The University of York Department of Computer Science

**Submitted in part fulfilment for the degree of BSc.**

# Approaches to Autonomous Vehicle Merge Management

Callum Hewitt

2nd May 2017

Supervisor: Lilian Blot

Number of words = 0, as counted using TeXcount.
This includes the body of the report, but not headers, 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 100m. 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.

## Acknowledgements

Thanks to my supervisor Lilian for the advice given in shaping this project.

# 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* [4] and in 1982, *Knight Rider* introduced KITT [5]. Today, autonomous vehicles can be found on roads around the world. Alphabet's WAYMO is gaining traction, with cars being tested in four different US states [6]. Tesla have deployed their beta Autopilot system into all of their vehicles produced since September 2014 [7]. The system has been both hailed for saving lives and blamed for ending them [8] [9].

All of the autonomous vehicle systems currently running on public roads are designed to work alongside human driven vehicles, limiting their benefits. In order to truly embrace autonomous vehicles, 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.

Autonomous vehicles 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 autonomous vehicle 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 autonomous vehicles is efficiency. Research suggests that by implementing fuel conserving driving strategies, autonomous vehicles could be up to 10% more fuel efficient than current EPA fuel economy test results [10]. Fuel efficient vehicles are becoming increasingly important, with landmark climate change deals such as 'The Paris Agreement' introducing limits on greenhouse gas emissions globally [11]. 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 autonomous vehicles could help bring electric vehicle range up to par.

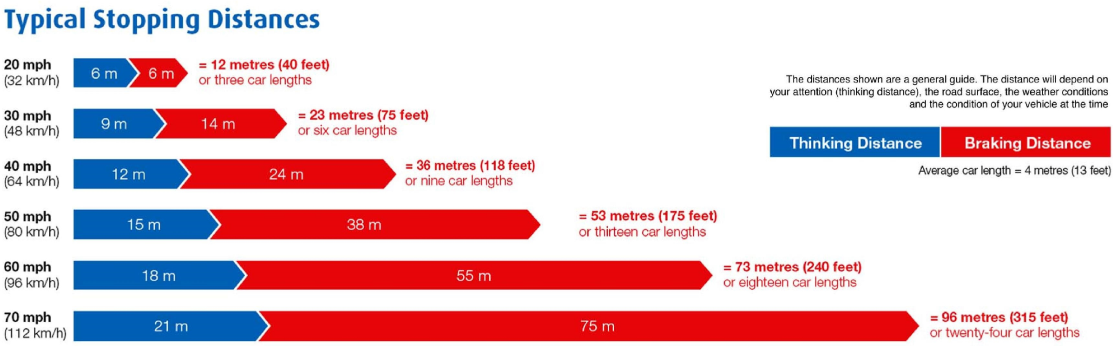



Figure 1.1: Diagram from Rule 126 in the UK Highway Code [1]

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 [12]. Automated driving strategies, in situations such as 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' [13].

As well as more quantifiable benefits, autonomous vehicles could also provide a level of comfort not currently available today. In a world where autonomous vehicles 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 autonomous vehicles, 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 [14].

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" [15]. The team managed to apply their simulator tested intersection software in a mixed reality test, using a real life autonomous vehicle [16]. 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. Autonomous vehicles 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. The focus is on how autonomous vehicles can self-organise to minimise delays in traffic, using safe and effective lane merging.

Chapter 2 examines existing work with merging autonomous vehicles 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 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. Chapter 7

# 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 [17]. 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. Appendix A.1.1 details these equations. Gipps' model worked well at describing the behaviour of traffic. However, translating this work to autonomous vehicles 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 autonomous vehicles. The gaps between successive vehicles are also larger than necessary. Autonomous vehicles are more precise than human drivers and can drive closer to their predecessors. Gipp's model also focuses solely on single-lane drivers.

In 2000 Treiber et al. suggested the 'Intelligent Driver Model' (IDM) [18]. This model, detailed further in Appendix A.1.2, 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 autonomous vehicle models, as seen in Kesting's work [19] in Section 2.3.2. However, similarly to Gipps' model, it solely focuses on single-lane drivers.

Gipps' model and the IDM also both fail to recognise, and incorporate, the use of vehicle-to-vehicle communication in their models. Autonomous vehicles 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 [20], 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 [13].

## 2.2 Centralised and Decentralised

We can divide approaches to autonomous vehicles 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 [21] 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 grid-based 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 4.2.

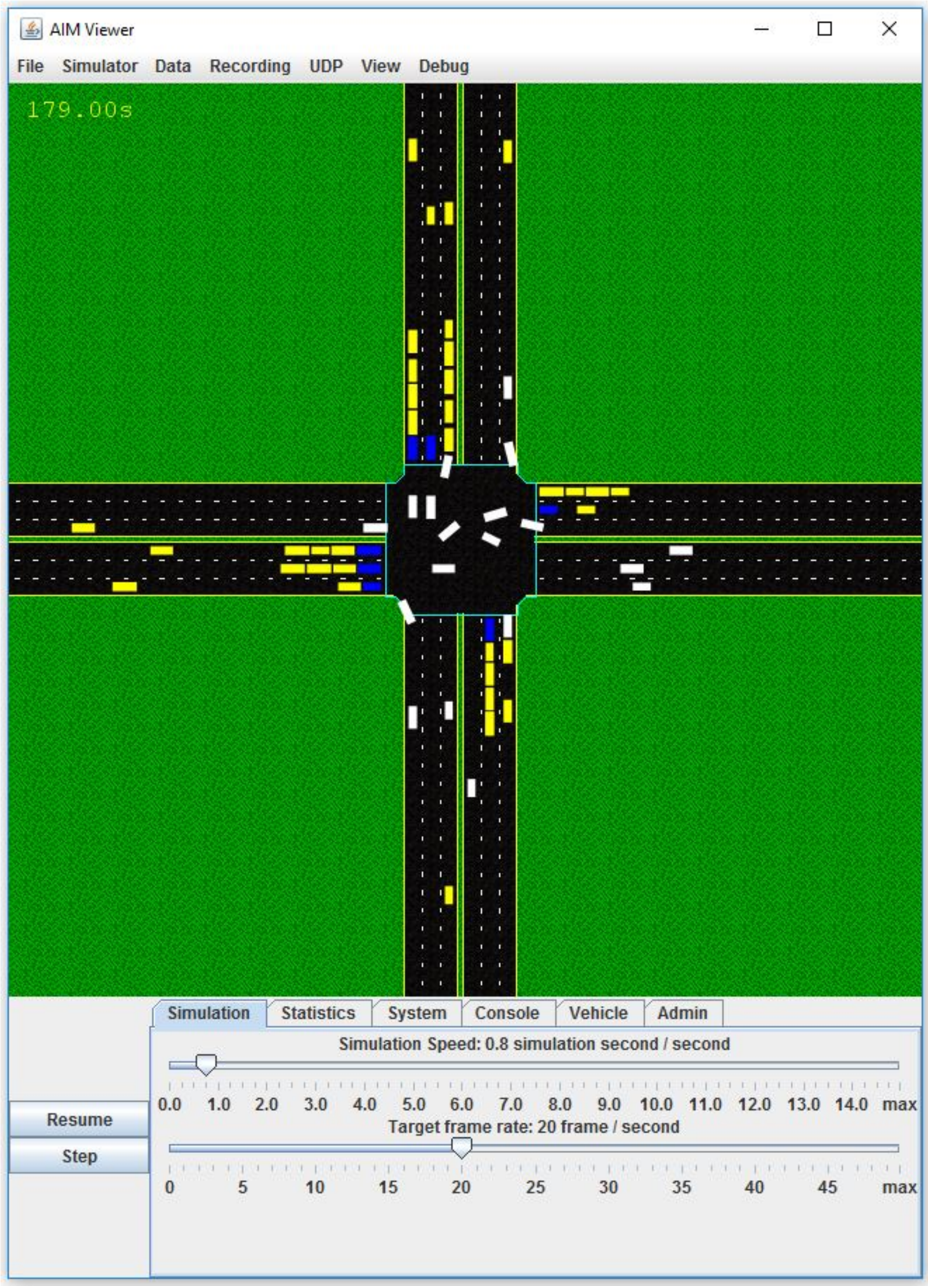



Figure 2.1: Screenshot of the AIM Simulator

### 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 [22] 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 the claim has priority dominates. Priority is indicated by the following rules, 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 who 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 its *Claim* 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.

Cost also becomes a major issue for centralised systems when you consider fast moving situations such as lane changing on a motorway. To implement Atagoziyev's model, vehicles must remain in range of the roadside infrastructure. This would mean that the infrastructure will have to continue on for a long distance, which could become very expensive, especially given the number of critical-positions on a motorway. Decentralised solutions reduce these costs massively.

Two examples of decentralised lane changing models are Gipps' 1986 driver decision model [2] and the MOBIL model developed by Kesting et al. in 2007 [19]. These models are decentralised and as such do not have to rely on roadside infrastructure in order to change lanes. This greatly reduces the cost of both implementations and allows the vehicles to be more flexible as to when they change lanes, no longer having to wait until they reach the lead up to a critical-position supported by roadside infrastructure. This flexibility means that vehicles could change lanes to increase their average velocity rather than just changing lanes in order to make a turn or leave the motorway at a critical-position. It also allows vehicles to deal with unexpected situations far from any roadside infrastructure. For example, a broken down car blocking a lane can be evaded. There is more information on Gipps' 1986 model and MOBIL in 2.3.

## 2.3 Making lane changing decisions

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 1986 Gipps' modelled driver behaviour in real world circumstances, characterising the decisions a driver has to make in order to determine whether to change lanes [2]. The paper was designed to be used with the Gipps' 1981 car-following model [17], explained in 2.1.

The model itself is constructed as a flow chart, in which the decision nodes are the choices a driver must make.You can see the flowchart in Figure 2.2.

After determining whether a lane change is feasible the model considers whether the driver needs to move into another lane because they are heading towards a critical point.

These decisions are modelled in nodes 3 and 4.

3 *Driver behaviour close to the intended turn*

If the driver is close to their intended turn then they will always attempt to change into their preferred lane. Only if blocked will they consider moving into another lane.

'Close' varies depending on regional differences and the level of traffic, but in the model, close is defined as the driver being within a distance equal to ten seconds of travel from the turn at the driver's desired speed.

4 *Urgency of changing lanes*

The urgency of changing lanes increases as the driver gets closer to their turn. The willingness of the driver to brake harder and accept smaller gaps increases as the driver gets closer to their intended turn.

In the implementation, the braking rate a driver is willing to when first becoming close doubles by the time the intended turn is reached.

$D_n$ is the location of the intended turn

$V_n$ is the desired (or free) speed of the driver

$b_n^*$ is the most severe braking the driver would otherwise be willing to undertake

$$b_n = \left[2 - V_n \frac{(D_n - x_n(t))}{10}\right] b_n^* \qquad (2.1)$$

Similarly to Gipps 1981 car-following model, this driver decision model is based on human driver behaviour and as such falls into similar pitfalls. There could be more optimal driver behaviours which would ensure that a driver is in the correct lane well before the critical position. However, because Gipps 1986 model is designed to model human driving behaviour we can expect it to perform less than optimally.

Atagoziyev's model is designed around autonomous vehicles, which allows it to take advantage of vehicle-to-vehicle communication. The model manages a set of vehicles over two lanes, organising their movements into their preferred lanes before they reach a critical position.

To do this the model keeps track of the gaps between vehicles and their relative speeds using roadside infrastructure. Then, a series of equations, each describing a different situation, are used to determine the behaviour of the vehicle.

In the paper SV (subject vehicle) refers to the vehicle that wants to change lanes. CL (current lane) is the vehicle in front of the SV. TL (target lane) is the vehicle the SV wants to be behind in its target lane. LV (lag vehicle) is the vehicle that will be behind SV once it moves to its target lane. Atagoziyev defines seven equations for manipulating vehicles between lanes. They are used in different contexts, each based on the relative positions of the surrounding vehicles. The contexts for each equation are given below, along with the behaviour from the equation during that context.

Case 1 *SV too close to CL in its current lane or TL if it changed lanes.*

SV slows down until the gap is sufficiently large enough.

Case 2 *SV has a large gap between itself and TL and CL, however, LV is too close to SV*

SV can approach CL and TL as long as the gap remains large enough. LV needs to open up a sufficient gap behind SV.

Case 3 *SV has the minimum allowable gap to TL and CL, but LV is too close*

SV follows the closest leader and waits until LV creates the necessary gap.

Case 4 *The gaps between SV and CL/TL/LV are sufficient. But CL/TL are not travelling at the 'nominal speed' established for all vehicles in this exchange*

SV maintains a sufficient gap, waiting for CL/TL to travel at nominal speed again.

Case 5 *SV and CL/TL/LV have sufficient gaps and CL/TL are travelling at nominal speed.*

SV performs the lane change, maintaining nominal speed.

Case 6 *SV obtains the minimum gap to CL/TL and LV maintains a sufficient gap. CL/TL are not travelling at nominal speed*

SV maintains the minimum gap, waiting for CL/TL to travel at nominal speed again.

Case 7 *SV obtains the minimum gap to CL/TL and LV maintains a sufficient gap. CL/TL are travelling at nominal speed*

SV performs the lane change, maintaining nominal speed.

These equations are the building blocks that lead to a lane change. The flowchart in Figure 2.3 shows how they work together to enact a single lane change.

Using this algorithm, we can move multiple vehicles into their correct lanes. LV always provides space to allow each changing vehicle into the correct lane. All of the vehicles are working with each together, such that they can all reach their goal. Comparing this to Gipps' 1986 model, where drivers are acting solely in their own interest, we can see that by having an external agent managing lane changes, the autonomous vehicles can achieve results that might not be possible in situations where drivers are acting selfishly. For example, in a grid locked situation, drivers following Gipps' model might never be able to move into their desired lane; however vehicles following Atagoziyev's model would open up spaces allowing vehicles to move through the traffic.

### 2.3.2 Lane Changing to improve overall velocity

Gipps' driver decisions model also considered situations where the driver does not have to be in any particular lane. The model considers the effects of transit lanes, heavy vehicles and the effect of the preceding vehicle on the driver's vehicle. These are shown in nodes 5 to 7 and 9 to 11 of Gipps' flowchart in Figure 2.2.

5 *Transit vehicles and lanes* Transit lanes are lanes dedicated solely for public transport and other high occupancy vehicles. These include vehicles such as buses, taxis and carpool cars. These vehicles are known in the model as 'transit vehicles'.

6 *Entry of nontransit vehicles into transit lanes* If there is an obstruction in the present lane, it is often considered to be a valid reason for a non-transit vehicle to enter a transit lane.

7 *Departure of nontransit vehicles from a transit lane* Once the obstruction has been cleared, nontransit vehicle must move back into a valid lane. This forced departure does not affect vehicles that are close to their intended turn.

9 *Relative advantages of present and target lanes* If the driver has not yet been forced to change lanes by any other factors, then they can look at the relative advantages of the present and target lanes, considering obstructions and then determining which lanes obstructions will have the least effect on their safe speed.

10 *The effect of heavy vehicles* If obstructions are level with each other or beyond the range a driver considers, then the driver considers the next heavy vehicle in each lane, as if it were the leading vehicle in an ordinary car following situation. The driver then selects the lane which will give them the higher speed.

11 *The effect of the preceding vehicle* If there are then no heavy vehicles, the driver considers the speed possible in each lane and then changes if they gain a 'sufficient' speed advantage. This is again, subjective, depending on the present lane, target lane and the type of vehicle.

Again, Gipps' 1986 model was built with human drivers in mind, and as such it fails to take advantage of the benefits of autonomous vehicles such as platooning and vehicle-to-vehicle communications.

Work by Kesting et al. in 2007 [19] 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 introduces the MOBIL or 'Minimising Overall Braking Induced by Lane Changes' model. The model uses two criterion that the vehicle must satisfy.

The first criterion deals with safety, ensuring that the deceleration of a successor vehicle $\tilde{a}_n$ in the target lane doesn't exceed a safety limit $b_{safe}$.

$$\tilde{a}_n \geq -b_{safe} \tag{2.2}$$

This criterion effectively puts a limit on the level of braking a vehicle changing lanes can cause another vehicle to undergo if it pulls out in front of it.

