

**MOTION PLANNING ALGORITHM FOR VEHICLE PARKING  
SIMULATION**

by

Amirul Ehsan b. Mohd Hilmi

Dissertation submitted in partial fulfillment of  
the requirements for the  
Bachelor of Technology (Hons)  
(Information Communication Technology)

JANUARY 2008

Universiti Teknologi PETRONAS  
Bandar Seri Iskandar  
31750 Tronoh  
Perak Darul Ridzuan

# CERTIFICATION OF APPROVAL


## MOTION PLANNING ALGORITHM FOR VEHICLE PARKING SIMULATION

by  
Amirul Ehsan bin Mohd Hilmi

Dissertation submitted in partial fulfilment of  
the requirement for the  
Bachelor of Technology (Hons)  
(Information and Communication Technology)

JANUARY 2008

Approved by,



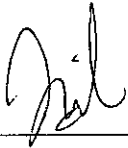
---

(YEY/KWANG HOOI)

Universiti Teknologi PETRONAS  
Bandar Seri Iskandar  
31750 Tronoh  
Perak Darul Ridzuan

## **CERTIFICATION OF ORIGINALITY**

This is to certify that I am responsible for the work submitted in this project, that the original work is my own except as specified in references and acknowledgements, and that original work contained herein have not been undertaken or done by unspecified sources or persons.



---

Amirul Ehsan bin Mohd Hilmi

## **ABSTRACT**

This project implements an intelligent autonomous vehicle parking control system. Intelligence of the system means that the car is capable of analyzing its own environment and act accordingly to it as a human being would do. This project focuses on creation of the motion planning algorithm of the system and developing a simulation to simulate the vehicle movements. The system controls the vehicle based on the vehicle and the target parking space coordinates. It will automatically generate the steering commands in order to park itself inside the parking space. Prototyping method is used in this project, where the prototype of the system is developed as soon as possible and it is then enhanced by adding more functions. The development of the system is using C# .net programming language. The system is successfully working as it has been tested multiple times in the simulation environment.

## **ACKNOWLEDGEMENT**

First and foremost, I would like to thank Allah the Almighty for His blessings that made all things possible while doing this project. I would like to express my deepest gratitude to my parents for their love and support. To my supervisor, Mr Yew Kwang Hooi, thank you so much for the guidance and support that he gave throughout the project. Without his advices and helps, this project may not be able to be completed within the given timeline. I also would like to thank other lecturers for their suggestions to improve the project.

I would also like to thank Matt Kincaid for allowing me to use his 2D car physics simulation to integrate it with my algorithm.

Last but not least, I would like to thank all my friends for the help and support that they have given me throughout the period of this project. They are the one who tested my system and with their feedback and suggestions I manage to achieve what I have now.

## TABLE OF CONTENTS

<b>CERTIFICATION OF APPROVAL .....</b>	<b>I</b>
<b>CERTIFICATION OF ORIGINALITY.....</b>	<b>II</b>
<b>ABSTRACT.....</b>	<b>III</b>
<b>ACKNOWLEDGEMENT.....</b>	<b>IV</b>
<b>TABLE OF CONTENTS .....</b>	<b>V</b>
<b>LIST OF FIGURES .....</b>	<b>VII</b>
<b>CHAPTER 1: INTRODUCTION.....</b>	<b>1</b>
1.1 Background study .....	1
1.2 Problem Statement.....	2
1.2.1 Problem Identification .....	2
1.2.2 Significant of the Project .....	2
1.3 Previous Work .....	4
1.4 Objectives and Scope.....	4
1.3.1 Objectives .....	4
1.3.2 Scope of Study .....	4
<b>CHAPTER 2: LITERATURE REVIEW.....</b>	<b>6</b>
2.1 Comparison with Related Works.....	6
2.2 Artificial Neural Network.....	8
2.2.1 Feed-Forward Neural Network Model.....	9
2.2.2 Back Propagation (BP) .....	11
2.3 Fuzzy Logic .....	12
2.3.1 Advantages of Fuzzy Logic .....	13
2.4 Cars Physics .....	15
<b>CHAPTER 3: METHODOLOGY.....</b>	<b>17</b>
3.1 Planning .....	18

3.3 Design .....	21
3.4 Development .....	27
3.5 Implementation and Prototyping.....	30
<b>CHAPTER 4: RESULT AND DISCUSSION.....</b>	<b>31</b>
4.1 Result .....	31
4.1.1 Simulation environment setup .....	31
4.1.2 Automated Parking Result .....	35
4.2 Discussion .....	39
<b>CHAPTER 5: RECOMMENDATION AND CONCLUSION .....</b>	<b>41</b>
5.1 Recommendation .....	41
5.1.1 Test with real vehicle .....	41
5.1.2 Develop the Vision System.....	42
5.2 Conclusion .....	43
<b>REFERENCES.....</b>	<b>45</b>
<b>APPENDIX.....</b>	<b>48</b>

## LIST OF FIGURES

Figure 2.1: Basic Neural Network	10
Figure 2.2: Neural Network with weighted inputs and limiter	10
Figure 2.3: Range of logical value for Boolean	12
Figure 2.4: Range of logical value for Fuzzy Logic	12
Figure 3.1: Prototype Model	17
Figure 3.2: Gantt Chart for the project	18
Figure 3.3: Basic Process Flow	21
Figure 3.4: Process Flow for the whole system	23
Figure 3.5: Process Flow in Auto-Park	25
Figure 4.1: Screenshot of simulation	32
Figure 4.2: Dropdown Menu of the Simulation	34
Figure 4.3: Initial state of the car	36
Figure 4.4: The car steer to the right	36
Figure 4.5: The car will steer straight	38
Figure 4.6: The car in the parking space	38



# CHAPTER 1

## INTRODUCTION

### CHAPTER 1: INTRODUCTION

#### 1.1 Background study

The Autonomous Vehicle and Robots (Autonomous Mobile Robots – AMR) have attracted a great number of researches due to the great challenge that this new domain of research offers: to endow these system with an intelligent reasoning capability, exploring their abilities to interact with the environment where they are inserted. The AMRs will recognize the environments through their sensors (e.g. Infrared, sonar, lasers and cameras) and from the information obtain, they will be able to plan and execute their actions.

Nowadays, mobile robots are used in a variety of different areas. A few examples are, to locate and disarm bombs, to transport materials, to explore hostile environment such as volcanoes and also other planets terrains.

## **1.2 Problem Statement**

### **1.2.1 Problem Identification**

For automated control of vehicular motions, there are basically two typical scenarios: a) driving on road and b) automatic parking. For the first scenario, quite a number of solutions have been developed for the case of following a lane or lane changing. The second scenario is more difficult to implement due to the space is less structured and more skills are required to plan the motion. Even for human beings, some still have difficulties in properly parked their cars. It is not an easy task to develop automatic parking skills for a car. The car needs to strictly follow a few sets of rules.

Besides that, there are only a few well documented research papers that are being published for the public usage regarding this topic. Even though all of us already know the big automobile company has already developed this type of technology, but their research papers are still being kept secret.

### **1.2.2 Significant of the Project**

The goal of this project is to come up with a working algorithm that will automatically park the vehicle. This will be greatly beneficial to the automobile industry and also the public. Consumer will surely want a car that can allow them to avoid the hassle of having to adjust their car to enter the parking space. And the automobile industry can moved one step ahead in term of technology and develop more advance features for a car in the future.

This research paper will also be available to everyone. This is to help other people that have the same intention as the author, which would like to pursue the research of more new technologies in this field.

### **1.3 Previous Work**

The researches on The Automated Guided Vehicle (AGV) have already been done by some research groups and companies (BMW, Mercedes-Benz, Toyota, and GMC) in the USA, Japan, UK, Italy, Germany and France. One of the outstanding studies was by the INRIA researchers, which implemented a control system used to park a vehicle in an autonomous way. They built up complete sensor based control architecture for the vehicle. They use the model-based approach that decomposes the motion into a number of “parallel parking” series. At each step of the motion, the orientation of the vehicle is identical at the beginning and at the end.

### **1.4 Objectives and Scope**

#### **1.3.1 Objectives**

- To develop an autonomous motion planning system that will allow a mobile vehicle to automatically park itself efficiently.

#### **1.3.2 Scope of Study**

The study will focus on developing the algorithm that will allow the car to autonomously move into empty parking space and avoiding any existing obstacles between the car and the parking space. This project will put greater emphasis only on the algorithm of the car movements. Due to the restricted amount of time and resources, this paper will not be covering the car's Vision System.

The algorithm will do the following tasks:

1. Planning the motion
2. Simulate the motion
3. Provide the best motion with minimal efforts

There are a few assumptions made during the development of this algorithm:

1. Since, I will not be covering the car's Vision System; the car will be given the positions of the empty parking space beforehand as if it has foreseen it.
2. The car will also be given the coordinates of its center point and the front point of the car.

A simulation of this project will be developed as to prove that the algorithm proposed is working and efficient. The project will be developed by using Microsoft Visual Studio 2008, using C#.Net programming language.

## **CHAPTER 2**

### **LITERATURE REVIEW**

#### **CHAPTER 2: LITERATURE REVIEW**

##### **2.1 Comparison with Related Works**

As mentioned earlier, one of the earliest outstanding studies was by the INRIA researchers, which implemented a control system used to park a vehicle in an autonomous way. The vehicle was equipped with 14 sonar sensors and it used a customized set of rules to accomplish the tasks.

Since this project will not be covering the Vision System for the car to be compared with, the author will straightly go to the motion planning. It is assumed that based on the information of the parking space positions and obstacles obtained from the Vision System, a local map will be built. The next step should be the motion planning of the car. This motion problem is still being debated by many researchers. According to Jin Xu, Guang Chen and Ming Xie (n.d.), there are two different categories of approaches.

The first category is skill-based. The control commands are generated in real time according to the current state. Fuzzy Logic and Neural Network are used to transfer skills of human beings to an intelligent vehicle. The second category aims to plan the whole motion series in advance and then send these commands consequently to the controllers to act. The project by INRIA falls under the second category while the propose solutions in this paper falls under the first category.

For the INRIA project, when the car finds a parking space with enough size, the system will estimate the beginning maneuver positions and moves the vehicle back to that position. Once the vehicle is correctly positioned, the parking maneuver will start. This task is accomplished by the usage of sinus based functions. The car movement has already been defined in the first place. M.R. Heinen, F.S. Osorio, F.J. Heinen and C. Kelber (2006) say that by using this sinus based functions to control the vehicle will produce softer and smoother movements. They said that the drawback for this technique is that it requires a curb barrier installation so that it can calculate the depth of the parking space; which restricts the usage of this method on conventional streets. They also said that this algorithm is quite limited because it only works specifically in parallel parking tasks and it needs to be manually coded.

The author's proposed solution is to break the whole motion down into several more steps instead of just one. This will allow the system to re-evaluate its own environment and then create a new motion movement. The cars will be shown a few sets of examples on how to perform parking motions correctly. The usage of back propagation in Artificial Neural Network (ANN) will allow the car to learn from this set of examples. The usage of fuzzy logic will allow the car to make the decisions on which route to follow. This will allow the car to react to any unexpected situations.

## 2.2 Artificial Neural Network

According to Christos Stergiou and Dimitrios Siganos (n.d.), an Artificial Neural Network (ANN) is an information processing paradigm that is inspired by the way biological nervous systems, such as the brain, process information. The key element of this paradigm is the novel structure of the information processing system. It is composed of a large number of highly interconnected processing elements (neurones) working in unison to solve specific problems. ANN, like people, learns by example. An ANN is configured for a specific application, such as pattern recognition or data classification, through a learning process. Learning in biological systems involves adjustments to the synaptic connections that exist between the neurones. This applies to ANN as well.

