old traffic light and stop sign systems are totally free of deadlock and starvation. Improving efficiency in AIM must not cause deadlock or starvation which means that every vehicle arriving at the intersection has to eventually pass, even if it is more efficient for the whole system to keep it stranded.

- AIM can be extended to a combined system to be able to serve both autonomous and human drivers. In this case, the intersections have to be equipped with both AIM and the traffic lights. This feature will not be discussed in details here because I do not use it in my thesis.

- It gives the highest priority to emergency vehicles and lets them pass with minimum delay.

- Last but not least, AIM is very efficient compared to traffic lights and stop signs: the average delay, or the average of additional traversal caused by the traffic congestion, shows that AIM is more than 10 times as efficient as the simulated traffic light system.

This chapter summarizes Dresner and Stone's fundamental work upon which this thesis builds. In the rest of this chapter, I review the fundamental idea of AIM, the reservation system, in Section 2.1, the vehicle-intersection communications in Section 2.2, the policy of the intersection manager in Section 2.4 and finally, the simulation results by Dresner and Stone in Section 2.5.

2.1 Reservation System: the Fundamental Idea

The basis of decision making in the intersection manager is very simple: the area of the intersection is tiled, and a tile-time reservation table keeps the reservation record of each tile at any time slot. Knowing the trajectory[1]of an approaching vehicle, the intersection manager reserves all tile-time cells on the trajectory of the vehicle (Figure 2.1), if all of them are available. Otherwise the reservation cannot be made (Figure 2.2)

The information exchanged between intersection managers and the driver agents entails a process called passing, which will be discussed in Section 2.2

The way that the intersection manager figures out the tile-time reservations is to run an internal simulation of the virtual vehicle given the information of the requesting vehicle that is available in the proposal. This simulation is the most time consuming and computationally expensive procedure that the intersection manager runs.

2.2 Agent Communication

The main part of the communication consists of the following messages. The complete protocol can be found in Dresner's thesis [9].

[1]Note that the trajectory of a vehicle is different from its path. The path contains only the spacial information, which is the sequence of tiles that the vehicle will pass through. The trajectory also contains temporal information like velocity and acceleration profiles. Knowing the trajectory of a vehicle, one can predict which tiles the vehicle will occupy at any given time.

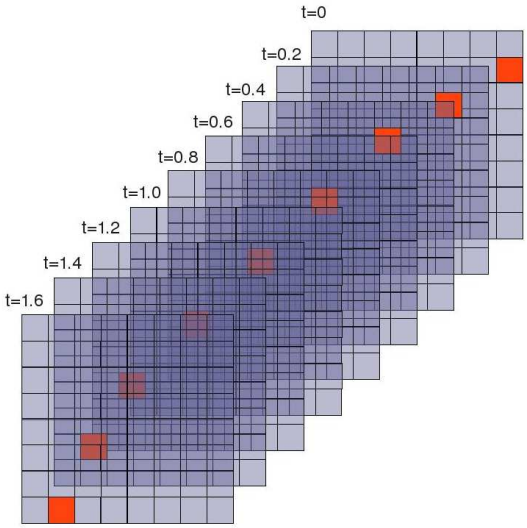

Figure 2.1: (Best viewed in color) The 3-dimensional reservation table with intersection tiles on the first and second dimensions (laid down on the page) and the time as the third dimension (perpendicular to the page). Each grid shows the tiles of intersection area at the time specified by the label above. A red tile is reserved, and a blue one is free. The tile-times reserved for a vehicle approaching from the right road and going to the bottom road are shown.

- **Request**: sent by a driver agent. It consists of the Vehicle Identification Number (VIN), the vehicle's physical properties, maximum and minimum velocity, maximum and minimum (negative) acceleration and one or more proposals. In each proposal, the arrival lane, the departure road, the acceleration profile and the estimations of arrival time and velocity are specified.

- **Confirm**: sent by the intersection manager in reply to request message. The confirm message comes with a reservation ID, after indicating

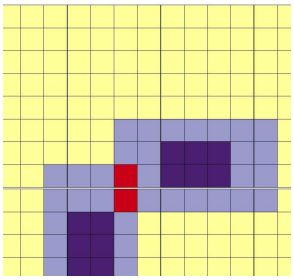

Figure 2.2: (Best viewed in color) A sample conflict that can happen between two requests submitted by two vehicles. The blue areas show the demanded tiles. The area of dark blue tiles will be occupied by the vehicle. To increase safety, the intersection manager pads this area with a buffer by reserving extra tiles around it (shown in light blue). If a second vehicle submits a request that demands any of these tiles (the red ones), the second request has to be rejected.

whether or not this vehicle is allowed to accelerate to maximum speed inside the intersection, and the early and late errors (the time that the vehicle can arrive earlier/later than the claimed arrival time), first and last time that the vehicle may enter the intersection and pass it safely and arrival time and velocity. The arrival time and velocities are the same as the ones in the request message by default.

- **Reject**: sent by the intersection manager in reply to a request message. It contains the reason of rejection (e.g., no available reservation, passing of emergency vehicle etc.) and also the next time the vehicle is allowed to communicate with the intersection.

In addition, the following messages are required to maintain the security and safety of system:

- **Cancel**: sent from a vehicle[2]to the intersection[3]to cancel a previously approved request. A vehicle may send a cancel message if it figures out that it cannot arrive at the specified time or velocity

- **Done**: sent from a vehicle to the intersection to inform it about completion of a reservation.

- **EmergencyStop**: sent from the intersection manager in an emergency situation to prevent previously approved vehicles from entering the intersection.

2.3 Proposals: the Vehicle Side

As mentioned earlier, each request sent by a vehicle may contain one or more proposals. Each proposal includes information about the trajectory of the vehicle. The arrival time and velocity have to be estimated. Each type of vehicle may have its own way of estimating its arrival time and velocity, but the values sent has to be those that the vehicle can commit to reach. The estimation of arrival time can be incorrect if the vehicle is too far from the intersection or is stuck in traffic. If the vehicle cannot arrive at the claimed arrival time or velocity, it has to send a cancellation message. A common case of reservation cancellation happens for vehicles stuck in congestion. These vehicles cannot arrive on time unless all preceding vehicles receive confirm

[3]I might use the term *vehicle* and *driver agent* equivalently when I talk about computation or communication. Same for *intersection* and *intersection manager*.

messages. To alleviate this problem, in the current version of AIM, a vehicle sends a request message only if no other vehicle is in front of it before the intersection.

2.4 Proposal Handling Policy: The Intersection Manager Side

The *Intersection manager's policy* is the logic behind the decision making in intersection manager. Once a request is acceptable, the intersection manager must decide whether to send back a confirmation message or to postpone the confirmation. A good reason to postpone the confirmation is to make a more deliberate decision with the objective of reducing the average delay of all vehicles. The current version of AIM follows the First Come First Served (FCFS) policy. Once the intersection manager receives a request, it examines that request and sends back a confirmation or a rejection message. The basic form of FCFS has to be extended to make exceptions for emergency vehicles [10].

The main motivation of this thesis is to propose a policy that outperforms FCFS in some of the scenarios that FCFS performs sub-optimally. The objective is to reduce the overall cost of delaying the vehicles. If there is no emergency or high priority vehicle (we will talk about prioritizing vehicles later), the cost can simply be an increasing function of the delay. When emergency or high priority vehicles exist, the cost is a function of the delay as well as the priority of the vehicle. We will discuss the details of this idea in

Chapter 4.

2.5 Evaluation of the Idea

Except for a few attempts to realize AIM in the real world (see Chapter 6 of this thesis and [7][8][17]), the ideas of AIM have always been evaluated in the simulations. The following sections discuss the evaluation mechanism and the simulation results.

2.5.1 Evaluation Mechanism

A good metric for the efficiency of the system is the average delay of vehicles.

The delay of each vehicle is the difference between the base traversal time and the real traversal time:

$$delay = T - T_{ideal}$$

The base or ideal traversal time T_{ideal} is the traversal time when there is no other traffic in the system. To calculate the base traversal time, a set of experiments have been done in which the vehicles appear one by one in the field, and their traversal times are recorded. Each time a vehicle passes, a new set of arrival and departure lanes is selected for that vehicle until all combinations of arrival and departure lanes are tried. The base traversal times are recorded and kept in a table. To calculate the average delay for each simulation run, the real traversal times of the vehicles need to be recorded and

subtracted by the corresponding base traversal time.

2.5.2 Simulation Results

Figure 2.3 shows the average delay. A comparison among simulation results of AIM, traffic light and stop sign in this figure shows that AIM outperforms two other mechanisms by an order of magnitude. Figure 2.3 shows that the average delay of vehicles in AIM is less than 10% of the delay for traffic light and stop sign.

To make sense of how this gain is valuable, we compared it to the data from the real world. Since the simulation and the real world settings are different, we are not able to compare the value of the delay. Instead, we can compare the improvement gained by modifications in the current traffic system to the improvement gained by AIM. The average time wasted per traveler in traffic congestion since 1982 (Figures 2.4), as reported by the Texas Transportation Institute and The Texas A&M University System in a 2009 Urban Mobility Report[18], shows that the delay increases due to the increase in the number of vehicles in the streets. The maximum delay was in 2005, 37.4 hours per traveler and the minimum delay was in 1985, 12 hours per traveler. A 10% decrease in the delay is detectable after 2005. Compare this improvement to the 90% improvement obtained by AIM in the simulation. This shows that AIM is worth using. Even though the real world improvement is expected to be less than simulation, we can still predict a great improvement in traffic congestion.

