Visualization of Platooning in Unity

TOBIAS ESTREEN

SOFIA NORD
Visualisering av Platooning i Unity

TOBIAS ESTREEN

SOFIA NORD
Abstract

The goal of this project was to create an accurate and flexible visualization of an already existing platooning simulation with the use of Unity. This was done by using the output of the simulation as input for the visualization with information about the speed, position and status of each vehicle in the platoon. A simple steering algorithm was created for test cases with a curved road. With no exact information between two input points, linear interpolation was utilized to estimate the velocity. Without adjusting the position at each input, the maximum errors between the visualization and the simulation for each vehicle were approximately 3.5 meters. After introducing a position adjustment at each input, the maximum errors decreased to between 0.6 and 0.8 meters at the cost of non-continuous motion. The error threshold for the visualization to be considered accurate is 2 meters, implying that the position adjustment is required for good results.

Keywords: Platooning, Simulation, Visualization, Unity
Sammanfattning

Målet med detta projekt var att skapa en noggrann och flexibel visualisering av en existerande konvojkörning simulering med hjälp av Unity. Detta gjordes genom att använda utdata från simuleringen som indata i visualiseringen med information om hastighet, position och status för varje fordon i konvojen. En enkel styrningsalgoritm skapades för testfall där vägen svänger. Utan exakt information mellan varje indata användes linjär interpolering för att uppskatta hastigheten. Utan att justera positionen vid varje indata blev de maximala felen mellan visualiseringen och simuleringen ungefär 3.5 meter för varje fordon. Efter det att positionsjustering introducerats minskade felen till mellan 0.6 och 0.8 meter men med icke-kontinuerlig rörelse hos fordonen. Felgränsen för att visualiseringen ska räknas som noggrann är 2 meter, vilket betyder att positionsjustering är nödvändig för bra resultat.

Nyckelord: Konvojkörning, Simulering, Visualisering, Unity
Acknowledgment

We are truly grateful because we managed to complete this visualization within the given time. This project could not have been completed without the guidance and encouragement from our supervisor Karl Meinke. For that we are sincerely thankful.
# Contents

1 Introduction 1  
1.1 Autonomous Driving ............................... 1  
1.2 Platooning ......................................... 1  
1.3 Unity Game Engine ................................. 2  

2 Material and method 2  
2.1 Model - Vehicle Dynamics ......................... 2  
2.2 Platooning Simulation ............................... 4  
  2.2.1 Scheme of blocks ............................... 5  
  2.2.2 Java Code ..................................... 5  
  2.2.3 Java to C# ..................................... 6  
2.3 Unity Program ..................................... 7  
  2.3.1 Create Vehicles ................................. 7  
  2.3.2 Generate Roads ................................. 7  
  2.3.3 Terrain ....................................... 7  
  2.3.4 UI - User Interface ............................ 8  
  2.3.5 Handling Velocity Inputs ....................... 8  
  2.3.6 Linear Velocity Approximation .................. 8  
  2.3.7 Steering ..................................... 8  
  2.3.8 Calculating Visualization Errors .............. 9  

3 Test Cases ....................................... 9  

4 Results ....................................... 10  
4.1 Emergency Braking on Straight Road ............... 10  
4.2 Curved Road without Position Adjustment .......... 11  
4.3 Curved Road with Position Adjustment ............. 11  

5 Conclusion .................................... 12  
5.1 Discussion of Results ............................ 12  
5.2 Discussion of Method ............................. 12  
5.3 Improvement and Future Work ...................... 13  
5.4 Final Words ..................................... 13
List of Figures

1. Friction coefficient model used ........................................ 3
2. Friction curves .............................................................. 4
3. Screen capture of the visualization .................................... 10

List of Tables

1. Commands of the platooning simulation .............................. 6
2. Error values of the emergency braking test case .................. 11
3. Error values of the first curved road test case ..................... 11
4. Error values of the second curved road test case .................. 11
1 Introduction

1.1 Autonomous Driving

Several vehicle manufacturers are investing in autonomous driving and the technology behind automated vehicles is constantly evolving. In SAE International’s J3016 document [1] six levels of automation in vehicles are presented, the lowest representing a vehicle without any kind of autonomy and the highest representing a fully autonomous vehicle. No vehicle currently exists that is fully driverless or that can be trusted with full autonomy. The highest level of autonomous driving available to us, at present, is vehicles of level 3, meaning they have neither full nor high autonomy and require a human driver present at all times [2].