From Christos Stergiou and Dimitrios Siganos (n.d.), neural networks, have a remarkable ability to derive meaning from complicated or imprecise data, can be used to extract patterns and detect trends that are too complex to be noticed by either humans or other computer techniques. A trained neural network can be thought of as an "expert" in the category of information it has been given to analyse. This expert can then be used to provide projections given new situations of interest and answer "what if" questions.

Other advantages include:

1. Adaptive learning: An ability to learn how to do tasks based on the data given for training or initial experience.
2. Self-Organisation: An ANN can create its own organisation or representation of the information it receives during learning time.
3. Real Time Operation: ANN computations may be carried out in parallel, and special hardware devices are being designed and manufactured which take advantage of this capability.

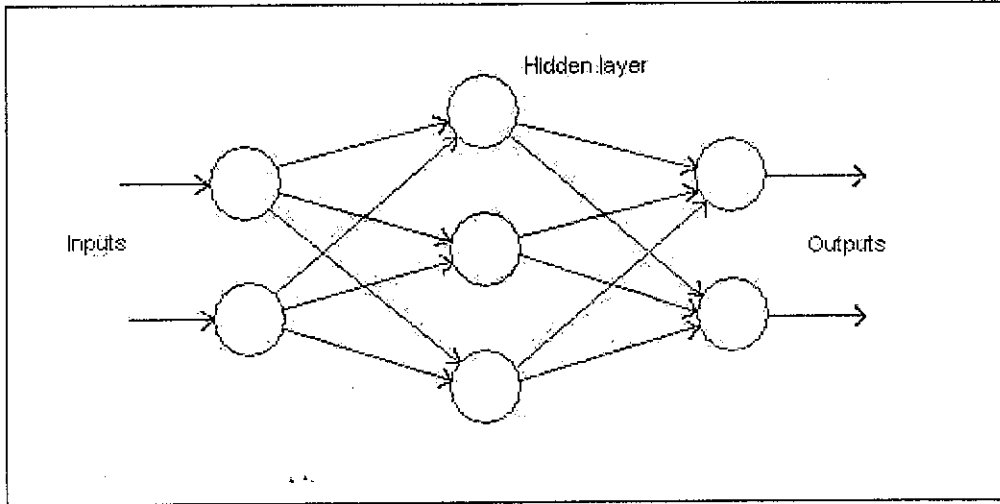


4. Fault Tolerance via Redundant Information Coding: Partial destruction of a network leads to the corresponding degradation of performance. However, some network capabilities may be retained even with major network damage.

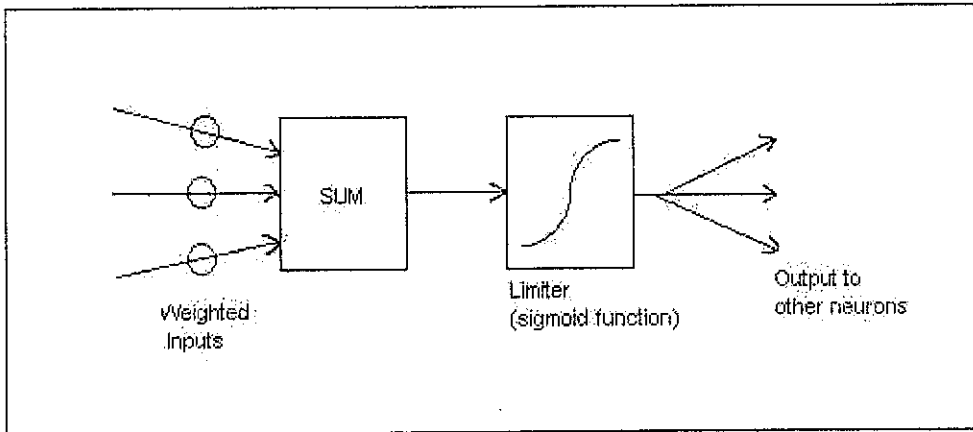
### **2.2.1 Feed-Forward Neural Network Model**

Feed-forward neural network is an artificial neural network where connections between the units do not form a directed cycle. The feed-forward neural network was the first and arguably the simplest type of artificial neural network devised. The information moves in only one direction, forward, from the input nodes, through the hidden nodes (if any) and to the output nodes.

Referring to **Figure 2.1** and **Figure 2.2**, each neurons will received a signal from the previous layer of neuron, and each of those signals is multiplied by a separate weight value. The weighted inputs are summed, and passed through a limiting function which scales the output to a fixed range of values. The output of the limiter is then broadcast to all of the neurons in the next layer. So, to use the network to solve a problem, we apply the input values to the inputs of the first layer, allow the signals to propagate through the network, and read the output values. For **Figure 2.1**, the stimulation is applied to the inputs of the first layer, and signals propagate through the middle (hidden) layer(s) to the output layer. Each link between neurons has a unique weighting value. For **Figure 2.2**, inputs from one or more previous neurons are individually weighted, then summed. The result is non-linearly scaled between 0 and +1, and the output value is passed on to the neurons in the next layer.



**Figure 2.1: Basic Neural Network**



**Figure 2.2: Neural Network with weighted inputs and limiter**

### **2.2.2 Back Propagation (BP)**

Since the real uniqueness or “intelligence” of the network exists in the values of the weights between the neurons, a method to adjust the weights to solve problems is needed. The most common type of learning algorithm is called Back Propagation (BP). It was first described by Paul Werbos in 1974, and further developed by David E. Rumelhart, Geoffrey E. Hinton and Ronald J. Williams in 1986.

A BP network learns by example, it means that we must provide it with a learning set that consists of some input example and the known correct output for each of those cases. These input-output examples are needed to show the network what types of solutions are expected and the BP algorithm will allow the network to adapt.

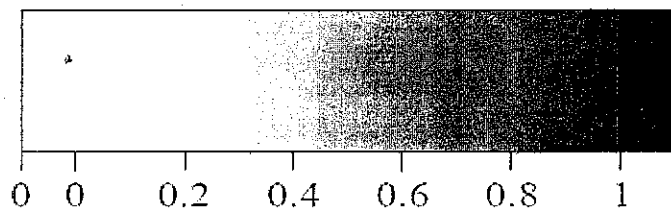
### 2.3 Fuzzy Logic

Fuzzy logic is derived from fuzzy set theory dealing with reasoning that is approximate rather than precisely deduced from classical predicate logic. It can be thought of as the application side of fuzzy set theory dealing with well thought out real world expert values for a complex problem. The main reasoning behind this theory is that people do not require precise, numerical information input, yet they are capable of highly adaptive control. Fuzzy Logic provides a simple way to arrive at a definite conclusion based on vague, ambiguous, imprecise, noisy or missing input. Fuzzy Logic mimics how we as human make a decision, only much faster.

Unlike the Boolean Logic that has only two values that is 1 and 0, Fuzzy Logic is multi-valued. It deals with degrees of membership and degrees of truth. Fuzzy Logic uses the range of logical values between 0 (completely false) and 1 (completely true). **Figure 2.3** and **Figure 2.4** shows the range of logical value for Boolean and Fuzzy logic respectively.



**Figure 2.3: Range of logical value for Boolean**



**Figure 2.4: Range of logical value for Fuzzy Logic**

Fuzzy Logic incorporates a simple, rule-based IF X AND Y, THEN Z, or constructs that are equivalent approach to solve problems rather than attempting to model a system mathematically. As an example, we take a look at a simple temperature regulator that uses fan.

IF temperature IS hot THEN speed up fan

IF temperature IS normal THEN maintain fan speed

IF temperature IS cold THEN slow down fan

IF temperature IS very cold THEN stop fan

These terms are imprecise yet can be very descriptive on what should happen. Fuzzy Logic is capable of determining the decisions when given this set of terms.

### **2.3.1 Advantages of Fuzzy Logic**

Below are the advantages of using Fuzzy Logic (FL) according to Steven D. Kaehler (n.d.):

1) It is inherently robust since it does not require precise, noise-free inputs and can be programmed to fail safely if a feedback sensor quits or is destroyed. The output control is a smooth control function despite a wide range of input variations.

2) Since the FL controller processes user-defined rules governing the target control system, it can be modified and tweaked easily to improve or drastically alter system performance. New sensors can easily be incorporated into the system simply by generating appropriate governing rules.

3) FL is not limited to a few feedback inputs and one or two control outputs, nor is it necessary to measure or compute rate-of-change parameters in order for it to be implemented. Any sensor data that provides some indication of a system's actions and reactions is sufficient. This allows the sensors to be inexpensive and imprecise thus keeping the overall system cost and complexity low.

4) Because of the rule-based operation, any reasonable number of inputs can be processed (1-8 or more) and numerous outputs (1-4 or more) generated, although defining the rulebase quickly becomes complex if too many inputs and outputs are chosen for a single implementation since rules defining their interrelations must also be defined. It would be better to break the control system into smaller chunks and use several smaller FL controllers distributed on the system, each with more limited responsibilities.

5) FL can control nonlinear systems that would be difficult or impossible to model mathematically. This opens doors for control systems that would normally be deemed unfeasible for automation.

## 2.4 Cars Physics

This project involves in planning the motion of the car, and also creating a simulation for it. Therefore, the car physics is an important thing to take into consideration. When talking about motion, everything will be related to the famous Newton's Law of motion.

1. Law 1 – A body tend to remain at rest or continue to move in a straight line at a constant velocity unless it is acted upon by an external force. This is the concept of inertia.
2. Law 2 – The acceleration of a body is proportional to the resultant force acting on the body, and this acceleration is in the same direction of the resultant force.
3. Law 3 – For every force acting on a body (action) there is an equal and opposite reacting force (reaction) in which the reaction is collinear to the acting force.

(David M. Bourg, 2002, p.1)

In the study of dynamics and motion, one particular formula that is often used that is based from the second law;  $F = mA$ , where  $F$  is force,  $m$  for the mass and  $A$  is for the acceleration.

For the simulation, each frame of the simulation, the system will accumulate the force from the wheels of the vehicle and will then calculate the resultant acceleration. So, the formula will be as below:

$$A = F / m$$

Based on this formula, the first Newton law can now be modified. The acceleration will be calculated and integrate it in the velocity. Without the acceleration, the velocity would be constant; hence the car will stay in motion, if no forces should act on it.

Next, the third Newton Law will also be applied in the simulation; any potential force the vehicle is applying to the ground, gets applied in the opposite direction of the vehicle.

