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

2016 Oct 12

Supervisor: Lilian Blot

Number of words = 2, as counted by `wc -w`.
This includes the body of the report only.

Abstract

Not completed

Not completed

Acknowledgements

Not completed

Contents

1 Introduction **12**

2 Literature Review **15**
2.1 Car Following Models 15
2.2 Centralised and Decentralised 18
2.2.1 Centralised Systems 19
2.2.2 Decentralised Systems 20
2.3 Making lane changing decisions 23
2.3.1 Lane Changing to hit a target lane 23
2.3.2 Lane Changing to improve overall velocity 26

3 Problem Analysis **32**
3.1 Merge Types 32
3.1.1 Single-to-Single Merge 32
3.1.2 Single-to-Single Merge with slip-road 33
3.1.3 Single-to-Double Merge 33
3.1.4 Single-to-Double Merge with slip-road 34
3.1.5 Double-to-Double Merge 34
3.1.6 Double-to-Double Merge with slip-road 35
3.1.7 Lane Obstruction Merge 35
3.2 Measuring Success 36
3.2.1 Collisions 37
3.2.2 Delay 38
3.2.3 Throughput 39
3.2.4 Velocity Changes 39
3.3 Merge Variance Factors 40
3.4 Requirements 42
3.4.1 Functional 43
3.4.2 Non-functional 46

4 Design **47**
4.1 Map 47
4.1.1 Target Lane Coordinates 47

4.1.2 Merging Lane Coordinates 49
4.2 Intersection Management System 50
4.3 Decentralised Communication System 50
4.4 Measuring Success 50

5 Implementation **53**
5.1 Generalising the Codebase 53
5.2 Merge Schemes 54
5.3 Simulation 54
5.3.1 Drivers 55
5.3.2 Merge Managers 56
5.3.3 Map 56
5.3.4 Simulation Control 57
5.4 Results Production 57
5.5 GUI 58
5.6 Maintainability and Testing 59
5.6.1 Unit Testing 59
5.6.2 Integration Tests 59

6 Results **60**

7 Results **61**

A Appendix **62**
A.1 S2S Map Calculations 62
A.2 Generalising the Codebase 66
A.2.1 aim4.driver 66
A.2.2 aim4.gui 67
A.2.3 aim4.map 68
A.2.4 aim4.sim 69
A.2.5 aim4.vehicle 69
A.3 Maps 70
A.3.1 MergeMapUtil 70

List of Figures

1.1 Diagram from Rule 126 in the UK Highway Code [**?**] . . . 12

2.1 Screenshot of the AIM Simulator 29
2.2 The flowchart for lane changing decisions from [**?**] 30
2.3 The flowchart for Atagoziyev's Lane Changing model equations 31

3.1 A road with a single-to-single lane merge (S2S) 33
3.2 A road with a single-to-single lane merge and slip-lane (S2S) 34
3.3 A real world single to triple lane merge with a slip-road. Image Copyright Nigel Cox. This work is licensed under the Creative Commons Attribution-Share Alike 2.0 Generic Licence. http://creativecommons.org/licenses/by-sa/2.0/ *TODO: Lilian, does this mean that I have to share my essay under Creative Commons? If so should I choose a different image?*[**?**] 35
3.4 A road with a single-to-double lane merge (S2D) 36
3.5 A road with a single-to-double lane merge and slip-lane (S2D) 37
3.6 A road with a double-to-double lane merge (D2D) 38
3.7 A road with a single-to-double lane merge and slip-lane (S2D) 39
3.8 A road with a lane obstruction 40
3.9 An S2S merge marked with some of the variable factors . 42

4.1 A diagram indicating the base width of the merging lane. 48
4.2 A diagram indicating the x and y adjustments for the merging lane. 51
4.3 A diagram indicating the X-adjustment used to find the point at which the two lanes connect. 52

5.1 Key for the class diagrams in this report. 53
5.2 Key for the class diagrams in this report. 55
5.3 The simulation screen for the S2S merge screen 59

A.1 A right-angled triangle used to calculate the size of the merging zone. 62
A.2 Two right-angled triangles used to calculate the x and y adjustments for the merge entrance. 63
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. 64
A.4 A right-angled triangle used to calculate where the two centre lines meet. The centre lines are indicated in pink. . 65
A.5 The original class structure for *Driver*. 66
A.6 The new class structure for *Driver*. 67
A.7 Panel layout in the original simulator. 71
A.8 Panel layout in the new simulator. 72
A.9 The class diagram for *SimViewer*. 73
A.10 The class diagram for *SimScreen*. 74
A.11 The class diagram for *SimSetupPanel*. 75
A.12 The original class structure for *BasicMap* and *SpawnPoint*. 76
A.13 The new class structure for *BasicMap* and *SpawnPoint*. . . . 77
A.14 The original class structure for *Simulator*. 78
A.15 The new class structure for *Simulator*. 78
A.16 The original class structure for *SimSetup*. 79
A.17 The new class structure for *SimSetup*. 80
A.18 The original class structure for *aim4.vehicle*. 81
A.19 The new class structure for *aim4.vehicle*. 81

List of Tables

3.1 Functional requirements table. 43
3.2 Non-functional requirements table. 46

1 Introduction

Autonomous vehicles used to exist solely in science fiction. In 1953, Isaac Asimov wrote Sally [**?**]; in 1982, Knight Rider introduced KITT [**?**]; but 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 [**?**]. Tesla have deployed their beta Autopilot system into all of their vehicles produced since September 2014. The system has been both hailed for saving lives and blamed for ending them [**?**] [**?**].

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.

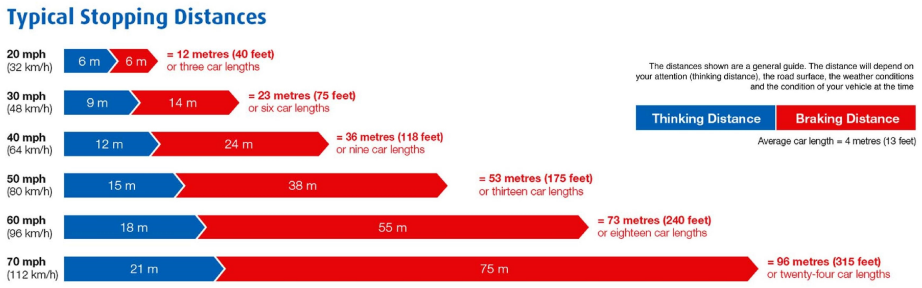

Figure 1.1: Diagram from Rule 126 in the UK Highway Code [**?**]

Another benefit of autonomous vehicles is efficiency. Research by Mer-

sky suggests that fuel conservation strategies could make autonomous vehicles up to 10% more fuel efficient than current EPA fuel economy test results [**?**]. Fuel efficient vehicles are becoming increasingly important, with landmark climate change deals such as 'The Paris Agreement' introducing limits on greenhouse gas emissions globally [**?**]. 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.

Congestion contributes to fuel loss 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 [**?**]. 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 a traffic jam. This ripple effect is known as a 'traffic wave' or 'traffic shock' [**?**].

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 testing yet [**?**].

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 multiagent framework for managing autonomous vehicles at intersections" [**?**]. The team managed to apply their simulator tested intersection software in a mixed reality test, using a real life autonomous vehicle [**?**]. This demonstrates how vital simulators are when testing safety critical systems.

In this project we make 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.

The aims of this project are as follows:

- Attempt to generalise the AIM codebase such that other simulators can be created for non-intersection related situations.
  - Create a decentralised system for managing lane merging.
  - Create a centralised system for managing lane merging.
- Compare the effectiveness of both strategies.

2 Literature Review

2.1 Car Following Models

Any autonomous-vehicle system will be based on 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 [**?**]. It was designed to mimic real-world driver behaviour, calculating a safe traveling 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' paper defines two equations, which provide constraints on the speed of vehicle n at time $t + \tau$. t is the current time and τ 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.

$$v_n(t+\tau) \leqslant v_n(t) + 2.5 a_n \tau \left(\frac{1 - v_n(t)}{V_n}\right)\left(\frac{0.025 + v_n(t)}{V_n}\right)^{1/2} \tag{2.1}$$

$v_n(t)$ 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

$$v_n(t+\tau) \leqslant b_n\tau + \sqrt{\left(b_n^2\tau^2 - b_n\left(2\Big[x_{n-1}(t) - s_{n-1} - x_n(t)\Big] - v_n(t)\tau - \frac{v_{n-1}(t)^2}{\hat{b}}\right)\right)} \tag{2.2}$$

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.

$$v_n(t) = \min\left((2.1), (2.2)\right) \tag{2.3}$$

This model works 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. An autonomous vehicle with perfect sensors would have an almost negligible τ, as the vehicles would have a very minimal reaction time. s_n would also need to be adjusted. The margin added can be much less, as autonomous vehicles are more precise than human drivers and can drive closer to their predecessors.

The model also ignores inter-vehicle communication. Autonomous vehicles could communicate their intentions to nearby vehicles, allowing them to act before they do. This could greatly reduce the following distance of successor vehicles, it would also allow vehicles to accelerate and move as a unit, or a 'platoon'. It would also allow autonomous vehicles to gain accurate values for $\hat{b}$.

In 2000 Treiber et al. suggested the 'Intelligent Driver Model' (IDM) [**?**]. In the IDM, the acceleration of vehicle α, $\dot{v}_\alpha$, is defined using a continuous function of its velocity, v_α; the distance to the rear of its predecessor, s_α; and the velocity difference of α and it's predecessor, also known as the approaching rate Δv_α. The vehicle interactions are solely based on α'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.

$$\dot{v_\alpha} = a^{(\alpha)} \left[1 - \left(\frac{v_\alpha}{v_0^{(\alpha)}} \right)^{\delta} \right] \tag{2.4}$$

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

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

$$\dot{v_\alpha} = -a^{(\alpha)} \left(\frac{s^*}{s_\alpha} \right)^2 \tag{2.5}$$

As the gap, s_α, between α and it's predecessor, gets closer to the desired minimum gap s^*, α decelerates.

Interpolating the two components gives us the IDM.

$$\dot{v_\alpha} = a^\alpha \left[1 - \left(\frac{v_\alpha}{v_0^\alpha} \right)^{\delta} - \left(\frac{s^*(v_\alpha, \Delta v_\alpha)}{s_\alpha} \right)^2 \right] \tag{2.6}$$

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

$$s^*(v, \Delta v) = s_0^{(\alpha)} + s_1^{(\alpha)} \sqrt{\frac{v}{v_0^{(\alpha)}}} + T^\alpha v + \frac{v \Delta v}{2\sqrt{a^{(\alpha)} b^{(\alpha)}}} \tag{2.7}$$

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 α 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 α.

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 [**?**]. However, similarly to Gipps' model, the standard IDM only applies to single lane traffic. It also ignores inter-vehicle communication and platooning opportunities.

In vehicle platoons, such as those analysed by Kamali in 2016 [**?**], each vehicle autonomously follow 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.

Kamali developed a model for an automated platoon, defining procedures for vehicles joining and leaving.

A joining vehicle can integrate at either the back or the middle of the platoon. The vehicle first sends a join request to the platoon leader. If the vehicle is at the back of the platoon the leader sends an agreement and the vehicle follows its predecessor. If the vehicle requests to join in front of another platoon vehicle, the leader first asks the platoon vehicle to increase space; once the space is large enough for the joining vehicle, the leader sends an agreement. The joining vehicle then manoeuvres into the space and follows the preceding vehicle. Having now joined the platoon, the vehicle sends a confirmation to the leader. The leader then requests that the vehicle that gave way for the joining vehicle decreases their spacing back to normal.

