The University of York Department of Computer Science

Submitted in part fulfilment for the degree of BEng.

Approaches to Autonomous Vehicle Merge Management

Callum Hewitt

2nd May 2017

Supervisor: Lilian Blot

Number of words = 11566, as counted by TeXcount.
This includes the body of the report, but not equations or appendices.

Abstract

In anticipation of a fully autonomous vehicle future, this project aims to develop and analyse systems that deal with lane merging. An existing autonomous vehicle simulator, used for vehicle intersections, was adapted and extended to develop merge simulations. The system itself proved unexpectedly difficult to work with, and further research will require the development of a more universal simulator. A merge management system based on a queue protocol was developed to manage incoming vehicles on a single lane to single lane merge. This was compared with an adaptation of the intersection management system. It was found that the intersection management system induced less delay on the vehicles than the queue system, particularly at high traffic levels. The queue protocol was tested further under different conditions. The angle at which the two lanes meet was found to have a substantial effect on the performance of the protocol. When the two lanes met at shallow angles, the queue system performed very poorly, but the performance improved rapidly as the angle approached 90°. The queue system was less effective at reducing delays on a lane when it had a higher speed limit than the lane it was merging with.The distance simulated before the vehicle reached the merge point was found to have no effect on the performance of the protocol when the distance was greater than 150m. It was concluded that an intersection protocol based system could perform better than the queue protocol if fully developed. The performance of the protocol at different angles, speed limits and simulated distances, provided insight into issues that merge systems will have to overcome in order to be integrated into real world infrastructure.

Contents

List of Figures

List of Tables

1 Introduction

Autonomous Vehicles, or AVs, used to exist solely in science fiction. In 1953, Isaac Asimov wrote *Sally* [2] and in 1982, *Knight Rider* introduced KITT [3]. Today, AVs can be found on roads around the world. Alphabet's WAYMO is gaining traction, with cars being tested in four different US states [4]. Tesla have deployed their beta Autopilot system into all of their vehicles produced since September 2014 [5]. The system has been both hailed for saving lives and blamed for ending them [6] [7].

All of the AV systems currently running on public roads are designed to work alongside human driven vehicles, limiting their benefits. In order to truly embrace AVs, we need to design systems which assume every vehicle on the road is automated. The advantages of these systems are numerous, but the most important benefit is safety.

AVs would be able to react to incidents much more quickly than a human driver could. A driver's 'thinking distance' often determines whether someone survives an accident or not. This distance can be greatly increased if the driver of the vehicle is under the influence of alcohol or narcotics. An AV would be able to react to accidents much more quickly than a human, reducing the thinking distance and improving road safety. Figure 1.1 shows the impact thinking distance can have on the overall stopping distance of vehicle.

Another benefit of AVs is efficiency. Research suggests that by implementing fuel conserving driving strategies, AVs could be up to 10% more fuel efficient than current EPA fuel economy test results [8]. Fuel efficient vehicles are becoming increasingly important, with landmark climate change deals such as 'The Paris Agreement' introducing limits on greenhouse gas emissions globally [9]. The introduction of electric vehicles into the car market is also an important factor to consider, as the driving range of such vehicles has still not managed to match that of their gasoline counterparts. Introducing efficient driving strategies through AVs could help bring electric vehicle range up to par.

Congestion contributes to fuel waste in quite a large way. In 2014, the US wasted an estimated 3.1 billion gallons (11.7 billion litres) of fuel due to congestion [10]. Automated driving strategies, in situations such as

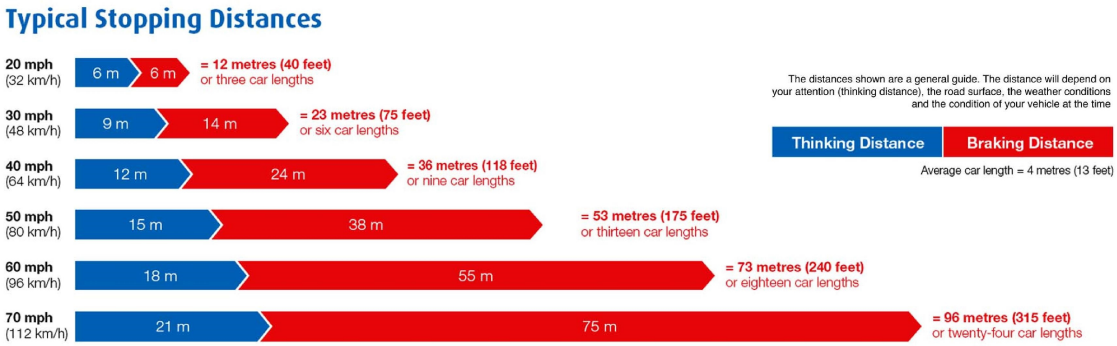

Figure 1.1: Diagram from Rule 126 in the UK Highway Code [1]

lane changes, could reduce congestion and improve efficiency. Dangerous lane changes don't even have to result in a crash to cause delays. If a car brakes due to a dangerous merge, it can cause a ripple effect, creating congestion. This ripple effect is known as a 'traffic shock' [11].

As well as more quantifiable benefits, AVs could also provide a level of comfort not currently available today. In a world where AVs are commonplace, it is not hard to imagine people working, reading or relaxing in their car instead of focusing on driving.

However, today there are still a number of concerns surrounding AVs, one major issue being the reliability of the systems governing the vehicle. Systems need to be responsive and accurate. They cannot afford to fail in such safety critical environments. Today, concerns over Tesla's Autopilot system are impacting the image of the company, and the system isn't even out of beta yet [12].

In order to address these concerns safely, we can create simulations which test the reliability of autonomous systems. Researchers at the University of Texas set up the Autonomous Intersection Management (AIM) project, which aims to "create a scalable, safe, and efficient multi-agent framework for managing Autonomous Vehicles at intersections" [13]. The team managed to apply their simulator tested intersection software in a mixed reality test, using a real life AV [14]. This demonstrates how vital simulators are when testing safety critical systems.

The motivation for the AIM system was to reduce congestion at intersections. Similarly to intersections, lane merges can be significant sources of congestion. AVs will need to be able to deal with various lane merge

situations if they are to become effective alternatives to manual vehicles. This project aims to develop a simulator that can effectively filter traffic through a lane merge. This simulator will be based on the AIM simulator codebase, which will also be considered for future AV projects if it can be adapted effectively. Using this simulator we aimed to compare different merge schemes, particularly looking at the effectiveness of decentralised systems against centralised systems. We also aimed to analyse how different merge conditions impact the performance of a merge system.

This project makes a number of assumptions. Firstly we assume that the sensors resolving the positions of the vehicle and it's surrounding obstacles are perfectly accurate. We also assume that all vehicles can reliably communicate with each other and with roadside infrastructure. These assumptions ignore existing areas of research which are not considered in this paper.

Chapter 2 examines existing work with merging AVs and compares centralised approaches to decentralised approaches. Chapter 3 takes a deeper look at the issues surrounding lane merges and the different types of lane merges that can be found on the roads today. Chapter 4 examines different approaches to the merge problem and their advantages and disadvantages. Chapter 5 details some of the problems encountered whilst implementing the merge approaches, and the reasons for some of the workarounds implemented. Chapter 6 analyses the performance of the Queue Merge Management system and compares it to a modified AIM implementation.

2 Literature Review

2.1 Car Following Models

Any autonomous-vehicle system will implement a 'car-following model', which defines actions for a vehicle based on the behaviour of its predecessors (the vehicles in front of it). One early car-following model was defined in 1981 by P.G. Gipps [15]. It was designed to mimic real-world driver behaviour, calculating a safe travelling speed for a vehicle based on the speed of its predecessor. A safe travel speed is defined as a speed at which the driver can safely stop if the preceding driver stops.

Gipps defined two equations applying constraints on the acceleration and braking profiles of the vehicles. Gipps' model worked well at describing the behaviour of traffic. However, translating this work to AVs poses a number of problems. Firstly, the work is based on the behaviour of real-world drivers in instrumented vehicles. This introduces human driver variables into the equations. Gipps' modelled reaction time, which will be far smaller for AVs. The gaps between successive vehicles are also larger than necessary. AVs are more precise than human drivers and can drive closer to their predecessors.

In 2000 Treiber et al. suggested the 'Intelligent Driver Model' (IDM) [16]. This model defines an acceleration profile for a vehicle as a continuous function. This function is based on the vehicle's current velocity, its desired velocity and the distance from the vehicle to its successor.

The IDM does not attempt to directly mimic human behaviour in traffic situations. It models a general acceleration and braking profile for a given vehicle. As such, it is well suited for adaptation by AV models, as seen in Kesting's work [17] in Section 2.3.2.

Gipps' model and the IDM also both fail to recognise, and incorporate, the use of vehicle-to-vehicle communication in their models. AVs could communicate with each other to help reduce overall travel time and improve efficiency. In vehicle platoons, such as those analysed by Kamali in 2016 [18], each vehicle autonomously follows it's predecessor, with the lead vehicle controlling the overall pace of the platoon. Platoons make heavy use of vehicle-to-vehicle (V2V) communication to allow vehicles

to join and leave, as well as to continuously control vehicle spacing and velocity. The advantage of a platoon is that all vehicles can accelerate and decelerate simultaneously reducing the effect of traffic shocks [11].

2.2 Centralised and Decentralised

We can divide approaches to AVs into centralised and decentralised solutions. Centralised solutions rely on an external agent to manage vehicles. Vehicles use vehicle-to-infrastructure (V2I) communication channels to send information and receive instructions from the external agent. Decentralised solutions use vehicle-to-vehicle (V2V) communication to let other vehicles know their state, their intentions, and to arrange any complex actions that might affect surrounding vehicles.

2.2.1 Centralised Systems

The Autonomous Intersection management system (AIM) described in [19] is an example of a centralised V2I system. The system works by dividing the intersection into a grid of $n \times n$ reservation tiles. Drivers 'call ahead' to the intersection sending information packets containing

1. The time the vehicle will arrive.
2. The velocity at which the vehicle will arrive
3. The direction the vehicle will be facing when it arrives
4. The vehicle's maximum velocity
5. The vehicle's maximum and minimum acceleration
6. The vehicle's length and width

The intersection infrastructure simulates the journey of the vehicle through the intersection, noting the tiles occupied by the vehicle at each time interval. If any cell is reserved at the same time step the intersection rejects the request. The driver will start decelerating and continue making requests until it obtains a reservation. It will not enter the intersection without a reservation, even if that means coming to a stop at the intersection.

A centralised reservation system works well in high traffic zones like intersections, because it forces all vehicles to communicate with a single

entity. This entity has a global-view of activity at the junction, allowing the system to make vehicle management decisions more easily. A V2V solution would require more complex communication protocols involving large numbers of vehicles. The volume of messages required for each vehicle to obtain a global view of the intersection would be considerably larger, and as such, most vehicles will never get a complete understanding of the status of the intersection.

This paper forms the foundation for the AMM protocol designed in Section 4.1.

2.2.2 Decentralised Systems

The main arguments against centralised systems generally tend to stem from concerns over feasibility and fault tolerance. A centralised V2I solution relies on one system always being available to manage vehicles. The original implementation of the AIM system works well, but if the system were to fail and no longer provide reservations, then approaching vehicles will simply halt at the intersection. In a worst case scenario, the system would still give reservations, but fail to compare them to reservations already in place, causing major car crashes in the intersection. Having, a single point of failure like this is a major concern, particularly when lives are on the line.

A paper by VanMiddlesworth et al. in 2008 [20] defined a decentralised version of the AIM model using V2V communication protocols. In VanMiddlesworth's model each vehicle can broadcast two different types of message. These messages are broadcast repeatedly with a specified period.

1. *Claim* This is a message indicating the vehicle's intention to traverse the intersection. It provides the vehicle's VIN, arrival lane, turning direction, arrival time and exit time. It also provides a message id, which increments when a new message is broadcast. Finally, the *Claim* message contains a boolean indicating whether the vehicle has stopped at the intersection.

2. *Cancel* This message releases any currently held reservation, it contains the vehicle's VIN and a message id, which acts the same as the message id in *Claim*.

Two *Claim* messages are in conflict if their paths, as determined by their lane and turn parameters, are incompatible and their time intervals, as

determined by their arrival and exit times, overlap. To resolve the conflict VanMiddlesworth's model determines which *Claim* has dominance. A claim C_1 dominates another claim C_2 if C_1's vehicle is stopped at the intersection and C_2's vehicle is not. If C_1 and C_2 both have the same value for the 'stopped at intersection' boolean, dominance is determined based on priority. Priority is indicated by the following rules, given in order of evaluation:

1. If neither vehicle is stopped at the intersection, the *Claim* with the earliest exit time has priority.
2. If both vehicles are stopped, the vehicle whose lane is 'on the right' has priority. This is defined similarly to current US 4-way stop rules.
3. If neither lane can be considered to be on the right the vehicle that is not making a turn has priority.
4. If no other priority order can be established, the vehicle with the lowest VIN has priority.

The protocol starts with approaching vehicles receiving messages from existing pending vehicles. An approaching vehicle may not start broadcasting it's own messages until it is within 'lurk distance' of the intersection.

Once within lurk distance the vehicle generates a *Claim* message for the earliest possible time the vehicle might arrive at the intersection. Once the vehicle has a *Claim* broadcasting, it may need to change it if it's looking like the vehicle might be late to the intersection, or if a competing *Claim* dominates it. A vehicle might also change its *Claim* to take advantage of a newly available time slot. In this situation the vehicle must then send a *Cancel* message and a new *Claim*. The *Cancel* message is sent repeatedly with the same period as the *Claim* message. Once the vehicle reaches the intersection it must traverse according to its current *Claim*, broadcasting throughout the traversal. At this point, the vehicle's claim cannot be dominated.

The main drive behind the unmanaged AIM intersection was to reduce cost. Adding in new infrastructure to an intersection costs money, and it might not be considered worthwhile for small intersections with only one or two lanes on each side. An unmanaged, decentralised system like that described by VanMiddlesworth would drastically reduce the cost to the state in creating automated road networks.

This paper forms the foundation for the decentralised protocol designed in Section 4.3.

2.3 Making lane changing decisions

Lane changes are a form of lane merging, in which the lanes are parallel. Some of the lane merging approaches here could help in designing merge protocols. The approaches could also be applied to multi-lane merges.

There are a number of reasons that a driver would want to change lanes. The most obvious being that the journey the driver wishes to complete requires the vehicle to move into a different lane. In this case the vehicle *must* change lanes before it reaches a critical position. Beyond this position the driver will need to change their planned route, most likely extending their journey time.

Another reason a driver might change lanes is in order to increase velocity, with the aim of reducing journey time. In general, a driver will aim to change lanes if their average velocity in their current lane is much less than that the velocity it could be achieving in another lane.

