

Moving Through Waypoints

Computer Graphics and Animation Programming Assignment 18



Ida Bagus Bhaskara (001201500076)

Rahmad Martin (001201500033)

Vera Debora Vitamas (001201500076)

CIT 2 2015

**Table of Contents**

Cover Page 1

**Table of Contents 2**

[Chapter 1 Introduction 2](#_Toc500108520)

[Chapter 2 Basic Theory 2](#_Toc500108521)

[2.1 Speed 2](#_Toc500108522)

[2.2 Velocity 3](#_Toc500108523)

[2.3 Acceleration 4](#_Toc500108524)

[2.4 Projectile Motion 5](#_Toc500108525)

[2.5 Torque 7](#_Toc500108526)

[Chapter 3 Implementation 10](#_Toc500108527)

[3.1 Main Interface of the Application 10](#_Toc500108528)

[3.2 Features of the Applicaton 11](#_Toc500108529)

[Chapter 4 Design 11](#_Toc500108530)

[4.1 Variables used in the Program 11](#_Toc500108531)

[4.2 Representation of the Car in the Application 12](#_Toc500108532)

[4.3 Representation of the Waypoints in the Application 12](#_Toc500108533)

[4.4 Next Waypoint Direction and Waypoint Collision Detection Algorithm 13](#_Toc500108534)

[4.5 Car Acceleration and Deceleration Algorithm 16](#_Toc500108535)

[Chapter 5 Evaluation 17](#_Toc500108536)

[5.1 Adding Waypoints 17](#_Toc500108537)

[5.2 Moving the Car to Waypoints and Stopping After It Has Reached the Final Waypoint 18](#_Toc500108538)

[5.3 Stopping the Car Abruptly and Making It Move Again 18](#_Toc500108539)

[5.4 Increasing the Car’s Acceleration 19](#_Toc500108540)

[Chapter 6 Work Log 20](#_Toc500108541)

[Chapter 7 Conclusion and Remarks 21](#_Toc500108542)

# Introduction

Emulating a car’s movement to make it seem that it is able to produce natural movements when it passes through several waypoints can be done. Using methods and algorithms covered in physics and calculus, the movements can be simulated digitally. This will be covered further in the Basic Theory section.

This program was created using Visual Basic programming language. In this report, we cover: basic theory, implementation, design, evaluations using test cases, the work log, and concluding remarks.

# Basic Theory

## Speed

Just as distance and displacement have distinctly different meanings (despite their similarities), so do speed and velocity. Speed is a [scalar quantity](http://www.physicsclassroom.com/Class/1DKin/U1L1b.cfm) that refers to "how fast an object is moving." Speed can be thought of as the rate at which an object covers distance. A fast-moving object has a high speed and covers a relatively large distance in a short amount of time. Contrast this to a slow-moving object that has a low speed; it covers a relatively small amount of distance in the same amount of time. An object with no movement at all has a zero speed.

The average speed during the course of a motion is often computed using the following formula:

*Average Speed = =*

The circumference of any circle can be computed using from the radius according to the equation:

Combining these two equations above will lead to a new equation relating the speed of an object moving in uniform circular motion to the radius of the circle and the time to make one cycle around the circle (**period**).

Where **r** represents the radius of the circle and **T** represents the period. This equation, like all equations, can be used as an algebraic recipe for problem solving. It also can be used to guide our thinking about the variables in the equation relate to each other. For instance, the equation suggests that for objects moving around circles of different radius in the same period, the object traversing the circle of larger radius must be traveling with the greatest speed.

# Velocity

Velocity is a [vector quantity](http://www.physicsclassroom.com/Class/1DKin/U1L1b.cfm) that refers to "the rate at which an object changes its position." Imagine a person moving rapidly - one step forward and one step back - always returning to the original starting position. While this might result in a frenzy of activity, it would result in a zero velocity. Because the person always returns to the original position, the motion would never result in a change in position. Since velocity is defined as the rate at which the position changes, this motion results in zero velocity. If a person in motion wishes to maximize their velocity, then that person must make every effort to maximize the amount that they are displaced from their original position. Every step must go into moving that person further from where he or she started. For certain, the person should never change directions and begin to return to the starting position.

Velocity is a vector quantity. As such, velocity is direction aware. When evaluating the velocity of an object, one must keep track of direction. It would not be enough to say that an object has a velocity of 55 mi/hr. One must include direction information in order to fully describe the velocity of the object. For instance, you must describe an object's velocity as being 55 mi/hr, **east**. This is one of the essential differences between speed and velocity. Speed is a scalar quantity and does not keep track of direction; velocity is a vector quantity and is direction aware.

