Optimizing Paths in a Weighted Plane featuring Mario Kart Wii

Independent Study Thesis

Presented in Partial Fulfillment of the Requirements for the Degree Bachelor of Arts in Mathematics in the Department of Mathematics at The College of Wooster

by
Jack Doughty
The College of Wooster
2021

Advised by:

Dr. Colby Long (Applied Mathematics)

Abstract

Mario Kart is often seen as a game of luck, but perhaps the most intriguing element to arguably the most iconic racing game of all time is how quickly one can drive on the race track. This is not just meant as a comparison of one vehicle's velocity to another, but rather a contrast of techniques and strategies used by players to complete laps as quickly as possible. Attempting to drive tracks as fast as possible has been around ever since the birth of the game, and with that in mind, I aimed to create an optimal strategy which would allow us to ultimately calculate one such optimal path around a sample race track.

We began by researching the intricacies of the game, and trying our best to mathematically describe observable behavior. The amount of essential mechanics and physics that needed to be accounted for, along with the data to be collected was practically boundless. This involved creating new vocabulary and models to fully depict all possible actions, along with a preliminary understanding of geodesics and weighted planes. While originally it was planned to be much more involved with machine learning, the project morphed over time into the creation of a mathematical method to properly optimize driving. Much of this required data was not obtainable through browsing the internet, which resulted in extensive computational analysis to produce metrics and data that was integral to this project. Some of these techniques involved data-mining game code, emulation of the game to a laptop, creation of code to display data live through emulation, scripting,

TASing, modeling, and much more. Ultimately this was all done in an endeavor to not only observe our vehicles behavior, but describe it mathematically.

Compiling this data took an extensive amount of time and effort, which unfortunately left not as much time as expected for the mathematical aspect of this project. Unfortunately not all of the work I previously researched was able to be implemented, but that was to be expected from such a large scale project. By the end of this venture, I managed to create an upper-bound path and time for the first straight away and turn, but also defined a methodology that could be replicated on any possible turn to find an optimal path. Should I have had more time, the implementation of a rudimentary Monte Carlo algorithm or potentially reinforced learning in the long term could have been created to truly optimize the mathematical models presented, as realistically you can only go so far into this branch of mathematics without going into incredibly complicated and tedious math.

This work is dedicated to The Mario Kart Wii community, Omega, Maximum, MY13, NW, WA, HT, and @ue.

Acknowledgments

None of this work could have been possible without the Mario Kart Wii TAS community, and I would like to whole-heartily acknowledge everything they have done to help with this project.

Contents

CHAPTER 1

Elementary Mathematics of Shortest Paths

1.1 Geodesics

One of the primary focuses of this project is trying to find optimal paths in weighted planes. We will dive into more of what each of those individual terms means later, but for now we can start by thinking about shortest paths. On non-weighted plane, a shortest path is synonymous with the quickest path, as well as the optimal path. This is rather simple when dealing with planes, but due to the complexity of our scenario we will have to expand into the realm of geodesics.

When trying to find an optimal path for our vehicle, we can immediately make comparisons to geodesics. A geodesic is a length-minimizing curve between two points. Normally in Euclidean Geometry it is just a straight line between the two points, but when referring to geodesics it is usually implied to be relevant in 3 dimensions. A common example is the geodesic lines on a sphere. As seen below in figure 1.1, length-minimizing curves are sketched out between points to indicate the shortest path that can be taken to each point from each point. Each of these we could define as a geodesic.

While this is a rather basic example, it does not only apply to spheres. The idea of geodesics applies to the vast majority of surfaces, with some other notable

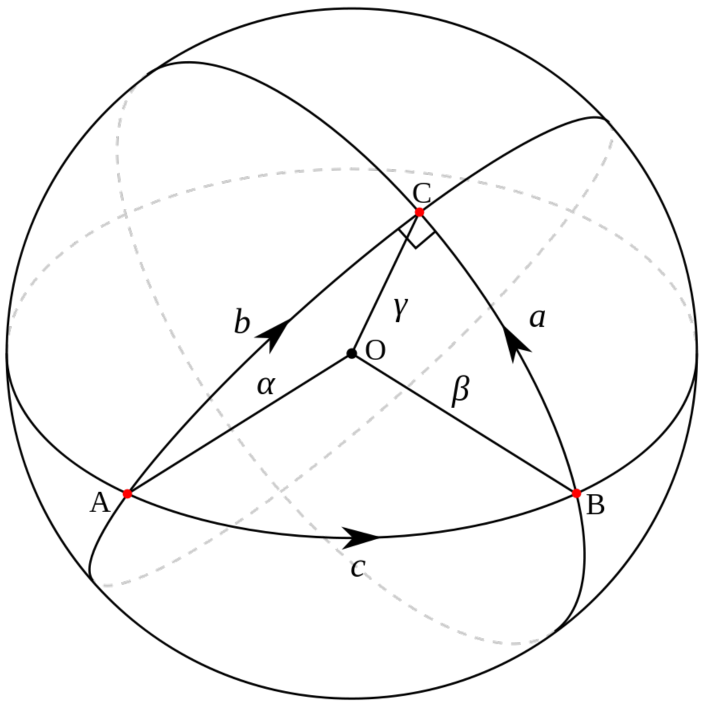


Figure 1.1: Geodesics on a sphere

examples revolving around Möbius strips, tori, cones, and saddles. If you want to find a geodesic for the sphere in figure 1.1, we can calculate our geodesic by first identifying the greater circle which passes through our two points. For example, if we want to identify the geodesic between points A and C, we would find the circle AC that passes through both the points and is a circumference of the sphere. From there, we know that our geodesic b lies on our circle AC as an arc. This we could then conclude is the shortest path along the sphere between points A and C. This should hopefully give a decent intro determining rather simple optimal paths in 3-dimensional space, but for our project we will only be focusing on 2-dimensional space.

1.2 Weighted Planes

Once again going back to Euclidean Geometry, it could be easy to assume the fastest path between two points is just a straight line Hershberger and Suri [4]. However, this changes drastically in a weighted plane. We can not assume that the fastest path for our vehicle to drive is a straight line between our beginning and ending, as it would likely violate several of the rules we need to follow within the game (for example, we cannot go through walls). We can imagine our course we need to drive on as a plane, and then subdivide this plane into weighted sections. Weighted means that there are modifiers that impact or inhibit the velocity and travel of our vehicle. For example, one section could have a modifier of 2, while another has a modifier of .5, and another has a modifier of 0. Our optimal path we could define not necessarily as a straight line, but rather the shortest path calculated via weight. It is often the case that the shortest weighted path is not the shortest path by distance.

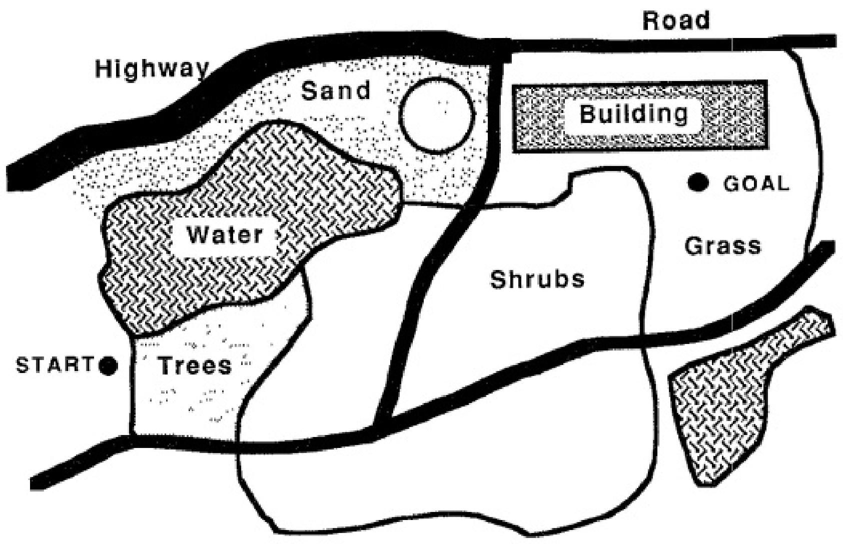


Figure 1.2: Real world example of a weighted plane Chen et al. [2]

As seen in figure 1.2 Chen et al. [2], we have a very illuminating example for how this concept of subdivided weighted planar paths may work in a much easier to comprehend scenario. Say we want to walk from the vertex labeled “START” to the vertex labeled “GOAL”. Our objective is not to solve the shortest path via Euclidean distance, rather find the shortest weighted path. Let’s imagine in this example it’s incredibly slow to walk through trees as it is a remarkably dense set of trees, and swimming is significantly slower than running. It would actually take much more time to take the direct path to the goal than going around in the sand.

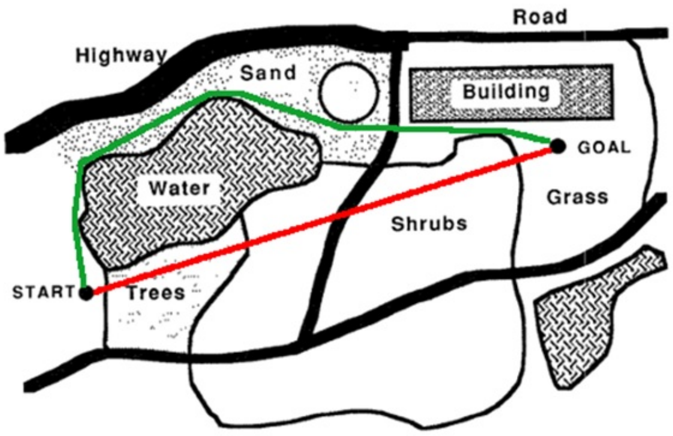


Figure 1.3: Optimal path of figure 1.2

The most optimal path would be the one marked in green on figure 1.3, not the shortest distance path marked in red. This is because our path in red not only goes through the slow tree section, but also through a giant section of shrubs which would be sure to slow us down whereas the roundabout path goes through no difficult terrain whatsoever.