A leaving vehicle sends a request to the leader. When it receives permission to leave the vehicle increases its spacing from its predecessor; once the vehicle is at its maximum distance from its predecessor the vehicle can change lanes. Once out of the convoy the vehicle sends an acknowledgement to the leader.

This model isn't very strict, acting as more of a set of requirements than a true model. The paper sets the requirements using pre-defined gaps, and has no strict calculations guiding following characteristics. It could be implemented using spacing rules from both the IDM and Gipps' model, however, by using V2V communication, the lead vehicle can control the actions of all vehicles in its platoon. Instead of using IDM or Gipps' model, the lead vehicle can control the gaps between vehicles so that they all increase and decrease simultaneously. The gaps could be based on the platoon's velocity, perhaps using (2.7) from the IDM. By centralising control in this way, vehicle platoons can avoid the traffic shock effect [**?**].

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 Vehicle Intersection management system (AIM) described in [**?**] 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.

A centralised system for lane changing was described in a paper by Atagoziyev et al. in 2016 [**?**]. This system uses roadside infrastructure to help groups of vehicles change lanes before they reach a 'critical-position', such as a motorway exit or intersection. The vehicles send their position and velocity information to the roadside infrastructure; the system then sends a number of orders to the vehicles such that they safely rearrange themselves into the correct lanes. Because the distance travelled by the vehicles involved can be large, particularly at high speeds, the system would struggle to use a reservation tile based system as AIM did, instead Atagoziyev's system manages the gaps between vehicles to safely relocate vehicles into the correct position. A more comprehensive overview is given in 2.3.1.

Atagoziyev's system would only need to be applied during the approach to critical-positions. Vehicles could be managed using platoons or another vehicle following model until that point. A centralised system provides a single communication point which manages all of the vehicles that want to change lanes. This helps to reduce the volume of communications required and creates an entity with a global view of the vehicles' positions and goals. A V2V solution would most likely require more communications and may never obtain a complete picture of the situation, possibly leading to sub-optimal lane changing orders.

Note that Atagoziyev's system could be adapted, such that all of the vehicles communicate with the platoon leader instead of roadside infrastructure. Though this could be called a V2V communication solution, the effective solution is still considered centralised, as all decisions are made by one entity.

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 (SPOF) like this is a major concern, particularly when lives are on the line.

A paper by Van Middlesworth et al. in 2008 [**?**] defined a decentralised

version of the AIM model using V2V communication protocols.

In Van Middlesworth 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 Van Middlesworth defines a priority ordering over claim messages using the following rules:

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 tries to make a reservation using a CLAIM message. The vehicle will continue generating CLAIM messages, searching for an available time block, increasing and decreasing

its velocity as necessary, until it finds one that allows it to traverse the intersection without any other CLAIM having priority over it.

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 arrives with a higher priority. 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. 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 has the highest priority CLAIM and cannot be interrupted.

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 Van Middlesworth 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 [**?**] and the MOBIL model developed by Kesting et al. in 2007 [**?**]. 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 [**?**]. The paper was designed to be used with the Gipps' 1981 car-following model [**?**], 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

$$b_n = \left[2 - V_n \frac{(D_n - x_n(t))}{10}\right] b_n^* \tag{2.8}$$

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 [**?**] 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}.

$$\tilde{a}_n \geq -b_{safe} \tag{2.9}$$

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:

$$\underbrace{\tilde{a}_c - a_c}_{\text{driver}} + p\Big(\underbrace{\tilde{a}_n - a_n}_{\text{new follower}} + \underbrace{\tilde{a}_o - a_o}_{\text{old follower}}\Big) > \Delta a_{th} \tag{2.10}$$

$\tilde{a}_x - a_x$ 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. Δa_{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

$$\tilde{a}_c + \tilde{a}_n + \tilde{a}_o > a_c + a_n + a_o \tag{2.11}$$

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.

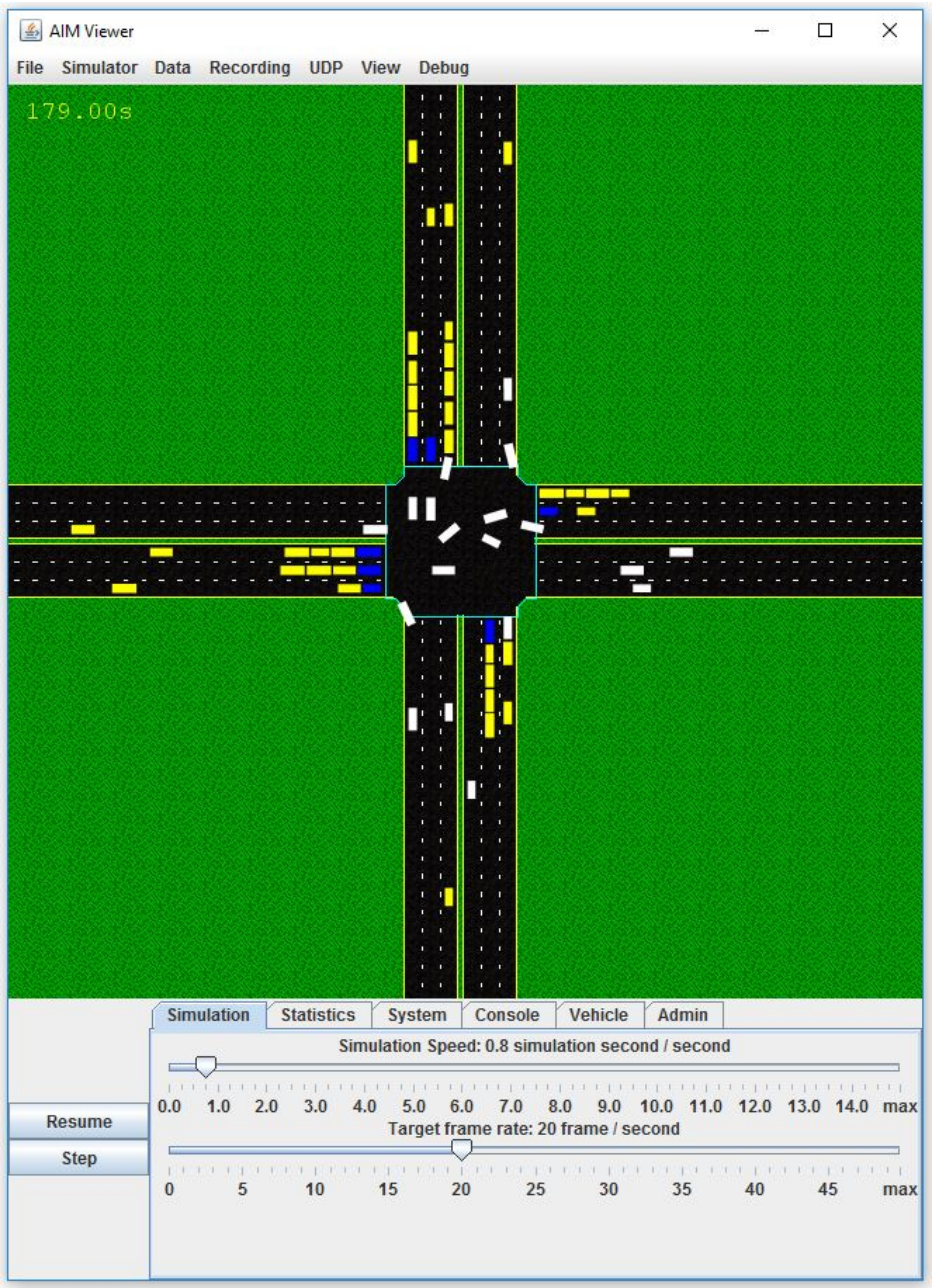

Figure 2.1: Screenshot of the AIM Simulator

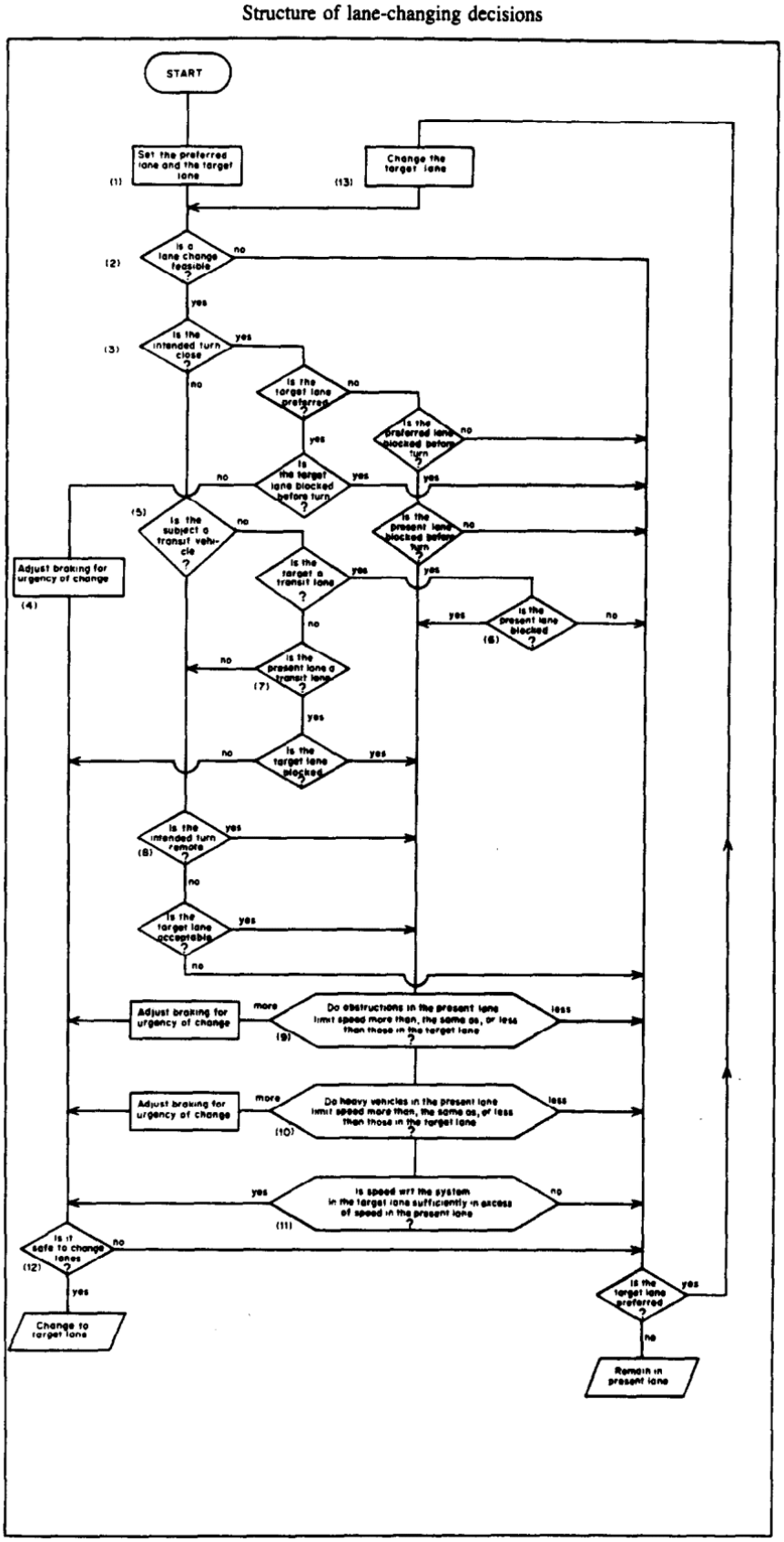

Figure 2.2: The flowchart for lane changing decisions from [**?**]

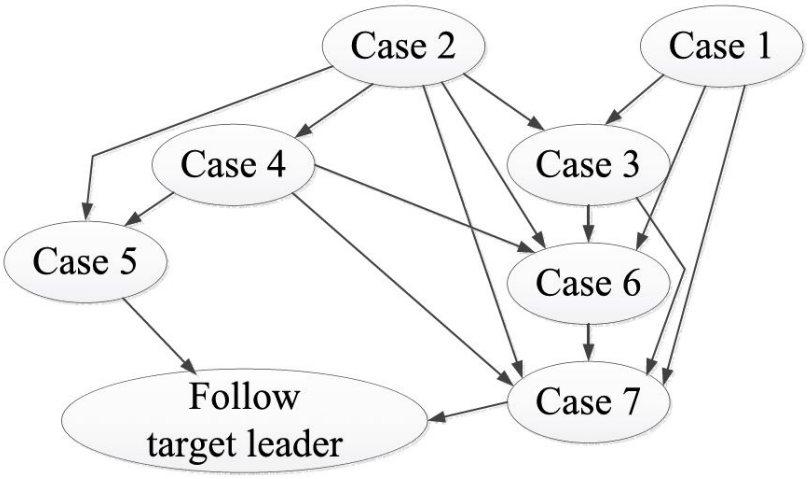

Figure 2.3: The flowchart for Atagoziyev's Lane Changing model equations

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 3.1. 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.