The task of describing the direction of the velocity vector is easy. The direction of the velocity vector is simply the same as the direction that an object is moving. It would not matter whether the object is speeding up or slowing down. If an object is moving rightwards, then its velocity is described as being rightwards. If an object is moving downwards, then its velocity is described as being downwards. So, an airplane moving towards the west with a speed of 300 mi/hr has a velocity of 300 mi/hr, west. Note that speed has no direction (it is a scalar) and the velocity at any instant is simply the speed value with a direction.

The average velocity during the course of a motion is often computed using the following formula:

the instantaneous velocity computed using the following formula:

# 2.3 [Acceleration](https://www.google.com/search?client=firefox-b-ab&q=acceleration&spell=1&sa=X&ved=0ahUKEwjU6eLeidXXAhWFOI8KHZuZAUsQvwUIJCgA&biw=765&bih=751)

Acceleration is a [vector quantity](http://www.physicsclassroom.com/Class/1DKin/U1L1b.cfm) that is defined as the rate at which an object changes its [velocity](http://www.physicsclassroom.com/Class/1DKin/U1L1d.cfm). An object is accelerating if it is changing its velocity. Sometimes an accelerating object will change its velocity by the same amount each second. As mentioned in the previous paragraph, the data table above show an object changing its velocity by 10 m/s in each consecutive second. This is referred to as a constant acceleration since the velocity is changing by a constant amount each second. This is referred to as a constant acceleration since the velocity is changing by a constant amount each second. An object with a constant acceleration should not be confused with an object with a constant velocity. If an object is changing its velocity -whether by a constant amount or a varying amount - then it is an accelerating object. And an object with a constant velocity is not accelerating. The data tables below depict motions of objects with a constant acceleration and a changing acceleration. Note that each object has a changing velocity.

The average acceleration (a) of any object over a given interval of time (t) can be calculated using the equation:

Since acceleration is a [vector quantity](http://www.physicsclassroom.com/Class/1DKin/U1L1b.cfm), it has a direction associated with it. The direction of the acceleration vector depends on two things:

* whether the object is speeding up or slowing down
* whether the object is moving in the *+* or- direction

The general principle for determining the acceleration is, if an object is slowing down, then its acceleration is in the opposite direction of its motion. This general principle can be applied to determine whether the sign of the acceleration of an object is positive or negative, right or left, up or down, etc.

# 2.4 Projectile Motion

Projectile motion is a form of motion where an object moves in a bilaterally symmetrical, parabolic path. The path that the object follows is called its trajectory. Projectile motion only occurs when there is one force applied at the beginning on the trajectory, after which the only interference is from gravity. In a previous atom we discussed what the various components of an object in projectile motion are. In this atom we will discuss the basic equations that go along with them in the special case in which the projectile initial positions are null

The initial velocity can be expressed as x components and y components:

In this equation, stands for initial velocity magnitude and θ refers to projectile angle. In projectile motion, there is no acceleration in the horizontal direction. The acceleration, a, in the vertical direction is just due to gravity, also known as free fall:

The horizontal velocity remains constant, but the vertical velocity varies linearly, because the acceleration is constant. At any time, t, the velocity is:

Pythagorean Theorem to find velocity:

At time, t, the displacement components are:

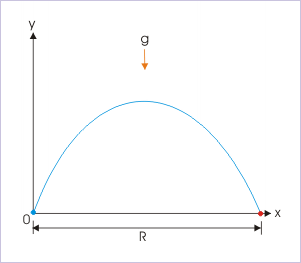
The equation for the magnitude of the displacement is

Use the displacement equations in the x and y direction to obtain an equation for the parabolic form of a projectile motion:

The maximum height is reached when . Using this can rearrange the velocity equation to find the time it will take for the object to reach maximum height

Where stands for the time it takes to reach maximum height. From the displacement equation we can find the maximum height

The range of the motion is fixed by the condition . Using this can rearrange the parabolic motion equation to find the range of the motion:

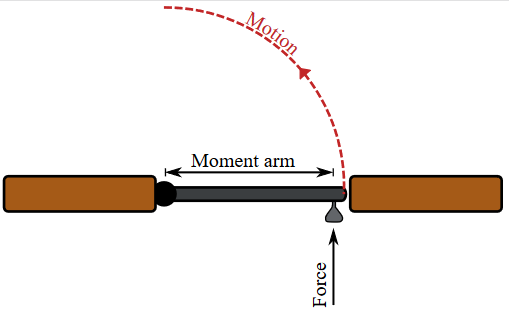


*The range of a trajectory is shown in this figure.*

# 2.5 Torque

Torque is a measure of the force that can cause an object to rotate about an axis. Just as force is what causes an object to accelerate in linear kinematics, torque is what causes an object to acquire angular acceleration. Torque is a vector quantity. The direction of the torque vector depends on the direction of the force on the axis.