Now that we have illustrated how the problem works in a more basic sense, let’s mathematically define and explain everything in a more rigorous manner. When we are supplied with any straight-line planar polygonal subdivision (P), we can further divide P as a set of various faces, edges, and vertices with each edge connecting two faces. Faces also may only connect at a shared edge or vertex. Edges must be closed line segments, and their endpoints must be vertices. If some edge e connects faces f and f' such that it is oriented with f on the right, then we can denote e as the following:

$$e = \bigcap(f, f')$$

Furthermore, if there is some undirected edge e we write:

$$e = f \bigcap f' \text{ Mitchell and Papadimitriou [7]}$$

We were provided with two significant points in our earlier example: s (start) and g (goal). Each of our faces also has an associated weight a specifying the “cost per unit distance” of traveling in that region. Our final objective then is starting at point s, we must find a path in this subdivided plane that minimizes some total cost function. Referring back to Figure 1.2, we can hypothesize that sections such as water or trees may have a larger "cost per unit distance" than something like sand, as sand is much easier to walk through and easier to traverse quickly than something like a thick forest or lake (this is very subjective I reckon but for the example let us just assume this is true). This mathematically explains why our shortest Euclidean distance is not always our most optimal, or shortest/least weighted path.

1.3 Trajectory Planning

Topology can also play a role in the creation of our optimal model for our vehicle to drive on a course. The first step to design a race driver model is represented by trajectory planning. What we call our optimal trajectory is the trajectory that allows the vehicle to obtain the lowest lap time. One good example of a model was academically described as follows:

Our robot A is similar to a car-like vehicle moving on a planar environment. Its body is a rectangle supported by four wheels: the two rear wheels' axle is fixed to A's body and the two front wheels are directional. A position of this robot is given by a configuration (x, y, θ, K), where (x, y) are the coordinates of a reference point R of the body, θ is the orientation of the body (i.e. the angle between the x-axis and the main axis of A) and K is the instantaneous curvature of R's curve and represents the orientation of the front wheels Scheuer and Laugier [8]

We can then replace a lot of the variables in this model to apply to our potential scenario. Our variables x and y can remain the same as we are also projecting our vehicle A onto a planar surface, along with angle θ which also represents our vehicle's angle from the x-axis. However, K would not represent the orientation of the front wheels of our vehicle, rather just the instantaneous angle for the velocity. With this fundamental understanding of how our vehicle would be modeled, we can now try to grasp further understanding by applying further constraints to our model.

1.4 Local and Global Planning

One of the most challenging aspects about forming weighted paths is analyzing each subdivision and assessing them in a piece wise manner. Not only does an optimal path have to manage each subdivision well at a local level, but it also has to properly assess the global layout of the plane. These two perspectives are referr4ed to as local and global planning Asadi and Atkins [1].

Mathematically speaking, the path planning is performed using something called a local planner. Local planners are associated but contrasted with a higher level method to obtain the global planner. The local planner does not take obstacles into account, it only searches for the shortest, feasible, and smooth path linking both configurations. two configurations, while the global planning deals more with collision avoidance with obstacles.

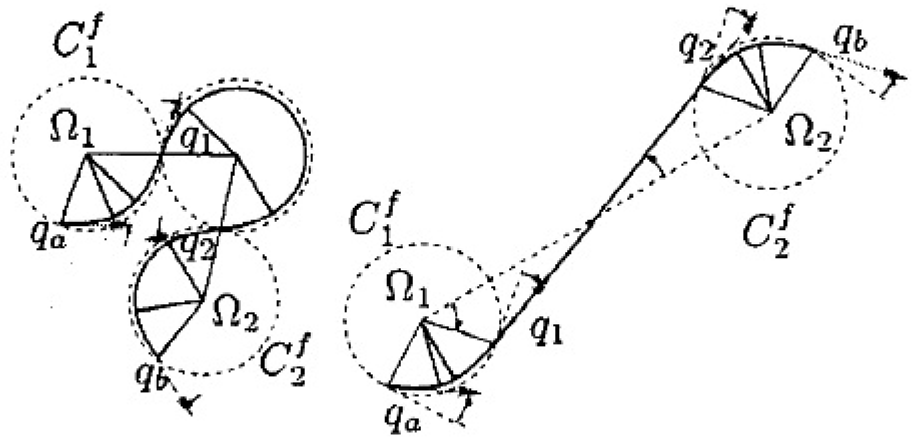


Figure 1.4: Basic examples of local planning for optimal paths Scheuer and Laugier [8]

As seen in Figure 1.4, we have two fundamental examples of local planning. The path of our vehicle can be seen as the solid curvature path, with q_a and q_b being our starting and ending points respectively in each example. While the polygon for all possible paths is not shown (the race track), we can see where each turn centers around. Using this knowledge, we have found a path for our vehicle that takes each turn as tight as possible, along with starting each turn at an optimal angle. Due to the context of this project, we will be focusing primarily on local planning as attempting to calculate even the smallest of sections would require substantial global planning.

Even so, these local planning sections will have many more complications than it may seem. For example Figure 1.5 shows how paths have to accommodate for the size or what is commonly referred to as the hitbox of our vehicle. However, we could just remodel the track or polygon for our path with such little obstacles to achieve a similar effect without delving into global planning.

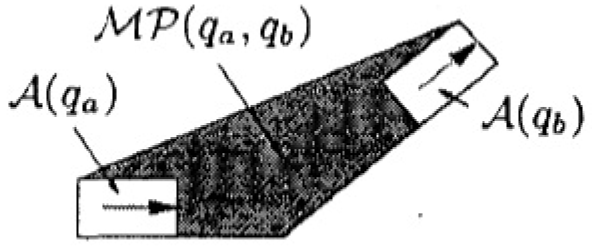


Figure 1.5: Motion polygon visualizing the constraints of our vehicle Scheuer and Laugier [8]

1.5 Mathematical context

While the mathematical research presented here is not incredibly rich with detail, this was largely due to the amount of time and effort required to collect data manually when I had very little guidance or understanding of what needed to be done. That being said, while it is quite brief it certainly explains most of the upper level mathematical concepts that will be utilized implicitly and explicitly throughout the rest of the thesis.

CHAPTER 2

Applications

2.1 How does Mario Kart Wii work, and how can we represent it mathematically?

Now that we've prefaced much of the strategies for finding shortest paths, we're now going to attempt to apply it to Mario Kart Wii. In Mario Kart, we will attempt to create an algorithm which identifies not necessarily the shortest length path, but the shortest weighted path. In Mario Kart Wii we will be playing in the time trials mode, which gives our character 3 mushroom items which allows it to temporarily increase its speed and drive through offroad. The end goal in time trial mode is to complete a lap around the track 3 times as fast as possible. This will involve our character moving in different methods in the game, and structuring our path around it. While finding a shortest path around a circuit would be rather trivial, implementing constraints on the shape of the path through vehicle movement and computing length as a function of time traveled as opposed to distance adds a much further step of depth for the game.

We can classify Mario Kart Wii as a deterministic game, which can be defined as repeated actions having consistent results Solan and Vielle [9]. A good example of this is that if we start at a specific position and angle on our race course and only hold down the gas button for x frames and it travels y in-game units, it will consistently travel y in-game units in those specific circumstances. This is incredibly important, as that means our mathematics will continue to produce the same results.

In a racing game, one of the most important aspects is the different methods of movement. Below are several of our primary methods:

- Wheelie
- Drifting
- Miniturbo
- Mushroom Boost
- Boost Panel

These are all of the classifications of actions that our vehicle can take as we define them. Not all of them are mutually exclusive, and some of them have several different methods and sub-methods we will delve into further in the chapter.

2.2 What are we measuring? How are we controlling our vehicle?

In the scope of this project there are several key things we must take note of. For starters, the game runs at 59.54 frames per second (or FPS) [3], but we round it up to 60 just for simplicity sake. This means we can only have 60 actions per second, or 60 unique inputs a second. We can classify an input as any possible combination of button presses that combine to make an instance. For example, we can press two different buttons in the same instance such as A and B, but we cannot press up and down on the joystick in the same instance as it is only possible to have 1 direction on the joystick at a time. However, it became clear very quickly that only allowing 60 actions a second would become very limiting mathematically so I decided to allow a non-finite amount of actions to be made in any time segment. For example, if we were to only take 60 actions per second our curve would no longer be smooth, it would look similar to a Riemann sum representation below the curve. Ignoring this will allow us to mathematically represent vehicle trajectory much more simply. If this were created into a discrete mathematics problem in a similar method, it would provide an entirely different set of mathematics to answer and an optimal solution could be more easily determined through reinforcement learning algorithms where our reward algorithm would have to do with setting up checkpoints along the track and only allowing our vehicle certain sets of inputs. while it could certainly be a valid endeavor, we decided to forgo that in this project.

With that out of the way, let us look more closely at what we can observe at any given time about our vehicle.