1.2 Platooning

Platooning is a relatively new concept within autonomous driving that allows vehicles to drive closer to each other on the road in a so called platoon. There are several ideas on the dynamics, but in general, a platoon formation includes a ‘leader’ vehicle and one or more ‘follower’ vehicles that follows the leader. All vehicles in the platoon are connected wirelessly, communicating with each other. The idea is to have several vehicles to one driver, having the following vehicles follow the leader while autonomously adjust their acceleration, deceleration and steering to the rest of the platoon, particularly to the vehicle in front. However, as mentioned earlier vehicles at this level of autonomy are not available to us yet. Even so, simple platooning systems where a driver is present in each vehicle have been developed, though not nearly as evolved as it can be. Daimler [3], a vehicle manufacturer, has recently tested a simple platoon on public highways in Oregon and Nevada with successful results. Simple platooning systems like this have been tested in other parts of the world as well and platoons may be officially introduced in a near future [4]. Currently, platooning is mostly of interest for use on highways, in particular for heavy-duty vehicles. A platoon may not work as good in urban areas as in rural areas due to the inefficiency of a platoon at intersections and roundabouts etc.

There are several on going projects on platooning worldwide and the goals or motives varies among them. Generally, they all aim to enjoy the many benefits that platooning offers. Since the vehicles in a platoon wirelessly communicate with each other, the technique enables them to drive closer to each
other which, due to aerodynamics, results in fuel savings and reduced carbon emissions. This is both ecologically and economically better, especially for heavy-duty vehicles. According to Scania [5], they achieved a 12 percent fuel saving for the following vehicles when testing a platoon on their test track. Other studies have shown similar results [6]. Depending on time gap and the driving of the leading vehicle this percentage varies. Regardless of the percentage, platooning can be considered an eco-driving system. Platooning is a promising way to increase traffic flow and number of vehicles on the road since the distance between them are smaller. Additionally, platooning may even enhance safety due to small speed variations and the relative low impact velocity in case of collision [7].

1.3 Unity Game Engine

Unity is a multi-platform game engine, mainly used to develop two- and three-dimensional games and simulations for computers, consoles, smart phones, tablets and others. Unity supports 2D and 3D graphics, user interfaces, AI path-finding tools among others. An important feature is scripting, which is only possible using C#.

In Unity it is also possible to download assets, which is a digital package with components that can be used freely within the terms of conditions. Purchasing assets is a good option to making your own since it can be hard to create realistic looking objects that act as desired. Some assets are completely free to download while others have to be purchased [8].

2 Material and method

2.1 Model - Vehicle Dynamics

The model that was used to simulate the platoon takes two physical factors into account when deciding the speed and position: wheel friction and air resistance. In order to create a lifelike simulation it is highly important that these are true to reality.

The wheel friction is calculated using the equation:

$$\text{wheel friction} = -\mu (\text{slip rate}) \times \frac{m}{4} \times g$$

(1)
where $\mu$ is the friction coefficient, $m$ is the mass of the vehicle and $g$ is the gravitational constant. The friction coefficient is a function of the slip rate:

$$\text{slip rate} = \frac{|v - r \omega|}{v}$$  \hspace{1cm} (2)

where $v$ is the velocity of the vehicle, $r$ is the radius of the wheel and $\omega$ is the angular velocity of the wheel. The friction coefficient is finally given by:

$$\mu(\text{slip rate}) = \begin{cases} 
\text{slip rate} \times \frac{\text{slip limit}}{\text{slip max}} & \text{if slip rate} \leq \text{slip max} \\
\frac{(\mu_{\text{max}} - \mu_{\text{limit}})(\text{slip rate} - \text{slip max})}{\text{slip max} - \text{slip limit}} + \mu_{\text{max}} & \text{if slip rate} < \text{slip limit} \\
\text{slip limit} & \text{if slip rate} > \text{slip limit}
\end{cases}$$  \hspace{1cm} (3)

where $\mu_{\text{max}}$ and slip max are the values at maximum friction and $\mu_{\text{limit}}$ and slip limit are the values at maximum slip rate. These values depend on the condition of the road.