Anyone who has ever opened a door has an intuitive understanding of torque. When a person opens a door, they push on the side of the door **farthest** from the hinges. Pushing on the side **closest** to the hinges requires considerably more force. Although the [work done](https://www.khanacademy.org/science/physics/torque-angular-momentum/torque-tutorial/a/science/physics/work-and-energy/work-and-energy-tutorial/a/what-is-work) is the same in both cases (the larger force would be applied over a smaller distance) people generally prefer to apply less force, hence the usual location of the door handle.



*Opening a door with maximum torque*

Torque can be either *static* or *dynamic*. A *static torque* is one which does not produce an angular acceleration. Someone pushing on a closed door is applying a static torque to the door because the door is not rotating about its hinges, despite the force applied. Someone pedaling a bicycle at constant speed is also applying a static torque because they are not accelerating. The drive shaft in a racing car accelerating from the start line is carrying a *dynamic torque* because it must be producing an angular acceleration of the wheels given that the car is accelerating along the track. The terminology used when describing torque can be confusing. Engineers sometimes use the term *moment*, or *moment of force* interchangeably with torque. The radius at which the force acts is sometimes called the *moment arm*.

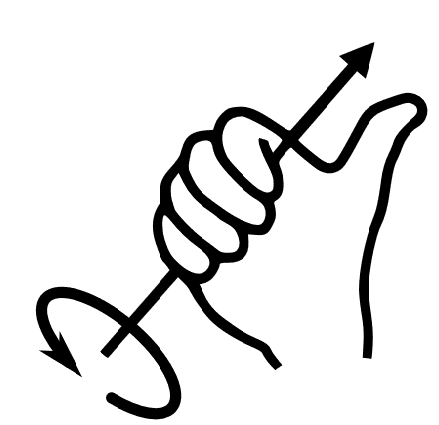
The magnitude of the torque vector τ for a torque produced by a given force F is

τ=F⋅rsin(θ)

Where *r* is the length of the moment arm and θ is the angle between the force vector and the moment arm. In the case of the door shown in above, the force is at right angles (90∘) to the moment arm, so the sine term becomes 1 and

τ=F⋅r

The direction of the torque vector is found by convention using the right-hand grip rule. If a hand is curled around the axis of rotation with the fingers pointing in the direction of the force, then the torque vector points in the direction of the thumb.



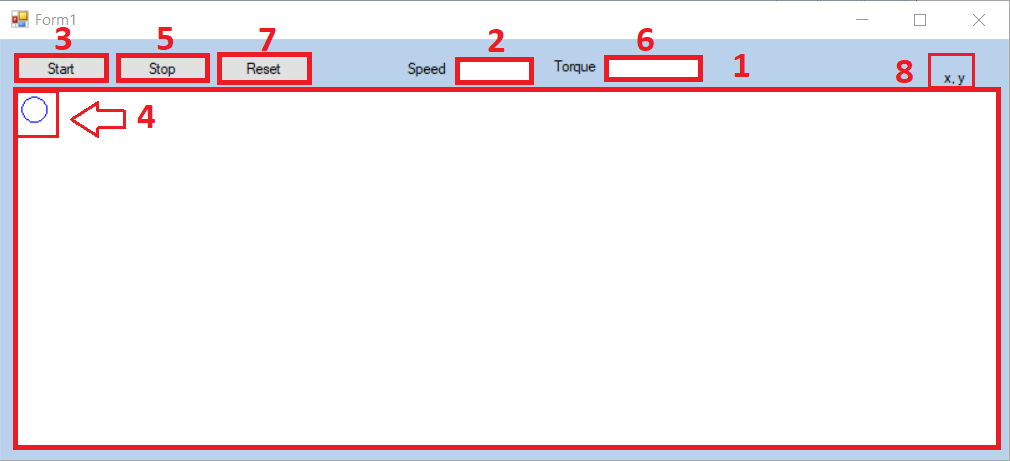
The [SI unit](https://en.wikipedia.org/wiki/International_System_of_Units) for torque is the Newton-meter. In imperial units, the Foot-pound is often used. This is confusing because colloquially the pound is sometimes used as a unit of mass and sometimes force. What is meant here is *pound-force*, the force due to earth gravity on a one-pound object. The magnitude of these units is often similar as 1 Nm≃1.74 ft⋅lbs.

Measuring a static torque in a non-rotating system is usually quite easy, and done by measuring a force. Given the length of the moment arm, the torque can be found directly. Measuring torque in a rotating system is considerably more difficult. One method works by measuring strain within the metal of a drive shaft which is transmitting torque and sending this information wirelessly.

# Implementation

## Main Interface of the Application

This application has one interface, which it uses as its main interface. The interface is pictured below:



The interface consists of the following components:

1. Canvas Picture Box

The canvas is used to display every object generated by the user using the application.

1. Speed Text Box

The speed textbox is used to enter speed values of the waypoint.

1. Start Button

This button is used to move the car towards the waypoint.