The second criterion is the 'incentive criterion' which is what motivates a driver to change lanes. This criterion introduces a 'politeness factor' $p$ which expresses the extent to which nearby vehicles affect a driver's lane changing decision.

The paper discusses the differences between symmetric ('US') lane changing rules and asymmetric ('European') passing rules, however in this paper we only use US lane changing rules. This gives the incentive criterion:

$$\underbrace{\tilde{a}_c - a_c}_{\text{driver}} + p\big(\underbrace{\tilde{a}_n - a_n}_{\text{new follower}} + \underbrace{\tilde{a}_o - a_o}_{\text{old follower}}\big) > \Delta a_{th} \tag{2.3}$$

$\tilde{a}_x - a_x$ is the utility a driver x gets due to the lane change, where $\tilde{a}_x$ is the acceleration of vehicle x after the lane change and $a_x$ was their acceleration before the lane change. $c$ is the vehicle changing lanes, $n$ is the vehicle behind $c$ once it changes lanes, and $o$ is the vehicle following $c$ before the lane change. $\Delta a_{th}$ is the threshold at which the driver will change lanes. It is designed to model inertia. A driver won't change lanes unless they get above a specific utility gain. The politeness factor $p$ varies from 0 to 1, where $p = 0$ is the most selfish behaviour and $p = 1$ describe drivers who won't change lanes unless collectively all of the drivers gain a utility greater than the threshold. When $p > 1$ drivers won't change lanes at all if it negatively affects the surrounding traffic, drivers will even go so far as to execute lane changes which reduce their own utility. Likewise drivers with $p < 0$ will go out of their way to negatively affect other drivers, even reducing their own utility to do so.

The idea of a MOBIL model means that drivers will only change when it increases the sum of all of the accelerations increases. This would be at $p = 1$ and $\Delta a_{th} = 0$. In this case the equation becomes

$$\tilde{a}_c + \tilde{a}_n + \tilde{a}_o > a_c + a_n + a_o \tag{2.4}$$

Kesting found that the most important parameter affecting the rate of lane changing was $p$. With a $p$ value of 1 the maximum lane changing rate was almost halved. Kesting also discovered that 'altruistic' lane changing behaviour increased the mean speed of both lanes involved in the simulation, improving overall traffic performance.

Comparing this to Gipps' 1986 approach we can see that MOBIL is far more considerate of other drivers and as such the overall speed of the vehicles on the road could be higher. MOBIL is also designed for autonomous vehicles, which allows it to enforce 'altruistic' behaviour from vehicles on the road. The model could also be extended to include V2V communications. Transmitting accurate braking profiles to other vehicles would make it easier to determine the effect a lane change would have on the overall system.

Structure of lane-changing decisions

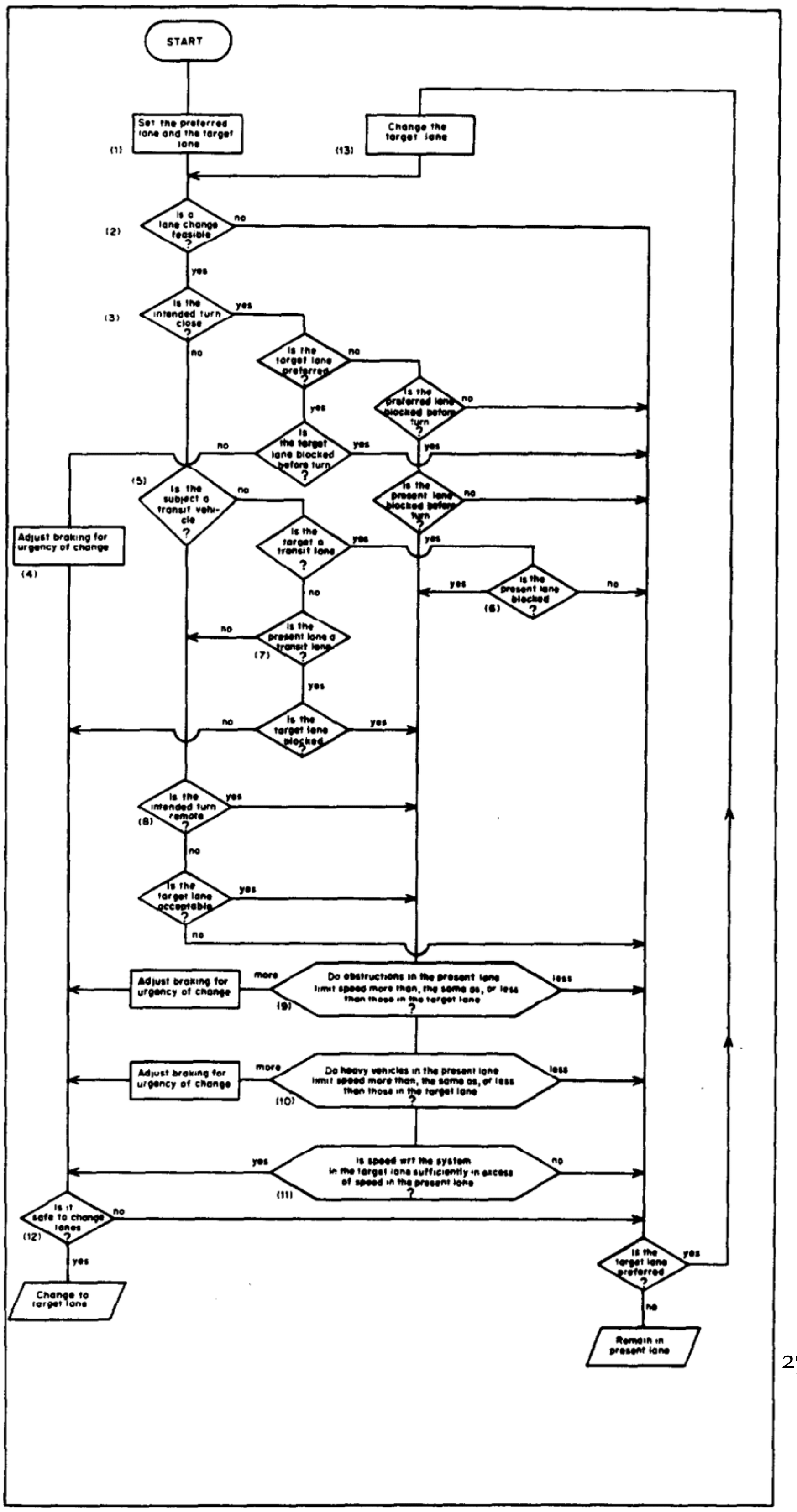





Figure 2.2: The flowchart for lane changing decisions from [2]

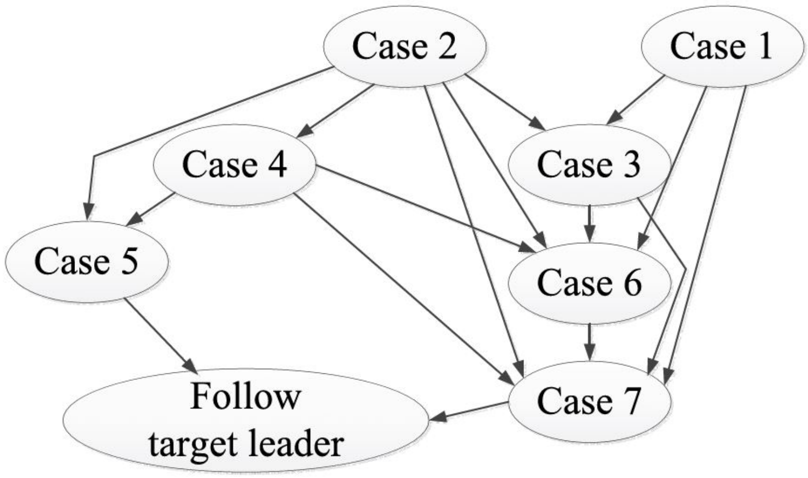



Figure 2.3: The flowchart for Atagoziyev's Lane Changing model equations

# 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 even more variance. Analysing the different merging scenarios helps to better define the problem and better create solutions for the scenario (3.1). The factors that could alter the behaviour of a merge scenario should also be defined. This makes it easier to introduce controls in the final simulator which manipulate these factors (3.3).

## 3.1 Merge Types

In this paper we focus on merges made at 'critical positions' such as junctions. This remains constant in all merge types. However, beyond this, there are many different kinds of merge scenarios that a vehicle might encounter.

### 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, as seen in Figure 3.2. In this situation we label the lane that vehicles are moving from the 'current lane' (CL), and we label the lane that vehicles move to the 'target lane' (TL). We describe the vehicles that start on the CL as 'merging vehicles' (MV) and the vehicles that start on the TL as 'target vehicles' (TV). We have our critical position where the CL and TL connect. We call the area in which the vehicles in the two lanes merge together our '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 merging vehicles, 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, as seen in Figure 3.3. Figure 3.1 shows a real world single to triple lane merge with a slip-road.

Figure 3.1: A real world single to triple lane merge with a slip-road. Source: [3]

The slip-road gives merging vehicles more time to travel parallel to the target lane 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. The effectiveness of slip-roads should change with length: the longer the slip-road, the more time MVs have to merge. This should improve the effectiveness of the merge position.

### 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, as seen in Figure 3.4. In this situation we have two target lanes. 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 merging vehicles. Two lanes also allows for more vehicles on the target lane 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, as seen in Figure 3.5.

### 3.1.3 Double-to-Double Merge

A double-to-double merge (D2D merge) describes a situation where a vehicle moves from a double lane road into another double lane road, as seen in Figure 3.6. We now have two merging lanes. The upper lane, 'merging lane 1' (ML1) merges into TL1 and the lower lane, 'merging lane 2' (ML2) merges into TL2.

With a D2D merge we now have to consider the effect of merging vehicles from ML2 driving across TL1. In addition, target vehicles can no longer laterally move out of the way of merging vehicles as they did before. Combining these factors with the wider range of options available to MVs, we can see that a D2D merge is far more complex than an S2D merge.

D2D merges can also take advantage of a slip-road, as seen in Figure 3.7. D2D merges with a slip-road work differently to other slip-road schemes. In this instance vehicles on ML1 and ML2 will both merge into TL1. However, ML2 merges into TL1 as vehicles on an S2D merge would. ML1 vehicles merge into TL1 as vehicles on an S2D merge would when there is a slip-road in play.

### 3.1.4 Lane Obstruction Merge

A lane obstruction merge is where a vehicle needs to change lanes to avoid an obstacle in their way, as seen in Figure 3.8. It is essentially an S2S merge although the vehicle will tend to move laterally to avoid the obstacle. In this situation the critical position is a point shortly before the obstruction on the CL.

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.

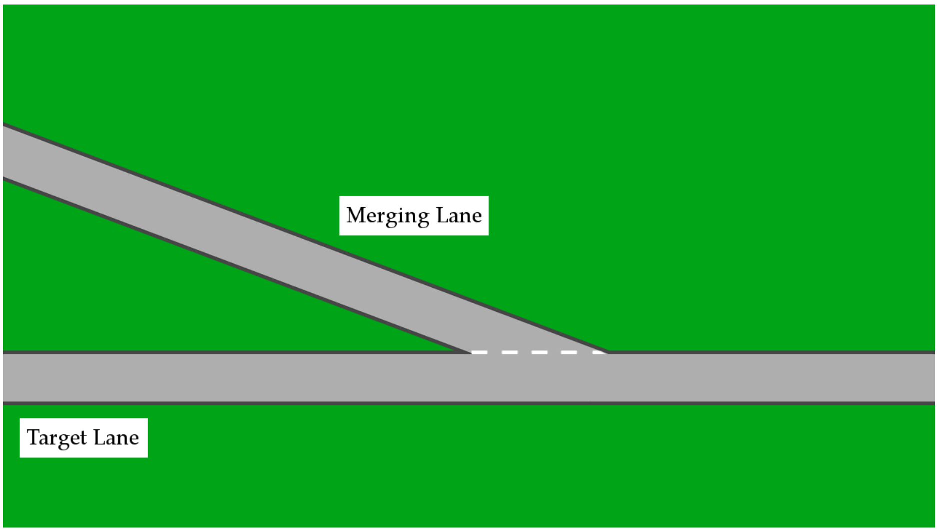



Figure 3.2: A road with a single-to-single lane merge (S2S)

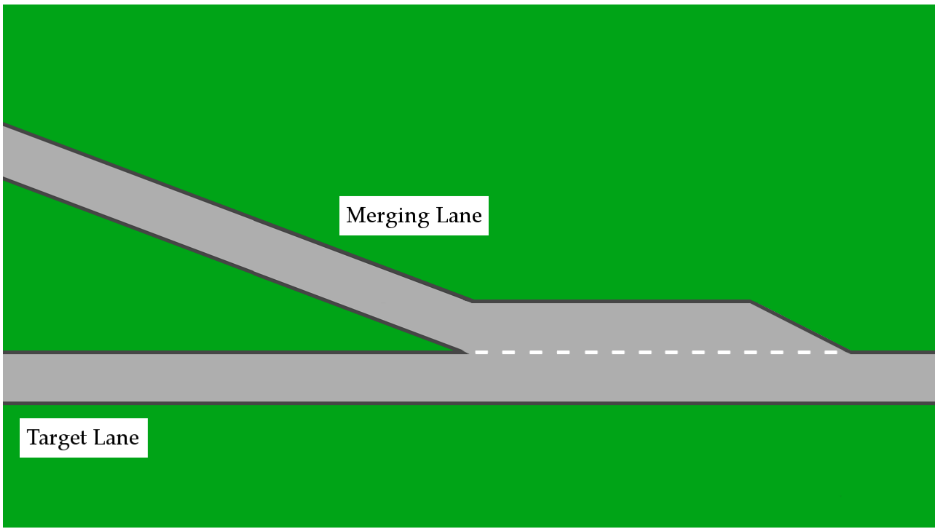



Figure 3.3: A road with a single-to-single lane merge and slip-lane (S2S)

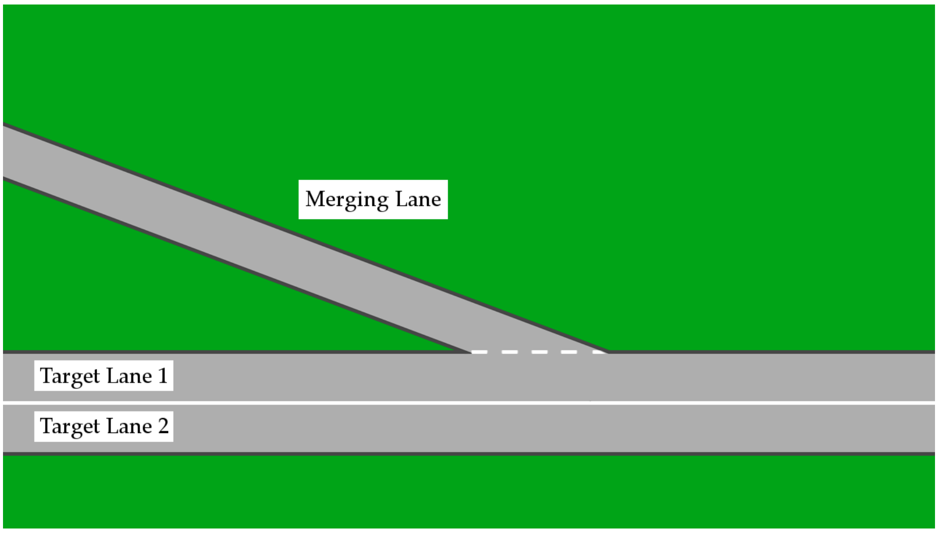



Figure 3.4: A road with a single-to-double lane merge (S2D)