The videos of simulation results of AIM are available online at `http://userweb.cs.utexas.edu/~aim/?p=video`

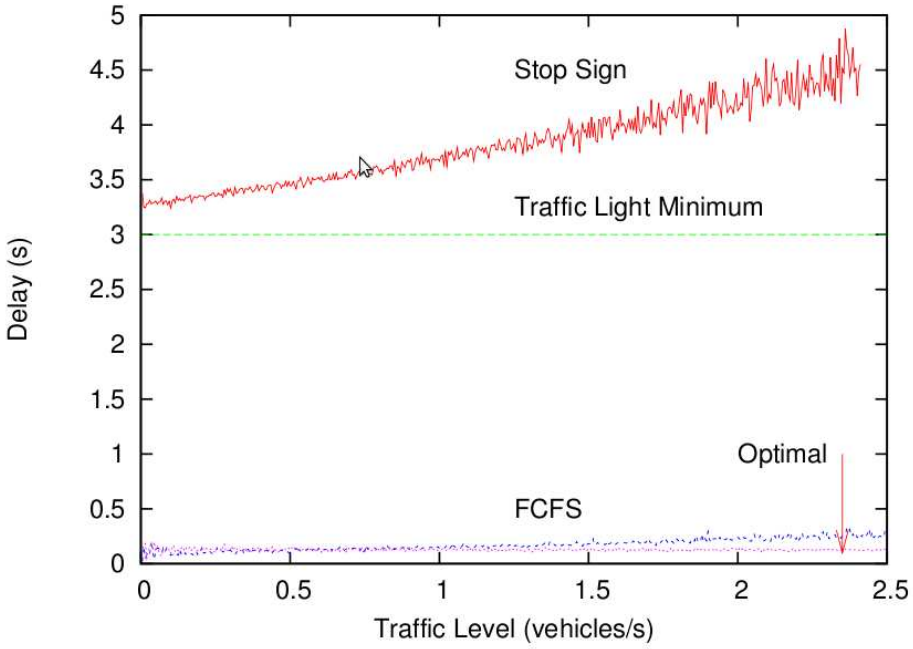

Figure 2.3: comparison of the average delay time among AIM, light and stop sign systems[13]

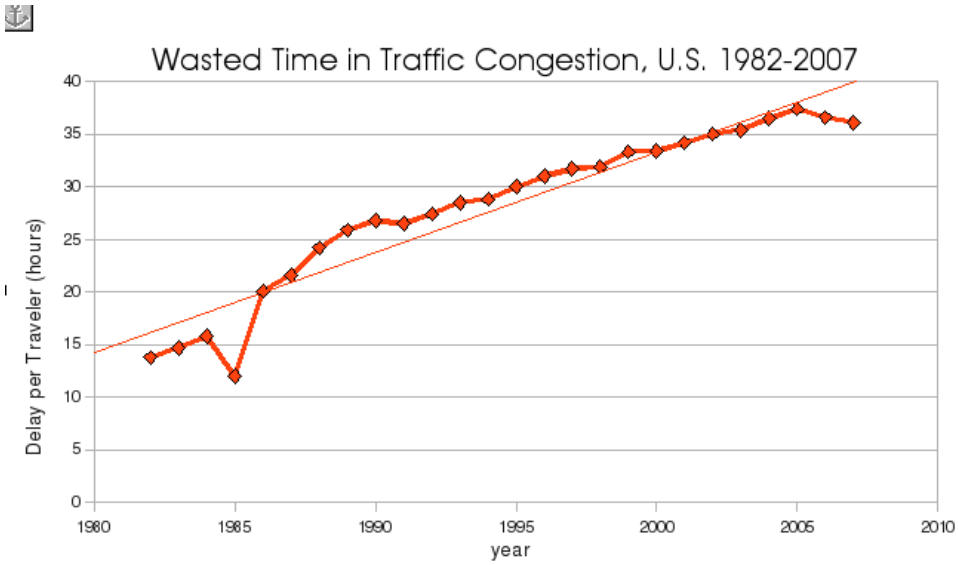

Figure 2.4: The delay per traveler for years 1982 until 2007.

Chapter 3

Substrate System

In this chapter, I explain the substrate system for our work discussed in the next chapters. In Section 3.1, I review the simulator that we have used to implement the ideas discussed in Chapter 4. Then I explain the Eco-be robots used in the physical visualization of AIM explained in Chapter 6.

3.1 AIM Simulator

The AIM simulator has been developed mainly by Kurt Dresner and Tsz-Chiu Au in the Java programming language. The main loop in the AIM simulator executes the following activities:

- spawn new vehicles randomly
- provide sensory information that autonomous drivers need for example the distance of the vehicle to the other vehicles and obstacles.
- update the state of each vehicle based on the intersection manager's decision and the vehicle's action
- remove vehicles outside of the simulated area

In a 4-way single intersection simulation, the vehicles are generated at each lane. A vehicle's destination road, that could be any of other 3 roads, may be determined by one of these three methods:

- the identity selector allows vehicles to go straight to the opposite road only
- the turn based selector has predetermined destination roads for each arrival lane. For example the destination of the vehicles in the most far right lane of each road is always the road in the right. Therefore these vehicles always turn right.
- the random selector selects a destination road randomly.

The destination lane will be selected by the intersection manager based on the closeness to the arrival lane and the availabilities in the reservation table.

The vehicle's size, speed and acceleration profiles are chosen based on the values for real vehicles. The state of the vehicles is specified with the position, velocity, heading, acceleration and steering angle. A unique identification number (VIN) is associated with each vehicle. Driver agents should take care to pilot the vehicles as well as follow all traffic rules and policies. For autonomous drivers this includes communicating with the intersection manager.

The main interface of aim4, the most current version of AIM at the time of doing this project, looks like Figure 3.1.

The graphic interface provides information and tools for debugging purposes. The vehicles are shown in yellow color by default and they turn to white after receiving a confirmation. The vehicles are shown in blue when they are waiting to receive a reply from the intersection. To ensure the safety of vehicles, extra tiles are required to be reserved around each vehicle. These tiles are shown in light blue within the graphic interface.

3.2 Miniature Robots Called Eco-be

Thumb size robots called Eco-be are made by CITIZEN® using wristwatch technology. The first version of these robots was released in March 2006 and was free to be used for research purposes. The second version was released in 2008 with an affordable price. The company hopes that the robots can be used as an affordable platform for education and research.

In this chapter we review the technical details of these robots and their application in Robocup Physical Visualization and Mixed Reality leagues. The eventual goal is to see how suitable the robots are to be used in the physical visualization of AIM.

The first version of Eco-be[1] receives remote control commands from a computer. This version is capable only of receiving data; it sends out no messages. These robots are composed of a tiny metal body on two wheels,

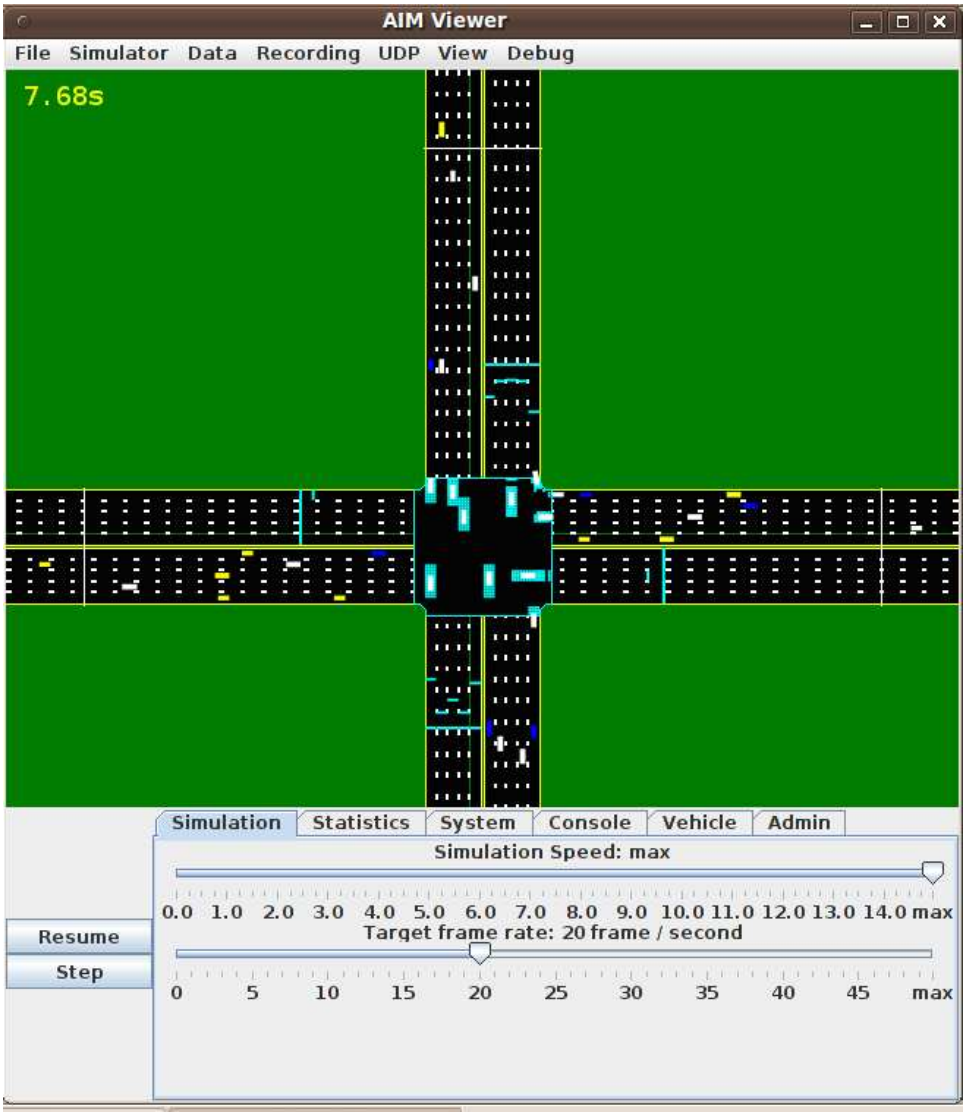

Figure 3.1: The main graphic interface of the AIM simulator

a battery, a control board, two stepper motors to move the wheels and an infrared receiver.

3.2.1 Physical Properties

The body of the first version of Eco-be is 1.8 cm wide and 2.5 cm tall (see Figure 3.2). Two wheels on the sides controlled separately enable it to move forward, backward and turn to the sides. The battery is a miniature one-cell rechargeable 3.7V battery with the capacity of 65 mAh. It slides into the middle of the body and has to be removed for charging.

3.2.2 Motors

Motors are two step motors customized from wristwatch motor units. The manufacturer has provided a fast and a slow speed in each direction. Considering that each of the two motors moves in five speeds (fast forward, slow forward, stop, slow backward and fast backward), 25 movements in total are possible for the robot. For example if the right wheel moves fast forward

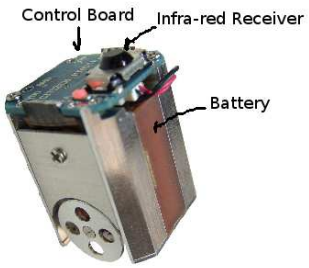

Figure 3.2: First Version of Eco-be

and the left one moves fast backward, the robot starts to turn around itself very quickly. But if the left one moves slow forward instead, the robot makes a left turn. More technical details of the motors are available in appendix A

3.2.3 IR Receiver

The robot receives commands through an infra-red (IR) sensor installed on the controller board on the top. The remote control commands are transmitted in 40MHz pulses with an on-to-off ratio of 50%. Since infra-red noises are everywhere, especially in environments full of electronic devices, a filter is required to filter out any infra-red signals except 40 MHz pulses.

3.2.4 Communication Protocol

Each message transmitted from the controller computer to the robots consists of a header and a body. To encode the data, the notions of *signal* and *space* have been used. Signal is when the transmitter sends 40MHz pulses, and space is when it is silent. The Table 3.2.4 shows the logic for encoding the message header and the logical states using signal and space durations.

If the message is a move command, then the body of the message is a 12-bit string consisting of 5 bits for ID, 3 bits for the left wheel, 3 bits for the right wheel and 1 parity bit. The speed of the motor turning the wheel has been encoded in 3 bits as shown in Table 3.2.4. Although there are three speeds in each direction in the table, only two of them are available in the real robots.