1. Car

This is the representation of the car in the application.

1. Stop Button

This button is used to stop the car while moving.

1. Torque Text Box

This Torque Text Box is used to input torque values of the waypoint.

1. Reset Button

This button is used to reset the application back to the original state.

1. Coordinate Label

This Coordinate label is used to show the current x and y coordinate of the mouse.

## Features of the Applicaton

The application has several features, namely:

1. Initializing the Position of the Car

The application can initialize the position of the car on load.

1. Setting Waypoint(s)

the application can set the waypoint(s) of the car to go through.

1. Setting the Speed and Torque of the Car

The application can input the values of speed and torque for the car to move.

1. Making the Car Move

The application can make the car move using start button.

1. Stopping the Car

The application can make the car stop while moving using the stop button.

1. Making the Car Accelerate

The car can accelerate from the moment it moves until it reaches maximum speed.

1. Make the Car Decelerate

The car can decelerate when it decides to stop or when the user demands it to stop.

1. Reset the Application

The application will revert back to its initial state whenever the user presses the reset button.

# Design

## Variables used in the Program

The following variables are used in this program:

* Dim X As Integer

This variable is used to temporarily store the X value for the rectangle.

* Dim Y As Integer

This variable is used to temporarily store the Y value for the rectangle.

* Dim canvas As Graphics

This variable is used to pass graphics to PictureBox1, which acts as the canvas.

* Dim waypoints = New List(Of Rectangle)

This variable is used to store waypoints in a list of Rectangle.

* Dim car = New Rectangle(5, 5, 20, 20)

This variable is used to define the car as a new Rectangle.

* Dim carPositions = New List(Of Point)

This variable is used store the positions of the car after it’s been calculated on the carMovement Sub in a list of Point.

* Dim waypointCounter As Integer = 0

This variable is used as a counter to count the waypoints. It is used mainly by carMovement Sub.

* Dim carPosCounter As Integer = 0

This variable is used as a counter to count the positions of the car. It is used mainly by moveCar\_Tick Sub.

* Dim speed As Integer

This variable is used to store the car’s speed. It is used mainly as a placeholder for the moveCar.Interval by moveCar\_Tick Sub.

* Dim decelerate As Boolean = False

This variable is used to indicate whether the car is supposed to decelerate or not.

* Dim firstTime As Boolean = True

This variable is used to check whether the Start button is used for the first time or not.

* Dim decCounter = 0

This variable is used as a counter whether should the deceleration continue or stop.

## Representation of the Car in the Application

The car is represented as a Rectangle in Dim car = New Rectangle(5, 5, 20, 20). It is called in various Subs, for example, in the Sub PictureBox1\_Paint to initialize the car when the application loads, in Sub moveCar\_Tick to determine the car’s current location, in Sub carMovement to calculate where would the car move, and in Sub RstBtn\_Click to revert the car back to its original position.

## Representation of the Waypoints in the Application

The car is represented as a Rectangle in Dim waypoints = New List(Of Rectangle). It is called in various Subs, for example, in the Sub PictureBox1\_Click to add the newly drawn waypoint to the list with waypoints.Add(waypoint), in Sub carMovement to calculate where would the car move, and in Sub RstBtn\_Click to revert the car back to its original position.

## Next Waypoint Direction and Waypoint Collision Detection Algorithm

Sub carMovement()

Dim dx, dy, d, dr, dur, x, y, NextPosX, NextPosY As Integer

'init

y = car.Location.Y

x = car.Location.X

NextPosX = x

NextPosY = y

'f.e

While waypointCounter < waypoints.Count

dx = waypoints(waypointCounter).X - NextPosX

dy = waypoints(waypointCounter).Y - NextPosY

If NextPosX <= waypoints(waypointCounter).X And NextPosY <= waypoints(waypointCounter).Y - 5 Then

If dx >= dy Then

dr = 2 \* dy

dur = 2 \* (dy - dx)

d = 2 \* dy - dx

Else 'dy < dx

dr = 2 \* dx

dur = 2 \* (dx - dy)

d = 2 \* dx - dy

End If

ElseIf NextPosX > waypoints(waypointCounter).X And NextPosY <= waypoints(waypointCounter).Y - 5 Then

If abs(dx) >= abs(dy) Then

dr = -2 \* dy

dur = -2 \* (dx + dy)

d = -dx - 2 \* dy

Else 'dy < dx

dr = -2 \* dx

dur = -2 \* (dy + dx)

d = -dy - 2 \* dx

End If

ElseIf NextPosX <= waypoints(waypointCounter).X And NextPosY > waypoints(waypointCounter).Y - 5 Then

