Automated intersection crossing: optimal control in simulation

Måns Lerjefors
Electrical Engineering
Chalmers University of Technology
Gothenburg, Sweden
mansle@student.chalmers.se

Leon Glass
Electrical Engineering
Chalmers University of Technology
Gothenburg, Sweden
leongl@student.chalmers.se

Hannes Bolmstedt
Electrical Engineering
Chalmers University of Technology
Gothenburg, Sweden
hanbol@student.chalmers.se

Erik Williamsson
Electrical Engineering
Chalmers University of Technology
Gothenburg, Sweden
wilerik@student.chalmers.se

***Abstract*—A simulation platform for optimized control of traffic flow in intersections has been developed. The simulation platform is developed in MATLAB by utilizing classes and all vehicles are controlled using model predictive control. Collisions are avoided by using a critical zone in which only one vehicle is allowed at a time. The platform allows for the user to easily define an intersection and start parameters of the vehicles.**

I. Introduction

A. Background

In Europe, intersections-related accidents are responsible for around 20% of traffic related deaths and around 40% of the non-fatal injuries [1]. Due to the high risk of accidents, these traffic scenarios are among the most regulated, with vehicles guided simultaneously by traffic lights, signs, road-markings and right-of-way rules. As a consequence, they often form bottlenecks in the traffic system and even when not causing congestion, existing coordination rules are inherently inefficient, enforcing unnecessary deceleration and stops, thereby wasting both fuel and time.

As the automotive industry is pushing for automated vehicles, new possibilities arise as it enables direct communication between the vehicles and or some sort of hub for the intersections themselves. A safe system for guiding automated vehicles through intersections could greatly increase efficiency as it would eliminate the need for traffic lights and could use smaller intervals between the cars while maintaining and possibly increasing safety.

B. Purpose

The project aims to use an existing Model Predictive Controller (MPC) from previous projects and integrate it into a new platform for simulation of automated intersections. The platform should be general enough to support multiple different intersections and enable additions of new intersections. The possibility to include different kinds of vehicles with different properties in the platform should be investigated. If possible, this would make the crossings resemble a real life crossing even more. For the simulation to run properly and have an "optimal" traffic flow through the intersection, some sort of heuristic to determine in which order the vehicles should cross the intersection is necessary and needs to be developed.

The overall goal is to create a simulation platform for automated intersections, with support for at least three different types of intersections and with the possibility to add more in the future.

C. Problem formulation

Specifically, the problem will be to generalize an intersection class that can generate different intersections on the basis of some predetermined input parameters. These should be decided with care, in order to keep it as general as possible. The class should visualize the intersection, but also output paths for the vehicles. The vehicles should have their own class with its needed properties. Together with this, different scenarios need to be generated. It is then convenient to develop a situation analyst that calculates the crossing order. All simulation scenarios and crossing orders will have to be fed to an MPC that controls the dynamics of the vehicles during simulation.

D. Boundaries

As previously stated, this project will limit itself to not developing its own optimal control policy. For simplicity reasons and due to the limitation of time, pedestrians will not be taken into consideration when developing the different classes. The optimal control problem will also be handled under the assumption that all vehicles in the intersection will be autonomous. No extensive investigation of systems of intersections will be done. The cars are not allowed to overtake each other and switch lanes.

II. Theory

In this section the necessary theory is presented which describes how the system is made up and especially our generalized intersection class.

A. General Intersection

To define an intersection some essential parameters are needed. Firstly, it is necessary to define how many roads are connected to the intersection. Henceforth, these connected roads are called legs. E.g. in Figure 1 the number of legs is two. Each leg consists , in turn, of a number of lanes, this number, in the example below, is two. At which angle each leg connects to the crossing is defined by its leg angle α. This value corresponds to the legs center line relative to the predefined zero angle of the intersection. The property LANEWIDTH describes the width of each lane. Typically in our implementation (see section III) few legs are used. We always use 2 lanes.