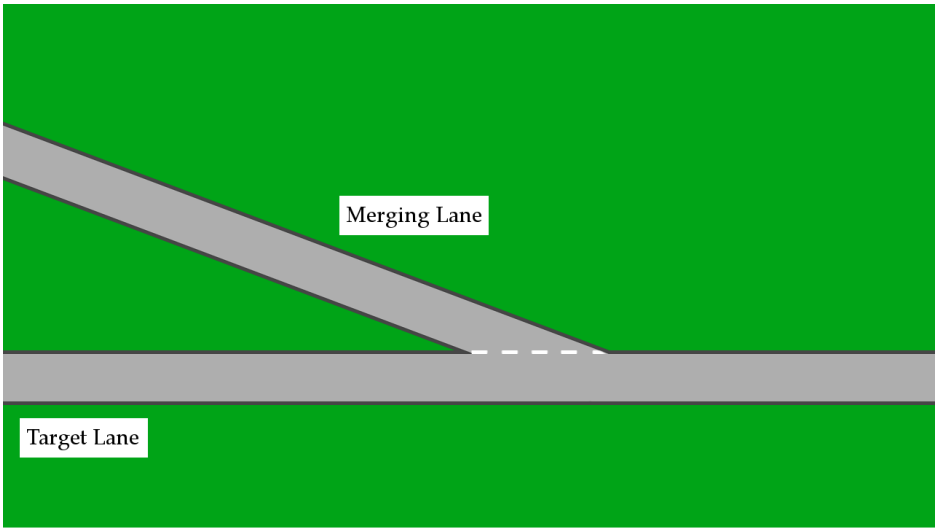

Figure 3.1: A road with a single-to-single lane merge (S2S)

3.1.2 Single-to-Single Merge with slip-road

Many S2S merges are performed with an attached slip-road, as seen in Figure 3.2. Figure 3.3 shows a real world single to triple lane merge with a slip-road.

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.

3.1.3 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

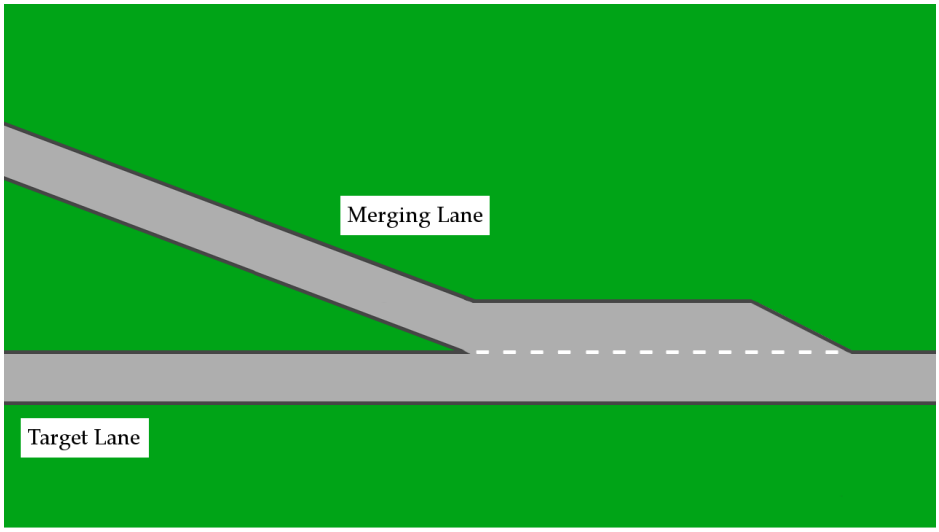

Figure 3.2: A road with a single-to-single lane merge and slip-lane (S2S)

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.

3.1.4 Single-to-Double Merge with slip-road

S2D merges can also take advantage of a slip-road, as seen in Figure 3.5.

3.1.5 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.

Figure 3.3: A real world single to triple lane merge with a slip-road. Image Copyright Nigel Cox. This work is licensed under the Creative Commons Attribution-Share Alike 2.0 Generic Licence. http://creativecommons.org/licenses/by-sa/2.0/ *TODO: Lilian, does this mean that I have to share my essay under Creative Commons? If so should I choose a different image?*[**?**]

3.1.6 Double-to-Double Merge with slip-road

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.

3.1.7 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

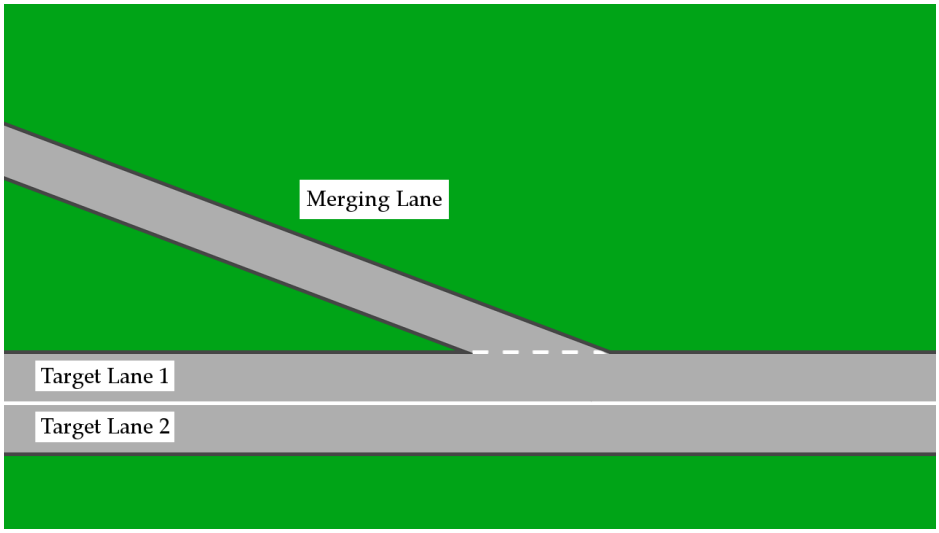

Figure 3.4: A road with a single-to-double lane merge (S2D)

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.

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

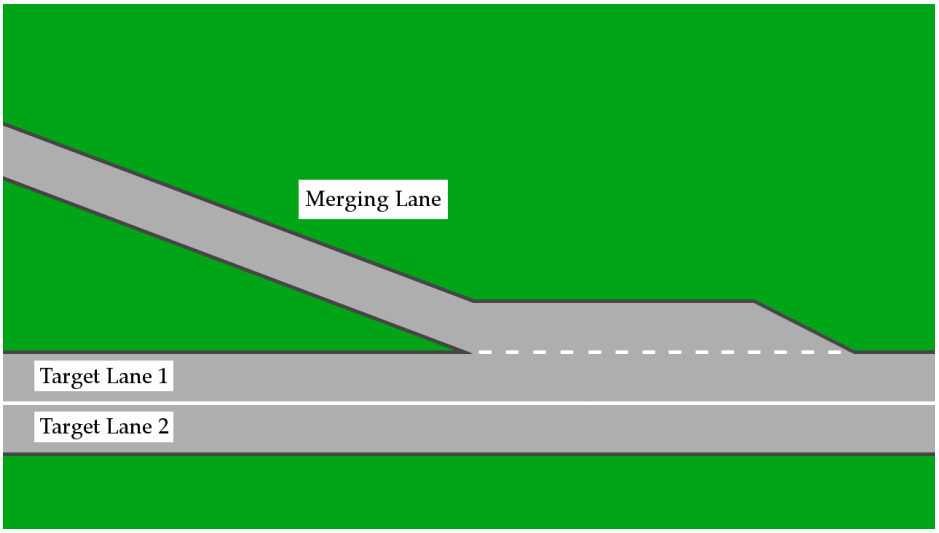

Figure 3.5: A road with a single-to-double lane merge and slip-lane (S2D)

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 [**?**].

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

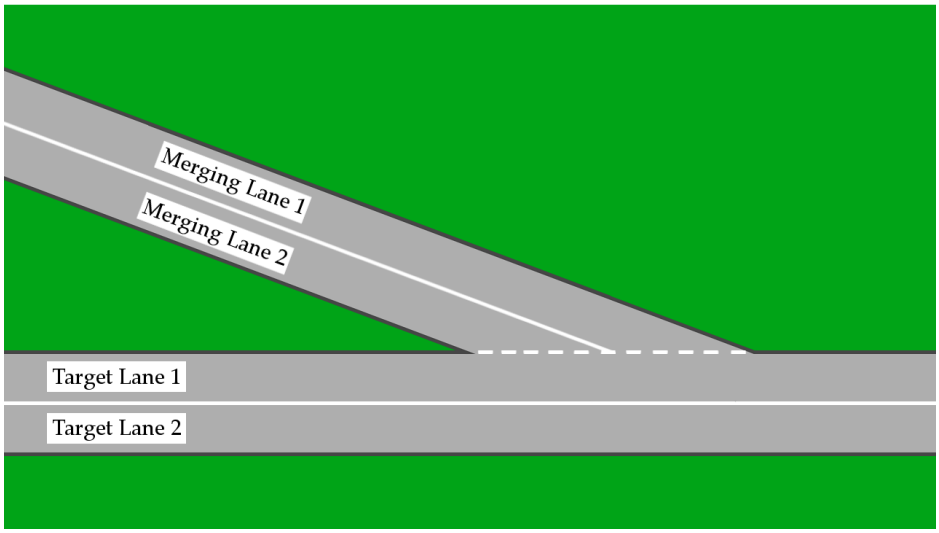

Figure 3.6: A road with a double-to-double lane merge (D2D)

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 [**?**] 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:

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

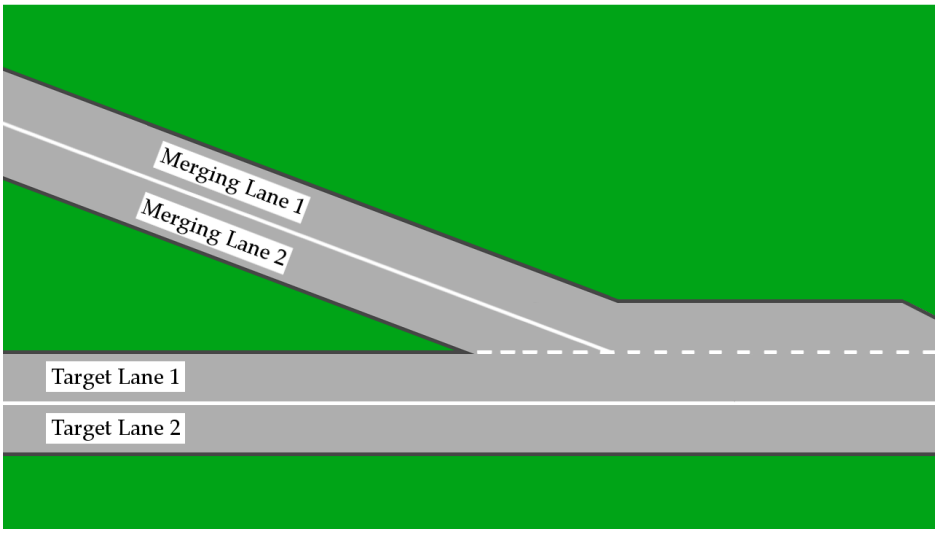

Figure 3.7: A road with a single-to-double lane merge and slip-lane (S2D)