If abs(dx) >= abs(dy) Then

dr = 2 \* dy

dur = 2 \* (dy + dx)

d = 2 \* dy + dx

Else 'dy < dx

dr = 2 \* dx

dur = 2 \* (dx + dy)

d = 2 \* dx + dy

End If

ElseIf NextPosX > waypoints(waypointCounter).X And NextPosY > waypoints(waypointCounter).Y - 5 Then

If abs(dx) >= abs(dy) Then

dr = -2 \* dy

dur = -2 \* (dy - dx)

d = dx - 2 \* dy

Else 'dy < dx

dr = -2 \* dx

dur = -2 \* (dx - dy)

d = -dx - 2 \* dy

End If

End If

dx = abs(dx)

dy = abs(dy)

'dim point1 as new point(x1 \* 20, y1 \* 20)

If NextPosX <= waypoints(waypointCounter).X - 5 And NextPosY <= waypoints(waypointCounter).Y - 5 Then

If dx >= dy Then

While x < waypoints(waypointCounter).X - 5

carPositions.Add(New Point(x, y))

If d <= 0 Then

'd <= 0 choose r

d = d + dr

Else 'd>0, choose ur

d = d + dur

y = y + 1

End If

x += 1

End While

Else 'dy < dx

While y < waypoints(waypointCounter).Y - 5

carPositions.Add(New Point(x, y))

If d <= 0 Then

'd <= 0 choose r

d = d + dr

Else 'd>0, choose ur

d = d + dur

x = x + 1

End If

y += 1

End While

End If

ElseIf NextPosX > waypoints(waypointCounter).X - 5 And NextPosY <= waypoints(waypointCounter).Y - 5 Then

If dx >= dy Then

While x > waypoints(waypointCounter).X - 5

carPositions.Add(New Point(x, y))

If d > 0 Then

'd <= 0 choose r

d = d + dr

Else 'd>0, choose ur

d = d + dur

y = y + 1

End If

x -= 1

End While

Else 'dy < dx

While y < waypoints(waypointCounter).Y

carPositions.Add(New Point(x, y))

If d <= 0 Then

'd <= 0 choose r

d = d + dr

Else 'd>0, choose ur

d = d + dur

x = x - 1

End If

y += 1

End While

End If

ElseIf NextPosX <= waypoints(waypointCounter).X - 5 And NextPosY > waypoints(waypointCounter).Y Then

If dx >= dy Then

While x < waypoints(waypointCounter).X - 5

carPositions.Add(New Point(x, y))

If d > 0 Then

'd <= 0 choose r

d = d + dr

Else 'd>0, choose ur

d = d + dur

y = y - 1

End If

x += 1

End While

Else 'dy > dx

While y > waypoints(waypointCounter).Y

carPositions.Add(New Point(x, y))

If d <= 0 Then

'd <= 0 choose r

d = d + dr

Else 'd>0, choose ur

d = d + dur

x = x + 1

End If

y -= 1

End While

End If

This Sub is used to calculate the path that the car will take when traversing through the waypoints and to determine the collision detection of the car with the waypoint(s). It works by using the diagonal line algorithm. It determines the dx and dy of point A, which in this case would be the car and point B, the waypoint. It then calculates the d, dur, and dr. Based on the d, whether it is <= 0 or > 0, the algorithm will choose either R or UR. This will determine the new point of the car’s position that is put into carPositions using carPositions.Add(New Point(x, y)).

ElseIf NextPosX > waypoints(waypointCounter).X - 5 And NextPosY > waypoints(waypointCounter).Y Then

If dx >= dy Then

While x > waypoints(waypointCounter).X - 5

carPositions.Add(New Point(x, y))

If d <= 0 Then

'd <= 0 choose r

d = d + dr

Else 'd>0, choose ur

d = d + dur

y = y - 1

End If

x -= 1

End While

Else 'dy > dx

While y > waypoints(waypointCounter).Y

carPositions.Add(New Point(x, y))

If d <= 0 Then

'd <= 0 choose r

d = d + dr

Else 'd>0, choose ur

d = d + dur

x = x - 1

End If

y -= 1

End While

End If

End If

waypointCounter += 1

NextPosY = y

NextPosX = x

End While

End Sub

## Car Acceleration and Deceleration Algorithm

Private Sub moveCar\_Tick(sender As Object, e As EventArgs) Handles moveCar.Tick

canvas.DrawEllipse(Pens.White, car)

If decelerate = False Then

decCounter = 0

If speed > 5 Then

speed = speed - 5

moveCar.Interval = speed

End If

If carPosCounter < carPositions.Count - 10 Then

car.Location = carPositions(carPosCounter)

carPosCounter = carPosCounter + 1

Else

decelerate = True

End If

Else

If decCounter < 10 Then

car.Location = carPositions(carPosCounter)

carPosCounter = carPosCounter + 1

If speed < 105 Then

speed = speed + 5

moveCar.Interval = speed

End If

End If

If speed >= 105 Or carPosCounter = carPositions.Count Then

moveCar.Enabled = False

End If

End If

canvas.DrawEllipse(Pens.Blue, car)