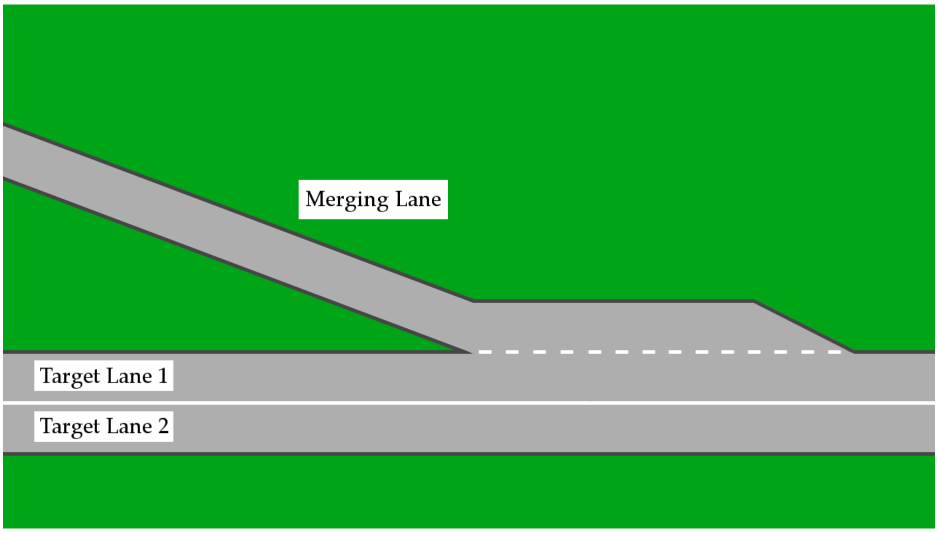



Figure 3.5: A road with a single-to-double lane merge and slip-lane (S2D)

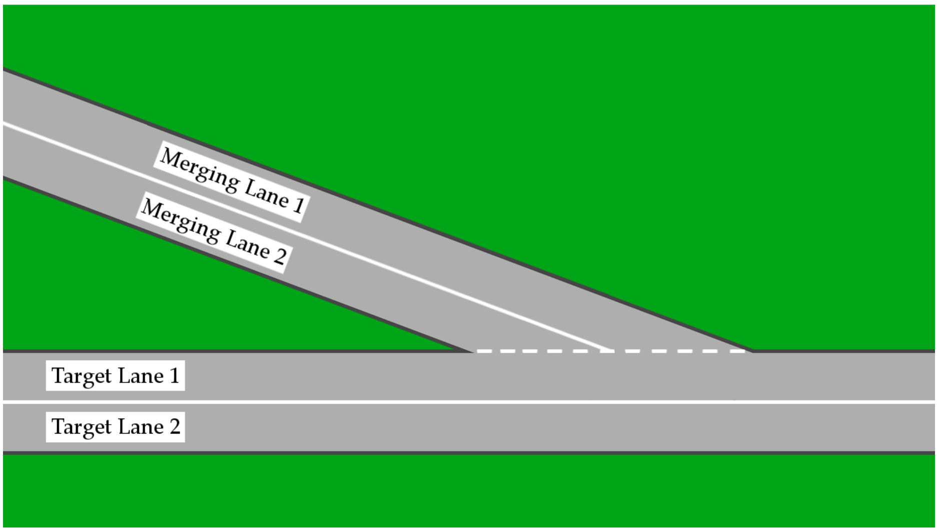



Figure 3.6: A road with a double-to-double lane merge (D2D)

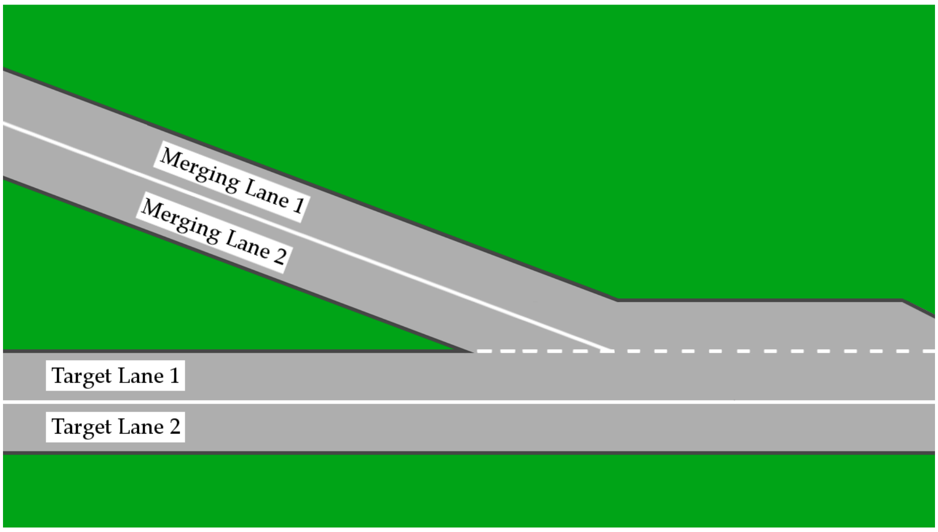



Figure 3.7: A road with a single-to-double lane merge and slip-lane (S2D)

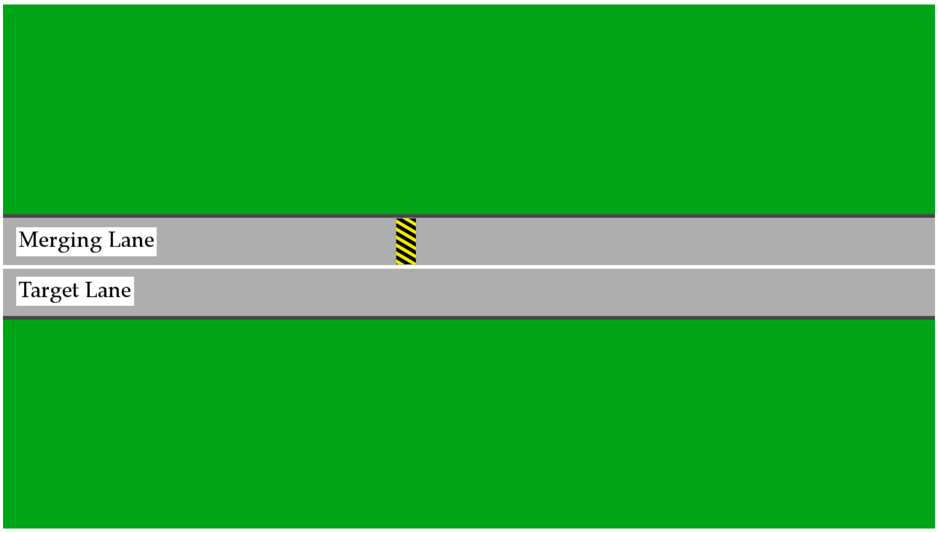



Figure 3.8: A road with a lane obstruction

## 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 have to satisfy the following conditions:

1. *No collisions* This means avoiding collisions at the critical position between merging vehicles and target vehicles, 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 both average delay and maximum delay. We do not want vehicles on one lane starving (not moving) for the benefit of vehicles on the other lane.
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 necessary, we should aim to minimises 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 [18].

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 [21] on AIM. We will measure delay in a similar manner, calculating both average delay and maximum delay.

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.

Dresner et al. also provide the following equation for measuring maximum worst case delay:

$$\max_{v_i \in C} \big(t(i) - t_0(i)\big) \tag{3.2}$$

Measuring maximum delay (and minimising it) is important, as we do not want to have a solution where some vehicles have extremely large delay times and others have very low delay times. This should help avoid a 'starvation' situation where some vehicles never get to complete their trips.

### 3.2.3 Throughput

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

$$\text{Vehicle throughput} = \frac{|C|}{t} \tag{3.3}$$

Here $t$ is the time it took for all of the vehicles in $C$ to pass through the critical position. If we want to measure the rate at which merging vehicles and target vehicles pass through the intersection separately we can change the definition of $C$ to reflect that.

### 3.2.4 Velocity Changes

Vehicles should aim want to reduce velocity changes as much as possible, aiming especially to eliminate 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 assume a constant acceleration/deceleration between the point at which a vehicle decides to start changing velocity and the point at which the vehicle has either reached it's target speed or made a new decision. We can use the time gap between those two points, and the change in the vehicle's velocity, to calculate it's acceleration or deceleration. We then take the maximum values of the acceleration of the vehicle and the deceleration of the vehicle.

## 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 3.9 shows some of the factors below, applied to an S2S merge.

- *Traffic Level* The traffic level changes the number of vehicles on the road at any one time. Effectively, it is used to measure traffic density.
- *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 target vehicles start being able to make decisions regarding the merge, and the point at which the target vehicles 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 merging vehicles start being able to make decisions regarding the merge, and the point at which the merging vehicles reach the merge zone.

- *Merging angle* The interior angle $\theta$ 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.

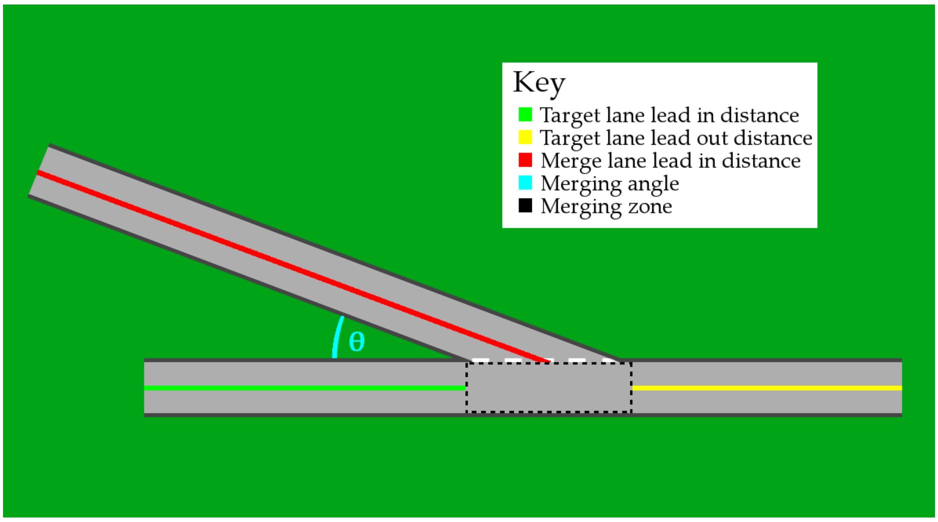



Figure 3.9: An S2S merge marked with some of the variable factors

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 be moving much more slowly. Increased traffic density also increases the likelihood of traffic shocks, and vehicle manoeuvrability is impacted.

Altering speed limits also affects performance. Higher speed limits allow vehicles to travel more quickly through the critical position and could improve performance. However, larger speed limits could also lead to larger deceleration and acceleration values. Problems could also arise if there are big differences between the speed limits of the merging lane and target lane. If the target lane speed limit is larger, merging vehicles will slow down target lane vehicles as they merge in. If the merging lane speed limit is larger then merging vehicles may have to declerate quickly to allow target vehicles to move through the merging zone.

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 lead out distance will mostly affect the result collected as the vehicle leaves the simulator. Longer lead out distances allow vehicles to return to their preferred speed. This should give more accurate results for a vehicle's maximum acceleration and deceleration.

The merging angle changes the length of the merge zone. Shallower angles will lead to longer merge zones due to the width of the merging lane. Shallower merging angles also reduce the turning angle for merging vehicles. This could lead to faster merges as merging vehicles have to make smaller changes to their heading.

## 3.4 Requirements

By 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.

### 3.4.1 Functional

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 all of the functional requirements.

Table 3.1: Functional requirements table.

<table>
<tr><th colspan="2">User Requirements</th><th colspan="2">System Requirements</th></tr>
<tr><td rowspan="3">FU.1</td><td rowspan="3">Users can run merge simulations.</td><td>FS.11</td><td>The system has controls for starting merge simulations</td></tr>
<tr><td>FS.12</td><td>The system can create merge simulations</td></tr>
<tr><td>FS.13</td><td>The system can run merge simulations</td></tr>
</table>

| User Requirements | | System Requirements | |
|---|---|---|---|
| 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 allowing running simulations to pause. |
| | | FS.32 | Merge simulations can be paused |
| FU.4 | User can end a simulation. | FS.41 | The system has controls ending 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. |
| FU.6 | Users can export the results of a merge simulation. | FS.61 | The system has controls for exporting results data. |
| | | 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 maximum delay of the simulation. |
| | | FS.65 | The results file contains the average delay of the simulation. |
| | | FS.66 | The results file contains the maximum acceleration of the simulation. |
| | | FS.67 | The results file contains the maximum deceleration of the simulation. |

| User Requirements | | System Requirements | |
|---|---|---|---|
| | | FS.68 | The results file contains the delay for each vehicle in the simulation. |
| | | FS.69 | The results file contains the maximum acceleration for each vehicle in the simulation. |
| | | FS.6a | 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" "Replacing the Stop Sign: Unmanaged Intersection Control for Autonomous Vehicles". |

| User Requirements | | System Requirements | |
|---|---|---|---|
| | | 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 target lane speed limit of an S2S simulation. | FS.111 | The system has controls for adjusting the target lane speed limit for an S2S simulation. |
| FU.12 | Users can control the merge lane speed limit of an S2S simulation. | FS.121 | The system has controls for adjusting the merge lane speed limit for an S2S simulation. |
| FU.13 | Users can control the target lane lead in distance of an S2S simulation. | FS.131 | The system has controls for adjusting the target lane lead in distance for an S2S simulation. |
| FU.14 | Users can control the target lane lead out distance of an S2S simulation. | FS.141 | The system has controls for adjusting the target lane lead out distance for an S2S simulation. |
| FU.15 | Users can control the merge lane lead in distance of an S2S simulation. | FS.151 | The system has controls for adjusting the merge lane lead in distance for an S2S simulation. |
| FU.16 | Users can control the merge angle of an S2S simulation. | FS.161 | The system has controls for adjusting the merge angle for an S2S simulation. |
| FU.17 | Users should be alerted of any collisions during a simulation. | FS.171 | The system should be able to alert the user if a collision occurs. |
| | | FS.172 | The simulation should detect collisions. |

### 3.4.2 Non-functional

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

Table 3.2: 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. |

# 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 autonomous vehicle 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 Map

In order to build the map using the user's parameters we will need to calculate the relative positions of the lane entrances and exits as well as the locations of the data collection lines and spawn points. A more detailed analysis of the mathematics used to calculate these positions can be found in Appendix A.2.

### 4.1.1 Target Lane Coordinates

The target lane entrance will have a Y-coordinate equal to half of the lane width. In fact, every coordinate that target vehicles travel on will have a Y-coordinate equal to half the lane width. The entrance's X-coordinate will depend on whether the target lane, plus the merge zone length, is longer than the base width of the merging lane, that is, the horizontal distance the merging lane covers. You can see this distance indicated in Figure 4.1.

If the target lane is longer then the entrance X-coordinate will be 0. If not then we have to do further calculations. The total width of the simulation is given by equation 4.1 if the target lane has $x = 0$ and equation 4.2 if not.

$$\begin{aligned} width = & targetLeadInDistance \\ & + mergeZoneWidth \\ & + targetLeadOutDistance \end{aligned} \tag{4.1}$$

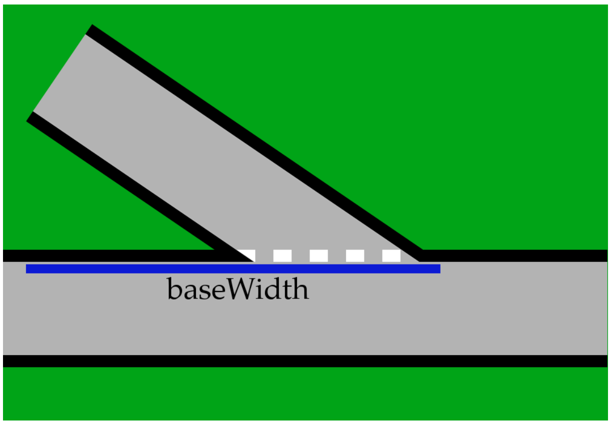



Figure 4.1: A diagram indicating the base width of the merging lane.

$$width = mergeBaseWidth + targetLeadOutDistance \tag{4.2}$$

So we can use equation 4.1 to calculate the target lane's entrance X-coordinate.

$$\begin{aligned} targetLaneStartX = & width \\ & - targetLeadInDistance \\ & - mergeZoneWidth \\ & - targetLeadOutDistance \end{aligned} \tag{4.3}$$

The target lane exit coordinates can be calculated similarly. The X-coordinate will be equal to the width of the simulator. With these co-ordinates calculated the simulator can place data collection lines and create spawn points for the target vehicles.