2.3.1 Lane Changing to hit a target lane

In 2016 Atagoziyev et al. described a centralised system for lane changes [21]. This system uses roadside infrastructure to help groups of vehicles change lanes before they reach a 'critical-position', such as a motorway exit or intersection. The vehicles send their position and velocity information to the roadside infrastructure. The system keeps track of the gaps between vehicles and their relative speeds, and then uses a series of equations to determine how to manipulate a vehicle into its desired lane. Each equation is used in a different context, based on the relative positions of surrounding vehicles. With the context identified, the system uses a finite state machine to determine the instructions to send to the vehicle.

Atagoziyev's model forces cooperation between vehicles by forcing them to move out of each others way in order to allow vehicles to reach their target lanes. This is far better than human drivers who have no requirement to act selflessly during lane changes. However, Atagoziyev's model does suffer from being a centralised system and cannot be applied to situations far from critical-positions. As most merges will be at 'critical-positions' however, this matters less to our problem. A simple merge

system that forces vehicles to take turns and allow others to pass through could work as well as Atagoziyev's model, which proved to be effective in multiple lane change scenarios.

2.3.2 Lane Changing to improve overall velocity

Work by Kesting et al. in 2007 [17] describes a decentralised model of lane changing that lets vehicles change lanes to increase velocity whilst still ensuring that the overall traffic flow is not disrupted. This helps to avoid traffic shocks and maintains smooth traffic flow. In order to do this, Kesting introduced the MOBIL or 'Minimising Overall Braking Induced by Lane Changes' model.

The model uses two criterion that the vehicle must satisfy. The first gives the vehicle a maximum deceleration value. The second is the 'incentive criterion', the motivation for a driver to change lanes. Whether a driver changes lanes or not is based on the relationship between the utility a driver gets by changing lanes, and the utility the vehicle behind the driver in the new lane loses by being pulled out in front of. How willing a driver is to sacrifice another driver's utility for their own is down to the driver's politeness factor p. With a p value of 0, drivers act selfishly, with no regard for the utility of other drivers. With a p value of 1, drivers act altruistically, only changing lanes when the is a net benefit to all drivers involved, at least above a set threshold. A p value of 1 caused the maximum lane changing rate to halve. Kesting also discovered that 'altruistic' lane changing behaviour increased the mean speed of both lanes involved in the simulation, improving overall traffic performance.

MOBIL could be used to manage merges with slip lanes, with drivers only merging when it has the least impact. The politeness factor would have to change as the vehicle came closer to the end of the slip lane however, as eventually the vehicle must merge, or suffer a collision.

3 Problem Analysis

Lane merging is not a straightforward problem with a single solution. There are many different types of lane merging scenarios as well as a number of factors which add more variance to the problem. Section 3.1 analyses some of the different merging scenarios to better define them. Section 3.2 indicates how success of a merge scheme can be measured and Section 3.3 defines some factors that could alter the behaviour of a given merge scenario.

3.1 Merge Types

In this paper we focus on merges made at 'critical positions' such as junctions. This is true for all of the merge scenarios analysed. Figure 3.1 illustrates some of the merge scenarios described.

3.1.1 Single-to-Single Merge

A single-to-single merge (S2S merge) describes a situation where a vehicle moves from a single lane road into another single lane road. In this situation we label the lane that vehicles are moving from, the 'merge lane' (ML), and we label the lane that vehicles move to, the 'target lane' (TL). The vehicles on the TL generally tend to be faster moving. We describe

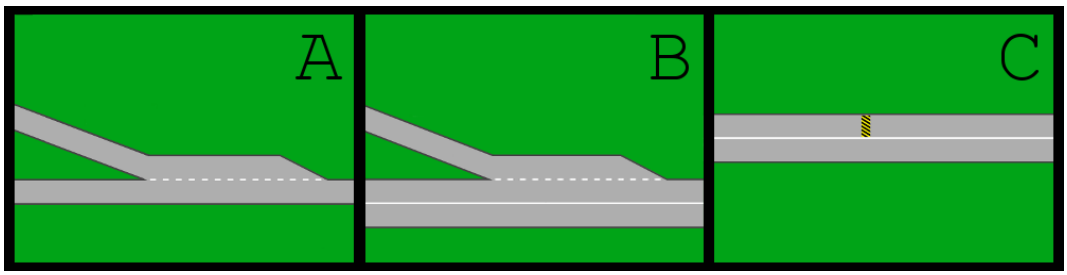

Figure 3.1: a) S2S with slip lane, b) S2D with slip lane, c) Lane obstruction

the vehicles that start on the ML as 'merging vehicles' (MVs) and the vehicles that start on the TL as 'TVs' (TVs). We have our critical position where the ML and TL intersect. We call this area the 'merge zone'.

The main issue with an S2S merge stems from the limited options available to vehicles arriving at the critical position. Target vehicles do not have the opportunity to move laterally out of the way of MVs, and vehicles on both lanes could struggle to reduce their velocity without affecting their successors.

Many S2S merges are performed with an attached slip-road. A slip-road gives MVs more time to travel parallel to the TL before merging. This makes the merge easier for both MVs and TVs as MVs don't slow down in front of TVs in order to make the turn into the TL. Figure 3.1 A shows an S2S Merge with a slip lane.

3.1.2 Single-to-Double Merge

A single-to-double merge (S2D merge) describes a situation where a vehicle moves from a single lane road into a double lane road. In this situation we have two TLs. The upper lane which directly links to the merging lane is called 'target lane 1' (TL1) and the lower lane is called 'target lane 2' (TL2). We still have only one critical position where the merging lane meets TL1.

An S2D merge provides more options for vehicles on the targets lanes at the critical position. Target vehicles now have the opportunity to move laterally to avoid MVs. Two lanes also allows for more vehicles on the TL which should give vehicles greater freedom to adjust their velocity without affecting their successors, at least when compared to the same number of vehicles on a single lane.

S2D merges can also take advantage of a slip-road. Figure 3.1 B shows an S2S Merge with a slip lane.

3.1.3 Lane Obstruction Merge

A lane obstruction merge is where a vehicle needs to change lanes to avoid an obstacle in their way. It is essentially an S2S merge although the vehicle moves laterally to avoid the obstacle. In this situation the critical position is a point shortly before the obstruction.

The obstacle could be a broken down vehicle or some debris on the road. Because of the unexpected nature of the obstacle it may sometimes be difficult to have a centralised approach to the problem. Although, if

the obstacle was a broken down vehicle, the vehicle might be able to act as the centralised system managing approaching vehicles. Figure 3.1 C shows an a lane obstruction merge scenario.

3.2 Measuring Success

In order to evaluate the effectiveness of solutions to the problems we need to define measurements of success.

Solutions to the merging problems above must satisfy the following conditions:

1. *No collisions* This means avoiding collisions at the critical position between MVs and TVs, as well as avoiding collisions between vehicles on the same lane.

2. *Minimise delays to both lanes* Vehicles should not suffer large delays to travel time due to the merge. This means measuring average delay for both the ML and TL to ensure that the system is not starving one lane for the benefit of vehicles on the other.

3. *Maximise throughput* By minimising delays and velocity loss we aim to maximise the throughput of the critical position.

4. *Minimise changes in velocity* Though not strictly a performance requirement, the system should minimise changes in vehicle velocity, for both passenger comfort and vehicle efficiency.

We need to measure how well solutions meet these conditions.

3.2.1 Collisions

Preventing collisions is a basic safety requirement for any autonomous vehicle system. We can measure this by comparing the positions of vehicles in the system, and ensuring that there is no overlap.

We should also consider measuring near misses. We can define a minimum spacing between vehicles, perhaps equal to the minimum braking distance of the vehicle plus an additional comfort distance. This would mimic the IDM model [16].

Any collisions that do happen should be reported immediately. The system should automatically be considered a failure.

3.2.2 Delay

Delay measures the effect that the critical position had on the overall journey of the vehicle. It is the primary metric considered in Dresner et al.'s 2004 paper [19] on AIM.

Dresner et al. provide the following equation for measuring average delay.

$$\frac{1}{|C|} \sum_{v_i \in C} \big(t(i) - t_0(i)\big) \tag{3.1}$$

C is the set of vehicles that pass through a critical position within a set time frame. Assuming no other vehicles on the road, a vehicle v_i would complete it's trip in time $t_0(i)$, otherwise v_i would complete it's trip in time $t(i)$. We can represent this trip for vehicles in the simulator as the time difference between the vehicle spawning in and the vehicle being removed from the simulator.

To ensure fairness the mean delay will be measured across each lane.

3.2.3 Throughput

By minimising delay we should also maximise throughput; the two are closely related. However we should also collect direct metrics.

$$\text{Vehicle throughput} = \frac{|C|}{t} \tag{3.2}$$

Here t is the time it took for all of the vehicles in C to pass through the critical position. We will also need throughput measurements for each lane separately.

3.2.4 Velocity Changes

Vehicles should aim to reduce velocity changes as much as possible, particularly rapid changes. Ideally autonomous vehicles should have very smooth acceleration and braking profiles. This both increases passenger comfort and improves fuel efficiency. To measure maximum acceleration and deceleration we can measure the change in velocity at each time step in the simulator, and divide this change by the time-step length.

3.3 Merge Variance Factors

In each of the scenarios in section 3.1 the road layout is fixed. More variance can be introduced to the scenarios by altering other factors. Not all of the factors below will be applicable in every merge scenario. Figure 3.2 shows some of the factors below, applied to an S2S merge.

- *Traffic Level* The traffic level changes the traffic density. It is measured in vehicles per hour per lane (vhl)
- *Target lane speed limit* The maximum speed that vehicles on the target lane can travel.
- *Merge lane speed limit* The maximum speed that vehicles on the merge lane can travel.
- *Target lane lead in distance* The distance between the point at which TVs arrive in the simulation and the point at which the TVs reach the merge zone.
- *Target lane lead out distance* The distance between the end of the merge zone and the end of the target lane (at least the end in the simulator).
- *Merge lane lead in distance* The distance between the point at which MVs arrive in the simulation and the point at which the MVs reach the merge zone.
- *Merging angle* The interior angle θ at the point where the merging lane meets the target lane.
- *Slip-road length* The length of the slip-road in a merge that uses a slip-road.

Traffic level has a fairly obvious effect on performance in a particular merge scenario. The more vehicles that try to merge together the more difficult it will be for the merge to happen. It will likely require vehicles to move more slowly. Increased traffic density also increases the likelihood of traffic shocks, and vehicle manoeuvrability is impacted.

Altering speed limits could also affect performance. Higher speed limits could put pressure on systems that process the vehicles. Vastly differing speed limits for each lane could impact how easily a vehicle finds it to merge into another lane, and adapt to its speed limit.

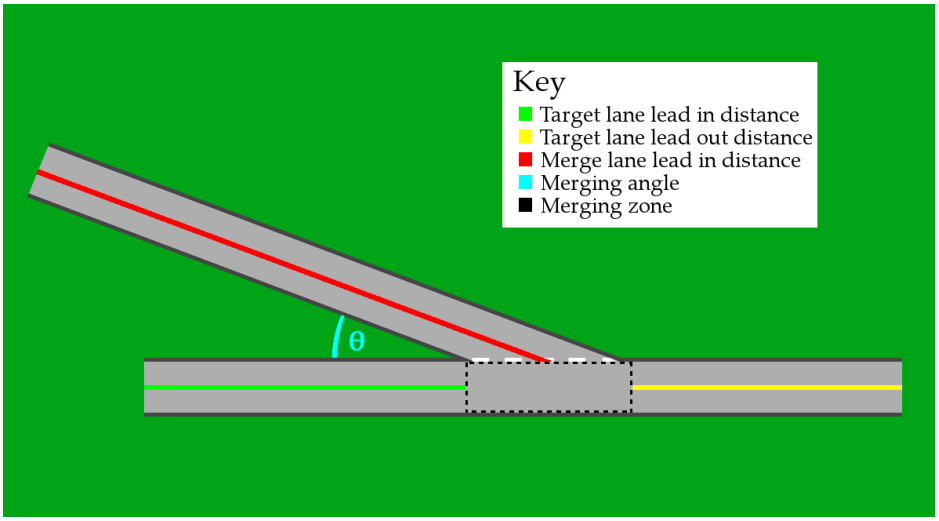

Figure 3.2: An S2S merge marked with some of the variance factors

Lead in distances change the amount of deliberation time vehicles have before they reach the junction. It also changes the distance they have to change their velocity in. This means that shorter distances could lead to larger acceleration and deceleration values as well as sub-optimal solutions to the merge.

The merging angle changes the heading at which MVs arrive, but also the length of the merge zone. This could impact how long it takes a vehicle to traverse the merge zone as well as affecting how easily the vehicles adjust to the new lane heading.

3.4 Requirements

Using the problem analysis above, we can define requirements for our final system. This system will not deal with every merge scenario described but will instead set the groundwork for further research. The complete requirements are given in Appendix A.1. A shortened version with the most important requirements are given here.

3.4.1 Functional Requirements

The functional requirements describe the functionality required within the simulator. They are broken into two sets of requirements. User requirements and system requirements. User requirements describe the behaviour expected from the simulator from a user perspective. System requirements are all associated with a user requirement. They describe the functionality required by the system in order to satisfy the user requirement. Table 3.1 shows some of the functional requirements.

Table 3.1: Functional requirements table.

User Requirements		System Requirements	
FU.3	Users can run merge simulations.	FS.13	The system can run merge simulations
FU.5	User can view the activities of a simulation.	FS.51	The system displays the current status of a simulation as it runs.
FU.6	Users can export the results of a merge simulation.	FS.61 FS.62	The system has controls for exporting results data. The system can produce a file containing results data from the simulation.
FU.7	Users can select and run an S2S merge simulation.	FS.71	The system can produce S2S simulations.
FU.8	Users can select a centralised merging scheme with S2S merge simulations.	FS.81	The system can use an AIM-like merge management system for the merging zone in an S2S simulation.
FU.9	Users can select a decentralised merging scheme with S2S merge simulations	FS.91	The system can use an merge management scheme similar to that described in "Replacing the Stop Sign: Unmanaged Intersection Control for Autonomous Vehicles" [20].
FU.16	Users should be alerted of any collisions during a simulation.	FS.161	The system should be able to alert the user if a collision occurs.

3.4.2 Non-functional Requirements

The non-functional requirements describe the expectations of the simulator that are not actions the simulator will perform. Table 3.2 shows some of the non-functional requirements for the simulator.

Table 3.2: Non-functional requirements table.

ID	Description
NS.6	All data displayed to the user should be accurate.
NS.7	All data in the results file should be accurate.
NS.9	Code should be written to allow for easy expansion.
NS.11	The simulator should be integrated into the AIM4 simulator without negatively affecting the performance of AIM simulations.