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

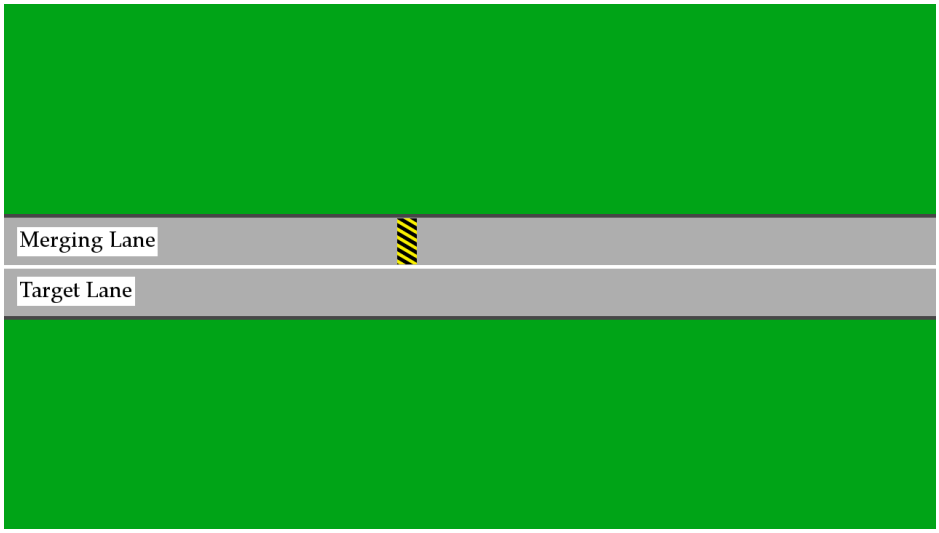

Figure 3.8: A road with a lane obstruction

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

ALL NEW TEXT BELOW

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 θ at the point where the merging lane meets the target lane.
- *Slip-road length* The length of the slip-road in a merge that uses a slip-road.

Traffic level has a fairly obvious effect on performance in a particular merge scenario. The more vehicles that try to merge together the more difficult it will be for the merge to happen. It will likely require vehicles to 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

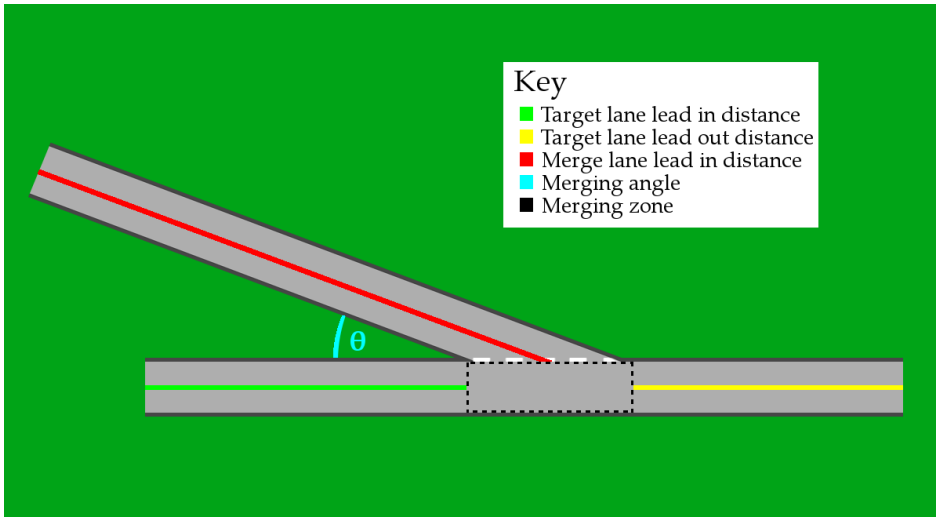

Figure 3.9: An S2S merge marked with some of the variable factors

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.

User Requirements		System Requirements	
FU.1	Users can run merge simulations.	FS.11	The system has controls for starting merge simulations
		FS.12	The system can create merge simulations
		FS.13	The system can run merge simulations
FU.2	User can manipulate the running speed of a simulation.	FS.21	The system has controls changing the speed of running simulations.
		FS.22	Merge simulations can have their run speed changed.
FU.3	User can pause simulations.	FS.31	The system has controls 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.

User Requirements		System Requirements	
		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.
		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.

User Requirements		System Requirements	
		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 [**?**].
		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.

User Requirements		System Requirements	
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

All new text 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.1.

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.

$$width = targetLeadInDistance + mergeZoneWidth + targetLeadOutDistance \tag{4.1}$$

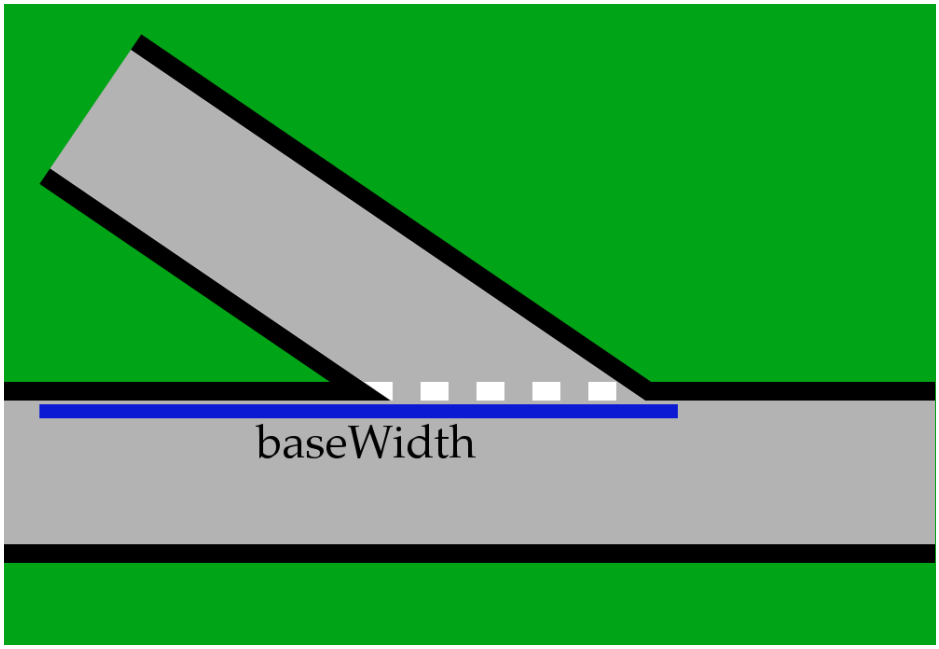

Figure 4.1: A diagram indicating the base width of the merging lane.

$$width = mergeBaseWidth + targetLeadOutDistance \quad (4.2)$$

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

$$targetLaneStartX = width - (targetLeadInDistance + mergeZoneWidth + targetLeadOutDistance) \quad (4.3)$$

The target lane exit coordinates can be calculated similarly. The X-coordinate will be equal to the width of the simulator. With these coordinates 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.

$$targetLaneZoneEntranceX = width - (mergeZoneWidth + targetLeadOutDistance) \quad (4.4)$$

$$targetLaneZoneExitX = width - targetLeadOutDistance \quad (4.5)$$

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.

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.

$$mergeLaneZoneEntranceX = width - targetLeadOutDistance - \frac{mergeZoneWidth}{2} \quad (4.6)$$

After the merge vehicles enter the merge zone they will deviate from the lane, however the merge lane centre does continue 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

$$connectionPointX = width - targetLeadOutDistance - \frac{mergeZoneWidth}{2} + centreXAdjustment \quad (4.7)$$

.

4.2 Intersection Management System

4.3 Decentralised Communication System

4.4 Measuring Success

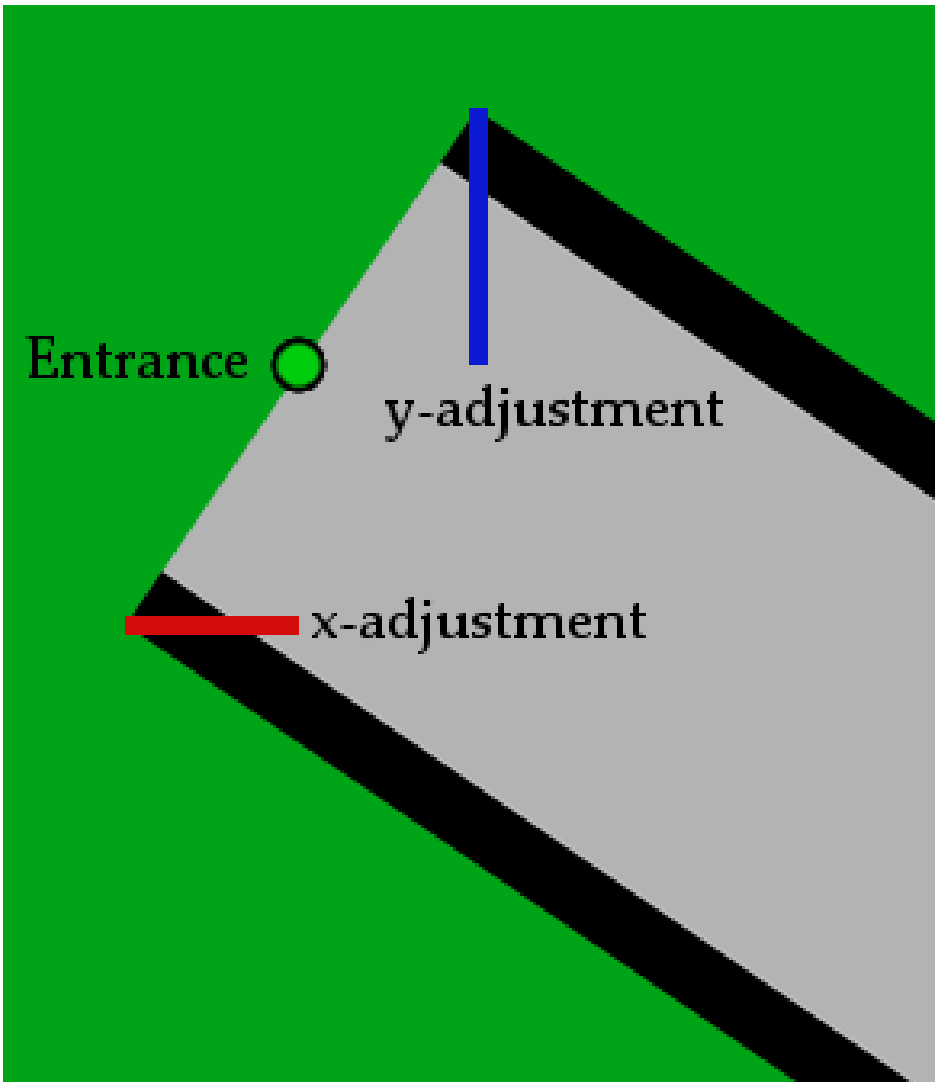

Figure 4.2: A diagram indicating the x and y adjustments for the merging lane.