Finally the X-coordinate indicating the entrance to the merge zone for target vehicles is given by equation 4.4. The exit X-coordinate is given by equation 4.5.

$$\begin{aligned} targetLaneZoneEntranceX = & width \\ & - mergeZoneWidth \\ & - targetLeadOutDistance \end{aligned} \tag{4.4}$$

$$targetLaneZoneExitX = width - targetLeadOutDistance \tag{4.5}$$

### 4.1.2 Merging Lane Coordinates

The merging lane entrance coordinates are more difficult to calculate. To get the X-coordinate we can calculate an x-adjustment from the far edge of the lane to the centre, as shown in Figure 4.2.

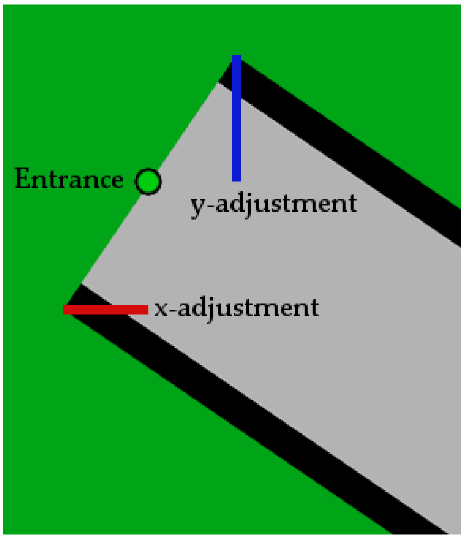



Figure 4.2: A diagram indicating the x and y adjustments for the merging lane.

To get the Y-coordinate we can calculate the distance the coordinate is above the target lane, and then add the width of the target lane to that. We can also use this length to calculate the height of the whole simulator. However, to do that we will also need to add a y-adjustment, also shown in Figure 4.2.

Merge vehicles will exit the merge zone in the same place as target vehicles (equation 4.5), however, they will enter the merge zone at the top. The Y-coordinate of this point is equal to the width of the target lane. The X-coordinate is given by equation 4.6.

$$\begin{aligned} mergeLaneZoneEntranceX = & width \\ & - targetLeadOutDistance \\ & - \frac{mergeZoneWidth}{2} \end{aligned} \tag{4.6}$$

After the merge vehicles enter the merge zone they will deviate from the lane, however the lane itself continues until it reaches the centre of the target lane. We know this centre has a Y-coordinate equal to half of the lane width. The X-coordinate is more difficult to calculate. We need to find another X-adjustment, denoted as centreXAdjustment, along the target lane centre line. We can then use equation 4.7 to find the X-coordinate. Figure 4.3 indicates the position of centreXAdjustment.

$$\begin{aligned} connectionPointX = &width \\ &- targetLeadOutDistance \\ &- \frac{mergeZoneWidth}{2} \\ &+ centreXAdjustment \end{aligned} \tag{4.7}$$

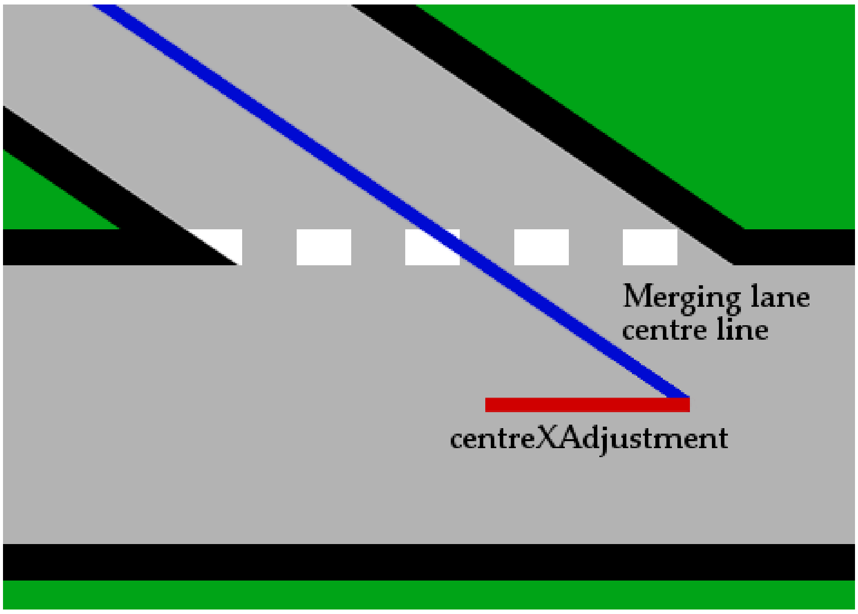



Figure 4.3: A diagram indicating the X-adjustment used to find the point at which the two lanes connect.

.

## 4.2 Autonomous Merge Management System

This centralised system is based on the AIM system [21]. 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 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 a reservation very close to the merge zone. These vehicles have to slow in order to not enter the merge without a reservation, and as such experience reduced traffic flow.

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 4.3.

## 4.3 Queue Merge Management System

The queue merge management system is another approach to centralised vehicle control. In 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 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 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 predecessor is added to the queue in a position before the vehicle. If the vehicle's predecessor has not yet been added 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. However, multiple vehicles on the same lane can be placed into the queue at any time, and the queue system does consider each lane equally, with no preference given to any one lane.

## 4.4 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" [22]. 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 [22] 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 its $C_1$'s vehicle is stopped at the merge and $C_2$'s vehicle is not. Or if $C_1$ and $C_2$ are either both stopped at the merge or both not stopped at the merge, and $C_1$ has priority over $C_2$. Priority is indicated by the following rules, 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 target lanes generally move more quickly than merge lanes.

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 predecessor 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 section 4.2. 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, an was built on top of the AIM simulator codebase. All class diagrams were created using IntelliJ IDEA 15.0.3 internal diagram tool. Figure 5.1 provides a key for understanding these diagrams.

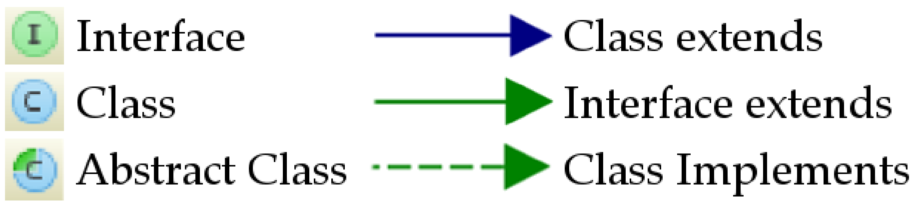



Figure 5.1: Key for the class diagrams in this report.

## 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 will 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' [23], also uses simulators built using AIM. 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 [24].

For brevity, I will only cover how the Vehicle classes were generalised. Appendix A.3 provides detailed coverage of both this change, and changes made to some of the other areas of the AIM codebase.

### 5.1.1 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 5.2 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 5.3 shows how the new structure links together.

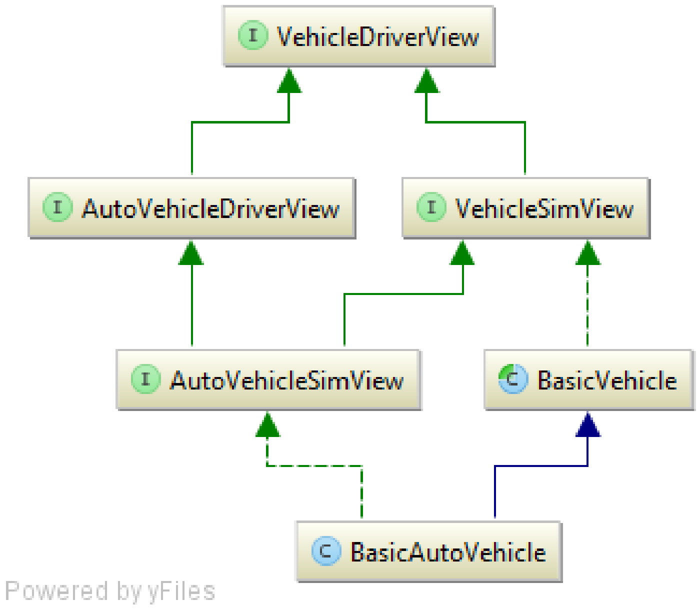



Figure 5.2: The original class structure for *aim4.vehicle*.

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

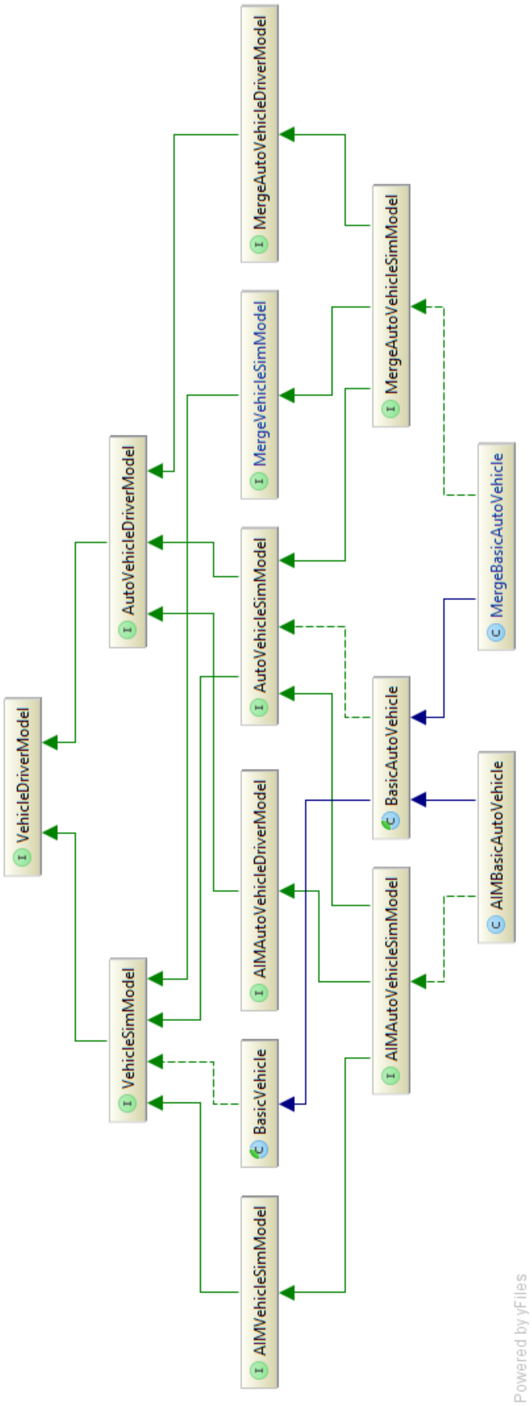



Figure 5.3: The new class structure for *aim4.vehicle*.

given a model of Driver and AutoDriver (beyond which they care very little) that they can use to access methods.

*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*.

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).

| Requirement Code | Acheived? |
|---|---|
| NS.11 | ✓ |
| NS.12 | ✓ |

## 5.2 Merge Schemes

The AIM 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. In the end, despite this approach there were significant issues with the system. Despite using very similar approaches to AIM, almost identical in areas, vehicles would continue to arrive early to their reserved times and vehicles would also collide consistently at intersections.

The system also suffered from some more fundamental problems. Reservations for merges are only taken by the lead vehicle in each lane, as vehicles behind them shouldn't be able to reserve ahead of the lead vehicle. This leads to slow downs at the merge zone as secondary vehicles won't be able to make reservations until the lead vehicle has entered the merge. If secondary vehicles also have to compete with vehicles on the other lane then it is likely that at least one of them will have to stop at the merge. This is fine for four-way stops, but for a merge where maintaining vehicle flow is the primary aim, this system will be far less effective.

The AIM protocol also fails to ensure that one lane does not suffer for the benefit of the other. Reservations are granted on a first-come-first-serve basis (though this could be changed by implementing a different reservation policy) and this can lead to long periods of time where one lane fails to make reservations whilst the other passes vehicles through quickly. Again, this fails to maintain traffic flow, one of the key aims of a merging protocol.

As a response to AIM, I developed an alternative centralised approach to the merge problem. Described in section **??**, this system uses a queue, as opposed to a space-time reservation matrix. This system is far less space-time optimal than the AIM system, however, this system is far more straight forward and does not cause non-lead vehicles to stop at the entrance to a merge. To allow a direct comparison between this system and an AIM based alternative, I created a modified version of an AIM simulation, where only one road out of an intersection, and two roads leading into it are enabled. This allows for a comparison between a Queue system set up at 90°and the AIM system.

The original plan was to compare a centralised system with a decentralised solution, based on the work from VanMiddlesworth [22]. However, this was never implemented due to time constraints.

| Requirement Code | Acheived? |
|---|---|
| FS.81 | ✗ |
| FS.91 | ✗ |

## 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 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 containing 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 more tightly, freeing up the lane for more vehicles. Figure 5.4 shows how these turns work.

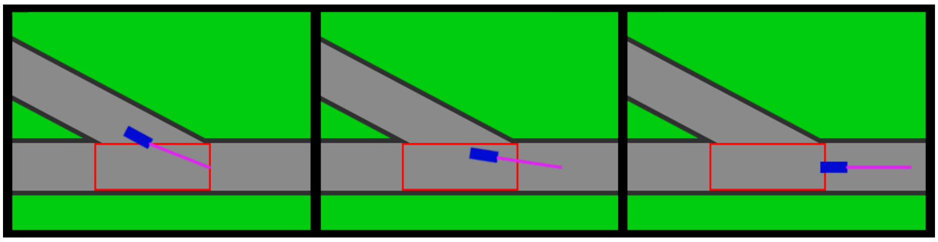


Figure 5.4: A diagram indicating how turning works through a merge. Red indicates the Merge Zone, Blue indicates a vehicle, Pink indicates the aiming distance and the point the vehicle is turning towards.

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. Even within the AIM simulations vehicles are colliding. Due to time constraints collision detection was removed from the project as I did not have time to go hunting for the error causing collisions. Overall it does not matter too much as the cars only collide momentarily before separating, but it does make it difficult to detect collision errors caused by merging algorithms. As far as I can tell, no collisions take place with the algorithm implemented, as it should be almost imposible, but without having a check in place this cannot be said for certain.

### 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 AIM based 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 deals 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 technical code is original and far simpler. This merge manager effectively just manages a queue and alerts vehicles when they are at the front of said queue. The AIM system is more complex, requiring the merge manager to monitor reservations and time the arrival of vehicles as they arrive. The AIM system relies very heavily on each vehicle arriving at their stated time and fails to handle vehicles well if they don't.

### 5.3.3 Map

The simulation map stores all of the spawn points, lanes and merge managers for the simulation. The calcualtions for lane positions and spawn points were created using the designs in 4.1.

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 Rectangle at the beginning of each lane, which works fine when lanes are at 90°. For my no vehicle zones, I had to draw a path around the start of each merge lane and create a 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 a vehicle specification and a time. The spawner 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 comparisons between the performance of AIM and the Queue protocol are now possible.

### 5.3.4 Simulation Control

The simulator itself is reponsible 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.

| Requirement Code | Acheived? |
|---|---|
| FS.12 | ✓ |
| FS.13 | ✓ |
| FS.22 | ✓ |
| FS.32 | ✓ |
| FS.42 | ✓ |
| FS.71 | ✓ |
| FS.73 | ✓ |
| FS.172 | ✗ |
| NS.3 | ✓ |
| NS.4 | ✓ |
| NS.8 | ✓ |

## 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. This allows for a comparison between AIM and Queue protocols at 90°.