4 Design

For the initial development of the simulator I focused on creating a working prototype for S2S merges. The S2S merge is one of the most simple merge scenarios that an AV might encounter. The designs here can then be expanded at a later date to include some of the other scenarios in 3.1.

4.1 Autonomous Merge Management System (AMM)

This centralised system is based on the AIM system [19]. Once in range, approaching vehicles send a request message to the system with their vehicle specification and their predicted arrival time and arrival velocity. A vehicle specification contains a vehicle's maximum velocity, length, width, maximum acceleration and maximum deceleration.

Upon receiving a request the management system simulates the vehicle's journey through the merge zone, which has been split into a grid. As the vehicle is simulated, the system notes the cells the vehicle moves through and the times at which it occupies each cell. The system then compares these 'space-time' values to its reservation store. The reservation store consists of sets of space-time values indicating where and when vehicles are expected to be in the merge zone. If the requesting vehicle does not clash with any of these reservations then the request is granted and the reservation store is updated. If not, the system rejects the request. The requesting vehicle will have to try and make a new reservation. They cannot enter the merge zone until they obtain a reservation.

Vehicles with a reservation continue towards the merge zone at the speed which allows them to enter at their scheduled arrival time. If a vehicle cannot maintain their reservation for any reason, it is their responsibility to alert the management system, at which point their reserved space-time values will be released and the vehicle will have to make a new reservation. Once a vehicle has successfully traversed the merge zone they will need to send a message to the management system to alert the system that they are leaving its zone of control.

This system makes good use of space-time, allowing multiple vehicles into the merge zone at the same time. However, the system does have a number of issues. The primary one is that the leading vehicles in both lanes, that is the ones before the merge zone, will always be the only two with reservations. Lead vehicles are the only vehicles that can accurately predict their arrival time, as they don't have to consider the braking behaviour of any predecessors. A consequence of this is that successor vehicles very close behind the lead vehicle are forced to make reservations very close to the merge zone. These vehicles have to slow down to avoid entering the merge without a reservation, and they become more delayed.

Another problem is that the system does not guarantee that one lane will not experience large delays at the expense of another. If a vehicle on the target lane arrives first and makes a reservation, then a vehicle on the merge lane arrives and has to wait, there is nothing to say that twenty more target lane vehicles will pass through before the merge vehicle ever manages to make a reservation. A queued merge management system might be able to counteract this, as seen in section 4.2.

4.2 Queue Merge Management System (QMM)

The queue merge management system is another approach to centralised vehicle control. Unlike a reservation based system, where vehicles enter the merge at a pre-arranged time; a queued merge management system controls vehicles with a simple go/no-go system. Vehicles are sent messages telling them to enter the merge zone, without knowing exactly when that might be.

As a vehicle approaches the merge it sends a request to the management system. This request contains the vehicle's ID, the ID of the vehicle's predecessor and the vehicle's distance to the merge zone. The predecessor ID and distance measurements are assumed to be accurate. In a real world scenario a vehicle could lie about these parameters, but as these parameters could be verified or collected by the management system instead, we assume them to be accurate.

An approaching vehicle has to be within a set distance of the merge zone in order to be added to the queue of vehicles being processed through the merge. This is to mitigate the effect of very fast vehicles being forced to slow down, in order to allow slower vehicles earlier in the queue to pass through the merge zone. Any request from a vehicle further away from the merge zone that this distance is rejected.

If the vehicle's request is from within an acceptable distance, then the queue system checks to see if the vehicle's predecessor has been added to the queue. There is a possibility that the predecessor has not managed to make a request yet. This check ensures that the vehicle's predecessor is added to the queue in a position before the vehicle. If the vehicle's predecessor has not yet been added, then the vehicle's request is rejected. Otherwise the vehicle's request is accepted with a confirmation message, and the vehicle's ID is added to the queue.

The vehicle is now awaiting a go message from the queue system. During this time it cannot enter the merge zone and must stop before it enters. The queue system sends go messages to vehicles in the queue as the previous vehicle that was sent a go message leaves the merge zone. This ensures that only one vehicle is in the merge zone at any time. Once a vehicle receives the go signal it must make its way towards the merge zone and traverse it. Once the vehicle leaves the merge zone it sends a message to the queue system confirming that it has safely traversed the merge and that it can let the next vehicle through.

This system sacrifices space efficiency for simplicity and a focus on fair access across both lanes. The system cannot have more than two vehicles in the merge zone at the same time, as the AIM-based system can. The system also fails to guarantee a vehicle a time at which it can move through the merge zone. The time a vehicle spends in the queue depends on the speed of the vehicles ahead of it.

4.3 Decentralised Merge Management System

The decentralised merge management system is based heavily on the work by VanMiddlesworth et al. in "Replacing the Stop Sign: Unmanaged Intersection Control for Autonomous Vehicles" [20]. This system is based on two message types, which each vehicle broadcasts to every other vehicle within range. The first message type is a *Claim*. This message is used to try and reserve access to the merge zone. It contains the vehicle's ID, lane, estimated arrival time, estimated exit time and a boolean indicating whether or not it has stopped at the merge. The second message type is a *Cancel*, used to release any reservations held by a vehicle. *Cancel* messages contain a vehicle's id. All message types also contain a message ID which monotonically increments with each new message sent by the vehicle. Vehicles broadcast both message types repeatedly with a constant period to ensure that the messages are received

by all other vehicles.

Again we assume all of the values provided by the vehicles are accurate. VanMiddlesworth [20] goes into further detail over dealing with selfish and malicious agents, however in this scenario we are assuming all vehicles are forced to provide accurate information.

Claims can compete with each other, in which case Claim dominance must be established. A claim C_1 dominates another claim C_2 if C_1's vehicle is stopped at the merge and C_2's vehicle is not. If C_1 and C_2 are either both stopped at the merge or both not stopped at the merge, dominance is determined based on priority. Priority is indicated by the following rules, given in order of evaluation:

1. *If neither vehicle is stopped at the merge* The *Claim* with the earliest exit time has priority.

2. *If both vehicles are stopped at the merge* The *Claim* on the target lane has priority. This is because in real world scenarios maintaining traffic flow generally means giving way to the target lane.

3. *Otherwise* The vehicle with the lowest ID has priority

As a vehicle approaches the merge it will receive messages from other agents. Before acting, the vehicle 'lurks' long enough to be reasonably sure of every pending *Claim* message. The vehicle then generates it's own *Claim* message based on the earliest possible arrival and exit time of the vehicle. Once a vehicle has a *Claim* they will continue broadcasting until they either complete the merge or have to change it. Once in the merge, the vehicle traverses in the manner their *Claim* indicates. They continue broadcasting their *Claim* during this time, but it cannot be dominated.

If the vehicle's arrival estimation changes (due to a preceding vehicle braking or something similar) the vehicle generates a new *Claim*. If another vehicle arrives at the merge before they do, the vehicle broadcasts *Cancel* repeatedly with the same period as *Claim*. This helps to deal with the 'lead vehicle only problem' found in the AMM system. Once cancelled the vehicle then attempts to make a new *Claim*.

This system has the advantage of being completely decentralised, requiring no extra infrastructure to handle requests. However, the complexity of this system is far higher, requiring multiple agents to perform complex simulations and attempt to maintain a global picture of the system. In a centralised system, these activities are far easier to organise.

5 Implementation

The final implementation was done in Java, and was built on top of the AIM simulator codebase.

5.1 Generalising the Codebase

The use of the AIM codebase was a project restriction imposed for research purposes. By working with the AIM simulator codebase I could learn how easy it is to work with, and analyse whether or not it would be a good codebase to continue expanding upon for future AV projects. Each simulator built for this project works alongside the AIM simulators, whilst being completely independent. The project: "A self-organising approach to autonomous vehicle car park management using a message-based protocol" [22], also required simulators to be built using AIM. In addition, both projects had to work alongside each other using the same codebase. To make sure that code coupling was reduced as much as possible, I worked closely with their project lead to generalise the codebase, breaking out useful shared features so that they could be accessed by all simulator types. The final code can be found at https://github.com/CallumHewitt/AVSimulatorProject [23]. Breaking out the code like this satisfied requirements *NS.11* and *NS.12*.

Appendix A.3 provides detailed coverage of the changes made to the AIM codebase.

5.2 Merge Schemes

The AMM protocol implementation was developed by examining the original AIM code and creating a modified version applicable to merges. Because the two systems are so similar, much of the code was duplicated. This could be refactored out a later date, but during development having full control over the actions taken by a vehicle without having to compromise to allow AIM to work correctly was very useful. Despite this approach there were significant issues with the system. Even using

very similar approaches to AIM, almost identical in areas, vehicles would continue to arrive early to their reserved times and vehicles would collide consistently at intersections. The reasons for these errors are described in more detail in Section 5.3.

To correctly implement the AMM system much of the fundamental generalised code that the system was built upon would have had to be replicated or rewritten to ensure that the system was working accurately. Because of this, and due to time constraints, a full version of the AMM system was not implemented. As such requirement *FS.81* was not satisfied. As an alternative, a modified version of the AIM simulator was developed. In this version the AIM simulator was restricted to only spawn vehicles from the south and west roads, with each vehicle's destination set to the east road. This effectively mimicked an AMM system with a 90°merge angle.

The QMM system was implemented as a replacement for a fully implemented AMM system. The system was expected to have good performance, possibly rivalling that of the AMM system. The system was also simpler to implement than the AMM system, and despite issues surrounding some of the generalised methods, the system was fully implemented. With QMM implemented, we now had a system which could be used to analyse how the merge variance factors, such as merge angle, speed limit and lead in distance, affect a system. We can also compare the performance of the AMM and QMM protocols.

The QMM system was implemented with a request distance limit of 150m. Vehicles further from the merge than this distance, cannot be added to the queue.

The original plan was to compare a centralised system with a decentralised solution, based on the work from VanMiddlesworth [20]. However, this was never implemented due to the difficulties with the system, and time constraints. As such requirement *FS.91* was not satisfied.

5.3 Simulation

Each simulation consists of multiple interacting agents, which makes it a difficult problem to implement. Using some of the generalised AIM classes helped to reduce the amount of time it took to implement these components. However, using AIM did introduce some complications and parts of the core code had to be rewritten to adjust for this.

5.3.1 Drivers

Driver agents are responsible for manipulating the vehicles in the simulation. They make requests to centralised merge managers and act upon the responses they are given. Each driver acts as a finite state machine, performing specific sets of actions for each state. Vehicles and Drivers both extend from generalised Vehicle and Driver classes, which contain useful functions for following lanes, turning and determining distances. However, some of these methods proved to be flawed.

One of the main issues was the assumption that lanes and roads will always meet at 90°. This caused a number of small issues throughout development, but one key problem was turning. A turn through an intersection in the AIM simulator is done by forcing the vehicle to point to a coordinate further down the lane the vehicle is following. This point is always exactly the same distance away from the vehicle, such that when the vehicle reaches a corner, and the lane it's following changes, the vehicle will turn towards that point gradually. This distance proved too much for some merges, and resulted in the vehicle making turns too gradually. This was fixed by setting the turn distance to always be the distance from the point at which the vehicle enters the merge zone, to the merge zone exit. The target point would also always lie in the centre of the target lane. This caused merging vehicles to turn tightly, freeing up the lane for more vehicles. Figure 5.1 shows how these turns work.

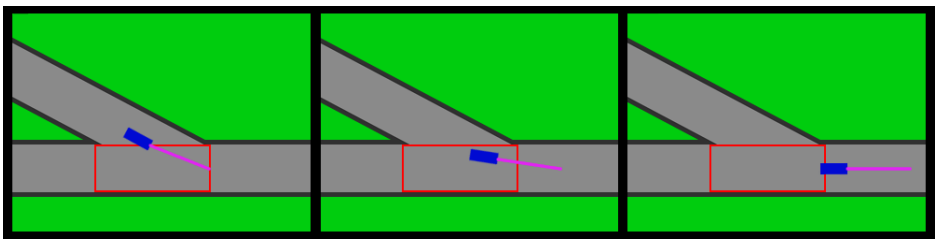

Figure 5.1: A digram indicating how turning works through a merge.

Another problem came about due to collisions. AIM had provided a method called *dontHitCarInFront* which calculates the distance from the vehicle to the vehicle in front, and then takes action to avoid a collision, slowing down if necessary. This method turns out to not be completely effective, particularly at high speeds. Even within the original AIM simulations, vehicles collide. Due to time constraints collision detection

was removed from the project, as fixing the issue would have required in depth analysis of core methods. As such requirement *FS.162* was not fulfilled. As far as I can tell, no collisions take place within the QMM merge zone, as it should be almost impossible by nature of the protocol. However, without having a check in place this cannot be said for certain. The AIM protocol also avoids collisions in the merge zone by switching to 'acceleration profiles' instead of using the *dontHitVehicleInFront* method.

5.3.2 Merge Managers

The role of a merge manager is to take requests from drivers and provide responses, controlling the flow of traffic through the merge zone. They were heavily influenced by the approaches taken by AIM. The AMM merge manager replicated much of the intersection manager code introduced by AIM. At a later date this could be refactored to reduce duplication, but this would most likely also require changes to Driver and Vehicle, as each type of merge manager is built to deal with different types of vehicles and drivers.

The final implementation of a merge manager, the *QueueV2IManager*, is designed similarly to AIM's *V2IIntersectionManager*, however much of the code is original and far simpler. The AIM system is more complex, requiring the merge manager to monitor reservations and time the arrival of vehicles. The AIM system relies very heavily on each vehicle arriving at their stated time and fails to handle vehicles well if they miss their reservation or arrive too early.

5.3.3 Map

The simulation map stores all of the spawn points, lanes and merge managers for the simulation.

By using some of the generalised components for calculating distances, creating the map was relatively straightforward, however, there were some components that failed to work as intended. The original AIM system assumes that every road meets at 90°and as such some methods were inappropriate for when lanes meet at other angles. One example of this is the no vehicle zone at the beginning of each lane. This zone stops multiple vehicles from spawning on top of each other. The original implementation calculated a *Rectangle2D* at the beginning of each lane, which works fine when lanes are at 90°. For my no vehicle zones, I had to define a path around the start of each merge lane and create the zone

shape from that. This more complicated approach was necessary due to the angles at which the merge lane can meet the target lane.

The map also controls the spawn points creating the vehicles. Most spawning behaviours were based on the AIM spawn points. However, to enable consistent testing we wanted to be able to repeat the experiment with the same vehicles over and over again. To do this I created a new vehicle spawn type that uses a JSON file to spawn vehicles. The file contains an array of pairs of vehicle specifications and spawn times. The spawn point reads this data in and spawns a vehicle with the given specification at the indicated time. I also implemented this type of spawner into the AIM system. This means that direct comparisons between the performance of AIM and QMM are now possible.