- Location- At any given point in our race, our vehicle will have an x, y, and z value that corresponds to its location on the track. Fortunately for us, we chose a racetrack which is entirely flat and has no changes in elevation so we can neglect to track any location on the z-axis. This leaves us with only x and y. Using these coordinates we can determine whether or not our vehicle is driving in the offroad, or next to an object, or on the track.

- Velocity- velocity is one of the more obvious measurements we need to take. Traveling at fast velocities is obviously more optimal than traveling at slow velocities (assuming all other variables are constant), so measuring velocity and determining where we can drive faster is going to be very important.

- Direction- This is incredibly essential for our vehicle. Without direction we do not know where we are going to be in our next observation, and since our direction can be modified instantaneously with inputs it is even more important that we measure this intently.

- Button presses- This is pretty self explanatory, but it is important our vehicle knows exactly what buttons were pressed when we control it.

- Stick position- This is actually a bit more complicated than it seems, and we will get more in depth with it later, but in essence there are several different directions the stick can be put to change the trajectory of the vehicle. This observed variable goes hand in hand with direction.

- Miniturbo charge- There is an arbitrary value named Miniturbo Charge which increases while in a drift until it hits a value of 270 [3]. It can then be "released" to get a slight momentary velocity boost. While they may not seem like a large amount, every little value counts when we are trying to optimize velocity. By keeping track of this value we can determine the exact instant that we can release our miniturbo, and if the optimal path involves us doing two miniturbos back to back (often referred to as a chain miniturbo) then we can do it as fast as possible by monitoring our miniturbo stat and releasing it on the first possible frame.

- Mushroom boost- When a mushroom is used, it raises our character to 117.81 $\frac{u}{f}$ for exactly 90 frames, while also making us immune to the effects of offroad for those 90 frames [5]. By noting how many frames we have in our mushroom, we can determine exactly how many frames we have left with top velocity and offroad immunity. This is incredibly important for optimizing our path in offroad sections, specifically through certain shortcuts.

- Mushroom count- This is simply how many mushrooms we have left. In reality we are only using 1 each lap so this may be redundant, but it does not hurt us to take it.

This should be the entirety of what we monitor for our vehicle, but there still is much to learn about several of these as they are much more complex than they may first appear. Finally, all measurements of velocity will be in $\frac{u}{f}$ or $\frac{u}{s}$ throughout the rest of this paper. Frames are simply $\frac{1}{60}$th of a second.

2.3 A deep dive into game mechanics

For such an old game, there is a surprisingly large amount of strategy to optimize driving to an incredible level. We will begin by looking into each of these techniques so we can eventually utilize some of them ourselves, as well as familiarize ourselves with common terminology.

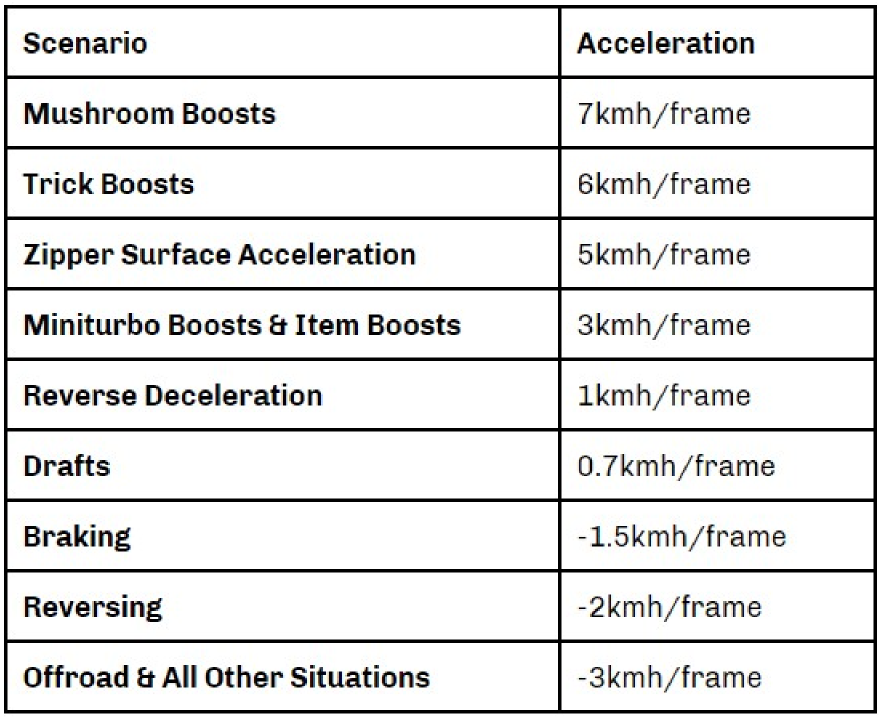
2.3.1 Acceleration

There are a various amount of ways a vehicle can accelerate and decelerate in this game. However, the magnitude of acceleration changes depending on the exact action. It is important to know exactly how these interactions change vehicle velocity, which are all displayed in figure 2.1.

2.3.2 Boosts

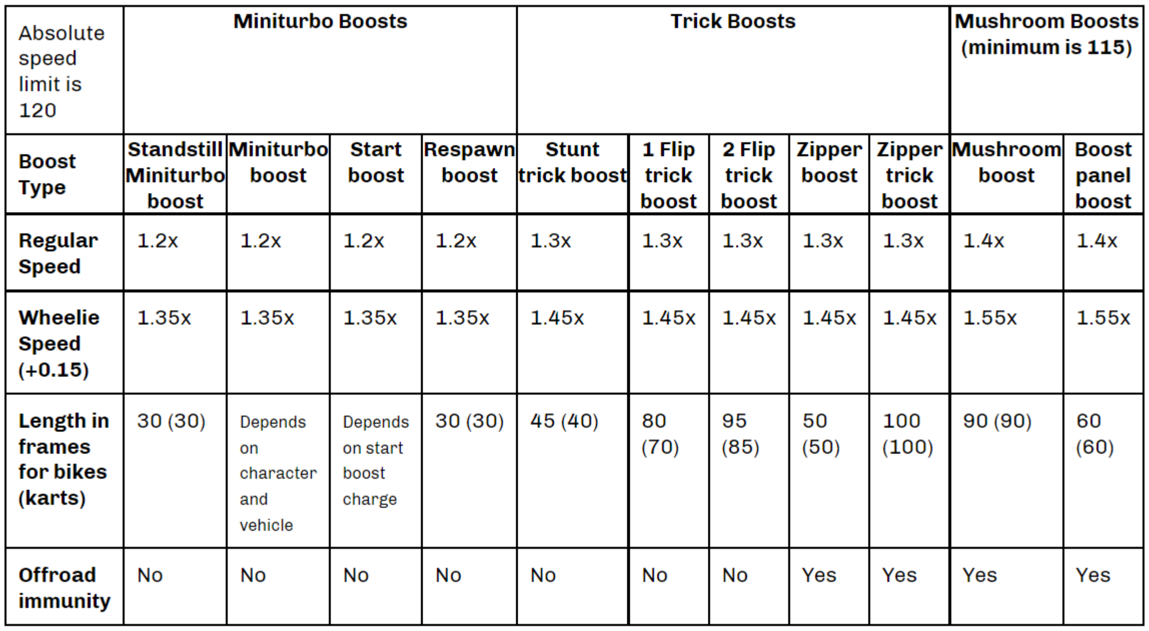
The most common way of accelerating beyond normal speeds is through the use of a boost. There are several different types of boosts, all of which are outlined below in figure 2.2.

These boosts do not stack acceleration however, so when the vehicle is acted



Scenario	Acceleration
Mushroom Boosts	7kmh/frame
Trick Boosts	6kmh/frame
Zipper Surface Acceleration	5kmh/frame
Miniturbo Boosts & Item Boosts	3kmh/frame
Reverse Deceleration	1kmh/frame
Drafts	0.7kmh/frame
Braking	-1.5kmh/frame
Reversing	-2kmh/frame
Offroad & All Other Situations	-3kmh/frame

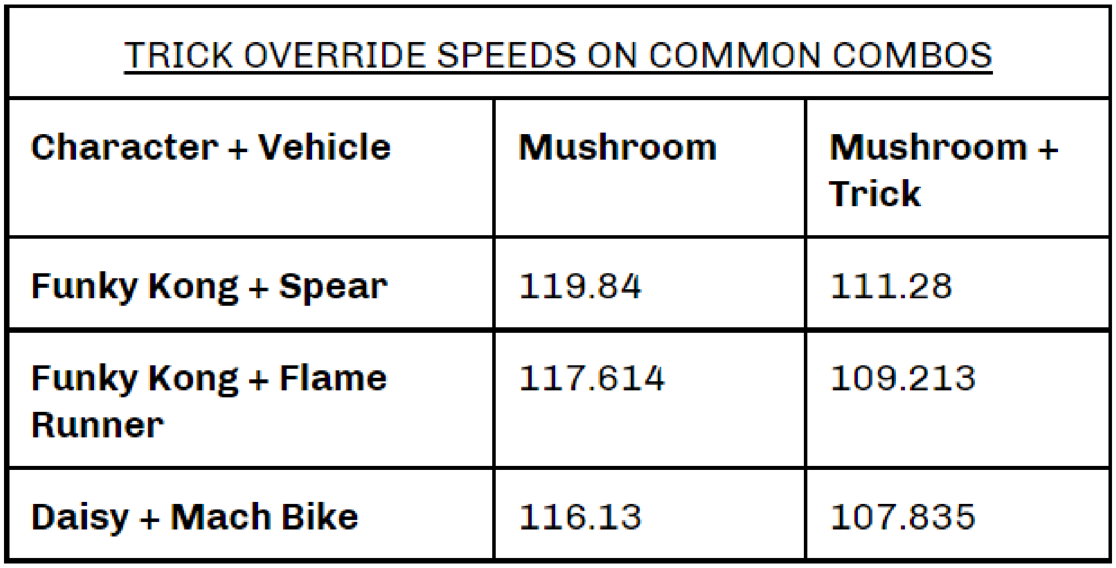
Figure 2.1: All Acceleration Types and Quantities [5]