Figure 1: Friction coefficient model used

The air resistance model used was given by:

$$F_{\text{air}} = \begin{cases} 
c \times (1 - (-0.9379 \times d + 12.8966)/100) \times v^2 & \text{if } d \leq 15m \\
c \times v^2 & \text{if } d > 15m
\end{cases}$$  \hspace{1cm} (4)
where $c$ is the air drag coefficient of the vehicle, $d$ is the distance to the vehicle in front and $v$ is the velocity of the vehicle.

In the simulation the vehicles used were heavy trucks and the road had winter driving condition. Therefore the constants were set to:

- $g = 9.8 \text{ m/s}^2$
- $m = 14000 \text{ kg}$
- $r = 0.5 \text{ m}$
- slip max $= 0.2$
- slip limit $= 0.3$
- $\mu_{max} = 0.3$
- $\mu_{limit} = 0.2$
- $c = 0.6$
- max engine torque $= 1700 \text{ Nm}$

### 2.2 Platooning Simulation

To simulate a platoon, the simulation creates vehicles and provides them with a position and a character of either leader or follower. The vehicles are created from a vehicle model that uses a block diagram where all functions of
Visualization of Platooning in Unity

2 MATERIAL AND METHOD

the vehicle, such as braking systems and inner communications, are defined in a particular block. An input command is given to the simulation that controls the leading vehicle’s behavior.

2.2.1 Scheme of blocks

There are two types of blocks defined in the vehicle model, Function blocks and Comms blocks. The Comms blocks transfer inter vehicular data and handles inter vehicular communication. In the vehicle model only one simple comms block is used to model a wireless communication unit in the simulation. The Function blocks are what controls the individual parts and functions of the vehicle. Some examples are the four blocks describing each wheel’s Anti Brake Locking system, one block that describes the Adaptive Cruise Control Calculator and another block that acts as the Brake Torque Calculator. All these blocks communicate with each other.

2.2.2 Java Code

The platooning simulation is written in Java and all constants, presented in section 2.1, are defined and used in this code. The simulation has 22 different strings of commands as input. These commands provide the leading vehicle with information about acceleration and braking. In other words, the input controls the leading vehicle. Ten of these commands are for acceleration and another ten are for braking, each of them corresponding to a percentage of the total power, that is how much the pedal is being pressed down. The last two commands, ’e’ and ’0’, refer to emergency braking and to an idle pedal position, respectively. All commands are presented in table 1.
Table 1: Commands of the platooning simulation

<table>
<thead>
<tr>
<th>Acceleration</th>
<th>Braking</th>
<th>Percentage of total power</th>
</tr>
</thead>
<tbody>
<tr>
<td>a1</td>
<td>b1</td>
<td>10</td>
</tr>
<tr>
<td>a2</td>
<td>b2</td>
<td>20</td>
</tr>
<tr>
<td>a3</td>
<td>b3</td>
<td>30</td>
</tr>
<tr>
<td>a4</td>
<td>b4</td>
<td>40</td>
</tr>
<tr>
<td>a5</td>
<td>b5</td>
<td>50</td>
</tr>
<tr>
<td>a6</td>
<td>b6</td>
<td>60</td>
</tr>
<tr>
<td>a7</td>
<td>b7</td>
<td>70</td>
</tr>
<tr>
<td>a8</td>
<td>b8</td>
<td>80</td>
</tr>
<tr>
<td>a9</td>
<td>b9</td>
<td>90</td>
</tr>
<tr>
<td>a10</td>
<td>b10</td>
<td>100</td>
</tr>
</tbody>
</table>