According to Matt Kincaid (n.d.), with a constant mass, and some generated forces, acceleration will be generated, which in turn will generate velocity, which will then generate the displacement (vehicle position). So, the given formula is as below where, A is the acceleration, F is the net force applied to it, m is the mass, P is the vehicle position, V is its linear velocity and T is the time step generated by the system:

$$A = F / m$$

$$V = V + A * T$$

$$P = P + V * T$$

Next is the rotation of the vehicle. Since this is 2D simulation, therefore the angular cases is quite similar to the linear case. Based on the formula above, instead of P, there is an Angle, instead of V, there is an angular Velocity, instead of a, there is an angular acceleration, instead of F, there is Torque and m will be the inertia. According to Matt Kincaid (n.d), the formula should look like this:

$$\text{AngA} = \text{Torque} / \text{Inertia}$$

$$\text{AngV} = \text{AngV} + \text{AngA} * T$$

$$\text{Angle} = \text{Angle} + \text{AngV} * T$$

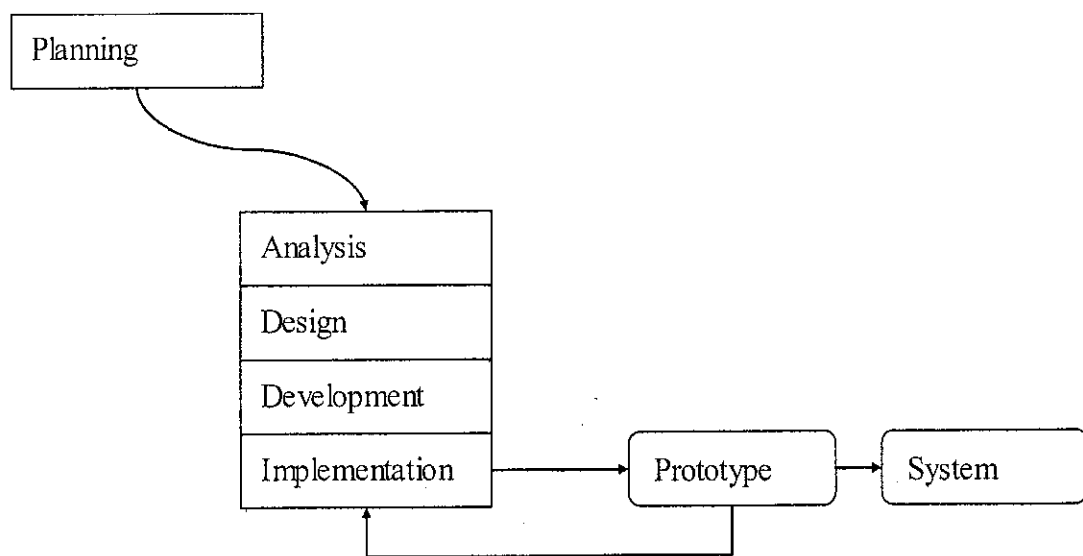


# CHAPTER 3

## METHODOLOGY

### CHAPTER 3: METHODOLOGY

As for the methodology, the author has decided to use the System Development Life Cycle (SDLC), Prototype model as shown in **Figure 3.1**.



**Figure 3.1: Prototype Model**

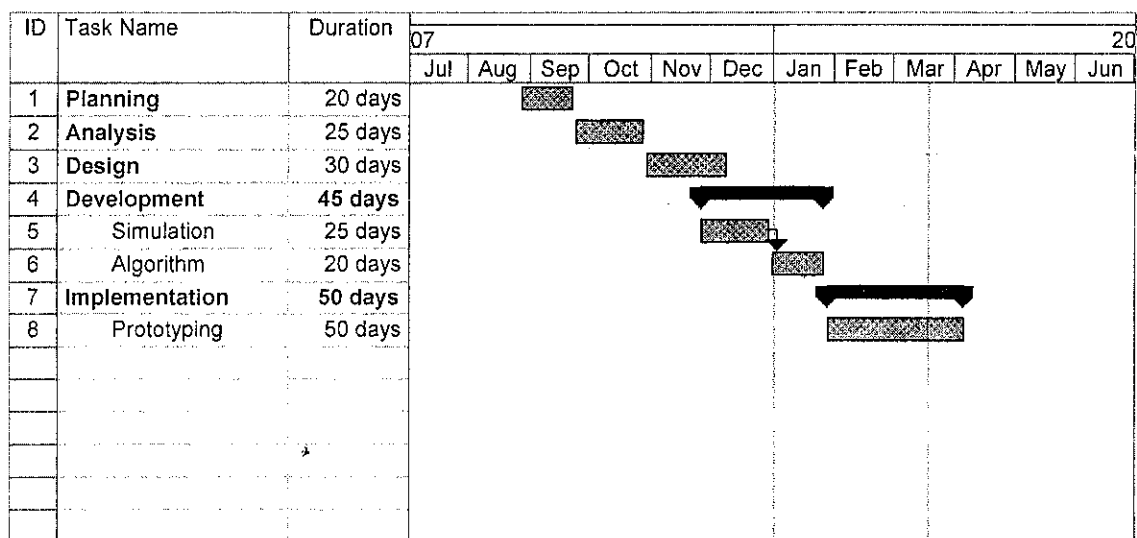
By choosing this model, the author plans to come out with a simple and functional prototype of the system as soon as possible. The prototype will be fully tested, and any more necessary functions will be added. And from then on, the prototype will be tested, added with more functions and refined to perfection. The author plans to have at least 3 or 4 prototypes.

### 3.1 Planning

During the planning stage, the author has defined the system that is going to be developed. This basically means that, the author has identified the problem statements, the scope of project, and the exact tasks that the system is required to do.

The scope of the project has also been identified during this planning stage. As mention earlier in this report, the scope of the project is focus more on the development of the algorithm. This project will not cover the Vision System of the car as it will require great amount of resources and time. This project also will only prove the algorithm by using simulation and not the actual vehicle.

In order to properly manage the project plan, the author has created a Gantt chart and divides the project into a few phases. This will allow easier project management and monitoring because the author will only need to focus on only certain tasks. **Figure 3.2** shows the Gantt chart for the project:



**Figure 3.2:** Gantt Chart for the project

Each phase has its own milestone. The phases and its milestones are as shown below:

1. Phase 1 – Come out with the system requirements.
2. Phase 2 – Develop the algorithm and simulation. Thoroughly test them
3. Phase 3 – Integrate the algorithm into the simulation and test it to prove that the algorithm is working properly.
4. Phase 4 – Enhancing the prototype

The milestone for each phase needs to be accomplished first before the project can proceed to the next phase.

The software that will be used in this project will be the Microsoft Visual Studio 2008 as this system will be programmed using the C# programming language.

### **3.2 Analysis**

The analysis stage is where the research regarding this project is being done. There are quite a number of subjects need to studied first in order for the author to fully understand the topics that will be involves in this project. The author need to full understands regarding Artificial Intelligence, analogy of the car, and also vectors.

It is identified that the system will consists of three major parts. The first one is that the car needs to have its own self awareness. This means that the car should be able to determine its position and its orientation. The car should be able to evaluate and sense its surrounding to find the empty parking space and also to detect any obstacles. All of this information is considered as the input for the car.

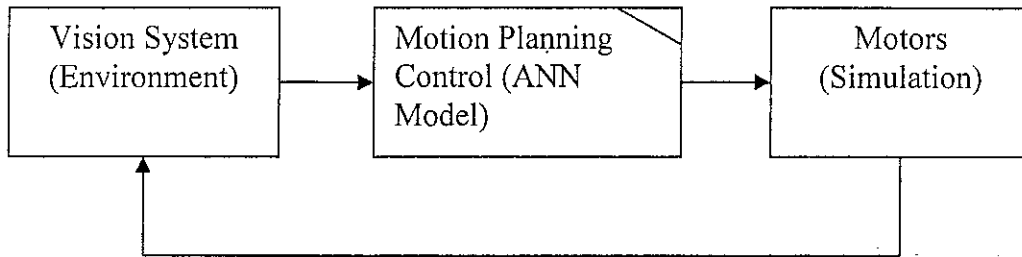
Next, the car should be able to plan the proper route based on the information that it has obtained earlier. This is basically the processing part of the system. The car should be able to plan its motion relevant to the input. For this part, the Neural Control Architecture will play an important role.

Finally, the car should be able to execute the motion planned. This is the output part of the system. In this part, the author needs to know and master the analogy of a realistic car motion. With all its restrictions and the correct trajectory needs to be considered for this project to succeed.

From the analysis and research done, the final findings for the requirements for the algorithm have been identified. It is required that the car should be able to:

1. Determine its own position and its orientation
2. Determine the empty parking space position
3. Calculate the distance between the car and the empty parking space.
4. Detects any obstacle between the car and the parking space.
5. Calculate its path from the given input.
6. Plan the motion of movement by its own.
7. Execute the planned motion properly.

### 3.3 Design



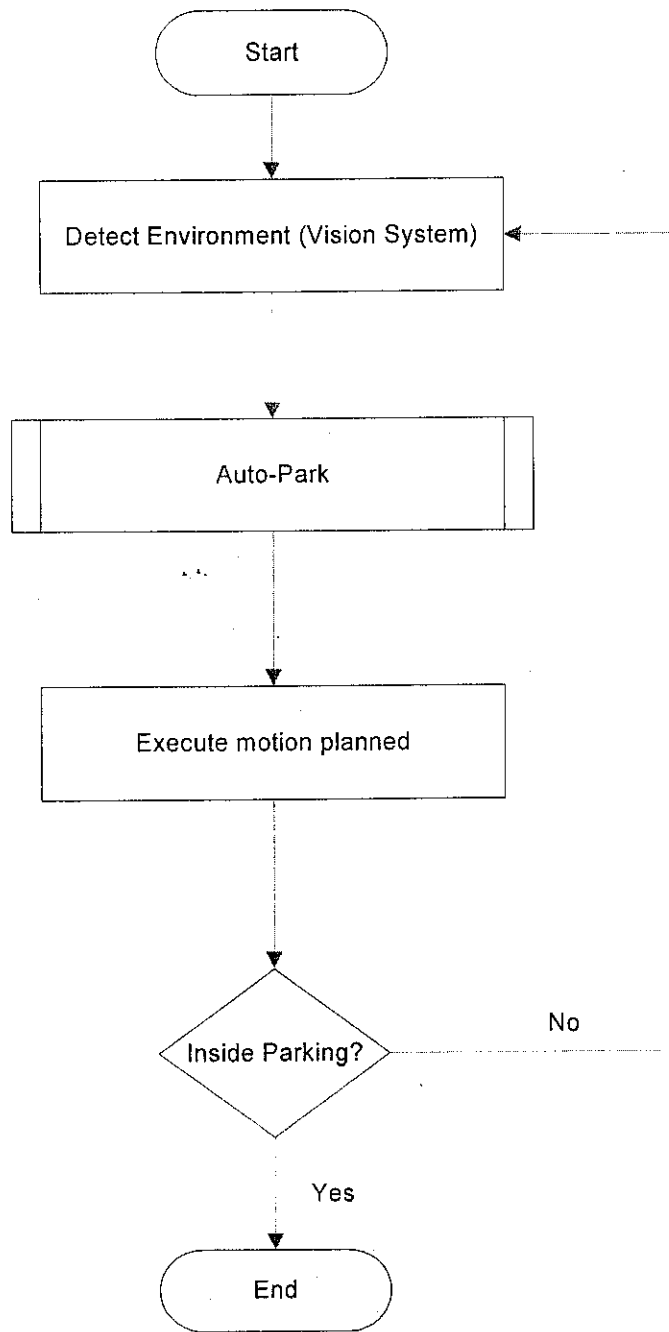
**Figure 3.3: Basic Process Flow**

Figure 3.3 shows the basic process flow of the system. Based on the requirements, it is identified that the car must have its self awareness. Where it can determine its position and orientation and also determine and detects its environment. This is the part that the Vision System will handle. Since the Vision System is not going to be developed, the author has discussed this matter with the instructor Mr. Yew Kwang Hooi to find some other alternatives for this matter.

From the discussion, an alternative way of giving the car its self awareness is by creating a grid view of everything in the simulation. Base on the grid value, the center point for the car will be given as the input to the system. This will allow the car to know its position. Next, in order to determine the car orientation, another point will need to be given to the system; this point will be the front point of the car. From these two points, the car can now know where its position is and where it is facing (orientation). The car's orientation will be calculated in form of degrees. This means the cars angle from the parking space. The size of the car is fixed from the start. Therefore, from these two points, the length and the width of the car can be determined.

Once the car knows its orientation, now it will sense its surrounding by using the Vision System. This Vision System will detect any empty parking space and will give the coordinates to the car. To emulate this case in the simulation, the car will directly be given the coordinates of the parking space and the obstacles. From the obtained coordinates, the car's algorithm will then calculate the area of the parking space whether it is big enough for the car to be parked inside it. The car will then calculate its distance from the parking space.