Absolute speed limit is 120	Miniturbo Boosts				Trick Boosts					Mushroom Boosts (minimum is 115)	
Boost Type	**Standstill Miniturbo boost**	**Miniturbo boost**	**Start boost**	**Respawn boost**	**Stunt trick boost**	**1 Flip trick boost**	**2 Flip trick boost**	**Zipper boost**	**Zipper trick boost**	**Mushroom boost**	**Boost panel boost**
Regular Speed	1.2x	1.2x	1.2x	1.2x	1.3x	1.3x	1.3x	1.3x	1.3x	1.4x	1.4x
Wheelie Speed (+0.15)	1.35x	1.35x	1.35x	1.35x	1.45x	1.45x	1.45x	1.45x	1.45x	1.55x	1.55x
Length in frames for bikes (karts)	30 (30)	Depends on character and vehicle	Depends on start boost charge	30 (30)	45 (40)	80 (70)	95 (85)	50 (50)	100 (100)	90 (90)	60 (60)
Offroad immunity	No	No	No	No	No	No	No	Yes	Yes	Yes	Yes

Figure 2.2: Types of boosts and their consequences [5]

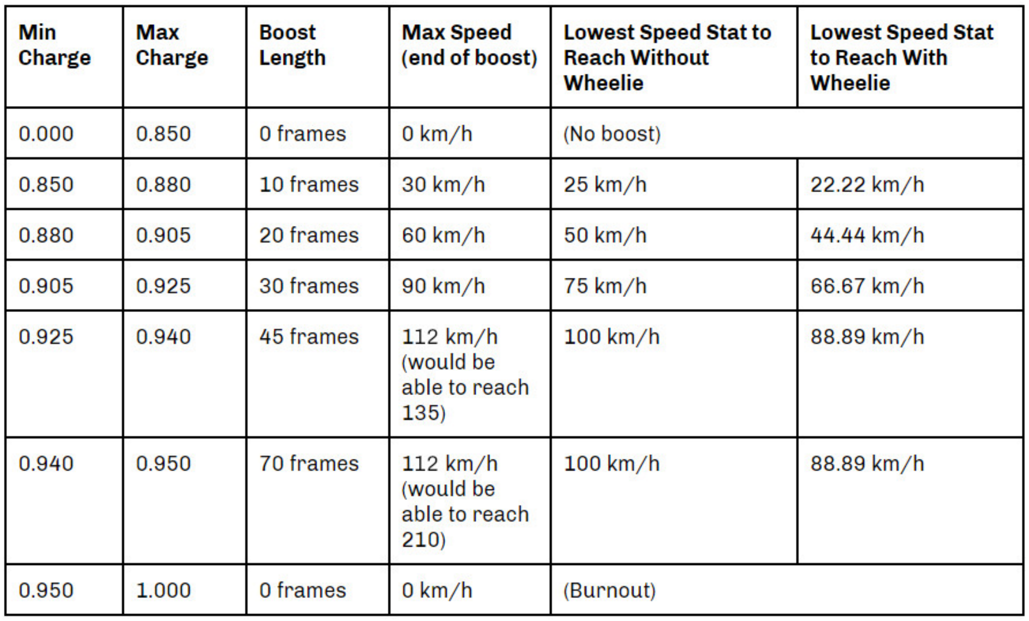
upon multiple boosts it simply applies the greatest positive value. A diagram representing this can seen in figure 2.3.



TRICK OVERRIDE SPEEDS ON COMMON COMBOS		
Character + Vehicle	**Mushroom**	**Mushroom + Trick**
Funky Kong + Spear	119.84	111.28
Funky Kong + Flame Runner	117.614	109.213
Daisy + Mach Bike	116.13	107.835

Figure 2.3: Overview for boost priority and their velocities [5]

The startup boost, which only occurs at the beginning of each race, is very unique in the sense that there are many different variations in terms of duration that can be achieved. Since it will always be more optimal to get the maximum length of boost we will use the lowest row of figure 2.4 in our calculations, the other rows are there to provide more context to the reader with regards to variation.



Min Charge	**Max Charge**	**Boost Length**	**Max Speed (end of boost)**	**Lowest Speed Stat to Reach Without Wheelie**	**Lowest Speed Stat to Reach With Wheelie**
0.000	0.850	0 frames	0 km/h	(No boost)	
0.850	0.880	10 frames	30 km/h	25 km/h	22.22 km/h
0.880	0.905	20 frames	60 km/h	50 km/h	44.44 km/h
0.905	0.925	30 frames	90 km/h	75 km/h	66.67 km/h
0.925	0.940	45 frames	112 km/h (would be able to reach 135)	100 km/h	88.89 km/h
0.940	0.950	70 frames	112 km/h (would be able to reach 210)	100 km/h	88.89 km/h
0.950	1.000	0 frames	0 km/h	(Burnout)	

Figure 2.4: Different start boosts and corresponding timing [5]

2.3.3 Wheelies

On other way to travel at a faster speed is a movement technique called a wheelie. When you start a wheelie, your vehicle remains in this wheelie for 180 frames (3 seconds) or until cancel by the user. While in a wheelie, inputting any buttons on the stick cause it to decelerate significantly. Due to this, it is important that the stick remains neutral while in a wheelie. As a trade-off, the vehicle's maximum speed is increased by 15%. This makes wheelie's optimal for straightaways, especially very long ones.

2.3.4 Joystick positions and Miniturbos

There are several different positions that a joystick can be at for any given time. In fact, there are a total of 225 different positions. Usually we refer to these positions either on a scale from -7 to 7, or from 0 to 14. In Figure 2.5 there is an example of exactly how this layout conforms to the stick, with 0,0 being the center.

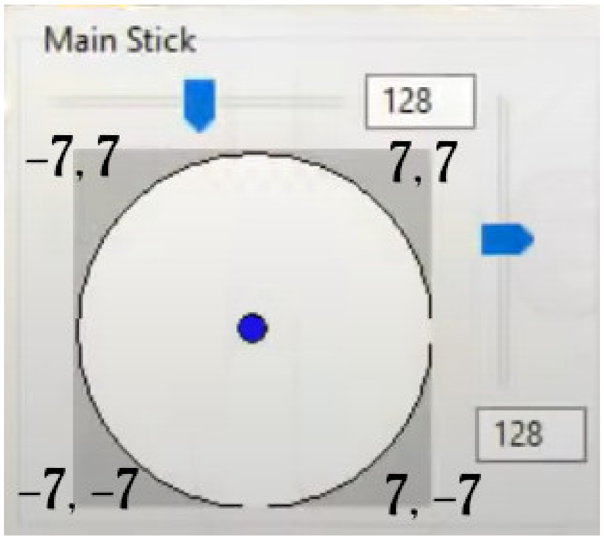


Figure 2.5: Diagram of the joystick and its input range

When performing a drift a miniturbo will charge up after a certain period of time, and when you release the drift, you will get a boost from this miniturbo.

Miniturbos charge at different rates depending on which direction you hold during that drift. Say you wish to turn right, our miniturbo will be created at the optimum pace if the stick is held with an x-coordinate between 3 and 7, and slower for an x-coordinate between -7 and 2.

We measure this rate of charge for a miniturbo numerically. We can use a miniturbo once our charge reaches a value of 270. When charging in the most optimal zone, we can charge by 5 units each frame. When using the least optimal zone, charge only increases by 2 units each frame.

2.3.5 Hitboxes

Like most objects in games, our vehicle has a hit box. A hit box is a geometric representation for the collision of our vehicle. One example of how this works is when if our vehicle drives directly into an object (say a wall), the hit box of our vehicle would collide with the hit box of the wall. This prevents objects from moving through each other. Properly determining the hitbox of our vehicle is important for determining exactly how tight we can take our turns on the track.

Much of the code for hitboxes was found by EjayB and Citrinitas [3]. First and foremost, we can lay out the hitboxes we have to deal with here as there are in fact several. Each of them come with a center point using x, y, and z coordinates. There is also a given radius of our sphere. From this data, we can begin to structure what this hitbox would look like. While the first 5 hitboxes are simple spheres, the later 2 are not. They appear to have very special properties that make it much more difficult to understand positioning. We can begin however by modeling the very basic 5 beginning hitboxes.