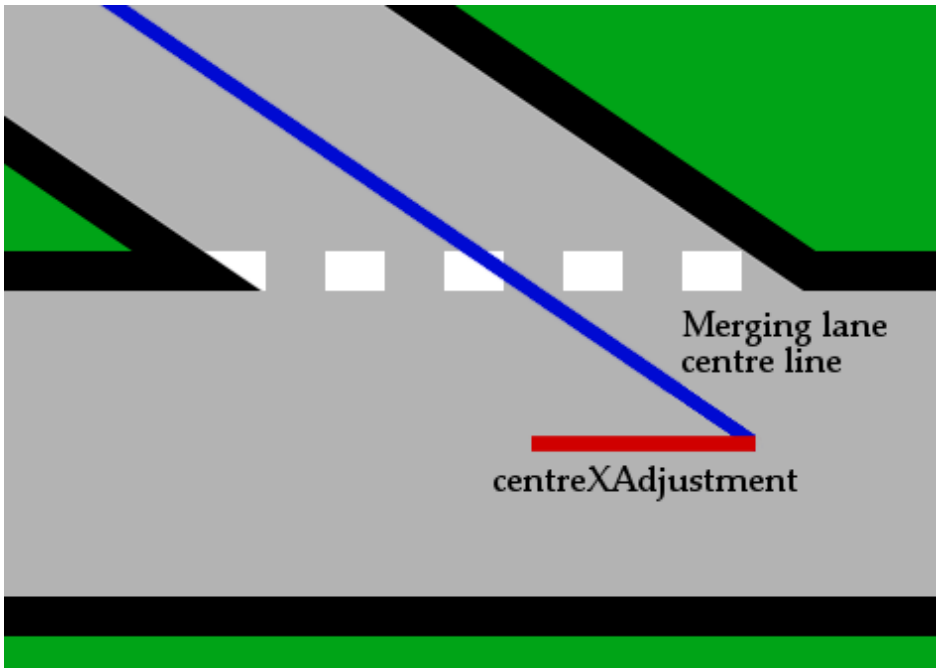

Figure 4.3: A diagram indicating the X-adjustment used to find the point at which the two lanes connect.

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.

5.1 Generalising the Codebase

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

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' [**?**], 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.

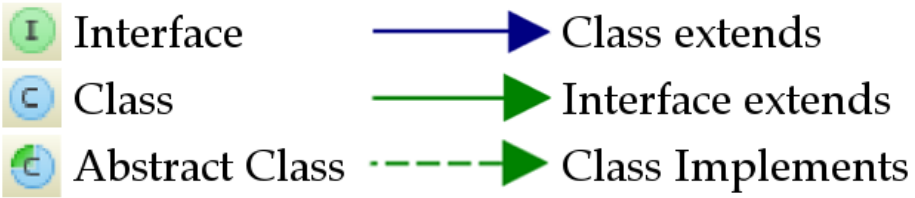

Figure 5.1: Key for the class diagrams in this report.

TODO: Decide what to put here Appendix A.2 provides detailed coverage

of both this change, and changes made to some of the other areas of the AIM codebase.

5.2 Merge Schemes

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

The original plan was to implement two merge schemes, an AIM based centralised protocol and a dentralised approach based on Replacing the Stop Sign: Unmanaged Intersection Control for Autonomous Vehicles [**?**].

- AIM – Failed to implement through copying code samples -> Far too complex. – Vehicles were not arriving on time and were also colliding – System also seemed like it would slow down the vehicles too much – Describe some implementation

- Queue – Response to AIM's failings – Describe how it was implemented

- Decentralised – Never implemented due to time constraints.

5.3 Simulation

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	✓

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 5.2 shows how these turns work.

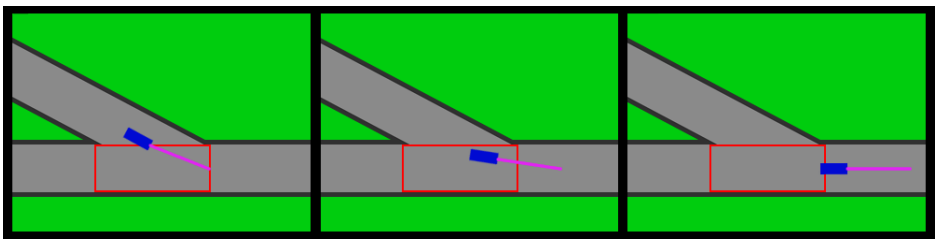

Figure 5.2: Key for the class diagrams in this report.

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.

5.4 Results Production

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

- Results generation to CSV file - Drivers, spawners and simulations adapted to provide the relevant export information - Setup in AIM as well

5.5 GUI

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	✓

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 that have left the simulation. Figure 5.3 shows the simulation screen for S2S merges.

– Consistent testing upload

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

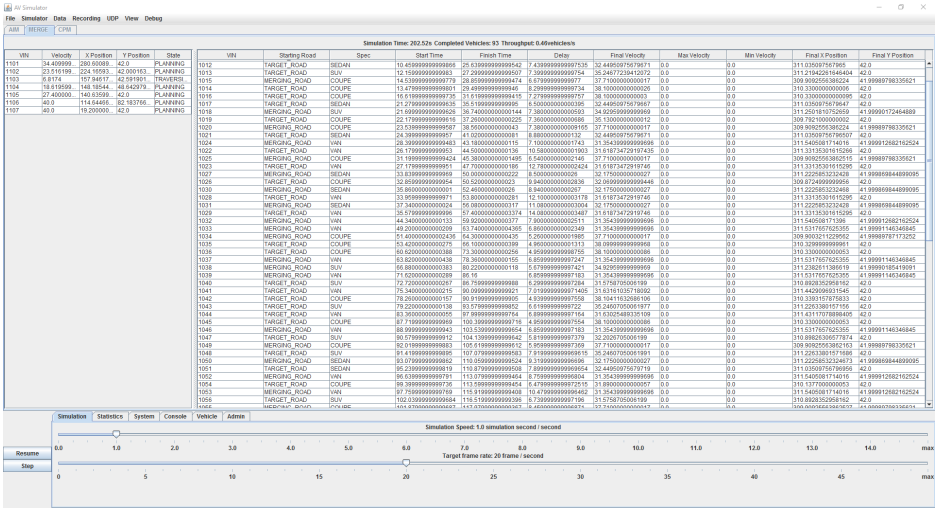

Figure 5.3: The simulation screen for the S2S merge screen

due to the removal of collision detection from simulations. All other GUI requirements were completed.

5.6 Maintainability and Testing

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

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

5.6.2 Integration Tests

6 Results

Lilian Notes:

1. It will be interesting to have a graph with |C| on the x-axis and throughput on y-axis. Similarly, a 3D graph where |Ctl| on the x-axis, |Cml| on the y-axis, and throughput on z-axis.

Callum Notes:

1. As a "things I'd do differently" or "changes to make" it would be cool to move to a Spring implementation. It might make it easier to add different simulator types.

7 Results

- Possible research : Behaviour of platoons and merging systems for them

A Appendix

A.1 S2S Map Calculations

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

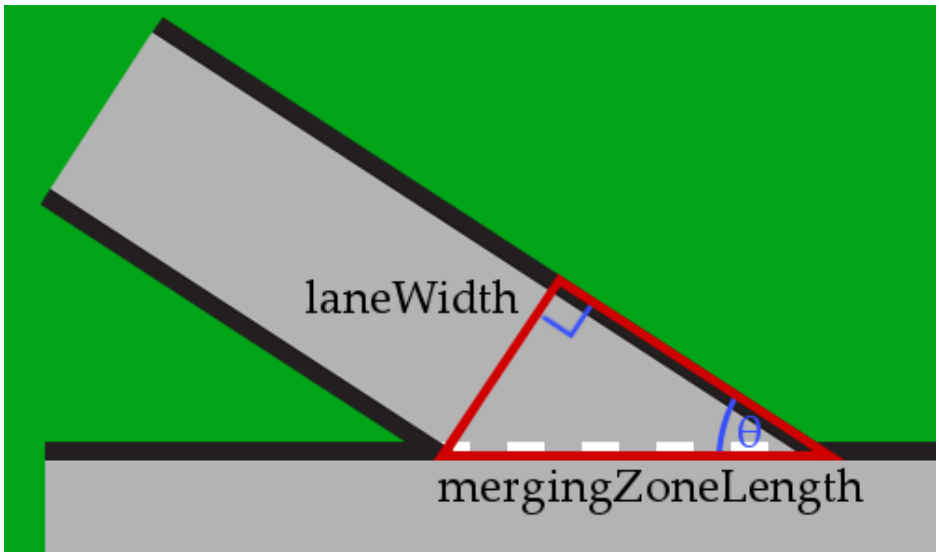

Figure A.1: A right-angled triangle used to calculate the size of the merging zone.

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

We also need to know whether the horizontal width of the merge lane on the map, or it's 'base width' is longer than the target lane's lead in

distance, plus the merge zone length. This will determine the width of the overall map, as if the merge lane's base length is longer then the target lane will not start with co-ordinate $x = 0$ as it would if the target lane determined the width of the map.

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

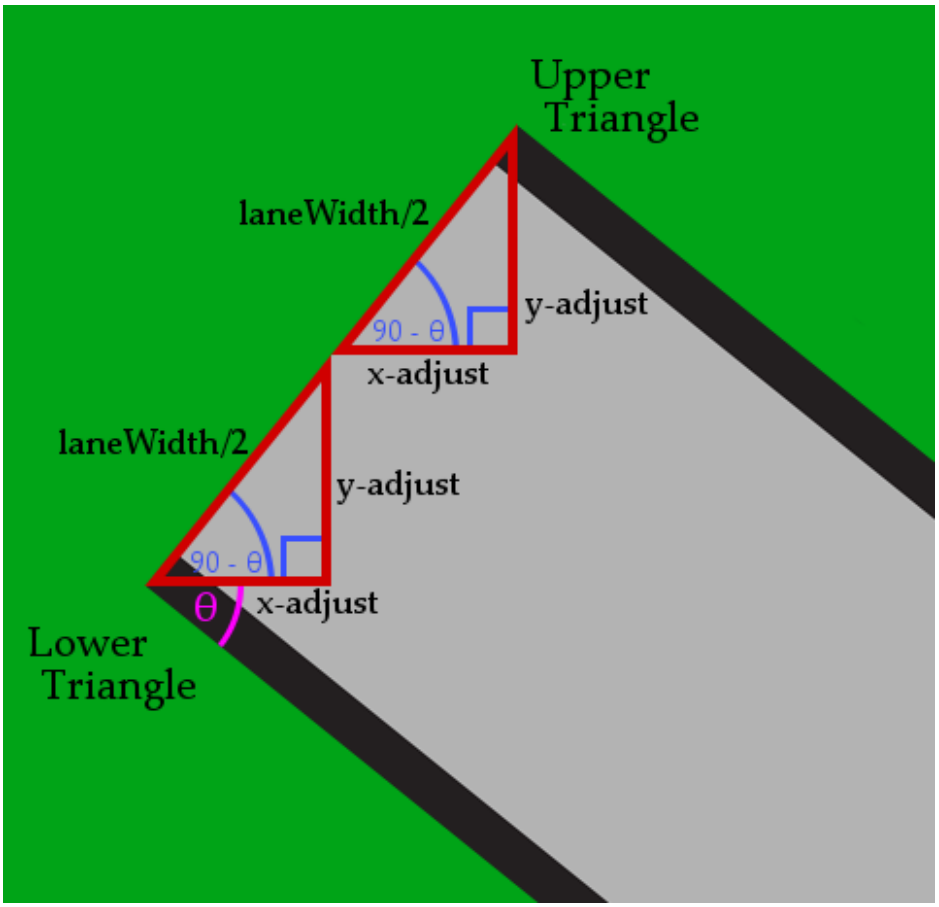

Figure A.2: Two right-angled triangles used to calculate the x and y adjustments for the merge entrance.