5.3.4 Simulation Control

The simulator itself is responsible for triggering and monitoring the actions of each component of the simulation. It delivers messages between merge managers and vehicles and moves vehicles through the simulation according to their specified velocities and headings. The simulator controller satisfies requirements *FS.12*, *FS.13*, *FS.22*, *FS.32*, *FS.42*, *FS.71*, *FS.73*, *NS.3*, *NS.4* and *NS.8* by managing all of the components in the simulation and producing results.

5.4 Results Production

In order to provide results Vehicle objects were provided with fields to store their statistics. These fields were:

- Delay
- Final Velocity
- Maximum Velocity
- Minimum Velocity
- Final X Position
- Final Y Position
- Start Time

- Finish Time
- Starting Road

These fields were implemented in both AIM and Merge vehicles. To calculate the delay the simulator first simulates each vehicle specification for both the merge and target roads. The completion times for each specification are recorded and then used to calculate the effect the merge protocol had on each vehicle. The start time and starting roads are initialised in each vehicle when they are created by their spawn point. The maximum and minimum velocities are dealt with after the *moveVehicles* method in the simulator. The simulator compares the current velocity of the vehicle to its stored maximum and minimum velocity and updates as necessary. The finishing variables are dealt with when the simulator removes them from the simulation and adds them to the completed vehicle store.

Maximum acceleration and deceleration measurements were not implemented, as such requirements *FS.6a*, *FS.6b*, *FS.6d* and *FS.6e* were not satisfied. All other requirements connected to *FU.6* and *NS.7* were satisfied. The current design of the AIM simulator uses the maximum deceleration for each vehicle as they approach the merge zone, braking at the last moment as aggressively as the vehicle's specification allows. This obviously makes maximum acceleration and deceleration measurements pointless. In real life braking profiles like this won't provide the most comfortable ride to passengers, and it also means that vehicles cannot pre-emptively slow down to maintain momentum whilst other vehicles to move through the merge. Further work could be done to expand both the AIM and Queue protocols to allow for pre-emptive acceleration profiles like these.

The results can be saved to a CSV file containing the throughput, maximum delay, mean delay and minimum delay of the system. There are also results for each vehicle. In addition to the list above, each result contains the vehicle's

- VIN
- Starting Road
- Vehicle Specification Name

5.5 GUI

The GUI for the project was built using Java Swing, extending the existing GUI. New simulator types are given a separate tab in the application with their own simulator setup and a display screen. The display screen can be modified for each simulation type, showing the relevant information for that simulation. We moved away from the AIM graphical implementation to a *StatScreen* implementation which shows information in text format instead. The S2S simulations display the current simulation time, number of completed vehicles and throughput. They also display two tables, one containing information about the vehicles currently in the simulation, and another for vehicles that have left the simulation.

Functional requirements *FS.82* and *FS.92* were not fully implemented as the AIM-like simulations and Decentralised simulations were never implemented in a fully working form. *FS.161* was also never implemented due to the removal of collision detection from simulations. All other GUI requirements were completed.

In order to enable repeat experiments with the same vehicles spawning at the same time, the GUI needed to support the input of merge schedule JSON files to the spawn points. This was implemented using *JFileChooser*. I also added an extra feature to the merge setup panel which allows users to generate spawn schedule files for a specific time period at a specified traffic level. This enables users to generate schedules without having access to the codebase.

5.6 Maintainability and Testing

5.6.1 Maintainability

The separability imposed between the AIM, Merge and Generalised classes allows developers to create new simulations quickly, without having concerns over the effect they'll have on existing work. In general, as long as the developers extend and modify certain key classes they can create without worry. For example, new developers will have to create a new *SimSetupPanel* and *SimViewer* panel in order for their simulation to appear in the GUI. The separability isn't perfect, there are some instances where AIM specific classes are used by Merge (mainly in the implementation of AMM), but in general most of the classes are broken out correctly, satisfying requirement *NS.9*.

In terms of maintaining existing code, the prospects are quite good. The majority of the complex code is documented using JavaDocs and comments, satisfying requirement *NS.10*. Once familiar with the codebase, it is relatively easy to find the files you are aiming to change. Some sections are quite complex however. Many AIM components are tightly coupled, particularly surrounding the I2V managers and reservation classes. These areas use callback interfaces that result in confusion over the role of each class. Refactoring this out would be hard, as there isn't really a perfect solution, but it could be improved with further development time. The separability of the different simulators also comes at a cost in terms of class structure complexity. The changes made to the vehicle and driver classes in Appendix A.3 should indicate how complex some of the class structures have become.

The main problem with the codebase surrounds the core generalised code. Because the system is based on AIM, attempting to develop something beyond the original specification, such as using lanes that don't meet at 90°, can be problematic. These problems have already been described in Section 5.3.

5.6.2 Unit Testing

Unit tests were mostly used to ensure getter and setter methods worked as expected. However, some unit tests were used to verify the behaviour of classes. To do this I used Mockito [24] to mock the behaviour of objects used by the test class so that I could prompt the test class into producing the expected results. Mockito was particularly useful in dealing with the components with high coupling.

5.6.3 Integration Tests

Integration tests were the most useful tests I used. They allowed me to find and remove problems with the simulators by observing how the map, vehicles, drivers and simulator objects interacted together, as this was usually where most of the errors were occurring. These tests also allowed me to identify key errors with the AIM simulator's approach that were causing problems with AMM.

6 Results

The simulator was designed to allow for variance in merge angles, lead in distances, speed limits, and traffic levels. This means that we can experiment to see what effect each of these variables has on the effectiveness of the QMM protocol. We can also compare the QMM protocol to the AIM protocol, using the modified version of the AIM simulator described in Section 5.2.

6.1 Experimental Procedure

All experiments were done using pre-generated spawn schedules. In each experiment I used 20 pairs of schedules (1 schedule per lane). Schedule pairs are identical for tests with the same speed limit and traffic rate. Vehicles spawned for 1000 simulated seconds, and all vehicles were allowed to complete. The spawn schedules would only fail to spawn a vehicle if the spawning area was occupied by another vehicle. This caused reduced numbers of completed vehicles if the system became congested enough to cause queues up to the spawning area.

6.2 Comparing AIM and QMM

By using the modified AIM simulator described in section 5.2, I obtained approximations for how well an AMM implementation would handle merges.

The AIM simulator has a lead in and lead out distance for each lane of 150 metres and is limited to 90°merges. These settings were duplicated for QMM. All of the lanes were set to have a speed limit of $20ms^{-1}$ (44.7mph or 72kph). The traffic rate (vehicles/hour/lane (vhl)) was altered to see how well the systems adjust to increasing levels of traffic.

In terms of reducing mean delay, both systems performed well at low traffic rates. From 500 to 1500vhl both systems kept mean delay below 2 seconds. AIM performed slightly better, hitting a mean delay of 0.97s with a standard deviation of 1.35s, QMM achieved a mean delay of 1.84s

with a standard deviation of 2.22s. As the traffic rates increased however, the performance of QMM degraded massively in comparison to AIM. At 2500vhl QMM hit a mean delay of 45.78s with a standard deviation of 18.13s. In comparison, AIM hit a mean delay of 5.43s and a standard deviation of 6.25s. Even with a comparatively high standard deviation like this, it's clear that AIM outperforms QMM in this respect. Figure 6.1 shows how the average delay increases over time for both systems.

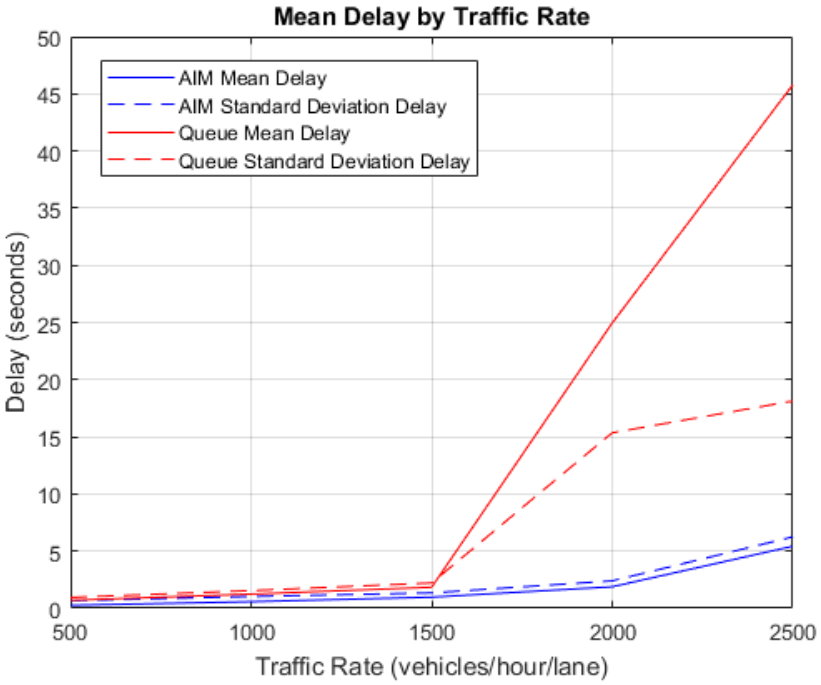

Figure 6.1: QMM and AIM mean dealy by merge management system.

In terms of balancing mean delay over both lanes, QMM also fails to perform as well as AIM. At 500vhl the mean target lane delay for QMM is 0.06s, with a standard deviation of 0.25s. For the merge lane, the delay is 1.36s, with a standard deviation of 0.95s. Comparatively AIM has a mean delay of 0.37s, with a standard deviation of 0.79s, for the target lane, and 0.16s, with a standard deviation of 0.47s, for the merge lane.

At higher traffic rates, the gap between the AIM lanes increases. At 2500vhl AIM has a mean delay of 1.75s, with a standard deviation of 1.54s, for the target lane, and 9.11s, with a standard deviation of 6.79s,

for the merge lane. These times are still far better than the mean delays in QMM. Figure 6.2 shows the delay for each lane under each system.

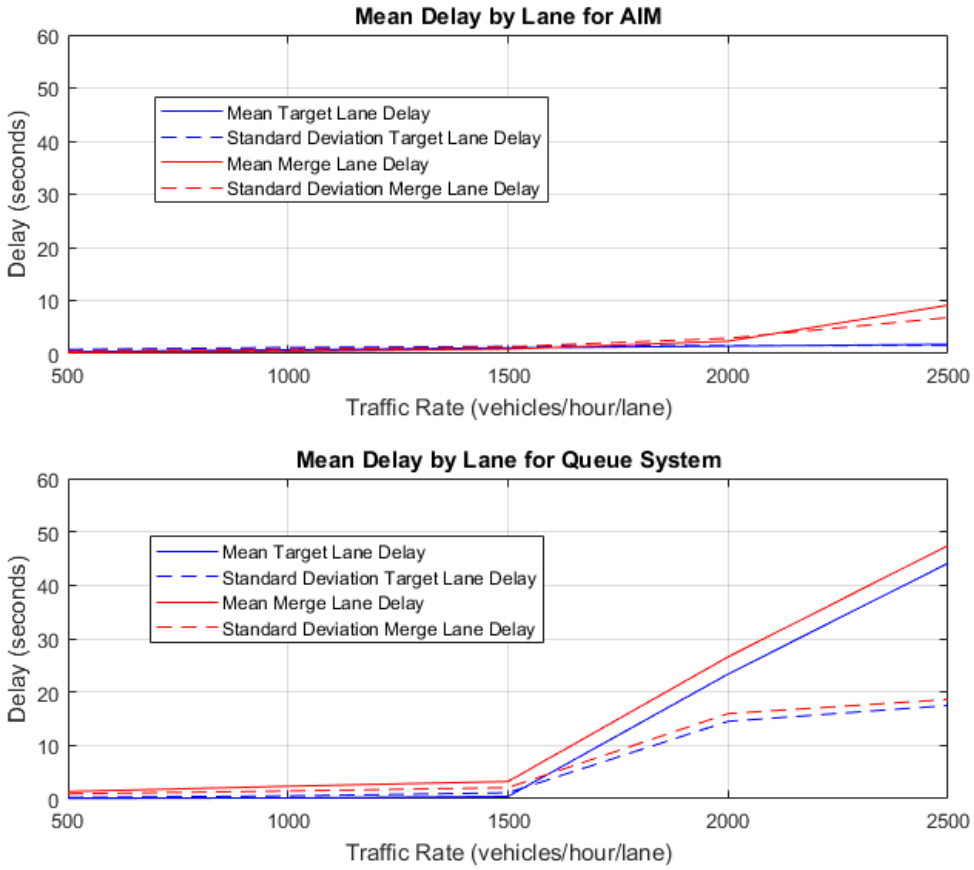

Figure 6.2: Mean delay by lane for QMM and AIM

Both systems manage to maintain similar throughputs until the traffic rate increases past 1500vhl. By 2500vhl, AIM can deal with an extra 366 vehicles per hour compared to QMM. Figure 6.3 shows the throughputs for each system.

These trends clearly demonstrate that AIM performs far more effectively at high traffic rates than QMM. AIM makes better use of space-time, so when the roads begin getting more congested, this pays off in AIMs favour. QMM ends up causing problems at these high traffic rates as

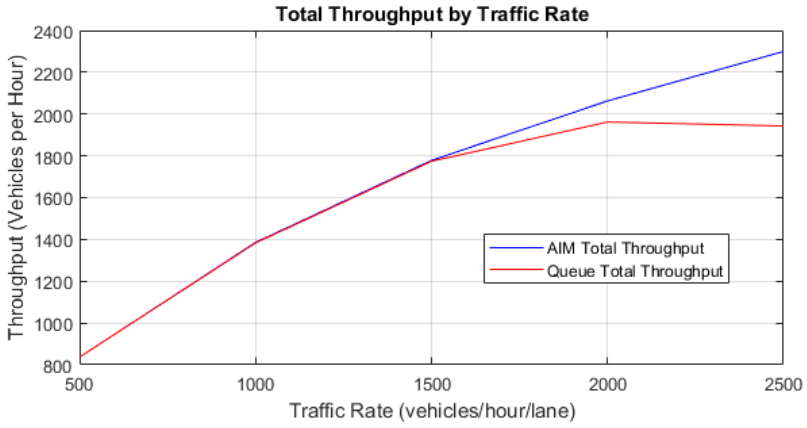

Figure 6.3: Throughputs for QMM and AIM

it only allows one vehicle into the merge zone at any one time. QMM would work well for controlling roads with lower traffic rates, however, if the roads are expected to deal with congestion or high volume fast flowing traffic, the AIM system becomes the only viable option.