| 0            | idle pedal position |
| 0            | emergency braking   |

When an input is registered it goes through the necessary blocks in order for the leader to act accordingly. Then the following vehicles receives commands on how they should act to the behavior of the vehicle in front. When all vehicles have their commands, an output with information of how the platoon handled the input command is sent back. Each input simulates \( t \) seconds of driving and the corresponding output presents the information of the platoon at the exact moment the command is given as well as the information after the simulated time \( t \). The information refers to the vehicle’s momentary speed, distance from the starting point, speed difference of the simulated time \( t \) and status (’good’, ’tooClose’, ’tooFar’ or ’crash’).

### 2.2.3 Java to C#

The biggest obstacle encountered was how to make the Java code for the platooning simulation compatible with Unity which only supports C#. There were several ways to solve this problem. Of course it is possible to rewrite or convert the Java code into C#, but that would have been very time consuming and rather inefficient. Another solution is to use Java Native Interface (JNI) so that the Java code can be read in a C# script, but Unity only supports JNI for android development, meaning the solution was not suitable for this purpose. A third solution is to save the output from the Java code in a separate text file and read that file in Unity, in other words, the output of the simulation is used as an input to the program. This method seemed both
flexible and fast, even though it requires the Java code to be run separately from the Unity program. Nevertheless, this method was the best solution to the problem.

2.3 Unity Program

To visualize platooning the output of the simulation was read and used to set speed and position for each vehicle in the Unity program. It was highly important not to use any built-in Unity physics since it could interfere with the physics used in the platooning simulation. The program creates all game objects for the visualization using scripting and assets. Through scripting the program also calculates the movements of the vehicles and the errors in position between visualization and simulation.

2.3.1 Create Vehicles

From the output of the simulation the program identifies the number of vehicles in the platoon and instantiates the same amount of clones of a given model, in this case Single Detailed Truck by VIS Games [9]. Each duplicate of the model is a separate game object given the correct starting position according to the output of the simulation.

2.3.2 Generate Roads

To generate a road it was fitting to use an asset called EasyRoads3D Pro [10]. This asset provided a good final look to the roads and it contained several useful features, mostly the ability to create roads through scripting. Through an array of three dimensional vectors a road is fitted to match those points. EasyRoads3D Pro also has the function to retrieve the vector of a position along the road given the distance travelled from the start which was helpful when extending the visualization to include steering.

2.3.3 Terrain

To obtain a lifelike look to the program several assets, textures and materials were used. As mentioned earlier, both the road and the vehicles were obtained from assets. As seen in the program there are trees placed in the background and a grass texture on the ground.
2.3.4 UI - User Interface

A user interface was added to clarify the visualization in real-time. The information that was deemed relevant was the current speed and status of each vehicle and the command being sent to the platoon.

2.3.5 Handling Velocity Inputs

The standard method to move objects in Unity is to add forces acting on them and let the built-in Unity physics handle friction, air drag, etc. To make sure that Unity does not interfere with the physics model used in the simulation, all velocity values in the two dimensional plane of the road are explicitly set for each vehicle to override the Unity physics engine. The velocity vector in the y-direction (height) is not altered to avoid the vehicle from floating in the air.

2.3.6 Linear Velocity Approximation

The simulated measuring devices used in the simulation calculates the position and velocity with a sample rate of 5 ms. The perfect solution would therefore be to update the position and velocity of the vehicles in Unity with the same frequency. However, due to performance limitations regarding how often Unity can read the input from the text file, it was not a feasible solution. The way this was handled instead was to set the output rate of the simulation to 500 ms and in the Unity program linearly interpolate the velocity between two measuring points.