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

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

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

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

To calculate the 'base width' of the merge lane we will also need to calculate the adjacent side of the triangle in Figure A.3. In this triangle the hypotenuse has a length equal to the merge lead in distance. Therefore, we can use equation A.4 to calculate the length of the adjacent side. After obtaining the length of this side we simply add the merge entrance X-adjustment and half the length of the merge zone to find the merge base width.

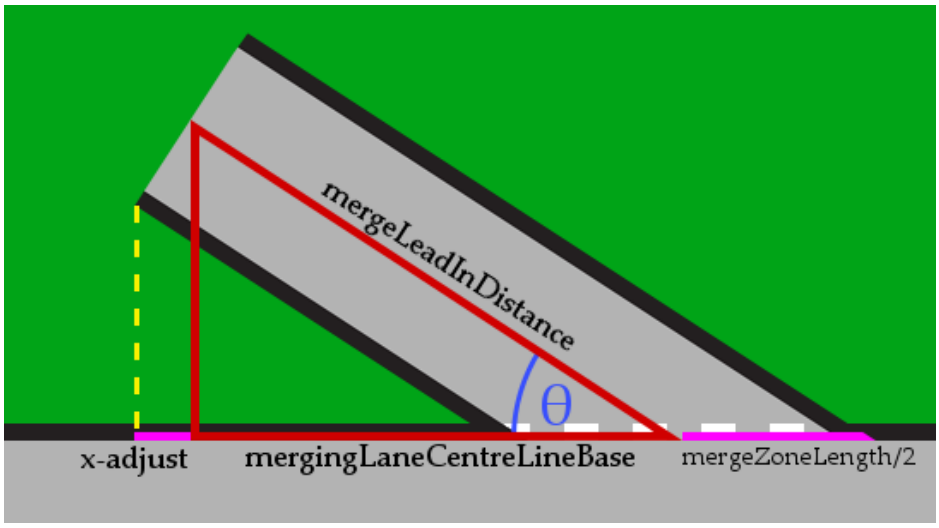

Figure A.3: A right-angled triangle used to help calculate the base width of the merge lane, along with the X-adjustment and merge-zone length.

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

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

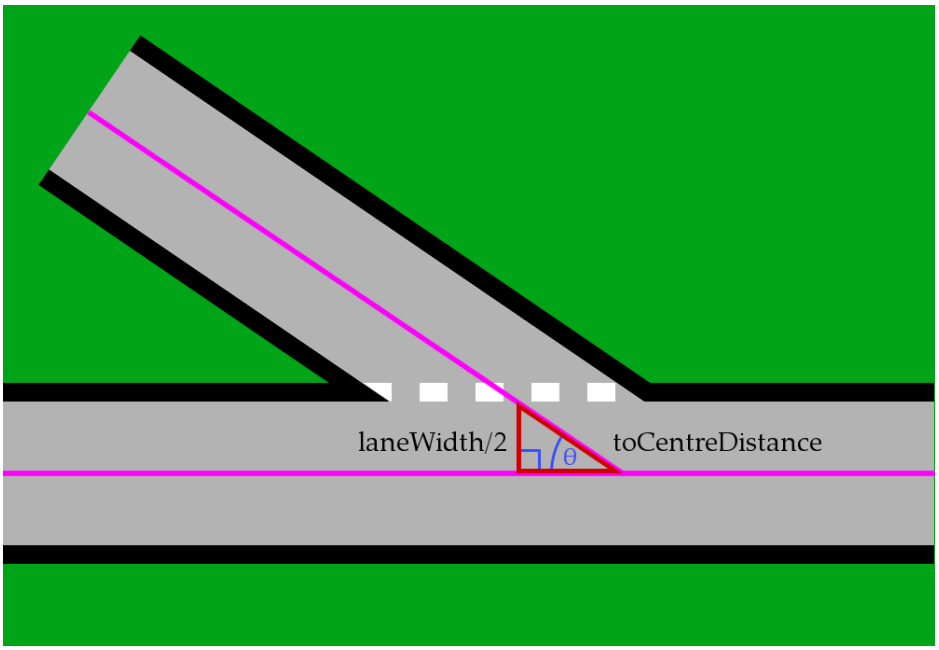

Figure A.4: A right-angled triangle used to calculate where the two centre lines meet. The centre lines are indicated in pink.

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

A.2 Generalising the Codebase

A.2.1 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.

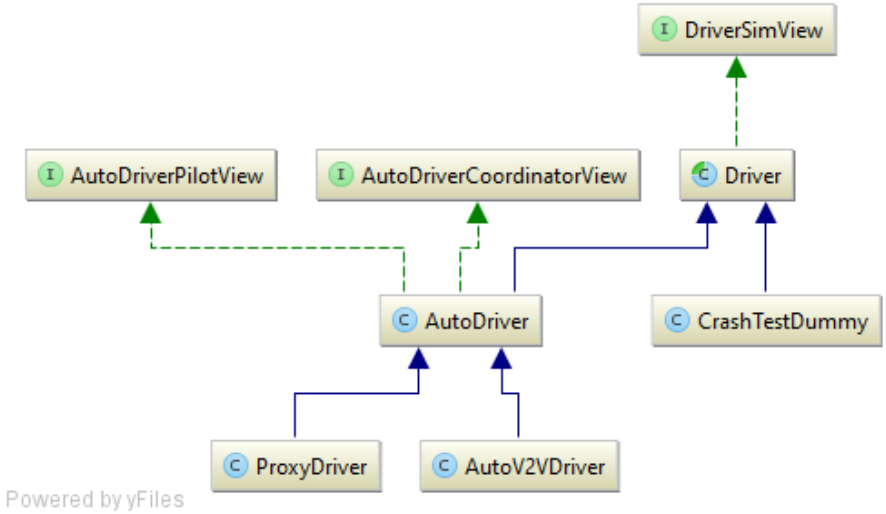

Figure A.5: The original class structure for *Driver*.

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.

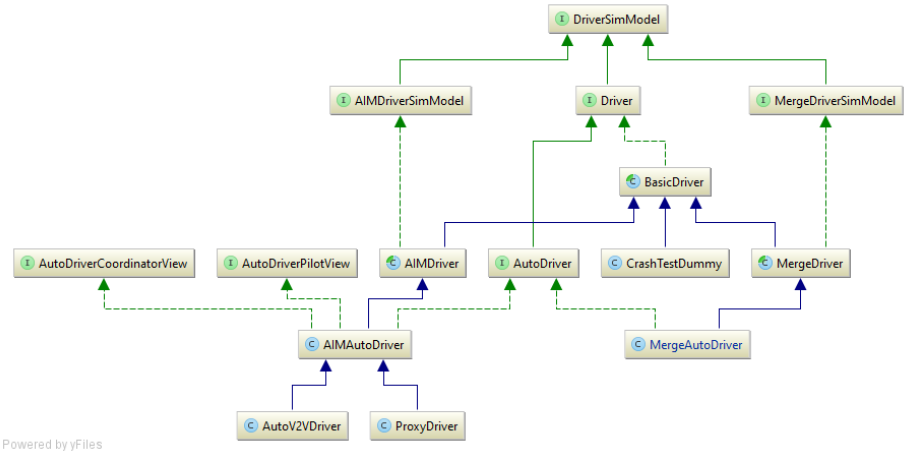

Figure A.6: The new class structure for *Driver*.

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.

A.2.2 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.7 and A.8.

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.9.

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

We also generalised the *SimSetupPanel* class to allow *SimViewer* to display non-AIM set-up controls. Figure A.11 shows the new class structure for *SimSetupPanel*.

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

A.2.3 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*.

The changes made to *aim4.map* were relatively straight-forward. The AIM specific features in *BasicMap* were extracted out in *BasicIntersectionMap* and *GridMap* was renamed to *GridIntersectionMap* and now inherits from the new interface. This allows for new map types, such as *MergeMap* to implement a map type without AIM features.

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

A.2.4 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.

A.2.5 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.2.1. 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.

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

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

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.3 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.3.1 MergeMapUtil

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

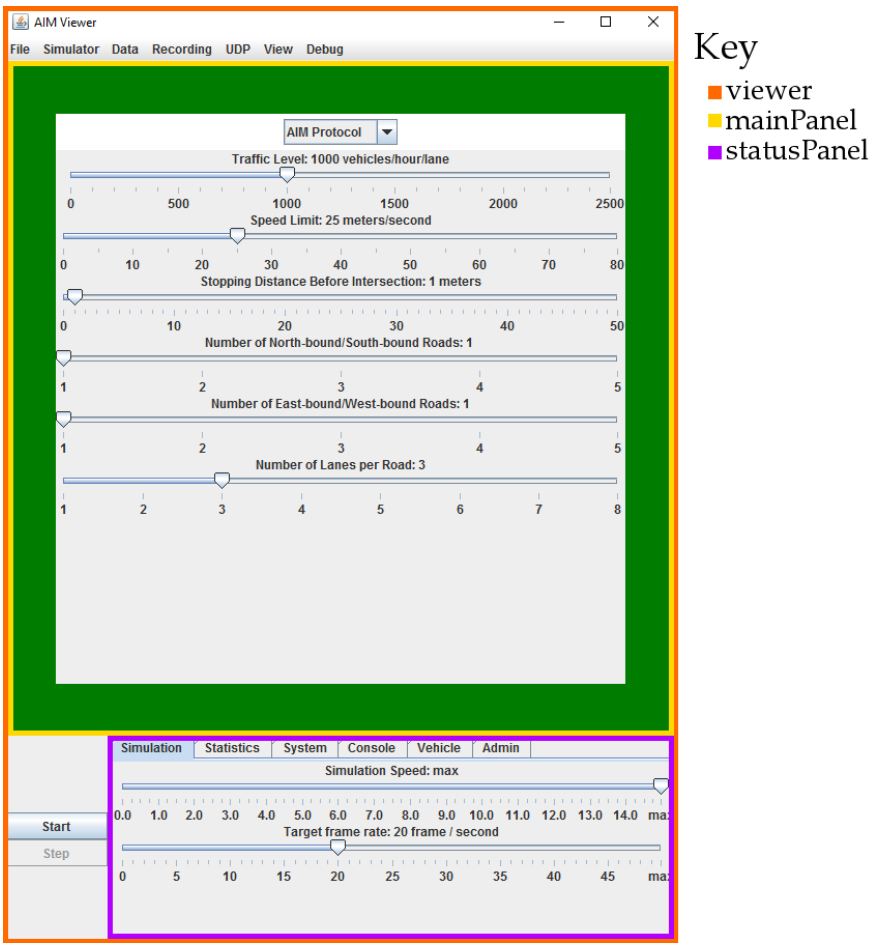

Figure A.7: Panel layout in the original simulator.