This test makes a good case for continuing attempts to develop an AMM system. This system would need to be able to work for other merge angles, as no research was conducted into the effectiveness of AIM at shallower angles. Further development would also need to done using a simulator which deals with some of the implementation problems found in the AIM simulator.

6.3 The Effect of the Merge Angle

The merge angle affects both the length of the merge zone and the angle at which vehicles join the target lane. For QMM to be effective, it should be able to deal with a range of merge angles.

For each test, the speed limit for both lanes was 20ms^{-1} (44.7mph or 72kph) and each lane had a lead in distance of 150m. The traffic rate was set to 1000vhl.

At shallow angles QMM performed very poorly. Figure 6.4 shows how the shallow angle performance compares to steeper angles. At the shallowest angle tested, 5°, the system had a mean delay of 138.98s with a standard deviation of 46.96s. After the merge angle increases to around 30°, performance improves greatly with the mean delay reaching 13.12s

with a standard deviation of 9.35s. By the time the merge zone has reached it's minimum length, the mean delay has decreased to 1.24s with a standard deviation of 1.55s. The delay is actually smaller at 85°at 1.23s but this is well within range of the standard deviation for 90°. The delay of both the target lane and merge lane follow very similar trends.

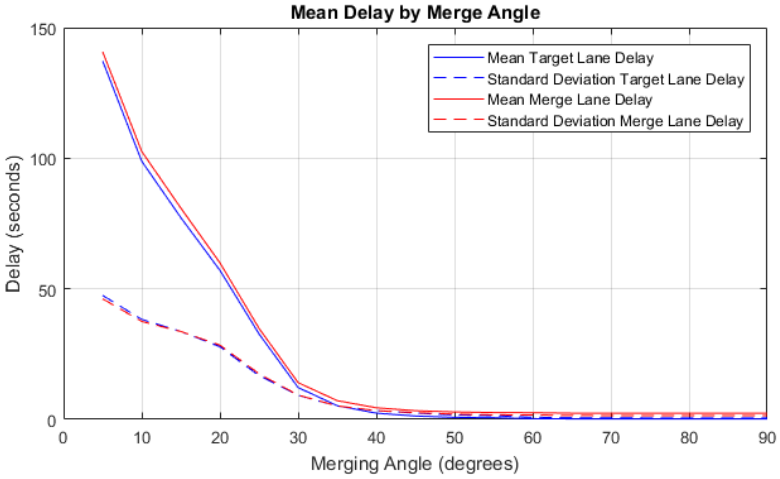

Figure 6.4: Mean delay by merge angle

In general throughput increases dramatically with the merge angle until around 35°where it levels off dramatically. The data does show a dip in the merge lane throughput measurements from 40°to 60°. Investigation showed that this was an error with vehicle spawn system which dropped vehicles after detecting that there was still another vehicle in the spawn area. These were false detections due to the change in angle. A solution to the problem would have required separate spawn schedules for each angle which reduces the comparability of the results. These results are considered to be outliers. Figure 6.5 shows how the throughput improves as the angle increases.

Shallow merge angles have such a drastic effect on performance because as the merge angle becomes shallower the merge zone gets longer. This means that the space time inefficiency of QMM has a huge effect on performance as vehicles have to wait for longer before they can enter the merge. One approach for dealing with shallow merges is to introduce a slip road instead of a merge zone. This would require a different system, as queuing wouldn't be appropriate in that situation.

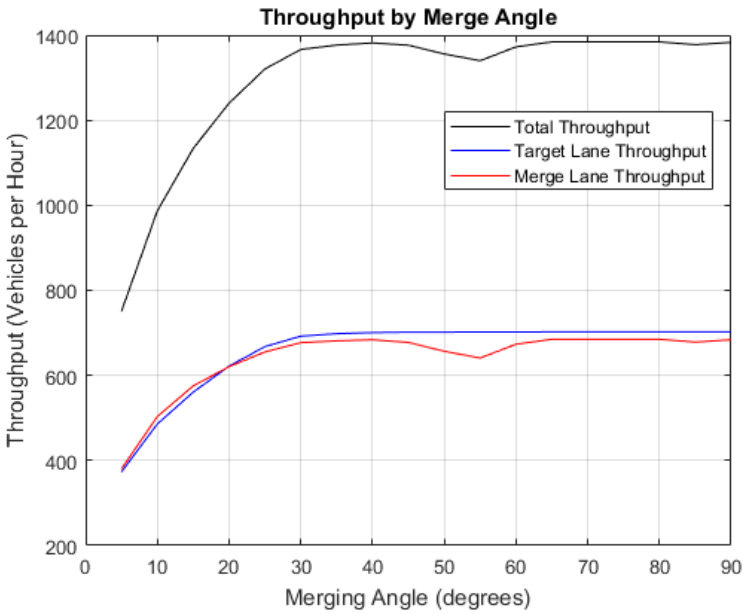

Figure 6.5: Throughput by merge angle

6.4 The Effect of Lead in Distances

The lead in distance had very little effect on the overall success of the simulation. Below 100m there was some increase in delay due to the 150m distance limit before vehicles are added to the queue. Appendix A.4 discusses these results in more detail.

6.5 The Effect of Differing Speed Limits

The speed limits of each lane and their differences impact how well a vehicle can move into another lane and adjust to its velocity. Many merges will be movements from lanes of differing speed, so its important that QMM can handles these merges.

The traffic rate was set to 1000vhl, the merge angle was 45°, and the lead in distances were both set to 150m. Pairs of speeds were compared. The speed limits used were $10ms^{-1}$, $20ms^{-1}$, $30ms^{-1}$, and $40ms^{-1}$ (22.4mph, 44.7mph, 67.1mph, and 89.5mph or 36kph, 72kph, 108kph, and 144kph respectively). These speeds, excluding $40ms^{-1}$, are very close to realistic

speed limits and should provide insight into the real world applicability of the system.

The target lane suffers the most interference from the merge lane whenever the target lane's speed limit is larger than the merge lane's speed limit. This can be seen well in the relationship between a target lane speed limit of $30ms^{-1}$ and a merge lane speed limit of $10ms^{-1}$. The mean delay on the target lane is 13.92s with a standard deviation of 3.56s. Comparatively the merge lane had a mean delay of 4.01s with a standard deviation of 2.87s.

Switching the lane speeds shows that the merge lane also experiences larger delays when the merge lane speed limit is larger than the target lane. When the target lane has a speed limit of $10ms^{-1}$ and the merge lane has a speed limit of $30ms^{-1}$, the merge lane has a mean delay of 12.00s with a standard deviation of 3.00s. In this case the target lane has a mean delay of 0.34s with a standard deviation of 0.95s.

It should be noted that when the speed gaps are smaller the effect on the lanes is reduced. Most of the speed limits containing a $40ms^{-1}$ lane also performed poorly. The velocity causes vehicles to accumulate too quickly for QMM to be able to sort them out effectively. Figure 6.6 shows how the mean delay is affected by different speed limit pairs.

The number of vehicles that complete for each lane is solely affected by the speed limit of the lane, the difference between the lane speed limits has very little effect. The throughput is likewise only affected by the speed limits of each lane independently.

These results show that QMM struggles to handle differences in velocity well, with the faster lane being impacted more heavily than the slower lane. This is because the faster lane is forced to slow down to accommodate the slower lane. There will always be delay on the faster lane due to this reason. The aim for a good merge system is to minimise the effect of the speed limit gap as much as possible.

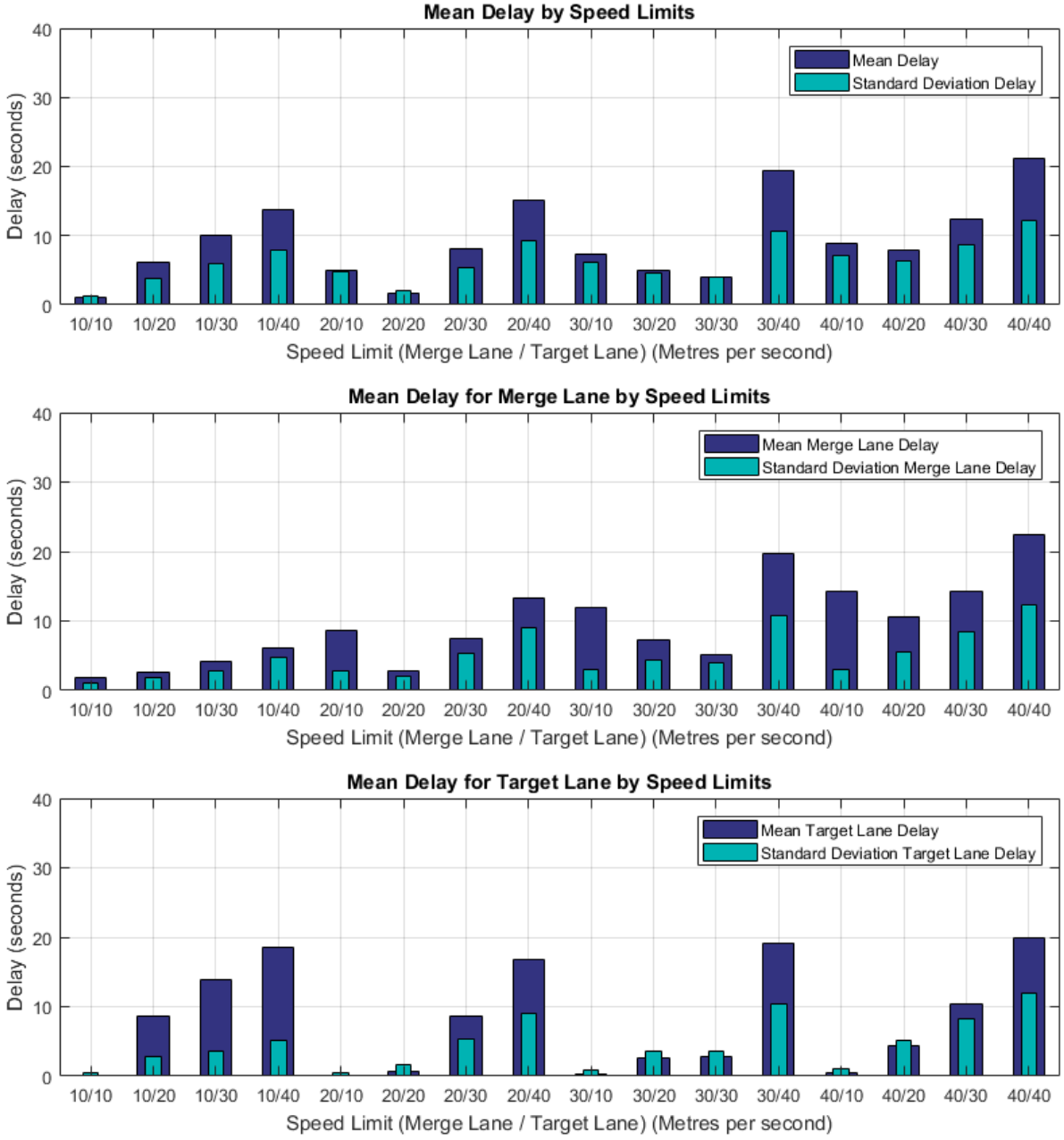

Figure 6.6: Mean delay by speed limit pair

7 Conclusion

This project had three main aims.

1. Design and simulate different AV approaches to a merge scenario, particularly surrounding centralised and decentralised approaches, and analyse their effectiveness.

2. Examine how the performance of merge management systems is affected by changing the conditions surrounding the merge.

3. Determine how well suited the AIM simulator codebase is for simulating other AV problems.

I successfully implemented a centralised approach to an S2S merge scenario using the QMM system. I also adapted the AIM system to emulate the expected behaviour of the AMM system at 90°. The results showed that the AIM system was more effective than the QMM system, particularly at high traffic rates when the QMM system failed to process vehicles efficiently. AIM's efficient use of space-time makes it much better suited for such high volume scenarios. This project has helped to show that developing an AMM system fully is worth investing research time into. A decentralised approach to the merge scenario could not be implemented due to time constraints, but a design based on the work of [20] was defined in Section 4.3.

The QMM system also tested under various merge conditions. The merge angle was found to have a very significant effect on the delay and throughput of the QMM system. At shallow angles the system performed extremely poorly to the large merge zone length. The differences in speed limit also had a large effect, impacting the faster lane's delay time quite heavily. The lead in distances had an almost negligible effect beyond very short distances. Even though the QMM system may not be developed further as a solution to the S2S merge problem, it did help to identify some of the issues that AV merge approaches will need to resolve.

Development with the AIM simulator codebase proved to be more difficult that initially anticipated. Implementation originally seemed to

be straightforward after breaking out the project into generalised and specific classes. The system had already provided a number of useful functions for driver and vehicle agents. However, as some of these proved to be ineffectual or not applicable to my project, many of them had to be rewritten or adapted. The ineffective collision prevention AIM provided and the assumption of 90°roads caused numerous issues making it difficult to develop simulators as rapidly as originally anticipated. My recommendation would be to either strip back the AIM system and reimplement the core methods, with a focus on creating a more general system, or alternatively, create a new system aimed at performing as a universal core for AV simulations.

Overall I feel that the research I've conducted should act as a starting point for further AV simulations. The next stage of development would be to see how effective a fully implemented AMM system would be at dealing with different merge angles, and with different speed limits on each lane. Once implemented, the AMM system could be adapted to other merge scenarios from Section 3.1, such as the S2D merge.

The Decentralised Merge Management system designed in Section 4.3 should also be developed into a fully working simulation and compared to the AMM system. The lack of infrastructure costs make decentralised solutions to the merge problem very desirable.

Other merge systems could also be implemented, helping to eliminate some of the foibles of the AIM, QMM and Decentralised systems. Systems developed with smoother braking profiles could help improve fuel efficiency and passenger comfort, and a system that deals with slip roads will have to be developed if road vehicles are ever going to become completely autonomous.

There are multiple areas of research to be investigated surrounding AV merging, not to mention the countless research possibilities in the AV field as a whole. My hope is that this project has helped to identify some of the key areas of development required to produce an effective merging system. If vehicles are to move to a fully autonomous future, then all of these research areas will need to be investigated to make sure that we are developing safe, efficient and effective solutions.

Bibliography

[1] (2016, October) General rules, techniques and advice for all drivers and riders (103 to 158). Website. Her Majesty's Government. Rule 126. [Online]. Available: https://www.gov.uk/guidance/the-highway-code/general-rules-techniques-and-advice-for-all-drivers-and-riders-103-to-158

[2] I. Asimov, "Sally," *Fantastic*, vol. 2, no. 3, May 1953.

[3] G. A. Larson, "Knight rider," Television, Sep. 1982.