To calculate the delay the simulator first simulates each vehicle specification for both the merge and target road. 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 move vehicles 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. AIMs current design uses the maximum deceleration for each vehicle as they approach the merge zone, braking at the last moment as agressively 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 preemptively 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 premptive 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. Each result contains the vehicle's

- VIN
- Starting Road

- Vehicle Specification Name
- Start Time
- Finish Time
- Delay
- Final Velocity
- Max Velocity
- Min Velocity
- Final X Position
- Final Y Position

| Requirement Code | Acheived? |
|---|---|
| FS.62 | ✓ |
| FS.63 | ✓ |
| FS.64 | ✓ |
| FS.65 | ✓ |
| FS.66 | ✗ |
| FS.67 | ✗ |
| FS.68 | ✓ |
| FS.69 | ✗ |
| FS.6a | ✗ |
| NS.7 | ✓ |

## 5.5 GUI

The GUI for the project was built using Java Swing, extending the existing AIM 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 full illustrated canvas 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

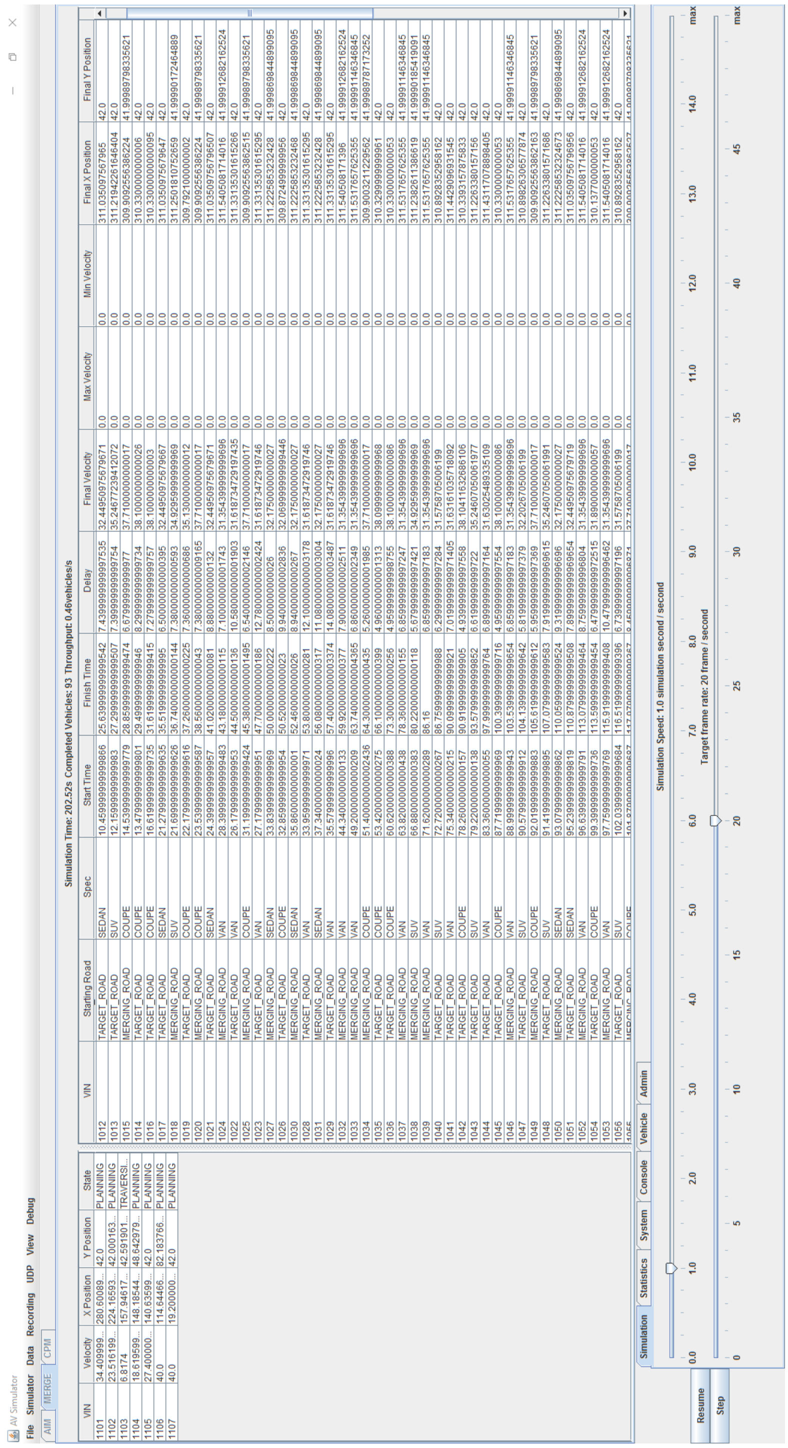


Figure 5.5: The simulation screen for the S2S merge screen

that have left the simulation. Figure 5.5 shows the simulation screen for S2S merges.

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.171* 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 uploading 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 to use the codebase.

| Requirement Code | Acheived? |
|---|---|
| FS.11 | ✓ |
| FS.21 | ✓ |
| FS.31 | ✓ |
| FS.41 | ✓ |
| FS.51 | ✓ |
| FS.61 | ✓ |
| FS.72 | ✓ |
| FS.82 | ✗ |
| FS.92 | ✗ |
| FS.101 | ✓ |
| FS.111 | ✓ |
| FS.121 | ✓ |
| FS.131 | ✓ |
| FS.141 | ✓ |
| FS.151 | ✓ |
| FS.161 | ✓ |
| FS.171 | ✗ |
| NS.1 | ✓ |
| NS.2 | ✓ |
| NS.5 | ✓ |
| NS.6 | ✓ |

## 5.6 Maintainability and Testing

### 5.6.1 Maintainability

The separability imposed between 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 an AIM based reservation system), but in general most of the classes are broken out correctly.

In terms of maintaining existing code, in general the prospects are quite good. 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 in this instance, but it could be done 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 classes in section 5.1 should indicate how complex some of the class structures have become.

### 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 [25] to mock the behaviour of objects used by the test class so that I could prompt the test class into producing the expected results.

### 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 occuring. These tests also allowed me to identify key errors with the AIM simulator's approach that were causing problems with the AIM based merge solution.

| Requirement Code | Acheived? |
|---|---|
| NS.9 | ✓ |
| NS.10 | ✓ |

# 6 Results

The simulator was designed to allow for variance in merge angle, 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 Queue protocol. We can also compare the Queue 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 density (or 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 can cause reduced numbers of completed vehicles if the system becomes congested enough to cause queues up to the spawning area.

## 6.2 Comparing AIM and Queue Protocols

By using the modified AIM simulator described in section 5.2, I obtained approximations for how well the AIM protocol handles 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 the queue merge type. 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 0.97s with a standard deviation of 1.35s, queue achieved a mean delay of 1.84s with a standard deviation of 2.22s. As the traffic rates increased however, the performance of the queue system degraded massively in comparison to AIM. At 2500vhl the queue system 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 high standard deviation like this, it's clear that AIM outperforms the queue system in this respect. Figure 6.1 shows how the two systems compare.

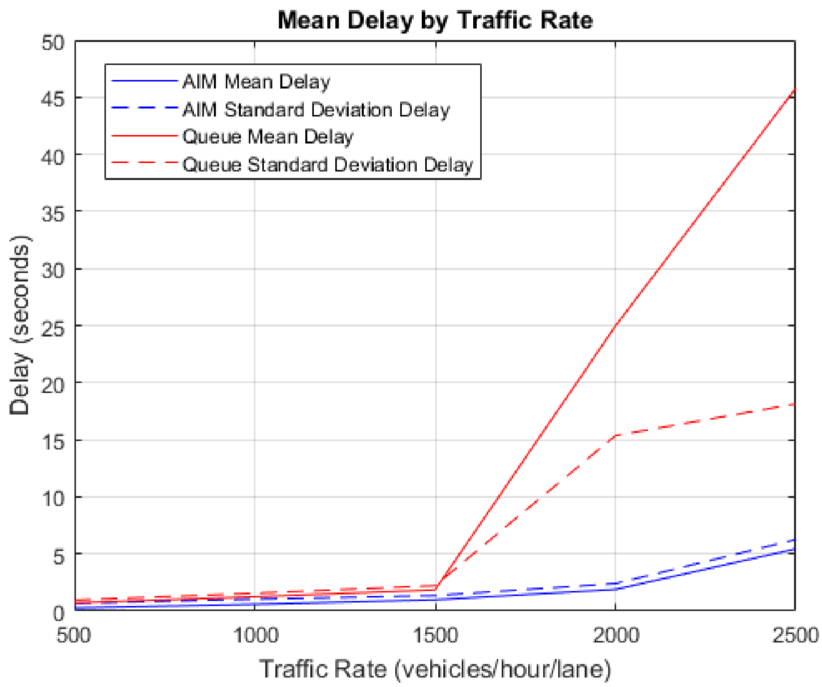



Figure 6.1: Plot showing mean delay/traffic rate performance by merge management system.

In terms of balancing mean delay over both lanes, the queue system also fails to perform as well as AIM. At 500vhl the mean target lane delay for the queue system 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 the queue system. Figure 6.2 shows how each lane performs under each system.

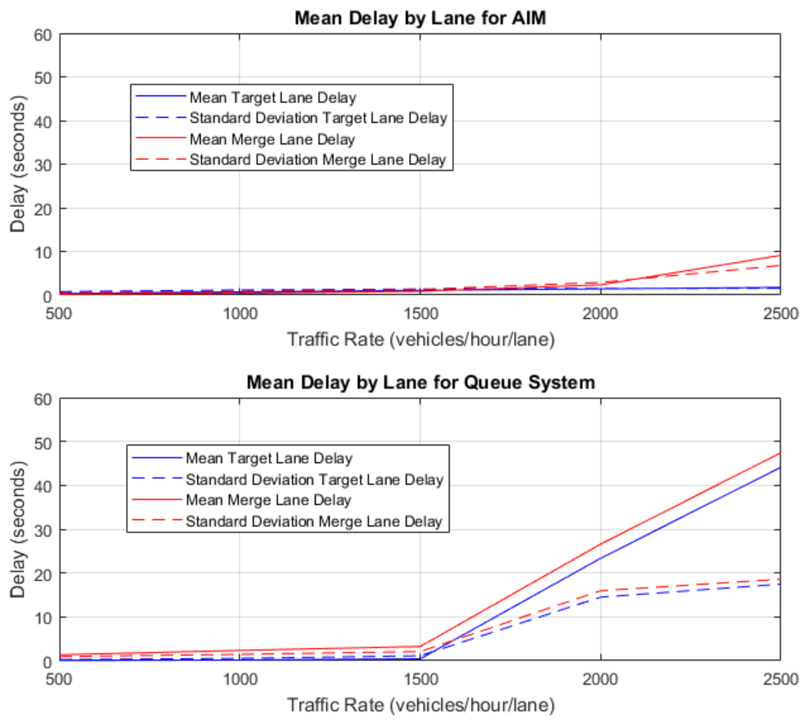



Figure 6.2: Plots showing mean delay by lane for both merge management systems

Both system 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 the queue system. Figure 6.3 shows the throughputs for each system.

These trends clearly demonstrate that AIM performs far more effectively at high traffic rates than the queue system. AIM makes better use of space-time, so when the roads begin getting more congested, this pays

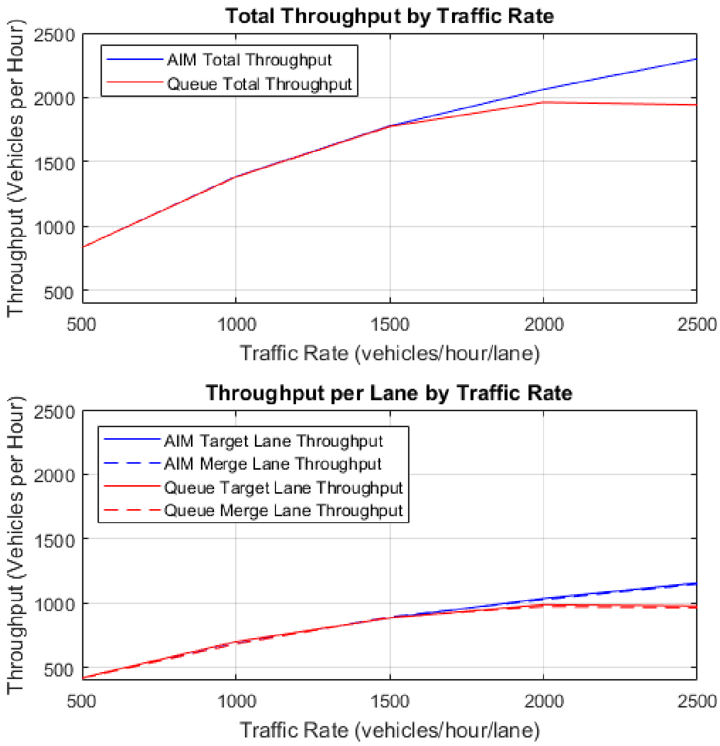



Figure 6.3: Plot showing throughput for both merge management systems

off in AIMs favour. The queue system ends up causing problems at these high traffic rates as it only allows one vehicle into the merge zone at any one time. A queue system 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 of the two.

This test makes a good case for continuing attempts to develop an AIM based system for merges. 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. As detailed in section **??**, collisions and early arrival times were constant problems with attempts to implement an AIM-based merge system 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, affecting how far they're needed to turn. For the queue system 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 the queue system performed very poorly. 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 has improved 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 with range of the standard deviation for 90°. The delay of both the target lane and merge lane follow very similar trends. Figure 6.4 shows how the delay decreases as the angle increases.

In general throughput increases dramatically with the merge angle until around 35°where 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 validity 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 the queue system 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.

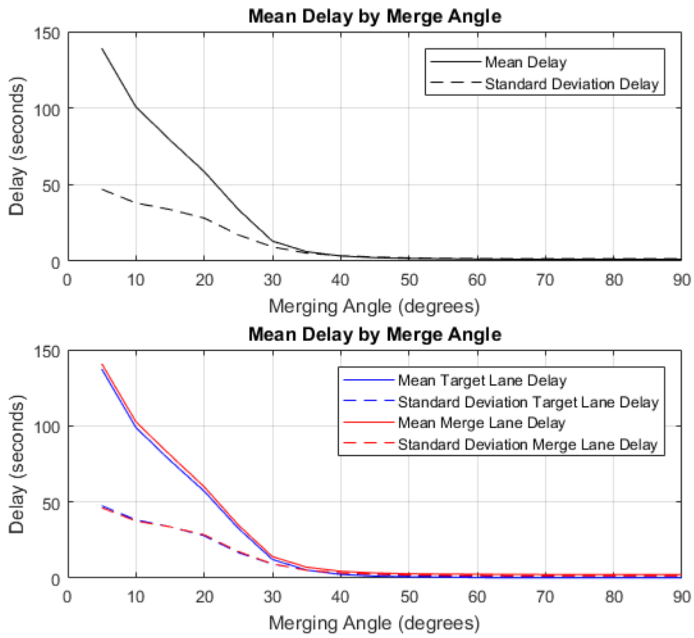



Figure 6.4: Plot showing mean delay by merge angle

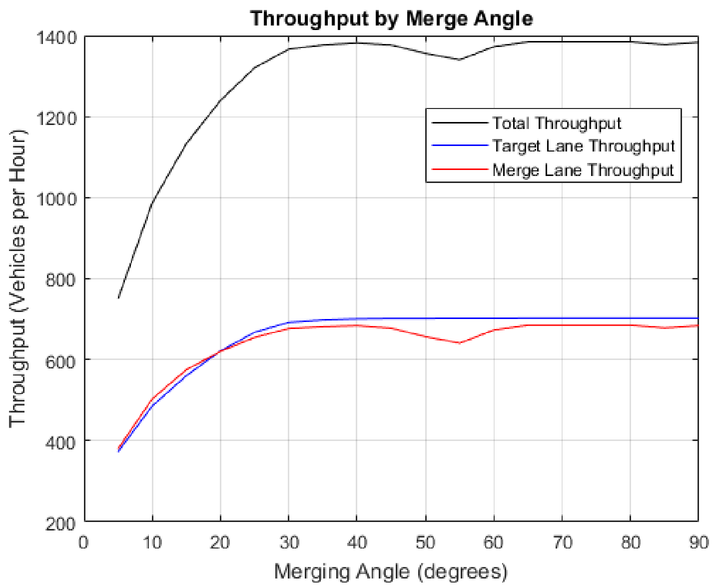



Figure 6.5: Plot showing throughput by merge angle

## 6.4 The Effect of Lead in Distances

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

Due to 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. Figure 6.6 shows how the delay relates to the lead in distance.

Beyond 150m the lead in distance had very little effect on the performance of the system. Throughput remained relatively constant throughout.