As for the other two not included hitboxes (Wheel 0 and Wheel 1), we can also add them to our model but it does not look quite right. Not only are they slightly below our XY plane, but one also appears to be slightly higher than the other. This

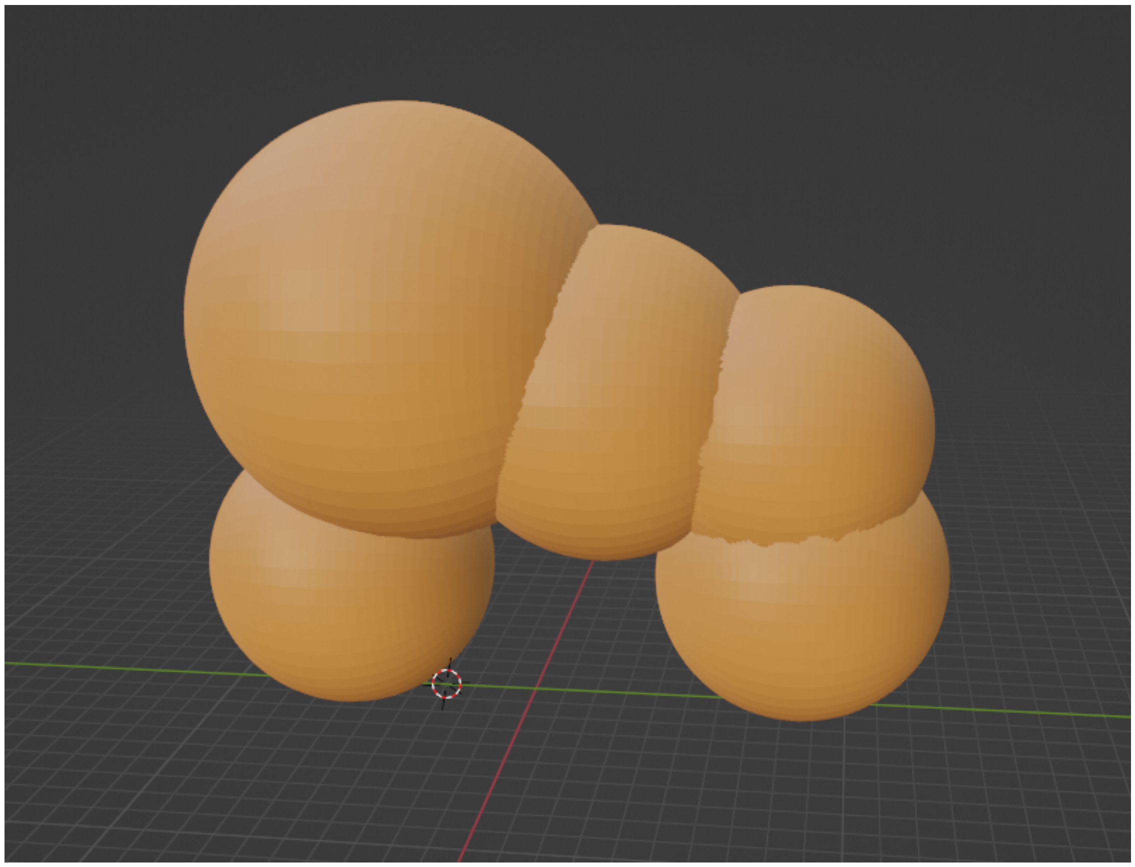


Figure 2.6: Basic hitbox with only the vehicle body

difference is 50 units. Based on in game representation however, it appears these two remained unused throughout development so they were probably just remnant code that was not removed.

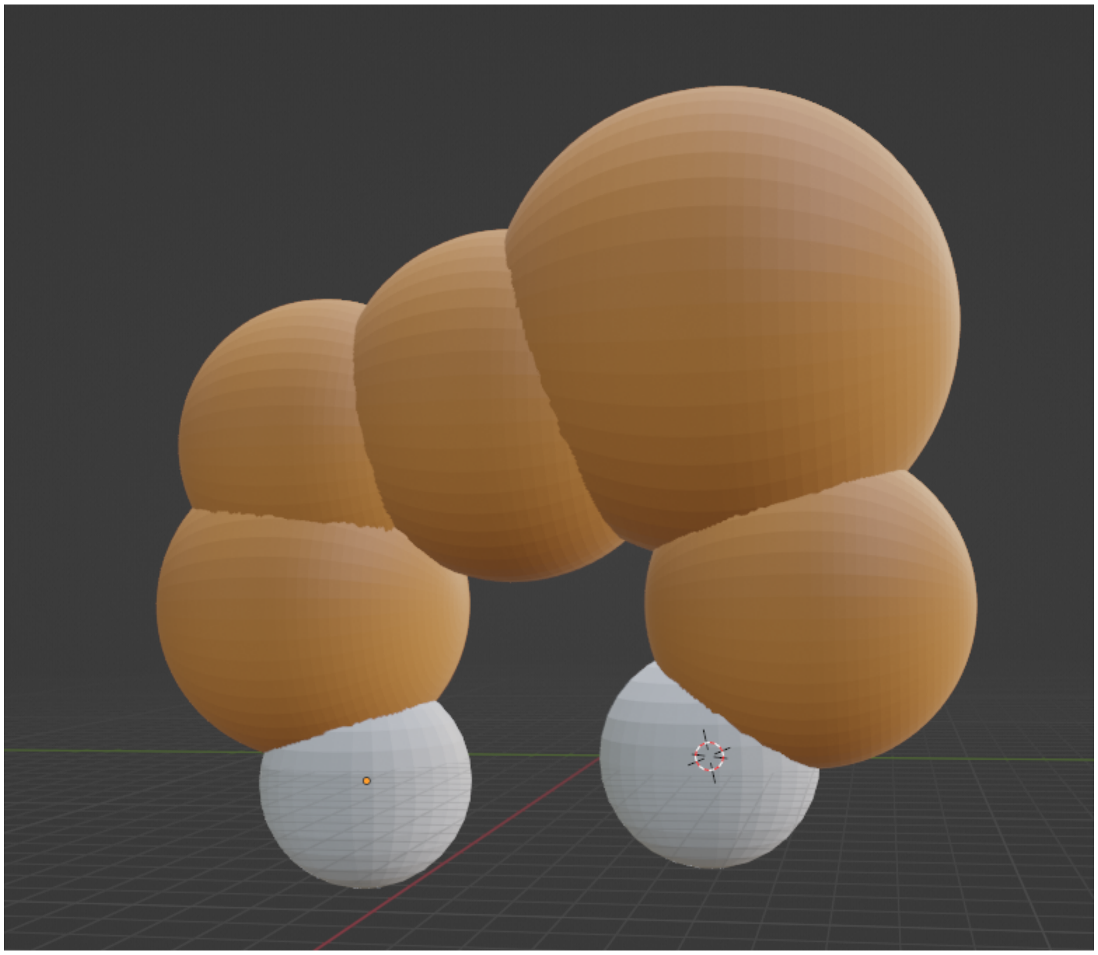


Figure 2.7: Basic hitbox with attached wheels

CHAPTER 3

Creating a Theoretical Lower Bound

3.1 Theoretical Lower Bound

Now that we have properly analyzed the vast amount of strategy available at our disposal, we can begin to optimize how we want to drive the track. Our main goal now is to create a theoretically optimal path for our vehicle, starting with the first straight and turn. We will then follow this up by creating a methodology that would allow anyone to calculate a theoretical optimal way to take any turn or straight on a track. In order to estimate this effectively and differentiate it from a viable method, we are going to ignore some constraints that we would normally adhere to. The most important of those being maneuverability and frames. In this lower bound, we assume not only that our vehicle can turn any angle instantly, but we also assume that it can do it faster than a frame. What this means is that our vehicle will always be as tight to the turn as physically possible.

Immediately there are many different possibilities for how to start this task. We chose to begin by determining a path sequentially, starting with the beginning straight away and the first turn. However, due to the nature of this game, should we make one mistake early on and realize it later, we would have to go back and redo all of the calculations from that point onwards since relative positioning and velocity for our vehicle will be permanently changed. With that being said, lets begin by analyzing the very beginning of the race.

3.2 Section 1: The First Alignment and Straight

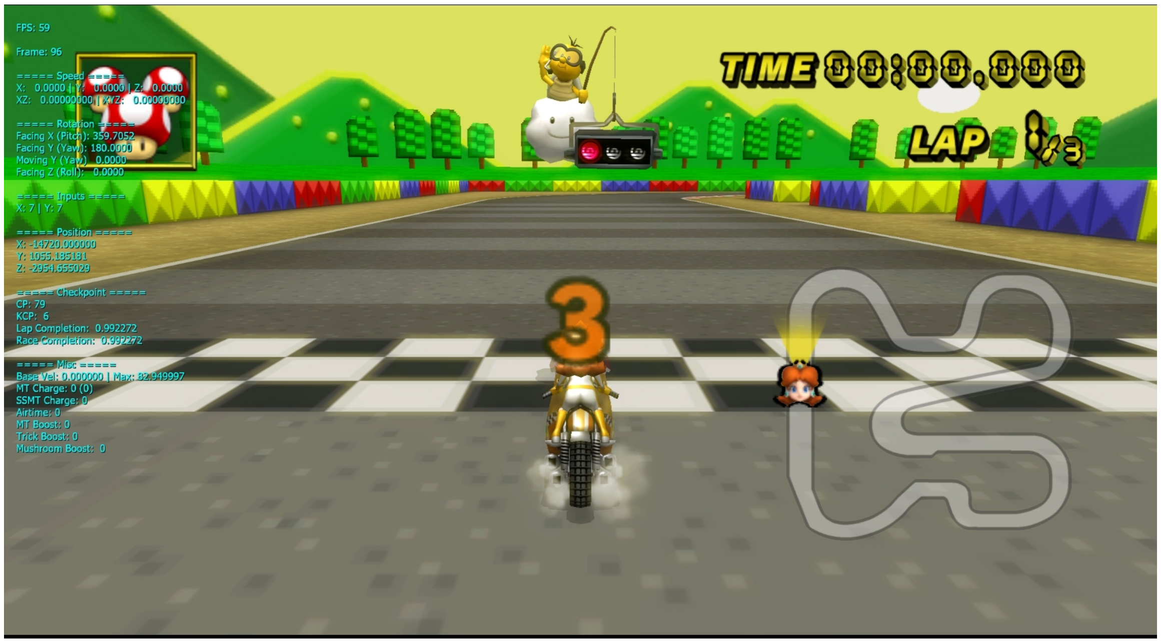
3.2.1 Starting Position

As is defined with all paths, we must have a start and end point. While the end point will be anywhere along the finish line that crosses the checkpoint, the start point is stationary and we are to assume it cannot be changed. While it is technically possible to change this start point through a technique called a start slide, we are choosing to forgo that for now as positioning along with rotation will be altered in a way that is very hard to quantify. Furthermore, the character and vehicle combination we are using (Daisy on the Mach Bike) benefits almost nothing from a start slide (calculations have put it in the past to be less that 2 milliseconds), so if we so decide we can just remove 2 milliseconds from our final time as it is a lower bound after all.