Before the car executes the planned motion, we need to know the possible vector for the car. In order to make this simulation as realistic as possible, the study of the real car movement is required. The car physics has been described earlier in the literature review and will be used for the simulation.

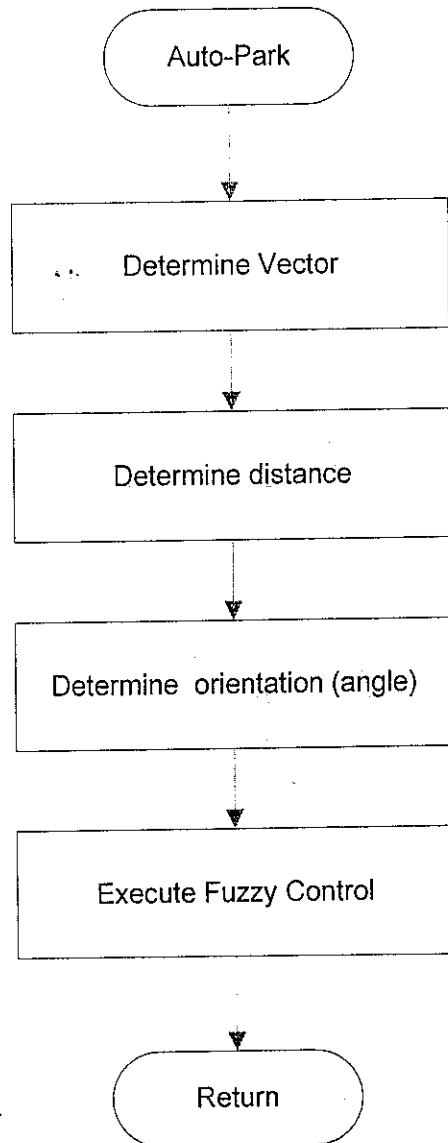


**Figure 3.4: Process Flow for the whole system**

Based on **Figure 3.4**, once the system has detected its environment and obtain all the necessary inputs, it will then find the best possible route that is to be embarked by the car. For the possible movement solutions, the author will create a few possible solutions depending on the situations. Different situations required different solutions. Such as, when is it better to do a reverse side parking, when to just go ahead and park from behind. This scenario is created in order to help the car to select which is better and it will then follow the solutions given. But there is a major weakness in this method. It is impossible to list down all the possible solutions as the as it will consume a lot of time.

The idea to overcome this is to break down the solutions into smaller ones. These small solutions, when combined, will provide the complete route for the car to the parking space. To implement this, the author will create a few simple points or route that the car can follow before it enter the parking space. These points or route will allow the car to calculate and analyze which is the correct and nearest route that the car should take. From this few given point, the car will later learned which route that it will need to take in order to reach the goal. **Figure 3.5** summarizes the process flow on how the vehicle will find the correct possible route.





**Figure 3.5:** Process Flow in Auto-Park

Once the correct path is selected, the car will then execute the path plan. It will then check whether it has completely park itself in the parking space or not, if not, then the system will again detect its environment, and repeat the process of finding the correct solution until the car is correctly park itself inside the parking space.

### 3.4 Development

For the development stage, 2 things will be developed. The first one is the simulation and its environment. The simulation will need to be created first before the algorithm. In the simulation there is a car, and a parking space. The car should have all the right physics and vectors assigned to it, as this will make it more realistic. This is quite a challenge for the author as the author has only learned physics during the secondary school period. Luckily, the author was able to find a simulation sample posted by Matt Kincaid in his tutorial to 2D Game Physics. The author decided to use the simulation as the basis and will then on improved what has been done by Matt Kincaid. The use of the simulation has been approved by Matt Kincaid himself in an email attached in the appendix.

The simulation is being created first instead of the algorithm is because, the environment is really important since the motion planning system can only calculate and plan the motion based on the inputs that the system obtain from the environment. Therefore, the author needs to identify earlier what are the inputs that are going to be available for the vehicle to capture. Based on this few information, the author is able to decide on what the control system will basically take in as the input and the system will respond to these inputs only. This allows the author to focused on the desired inputs rather than catering for all the inputs that are unnecessary.

Next, the algorithm for the control system is developed. This is the tricky part as it requires the system to process the value of all the obtained input and then plan on what is the car is going to do next. How the car will move is planned properly based on the inputs. The car's movements will be restricted to its physical boundaries and the control system will need to consider this before planning the motion of the car.

As for the algorithm for the project, the author has decided to use fuzzy logic to make the decisions. The input for the system will be:

- Coordinates of the center point of the car
- Coordinates of the center point of the parking space
- Angle

From the two coordinate points, the system will then calculate where the parking space is located, whether it is at the left hand side, or right hand side. The angle will be used to determine where the car is currently facing. Here is the if – else code for checking the system:

```
if ((car_angle >= -0.05 && car_angle <= 0.05) && (VectorX >= -3.0
&& VectorX <= 3.0))
{
    if (vehicle.m_velocity.X >= 8 || vehicle.m_velocity.Y >= 8
|| vehicle.m_angularVelocity >= 8)
        throttle = no_throttle;
    else
        throttle = full_forward;
}
else if (VectorX <= -20)
{
    if (vehicle.m_velocity.X >= 8 ||
vehicle.m_velocity.Y >= 8 || vehicle.m_angularVelocity >= 8)
        throttle = no_throttle;
    else
        throttle = semi_forward;

    steering = right;
    if (car_angle <= -1.3)
        steering = straight;
}
else if (VectorX >= 20)
{
    if (vehicle.m_velocity.X >= 8 ||
vehicle.m_velocity.Y >= 8 || vehicle.m_angularVelocity >= 8)
        throttle = no_throttle; steering = left;

    if (car_angle >= 0.05)
        steering = right;
    else if (car_angle <= -0.05)
        steering = left;
    else
        steering = straight;
}
```

The system will keep on looping for this section of codes to continuously check where and how to move the vehicle around. The system will also continuously check whether the car is inside the parking space or not by repeating this line of code every time the graphics is being rendered.

```
float x = vehicle.m_position.X;
float y = vehicle.m_position.Y;

if (y > ParkPointA.Y + 8 && y < ParkPointC.Y - 8 && x > ParkPointA.X +
1.5 && x < ParkPointB.X - 1.5)
inParking = true;
```

### **3.5 Implementation and Prototyping**

This is the stage where the developed algorithm will be integrated into the simulation that has been created in the previous stage. Basically, when the development process of the system is finished, it will become the prototype system only. The whole system can now be fully tested to prove whether the algorithm is correct or there are still some errors. The prototype system then is tested by the author and other people to get their feedbacks regarding the system's performance.

Based on these feedbacks, the author will go back to the analysis phase to analyze all the new improved requirements or functions that needs to be added into the system. Next is the design phase, where the author makes a few changes in the system design to cater for the new functions and requirements added. These cycles are going to be repeated a few times until the prototype is operating at an optimal performance with less bugs and errors.

## CHAPTER 4

### RESULT AND DISCUSSION

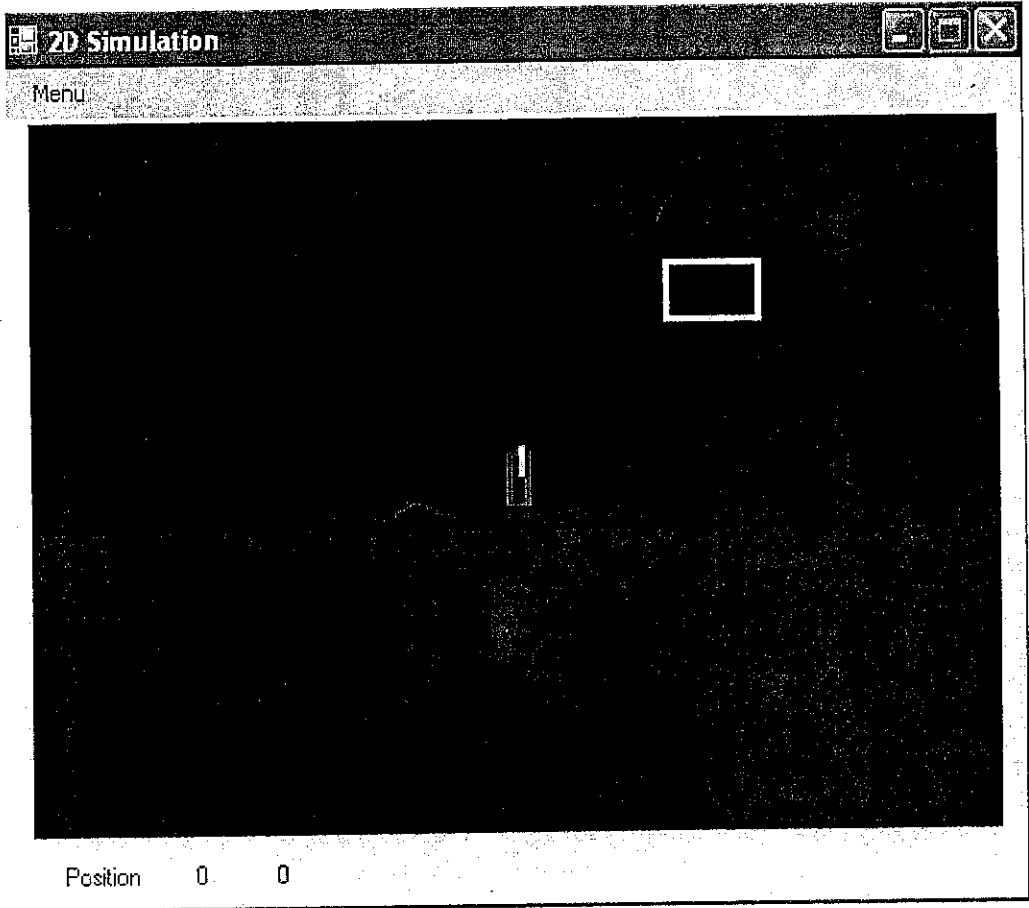
#### CHAPTER 4: RESULT AND DISCUSSION

##### 4.1 Result

The prototype that has been developed has been producing quite an excellent result. The simulation environment which is the parking space, the car and also the car's physics engine has been properly setup to allow the system to run. The algorithm is working brilliantly. The car now is able to park itself inside the parking system without any problems.