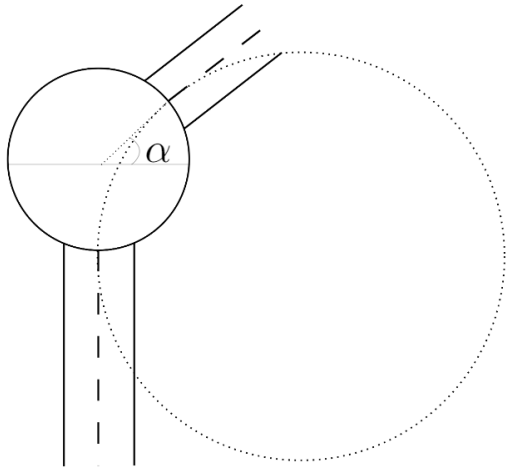

Figure 1: This figure shows a simple intersection generated according to our system. The intersection consist of two legs with two lanes each and a leg angle of α. The circle connecting the lanes is the area of the critical zone. A second circle is located on the right. Its arc connecting the points where either leg enters the critical zone constitutes a vehicles path through the critical zone.

Firstly, as can be seen in figure 1 a circle centered at the origin with radius r_c is plotted. This circle is defined as the crossing circle and will be used to connect the legs to each other. It also has another use as it defines the critical zone, i.e. the zone within which only one vehicle is allowed at a time. The legs are always straight and point towards the origin outside of the crossing circle. The two legs in the example in figure 1 have the leg vector $[\alpha \quad \frac{3\pi}{2}]$. The first leg is positioned at an angle α and the second vertically. This vector is generally defined as the LEGVECTOR and is a vector that represents the angle of each leg, relative to an angle of zero in the central frame of reference. This leads to the representation

$$\vec{\alpha} = [\alpha_1 \quad \alpha_2 \quad \dots \quad \alpha_n]^{\mathrm{T}}, \tag{1}$$

where n is the number of legs.
Secondly, a number of circles are constructed to connect the legs within the crossing circle. One illustration of this can be seen in figure 1 where a larger circle is present. If the radius of the large circle is changed, the circle within the crossing circle can describe every vehicle path and lane marking between the two legs. The angle where each arc starts and ends is called θ_1 and θ_2 respectively. Hence, the arc's angle is

$$\Delta\theta = \theta_2 - \theta_1. \tag{2}$$

Lastly, it is necessary to know what pair of legs that should be connected. For example a T-crossing has all its legs connected. But, a crossing that merges two legs into one has less merges than pairs. The connected legs are represented in a matrix called LEGSCONNECT or LC. This can be written as

$$LC = \begin{bmatrix} l_1 & l_2 \\ l_3 & l_4 \\ \vdots & \vdots \\ l_{n-1} & l_n \end{bmatrix}, \tag{3}$$

where n is the number of legs, again. Note that all combinations of legs are possible on each row, for example $[l_3 \quad l_2]$ is possible, since one leg can be connected to multiple different legs. Every row is then a pair that connects two legs. For example, to create a crossing where two legs are merged into a third leg, we name the intersections legs l_1, l_2 and l_3. The LC matrix to realize this is

$$LC = \begin{bmatrix} l_1 & l_3 \\ l_2 & l_3 \end{bmatrix}. \tag{4}$$

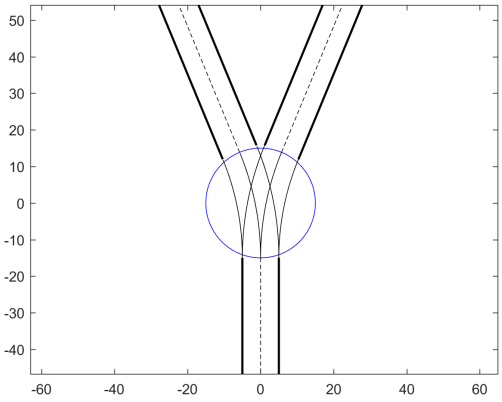

Figure 2: This figure illustrates an intersection merging two legs into one.

1) Trigonometry: It is important to understand the trigonometry the intersections are based on to understand its uses and limitations. The most basic mechanism is that every is leg is straight and points towards the common center. Each leg ends where the its center reaches the edge of the central circle, hereinafter called the crossing circle. This means that the inner edge of each leg is tangent to the crossing circle. These tangents have a few important points along them, one for each lane marking and one for each vehicle path. These points are marked as small circle markers along the tangents in figure 3. This poses the question of how to connect these points between different legs.