Alas, we must find out what our exact starting position is. Using the emulator and a very helpful position code, we can see determine our exact starting coordinates.

As seen in figure 3.1, we can see our starting position on the course is approximately $(-14720, 1055.2, -2954.7)$. It is important to note that for whatever reason the vertical access is displayed as the y-axis in our game, not the commonly used z-axis. When properly adjusted, this coordinate corresponds to the point in the modeling software Blender shown in figure 3.2. Keep this in mind for all future references.

Not only does this make logical sense, but it also properly corresponds to the image we received the coordinates from. This means that we have an effective way of converting our vehicle's position within the game to a position on the model within Blender. With this now verified, we choose to assume all coordinates we



```
FPS: 59

Frame: 96

===== Speed =====
X:   0.0000 | Y:   0.0000 | Z:   0.0000
XZ:   0.00000000 | XYZ:   0.00000000

===== Rotation =====
Facing X (Pitch): 359.7052
Facing Y (Yaw): 180.0000
Moving Y (Yaw)   0.0000
Facing Z (Roll):   0.0000

===== Inputs =====
X: 7 | Y: 7

===== Position =====
X: -14720.000000
Y: 1055.185181
Z: -2954.655029

===== Checkpoint =====
CP: 79
KCP:  6
Lap Completion:  0.992272
Race Completion:  0.992272

===== Misc =====
Base Vel: 0.000000 | Max: 82.949997
MT Charge: 0 (0)
SSMT Charge: 0
Airtime: 0
MT Boost: 0
Trick Boost: 0
Mushroom Boost:  0
```

Figure 3.1: Starting position and its coordinates

receive from the game are legitimate and can be converted into our model, as well as from our model into the game. We can also verify this assumption with a fair bit of other software such as Lorenzi's KMP editor which also gave us coordinates on the same system when placing KMP objects in the model (such as pipes)

3.2.2 Starting Alignment

However, our next is the task is much more challenging than the first. We need to determine how we align, or in layman's terms angle ourselves at the beginning. Our main goal of this is to hop over as much offroad as possible. This is because logically we want to start the turn as far to the right as possible, as it makes the total distance for the turn shorter. As seen in the figure 3.3, there are 2 alignments we can immediately analyze. The first of which is the line on the right. While it certainly would result in a shorter total path for the first turn, it also goes above the offroad which our vehicle is not allowed to touch. If we can verify our vehicle can hop over

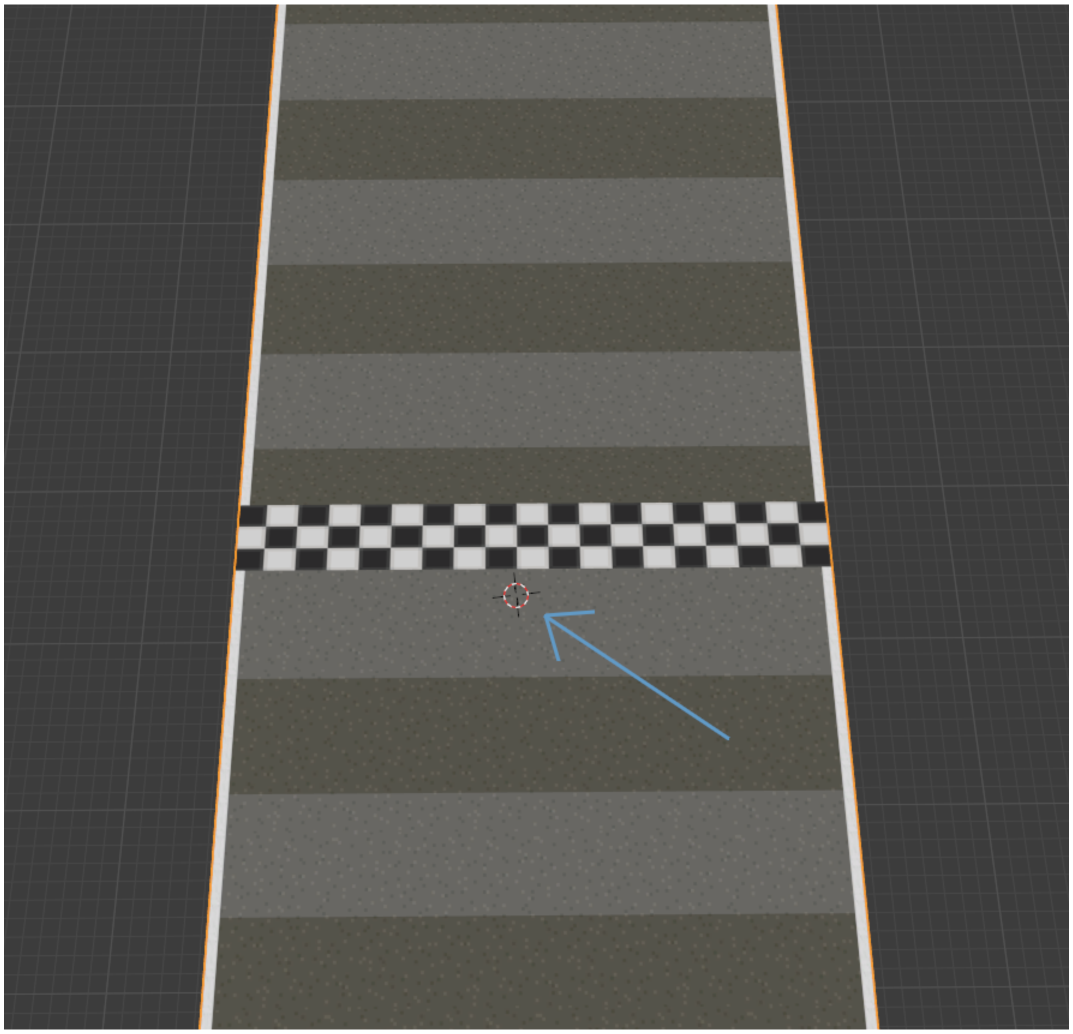


Figure 3.2: Starting position within Blender

the offroad at the start of its drift, it would lead to a much faster turn and shorter path than our secondary possibility, which just drives around the offroad.

There are several ways we can test this, one of which is to find exactly how many frames a hop allows our vehicle to stay in air. While this would help, it would be more difficult to calculate a total distance traveled in air due to the vehicle's deceleration. The method we settled on was to just record the exact distance a vehicle travels from when it starts the hop until it lands. Once we determine this value, if it is larger than or equal to the distance our vehicle needs to travel over the offroad, the right most path is our optimal alignment. However, if it is too short

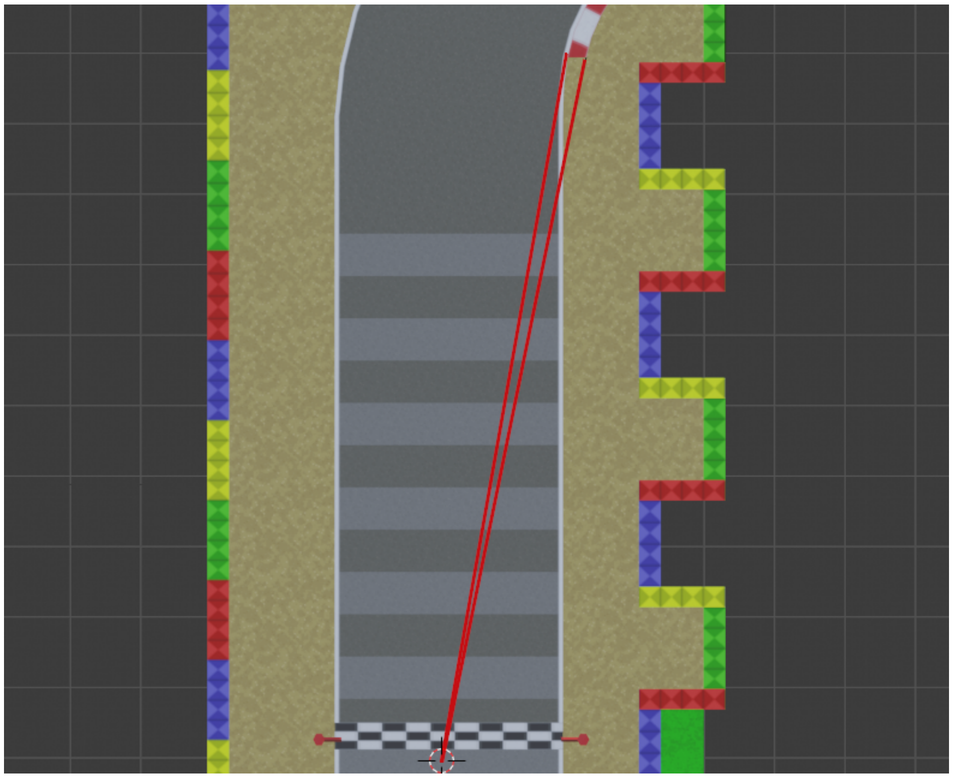


Figure 3.3: Possible first alignments. Red lines represent possible optimal trajectories for our vehicle

then we can calculate exactly how much offroad we can cut off with the hop, and plan a route accordingly which cuts off as much offroad as possible.