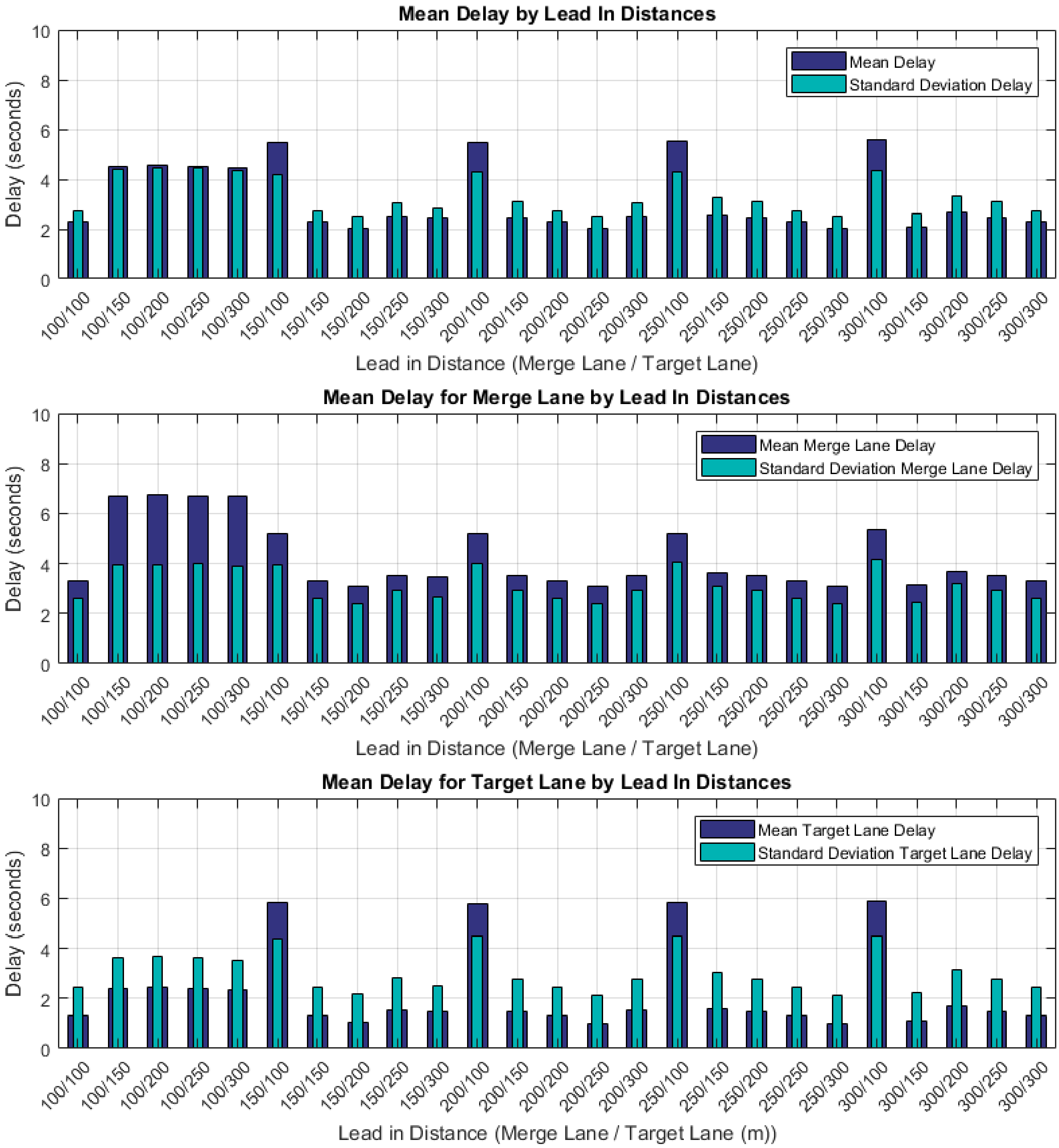



Figure 6.6: Bar chart showing mean delay by lead in distance

## 6.5 The Effect of Differing Speed Limits

The speed limits of each lane and their difference impacts how well a vehicle can move into another lane and adjust to it's velocity. Many merges will be movements from lanes of differing speed, so its important that the queue system 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 velocity is larger than the merge lane. 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 is travelling faster 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. When the merge lane speed was $30ms^{-1}$ and the target lane speed was $20ms^{-1}$ the system performed about as well as when both lanes have a speed limit of $30ms^{-1}$, at least on the target lane. It should also be noted that most of the speed limits containing a $40ms^{-1}$ lane performed poorly. The velocity causes vehicles to accumulate too quickly for the queue system to be able to sort them out effectively. Figure 6.8 shows how the mean delay is affected by difference speed limit pairs.

The number of vehicles that complete for each lane is solely affected by the speed limit of the lane, the differences between the lanes has very little effect. The throughput is likewise only affected by the speed limits of each lane independently. Figure 6.7 shows how the throughput is affected by the speed limits.

These results show that the queue system struggles to handle differ-

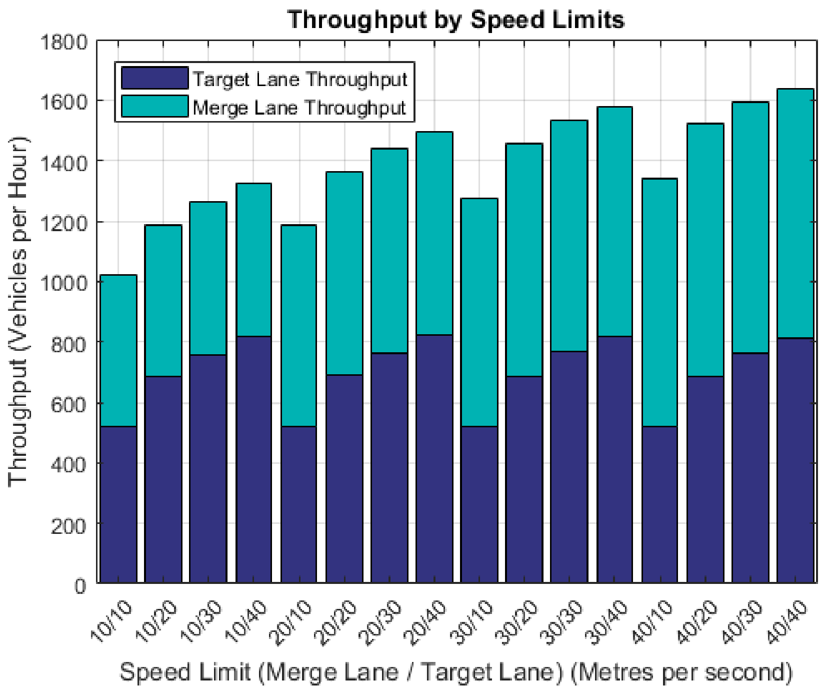



Figure 6.7: Bar chart showing throughput by speed limit pair

ences 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. In an ideal world a system should be able to slow down vehicles more gradually to allow vehicles in, before speeding up again. As it stands both the AIM and queue systems slow vehicles down at their maximum deceleration.

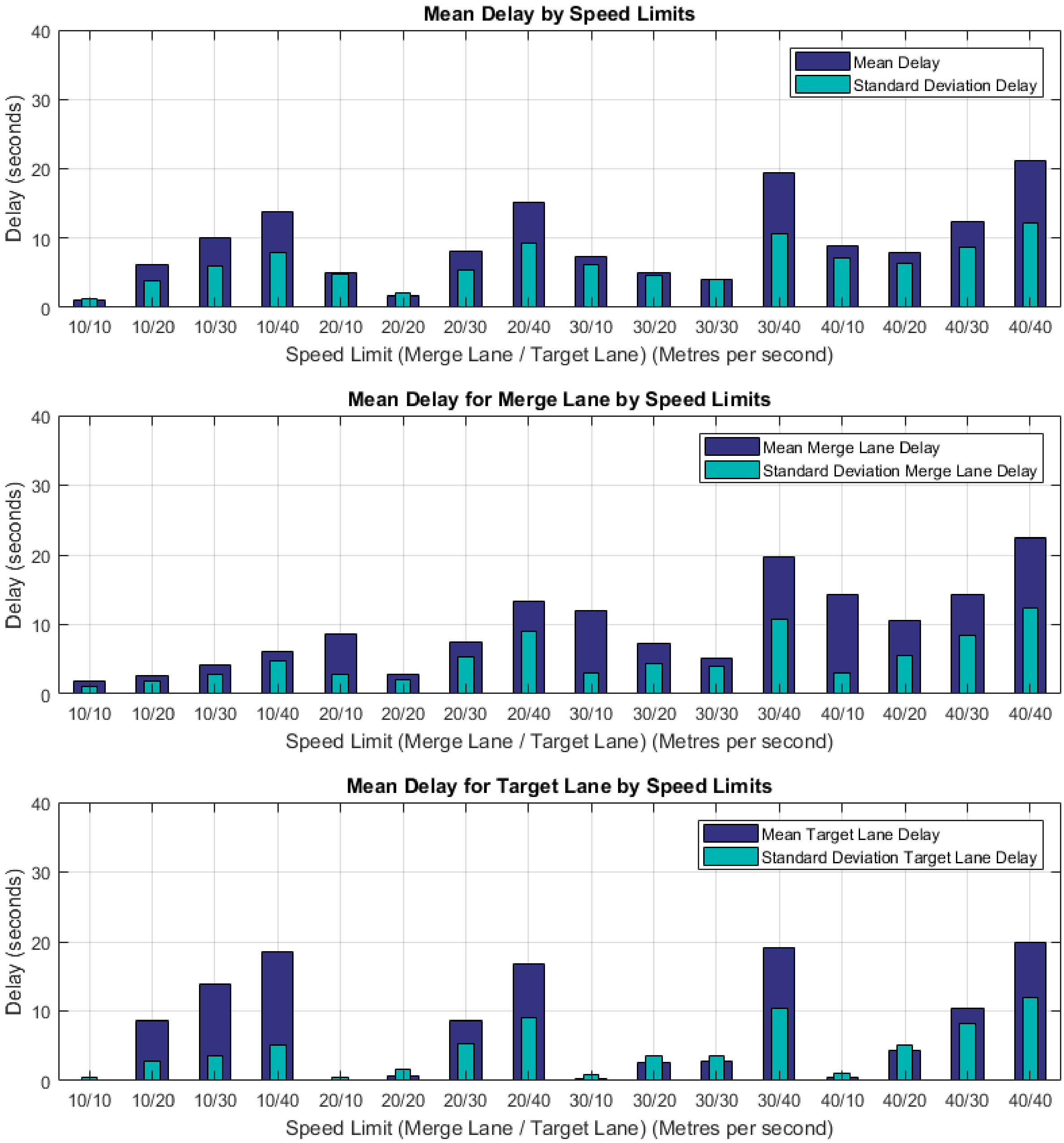



Figure 6.8: Bar chart showing mean delay by speed limit pair

# 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 fails 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 implemented due to time constraints, but a design based on the work of [22] was defined in Section 4.4.

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 never be used as a solution to the S2S merge problem, it did help to identify some of the issues that more successful 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 and assumption of 90°roads caused numerous issues making it difficult to develop the 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. I would be interested in seeing 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 also be adapted to other merge scenarios from Section 3.1, such as the S2D merge.

I would also like to see the Decentralised Merge Management system designed in Section 4.4 developed into a fully working simulation and then 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] P. Gipps, "A model for the structure of lane-changing decisions," *Transportation Research Part B: Methodological*, vol. 20, no. 5, pp. 403–414, October 1986.

[3] L. Sommerfeld. (2017, Jan.) Zipper merging may be frustrating, but it works. Online. Driving. [Online]. Available: http://driving.ca/auto-news/news/zipper-merging-may-be-frustrating-but-it-works

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

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

[6] (2016, Dec.) Waymo. Alphabet. [Online]. Available: https://waymo.com/ontheroad/

[7] (2017) Press kit | tesla. Website. Tesla Motors. [Online]. Available: https://www.tesla.com/presskit/autopilot

[8] (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

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

[10] 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

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

[12] 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

[13] 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

[14] (2016, September) Tesla criticised over autopilot safety. Website. BBC. [Online]. Available: http://www.bbc.co.uk/news/technology-37372933

[15] (2016, October) Aim homepage. Website. The University of Texas. [Online]. Available: http://www.cs.utexas.edu/~aim/

[16] 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

[17] 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

[18] 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

[19] 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.

[20] 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

[21] 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.

[22] 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"

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

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

[25] 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 Literature Review Further Content

### A.1.1 Gipps 1981 Equations

Gipps' 1981 car-following model paper [17] defines two equations, which provide constraints on the speed of vehicle $n$ at time $t + \tau$. $t$ is the current time and $\tau$ is the apparent reaction time, a constant for all vehicles. The first equation defines the acceleration constraint of the vehicle. It was obtained using measurements from an instrumented car.

$$v_n(t+\tau) \leqslant v_n(t) + 2.5 a_n \tau \left( \frac{1 - v_n(t)}{V_n} \right) \left( \frac{0.025 + v_n(t)}{V_n} \right)^{1/2} \tag{A.1}$$

$v_n(t)$ is the speed of vehicle $n$ at time $t$. $a_n$ is the maximum acceleration the driver of vehicle $n$ wishes to undertake. $V_n$ is the target speed for vehicle $n$. The equation shows that the driver accelerates until close to their target speed. Then, they reduce their acceleration until it reaches zero. At this point the vehicle should be travelling at it's target speed.

The second constraint is the braking profile of the vehicle. This is given as

$$\begin{aligned} &v_n(t+\tau) \leqslant \\ &b_n \tau + \sqrt{\left( b_n^2 \tau^2 - b_n \left( 2 \left[ x_{n-1}(t) - s_{n-1} - x_n(t) \right] - v_n(t)\tau - \frac{v_{n-1}(t)^2}{\hat{b}} \right) \right)} \end{aligned} \tag{A.2}$$

$b_n$ is the most severe braking the driver of vehicle $n$ wishes to undertake. It is always a negative value, and should be considered negative acceleration. $\hat{b}$ is the driver of vehicle $n$'s best guess at $b_{n-1}$ where $n-1$ is $n$'s predecessor. $x_n(t)$ is the location of the front of vehicle $n$ at time $t$. $s_n$ is the effective size of vehicle $n$. This is equal to the physical length of

$n$, plus a margin $n$'s successor is not willing to enter, even when $n$ is at rest.

Therefore, at time $t+\tau$, assuming the driver travels as fast as is safe, and within the limitations of the vehicle, their speed is given by the minimum of these two equations.

$$v_n(t) = \min\left((\mathrm{A.1}), (\mathrm{A.2})\right) \tag{A.3}$$

### A.1.2 The Intelligent Driver Model

In 2000 Treiber et al. suggested the 'Intelligent Driver Model' (IDM) [18]. In the IDM, the acceleration of vehicle $\alpha$, $\dot{v_\alpha}$, is defined using a continuous function of its velocity, $v_\alpha$; the distance to the rear of its predecessor, $s_\alpha$; and the velocity difference of $\alpha$ and it's predecessor, also known as the approaching rate $\Delta v_\alpha$. The vehicle interactions are solely based on $\alpha$'s relative acceleration to its predecessor. The model only provides position information for a vehicle in relation to its predecessor, and it does not provide its velocity at a given time, as Gipps' model does.

The IDM is broken into two components. The first describes the behaviour of a vehicle on a free road.

$$\dot{v_\alpha} = a^{(\alpha)}\left[1 - \left(\frac{v_\alpha}{v_0^{(\alpha)}}\right)^\delta\right] \tag{A.4}$$

Here $a^{(\alpha)}$ is the maximum acceleration of vehicle $\alpha$ and $v_0^\alpha$ is the desired velocity of $\alpha$. $\delta$ is the acceleration exponent, which is typically 4.

The second component describes the behaviour of a vehicle as it approaches its predecessor.

$$\dot{v_\alpha} = -a^{(\alpha)}\left(\frac{s^*}{s_\alpha}\right)^2 \tag{A.5}$$

As the gap, $s_\alpha$, between $\alpha$ and it's predecessor, gets closer to the desired minimum gap $s^*$, $\alpha$ decelerates.

Interpolating the two components gives us the IDM.

$$\dot{v_\alpha} = a^\alpha\left[1 - \left(\frac{v_\alpha}{v_0^\alpha}\right)^\delta - \left(\frac{s^*(v_\alpha, \Delta v_\alpha)}{s_\alpha}\right)^2\right] \tag{A.6}$$

The desired minimum gap in the IDM varies dynamically with velocity and approaching rate. It is given by the following function.

$$s^*(v, \Delta v) = s_0^{(\alpha)} + s_1^{(\alpha)} \sqrt{\frac{v}{v_0^{(\alpha)}}} + T^{\alpha} v + \frac{v \Delta v}{2\sqrt{a^{(\alpha)} b^{(\alpha)}}} \tag{A.7}$$

The equation takes the bumper-to-bumper space $s_0^{(\alpha)}$, also known as the minimum jam distance, and adds the comfortable jam distance $s_1^{(\alpha)}$. The bumper-to-bumper space is the minimum gap between $\alpha$ and its predecessor in stationary traffic. The comfortable jam distance is an extra distance added on for comfort, and to allow for a slower driver reaction time. In the paper, this value is set to 0. We can also consider it negligible for autonomous vehicles. $T$ is the safe time headway; it represents the time required for the vehicle to safely come to a stop. Finally $b^{(\alpha)}$ is the desired deceleration for $\alpha$.