The next key step is to realize that the tangents of the crossing circle for two different legs will meet at a common point if extended, as long as the legs are not exactly opposite. This point is the center point of the large circle in figure 1. This is comes in handy as we can take two legs with arbitrary angles, and connect every point along the tangents with arcs around the point where the tangents intersect. The only thing that will change when connecting different points along the tangents is the radius. The semi circles corresponding to the lane markings can be seen in figure 3. The only case where this does not hold is in the case of two opposite legs, in which case the legs can be trivially connected with straight lines.

The way the legs are connected within the crossing circle means that the size of the crossing circle must be chosen carefully. The crossing circle must be large enough so that the legs do not overlap outside of the crossing circle.

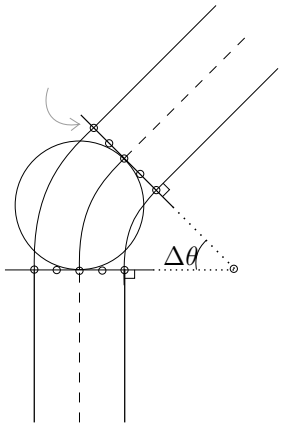

Figure 3: This figure shows the straight pars of the legs stopping at the tangents of the crossing circle. The tangents of two different legs can be extended to meet at a point which is used as the center for the semi circles connecting the legs. The grey arrow marks the edge of the critical zone.

B. Vehicle

The vehicles are modeled as point masses to simplify the simulation. These are later implemented in a POINTMASSVEHICLE class. A vehicle has a maximum speed called v_{max}. This is calculated from setting a maximum centripetal acceleration a, and using

$$v_{max} = \sqrt{a \cdot r}. \tag{5}$$

where r is the radius. The maximum vehicle speeds are also limited by the vehicle parameters in the longitudinal and lateral direction. For the purposes of control, each vehicle is modeled by a two-dimensional state space. Since vehicles are point masses moving on a predefined path, they can be modeled by a double integrator.

An essential part of the constraints put on admissible trajectories of the system are going to be constraints prohibiting collision of vehicles. To facilitate easy formulation of these constraints we model the vehicles' dynamics using as states time and lethargy (inverted velocity) as dependent on the vehicles position on its predetermined path. Thus, the i-th vehicle's state reads

$$x_i(p) = \begin{bmatrix} t_i(p) & z_i(p) \end{bmatrix}^T, \tag{6}$$

where p is a position on the path, $t_i(p)$ is a the time at which the vehicle is at position p, and $z_i(p)$ is the vehicle's lethargy at position p. The vehicles' dynamics are given by

$$\mathbf{x}_i'(p) = \mathbf{A}\mathbf{x}_i(p) + \mathbf{B}u_i, \tag{7}$$

where

$$\mathbf{A} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \tag{8}$$

and

$$\mathbf{B} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \tag{9}$$

and u_i is the system input.

This problem formulation allows for easy implementation of a variety of relevant constraints demanded by the nature of this application. Among these are limiting maximum speed and acceleration as a consequence of path curvature, as well as requiring vehicles to pass a certain point earlier or later than a specified time. This in effect means that collision avoidance constraints can be formulated efficiently.

C. Situation Analyst

The situation analyst function has the purpose of ordering an array of vehicles heuristically to improve the time needed for all vehicles to traverse the intersection.

The situation analyst estimates each vehicles arrival time using its current speed and distance from the intersection's critical zone. Each vehicle's estimated arrival time is taken to be

$$t_{\text{est}} = \frac{s_s}{v_0}, \tag{10}$$

where s_s is the vehicles distance from the critical zone and v_0 is the vehicle's speed. Vehicles are then ordered according to their estimated time of arrival at the intersection's critical zone.

We take the following measure to prevent vehicles having to take over each other in order to adhere to the crossing order postulated by the situation analyst function: if a vehicle is located behind another vehicle in the same lane, its estimated arrival time is forced to be at least two seconds after the preceding vehicle's arrival time. This prevents the situation

analyst from ever ordering vehicles in such a way as would necessitate overtaking.