Table 3.1: Logical Encoding of Signal and Space

Logical State	**Signal Duration**	**Space Duration**
HEADER	1228 ms	819 ms
ONE	273 ms	546 ms
ZERO	273 ms	273 ms

Table 3.2: Encoding of Motor Speeds

Acronym	**3-bit Code**	**Description**
FF	001	Forward Fast
FM	010	Forward Medium
FS	011	Forward Slow
ST	100	Stop
BS	101	Backward Slow
BM	110	Backward Medium
BF	111	Backward Fast

The other types of messages are for settings, for example, to set the ID of the robot, to change the delay value (the delay between steps of the motors), to reset the robot, to lock and unlock, to make the robot sleep, etc. The messages carrying the settings commands also consist of 12-bit strings: 5 for ID, 6 for the command and 1 for parity.

3.2.5 Robot Firmware and the Processor

The internal processor of the first version of Eco-be is a PIC micro-controller. It has been installed on the control board, on the top of the robots. It receives data from the IR receiver as interrupt output and sends out move commands to motors. The technical details about the micro-controller are

available in appendix A

The program loaded on the micro-controller takes care of processing the received IR commands, managing the internal variables, and sending the move commands to the wheels. Every time a command is broadcast, each robot receives it and checks the ID field in the message to see if this command belongs to it. As mentioned earlier, the commands are either settings commands or move commands. When settings commands are received, firmware changes only the value of the internal variables. When the move commands are received, the micro-controller modifies the interval variables for the desired velocity and then starts sending move commands to the motors. With each move command, the motor moves one step. Therefore the velocity of motors depends on the delay between commands. The processor keeps sending the move commands to the motors until the robot receives a new command.

3.2.6 Controller Software and IR transmitter

A computer controls all the robots remotely by broadcasting infrared commands. Linux Infrared Controller (LIRC) can be used for coding commands and sending them out as infrared pulses.

We have used a setting in which LIRC sends the commands to one of the pins of a DB9 port. If a DB9 port does not exist on the main board of the computer, a PCMCIA card may be used. The way that LIRC uses DB9 is totally different from the conventional protocol of this port: LIRC controls the voltage of the target pin directly instead of modulating the data using DB9

protocol.

3.2.7 Drawbacks of the First Version

The first version of Eco-bes has major drawbacks that make it inefficient for research and demo purposes:

- The processing power is very limited such that a robot cannot move while it processes received commands. As mentioned earlier, a single not very powerful micro-controller is taking care of parsing the received commands as well as sending the move commands to the motors. To keep the motors continuously moving, the message parsing has to be done in the time gap between consequent move commands. In the old version, this time gap is not long enough for message parsing. Therefore every time a robot is processing a received command, the wheels stop, and after the message parsing is done, the robot starts moving again. Note that the messages are broadcast. Therefore each robot has to process all the transmitted messages to the point that it figures out that this message targets it or not.

  Keeping this introduction in mind, assume that 10 robots are active simultaneously. If we want to send a new command to each robot every 1 sec, 10 messages should be sent in 1 sec which means that all of the robots stop 10 times/sec. To avoid this, one solution is to mute an infrared receiver for a while after receiving a command (this solution is already available in the robot's firmware). This would limit the rate of

sending messages to robots (when the infrared receiver of a robot is off, there is no benefit in sending a message to it). With a robot that stops every time it receives a message, it would be less desirable to have a high message rate. This drawback should be noticed in the move strategies which we expect robots to take: the strategies should be implemented such that the robot moves on a piece-wise path and in each piece, the robot runs a single move command.

- The inputs are noisy. In spite of the fact that the IR receiver of robots filters inputs, the robots are still taking noises as input commands in environments with high amount of IR noise.

- Messages may be lost because IR receivers sometimes filter out input commands. There is no way to detect that a robot has received a command or not because the robot is only a receiver and is not able to send acknowledgment messages.

- Battery's wires are delicate and could be damaged easily if the operators attach and detach them carelessly.

3.2.8 The Second Version

Almost all of these problems have been solved in the second version [4]. This version also has a sandwich control board with a removable part on top and a fixed part on the bottom; this facilitates using customization (see Figure 3.3). However the second version is not substantially different from the first

version in terms of functionality. Since the first version was completely tested and ready to use at the time of this project, we used it for our experiments.

3.2.9 Eco-bes at Robocup

Eco-bes have participated in Robocup since 2007 in Physical Visualization and Mixed-reality leagues. The mixed reality framework consists of real components, the robots and virtual components. In the soccer tournament, the virtual components are the field, the goal and the ball. An overhead camera is capturing the video of the field, and the video will be processed to find the location and orientation of each robot. The 2007 league used the first version, but since 2008 the second version with limited functionality has replaced the first version. In 2008, the Mixed-reality league consisted two competitions besides soccer: in the technical development competition, participant teams were qualified based on their contribution in developing hardware and software for the league, e.g., server and client, vision system, adding sensors to the new version of Eco-bes, etc. The application development competition focused on using Eco-bes for any kind of research or educational purposes. The UTAustinVilla team of the University of Texas at Austin participated in the Mixed-reality league focused mainly on application of Eco-bes in physical visualization of Autonomous Intersection Management system [14]. The details of this project will be discussed in Chapter 6.

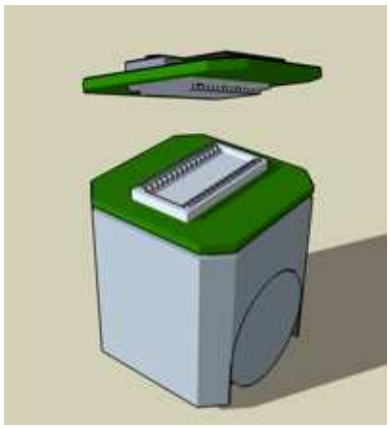

Figure 3.3: The Second Version of Eco-be

Chapter 4

Delayed Response Policy

As mentioned earlier, the intersection manager in AIM uses an FCFS policy, which is the most primitive method of managing requests.

In this chapter, I discuss a new category of intersection manager policies, which I refer to as *delayed response policies*.

After explaining the general idea of delayed response policy in Section 4.1, I will introduce a sub-division of these policies, *batch policies*, in Section 4.2. The remainder of the chapter discusses how to tune batch policy parameters to yield the best performance.

4.1 Delayed Response Policy

I use the term *delayed response* [10] to indicate the opposite of a *reactive* response. A reactive policy, for example FCFS, replies to each request immediately. A delayed response policy may postpone replying in order to receive more requests and make a more deliberate decision by comparing the requests. Note that the final decision is always made by the reservation system based on the availability of the reservation table. However, the policy can affect the final outcome of the intersection manager by changing the order of

proposals or disabling some of the proposals (i.e., rejecting them even if the reservation is available).

Some modification may be required if the AIM policy changes from reactive type to delayed response type. A vehicle may resend its request if it does not receive any reply within a limited timeout. This mechanism is designed to prevent starvation in the real world system, in which the messages may be lost. In a delayed response policy, the intersection manager has to be careful to reply within this timeout. Otherwise, the vehicle assumes that the message is lost and will keep resending the same request. If the delayed response policy is unable to reply before the end of a vehicle's timeout, it can reply through an acknowledgment message, to confirm that it has received the request. In the real world system, this mechanism is required to be used. In simulation, because the messages never get lost, the timeout mechanism of the vehicles may be disabled. The other limitation is the *expiration time* of a proposal, which means that the vehicle can commit to arrive at the specified time and velocity only if it receives a reply before this time.

These criteria will be discussed in depth in next section on *Batch Policy*, a delayed response policy that we have designed and simulated.

4.2 Batch Policy

We designed a delayed response policy that processes a batch of requests at the end of equal-length time intervals called *processing intervals*. The basis for dividing the arrived requests into batches in the proposed method is the

arrival time[1]. Precisely speaking, because the requests may contain several proposals, each with a separate arrival, the batch policy needs to split the requests into proposals, distribute them between batches and process each proposal in its own batch. Note that the intersection manager cannot accept more than one proposal from each request. Therefore, once a proposal is accepted, the other proposals by the same vehicle have to be removed from the batches.

Figure 4.1 summarizes the idea of batch policy. The policy has two processes:

1. Submission. In the submission process, the policy distinguishes between on-time proposals and late proposals (i.e., the proposals that will expire before the next processing time). This part is shown in blue in the figure. Note that late proposals are called "late" because they have been submitted late relative to their arrival. Therefore, for two proposals submitted at the same time, the one for which the arrival time is sooner could be late while the one for which the arrival time is later could be on-time. The Pseudo-code 1 explains the submission process.

2. Processing. In the processing process, the policy selects the target batch and processes the proposals within the batch. This part is shown in red.

[1]Note that by arrival time, I always mean the time that the vehicle arrives at the intersection. To refer to the time that the intersection manager receives the request, I use *submission time.*

Algorithm 1 The pseudo-code of submission process.

```
for each arrived request do
  for proposals in request do
    if t_exp > t_deadline then
      add this proposal to the queue of proposals sorted on arrival time
    else
      this proposal is late
    end if
  end for
  if all proposals of a request are late then
    send the request to FCFS
  end if
end for
```

The expiration time, t_{exp}, of a proposal is the end of the time interval in which the vehicle expects to receive a reply for this proposal. The responsibility of vehicles is to propose arrival times and velocities such that they can commit to arriving on-time at a specified velocity if they receive the reply before the expiration time. They are also required to send the expiration time of each proposal to the intersection manager. In the current version of the AIM simulator, vehicles do not include the expiration time in their messages because we want to keep the messages as short as possible and do not want to include unnecessary information in them. Therefore we chose a constant expiration interval, $EXPIRATION\_INT$, for all the proposals. The expiration time of each proposal is the arrival time subtracted by the $EXPIRATION\_INT$. Later, I will discuss how to choose $EXPIRATION\_INT$.

The deadline, $t_{deadline}$, is specified by the batch policy in each processing cycle. The Algorithm 4.2 shows the updates in each processing cycle.