##### 4.1.1 Simulation environment setup

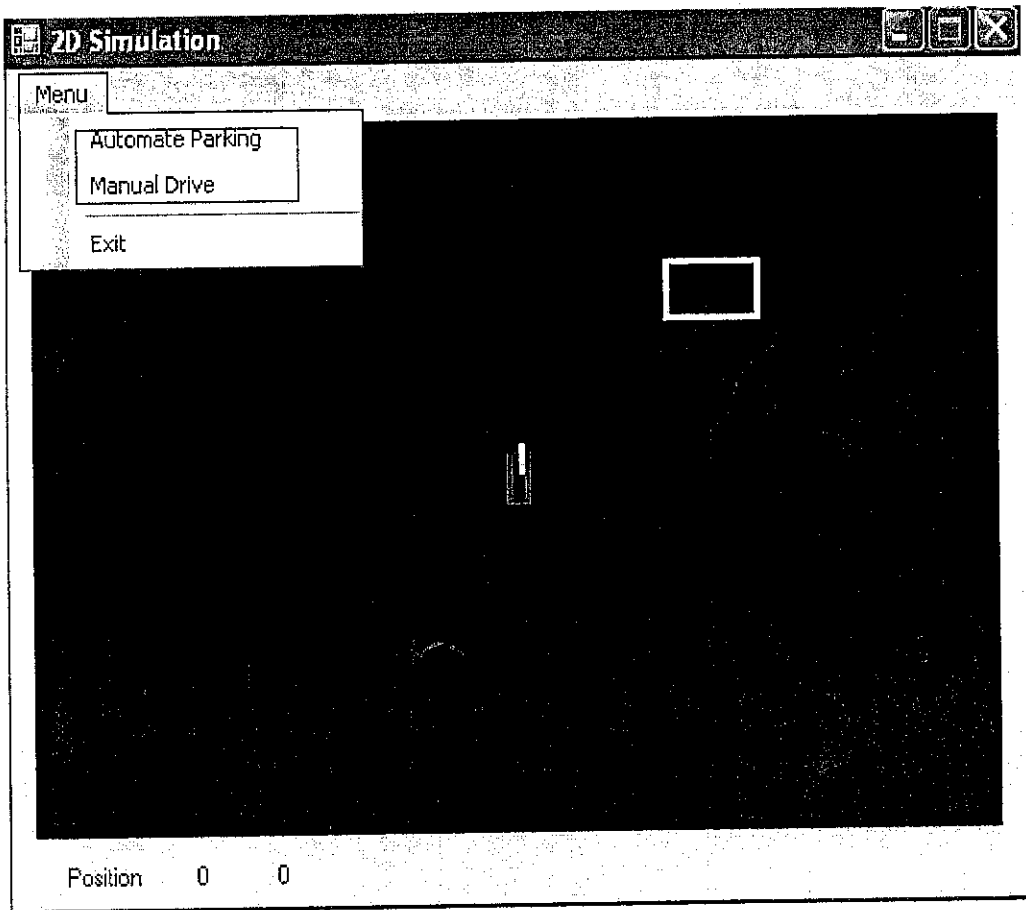
The simulation environment for the system has been properly setup where most of the required variable will be made available to the car to process and reacts based on this information. The simulation contains a car and an empty parking space setup in a flat 2D background. The car has a marker to indicate that that is the front side of the car. Since it can be quite confusing which is the front and which is the rear without the indicator. The placement of everything in the simulation world is based on the coordinates that is based on the car's initial position. To simplify the matters and to make for easier calculations for the physics of the car, the car's center of gravity initial position will be the origin of the coordinates. Therefore, the car's initial position is where the coordinate of (0, 0) will be. **Figure 4.1** shows the simulation background with the car and the empty parking space is in position.



**Figure 4.1: Screenshot of simulation environment**



As shown in **Figure 4.1**, notice that on the bottom left side of the windows form is the position of the car. The initial position for x – axis and the y – axis will be both 0. **Figure 4.2** below shows the selectable options from the drop down menu. When the user wanted to park the car, the user will need to select the “automate parking” option from the menu. By default, the system is in the manual drive state where the user will be able to control the car using the up, down, left, right arrow button and also the shift button. The up arrow button will throttle the car forward while the down arrow button will make the car reverse. Left and Right arrow button will steer the car to the left and right respectively. The shift button is the brake button and the car will stop when the shift button is held.



**Figure 4.2: Dropdown Menu of the Simulation**

#### 4.1.2 Automated Parking Result

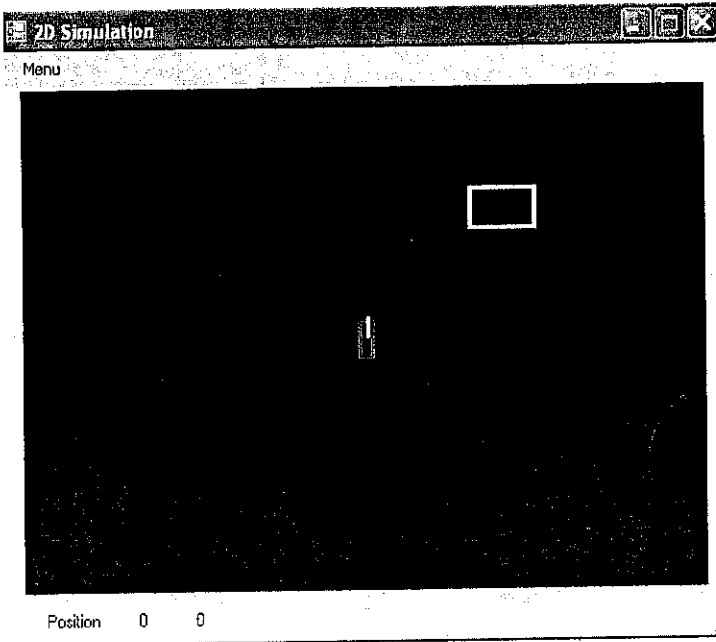
After the simulation environment is completely developed, it is time to develop the algorithm for the parking system. The algorithm is considered to be a successful one as the system is able to park the car inside the empty parking space successfully. The system only requires these two inputs from the environment:

The (X, Y) coordinates of the car

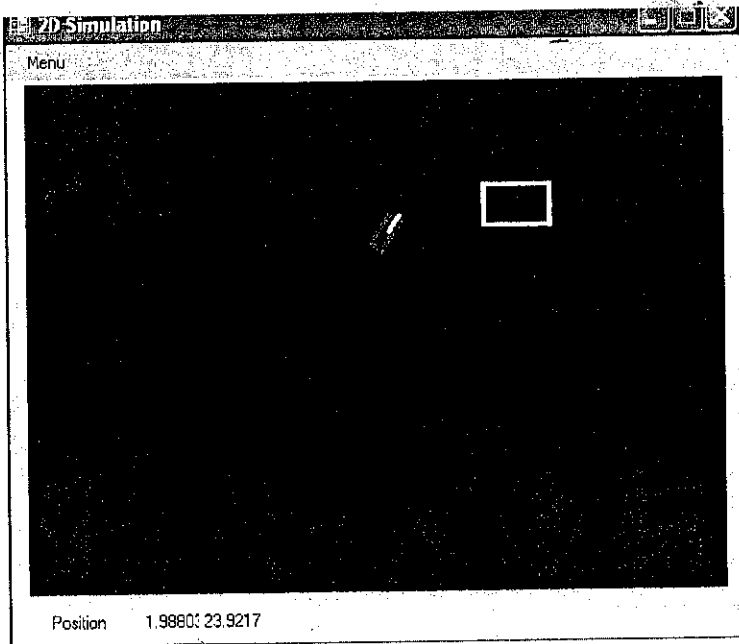
The (X, Y) coordinates of the parking space

From these points, the system is able to project the size of the car and also the size of the parking space. The distance of the car from the parking space and also its orientation from the parking space can be calculated by calculating the vectors of these two points. Let say for example, the car is at the origin, while the coordinates for the parking space is at (10, 25). So the vectors between the two coordinates would be  $10i + 25j$ . This means that the parking space is 25 grids in front of the car, and 10 grids to the right hand side of the car. If the value is  $-Xi + -Yj$ , we can safely say that the parking space is located X grids to the left hand side of the car and Y grids to the rear of the car.

Once the car knows the distance between itself and the parking space, the car is able to compute and calculate its next movements. The system will need to check for certain conditions and if meets the condition, the system will then select the suitable actions.



**Figure 4.3:** Initial state of the car



**Figure 4.3** shows the initial state of the car. During this state the car will scan its surrounding looking for any empty parking space. But in this case the coordinates of the parking space is being given to the car. From this information, the car will then calculate the vectors between itself and the parking space. This will provide the information on the distance and the orientation of the car. Once the car learned that the parking space is located in front and to the right hand side of the vehicle, the car will then initiate its motion. The car will drive straight on ahead, until it reaches a certain point where it is near enough to the parking space, it will then start to steer the car to the right. This is as shown in **Figure 4.4**.

The car will continuously check the orientation of itself and also the parking space. As shown in **Figure 4.5**, when the car checked that it is now aligned with the parking space, it will then stop so steer to the right anymore and will just throttle ahead. Finally, in **Figure 4.6** shows that when the car is inside the parking space it will totally halt its motion as it has successfully accomplished its tasks.

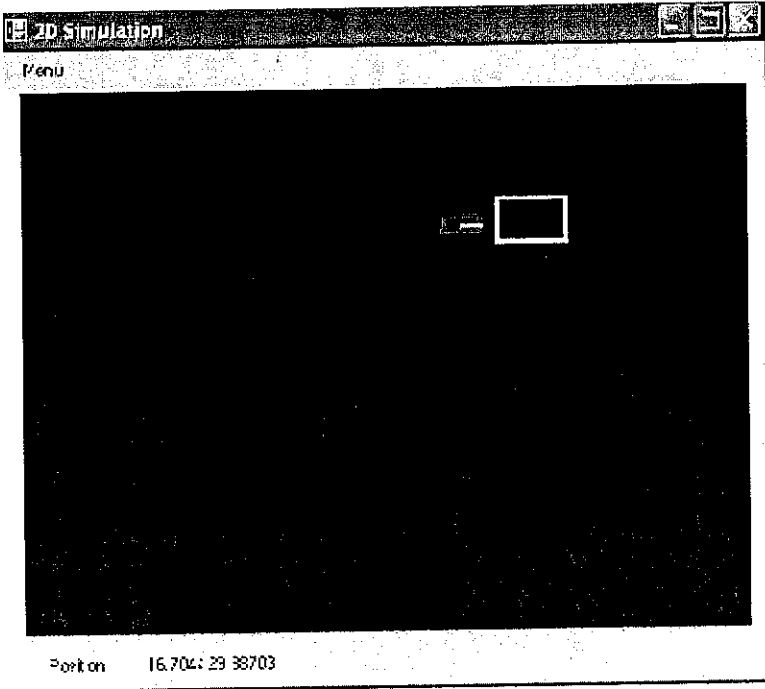


Figure 4.5: The car will steer straight

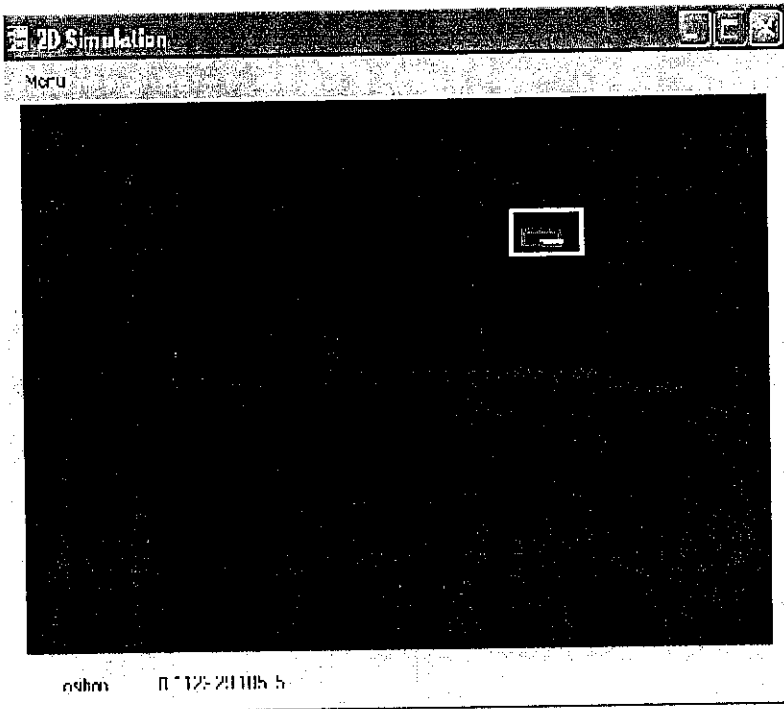


Figure 4.6: The car in the parking space

## 4.2 Discussion

As of now, the author has been able to reach all of the milestones that have been set during the planning stage of this project. The project is at the moment in the Phase 4 which is the enhancing the prototype phase. The author considers that this phase can be a never ending one as improvement will always be required. There are still a lot of things that can be done to improve this system. But due to the limited amount of time that the author has, the project will have to end at some point.