D. Model Predictive Control

Model Predictive Control (MPC) is a control technique highly suited for control tasks, which place central importance on constraint satisfaction. MPC looks to use the system model to numerically compute input and state trajectories over a certain prediction horizon, which satisfy all constraints (feasibility) and are optimal with respect to a specific criterion. Among the prerequisites for this are a discrete state space formulation of the system, a suitable choice of optimality criterion and constraints, as well as sufficient computational resources.

The state space of the entire system for the purposes of the MPC is taken to be the concatenation of the state spaces of all vehicles. This state space model is then discretized using zero order hold.

A variety of constraints are enforced. Vehicles are prevented from colliding by constraining the times at which each vehicle may be in the critical zone of the intersection. Vehicles are prevented from going at dangerous or prohibited speeds by constraining their lethargy.

This information is used to convert the problem to standard quadratic form, which means to the form

$$\begin{aligned} \min_{\mathbf{v}} \quad & \frac{1}{2}\mathbf{v}\mathbf{H}\mathbf{v}^T + \mathbf{f}^T\mathbf{v} \\ \text{s.t.} \quad & \mathbf{A}\mathbf{v} = \mathbf{b} \\ & \mathbf{b}_\mathrm{l} \leq \mathbf{v} \leq \mathbf{b}_\mathrm{u} \\ & \mathbf{c}_\mathrm{l} \leq \mathbf{C}\mathbf{v} \leq \mathbf{c}_\mathrm{u}, \end{aligned} \tag{11}$$

where the matrices $\mathbf{H}$ and $\mathbf{f}$ are a result of our choice of optimality criterion, the matrices $\mathbf{A}$ and $\mathbf{b}$ are a result of our state space model and the initial conditions, and the matrices $\mathbf{b}_\mathrm{l}$, $\mathbf{b}_u$, $\mathbf{c}_l$, $\mathbf{c}_u$ and $\mathbf{C}$ are a result of our choice of constraints. The vector $\mathbf{v}$ contains the system trajectory in terms of state and input.

III. Implementation

In this section we explain our method to implement the generalized intersection, the vehicles and the situation analyst. The simulation platform is built with object oriented programming in MATLAB, where the intersection and vehicles are implemented as classes. Together they are part of a main class called *Task* containing all information about the scenario.

The structure of the simulation platform can be seen in figure 4, which shows the order of the most important steps of the program when running the main file. The first block is *Scenariogen* in which the input data for the vehicles and intersection class is defined. The vehicle and intersection class are then created from this data before both are added to the main class, *Task*. The structure of *Task* can be seen in figure 5.

After the main class has been created, the vehicles' trajectories are generated in *Task.V.gen_veh_paths*, and the crossing order of the vehicles is generated in *Task.SituationAnalyst*. The main class *Task*, then contains all information necessary to perform MPC. The MPC updates on every iteration together with the situation analyst. Every iteration saves a snapshot of the dynamics. In the end every snapshot is plotted, which creates the full simulation.

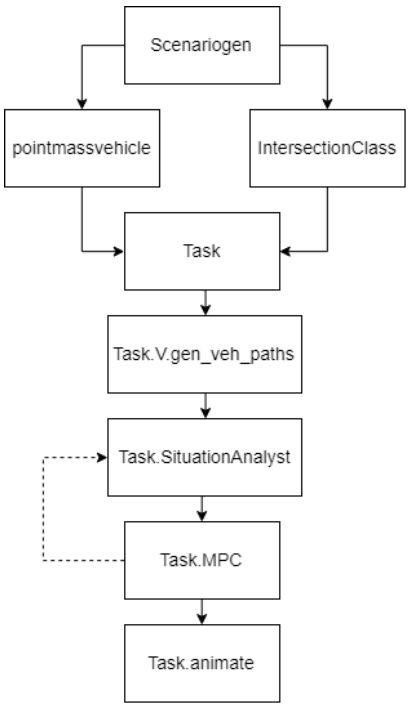

Figure 4: This figure illustrates the structure of our code.