2.3.7 Steering

No steering control system is currently implemented in the simulation that was used for this project, but a simplified path-following method was used to keep the vehicle in the middle of the road in the cases using a curved road to showcase the capabilities of the Unity software. As mentioned earlier, EasyRoads3D Pro has a function that returns the position of the road given the distance travelled from the start. To calculate the direction of the road at the position of each vehicle, the coordinates of the road was gathered for both the starting point and the end point as well as 0.5m down the road from these points. The direction of the road was then estimated by subtracting
the starting point vector from the one slightly further forward. The same was done for the ending point yielding two rotations, one for the start of the output and one for the end of the output (500 ms later). A spherical linear interpolation was then used to approximate the rotation of the vehicles as they were driving between the two points since the rotational speed of the vehicle is close to constant during the road section covered during the short timespan between two simulation outputs. The velocities of the vehicles were then set to match to direction that they were facing.

2.3.8 Calculating Visualization Errors

Unity updates the values of all game objects’ position and velocity with a constant frequency, by default with 0.02s between each update. In the case of any lag while running the visualization, an error in position occurs since the velocity does not update during the missed frame. Any errors in the visualization can be fixed by moving the vehicles to the correct position according to the simulation every time a new command is read from the text file, later referred to as position adjustment. When measuring the error in the position between the visualization and the simulation, the distance to the correct position is calculated in the direction of the road (forward), perpendicular to the road (sideways) and in total distance. The maximum value of these errors during the test cases are then saved to measure the accuracy of the visualization.

3 Test Cases

Three different test cases were simulated for comparison and to evaluate the visualization. In all test cases the platoon contained four trucks and each input simulated a time of \( t = 0.5 \) seconds.

In the first test case the simulation simulated emergency braking. The program was set to visualize the simulation on a straight road without using the algorithm for position adjustment.

In the second test case the program generated a curved road where a simulation of the platoon accelerating and braking were visualized. The algorithm for position adjustment was not used in this test case either.

The last test case visualized the same simulation as in the second test case. However, here the algorithm for position adjustment was used.
4 Results

The accuracy of the visualizations can be evaluated both visually and mathematically. For all test cases the difference in position between the visualization and the simulation is determined at each input. In the visualization it is possible to observe the platoon’s movements along the road to visually evaluate how well it performs/follows the desired route. A screen capture of the visualization is presented in figure 3 below. The bar at the bottom presents the command currently being sent and the speed of each vehicle. The spheres above the vehicles visualize the status; green being ‘good’, yellow being ‘tooClose’ and red being ‘crash’. The leader has a white sphere since it has no status.

Figure 3: Screen capture of the visualization

The results of the test cases present the position differences as error values and a brief explanation of the visuals of the visualization.

4.1 Emergency Braking on Straight Road

The visualization of the emergency brake test case resulted in the platoon moving smoothly along the straight road. The highest differences in position for each vehicle are presented in table 2 as the max forward, sideways and total distance error values.
### 4 RESULTS

#### Table 2: Max error values of the position for each vehicle

<table>
<thead>
<tr>
<th></th>
<th>Leader</th>
<th>Follower 1</th>
<th>Follower 2</th>
<th>Follower 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Max forward error (m)</td>
<td>3.05</td>
<td>3.33</td>
<td>3.62</td>
<td>3.71</td>
</tr>
<tr>
<td>Max sideways error (m)</td>
<td>0.14</td>
<td>0.15</td>
<td>0.17</td>
<td>0.17</td>
</tr>
<tr>
<td>Max distance error (m)</td>
<td>3.05</td>
<td>3.33</td>
<td>3.62</td>
<td>3.71</td>
</tr>
</tbody>
</table>

#### 4.2 Curved Road without Position Adjustment

In the visualization of the first curved road test case the platoon moved smoothly along the road. The highest differences in position for each vehicle are presented in table 3 as the max forward, sideways and total distance error values.