## 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.8.

$$mergeZoneLength = \frac{laneWidth}{\sin(\theta)} \tag{A.8}$$

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

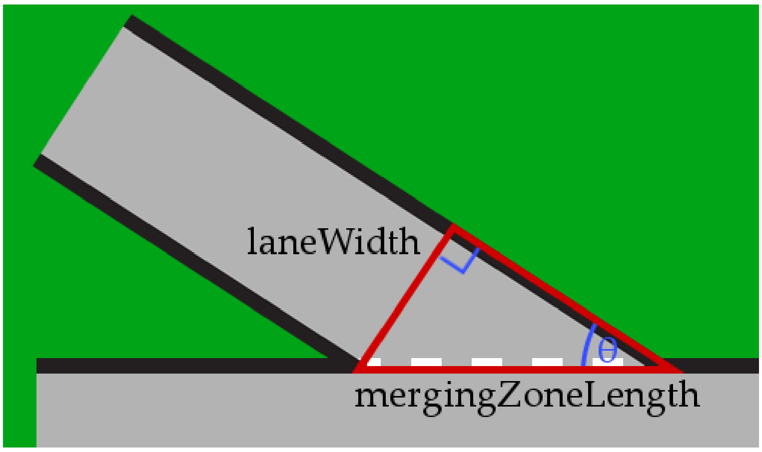



Figure A.1: A right-angled triangle used to calculate the size of the merging zone.

These triangles have the same dimensions and have an interior angle of 90 - $\theta$ 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.9 to calculate the X-adjustment and equation A.10 to calculate the Y-adjustment.

$$\text{x-adjust} = \frac{laneWidth}{2}\cos(90-\theta) \tag{A.9}$$

$$\text{y-adjust} = \frac{laneWidth}{2}\sin(90-\theta) \tag{A.10}$$

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.11 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.

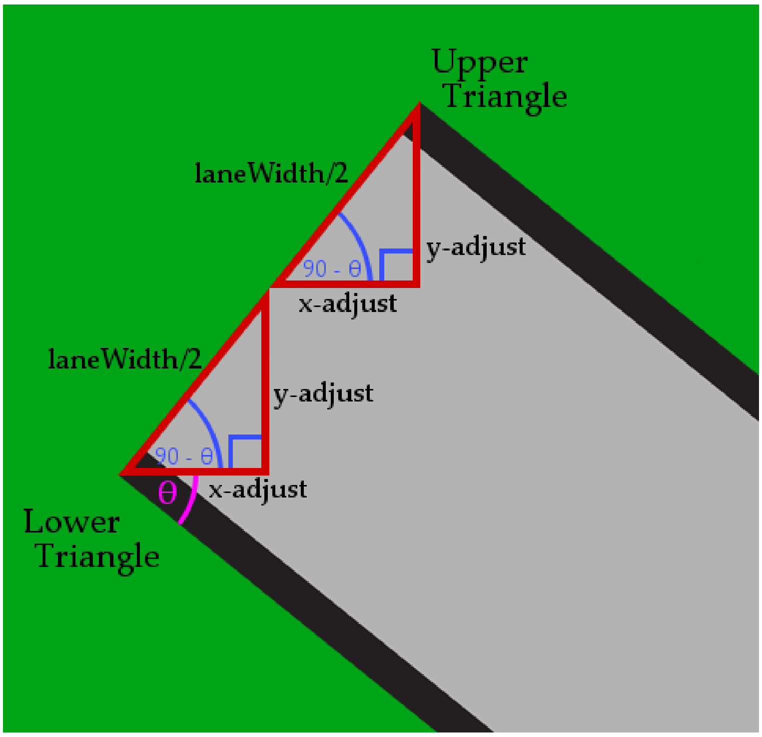



Figure A.2: Two right-angled triangles used to calculate the x and y adjustments for the merge entrance.

$$mergingLaneCentreLineBase = mergeLeadInDistance \cos(\theta) \tag{A.11}$$

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.12 to find the X-adjustment from the merge zone centre to the point where the two centre lines cross.

$$toCentreDistance = fraclaneWidth2 \tan(\theta) \tag{A.12}$$

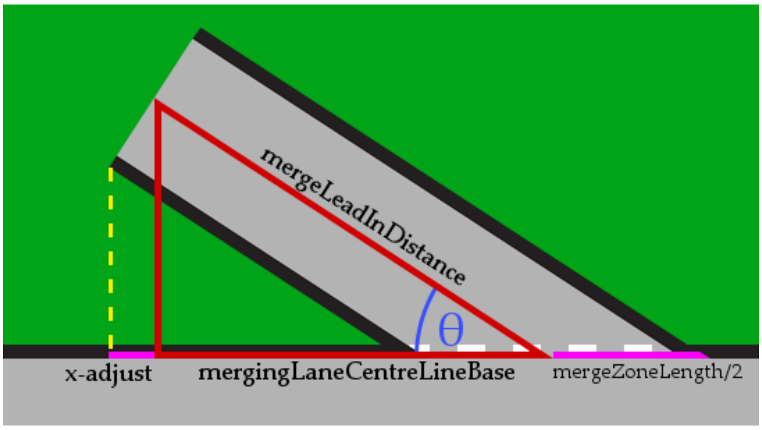



Figure A.3: A right-angled triangle used to help calculate the base width of the merge lane, along with the X-adjustment and merge-zone length.

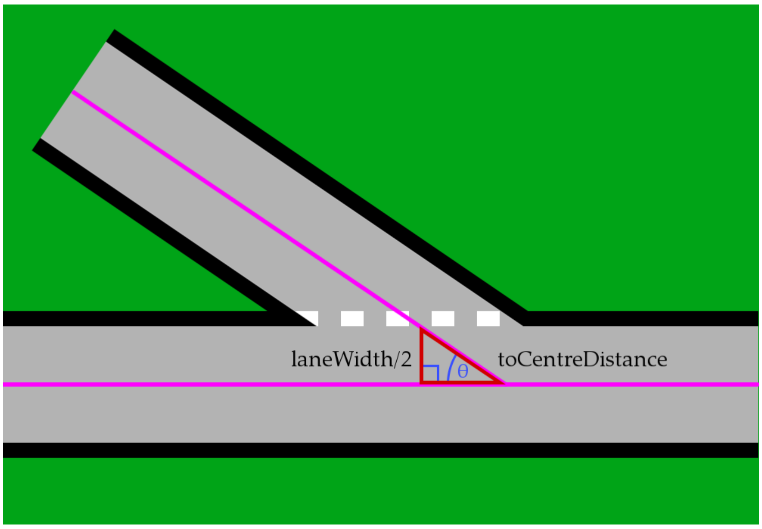



Figure A.4: A right-angled triangle used to calculate where the two centre lines meet. The centre lines are indicated in pink.

## A.3 Generalising the Codebase

### 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.5.

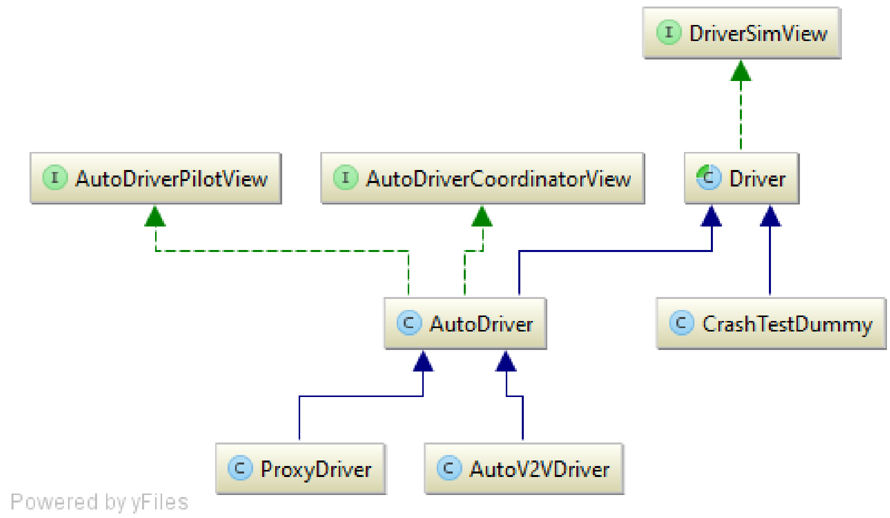



Figure A.5: The original class structure for *Driver*.

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.6 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* 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.

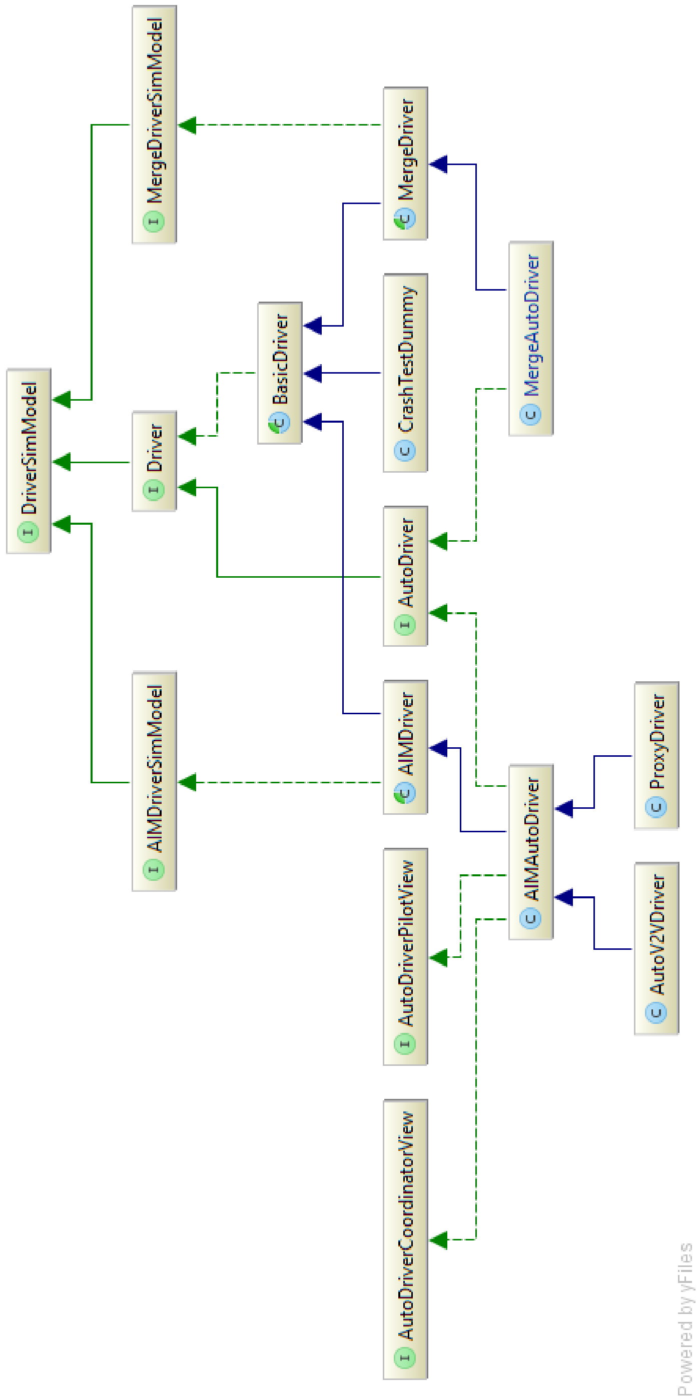



Figure A.6: The new class structure for *Driver*.

### 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. You can see the structural changes in Figures A.10 and A.11.

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.7.

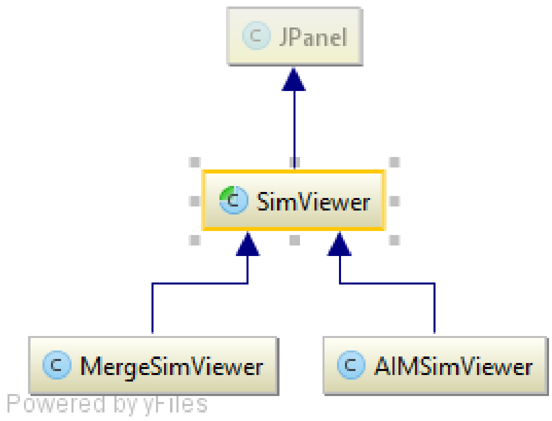



Figure A.7: The class diagram for *SimViewer*.

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.8 shows how *MergeStatScreen* and *Canvas* using *SimScreen*.

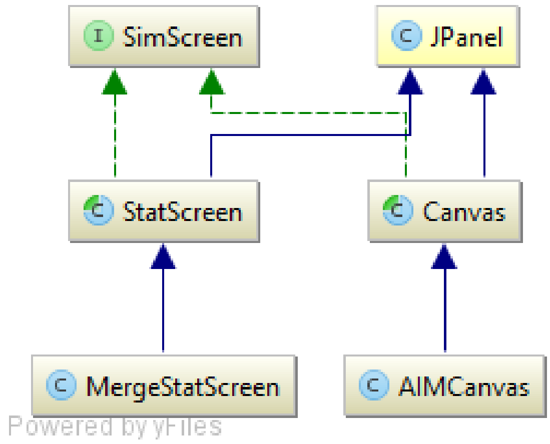



Figure A.8: The class diagram for *SimScreen*.

We also generalised the *SimSetupPanel* class to allow *SimViewer* to display non-AIM set-up controls. Figure A.9 shows the new class structure for *SimSetupPanel*.

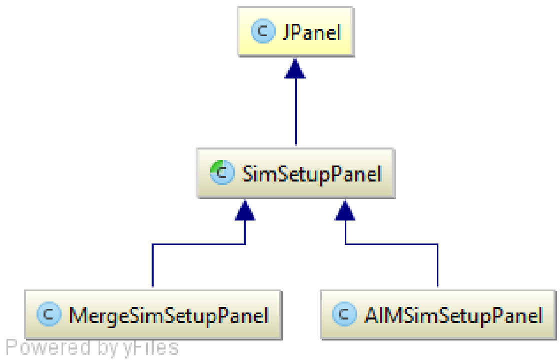



Figure A.9: The class diagram for *SimSetupPanel*.

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.