A. Task class

The task class is the main class of the simulation. It is within this class that all necessary information and functionality is contained. The structure of this class can be seen in figure 5. As can be seen in this figure, the intersection class and pointmassvehicle class works as subclasses to the task class, providing the task structure with the necessary intersection and vehicle data. The data contained in the task class are mainly related to the optimization process of the MPC. Except from the class initiating function where the length of the MPC horizon, the number of MPC updates, the sampling time and the optimization solver are chosen and initiated, the task class consists of five methods. The *SituationAnalyst* described in the theory section is one of these methods. It generates the crossing order based on a heuristic. Task has a method called *ScaleFact* creating scaling factors which are necessary for the optimization solver. The loop where the MPC trajectories are solved for the number of updates defined in the initiation method, is done in the method *MPC*. The *MPC* method outputs the results as the vectors *traveled, veloc, acc* and *timesteps*. These vectors corresponds the traveled distance of each car at each timestep, the velocity of each car at each timestep,

the acceleration of each car at each timestep and how many seconds each timestep is.

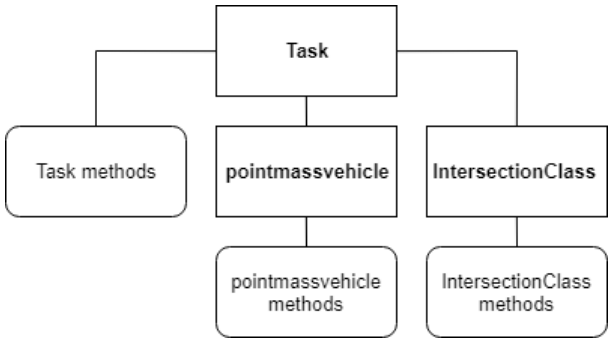

Figure 5: This figure depicts the structure of the Task class.

B. Intersection class

The intersection is implemented by using a class in MATLAB, this class will be referred to as the intersection class. An object of the class, i.e an intersection, is created by calling a constructor method. The constructor method takes a set of inputs which defines the intersection and generates all the properties of the class. The inputs are explained in II-A and the class constructor has the following call:

```
I = intersectionClass(numlegs, legangle,...
numlanes, lanewidth, controlradius,...
turnzone, legsconnect);
```

Where *numlegs* is the number of legs, *legangle* is the angle of the legs, *numlanes* is the number of lanes, *lanewidth* is the lane width, *controlradius* is the radius of the circle within which the vehicles are controlled, *turnzone* is the radius of the crossing circle, *legsconnect* are the pairs of legs that should be connected.

The second method of the intersection class is the plot function, called *plot_intersection*. This method only uses the object itself as input as all necessary information is stored in the properties of the object, from which the method plots the lane markings of the intersection. Plain black lines are used for all lane markings within the crossing circle to avoid cluttering. Outside the crossing circle, the edge of the road is marked with a thicker black line and lane dividers are dotted lines.

C. Vehicle class

The vehicle class holds all the information about the vehicles, current position, maximum speed etc.

The vehicle class has the method *gen_veh_paths*. This method is used to generate the path for each vehicle, including velocity constraints along the path. The method uses the vehicle class and intersection class as input and outputs the updated vehicle class with all information about the path. The maximum velocity at every point along the path is the minimum of the speed limit of the road and the speed given by the maximum lateral acceleration during the turn.

D. Limitations of implementation

These are the current overall limitations. However, some limitations are coming from only certain parts of the code which enables future improvements. This is discussed in more detail later in chapter V, Discussion.

- All legs must point toward a common center point of the intersection
- The legs are straight outside the crossing circle
- Common crossing circle for every leg
- Only two lanes for each leg
- If two legs are connected, every lane is connected
- The critical zone is the same as the crossing circle
- The vehicles are point mass and the position of a vehicle is based on a singular point
- Vehicles can not switch lanes

IV. Results

In this section we present the results from simulating the whole implemented model presented in III.

A. Scenario 1

The resulting simulation having a T-crossing with 4 vehicles starting at initial speeds $\mathbf{v_0} = [38 \quad 40 \quad 40 \quad 40]^{\mathrm{T}}/3.6$ m/s, is illustrated in Figure 6 below. Two vehicles start at leg number two (located at $\pi/2$) and one vehicle on leg 1 and 3.

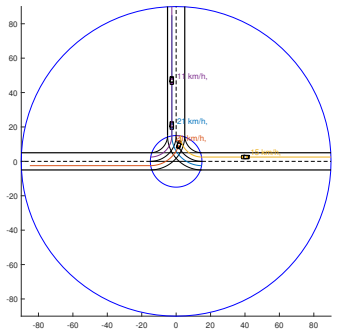

Figure 6: A snapshot of the simulated model using a generated T-crossing from our general intersection class presented in II-A. As can be seen, the vehicles follow the desired paths.