#### Table 3: Max error values of the position for each vehicle

<table>
<thead>
<tr>
<th></th>
<th>Leader</th>
<th>Follower 1</th>
<th>Follower 2</th>
<th>Follower 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Max forward error (m)</td>
<td>3.08</td>
<td>3.15</td>
<td>3.71</td>
<td>3.42</td>
</tr>
<tr>
<td>Max sideways error (m)</td>
<td>1.45</td>
<td>1.24</td>
<td>1.32</td>
<td>1.35</td>
</tr>
<tr>
<td>Max distance error (m)</td>
<td>3.27</td>
<td>3.36</td>
<td>3.80</td>
<td>3.49</td>
</tr>
</tbody>
</table>

#### 4.3 Curved Road with Position Adjustment

Now and then in the visualization the vehicles performed a non-continuous motion. Even so the platoon followed the road through each curve. The highest differences in position for each vehicle are presented in table 4 as the max forward, sideways and total distance error values.

#### Table 4: Max error values of the position for each vehicle

<table>
<thead>
<tr>
<th></th>
<th>Leader</th>
<th>Follower 1</th>
<th>Follower 2</th>
<th>Follower 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Max forward error (m)</td>
<td>0.55</td>
<td>0.60</td>
<td>0.66</td>
<td>0.73</td>
</tr>
<tr>
<td>Max sideways error (m)</td>
<td>0.53</td>
<td>0.66</td>
<td>0.50</td>
<td>0.46</td>
</tr>
<tr>
<td>Max distance error (m)</td>
<td>0.71</td>
<td>0.63</td>
<td>0.74</td>
<td>0.78</td>
</tr>
</tbody>
</table>
5 Conclusion

5.1 Discussion of Results

When comparing table 2 to table 3 it is clear that adding the steering algorithm does not increase the maximum error of the total distance traveled. A crash status is indicated whenever vehicles are within two meters of each other, which means that a maximum error in position of approximately 3.5 meters is a bit too much to accurately visualize the simulation. Referring to table 4, when the position adjustment is introduced, the errors decrease to acceptable levels for the purpose of the visualization, but at the cost of non-continuous motion of the vehicles.

5.2 Discussion of Method

Without using the scripting feature in Unity, the roads would have been created with a drag-and-drop method which would not have been nearly as accurate nor that flexible to work with. Using EasyRoads3D Pro was a good decision, it made the procedure easier and it visualized the desired roads more accurately while providing the tools to develop a simple steering algorithm.

The linear velocity approximation used implies that the acceleration between two inputs is constant which is not entirely true since the forces from the wheel friction and the air resistance are both functions of the velocity. These forces are however rather insignificant compared to the constant acceleration from the engine and the brakes. While the approximation gave fairly accurate results, they were not considered good enough for the visualization. Adding position adjustment to decrease the errors, while the motion is not continuous, yielded results that were deemed good enough for visualization since the important information whether or not the platoon crashed is solely handled by the simulation, with no risk of false positives or false negatives due to inaccuracies in the way the Unity program is scripted.

The simple steering algorithm that was created managed to keep the vehicles close to the center of the road as long as it contained no sharp turns and can easily be altered in the future to handle simulations with more advanced steering systems.
5.3 Improvement and Future Work

There are ways to further decrease the error between the visualization and the simulation without the need of position adjustment at each input, when trying to achieve both continuous and accurate movement of the vehicles. One way would be to decrease the time between two inputs from the 0.5 seconds that were used for these results. The problem here is finding the optimal input rate that lowers the error while not causing the entire visualization to lag due to performance issues. Another way to decrease the errors would be to use a higher order interpolation method to approximate the velocity using more than two data points, to account for the non-constant acceleration.

As for now, the platooning program does not have an algorithm for steering or speed adjustments to the curves of the road. Obviously this is an important part and it is probably the next big step in the development of the platooning simulation. Another area is city driving, e.g. the platoon must be able to handle roundabouts, crosswalks and intersections. However, this may not be a priority since a platoon might not be that efficient in urban environments. The platoon would, most likely, be disrupted by other vehicles and the concept of a platoon would fall.

5.4 Final Words

The goal of the project was to create a flexible and accurate visualization, which we consider was achieved. Even so, there are ways to improve the program to decrease the errors. The program was developed to easily add further functionality and to investigate the potential of the Unity Game Engine. To conclude, the project was performed with success.
References