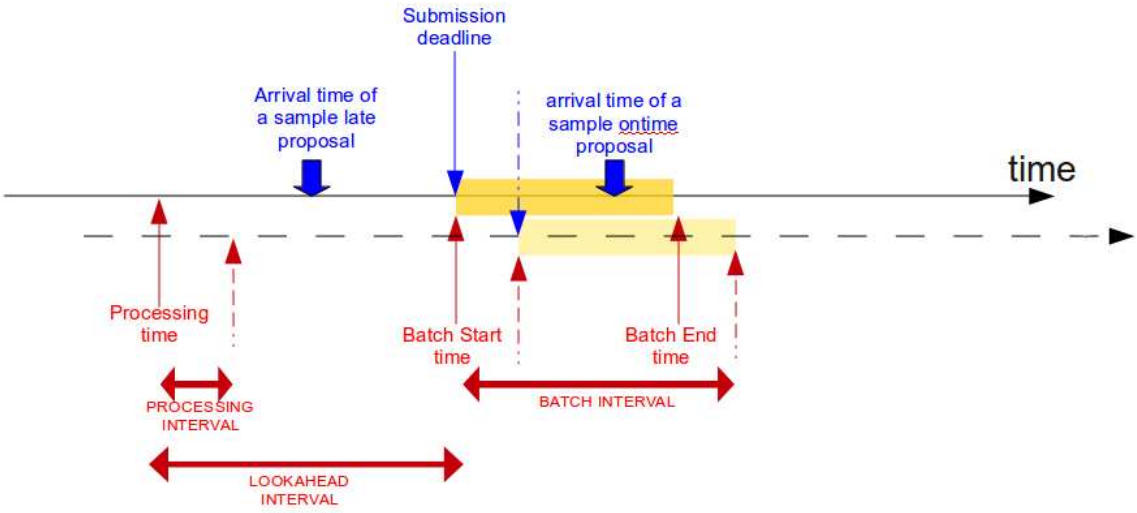

Figure 4.1: (Best viewed in color) The idea behind the Batch Policy. The blue indicators show the submission and the red indicators show the processing. The solid arrows show the current timing and the dashed arrows show the timing of the next processing cycle.

The $PROCESSING\_INT$, $LOOKAHEAD\_INT$ and $BATCH\_INT$ are the length of processing, look-ahead and batch intervals, respectively. I will discuss the algorithms used to process the proposals in batch in Chapter 5. Before that, however, it is necessary to examine how to tune the parameters to achieve the best performance.

4.3 Tuning the Parameters of Batch Policy

The parameters in a batch policy, $LOOKAHEAD\_INT$, $BATCH\_INT$ and the $PROCESSING\_INT$ have to be chosen to reach the goals that will be discussed in the following sections.

Algorithm 2 The pseudo-code for processing process.

```
if time is a multiple of the PROCESSING_INT then
  t_start ← t + LOOKAHEAD_INT
  t_end ← t_start + BATCH_INT
  for each proposal in the queue for which t_arrival ∈ [t_start, t_end] do
    put the proposal in batch
  end for
  process the proposals inside the batch
  t_processing ← t_processing + PROCESSING_INT
  t_deadline ← t_processing + LOOKAHEAD_INT
end if
```

4.3.1 Minimizing the Percentage of Late Proposals

Because the late proposals bypass batch policy and go to FCFS directly (see the submission process), the batch policy cannot improve the overall performance significantly if the percentage of late proposals is high. To reduce this percentage, we must keep the $t_{deadline}$ as early as possible by reducing $LOOKAHEAD\_INT$. The minimum value for the $LOOKAHEAD\_INT$ interval is the time required to process the message in the intersection manager and send a reply to the vehicle, plus the time the vehicle needs to process the reply message.

4.3.2 Minimize the Difference between the Submission Time and the Reply Time

If a vehicle receives a rejection, it submits new proposals until it gets a reservation. If the reply time is too long, the vehicle loses opportunities to get the reservation. In the worst case, the vehicle may end up arriving at the intersection and stop, awaiting a reply. The effects of batch policy parameters

are explained below:

- $PROCESSING\_INT$: Obviously, the reply time will be shorter if the batch is processed in shorter periods.
- $LOOKAHEAD\_INT$: The reply time to a proposal is shorter when processed sooner. Therefore, increasing $LOOKAHEAD\_INT$ decreases the reply time.
- $BATCH\_INT$: If the $BATCH\_INT$ is short, the proposals in the queue must wait to be processed in subsequent batches. Therefore a shorter $BATCH\_INT$ increases the reply time.

4.3.3 Maximize the Number of Proposals within Each Batch

The batch policy can improve performance by letting proposals within the same batch compete against each other. Therefore, the more proposals within the same batch, the higher degree of freedom for decision making. For an arbitrary intersection, we can increase the number of proposals within the batch by increasing the length of $BATCH\_INT$. Increasing $PROCESSING\_INT$ has the same effect. For example if we double $PROCESSING\_INT$, every two batches merge together to make a bigger batch.

4.3.4 Choose the Proposals with Close Arrival Times for Batching

I explained in Chapter 2 that the intersection manager examines each proposal by running an internal simulation of its trajectory in a reservation table.

To examine whether two arbitrary vehicles can get a reservation simultaneously, one could run an internal simulation for two of them in an empty reservation table. This is a computationally expensive procedure. An alternative simple heuristic says that two vehicles do not collide if their paths do not intersect. Also, if the arrival times of their proposals are close enough, they collide if their paths intersect. I will explain in Chapter 5 why it is important to know whether any two proposals collide. The point here is that to be able to use the simple heuristic, it is required that the arrival times of the proposals are close enough. To achieve this, it is crucial to keep the $BATCH\_INT$ as low as possible, because the maximum difference of arrival times of the proposals within a batch is equal to $BATCH\_INT$.

4.3.5 A Heuristic to Tune the Parameters

To satisfy the objectives mentioned in sections 4.3.1 to 4.3.4, we followed the steps below to tune the parameters:

1. $LOOKAHEAD\_INT$: The percentage of late requests increases monotonically with the length of $LOOKAHEAD\_INT$ (Figure 4.2). Therefore, I chose $LOOKAHEAD\_TIME = 0.02s$, which is the time required

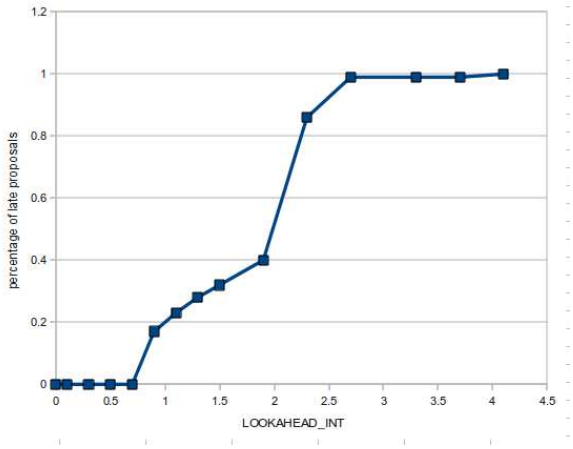

Figure 4.2: The percentage of late proposal versus $LOOKAHEAD\_INT$. The data is for a 3*3 intersection with traffic level of 0.3 vehicles/seconds/lane. Simulation time is 300 seconds and each data point averaged on 30 trials.

for computation and communication. This time is very small in comparison to the system's time scale and the percentage of late proposal is zero for this value.

2. $PROCESSING\_INT$: Choosing the $PROCESSING\_INT$ as short as possible adds the minimum delay to FCFS. Figures 4.3 shows the percentage of the average extra delay versus $PROCESSING\_INT$. To reduce the extra delay, we might want to choose $PROCESSING\_INT$ as low as possible. The problem with small $PROCESSING\_INT$ is that there are fewer proposals within a batch. Figure 4.4 shows how the distribution of the number of proposals in the same batch changes with $PROCESSING\_INT$ ($BATCH\_INT = 10s$). Given these two analyses, we decided to choose $PROCESSING\_INT = 0.12$,

for which the extra delay is less than 30% (see Figure 4.3) and the average number of batched proposals is 6 (see Figure 4.4). The value of $PROCESSING\_INT$ also affects the average number of requests sent by each vehicle. Figure 4.5 shows that the average number of requests drops significantly when $PROCESSING\_INT$ increases. For the chosen value, the average number of requests per vehicle is almost half of FCFS. This is a side advantage of batch policy.

3. $EXPIRATION\_INT$, as shown in Figure 4.3, a large number of vehicles cannot get the reservation for $PROCESSING\_INT$ bigger than 0.5, which means that the intersection manager passes the expiration time of a large number of proposals. Almost all of these proposals are submitted by the vehicles right behind the intersection for which $t_{arrival} - t_{submission}$ is roughly 2.8s for the majority of the proposals. In the worst case, a proposal was submitted immediately after the previous processing time. If $PROCESSING\_INT$ is 0.5, then the next processing time is 0.5 seconds after the submission of this proposal. The deadline is $LOOKAHEAD\_INT$=0.02 seconds after the processing time which is 0.52 seconds after the submission of the proposal. The expiration time must be before the deadline. We have assumed that the expiration time is $EXPIRATION\_INT$ seconds before the arrival time for all the proposals. Therefore, I chose $EXPIRATION\_INT$ to be 2.8-0.52=2.28 as the best estimation of the expiration intervals of the proposals.

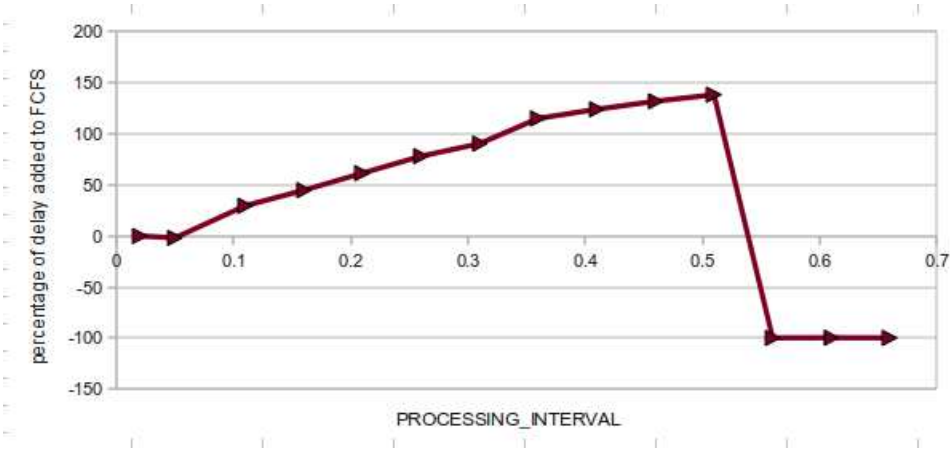

Figure 4.3: The delay added to the system by batch policy versus. $PROCESSING\_INT$. This delay is inevitable because batch policy replies later than FCFS. However, we can control this delay by choosing $PROCESSING\_INT$ appropriately. The simulation is on a 3*3 intersection with traffic level of 0.3 vehicles/seconds/lane. The simulation runs for 300 seconds. Each data point averaged over 30 trials. For $PROCESSING\_INT$ more than the 0.5, the confirmation message will be sent after the expiration time of the most of the proposals. The vehicles that are sending these proposals has to cancel the reservation every time. Therefore, they will never get a reservation. In fact, the very low average delay in the figure comes from a very low percentage of vehicles that can get a reservation and pass the intersection. Later, I will explain how to choose $EXPIRATION\_INT$ based on this phenomenon.