B. Scenario 2

In this case the same scenario is used for the vehicles, using a four-way intersection. Now each vehicle can start on its own lane. All of this is illustrated in Figure 7 below.

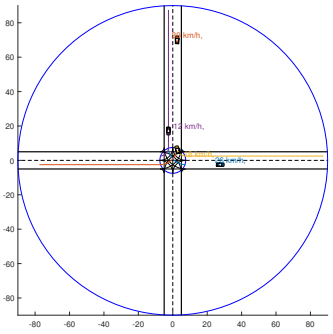

Figure 7: A snapshot of the simulated model using a generated four-way intersection from our general intersection class presented in II-A. As can be seen, the vehicles follow the desired paths.

C. Scenario 3

In this scenario an intersection with 6 legs are generated. Each leg is distributed with an angle of 45 degrees between each other. There are 6 vehicles with an initial speed vector $\mathbf{v_0} = \begin{bmatrix}37 & 38 & 40 & 30 & 40 & 40\end{bmatrix}^{\mathrm{T}}/3.6$ m/s. The result of this simulation can be seen in figure 8 below. This scenario goes to show the versatility of our intersection class.

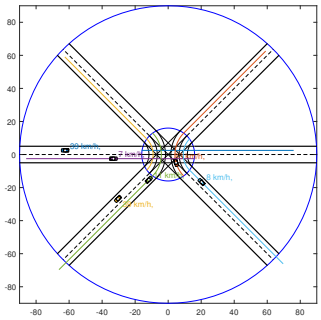

Figure 8: An intersection with 6 legs and 6 vehicles generated from the general intersection class.

V. Discussion

This sections discusses the most important results in the report, and gives insights on future improvements.

A. Intersection class

The whole intersection class is based around the idea of a common center point with a crossing circle. These assumptions is what makes it possible to connect every leg and lane with semi-circles. If a completely new type of intersection is desired, it is recommended to create a new separate intersection class to accommodate for this.

The number of lanes can easily be changed with regards to the intersection class, however other parts of the code currently only allow for two lanes for each leg. For example the MPC and *gen_veh_paths*, of which the at least latter should be fairly easy to adapt to accommodate for more lanes.

B. The critical zone

The MPC uses collision constrains based on a critical zone since this project is built on previous code which used a critical zone. This makes the problem easier as there is only one region in which collisions are possible and to only have one vehicle in this region at a time guarantees no collisions. However, since it is in reality possible to have two or more vehicles in the critical zone at once without collisions, the traffic flow can be optimized further.

The limitations of using a critical zone becomes clear when two vehicles that either use the same path, or are on paths that never intersects wait for each other to exit the critical zone. Thus for further improvement, it would be beneficial to find every collision points of every vehicle path and base the constraints on these.

For an arbitrary intersection, it has proven to be quite a tricky problem to find all the collision points [2]. However, thanks to the geometry we use to construct our intersections, it is a fairly simple problem to find the collision points as it is a problem of finding intersection points of known circles.

C. Vehicle class

While some parameters of the vehicles can be changed, such as maximum speed and maximum lateral velocity a completely general vehicle class has not been implemented. Due to the use of a critical zone rather than collision points, exact dimensions of the vehicles are not very important. The critical zone has by nature a large margin of safety as it includes the whole crossing part of the intersection.

D. Situation analyst

This function is purely heuristic and can be improved upon. For most situations without sequences of cars following each other closely from multiple entry legs this function works acceptably. However, when there are sequences of cars following each other closely from multiple entry legs, cars have to brake in an alternating pattern between the entry legs, which seems suboptimal.

References

[1] European comission. Traffic Safety basic Facts on Junctions. European comission, Directorate General for Transport, June 2017

[2] Fuquan Pan , Lixia Zhang , Jian Lu , Jiguang James Zhao Fengyuan-Wang (2013) A Method for Determining the Number of Traffic Conflict Points BetweenVehicles at Major–Minor Highway Intersections, Traffic Injury Prevention, 14:4, 424-433