[4] (2016, Dec.) Waymo. Alphabet. [Online]. Available: https://waymo.com/ontheroad/

[5] (2017) Press kit | tesla. Website. Tesla Motors. [Online]. Available: https://www.tesla.com/presskit/autopilot

[6] (2016, August) Tesla car drives owner to hospital after he suffers pulmonary embolism. Website. BBC. [Online]. Available: http://www.bbc.co.uk/newsbeat/article/37009696/tesla-car-drives-owner-to-hospital-after-he-suffers-pulmonary-embolism

[7] D. Lee. (2016, June) Us opens investigation into tesla after fatal crash. Website. BBC. [Online]. Available: http://www.bbc.co.uk/news/technology-36680043

[8] A. C. Mersky and C. Samaras, "Fuel economy testing of autonomous vehicles," *Transportation Research Part C: Emerging Technologies*, vol. 65, pp. 31–48, apr 2016. [Online]. Available: http://dx.doi.org/10.1016/j.trc.2016.01.001

[9] T. U. Nations, "The paris agreement," Oct. 2016. [Online]. Available: http://unfccc.int/files/essential_background/convention/application/pdf/english_paris_agreement.pdf

[10] T. L. J. B. David Schrank, Bill Eisele. (2015, August) 2015 urban mobility scorecard. Website. Texas A&M Transportation Institute &

INRIX. [Online]. Available: http://d2dtl5nnlpfror.cloudfront.net/tti.tamu.edu/documents/mobility-scorecard-2015-wappx.pdf

[11] C. F. Daganzo, "The cell transmission model: A dynamic representation of highway traffic consistent with the hydrodynamic theory," *Transportation Research Part B: Methodological*, vol. 28, no. 4, pp. 269–287, aug 1994. [Online]. Available: http://dx.doi.org/10.1016/0191-2615(94)90002-7

[12] (2016, September) Tesla criticised over autopilot safety. Website. BBC. [Online]. Available: http://www.bbc.co.uk/news/technology-37372933

[13] (2016, October) Aim homepage. Website. The University of Texas. [Online]. Available: http://www.cs.utexas.edu/~aim/

[14] M. Quinlan, T.-C. Au, J. Zhu, N. Stiurca, and P. Stone, "Bringing simulation to life: A mixed reality autonomous intersection," in *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*. Institute of Electrical and Electronics Engineers (IEEE), oct 2010. [Online]. Available: http://dx.doi.org/10.1109/iros.2010.5651993

[15] P. Gipps, "A behavioural car-following model for computer simulation," *Transportation Research Part B: Methodological*, vol. 15, no. 2, pp. 105–111, apr 1981. [Online]. Available: http://dx.doi.org/10.1016/0191-2615(81)90037-0

[16] M. Treiber, A. Hennecke, and D. Helbing, "Congested traffic states in empirical observations and microscopic simulations," *Physical Review E*, vol. 62, no. 2, pp. 1805–1824, aug 2000. [Online]. Available: http://dx.doi.org/10.1103/physreve.62.1805

[17] D. H. Arne Kesting, Martin Treiber, "General lane-changing model mobil for car-following models," *Transportation Research Record: Journal of the Transportation Research Board*, vol. 1999, pp. 86–94, 2007.

[18] M. Kamali, L. A. Dennis, O. McAree, M. Fisher, and S. M. Veres, "Formal verification of autonomous vehicle platooning," *CoRR*, vol. abs/1602.01718, 2016. [Online]. Available: http://arxiv.org/abs/1602.01718

[19] P. S. Kurt Dresner, "Multiagent traffic management: A reservation-based intersection control mechanism," in *Third International Joint*

Conference on Autonomous Agents and Multiagent Systems (AAMAS 04), vol. 2, Jul. 2004, pp. 530–537.

[20] M. VanMiddlesworth, K. Dresner, and P. Stone, "Replacing the stop sign: Unmanaged intersection control for autonomous vehicles," in *AAMAS Workshop on Agents in Traffic and Transportation*, Estoril, Portugal, May 2008, pp. 94–101. [Online]. Available: "http://www.cs.utexas.edu/users/ai-lab/?ATT08-vanmiddlesworth"

[21] M. Atagoziyev, K. W. Schmidt, and E. G. Schmidt, "Lane change scheduling for autonomous vehicles," *IFAC-PapersOnLine*, vol. 49, no. 3, pp. 61–66, 2016. [Online]. Available: http://dx.doi.org/10.1016/j.ifacol.2016.07.011

[22] R. Milligan, "A self-organising approach to autonomous vehicle car park management using a message-based protocol," Apr. 2017.

[23] C. Hewitt and R. Milligan. (2017, Apr.) Callumhewitt/avsimulatorproject. Website. [Online]. Available: https://github.com/CallumHewitt/AVSimulatorProject

[24] S. Faber, B. Dutheil, R. Winterhalter, T. van der Lippe, M. Grzejszczak, and M. ZajÄĚczkowski. (2017, Mar.) Mockito framework site. Website. [Online]. Available: http://site.mockito.org/

A Appendix

A.1 Requirements

A.1.1 Functional Requirements

Table A.1: Functional requirements table.

User Requirements		System Requirements	
FU.1	Users can run merge simulations.	FS.11	The system has controls for starting merge simulations
		FS.12	The system can create merge simulations
		FS.13	The system can run merge simulations
FU.2	User can manipulate the running speed of a simulation.	FS.21	The system has controls changing the speed of running simulations.
		FS.22	Merge simulations can have their run speed changed.
FU.3	User can pause simulations.	FS.31	The system has controls to pause simulations.
		FS.32	Merge simulations can be paused
FU.4	User can end a simulation.	FS.41	The system has controls to end simulations.
		FS.42	Merge simulations can be ended.
FU.5	User can view the activities of a simulation.	FS.51	The system displays the current status of a simulation as it runs.
		FS.61	The system has controls for exporting results data.
FU.6	Users can export the results of a merge simulation.		

User Requirements		System Requirements	
		FS.62	The system can produce a file containing results data from the simulation.
		FS.63	The results file contains the total throughput of the simulation.
		FS.64	The results file contains the ML throughput of the simulation.
		FS.65	The results file contains the TL throughput of the simulation.
		FS.66	The results file contains the maximum delay of the simulation.
		FS.67	The results file contains the average delay of the simulation.
		FS.68	The results file contains the ML average delay of the simulation.
		FS.69	The results file contains the TL average delay of the simulation.
		FS.6a	The results file contains the maximum acceleration of the simulation.
		FS.6b	The results file contains the maximum deceleration of the simulation.
		FS.6c	The results file contains the delay for each vehicle in the simulation.
		FS.6d	The results file contains the maximum acceleration for each vehicle in the simulation.

User Requirements		System Requirements	
		FS.6e	The results file contains the maximum deceleration for each vehicle in the simulation.
FU.7	Users can select and run an S2S merge simulation.	FS.71	The system can produce S2S simulations.
		FS.72	The system has controls allowing S2S simulations to be selected.
		FS.73	The system can run S2S simulations
FU.8	Users can select a centralised merging scheme with S2S merge simulations.	FS.81	The system can use an AIM-like merge management system for the merging zone in an S2S simulation.
		FS.82	The system has controls allowing users to select the merge scheme indicated in FS.81.
FU.9	Users can select a decentralised merging scheme with S2S merge simulations.	FS.91	The system can use an merge management scheme similar to that described in "Replacing the Stop Sign: Unmanaged Intersection Control for Autonomous Vehicles" [20].
		FS.92	The system has controls allowing users to select the merge scheme indicated in FS.91.
FU.10	Users can control the traffic level of an S2S simulation.	FS.101	The system has controls for adjusting the traffic for an S2S simulation.
FU.11	Users can control the TL speed limit of an S2S simulation.	FS.111	The system has controls for adjusting the TL speed limit for an S2S simulation.
FU.12	Users can control the ML speed limit of an S2S simulation.	FS.121	The system has controls for adjusting the ML speed limit for an S2S simulation.

User Requirements		System Requirements	
FU.13	Users can control the TL lead in distance of an S2S simulation.	FS.131	The system has controls for adjusting the TL lead in distance for an S2S simulation.
FU.14	Users can control the ML lead in distance of an S2S simulation.	FS.141	The system has controls for adjusting the ML lead in distance for an S2S simulation.
FU.15	Users can control the merge angle of an S2S simulation.	FS.151	The system has controls for adjusting the merge angle for an S2S simulation.
FU.16	Users should be alerted of any collisions during a simulation.	FS.161	The system should be able to alert the user if a collision occurs.
		FS.162	The simulation should detect collisions.

A.1.2 Non-functional Requirements

Table A.2: Non-functional requirements table.

ID	Description
NS.1	All system controls come from standard JComponents.
NS.2	All controls are clearly labeled.
NS.3	Simulation creation should take no longer than 3 seconds.
NS.4	Simulations should be able to run faster than real-time.
NS.5	Simulations should update the GUI frequently.
NS.6	All data displayed to the user should be accurate.
NS.7	All data in the results file should be accurate.
NS.8	Created simulators should accurately represent their merge scenario with the provided modifier factors.
NS.9	Code should be written to allow for easy expansion.
NS.10	Code should be well documented.
NS.11	The simulator should be integrated into the AIM4 simulator without negatively affecting the performance of AIM simulations.
NS.12	It should be easy to add new simulator types in AIM4 alongside AIM and Merge simulations.

A.2 S2S Map Calculations

To start with, we need to calculate the dimensions of the merging zone. The height of the merging zone will be the same as lane width of the target lane. The length can be calculated using the right-angled triangle in Figure A.1. Using this triangle and some trigonometry we can calculate the length of the merge zone (mergingZoneLength in Fig. A.1) using equation A.1.

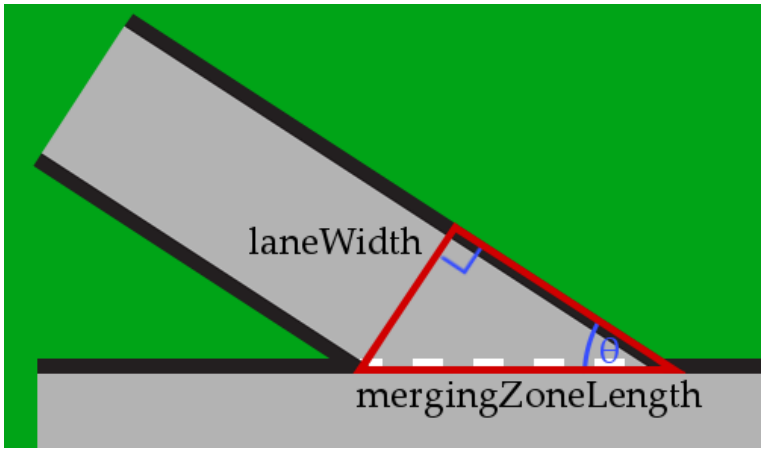

Figure A.1: A right-angled triangle for the merging zone.

$$mergeZoneLength = \frac{laneWidth}{\sin(\theta)} \tag{A.1}$$

We also need to know whether the horizontal width of the merge lane on the map, or it's 'base width' is longer than the target lane's lead in distance, plus the merge zone length. This will determine the width of the overall map, as if the merge lane's base length is longer then the target lane will not start with co-ordinate $x = 0$ as it would if the target lane determined the width of the map.

Firstly we need to calculate the X and Y adjustments at the merge lane entrance. Because the vehicles drive in the centre of the lane and the merge lead in distance is defined by the middle line of the lane we still need to calculate how far the lane extends in the x and y directions due to it's width. To do this we can use the right-angled triangles shown in Figure A.2

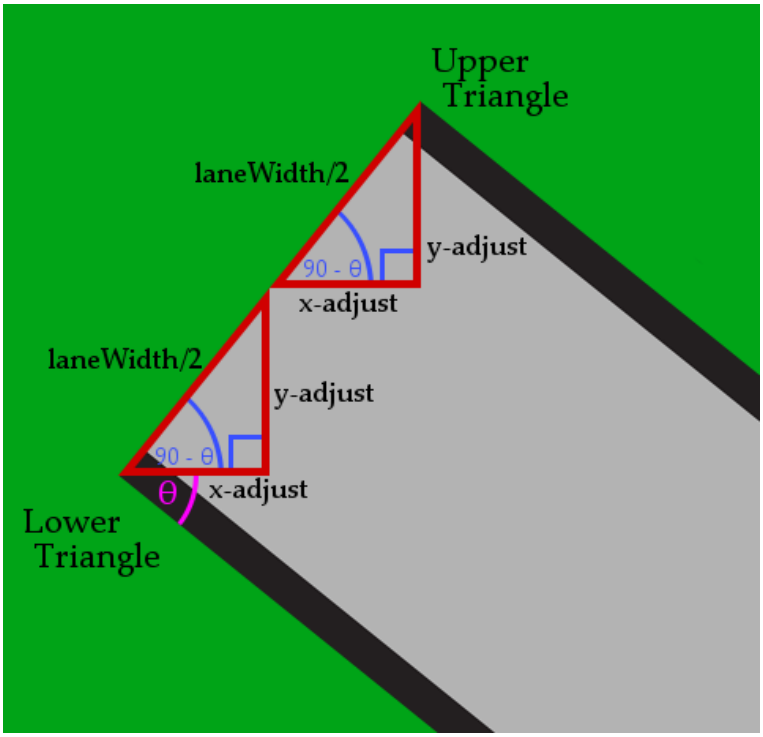

Figure A.2: Two right-angled triangles for the x and y adjustments.

These triangles have the same dimensions and have an interior angle of 90 - θ due to the 'alternate angle' or 'z-angle' rule. Each triangle has a hypotenuse with a length equal to half the width of the lane.

The X-adjustment for the merge entrance is the length of the adjacent side of one of the lower triangle and the Y-adjustment for the merge entrance is the length of the opposite side of the upper triangle (though both triangles do have the same dimensions). We can use equation A.2 to calculate the X-adjustment and equation A.3 to calculate the Y-adjustment.

$$\text{x-adjust} = \frac{laneWidth}{2}\cos(90-\theta) \tag{A.2}$$

$$\text{y-adjust} = \frac{laneWidth}{2}\sin(90-\theta) \tag{A.3}$$

To calculate the 'base width' of the merge lane we will also need to calculate the adjacent side of the triangle in Figure A.3. In this triangle the hypotenuse has a length equal to the merge lead in distance. Therefore,

we can use equation A.4 to calculate the length of the adjacent side. After obtaining the length of this side we simply add the merge entrance X-adjustment and half the length of the merge zone to find the merge base width.