End Sub

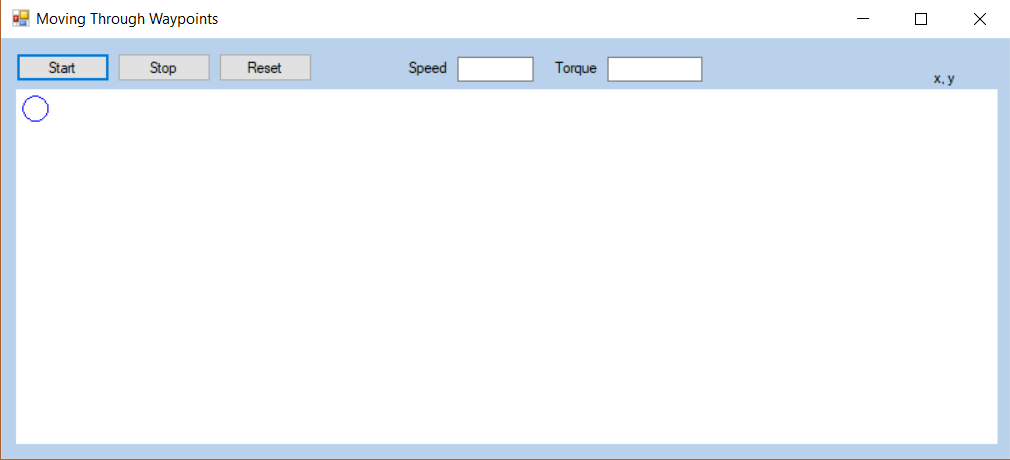
The Sub is used to determine whether the car should decelerate and at which moment the car should start accelerating and also determines its maximum speed. The variable speed is used as a placeholder for the Interval for the moveCar Timer. When the car accelerates, the variable speed will be decreased by 5 and the variable speed will not reach the value of > 5, as it is the maximum speed. The car can also decelerate and it can be triggered by the user or the algorithm. When carPosCounter is < carPositions.Count – 10, it will turn the Boolean decelerate = True and will increase the speed by 5 until it has reached 105 or when carPosCounter = carPositions.Count.

# Evaluation

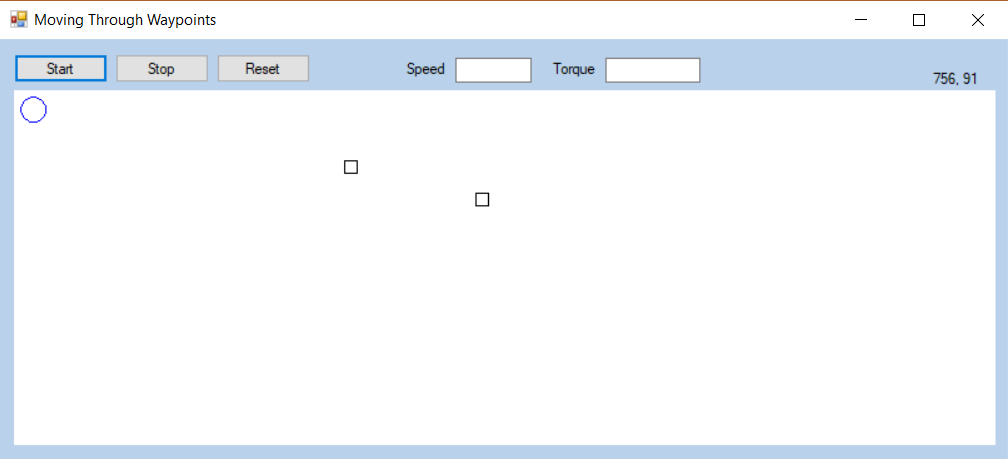
To ensure that the application works as it is designed, test cases are conducted. These test cases are as follows:

## Adding Waypoints

In this part the car has not move yet. Set the speed or waypoints first to make the car move.