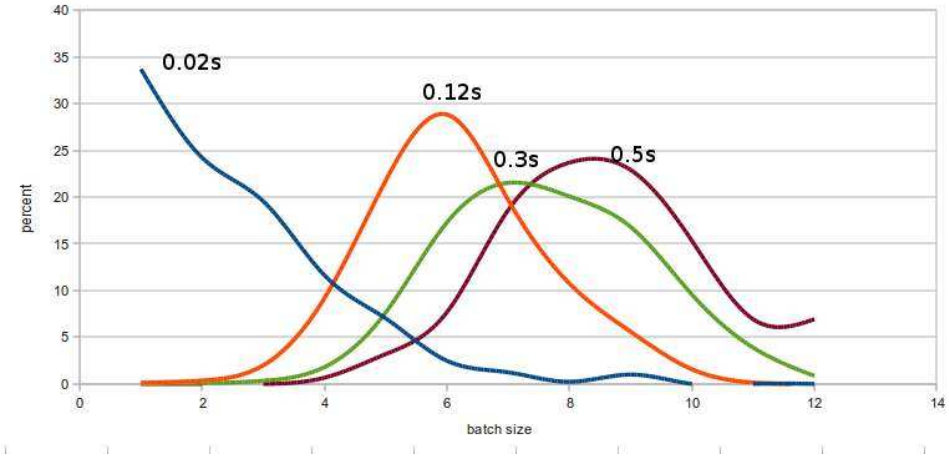

Figure 4.4: The distribution of batch size for $PROCESSING\_INT = 0.02$, 0.12, 0.3 and 0.5.

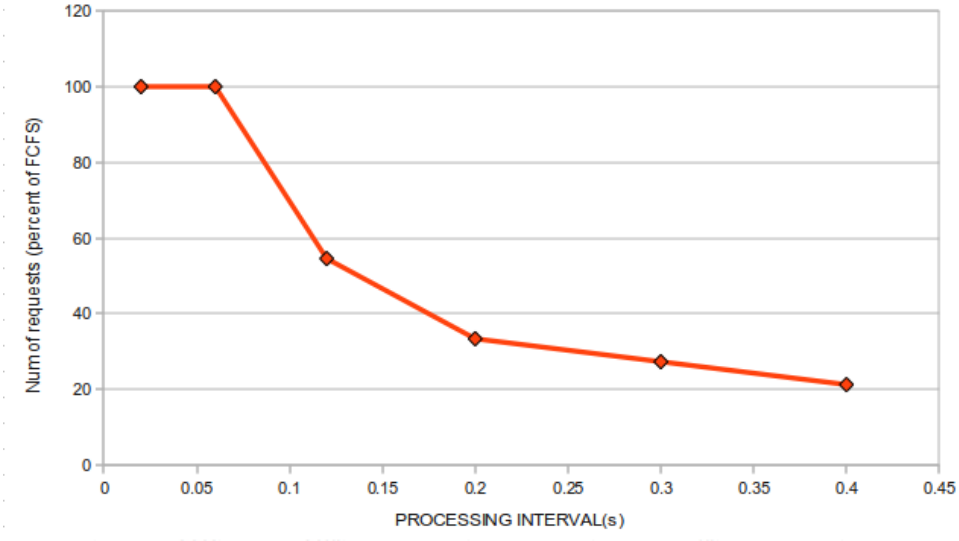

Figure 4.5: The average number of requests sent by each vehicle versus $PROCESSING\_INT$.

4. $BATCH\_INT$: A bigger $BATCH\_INT$ is better because we can have more proposals within a batch. However, as mentioned above, it is important to have proposals with close arrival times within a batch. Therefore, we reduced the $BATCH\_INT$ until the number of proposals within a batch starts to drop significantly. We chose $BATCH\_INT = 3s$.

By tuning the parameters of the batch policy, we got the best performance while minimizing the side effects of this policy. However, this policy cannot improve the performance before using a good strategy of managing proposals within a batch. In the next chapter I discuss some of these strategies that we have designed and tested.

Chapter 5

Strategies for Processing Proposals in Batch Policy

In this chapter different strategies for processing proposals within a batch are investigated. The final decision on accepting or rejecting proposals is always made by the reservation system. The policy can affect this decision, for example, by changing the order of proposals. When one proposal is accepted, the availability of the reservation table changes and the other proposals that conflict with the accepted proposal cannot get the reservation. Therefore, it is important to examine a more eligible proposal, for example a proposal submitted by a vehicle that have been waiting longer than other vehicles, first. The policy also can affect the decision by disabling proposals (i.e., they will not be examined by the reservation system). In this chapter two strategies are discussed: cost-based reordering and clique-based reordering, in Sections 5.3 and 5.4, respectively. Both strategies are based on a cost function that has been explained in Section 5.2. Calculation of cost is based on predicted delay of each vehicle, which will be discussed in Section 5.1.

5.1 Delay Prediction

To be able to make decisions for proposals within a batch, we use methods that use predicted delay as a basis for prioritizing the proposals. In this section, the delay prediction is explained.

The delay of a vehicle is the difference between its real traversal time and ideal traversal time. To measure the real traversal time, the vehicle has to pass the intersection. However, we are interested in predicting this delay before the vehicle arrives at the intersection.

We calculate the predicted delay in three parts:

1. $delay_{back}$: the delay of a vehicle when it is behind another vehicle and does not send any message. Remember that in the current version a vehicle can send messages only if there is no other vehicle between itself and the intersection.

2. $delay_{front}$: the delay of a vehicle after it submits its first request until it arrives at the intersection.

3. $delay_{passing}$: the delay of a vehicle when it is in the intersection and after passing it.

Because the first part, $delay_{back}$, uses the $delay_{front}$ values of other vehicles, the $delay_{front}$ is explained first.

The $delay_{front}$ of a vehicle is zero if the first proposal submitted by this vehicle is accepted. If the first proposal is rejected there will be a time waste

to submit the next proposals. In the next proposals, the arrival time could be later than the first proposal due to reduction in speed. Therefore, there could be a time waste before the vehicle arrives at the intersection. These two parts are captured by the simple following formula:

$$delay_{front}(p_i) = t_{arrival}(p_1) - t_{arrival}(p_i)$$

in which p_i is the i^{th} proposal by this vehicle. The $delay_{back}$ is non-zero when a vehicle has to slow down or stop behind another vehicle, in what is called a congestion. Based on the information in submitted requests, there is no way to figure out what the previous state of this vehicle was; did it just arrive or it has been waiting for a while? Therefore we use a heuristic to distinguish the vehicles in congestion: the vehicles in congestion always submit their requests from a fixed position before the intersection, which we call the stop position. This position is where the vehicles stop, waiting to get a reservation. In the proposals submitted from this position, the difference between submission time and arrival time is more or less constant (In the previous chapter we mentioned that it is almost 2.8s for the majority of the vehicles. This value can be changed by changing the stop position.). Therefore we can distinguish this proposal by comparing the difference between arrival time and submission time to a threshold. If this difference exceeds the threshold, the vehicle has not been in congestion and $delay_{back}$ is zero.

For the vehicles in congestion, the $delay_{back}$ can be estimated from the predicted delay of vehicles that had been in the same lane. I use notation v_i for the i^{th} vehicle in the lane. Once v_n arrives at the front of the lane, the

$delay_{back}$ will be calculated for this vehicle using $delay_{front}(v_1)$, $delay_{front}(v_2)$, ... and $delay_{front}(v_{n-1})$. The delay of v_n, while v_i has been in the front row, is

$$delay_{back,i}(v_n) = \max\{delay_{front}(v_i) - \frac{n-i}{D}, 0\}$$

in which D is the traffic level (vehicles/seconds) of this lane. Therefore, $\frac{n-i}{D}$ is the expected time for arriving (n-i) vehicles (i.e., v_{i+1} ... v_n). According to this formula, if the $delay_{front}$ of v_i is larger than the time required for arriving v_n then the delay of v_n behind v_i is non-zero which is reasonable. In practice the delay of v_n behind v_i is larger than this value because of the time required for acceleration and deceleration of vehicles in congestion. Therefore we modify the formula as below

$$delay_{back,i}(v_n) = \max\{delay_{front}(v_i) - c \cdot \frac{n-i}{D}, 0\}$$

in which c is a coefficient less than 1. The $delay_{back}$ of v_n is calculated by the sum of $delay_{back,i}$:

$$delay_{back}(v_n) = \sum_{i=1}^{n-1} \max\{delay_{front}(v_i) - c \cdot \frac{n-i}{D}, 0\}$$

The third part of the delay is $delay_{passing}$ which may be observed due to reduction in speed of vehicles before passing the intersection. Since in the current version of AIM, acceleration to the ideal speed is allowed inside the intersection, we ignore this delay.

Figure 5.1 shows the predicted delay versus the real delay. This figure shows that for most of the data points, especially for those with high delay values, the predicted delay is close to the real delay. This is important, because we want our policy to work efficient, especially for high delay vehicles. The

correlation coefficient for the linear regression is also high.

A good delay prediction is very important for us because we use the predicted delay values to reduce the overall average delay. In the next section, we discuss strategies to manage the proposals in a batch using the estimation of delay.

5.2 Cost Function

We define a cost value that increases with the predicted delay for each proposal. The objective is to minimize the overall cost of the vehicles. This idea has already been used in computer task scheduling [19]. We use a polynomial cost function,

$$f(delay) = a * (delay)^b + \epsilon$$

in which a and b are coefficients specific to the vehicles, and ϵ is a very small value (0.2s) just to prevent division by zero in calculations. To prevent starvation, the cost function should grow faster than the linear function. Therefore b has to be greater than 1. To prioritize vehicles we choose a and b such that vehicles with the same delay can have different costs. For example the cost of delaying a public transport vehicle or a gas or food truck must be larger than a small size vehicle. The cost of delaying an emergency vehicle must be extremely high.

5.3 Cost-based Reordering Strategy

A simple way of reordering proposals within a batch is to sort them by their cost values and let the ones with higher costs have priority.

A drawback of this simple algorithm is that it does not maximize the number of proposals that are accepted. Assume that the proposal with the highest cost gets the reservation. A large number of other proposals in the batch may be rejected because of this single proposal. If the costs of proposals are more or less the same, then this is not a good decision because the sum of the costs of rejected proposals can be much higher than the single accepted proposals. We designed the *Clique-based reordering strategy* to alleviate this drawback.

5.4 Clique Based Reordering Strategy

The concept behind the clique based reordering is to group the vehicles that can pass the intersection simultaneously and give the priority to the group with the highest cost. The objective is to give priority to the group with the maximum number of proposals with the highest total cost. This strategy runs the steps below:

- Create a graph called clearance graph in which each node represents a proposal
- Weight each node with the cost of its proposal

- Find the maximal clique, i.e., the clique with the maximum weight
- To build the reordered list, add the nodes in the clique first and the rest of the nodes (sorted by their weights) next.

The following subsections explain how to create the clearance graph and find the maximal clique.