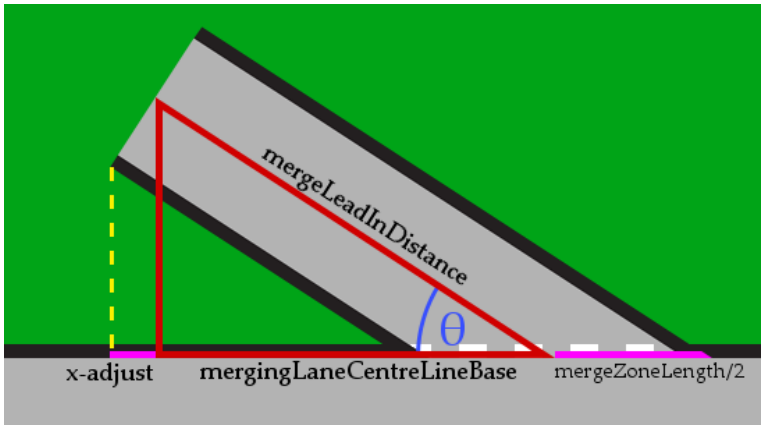

Figure A.3: A right-angled triangle for the the base width of the merge lane, with the X-adjustment and merge-zone length.

$$mergingLaneCentreLineBase = mergeLeadInDistance \cos(\theta) \quad \text{(A.4)}$$

We also need to find the point at which the merging lane's centre line crosses the target lane's centre line in the merge zone. We know the Y-coordinate for this point as it will be the same as the Y-coordinate of the target lane centre line. We also know the X-coordinate of the point at which the merge lane's centre line meets the target lane. We can use these two co-ordinates to create the triangle shown in Figure A.4. We can then use equation A.5 to find the X-adjustment from the merge zone centre to the point where the two centre lines cross.

$$toCentreDistance = fraclaneWidth2 \tan(\theta) \quad \text{(A.5)}$$

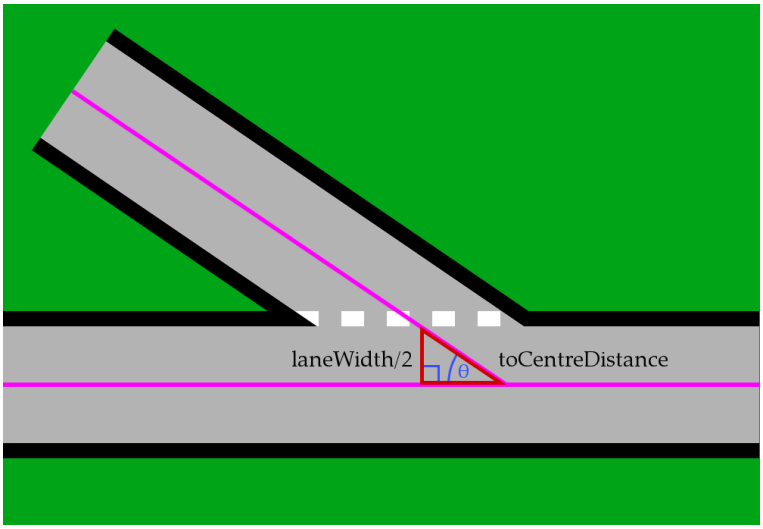

Figure A.4: A right-angled triangle for lane meeting point calculations

A.3 Generalising the Codebase

All class diagrams were created using the 'IntelliJ IDEA 15.0.3' internal diagram tool. Figure A.5 provides a key for understanding these diagrams.

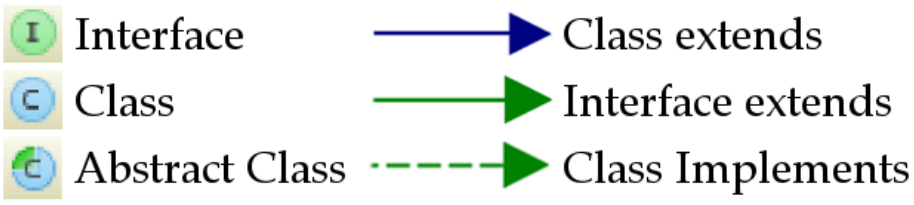

Figure A.5: Key for the class diagrams in this report.

A.3.1 aim4.driver

aim4.driver controls how a vehicle behaves on the map. In the original simulator the drivers were built to deal with 4-way intersections, with general functionality tied into the same class. You can see how this was

done in Figure A.6.

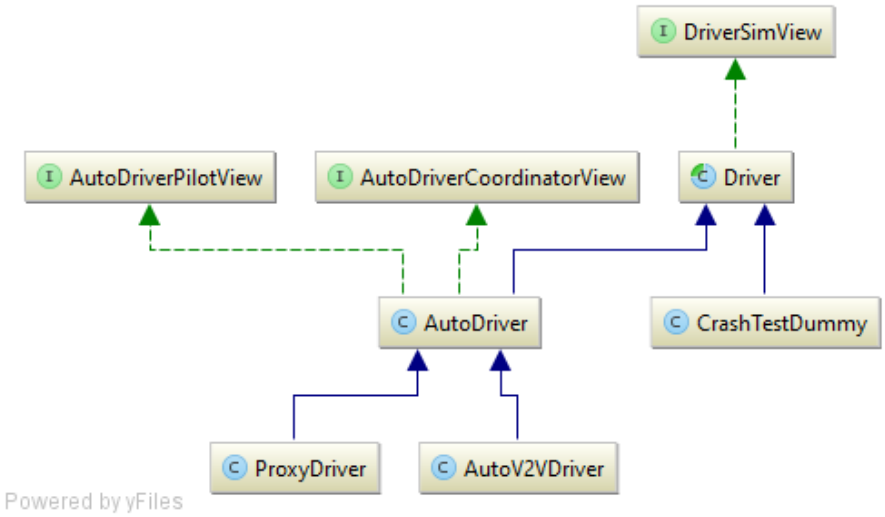

Figure A.6: The original class structure for *Driver*.

The first major change was renaming *DriverSimView*, *AutoDriverPilotView*, and *AutoDriverCoordinatorView* to end in *Model* instead of *View*. These interfaces are used to limit the methods that other classes can access in Driver and AutoDriver, thus changing their 'view' of that class. We felt that *View* could cause confusion with the GUI elements of the simulator; we instead chose to refer to these interfaces as *Models*, because the accessors are effectively given a model of Driver and AutoDriver (beyond which they care very little) that they can use to access methods.

The next change was separating out all of the AIM specific code into its own classes and interfaces. You can see how this was done in Figure A.7 with *AIMDriverSimModel* and *AIMDriver*. The merge specific code found in *MergeDriverSimModel*, *MergeDriver* and *MergeAutoDriver* is structured in a very similar manner to its AIM counterpart, taking advantage of the generalised code.

As a consequence of breaking out the code like this, a number of additional changes had to be made. Driver was changed into an interface and a new class *BasicDriver*. *Driver* is simply used as an interface for accessing Drivers in non-simulation contexts (such as *BasicVehicle*). *BasicDriver*

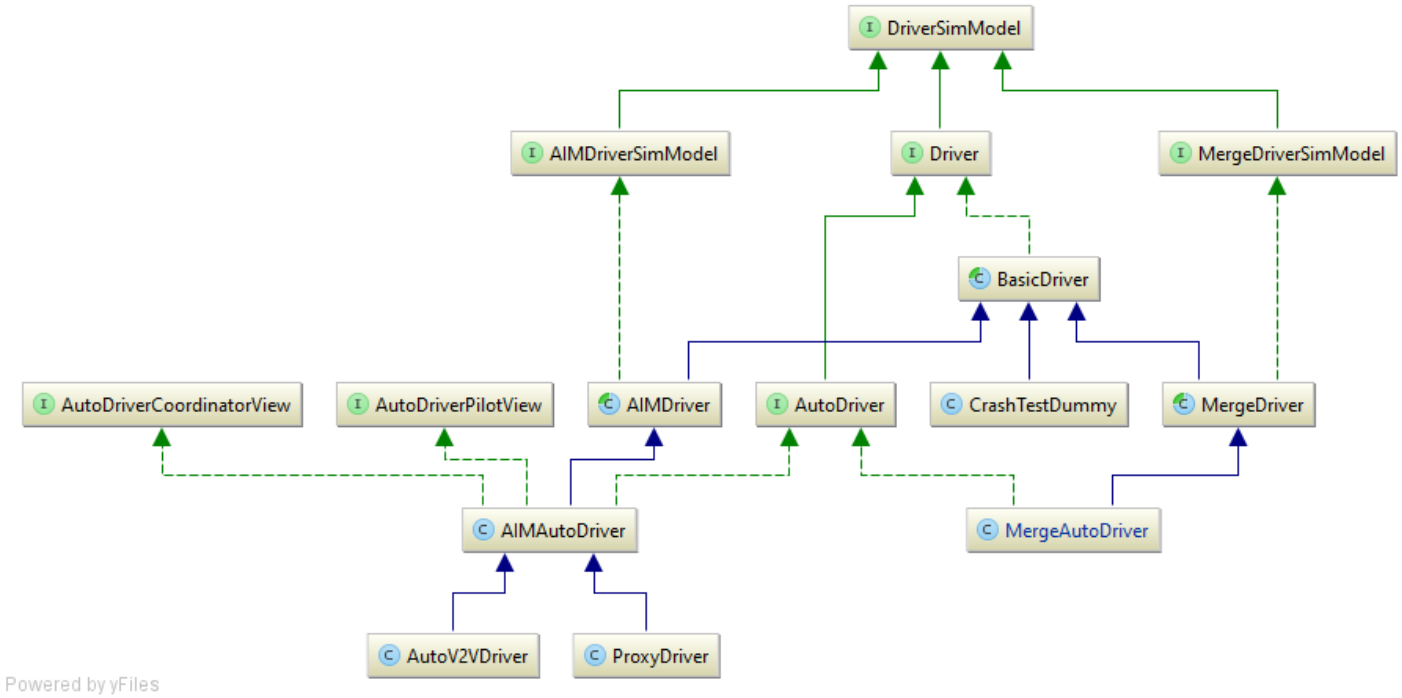

Figure A.7: The new class structure for *Driver*.

contains the generalised functionality all *Driver* objects should need, with AIM specific activities moved to *AIMDriver*. Extending from *Driver* is the *AutoDriver* interface, which adds no new methods but is instead used to categorise autonomous drivers. *AIMAutoDriver* contains almost exactly the same code as the original *AutoDriver* class.

A.3.2 aim4.gui

aim4.gui controls the GUI for the simulator. We had to adjust this to allow for non-AIM simulations to be run. We chose to use tabs to allow users to switch between simulators (these are greyed out when a simulation is running). To make adding new tabs and simulation screens easier we had to refactor *Viewer* into smaller, separate components.

In the original simulator *Viewer* displays the simulator set-up controls, *SimSetupPanel*, and the simulation viewer *Canvas* inside *mainPanel*. *mainPanel* is a *JPanel* with a *CardLayout* allowing the panel to switch between displaying the set-up controls and the viewer. In the new simulator we replaced *mainPanel* with *tabbedPane*, a *JTabbedPane* object that allows users to switch between the different simulators using tabs. Each tab displays a *SimViewer*, which behaves in a similar way to *mainPanel* allowing users to switch between the set-up screen and the simulation screen using *CardLayout*. Each simulator will have their own SimViewer type, as shown in Figure A.8.

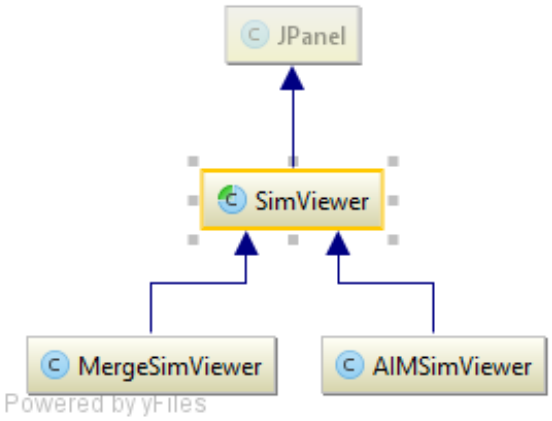

Figure A.8: The class diagram for *SimViewer*.

We didn't want to force new simulators to use a full representation of vehicles on screen, as *Canvas* does. To avoid this we created a new interface *SimScreen* which *SimViewer* uses to describe it's viewer card. Any class implementing *SimScreen* can be used as the viewer for a simulation. Figure A.9 shows how *MergeStatScreen* and *Canvas* using *SimScreen*.

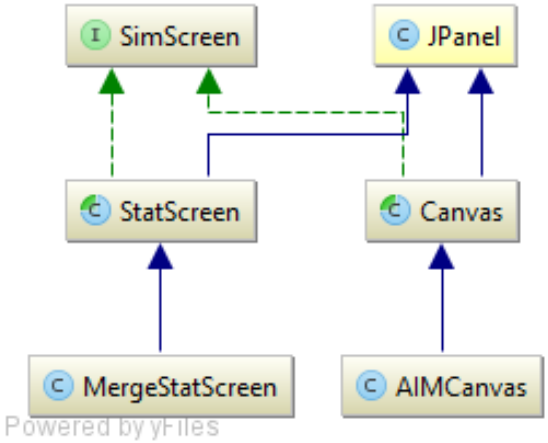

Figure A.9: The class diagram for *SimScreen*.

We also generalised the *SimSetupPanel* class to allow *SimViewer* to display non-AIM set-up controls. Figure A.10 shows the new class structure for *SimSetupPanel*.

We also made a small adjustment to the behaviour of the reset option in the menu. Now the simulator must be paused in order for the reset button to be active. We did this because resetting the simulator without pausing was creating *NullPointerException*s.

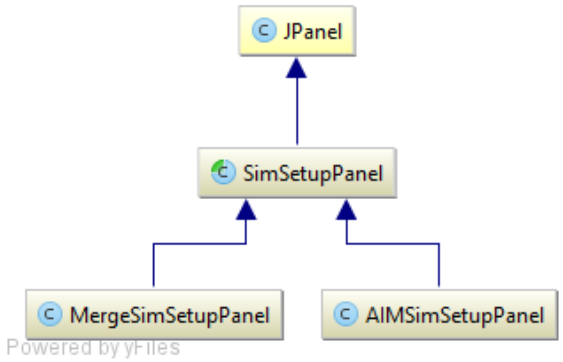

Figure A.10: The class diagram for *SimSetupPanel*.

A.3.3 aim4.map

aim4.map is used to describe the environment vehicles are required to navigate. They also spawn vehicles that then drive through the map. Figures A.11 and A.12 show the original and new class structure for *aim4.map*.

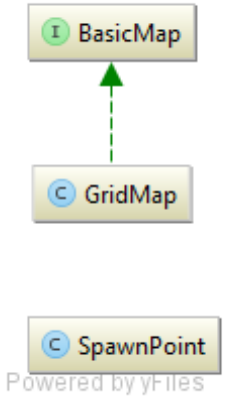

Figure A.11: The original class structure for *BasicMap* and *SpawnPoint*.