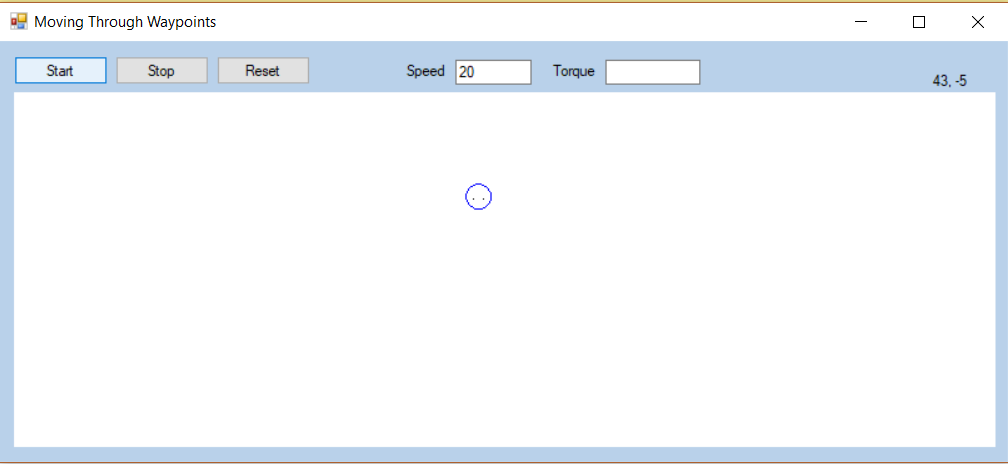
In this part, the waypoint has been set.



The test case is successful since the applcation proves to be able to create a waypoint.

## Moving the Car to Waypoints and Stopping After It Has Reached the Final Waypoint

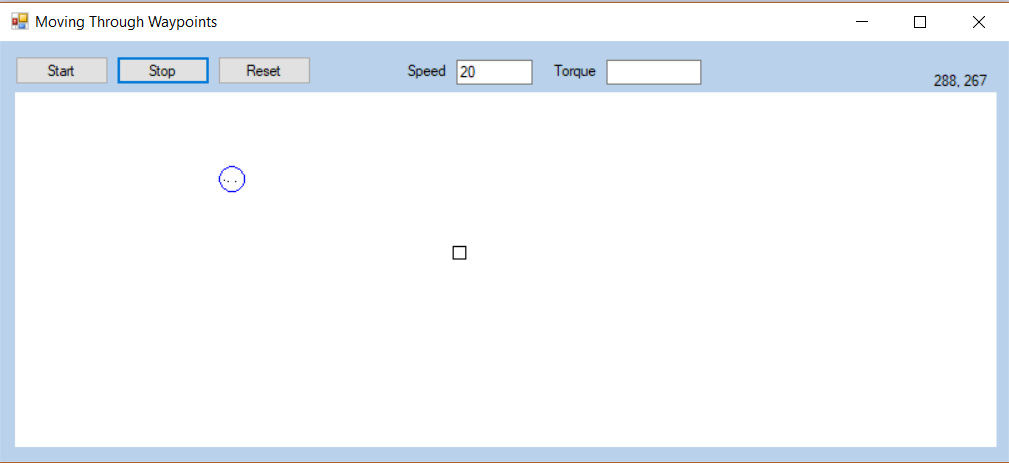
In this test case, the car will be tested whether it will stop on the final waypoint or will it move past the waypoint.

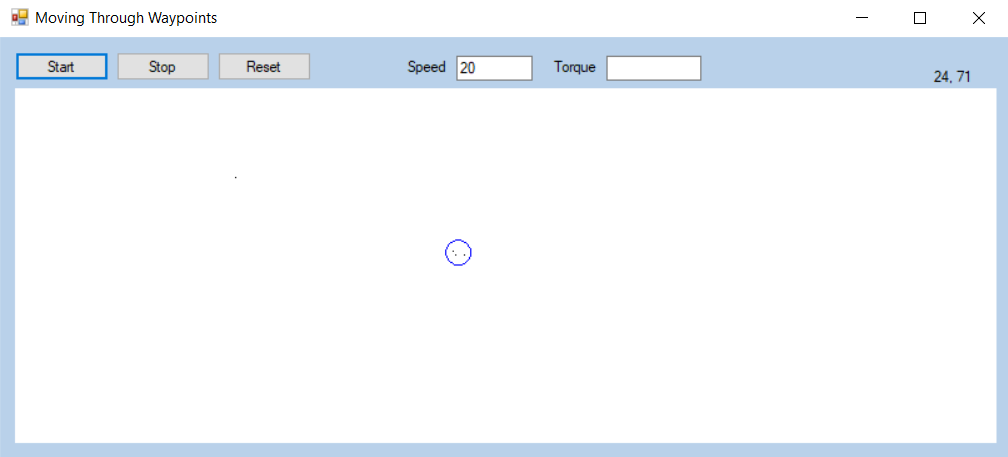


The test case is successful since the applcation proves to be able to stop the car when it reaches the final waypoint.