5.4.1 Clearance Graph

The clearance graph represents the collision pattern of proposals. Each graph node represents a proposal. An edge between two nodes exists if the correspondent proposals are compatible, which means they can be accepted simultaneously. To check this, the intersection manager needs to simulate trajectories using an internal virtual vehicle, which is a time consuming and computationally expensive simulation. Therefore we only check if the paths of two trajectories intersect. As we have discussed in Section 4.3.4, if the arrival times of the proposals are close, there is a high chance that two proposals are not compatible when their paths intersect. But two vehicles are always compatible if their paths do not intersect.

5.4.2 Maximal Clique

Finding the maximal clique – which is the clique with the maximum weight – is an NP-hard problem. Many greedy algorithms have been proposed to solve maximal clique problems. The steps of a simple greedy algorithm are shown in Pseudo-code 5.4.2:

Algorithm 3 A greedy algorithm to find the maximal clique.

sort all of the clearance graph nodes based on their weights
pick the first node from the sorted list as the first node of the maximal clique
scan the sorted list, and if a nodes has edges with all of the nodes in the maximal clique, add it to the clique

The drawback of this algorithm is that if the weight of the first node is not significantly bigger than the weight of the others, the answer is suboptimal. Assume the case in Figure 5.2. For simplicity, the nodes are numbered in the order of their weights. The algorithm above finds the clique $1, 2, 5$ for which the weight is 5, while the real maximal clique is $2, 3, 4, 5$ for which the weight is 6.

To improve the algorithm, we repeat the last two steps of the above pseudo-code for the k next heaviest nodes that are not in the detected cliques to find k cliques. Then we compare cliques to find the maximal clique. See the Pseudo-code 5.4.2.

Algorithm 4 Modified version of the greedy algorithm in Pseudo-code 5.2.

sort all of the clearance graph nodes based on their weights.
for k times **do**
  pick the heaviest node which is not labeled, add it to the k^{th} clique. Label this node
  scan the sorted list, if a node can be added to the k^{th} clique, add it to the k^{th} clique and label the node
  compare all k cliques to find the one with the maximum weight
end for

Because the run time of finding each clique is $O(n)$, the run time of this algorithm is $O(kn)$. if we choose $k << n$ for big ns, then the run time is

linear. The solution found by this algorithm can be very close to optimal if k is large enough.

5.5 Simulation Results

We simulated AIM with cost-based and clique-based strategies and compared the average delay. In both batch policies, the cost function is $cost(delay) = delay^2$ for all the vehicles unless we mention that for some vehicles it is different.

In the clique-based strategy the parameter k is 2.

Figure 5.3 shows the average delay for FCFS, cost based reordering and clique-based reordering. Figure 5.4 shows the percent improvement gained by reordering strategies. These two figures show that the improvement is the largest for medium levels of traffic. It also shows that there is no big difference between cost-based and clique based strategies which can be due to the small size of batches.

We also compared the strategies for an uneven traffic pattern in which there is a high degree of traffic in the horizontal road and a low degree of traffic in the vertical road. These kinds of traffic patterns are very common in urban settings. However, they have never been tested with AIM. In this traffic pattern, all the vehicles in the high traffic roads go straight but in the low traffic roads, the vehicles may choose to go straight or turn. This is close to a realistic pattern because it keeps the traffic level high in the out-band of high

traffic roads and keeps it low in the out-band of low traffic roads. Figures 5.5 and 5.6 show that when the difference in high and low traffic levels is higher, both reordering strategies are much better than FCFS.

The study of the effect of number of lanes per road is shown in Figure 5.7. Roughly speaking, we can have high improvement for medium size intersections. For small intersections– e.g., 1 lane per road– we can not get good improvement because the batch size is very small.

5.6 Discussion

Comparing the delay for FCFS and variations of batch policy show that batch policy can improve the delay up to 35%. The maximum improvement occurs around traffic level of 0.3. The improvements drop for higher traffic levels because of the congestion. In the current version of AIM, the vehicles cannot submit requests while there are other vehicles between them and the intersection. Therefore the policy cannot detect lanes with more congestion and help them to be evacuated more quickly. We predict that if we ease the constraint by allowing the vehicles in the back to submit requests and get a reservation if all the vehicles in front of them already have one, then a good deal of improvement can be achieved because all the vehicles coming from the same lane potentially can be accepted simultaneously. Therefore the policy can manage to group the vehicles in the same lane and let them pass the intersection together.

The batch policy also makes a significant improvement for big inter-

sections. The simulation results show that the improvement is not significant for very small intersections, but for bigger ones it could be as high as 25% for 5*5 intersections. The reason is that the number of proposals within a batch is higher for bigger intersections. Therefore batch policy has more degree of freedom to make the decision.

A side advantage of batch is that it reduces the number of messages sent from the vehicles to intersections and Vice Versa. In simulation, it is not very important. However in reality, lower message traffic level can improve the communication and lower the overhead of message processing in intersection managers.

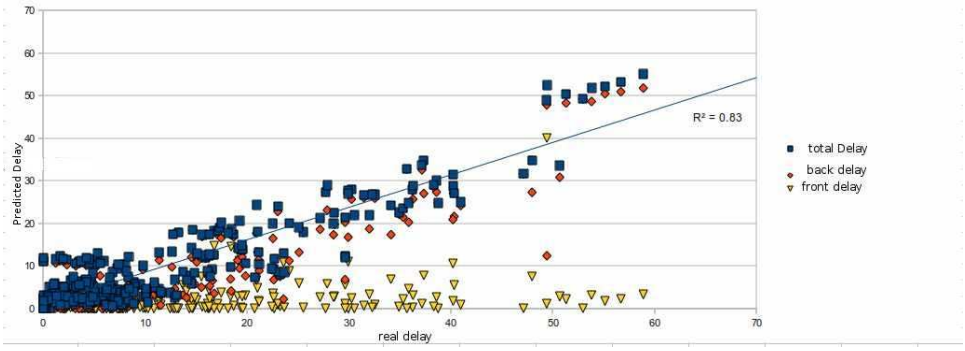

Figure 5.1: The predicted delay versus the real delay. Each point is a single vehicle. The results belong to a simulation for a 3*3 intersection with a traffic level of 0.7 vehicles/seconds/lane. The yellow triangles, red dots and blue squares are $delay_{front}$, $delay_{back}$ and total predicted delay respectively. The line shows the linear regression for predicted delay. The correlation coefficient for this regression is shown at top right.

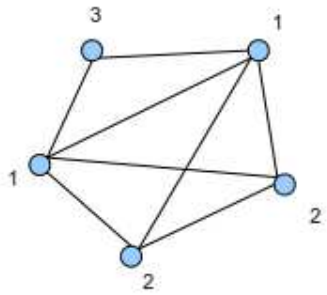

Figure 5.2: A simple case that the greedy algorithm for finding the maximal clique finds the suboptimal answer.

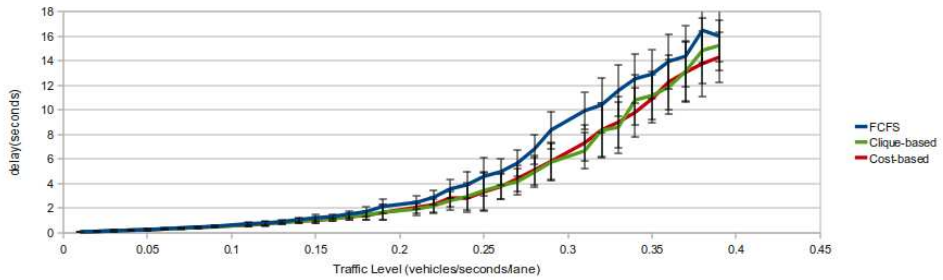

Figure 5.3: Average delay for a 3*3 intersection. Simulation time is 300s. Each data point is average over 30 trials.

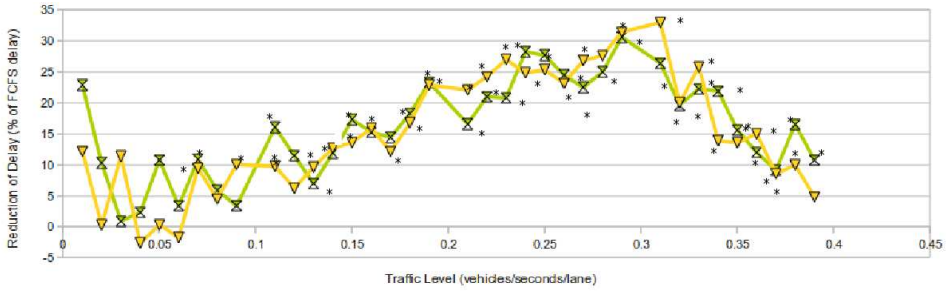

Figure 5.4: Percent improvement gained by reordering strategies. The improvement for cost-based strategy is shown in red and for clique-based strategy in blue. The statistically significant improvements are marked with *.

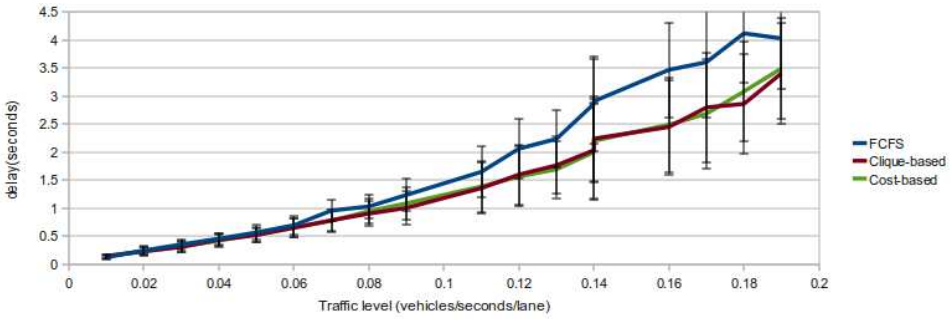

Figure 5.5: average delay for a $3 * 3$ intersection. The traffic level of the high traffic road is 0.4 and the traffic level of the low traffic road is on the x axis. Simulation time is 300s and each data point is an average of 30 trails.

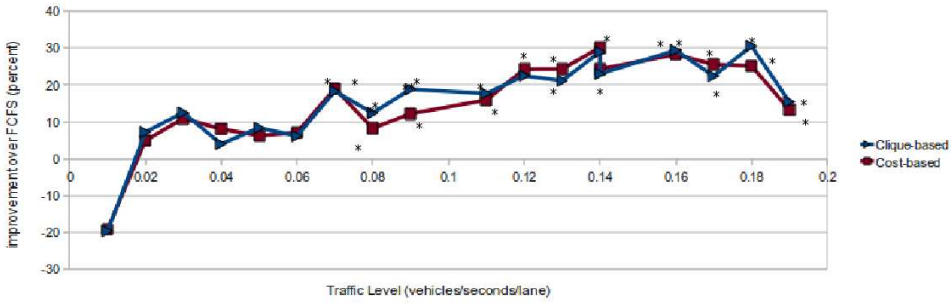

Figure 5.6: Percent improvement gained by reordering strategies for uneven traffic patterns. The improvement for cost-based strategy is shown in green and for clique-based strategy in yellow. The statistically significant improvements are marked with *.