The changes made to *aim4.map* were relatively straight-forward. The AIM specific features in *BasicMap* were extracted out in *BasicIntersectionMap* and *GridMap* was renamed to *GridIntersectionMap* and now inherits

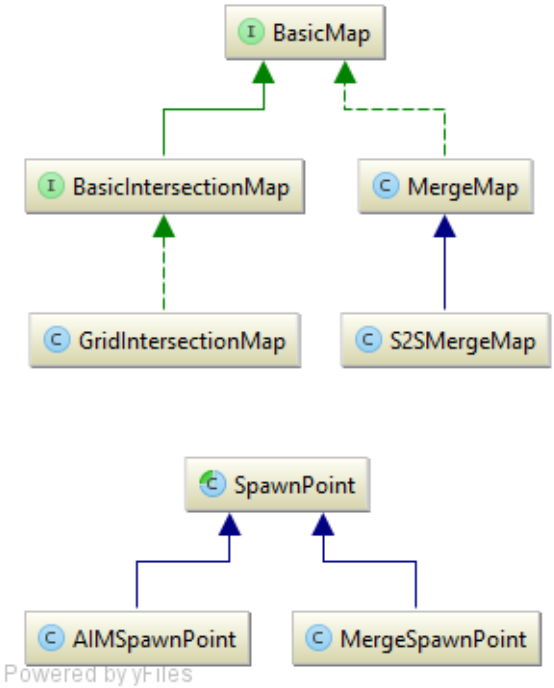

Figure A.12: The new class structure for *BasicMap* and *SpawnPoint*.

from the new interface. This allows for new map types, such as *MergeMap* to implement a map type without AIM features.

SpawnPoint was also broken out into general and AIM specific features. This had to be done because *SpawnPoint* used to create *SpawnSpec* objects with *destination* fields. *destination* is an AIM specific field relating to the intersection exit a vehicle plans to reach. By extracting this out new map types can spawn vehicles with *SpawnSpec* instances specific to their map type.

A.3.4 aim4.sim

aim4.sim contains the code responsible for constructing and running simulations. The original code was very focussed on AIM simulations and so we had to break the interfaces to allow for different types of simulators.

Simulator is an interface that new simulators need to implement. We

decided to extract out some of the AIM specific features into *AIMSimulator*. We also added an override to *getMap()*, forcing AIM simulators to use *BasicIntersectionMap* maps. The class structure changes can be seen in Figures A.13 and A.14.

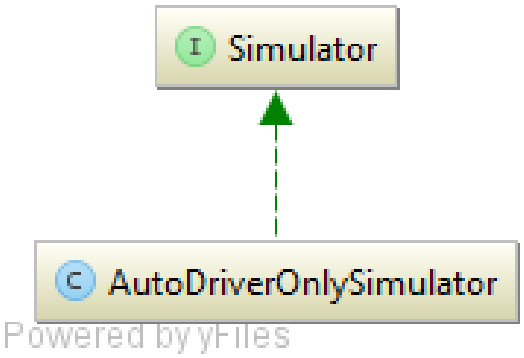

Figure A.13: The original class structure for *Simulator*.

SimSetup was also modified to separate AIM specific set-up options and simulator creation code from other simulators.

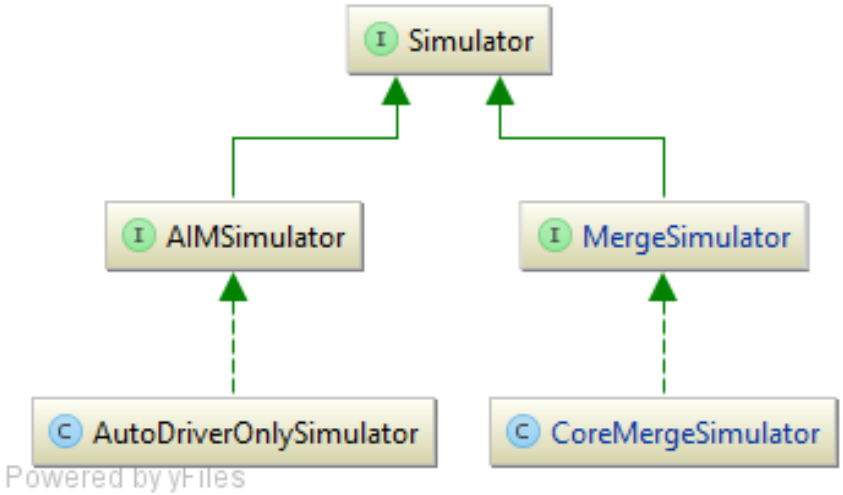

Figure A.14: The new class structure for *Simulator*.

A.3.5 aim4.vehicle

aim4.vehicle controls the different vehicles used during simulations. Vehicles are used by both *Driver* and *Simulator* instances. To allow them to do that the original simulator code used *View* interfaces similar to those in A.3.1. Figure A.15 shows how these interfaces link together. Extracting AIM behaviour was quite difficult because of how interconnected these interfaces were. The solution we came up with was to create AIM specific interfaces and link them together in a similar manner, inheriting from the generic ones if possible. Figure A.16 shows how the new structure links together.

The first change made to *aim4.vehicle* was to rename all of the files ending in *View* to end in *Model* instead. This matches the changes made to *aim4.driver*.

AIMVehicleSimModel and *AIMAutoVehicleDriverModel* are at the top of the AIM interface tree. They both extend their generic counterparts. *AIMAutoVehicleSimModel* extends these two interfaces along with *AutoVehicleSimModel*. This matches up to the original inheritance structure. Any future vehicles will need to create their own version of these interfaces, as seen in *MergeVehicleSimModel*, *MergeAutoVehicleDriverModel* and *MergeAutoVehicleSimModel*.

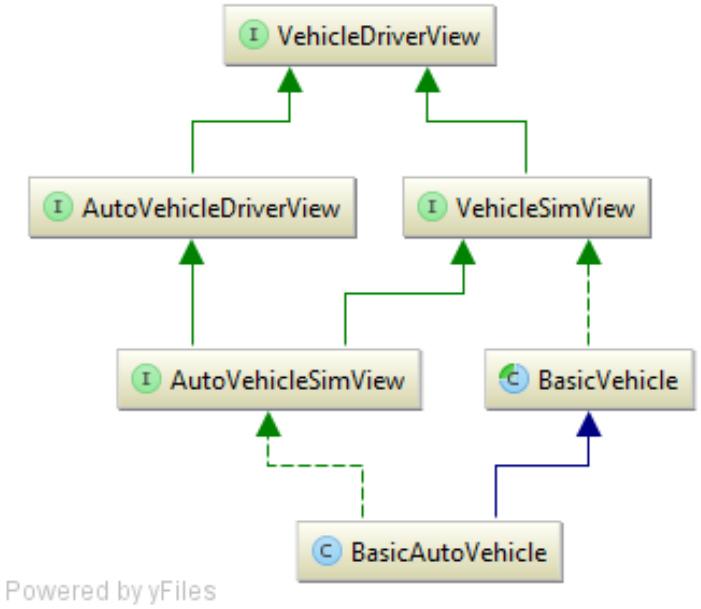

Figure A.15: The original class structure for *aim4.vehicle*.

In terms of classes we made *BasicAutoVehicle* abstract and extracted out AIM specific behaviour to *AIMBasicAutoVehicle*. *BasicAutoVehicle* had to be abstract because we wanted to force *getDriver()* to be overridden in subclasses to retrieve the simulator specific *AutoDriver* for that vehicle (for example *AIMAutoDriver* in AIM simulators).

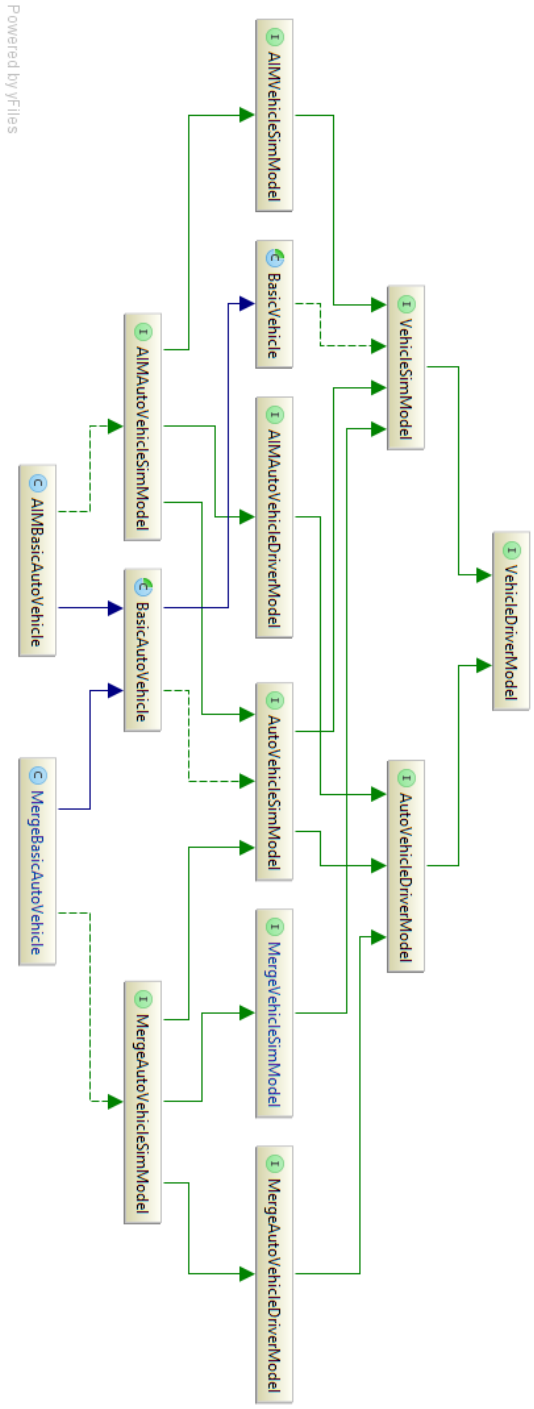

Figure A.16: The new class structure for *aim4.vehicle*.

A.4 Lead in Distance Analysis

The lead in distance adjustments affect how long vehicles have to organise and communicate with QMM. Different lead in distance pairs were tested. The traffic rate was set to 1000vhl, the merge angle was set to 45°, and the speed limit was set to 20ms^{-1} (44.7mph or 72kph).

Due to the 150m distance limit on requests, lead ins at 100m were worse for the lane with the 100m lead in. The only exceptions were when both lanes had lead in distances of 100m.

If one lane has a 100m lead, then the other lane also suffers a performance hit. This affect is most noticeable when the target lane has the 100m lead, but this effect is also seen with the merge lane.

Beyond 150m the lead in distance had very little effect on the performance of the system. Throughput remained relatively constant throughout. Appendix A.17 shows how the mean delay relates to the lead in distance.

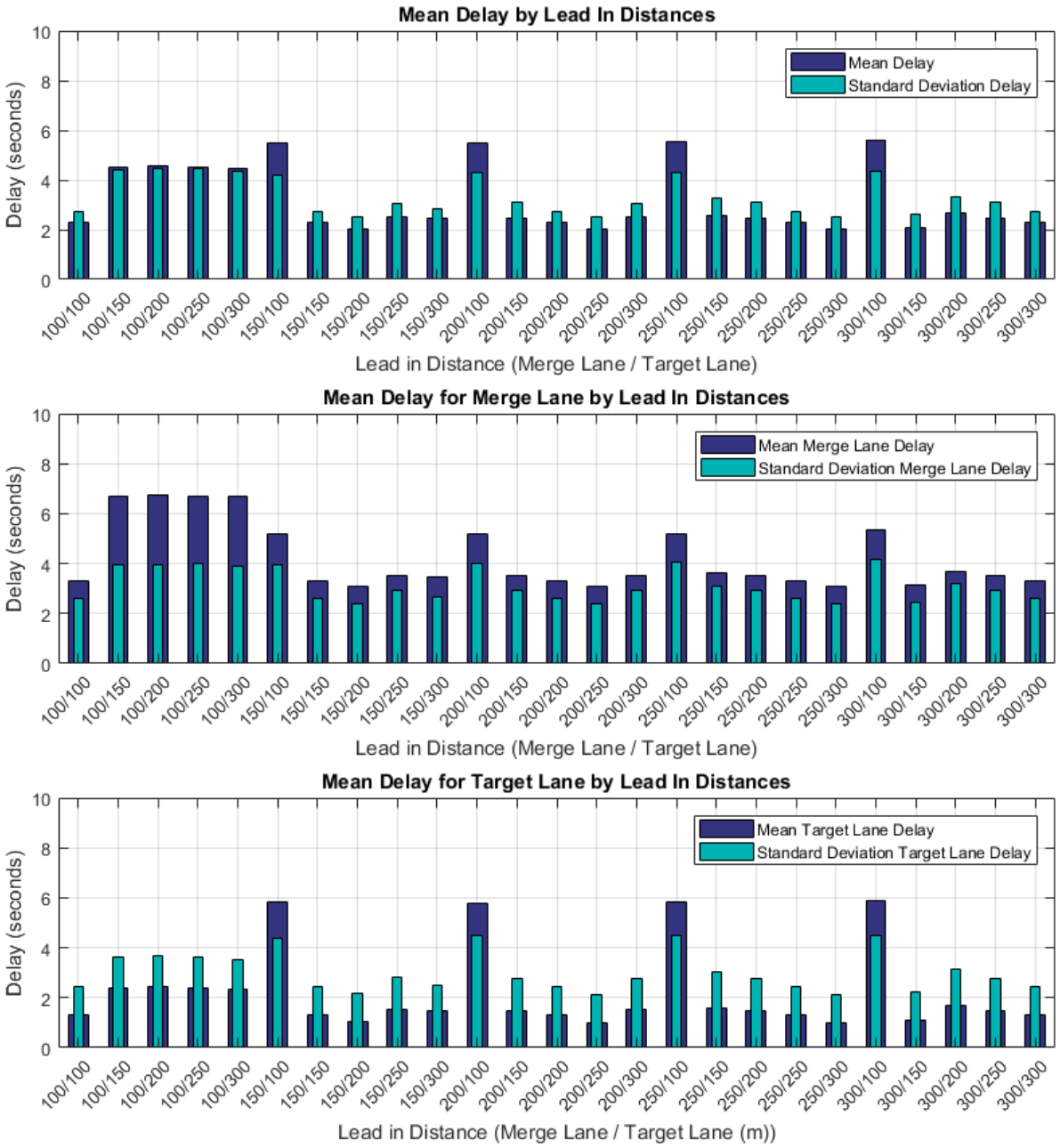

Figure A.17: Bar chart showing mean delay by lead in distance