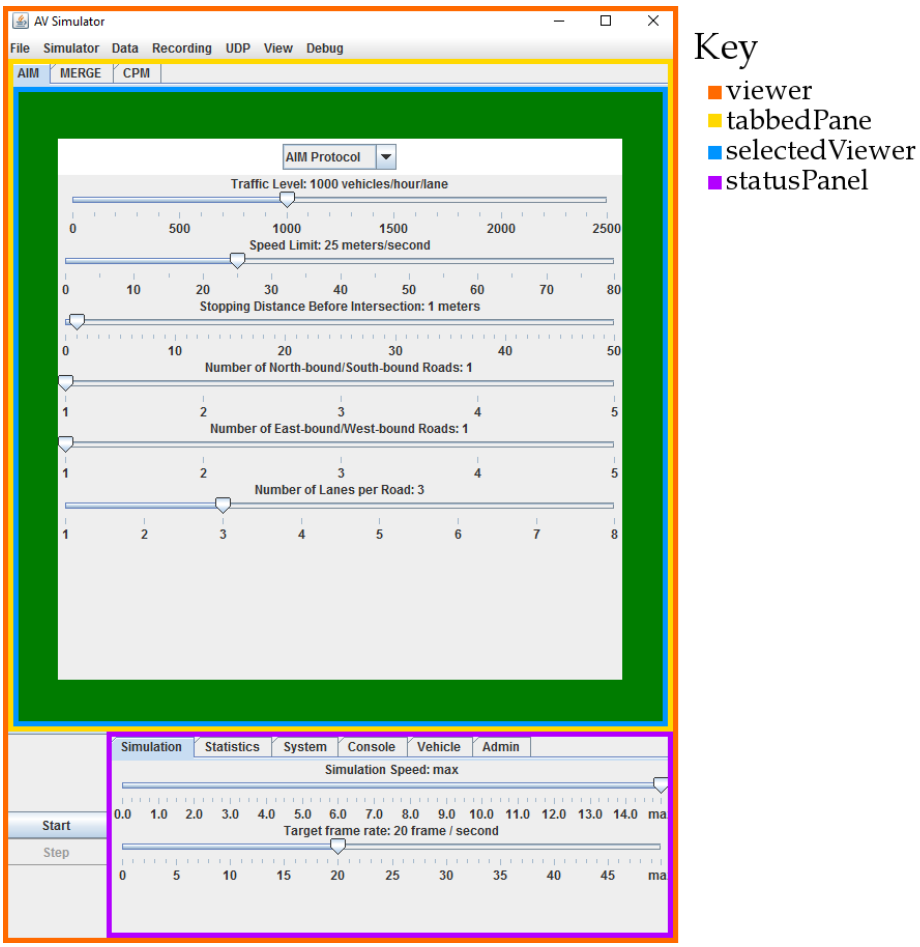

Figure A.8: Panel layout in the new simulator.

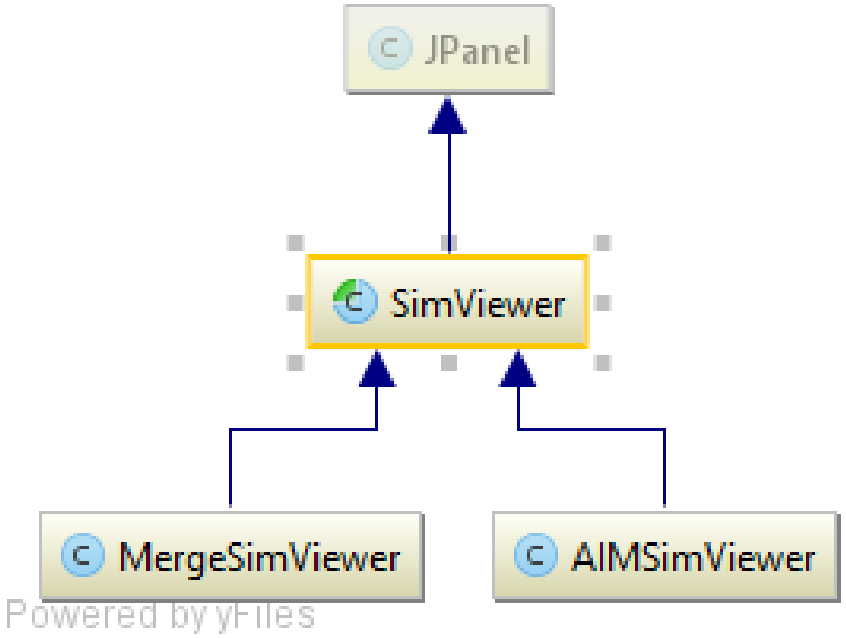

Figure A.9: The class diagram for *SimViewer*.

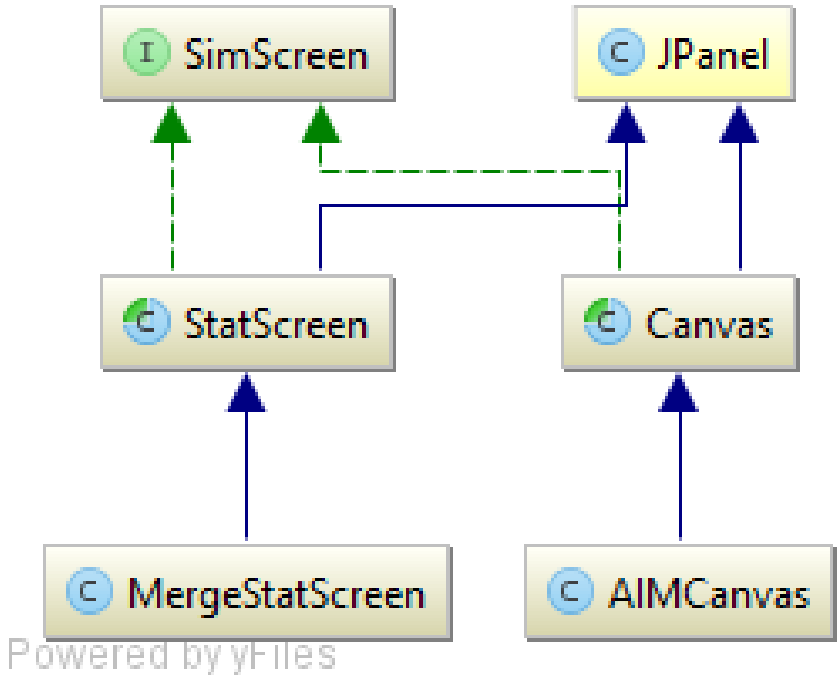

Figure A.10: The class diagram for *SimScreen*.

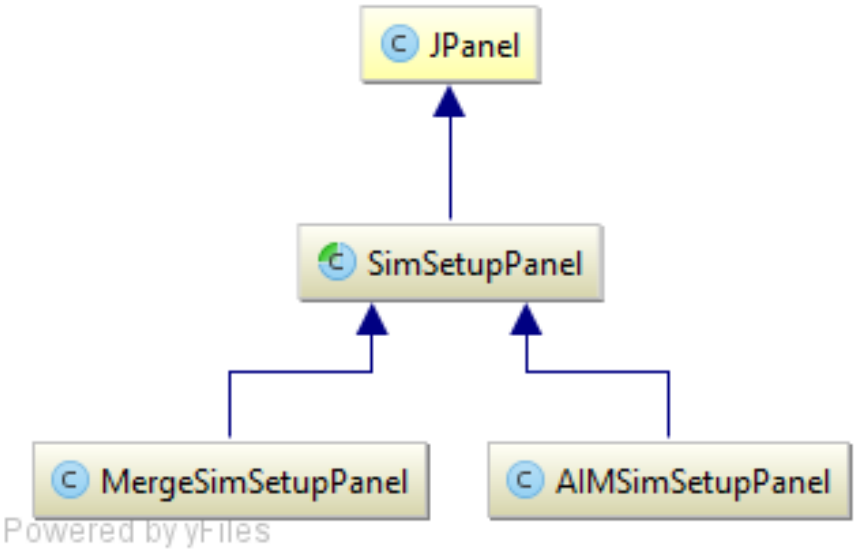

Figure A.11: The class diagram for *SimSetupPanel*.

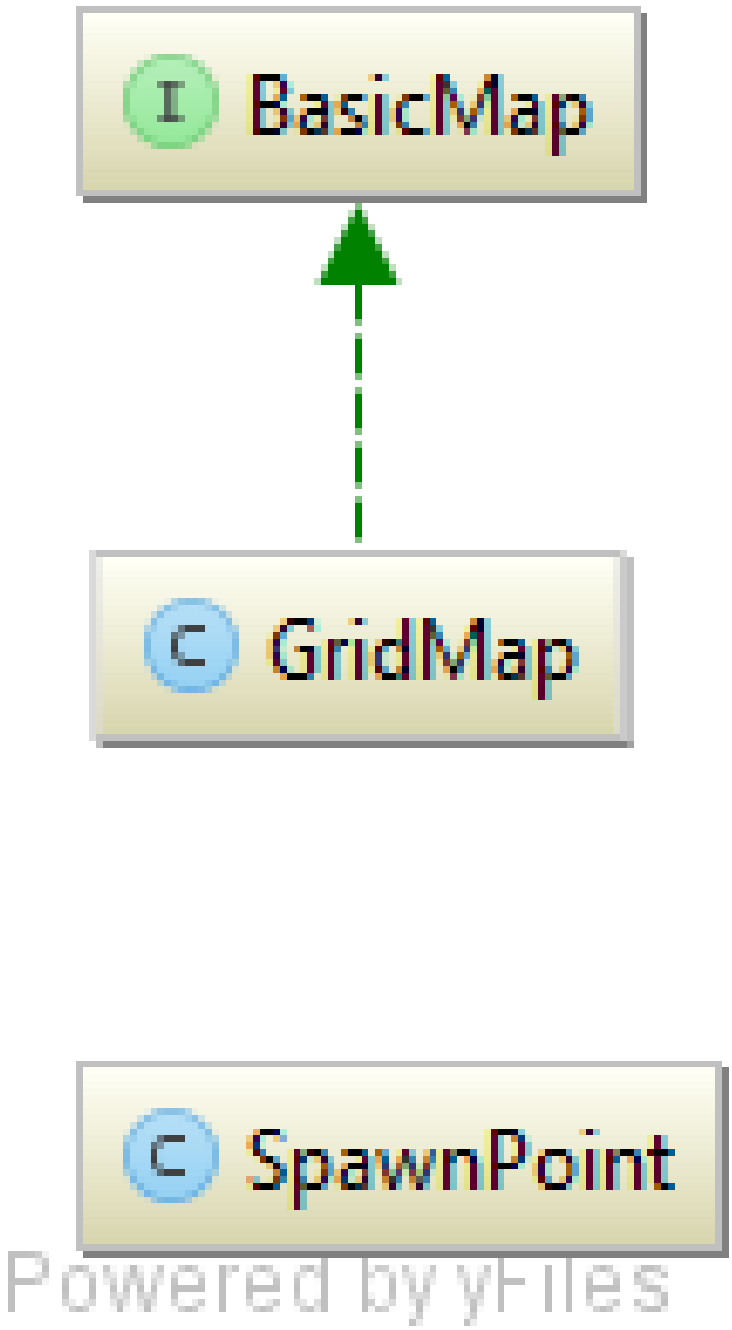

Figure A.12: The original class structure for *BasicMap* and *SpawnPoint*.