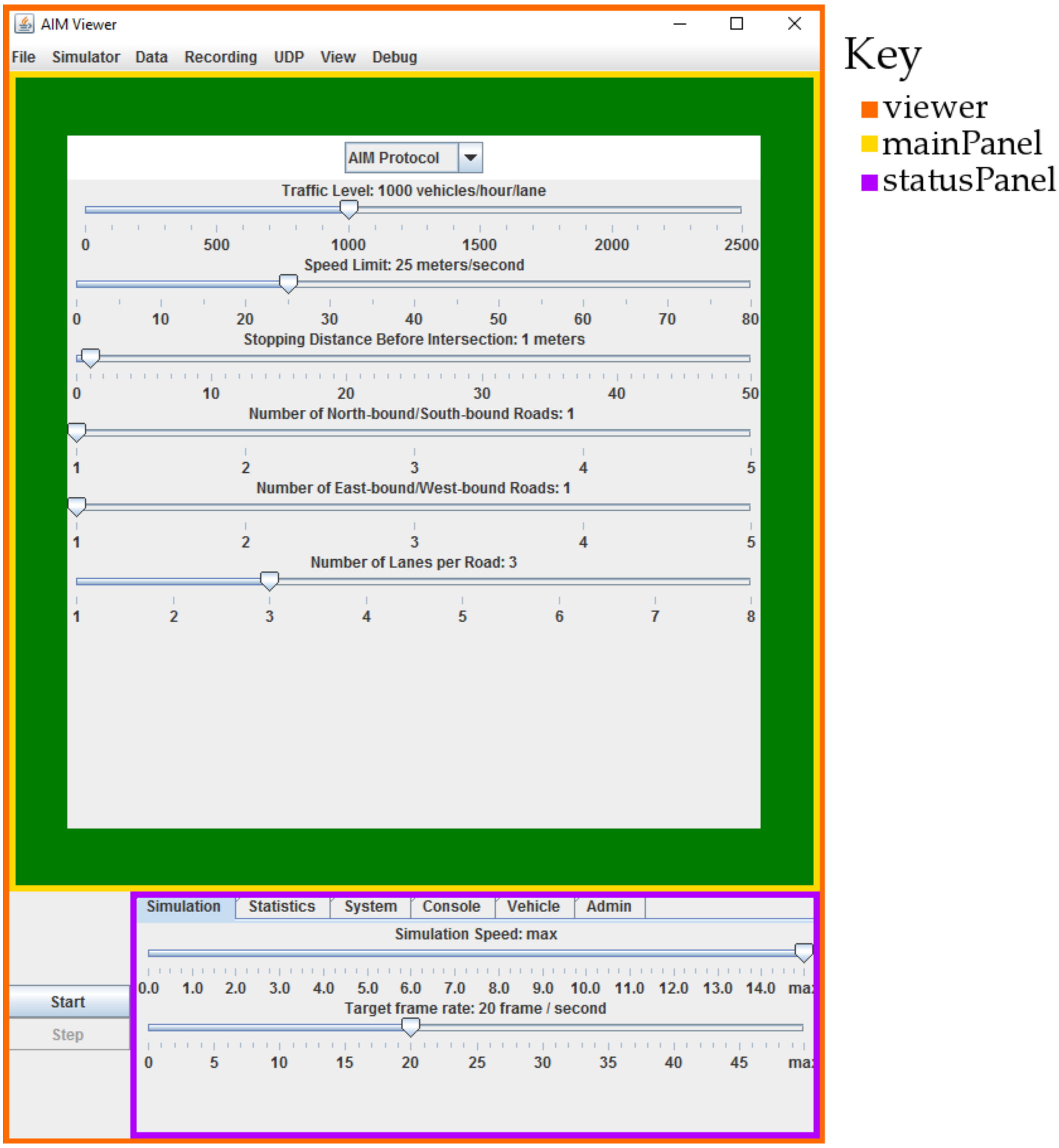



Figure A.10: Panel layout in the original simulator.

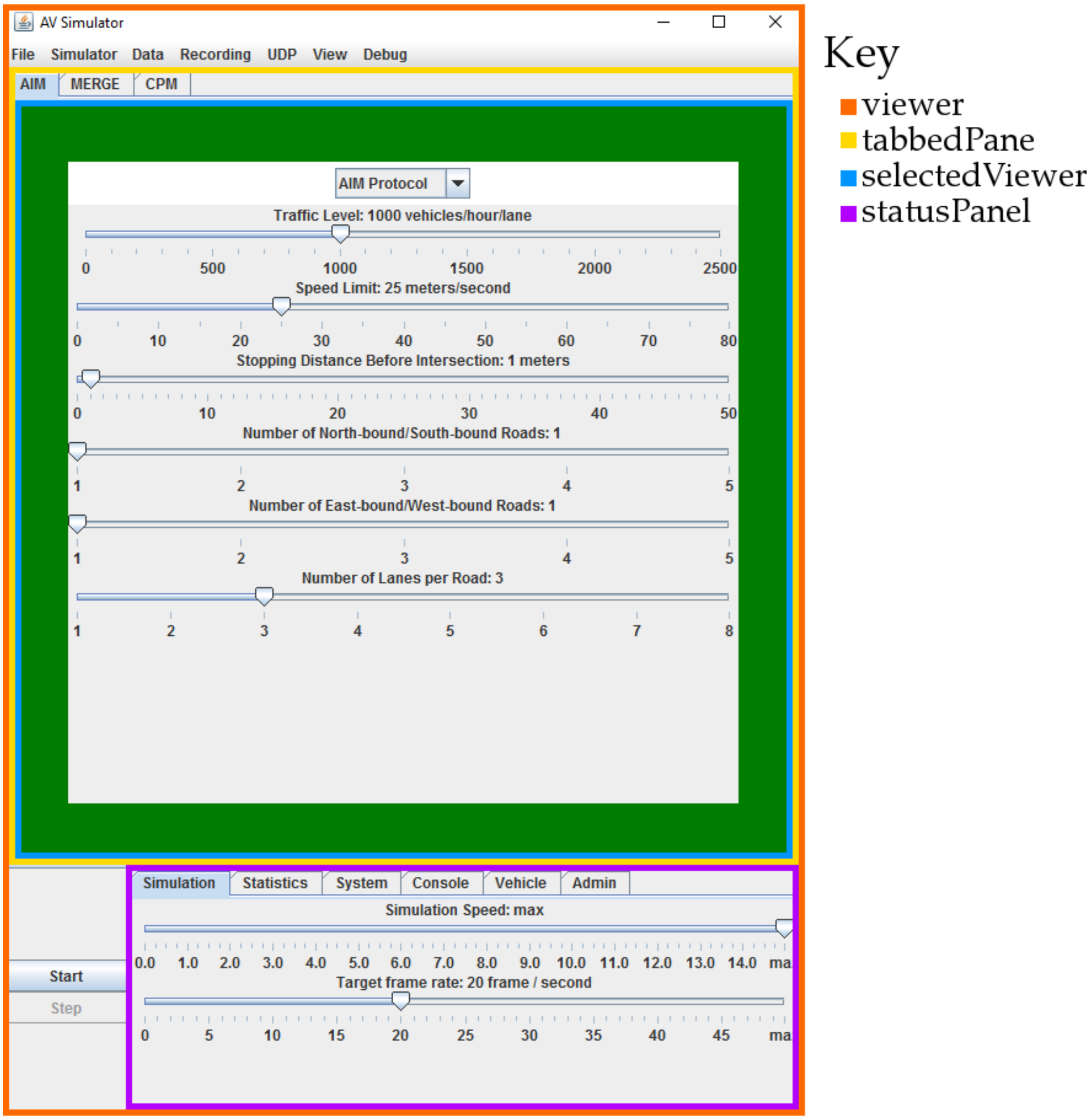



Figure A.11: Panel layout in the new simulator.

### 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.12 and A.13 show the original and new class structure for *aim4.map*.

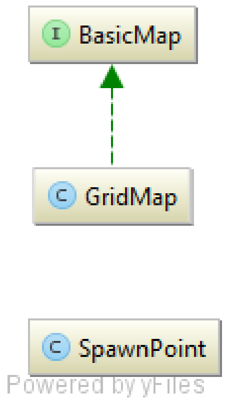
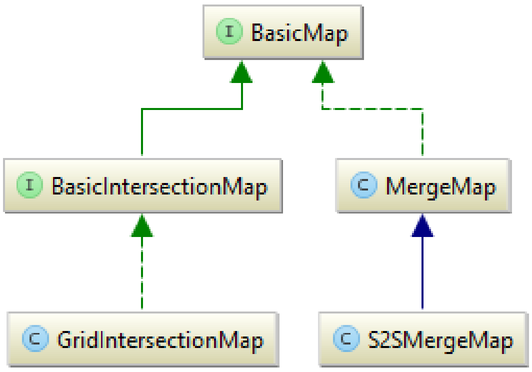



Figure A.12: The original class structure for *BasicMap* and *SpawnPoint*.

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



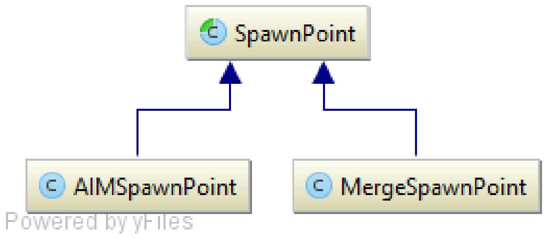




Figure A.13: The new class structure for *BasicMap* and *SpawnPoint*.

### 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.14 and A.15.

*SimSetup* was also modified to separate AIM specific set-up options and simulator creation code from other simulators. Figures A.16 and A.17 show how these classes were altered.

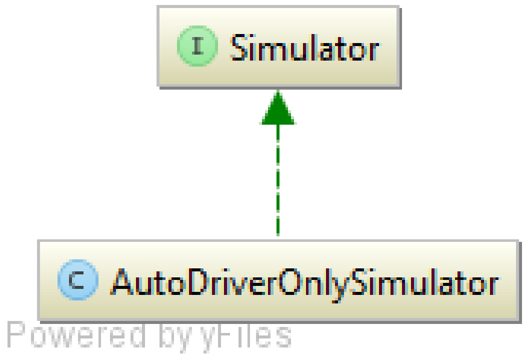



Figure A.14: The original class structure for *Simulator*.

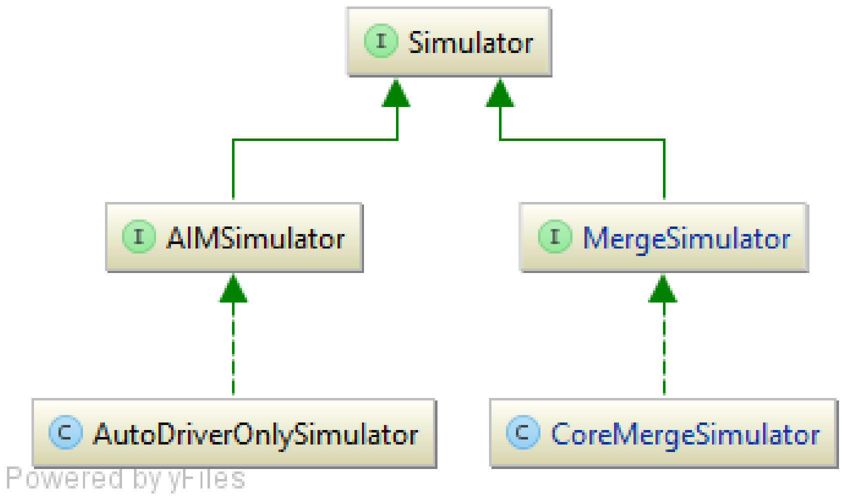



Figure A.15: The new class structure for *Simulator*.

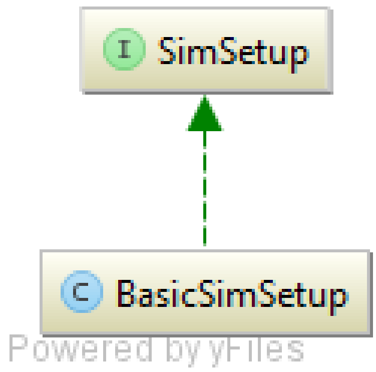



Figure A.16: The original class structure for *SimSetup*.

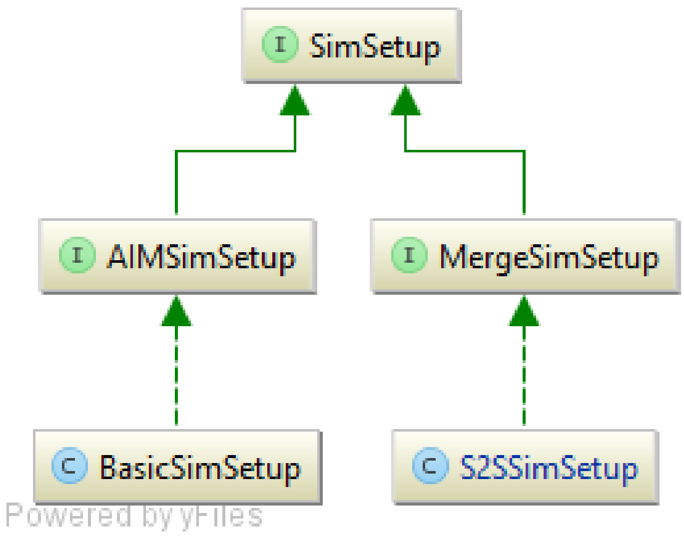



Figure A.17: The new class structure for *SimSetup*.

### 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.18 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.19 shows how the new structure links together.

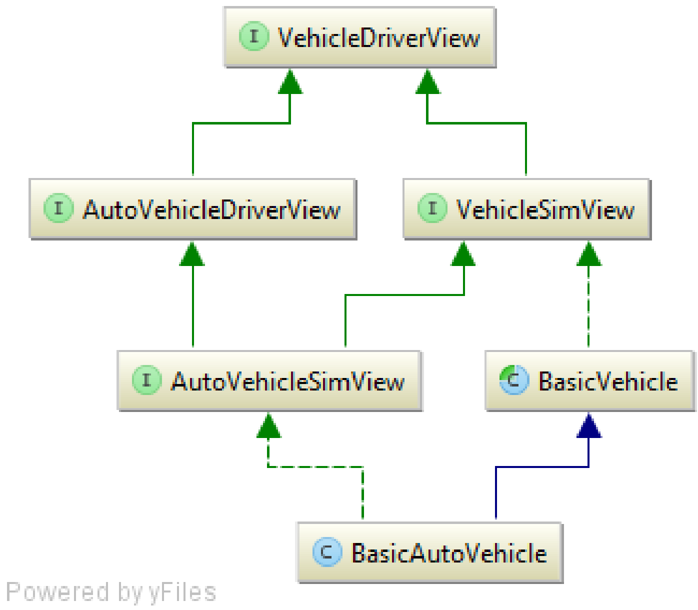



Figure A.18: The original class structure for *aim4.vehicle*.

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.

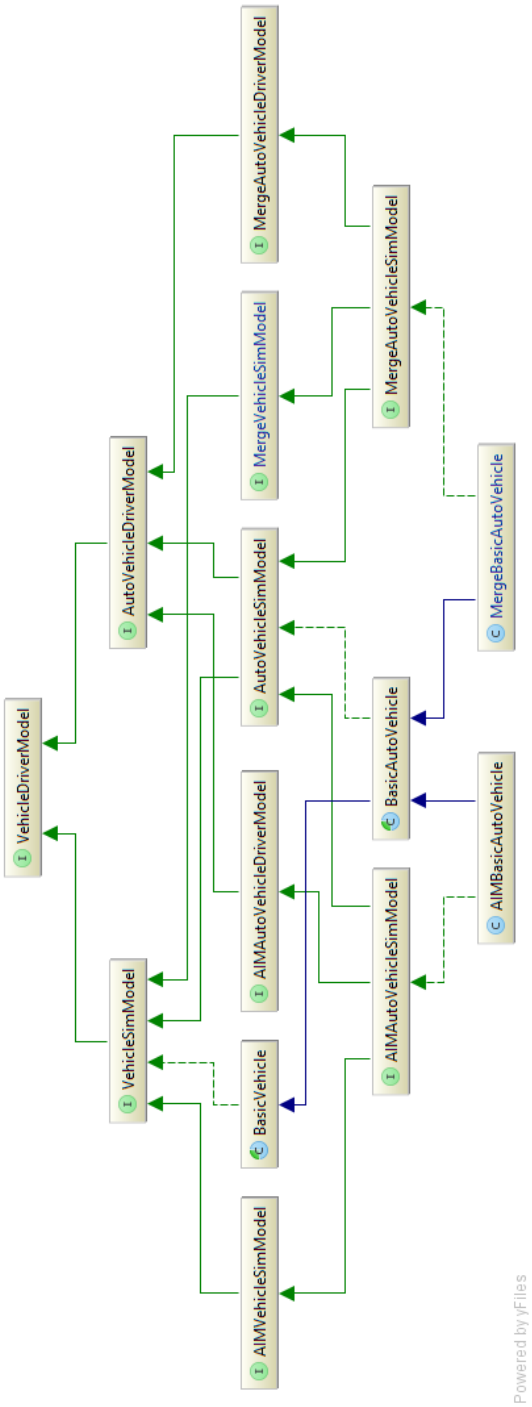



Figure A.19: The new class structure for *aim4.vehicle*.

Any future vehicles will need to create their own version of these interfaces, as seen in *MergeVehicleSimModel*, *MergeAutoVehicleDriverModel* and *MergeAutoVehicleSimModel*.

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).

## A.4 Maps

All maps testing Merge functionality implement *BasicMap*. I created a generalised implementation called *MergeMap* which satisfies the basic functionality of BasicMap as well as some protected accessors. All maps used during simulations extend *MergeMap*.

### A.4.1 MergeMapUtil

Similar to AIM's *GridMapUtil*, *MergeMapUtil* provides useful functions to *MergeMap*, including *SpawnPoint VehicleSpec* generators.