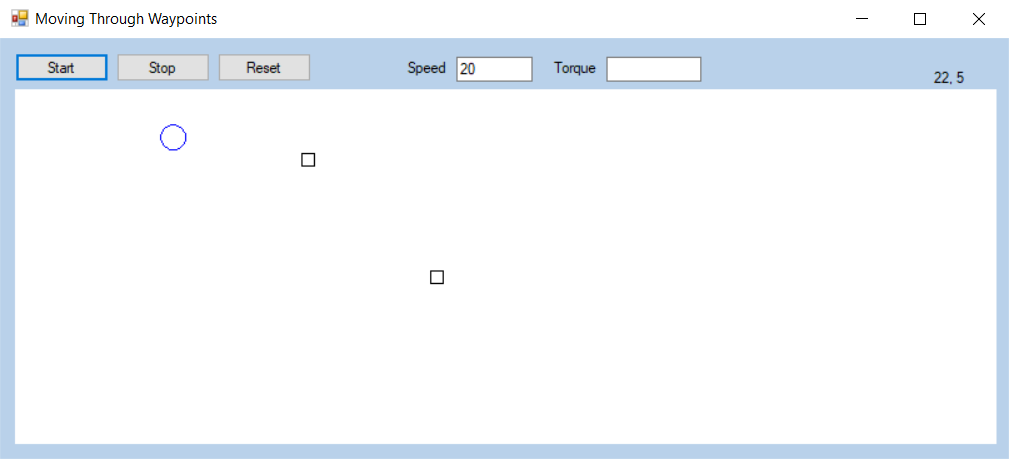
## Stopping the Car Abruptly and Making It Move Again

In this test case, the car will be stopped abruptly and then it will be moved again after it has stopped.



 The test case is successful since the applcation proves to be able to move the car again after the car has been stopped

## Increasing the Car’s Acceleration

In this test case, the car’s speed will be accelerated by time until it reaches the maximum speed.

The test case is successful since the applcation proves to be able to accelerate the car.

# Work Log

The work log is extracted directly from Visual Studio’s Git Log History, which is also available publicly at <https://github.com/bakanui/MovingThroughWaypoints/commits/master>.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| No | Author | Date | Time | Commit Message |
| 1 | Bhaskara Ida Bagus | 11/21/2017 | 7:09:33 PM | Initialized project |
| 2 | Bhaskara Ida Bagus | 11/21/2017 | 7:16:34 PM | Added blank report document |
| 3 | Bhaskara Ida Bagus | 11/21/2017 | 9:24:34 PM | Update report with blank chapters |
| 4 | Bhaskara Ida Bagus | 11/22/2017 | 6:14:05 PM | Migrating to WinForms from WPF |
| 5 | Bhaskara Ida Bagus | 11/22/2017 | 9:20:21 PM | Now able to add waypoints via mouseclick |
| 6 | Bhaskara Ida Bagus | 11/22/2017 | 9:21:04 PM | Add car with init coordinates (5, 5) |
| 7 | Vera Debora Vitamas | 11/22/2017 | 9:51:51 PM | unfixed Basic Theory |
| 8 | Rahmad Martin | 11/24/2017 | 7:31:38 AM | #car now can move through waypoints but so fast |
| 9 | Vera Debora Vitamas | 11/25/2017 | 12:10:31 PM | Unfixed Basic Theory. The algorithm does not exist yet. |
| 10 | Rahmad Martin | 11/27/2017 | 11:29:47 PM | #car update #delete picturebox #justtofirstwaypoint |
| 11 | Rahmad Martin | 11/28/2017 | 10:49:44 PM | #moving through waypoints |
| 12 | Vera Debora Vitamas | 12/1/2017 | 12:52:35 AM | was trying to make an acceleration |
| 13 | Vera Debora Vitamas | 12/3/2017 | 12:36:45 PM | Fixed Basic Theory. |
| 14 | Bhaskara Ida Bagus | 12/3/2017 | 4:51:14 PM | Simplifying the code, add speed |
| 15 | Vera Debora Vitamas | 12/3/2017 | 6:01:58 PM | Accelerate and Decelerate |
| 16 | Vera Debora Vitamas | 12/3/2017 | 6:28:08 PM | update design |
| 17 | Vera Debora Vitamas | 12/3/2017 | 7:44:03 PM | Update design with Torque Textbox |
| 18 | Vera Debora Vitamas | 12/3/2017 | 9:07:42 PM | made an implementation |
| 19 | Bhaskara Ida Bagus | 12/3/2017 | 8:53:25 PM | Fix car stop while decelerating, forgot to code the reset button |
| 20 | Vera Debora Vitamas | 12/3/2017 | 9:09:39 PM | design merging |
| 21 | Vera Debora Vitamas | 12/3/2017 | 10:20:34 PM | updating evaluation |
| 22 | Bhaskara Ida Bagus | 12/3/2017 | 11:58:34 PM | Finalizing report |

# Conclusion and Remarks

The program works as expected, although we didn’t manage to get the torque algorithm working. Other than that, the application proves to be successful in moving the car to the waypoint(s), generating the waypoint(s), accelerating and decelerating the car, stopping the car, and resetting the state of the program to its initial state.

Through this assignment we learned that hard work pays off. We feel that we did better on our time management this time, but even then we feel that we didn’t manage to deliver what is expected of us.