All the other phases of this project can be considered as a success. During the phase 2 and 3 of this project, this project's progress is not according to the timeline as it falls behind schedule for a few weeks. This is during the part where the author has to develop the 2D simulation for the system. This is especially challenging as the author has little to no background at all in the graphics programming. The author then came across a simulation created by Matt Kincaid, in his tutorial for 2D game physics, and then decided to use his simulation and modify it accordingly instead of creating everything from scratch. Matt Kincaid has approved of the use of his simulation. This has greatly helps the progress of this project.

The project is now successfully working in the situation that has been set by the author. The vehicle should now be able to park itself no matter where it is. The user can drive the vehicle to where ever the location is, and when the automate parking is selected, the car will be able to detect the parking space and then will be able to maneuver itself to enter the empty parking space.

The author now plans to further develop the system by adding more functions that will allow the user to add objects as obstacles. These will really test the vehicle's intelligence in planning its motion and executing it.



# **CHAPTER 5**

## **RECOMMENDATION AND CONCLUSION**

### **CHAPTER 5: RECOMMENDATION AND CONCLUSION**

#### **5.1 Recommendation**

Due to the time and resource constraint in the development of this project, the author can only do so much during these two semesters. The author focuses on the main functions that are going to allow the system to work with optimal performance. So, quite a number of assumptions have been made. As for the recommendation, there are quite a lot of things that can be improved to the system that has been created.

##### **5.1.1 Test with real vehicle**

Since in this project the system is only tested using a simulation, it is highly recommended that this system will next be tested in the real world to see how the algorithm will work. One way to do this is by using a modified remote control vehicle. This can be considered as the “small scale” testing. The remote control vehicle will be equipped with the chips and the algorithm will be downloaded into the chip and this will allow the car to process the inputs and plan its motion.

For the “large scale” testing, a real car can be used and equipped with the necessary hardware and system to allow it to plan and execute the motion.

This is really challenging as when testing in real world, there are a lot of other factors that will come into action. The developer will have to consider the vehicle's traction with the ground, the real force that opposes the vehicle's movement, the gravity force that acts on the vehicle. These factors are highly important when trying to test the vehicle in real life.

### **5.1.2 Develop the Vision System**

Since in this project, the author did not create the Vision System, it is a good thing for future improvements. This will allow the car to be tested in real life environment instead of just in simulations. It will enhance the capability of the system greatly. In order to do this, the car will need to be equipped with a number of detecting devices, be it infra-red sensors or even cameras. Then, the vision system will have to be able to filter only the necessary information and will then pass it to the system. The filtering process of the inputs is challenging. The vision system will need to determine what exactly it wants to capture and then it will need to dispose of all the unnecessary information. There will also be a lot of noises in the capture inputs; this has to be taken care of as well.

## 5.2 Conclusion

As for the conclusion, the project can be considered as a successful one. All the milestones set for this project has been reached and most importantly the objectives of this project which is “To develop an autonomous motion planning system that will allow a mobile vehicle to automatically park itself efficiently” has been successfully achieved. The system that has been developed is now able to plan the motion and executing it perfectly.

From this project, the author manages to learn lots of things. Among them is the project management skill. An important thing in a project is the duration of the project. Therefore the planning of the project can be considered as crucial for the success of a project. The planning stage need to really be able to allocate the right amount of time for certain tasks. Key milestones need to be set as this will serve as a guide for the developer regarding the time line of the project. The key milestone is a great way to make sure that the developer is alert on the important dates. The resource management such as cost and money is of less issue in this project as this project doesn't really require any expensive hardware to be bought. The author also learns how to manage stress the right way. During this two semester period, the author feels that the pressure is high during the last few months towards the end of the project.

For the technical part, the author manages to learn on the physics of a moving object. How the Newton's Law is applied to the objects and how it is done in the programming codes. The author also manages to learn a few things on graphics programming from the simulation by Matt Kincaid. The author also notice that the object – oriented programming is really important when developing quite a long program as tough solutions can be made easy with the aid of object – oriented programming.

Lastly, the author has really learned a lot during this project period. The author really hopes that this project will serve its purpose to help the community and the automobile industry to grow further in the technological area.

## REFERENCES

1. Steven M. Lavalle, 2006, "Planning Algorithms", Cambridge University Press
2. David M. Bourg, 2002, "Physics for Game Developers", O'Reilly & Associates, Inc.
3. Wolfgang A. Daxwanger, Gunther K. Schmidt 1995, "*Skill-based Visual Parking Control Using Neural and Fuzzy Networks*", Department for Automatic Control Engineering, Technical University of Munich.
4. Milten Roberto Heinen, Fernando Santos Osorio, Farlei Jose Heinen, Christian Kelber, 2006, "*SEVA3D: Autonomous Vehicles Parking Simulator in a three dimensional environment.*"
5. Jin Xu, Guang Chen, Ming Xie,(n.d.) "Vision-Guided Automatic Parking For Smart Car"
6. Se-Young Oh, *Senior Member, IEEE*, Jeong-Hoon Lee and Doo Hyun Chooi, 2000, "*A New Reinforcement Learning Vehicle Control Architecture For Vision-Based Road Following*"
7. Dario Maravall, Miguel Angel Patricio and Javier de Lope, 2003, "*Automatic Car Parking: A Reinforcement Learning Approach*" Department of A.I, Faculty of Computer Science, Universidad Politecnica de Madrid.

8. A. Atreya, B. Cattle, S. Momen, B. Collins, A. Downey, G. Franken, J. Glass, Z. Glass, J. Herbach, A. Saxe, I. Ashwash, C. Baldassano, W. Hu, U. Javed, J. Mayer, D. Benjamin, L. Gorman, D. Yu, 2006, "*DARPA Urban Challenge Princeton University Technical Paper*", Princeton University
9. Matt Kincaid, "2D Car Physics"
10. Ingrid Russell of the University of Hartford, "Neural Networks Module", 1996<<http://uhavax.hartford.edu/compsci/neural-networks-definition.html>>
11. Pete McCollum, "An Introduction to Back-Propagation Neural Networks",<<http://www.seattlerobotics.org/encoder/nov98/neural.html>>
12. Steven D. Kaehler, "FUZZY LOGIC - AN INTRODUCTION", <[http://www.seattlerobotics.org/encoder/mar98/fuz/fl\\_part1.html#INTRODUCTION](http://www.seattlerobotics.org/encoder/mar98/fuz/fl_part1.html#INTRODUCTION)>
13. F. Markus Jönsson, "An optimal pathfinder for vehicles in real-world digital terrain maps", 1997 <<http://www.student.nada.kth.se/~f93-maj/pathfinder/contents.html>>
14. Christos Stergiou and Dimitrios Siganos, "NEURAL NETWORKS",<[http://www.doc.ic.ac.uk/~nd/surprise\\_96/journal/vol4/cs11/report.html#Introduction%20to%20neural%20networks](http://www.doc.ic.ac.uk/~nd/surprise_96/journal/vol4/cs11/report.html#Introduction%20to%20neural%20networks)>
15. <http://www.definethat.com/hitting.asp?ID=3601>

16. [http://en.wikipedia.org/wiki/Artificial neural network](http://en.wikipedia.org/wiki/Artificial_neural_network)

17. [http://en.wikipedia.org/wiki/Fuzzy logic#Misconceptions and controversie](http://en.wikipedia.org/wiki/Fuzzy_logic#Misconceptions_and_controversies)

s

## **APPENDIX**



Here is the email snippet from Matt Kincaid allowing the author to use his simulation in this project:

hi amirul,

Feel free to use the demo how ever you'd like. just remember to give me credit.

I'm not sure why a negative throttle wasnt working for you. it worked fine for me.

I modified this line:

```
vehicle.SetThrottle(throttle, menu.Checked);
```

to be

```
vehicle.SetThrottle(-throttle, menu.Checked);
```

To make the vehicle stop instantly you'll have to zero its velocity. So in the RigidBody object add a function that can set the value of `m_velocity`. And call this with a vector of zero zero.

Good luck!

On Sun, Mar 23, 2008 at 4:37 AM, amirul ehsan

<[azurieza@yahoo.com](mailto:azurieza@yahoo.com)> wrote:

Hello there,

I'm Amirul. I'm doing a project on vehicle parking simulation when I came across your 2D car physics tutorial. Can I use your simulation from the tutorial for my project..?I'd like to integrate my algorithm into your simulation.

But, there's few things I'd like to know, I've tried to set the throttle value to negative to make the car reverse, but the car motion doesn't seems right..and how do I make the car to instant stop..?

Cheers~

-Amirul-

---

Be a better friend, newshound, and know-it-all with Yahoo!  
Mobile. Try it now.

--  
Matt Kincaid

Programmer

[kincaid05@gmail.com](mailto:kincaid05@gmail.com)

Cell: (260) 241-4002

Here is the source code for the whole system:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Drawing.Drawing2D;
using System.Text;
using System.Windows.Forms;

namespace Simulation_2D
{
    static class Program
    {
        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new frmMain());
        }
    }

    //our main application form
    public partial class frmMain : Form
    {
        //graphics
        Graphics graphics; //gdi+
        Bitmap backbuffer;
        Size buffersize;
        const float screenScale = 3.0f;
        Timer timer = new Timer();
        public Rectangle ParkRect = new Rectangle();
        float VectorX, VectorY;
        bool inParking, Park1, Park2;
        float car_angle;

        Vector ParkPointA = new Vector();
        Vector ParkPointB = new Vector();
        Vector ParkPointC = new Vector();
        Vector ParkPointD = new Vector();
        Vector ParkCenterPoint = new Vector();
    }
}
```

```

Vector CarPointA = new Vector();
Vector CarPointB = new Vector();
Vector CarPointC = new Vector();
Vector CarPointD = new Vector();

//keyboard controls
bool leftHeld = false, rightHeld = false;
bool upHeld = false, downHeld = false;
bool shiftHeld = false;
bool Automate = false;

//vehicle controls
float steering = 0; //-1 is full left, 0 is center, 1 is full
right float throttle = 0; //0 is coasting, 1 is full throttle
float reverse = 0; //0 is stop, -1 is full reverse
bool brakes = false;

//game objects
Vehicle vehicle = new Vehicle();

public frmMain()
{
    InitializeComponent();
    Application.Idle += new EventHandler(ApplicationIdle);

    screen.Paint += new PaintEventHandler(screen_Paint);
    this.KeyDown += new KeyEventHandler(onKeyDown);
    this.KeyUp += new KeyEventHandler(onKeyUp);

    Init(screen.Size);
}

//intialize rendering
private void Init(Size size)
{
    //setup rendering device
    buffersize = size;
    backbuffer = new Bitmap(buffersize.Width,
buffersize.Height);
    graphics = Graphics.FromImage(backbuffer);

    timer.GetETime(); //reset timer

    vehicle.Setup(new Vector(7, 12) / 2.0f, 5, Color.Red);
    vehicle.SetLocation(new Vector(0, 0), 0);
}

//main rendering function
private void Render(Graphics g)

```

```

    {
        //clear back buffer
        graphics.Clear(Color.Gray);
        graphics.ResetTransform();
        graphics.ScaleTransform(screenScale, -screenScale);
        graphics.TranslateTransform(bufferSize.Width / 2.0f /
screenScale, -bufferSize.Height / 2.0f / screenScale);

        //draw to back buffer
        DrawScreen();