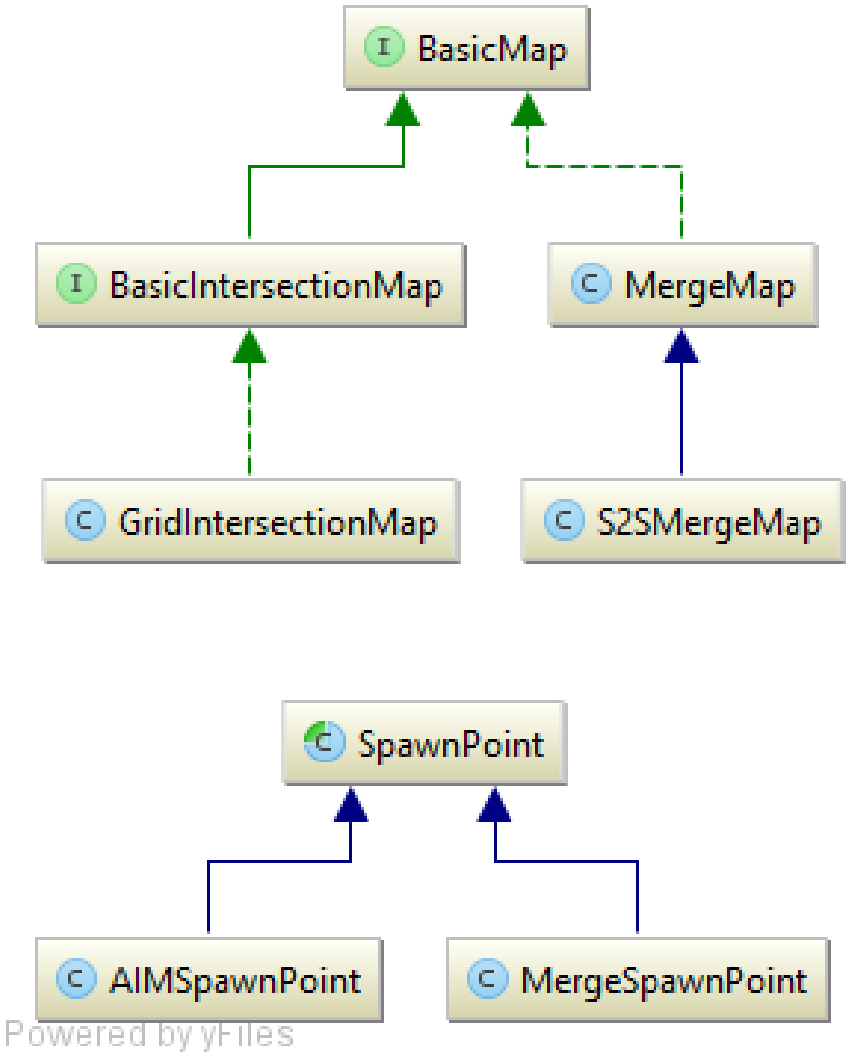

Figure A.13: The new class structure for *BasicMap* and *SpawnPoint*.

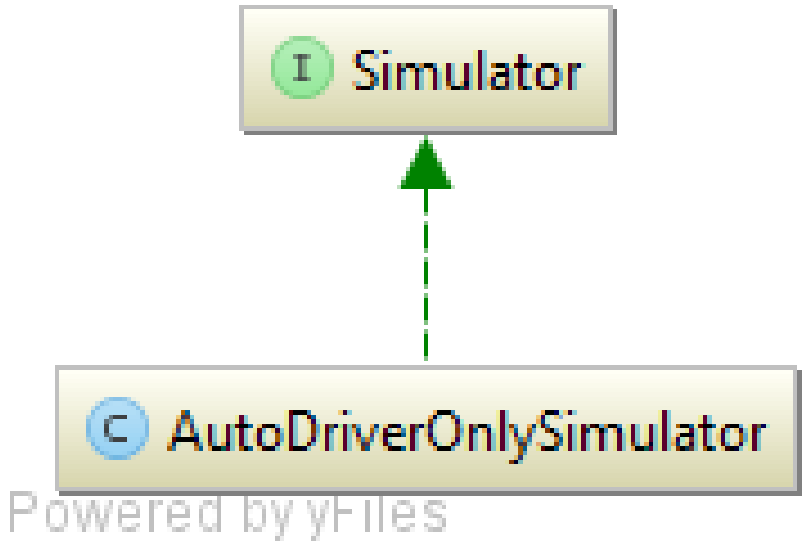

Figure A.14: The original class structure for *Simulator*.

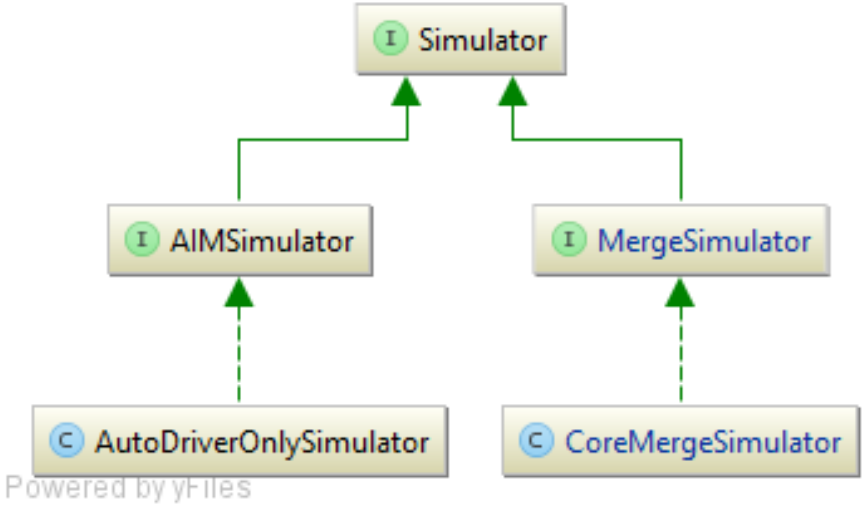

Figure A.15: The new class structure for *Simulator*.

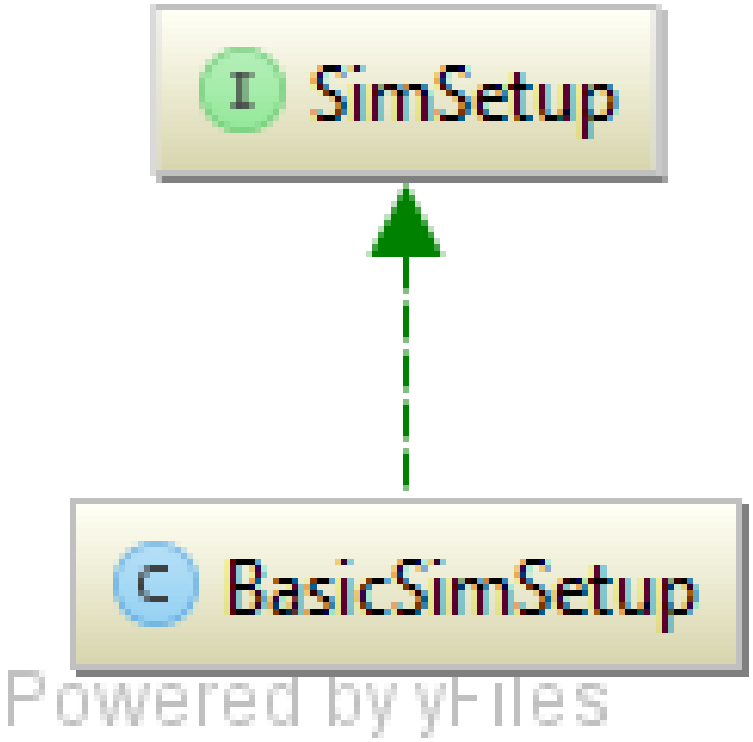

Figure A.16: The original class structure for *SimSetup*.

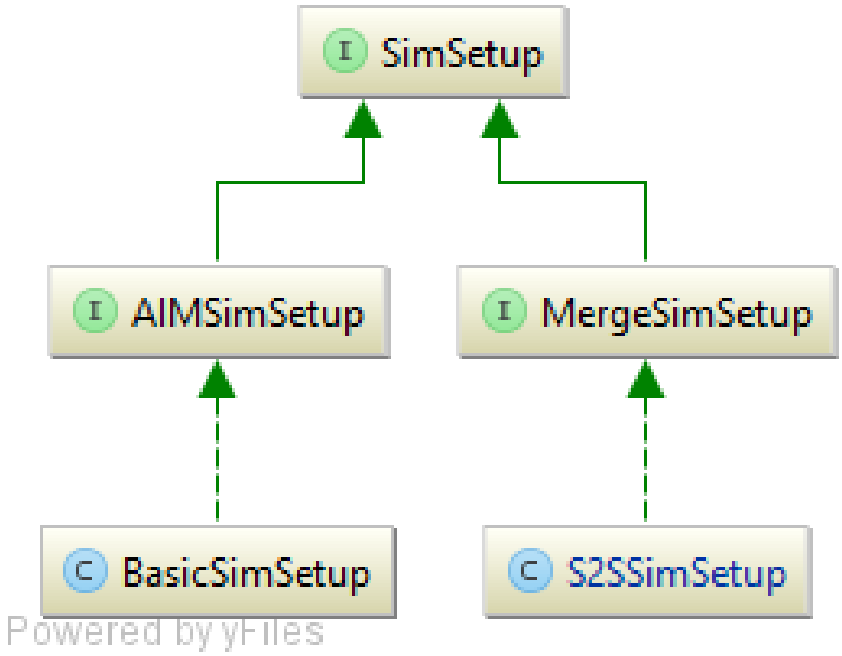

Figure A.17: The new class structure for *SimSetup*.

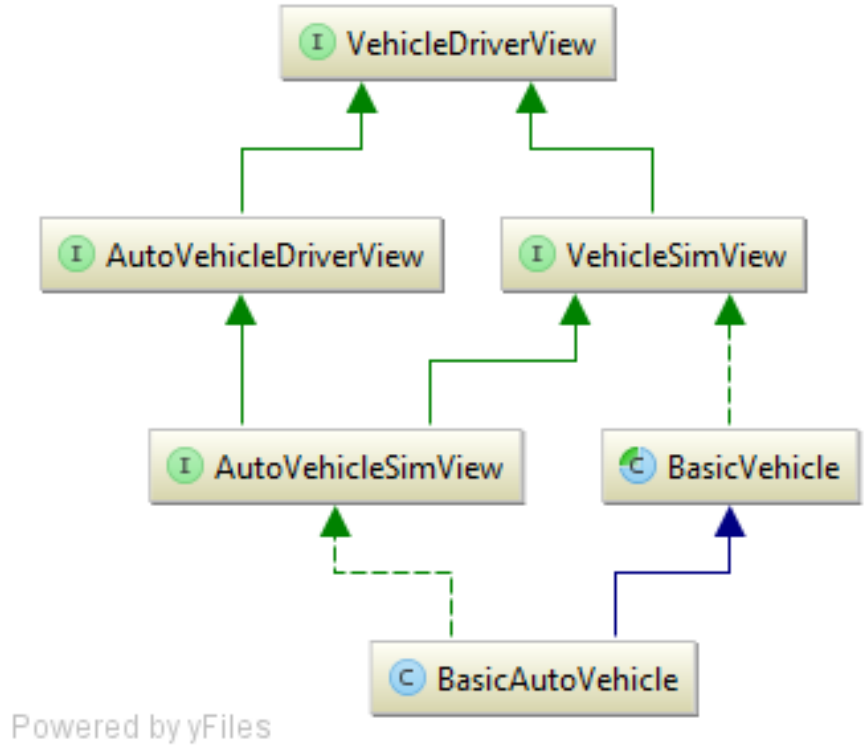

Figure A.18: The original class structure for *aim4.vehicle*.

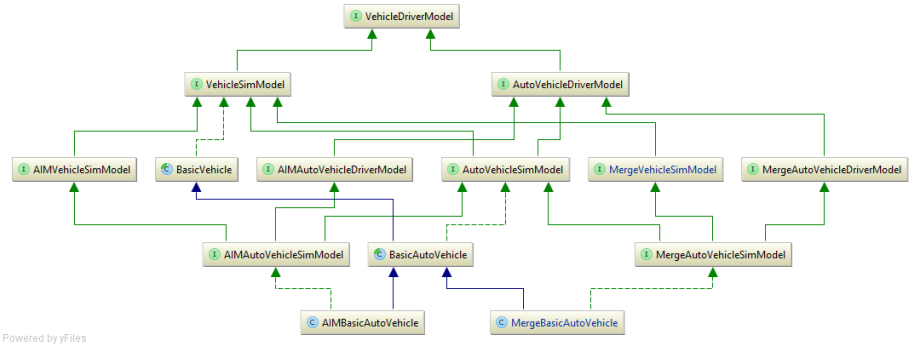

Figure A.19: The new class structure for *aim4.vehicle*.