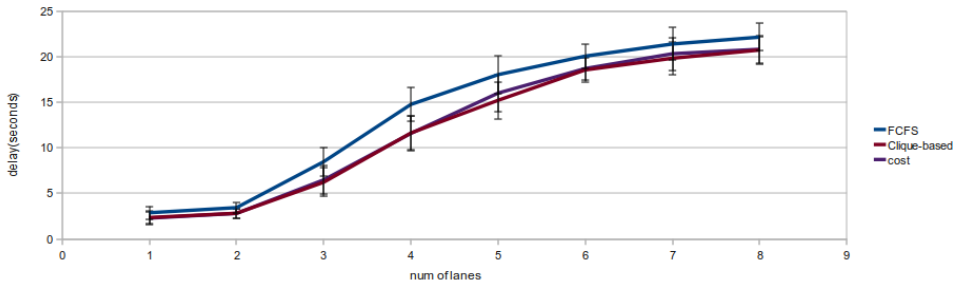

Figure 5.7: Average delay for a 3*3 intersection with a traffic level of 0.3. Simulation time is 300s and each point is an average of 30 trails.

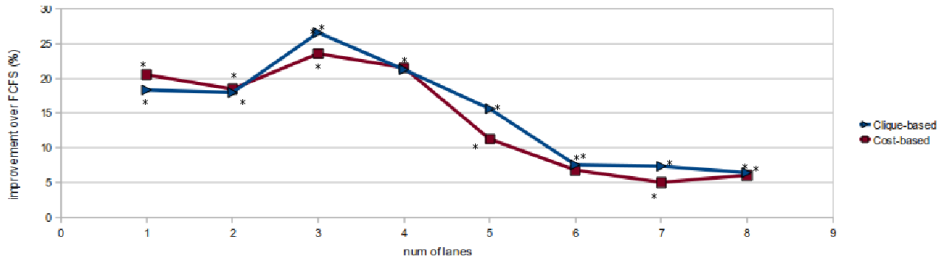

Figure 5.8: Percent improvement gained by reordering strategies. The statistically significant improvements are marked with *.

Chapter 6

Physical Visualization of AIM using Eco-be

Since AIM has shown brilliant results in simulations, it is worth testing in a real-world situation. However testing it using many full size autonomous vehicles is extremely costly and may be dangerous. Although the system has been tested with a single autonomous vehicle in a mixed-reality framework [8], testing on inexpensive robots in lab settings is more reasonable. To use Eco-be for this purpose, the following considerations have to be taken into account:

- Eco-bes are small in size. The whole system can be implemented on a table size field and one can easily pack it up and move it anywhere for demonstration in classrooms or conferences.
- Eco-bes can produce the vehicle movements that are required in AIM simulators. The first version has only two levels of velocity, therefore cannot turn with any angle. But the velocity of the new version has a sufficient resolution to simulate the speed changes of a vehicle and the varieties of turning angles required for turning vehicles in a multi-lane intersection.
- None of the Eco-be versions are autonomous because no version can sense the environment or make decision by its own. It relies on a centralized

computer instead. In practice, these robots are only the place holders of virtual vehicles.

- Eco-bes are inexpensive, so a multi-agent system that uses them is affordable.
- Eco-bes implement real world challenges into the system. However the challenges they face are different from the challenges of full-size vehicles per se.

Due to limitations of the first version, we implemented a simplified version of AIM (Section 6.1). Also, there is only one vehicle in the system for which a robot exists in the real-world. The other vehicles are virtual.

6.1 Simplified AIM Simulator

The system has been implemented using a very simplified version of AIM with the constraints below. All of these constraints have been applied because the first version of Eco-bes are hard to control.

- Each road has two lanes with a single vehicle in each lane. The vehicles do not change their lanes. In fact, the paths of the vehicles never intersect unless they are inside the intersection.
- Vehicles always choose to go straight and no turn is allowed.
- When the vehicles arrive at the end of the road, they stop and start moving backward. In fact, they do lap running in their own lane.

- Vehicles can only switch from the fast speed to the slow speed or stop right before the intersection if no reservation has been granted for them. Also, they do not have to provide their acceleration profile to the intersection manager because they can accelerate or decelerate in a very short time compared to the time scale of the experiment.

6.2 Field Settings

The field is (approximately) 20inches * 30inches in size on a regular table. Four infra-red lamps are hung over (approximately) 20" on the top of the fields. An overhead camera (a simple webcam) captures the video of the field from the top center.

6.3 Vision System

The video of the overhead camera needs to be processed to find the position, ID and the orientation of robots. Each robot has a marker on the top for easy detection (Figure 6.1). The red disk helps to find the position of the robot while the black and white codes that are translated to binary codes help to find the ID and the orientation. The idea of a marker to detect Eco-bes has already been used in Robocup 2007's physical visualization league.

6.3.1 Position Detection and Tracking

To reduce the computational overhead of image processing, the detection and tracking modes have been separated. In the detection mode, the

Figure 6.1: A sample marker for easy detection and identification of robots.

whole image will be scanned for red disks using pattern matching, which is a computationally expensive method. After the red disks have been detected, a lower cost pattern matching that only looks in the neighborhood area of red disks will be used to track them in the next frames. In our system, the tracking is almost 25 times faster than detection. (500 ms for detection and 20 ms for tracking on a 2.4Ghz machine, with 1G of RAM).

6.3.2 ID and Orientation Detection

The Algorithm 6.3.2 finds the ID and the orientation of the robot. All the robot IDs have been recorded in an ID book with the binary codes of the markers.

This algorithm can detect IDs reliably except for those IDs that are very similar. Therefore we used the IDs that are very different and can be detected reliably in our experiments.

Algorithm 5 The algorithm to find the ID and the orientation of a robot.

string ← Pick N pixels on a circle around the red disk. code their brightness with 0 (if bright), 1 (if dark) and *don't care* (if between)
for each $candidateID$ in IDBook **do**
  $dupCandidateID$ ← duplicate the binary code and concatenate the duplicated to the original one.(e.g. 11000011 → 1100001111000011)
  for $shift = 0$ to N **do**
    calculate the hamming distance between *string* and the $dupCandidateID(shift$ to $shift + N)$. *Don't care*s are 0.5.
  end for
  $id \leftarrow argMin(hammingDistance)$
  $orientation \leftarrow shift * argMin(hammingDistance)/(2 * N * \pi)$
end for

6.4 Motion Control for Lap Running

The robots are supposed to do lap running. They go straight forward until the end of the pass then come back on the same lane using backward movement. As mentioned earlier, we decided to choose this kind of path because we wanted to have a simple motion control system. To implement a simple motion control, we designed a *tiny turn* command: the robot turns to the right or left for *tinyTurnRight* or *tinyTurnLeft* commands respectively and then continue to go straight after a fraction of a second. Using this command we designed a multi-region motion controller (see Figure 6.2). The multi-region controllers are a class of controllers that have a separate policy for each sub-space of input space. The policy may generate a constant output (as in our controller), a PID (Proportional, Integrative, Derivative) output or a more sophisticated controller output. The controller in Figure 6.2 turns the robot toward the track every time that the deviation of the robot from the track

passes a threshold. We believe this controller can keep the robot on track using the minimum number of commands.

We recorded the path of the robot controlled by this controller. The results look acceptable when a single robot is running (See Figure 6.3). After that we repeated the experiment with two robots. We also prevented the controller to send any message to a robot when it is in the middle part of the path, the area that is supposed to be the intersection area when we run the AIM. The reason is that, as mentioned earlier, the robots stop every time that they receive a message, so we do not want them to stop in the middle of the intersection. Figure 6.4 shows the paths of the robots. It is obvious from this figure that the robots are not very well controlled any more.

This experiment shows that the first version of the robots is not so controllable that AIM can handle many of them at the same time. Note that we have also tested other control mechanisms and the multi-region controller was the best among them.

The other reason that we didn't want to keep the robots on the track by sending motion control commands was that when we looked at the ratio of motion control and the traffic control (AIM) commands, we saw that the motion control adds an overhead of almost 400% to the total number of commands (see Figure 6.5). Knowing that the number of messages in AIM is already high, we doubt that the robots can handle the overhead added by motion control.

Consequently, we decided to implement a simple idea to keep the robots on the track: we used a piece of glassy plastic with grooves along the tracks. The robot's wheels fall into the grooves and move forward and backward. Now, the only type of motion control comments required is direction change commands when the robot reaches the end of the path. Figure 6.6 shows a photo taken from robots moving on grooves.

6.5 Results and Discussion

We tested the system with a single robot running as one of the vehicles while the rest of the vehicles are virtual. In this system, we ran the simplified AIM simulator in which one of the vehicles is linked to a Eco-be. The AIM simulator updates the robot with the position and orientation of the vehicle. If the vehicle stops behind the intersection, due to rejection of its request for reservation, a stop command is required to be sent to the robot. If the vehicle reaches the end of its path, a new move command is needed to be sent to the robot to change its direction. The position and the orientation of the vehicle is updated by the position and the orientation of the robot calculated from the images captured by the overhead camera. In this experiment, we observed that the robot is fast and reliable enough to play the role of a vehicle in AIM.

To do experiments beyond this point, the first version of Eco-bes is not a good candidate and the second version may be used. However for several good reasons, we decided not to implement the system using the second version of Eco-be: first, at the time we were working on this project, the hardware and the

software of the second version were still under development and the procedure for making them run was troublesome. The second, and the more important reason, was that we didn't see significant advantage in physical visualization of the system with such a simple robot. As mentioned earlier, the physical challenges that these robots face do not perfectly match with the challenges of real vehicles. For example, the robots are controlled by a centralized system that is subject to a single point failure, a problem that does not happen for real vehicles. On the other side, the acceleration or deceleration of robots happen in a very short time relative to the time scale of the system, while for real vehicles the acceleration profiles plays an important role in the trajectory simulation.

One might be still interested to continue this project for a couple of reasons: first, even if the robots are not suitable options for research, they may be still used with AIM for education or demonstration purposes. Second, there are other studies done on AIM in simulation that may never be wise to do with real vehicles. One is the crash simulation studies (see [12]). Therefore the second version of Eco-bes or a more complicated robot like Khepera[3] might be used to study the crashes in real-world settings.

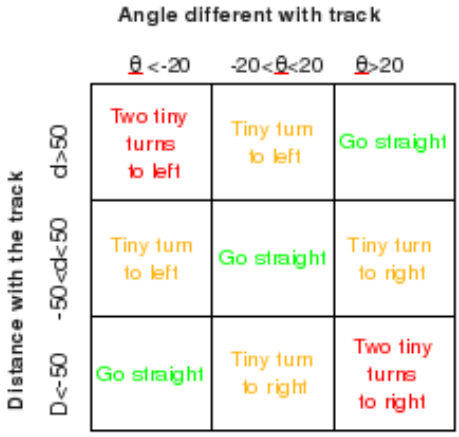

Figure 6.2: A multi-region controller to keep the robot nearby the track. D on the rows is the distance of the robot from the track and θ on the columns is the angle between the orientation of the robot and the direction of the track.