        CalculateDistance();
        car_angle = CalculateAngle();

        //display coordinate of the vehicle
        label1.Text = vehicle.m_position.X.ToString();
        label2.Text = vehicle.m_position.Y.ToString();

        //Display the distance between car and parking space
        label5.Text = VectorX.ToString();
        label6.Text = VectorY.ToString();

        //Display vehicle's velocity and angle for testing purposes
        label3.Text = vehicle.m_velocity.X.ToString();
        label4.Text = vehicle.m_velocity.Y.ToString();
        //label7.Text = car_angle.ToString();

        //label8.Text = vehicle.rect.X.ToString();
        //label9.Text = vehicle.rect.Y.ToString();

        //label10.Text = timer.etime.ToString();
        //label11.Text = timer.lastTime.ToString();

        //present back buffer
        g.DrawImage(backbuffer, new Rectangle(0, 0,
bufferSize.Width, bufferSize.Height), 0, 0, bufferSize.Width,
bufferSize.Height, GraphicsUnit.Pixel);
    }

    //draw the screen
    private void DrawScreen()
    {
        vehicle.Draw(graphics, bufferSize);
        DrawParkSpace(graphics, bufferSize);
    }

    //process game logic
    private void DoFrame()
    {

```

```

//get elapsed time since last frame
float etime = timer.GetETime();

//process input
InitPoint();

//MainSelectRoute();
if (Automate)
    MainAutoPark();
else
    ProcessInput();

//apply vehicle controls
vehicle.SetSteering(steering);
vehicle.SetThrottle(throttle, menu.Checked);
vehicle.SetThrottle(reverse, menu.Checked);
vehicle.SetBrakes(brakes);

//integrate vehicle physics
vehicle.Update(etime);

//keep the vehicle on the screen
ConstrainVehicle();

//redraw our screen
screen.Invalidate();
}

//keep the vehicle on the screen
private void ConstrainVehicle()
{
    Vector position = vehicle.GetPosition();
    Vector screenSize = new Vector(screen.Width / screenScale,
screen.Height / screenScale);

    while (position.X > screenSize.X / 2.0f) { position.X -=
screenSize.X; }
    while (position.Y > screenSize.Y / 2.0f) { position.Y -=
screenSize.Y; }
    while (position.X < -screenSize.X / 2.0f) { position.X +=
screenSize.X; }
    while (position.Y < -screenSize.Y / 2.0f) { position.Y +=
screenSize.Y; }
}

//process keyboard input
private void ProcessInput()
{
    if (leftHeld)
        steering = -1;
}

```

```

else if (rightHeld)
    steering = 1;
else
    steering = 0;

if (upHeld)
    throttle = 1;
else
    throttle = 0;

if (downHeld)
    reverse = -1;
else
    reverse = 0;

if (shiftHeld)
    brakes = true;
else
    brakes = false;
}

private void onKeyDown(object sender, KeyEventArgs e)
{
    switch (e.KeyCode)
    {
        case Keys.Left:
            leftHeld = true;
            break;
        case Keys.Right:
            rightHeld = true;
            break;
        case Keys.Up:
            upHeld = true;
            break;
        case Keys.Down:
            downHeld = true;
            break;
        case Keys.ShiftKey:
            shiftHeld = true;
            break;
        default: //no match found
            return; //return so handled doesnt get set
    }

    //match found
    e.Handled = true;
}

private void onKeyUp(object sender, KeyEventArgs e)
{

```

```

switch (e.KeyCode)
{
    case Keys.Left:
        leftHeld = false;
        break;
    case Keys.Right:
        rightHeld = false;
        break;
    case Keys.Up:
        upHeld = false;
        break;
    case Keys.Down:
        downHeld = false;
        break;
    case Keys.ShiftKey:
        shiftHeld = false;
        break;
    default: //no match found
        return; //return so handled dosnt get set
}

//match found
e.Handled = true;
}

//rendering - only when screen is invalidated
private void screen_Paint(object sender, PaintEventArgs e)
{
    Render(e.Graphics);
}

//when the os gives us time, run the game
private void ApplicationIdle(object sender, EventArgs e)
{
    // While the application is still idle, run frame routine.
    DoFrame();
}

private void MenuExit_Click(object sender, EventArgs e)
{
    this.Close();
}

////////////////////////////////////
////////////////////////////////////
//draw the parking space
public void DrawParkSpace(Graphics g, Size buffersize)
{
    //ParkRect.X = -4;      //for reverse park

```



```

//ParkRect.Y = -28;
//ParkRect.X = -4;      //for straight park
//ParkRect.Y = 20;
//park = 1;
//ParkRect.Width = 9;
//ParkRect.Height = 15;

//ParkRect.X = -35;
//ParkRect.Y = 25;
ParkRect.X = 30;
ParkRect.Y = 40;
ParkRect.Width = 13;
ParkRect.Height = 20;

g.DrawRectangle(new Pen(Color.White), ParkRect);
}

public void InitPoint()
{
    ParkPointA.X = ParkRect.X;
    ParkPointA.Y = ParkRect.Y;
    ParkPointB.X = ParkPointA.X + ParkRect.Width;
    ParkPointB.Y = ParkPointA.Y;
    ParkPointC.X = ParkPointA.X;
    ParkPointC.Y = ParkPointA.Y + ParkRect.Height;
    ParkPointD.X = ParkPointB.X;
    ParkPointD.Y = ParkPointC.Y;
    ParkCenterPoint.X = ParkPointA.X + ParkRect.Width / 2;
    ParkCenterPoint.Y = ParkPointA.Y + ParkRect.Height / 2;
}

public void MainAutoPark()
{
    //check if in parking space
    float x = vehicle.m_position.X;
    float y = vehicle.m_position.Y;
    if (y > ParkPointA.Y + 8 && y < ParkPointC.Y - 8 && x >
ParkPointA.X + 1.5 && x < ParkPointB.X - 1.5)
        inParking = true;

    //if ((car_angle >= -0.05 && car_angle <= 0.05) ||
(car_angle >= -3.15 && car_angle <= -3.05) ||
//      (car_angle >= 3.05 && car_angle <= 3.15))
    if (inParking)
        brakes = true;
    else
        FuzzyControl();
}

public void CalculateDistance()

```

```

{
    VectorX = vehicle.m_position.X - ParkCenterPoint.X;
    VectorY = vehicle.m_position.Y - ParkCenterPoint.Y;
}

public void FuzzyControl()
{
    // for throttle
    float full_forward= 1.0f;
    float semi_forward = 0.1f;
    float no_throttle = 0.0f;
    float full_backward = -1.0f;
    float semi_backward = -0.05f;

    //for steering
    float left = -1.0f;
    float semi_left = -0.5f;
    float right = 1.0f;
    float semi_right = 0.5f;
    float straight = 0.0f;

    if ((car_angle >= -0.05 && car_angle <= 0.05) &&
(VectorX >= -3.0 && VectorX <= 3.0))
    {
        if (vehicle.m_velocity.X >= 8 ||
vehicle.m_velocity.Y >= 8 || vehicle.m_angularVelocity >= 8)
            throttle = no_throttle;
        else
            throttle = full_forward;
    }
    else if (VectorX <= -20)
    {
        if (vehicle.m_velocity.X >= 8 ||
vehicle.m_velocity.Y >= 8 || vehicle.m_angularVelocity >= 8)
            throttle = no_throttle;
        else
            throttle = semi_forward;

        steering = right;
        if (car_angle <= -1.3)
            steering = straight;
    }
    else if (VectorX >= 20)
    {
        if (vehicle.m_velocity.X >= 8 ||
vehicle.m_velocity.Y >= 8 || vehicle.m_angularVelocity >= 8)
            throttle = no_throttle; steering = left;

        if (car_angle >= 0.05)
            steering = right;
    }
}

```

```

        else if (car_angle <= -0.05)
            steering = left;
        else
            steering = straight;
    }

    //else
    //    brakes = true;
}

public float CalculateAngle()
{
    float newAngle = vehicle.m_angle;;

    while (newAngle >= 6)
        newAngle = newAngle - 6;

    while (newAngle <= -6)
        newAngle = newAngle + 6;

    return newAngle;
}

private void testToolStripMenuItem_Click(object sender,
EventArgs e)
{
    Automate = true;
}

private void manualDriveToolStripMenuItem_Click(object sender,
EventArgs e)
{
    Automate = false;
}

////////////////////////////////////
////////////////////////////////////
}

//mini 2d vector :)
class Vector
{
    public float X, Y;

    public Vector() { X = 0; Y = 0; }
    public Vector(float x, float y) { X = x; Y = y; }

    //length property

```

```

public float Length
{
    get
    {
        return (float)Math.Sqrt((double)(X * X + Y * Y));
    }
}

//addition
public static Vector operator +(Vector L, Vector R)
{
    return new Vector(L.X + R.X, L.Y + R.Y);
}

//subtraction
public static Vector operator -(Vector L, Vector R)
{
    return new Vector(L.X - R.X, L.Y - R.Y);
}

//negative
public static Vector operator -(Vector R)
{
    Vector temp = new Vector(-R.X, -R.Y);
    return temp;
}

//scalar multiply
public static Vector operator *(Vector L, float R)
{
    return new Vector(L.X * R, L.Y * R);
}

//divide multiply
public static Vector operator /(Vector L, float R)
{
    return new Vector(L.X / R, L.Y / R);
}

//dot product
public static float operator *(Vector L, Vector R)
{
    return (L.X * R.X + L.Y * R.Y);
}

//cross product, in 2d this is a scalar since we know it points
in the Z direction
public static float operator %(Vector L, Vector R)
{
    return (L.X * R.Y - L.Y * R.X);
}

```

```

    }

    //normalize the vector
    public void normalize()
    {
        float mag = Length;

        X /= mag;
        Y /= mag;
    }

    //project this vector on to v
    public Vector Project(Vector v)
    {
        //projected vector = (this dot v) * v;
        float thisDotV = this * v;
        return v * thisDotV;
    }

    //project this vector on to v, return signed magnitude
    public Vector Project(Vector v, out float mag)
    {
        //projected vector = (this dot v) * v;
        float thisDotV = this * v;
        mag = thisDotV;
        return v * thisDotV;
    }
}

//keep track of time between frames
class Timer
{
    //store last time sample
    public int lastTime = Environment.TickCount;
    public float etime;

    //calculate and return elapsed time since last call
    public float GetETime()
    {
        etime = (Environment.TickCount - lastTime) / 1000.0f;
        lastTime = Environment.TickCount;

        return etime;
    }
}
}

```

```

//our simulation object
class RigidBody
{
    //linear properties
    public Vector m_position = new Vector();
    public Vector m_velocity = new Vector();
    public Vector m_forces = new Vector();
    public float m_mass;

    //angular properties
    public float m_angle;
    public float m_angularVelocity;
    public float m_torque;
    public float m_inertia;

    //graphical properties
    public Vector m_halfSize = new Vector();
    public Rectangle rect = new Rectangle();
    public Color m_color;

    public RigidBody()
    {
        //set these defaults so we dont get divide by zeros
        m_mass = 1.0f;
        m_inertia = 1.0f;
    }

    //intialize out parameters
    public void Setup(Vector halfSize, float mass, Color color)
    {
        //store physical parameters
        m_halfSize = halfSize;
        m_mass = mass;
        m_color = color;
        m_inertia = (1.0f / 12.0f) * (halfSize.X * halfSize.X) *
(halfSize.Y * halfSize.Y) * mass;