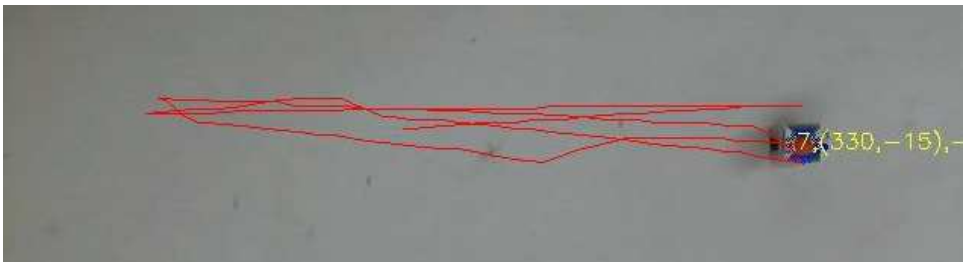

Figure 6.3: The path of a single robot controlled by the multi-region controller.

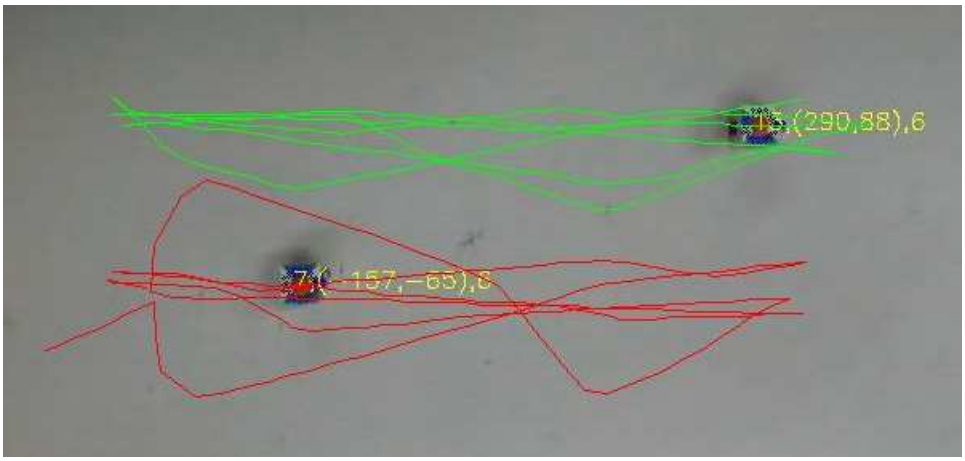

Figure 6.4: The paths of two robots controlled by the multi-region controller. The controller's commands are blocked when the robots are in the middle of path which is supposed to be the intersection area in AIM.

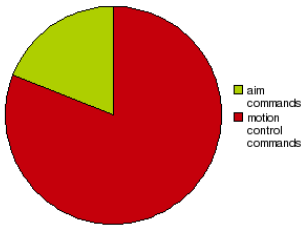

Figure 6.5: Motion control commands (red) vs. traffic control commands (green). The motion control commands add an overhead of almost 400% to the system

Figure 6.6: Photo of the robots moving on the grooves made on a glassy plastic.

Chapter 7

Conclusion and Future Work

The main contribution of this thesis was finding an intersection control mechanism to reduce an autonomous vehicle's delay. Previous work proposed FCFS policy in which the reservation requests are served as soon as they are received. We examined a sub-category of delayed response policies, called batch policies in which proposals are grouped in batches based on their arrival times. At each processing cycle, the proposals within the target batch will be processed. We proposed two methods of processing the proposals: the cost-based strategy and the clique-based strategy. The simulation results show that both of these strategies make significant improvement in comparison with FCFS. But they are not significantly different from one another. We predict that clique-based reordering strategy could perform better if the number of proposals within a batch is larger. The number of proposals within a batch can be increased significantly if we modify AIM to allow the vehicles that are behind another vehicles to submit requests. However, if a vehicle that is behind another vehicles receives a confirmation for reservation, it has to cancel it unless if all the vehicles in front of have reservations. Therefore the intersection manager needs to keep the track of reservations in each lane and examine a request by a vehicle, only if all the vehicles in front of this vehicle

have reservations.

A side advantage of batch policy is that it reduces the number of messages sent from the vehicles to intersections and Vice Versa. In simulation, it is not very important. However in reality, lower message traffic level can improve the communication and lower the overhead of message processing in intersection managers.

The AIM system can be improved both in average delay and average number of requests if the design approach is changed from the *request-response* approach to a *directive* approach. As in the request-response approach, the vehicles may need to send multiple request messages the message traffic is high. Also, because they do not have the information about the state of the reservation system, they cannot specify the arrival time and velocity deliberately. In a directive approach, a vehicle may send a request containing information about its physical properties as well as acceleration limits and let the intersection manager make the decision about the arrival time and velocity. This approach is particularly better for the vehicles stopped behind the intersection because the number of requests sent by these vehicles is usually much higher than the other vehicles. Also, because they are already stopped, they can arrive at any time and velocity within the range of their acceleration limits without changing their state. Using a directive approach combined with a delayed response policy, the average delay can be improved because the intersection manager has a higher degree of freedom to make the decision for the arrived proposals. This approach can also reduce the average number of requests to

1 request/vehicle (It can be occasionally higher than 1, if a cancellation of confirmation happens). A drawback of directive approach is that it centralizes computations in the intersection manager and reduced the autonomy of vehicles. It also increases the complexity of decision making in policy.

In an attempt to build the system in reality, we used miniature robots called Eco-be. Due to their cost and size, Eco-bes are good candidates for testing a multi-agent system with many agents. Despite the fact that the physical challenges of Eco-bes do not perfectly match those of full size autonomous vehicles, they are still useful for demonstration and education purposes as well as for the study of collisions for which experiments with full size vehicles are costly and dangerous. Because of limitations of the first version of Eco-bes, we tested a simplified version of AIM in a mixed reality framework in which a single vehicle is linked to a robot and the rest of the vehicles exist only in a virtual world. One can test the system with a higher number of robots using the second version of Eco-bes or other kinds of mobile robots.

Appendices

Appendix A

Technical Information of Eco-be

Table A.1: Stepper Motor Details

Type	Step Motor
Dimensions	7.0 * 8.5 * 1.9
Configuration	2 coils and 1 rotor
Control Terminals	4
Gear Ratio	1:240
Torque (gf . cm at 2.8 V)	between 2.0 and 4.0
Power Consumption at 200 rpm (mA)	between 4 and 12
Normal Rotation(rpm)	12.000
Direction	both standard and reverse

Table A.2: Micro-Controller Details

Manufacturer	Microchip
Type	PIC
Family	PIC18
Model	1220
Package	20-Pin SSOP
Operating Frequency	DC -¿ 40 MHz
Architecture	8bit
Instruction Set	75 instructions
Program Memory	4 kb Flash
SRAM	256-bytes
EEPROM	256-bytes
I/O	16
A/D	7
ECCP (PWM)	1
Timers 8/16 bit	1/3

Bibliography

[1] Eco-be manual.

[2] Intelligent transportation systems. http://en.wikipedia.org/wiki/Intelligent_transportation_system.

[3] Khepera ii homepage. http://www.k-team.com/mobile-robotics-products/khepera-ii.

[4] Mixed reality robot concept 2008. http://jeap-res.ams.eng.osaka-u.ac.jp/~guerra/MR08/TheConcept.html.

[5] Smart car. http://en.wikipedia.org/wiki/Smart_car#Intelligent_car.

[6] DARPA Urban Challenge. http://www.darpa.mil/grandchallenge/index.asp, 2007.

[7] Tsz-Chiu Au, Michael Quinlan, Nicu Stiurca, Jesse Zhu, and Peter Stone. Planning for improving throughput in autonomous intersection management. In *ICAPS'10 Workshop on Combining Action and Motion Planning*, 2010.

[8] Tsz-Chiu Au and Peter Stone. Motion planning algorithms for autonomous intersection management. In *AAAI 2010 Workshop on Bridging The Gap Between Task And Motion Planning (BTAMP)*, 2010.

[9] Kurt Dresner. *Autonomous Intersection Management*. PhD thesis, The University of Texas at Austin, 2009.

[10] Kurt Dresner and Peter Stone. Multiagent traffic management: Opportunities for multiagent learning. In K. Tuyls et al., editor, *LAMAS 2005*, volume 3898 of *Lecture Notes in Artificial Intelligence*, pages 129–138. Springer Verlag, Berlin, 2006.

[11] Kurt Dresner and Peter Stone. Sharing the road: Autonomous vehicles meet human drivers. In *The 20th International Joint Conference on Artificial Intelligence*, pages 1263–68, January 2007.

[12] Kurt Dresner and Peter Stone. Mitigating catastrophic failure at intersections of autonomous vehicles. In *AAMAS Workshop on Agents in Traffic and Transportation*, pages 78–85, Estoril, Portugal, May 2008.

[13] Kurt Dresner and Peter Stone. A multiagent approach to autonomous intersection management. *Journal of Artificial Intelligence Research*, 31:591–656, March 2008.

[14] Stone P. Shahidi N. Some problems of recurrent interestsome problems of recurrent interest. Technical report, Department of Computer Sciences, University of Texas at Austin, Austin, TX, 78712-0233, 2008.

[15] NHTSA. People saving people, on the road to a healthier future, Sept 1997. available at http://www.nhtsa.dot.gov/nhtsa/whatis/planning/2020Report/2020re

[16] NHTSA. Traffic safety facts, June 2009. available at http://www-nrd.nhtsa.dot.gov/Pubs/811172.pdf.

[17] Michael Quinlan, Tsz-Chiu Au, Jesse Zhu, Nicolae Stiurca, and Peter Stone. Bringing simulation to life: A mixed reality autonomous intersection. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2010. To appear.

[18] David Schrank and Tim Lomax. Urban Mobility Report 2009. `http://tti.tamu.edu/documents/mobility\_report\_2009\_wappx.pdf`, July 2009.

[19] Shimon Whiteson and Peter Stone. Adaptive job routing and scheduling. *Engineering Applications of Artificial Intelligence*, 17(7):855–869, 2004.

Vita

Neda Shahidi received her B.Sc. degree from University of Tehran in 2004 and her M.Sc. degree from University of Texas at Austin in 2010, both in Electrical and Computer Engineering. She has focused her researches on Artificial Intelligence including Intelligent Controllers, Evolutionary Algorithms, Artificial Emotions, Robotics and Autonomous Intersection Managment during the years 2002-2010. Since 2009, she has also started working in Computational Neuroscience.

Permanent address: 3577 Lake Austin Blvd. Apt A
Austin, Texas 78703

This thesis was typeset with LaTeX[†] by the author.

[†]LaTeX is a document preparation system developed by Leslie Lamport as a special version of Donald Knuth's TeX Program.