        //generate our viewable rectangle
        rect.X = (int)-m_halfSize.X;
        rect.Y = (int)-m_halfSize.Y;
        rect.Width = (int)(m_halfSize.X * 2.0f);
        rect.Height = (int)(m_halfSize.Y * 2.0f);
        //rect.X = 0;
        //rect.Y = 0;
        //rect.Height = 11;
        //rect.Width = 6;
    }

    public void SetLocation(Vector position, float angle)
    {

```

```

        m_position = position;
        m_angle = angle;
    }

    public Vector GetPosition()
    {
        return m_position;
    }

    public void Update(float timeStep)
    {
        //integrate physics
        //linear
        Vector acceleration = m_forces / m_mass;
        m_velocity += acceleration * timeStep;
        m_position += m_velocity * timeStep;
        m_forces = new Vector(0, 0); //clear forces

        //angular
        float angAcc = m_torque / m_inertia;
        m_angularVelocity += angAcc * timeStep;
        m_angle += m_angularVelocity * timeStep;
        m_torque = 0; //clear torque
    }

    public void Draw(Graphics graphics, Size buffersize)
    {
        //store transform, (like opengl's glPushMatrix())
        Matrix mat1 = graphics.Transform;

        //transform into position
        graphics.TranslateTransform(m_position.X, m_position.Y);
        graphics.RotateTransform(m_angle / (float)Math.PI *
180.0f);

        try
        {
            //draw body
            graphics.DrawRectangle(new Pen(m_color), rect);

            //draw line in the "forward direction"
            //graphics.DrawLine(new Pen(Color.Yellow), 1, 0, 1, 5);

            //draw line in the "forward direction"
            graphics.DrawLine(new Pen(Color.Yellow), rect.X +
rect.Width / 2, rect.Y + rect.Height / 2, rect.X + rect.Width / 2,
rect.Y + rect.Height);
        }
        catch (OverflowException exc)
        {

```

```

        //physics overflow :(
    }

    //restore transform
    graphics.Transform = mat1;
}

//take a relative vector and make it a world vector
public Vector RelativeToWorld(Vector relative)
{
    Matrix mat = new Matrix();
    PointF[] vectors = new PointF[1];

    vectors[0].X = relative.X;
    vectors[0].Y = relative.Y;

    mat.Rotate(m_angle / (float)Math.PI * 180.0f);
    mat.TransformVectors(vectors);

    return new Vector(vectors[0].X, vectors[0].Y);
}

//take a world vector and make it a relative vector
public Vector WorldToRelative(Vector world)
{
    Matrix mat = new Matrix();
    PointF[] vectors = new PointF[1];

    vectors[0].X = world.X;
    vectors[0].Y = world.Y;

    mat.Rotate(-m_angle / (float)Math.PI * 180.0f);
    mat.TransformVectors(vectors);

    return new Vector(vectors[0].X, vectors[0].Y);
}

//velocity of a point on body
public Vector PointVel(Vector worldOffset)
{
    Vector tangent = new Vector(-worldOffset.Y, worldOffset.X);
    return tangent * m_angularVelocity + m_velocity;
}

public void AddForce(Vector worldForce, Vector worldOffset)
{
    //add linear force
    m_forces += worldForce;
    //and it's associated torque
    m_torque += worldOffset % worldForce;
}

```



```

    }
}

//our vehicle object
class Vehicle : Rigidbody
{
    public class Wheel
    {
        public Vector m_forwardAxis, m_sideAxis;
        public float m_wheelTorque, m_wheelSpeed, m_wheelInertia,
m_wheelRadius;
        public Vector m_Position = new Vector();

        public Wheel(Vector position, float radius)
        {
            m_Position = position;
            SetSteeringAngle(0);
            m_wheelSpeed = 0;
            m_wheelRadius = radius;
            m_wheelInertia = radius * radius; //fake value
        }

        public void SetSteeringAngle(float newAngle)
        {
            Matrix mat = new Matrix();
            PointF[] vectors = new PointF[2];

            //foward vector
            vectors[0].X = 0;
            vectors[0].Y = 1;
            //side vector
            vectors[1].X = -1;
            vectors[1].Y = 0;

            mat.Rotate(newAngle / (float)Math.PI * 180.0f);
            mat.TransformVectors(vectors);

            m_forwardAxis = new Vector(vectors[0].X, vectors[0].Y);
            m_sideAxis = new Vector(vectors[1].X, vectors[1].Y);
        }

        public void AddTransmissionTorque(float newValue)
        {
            m_wheelTorque += newValue;
        }

        public float GetWheelSpeed()
        {
            return m_wheelSpeed;
        }
    }
}

```

```

    }

    public Vector GetAttachPoint()
    {
        return m_Position;
    }

    public Vector CalculateForce(Vector relativeGroundSpeed,
float timeStep)
    {
        //calculate speed of tire patch at ground
        Vector patchSpeed = -m_forwardAxis * m_wheelSpeed *
m_wheelRadius;

        //get velocity difference between ground and patch
        Vector velDifference = relativeGroundSpeed +
patchSpeed;

        //project ground speed onto side axis
        float forwardMag = 0;
        Vector sideVel = velDifference.Project(m_sideAxis);
        Vector forwardVel =
velDifference.Project(m_forwardAxis, out forwardMag);

        //calculate super fake friction forces
        //calculate response force
        Vector responseForce = -sideVel * 2.0f;
        responseForce -= forwardVel;

        //calculate torque on wheel
        m_wheelTorque += forwardMag * m_wheelRadius;

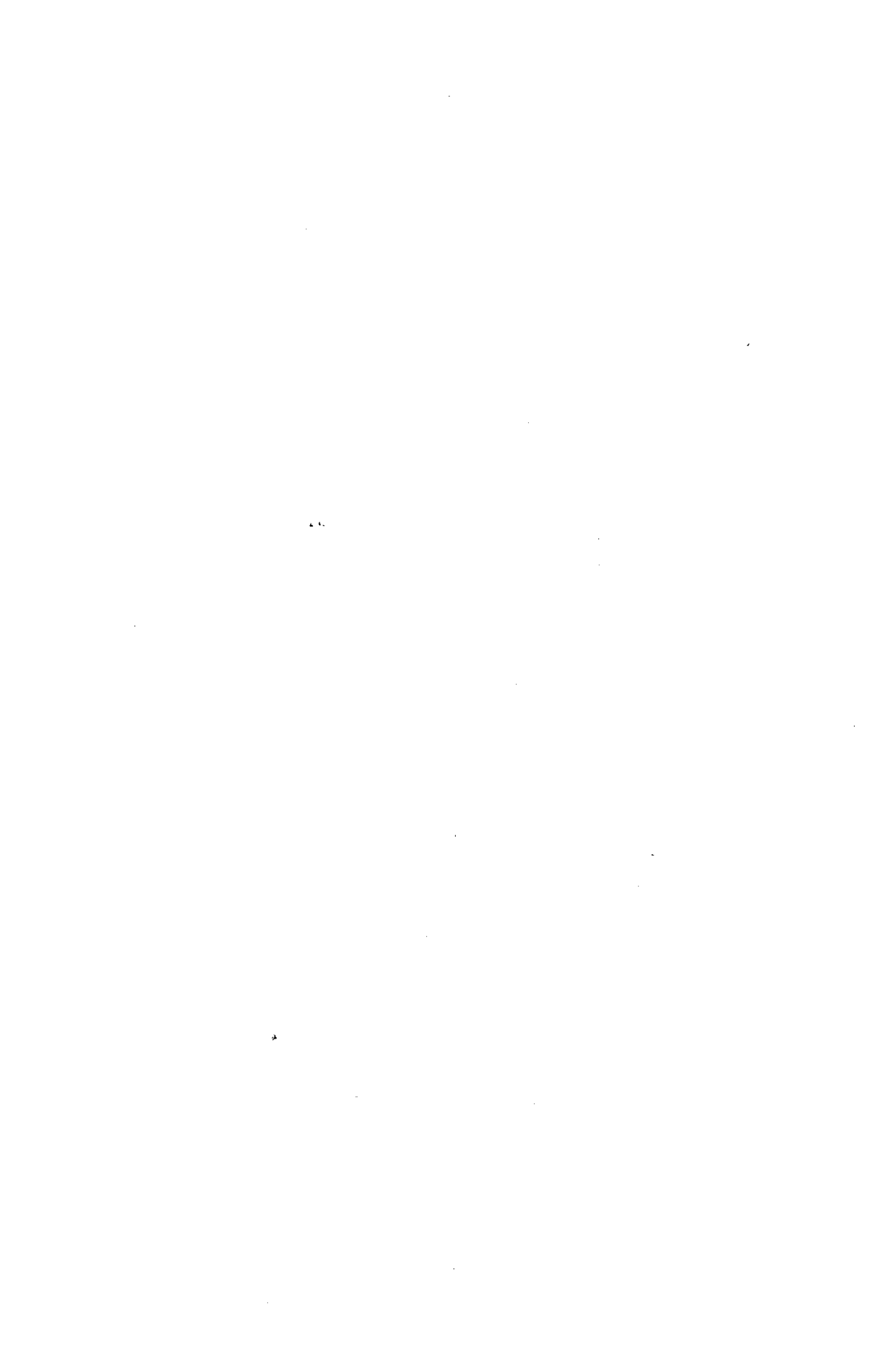
        //integrate total torque into wheel
        m_wheelSpeed += m_wheelTorque / m_wheelInertia *
timeStep;

        //clear our transmission torque accumulator
        m_wheelTorque = 0;

        //return force acting on body
        return responseForce;
    }
}
private Wheel[] wheels = new Wheel[4];

new public void Setup(Vector halfSize, float mass, Color color)
{
    //front wheels
    wheels[0] = new Wheel(new Vector(halfSize.X, halfSize.Y),
0.5f);

```



```

0.5f);
    wheels[1] = new Wheel(new Vector(-halfSize.X, halfSize.Y),
//rear wheels
0.5f);
    wheels[2] = new Wheel(new Vector(halfSize.X, -halfSize.Y),
0.5f);
    wheels[3] = new Wheel(new Vector(-halfSize.X, -halfSize.Y),

    base.Setup(halfSize, mass, color);
}

public void SetSteering(float steering)
{
    const float steeringLock = 0.75f;

    //apply steering angle to front wheels
    wheels[0].SetSteeringAngle(-steering * steeringLock);
    wheels[1].SetSteeringAngle(-steering * steeringLock);
}

public void SetThrottle(float throttle, bool allWheel)
{
    const float torque = 20.0f;

    //apply transmission torque to back wheels
    if (allWheel)
    {
        wheels[0].AddTransmissionTorque(throttle * torque);
        wheels[1].AddTransmissionTorque(throttle * torque);
    }

    wheels[2].AddTransmissionTorque(throttle * torque);
    wheels[3].AddTransmissionTorque(throttle * torque);
}

//
public void SetBrakes(bool brakes)
{
    if (brakes)
    {
        m_velocity.Y = 0.0f;
        m_velocity.X = 0.0f;
        m_angularVelocity = 0.0f;
        foreach (Wheel wheel in wheels)
            wheel.m_wheelSpeed = 0;
    }
}
}

```

```

new public void Update(float timeStep)
{
    foreach (Wheel wheel in wheels)
    {
        //wheel.m_wheelSpeed = 30.0f;
        Vector worldWheelOffset =
base.RelativeToWorld(wheel.GetAttachPoint());
        Vector worldGroundVel =
base.PointVel(worldWheelOffset);
        Vector relativeGroundSpeed =
base.WorldToRelative(worldGroundVel);
        Vector relativeResponseForce =
wheel.CalculateForce(relativeGroundSpeed, timeStep);
        Vector worldResponseForce =
base.RelativeToWorld(relativeResponseForce);

        base.AddForce(worldResponseForce, worldWheelOffset);
    }

    base.Update(timeStep);
}
}

```