We measure our distance by aligning our vehicle towards the turn at full speed in a wheelie, and then initiating our drift. We do not immediately hold a direction down to maximize airtime, and due to the constraints we removed in order to calculate this lower bound this is perfectly viable for determining the distance traveled. Once this was setup, we recorded the hop and then looked at the frame the vehicle began to hop, and the frame on which it landed, and compared coordinates. These reference images can be seen in the figure 3.4 and 3.5 below.

From this, we can determine our starting coordinate is $(-13033, -11875)$ and our landing coordinate is $(-12611, -13158)$. Using a simple distance formula, we can calculate:

$$D^2 = (-13033 - (-12611))^2 + (-11875 - (-13158))^2$$

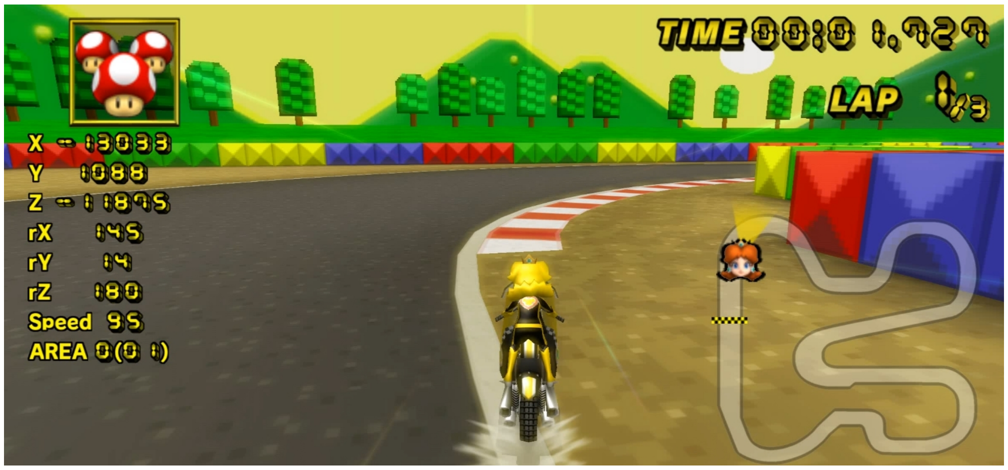


Figure 3.4: Frame 1 of hop

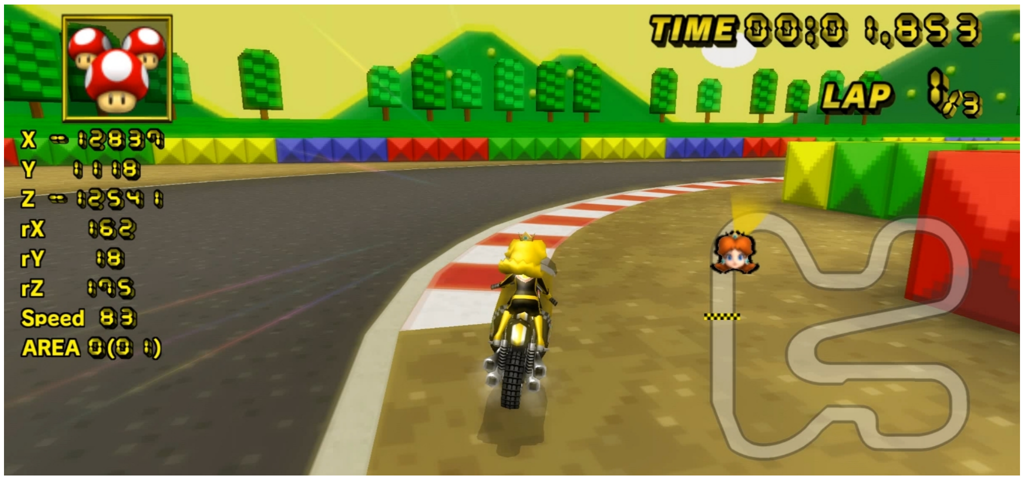


Figure 3.5: Frame 1 of landing from hop

$$D^2 = 422^2 + 1283^2$$

$$D^2 = 1824173$$

$$D = 1350.6$$

We have now determined our vehicle can travel 1350.6 units in the air from start to finish. If we are to determine if this is enough, we have to model our path in Blender and determine if the line segment over the offroad is less than or equal to 1350.6.

According to our calculations in Blender, we can determine this cut over the

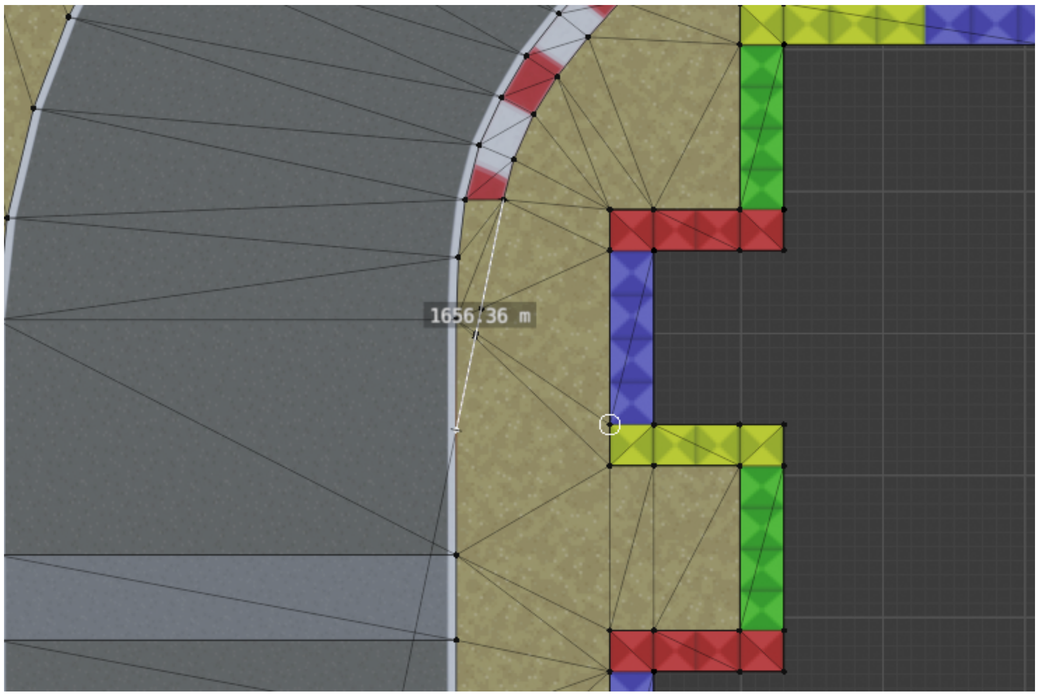


Figure 3.6: Length of offroad in Blender

offroad is 1656.4 units. Unfortunately it appears that the distance is a bit too long for us to hop over effectively. This means we are going to have to find the optimal path to cover exactly 1350.6 units of offroad with our hop. We can do this using trigonometry. Using Blender to construe a right angle triangle, using a similar method as above we can determine exactly where our path goes over the offroad.

While this figure may look correct, it is actually making an incorrect assumption about the length of the opposite side to our unknown angle. We do not know its exact length. Furthermore, due to the way this problem was setup, the length of our adjacent angle will also be changing when we try to determine an appropriate solution. A proper overview of the problem would actually look something like this:

$$tan(x) = \frac{1713.92}{9987.9 - z}$$
$$cos(x) = \frac{z}{1350.6}$$

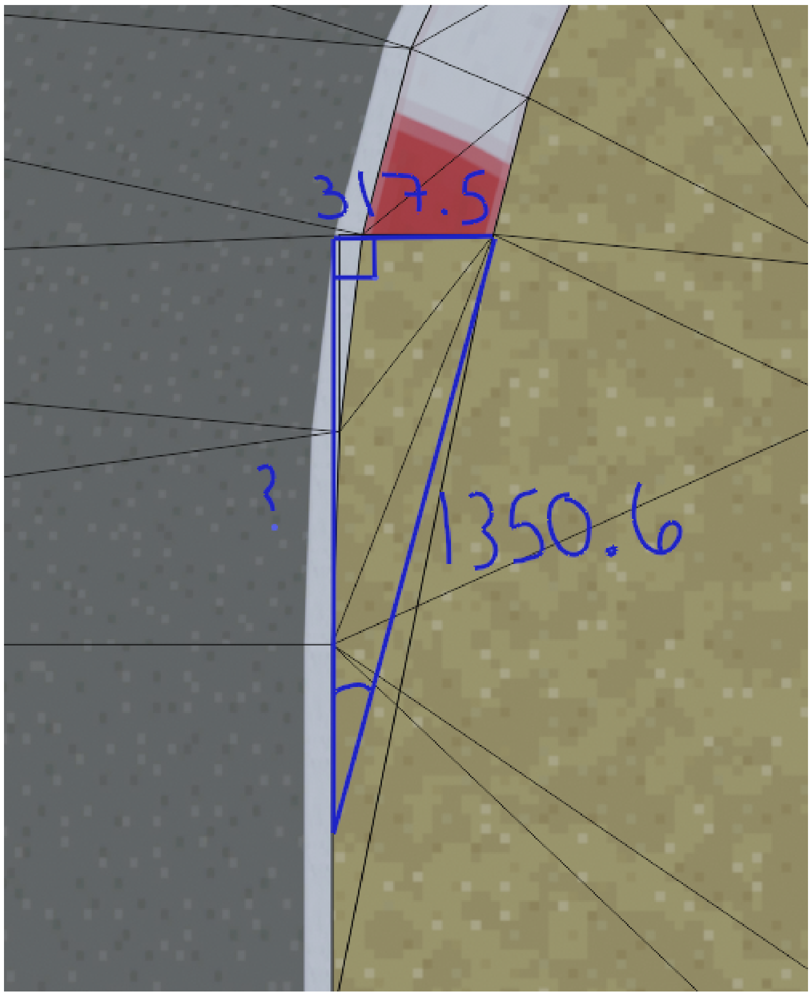


Figure 3.7: Incorrect conditions to make system of equations

Unfortunately for us this is a really nasty system, but with a little of computing power we are able to encounter 2 possible solutions:

$$x = 6.28319n + 0.195322$$

$$z = 1350.6 * \cos(6.28319n + 0.195322)$$

along with

$$x = 6.28319n - 2.99137$$

$$z = 1350.6 * \cos(2.99137 - 6.28319n)$$

From these two equations, we notices something interesting about the last term in each sequence. Each of these terms are exactly π apart. Hence we can simply use

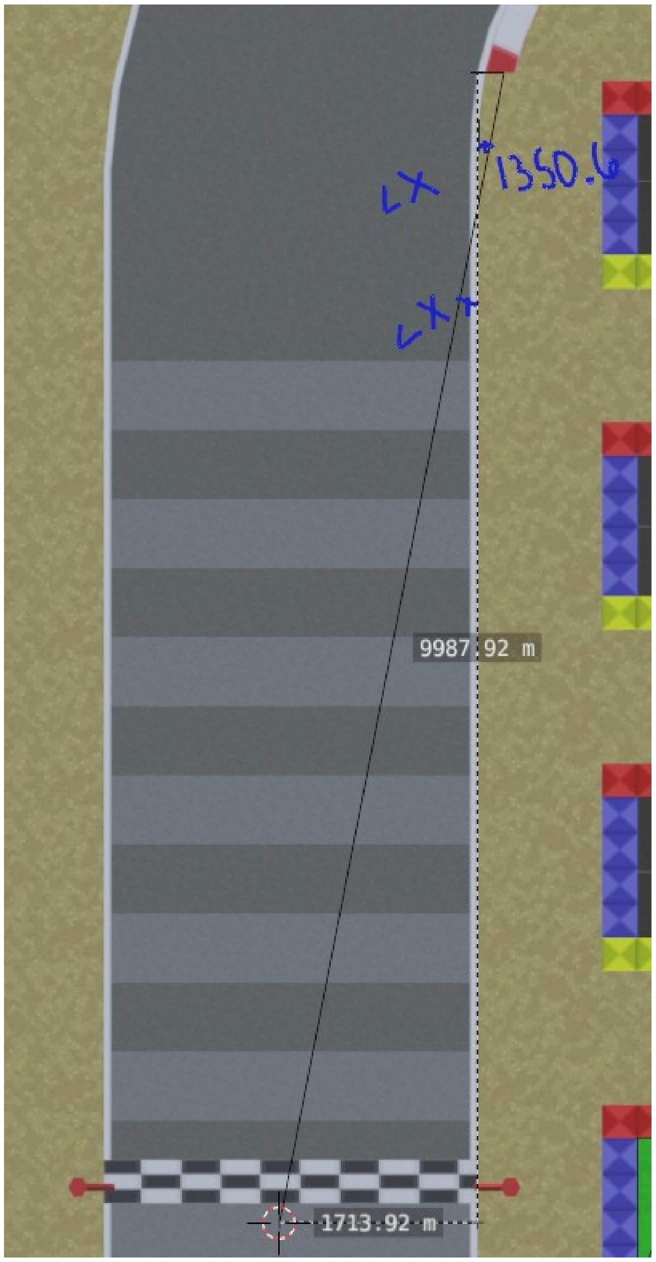


Figure 3.8: Proper conditions to make system of equations

the first equation and add a $+2\pi k$ to each term (albeit within the cosine term in the second). We can then attempt to find a solution for this problem.

Let y be denoted as the angle between the horizontal base edge of length 1713.92 and our eventual path. We can determine from this that $y = 90 - x$ due to the right angle presented in the only non-variable angle in the triangle. Ultimately solving for y will give us our optimal starting alignment.

We can begin by letting $n = 0$, in turn letting $x = .195$. From here, we know that $z = 1350.6 * cos(0.195) = 1324.919$. This value then gives us a point where our path will intersect with the vertical line of length 9987.92. We can then go back into Blender and test this value and see how our hypotenuse approaches 1350.6.

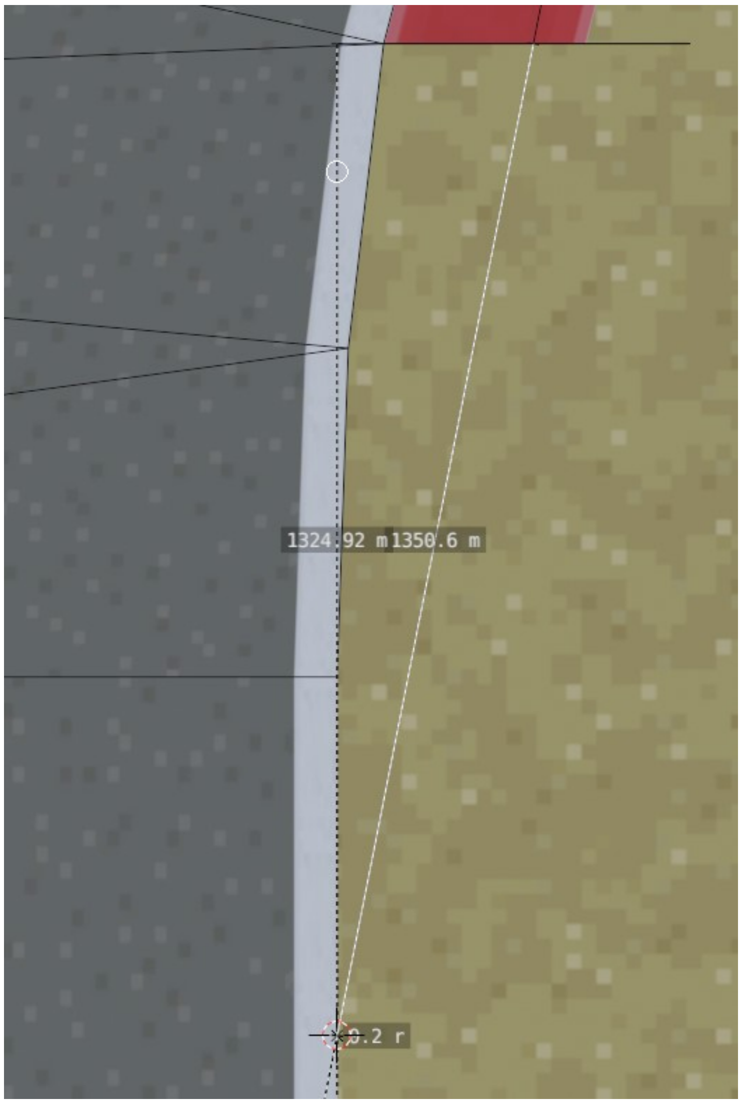


Figure 3.9: Successful angle for the first alignment

As you can see, our hypotenuse is equivalent to exactly 1350.6, which is the maximum distance our vehicle can travel over the offroad. This allows us to conclude that the best angle for our character to align at the beginning is 11.173 degrees to the right.

3.2.3 Calculating Duration of First Straight

Now we need to calculate exactly how long it takes for us to reach the edge of that offroad before we start our first turn. To do this, we can start by calculating our distance that we need to travel and make the assumption that it is fastest to go in a straight line. We could do this using trigonometry again given we have all the values to calculate it, but a much simpler way is using our model in Blender that we already have.

From this we can see our distance for the first straight is 8830.91, which allows us to then calculate how fast our vehicle will travel. It is important to note that our rate of travel will be non-linear, as there are multiple factors at play here for acceleration and deceleration. This sets us up well to use integration and derivation to determine exactly how long this travel will take.

Referencing figure 2.5: *Different start boosts and corresponding timing*, we see the different amounts and length of acceleration present from our start boosts. Lets assume that it is more optimal to get the boost for the maximum length, as getting it for any shorter period of time will result in less speed. This means we will be using the values in the 6th row. Our boost length is 70 frames, which we can translate into 1.167 seconds of acceleration.

As for how much acceleration, we can use figure 2.1: *All Acceleration types and quantities* to see exactly how strong our acceleration is. The start boost is classified as a miniturbo boost, which means we accelerate at $3\frac{u}{f^2}$. Combining this means for the first 1.167 seconds, our vehicle accelerates at a rate of $3\frac{u}{f^2}$.

While it was abbreviated in figure 2.1, the actual maximum speed of our vehicle is $111.983\frac{u}{f}$. From this, we can determine that it only takes 37.326 frames, or 0.622

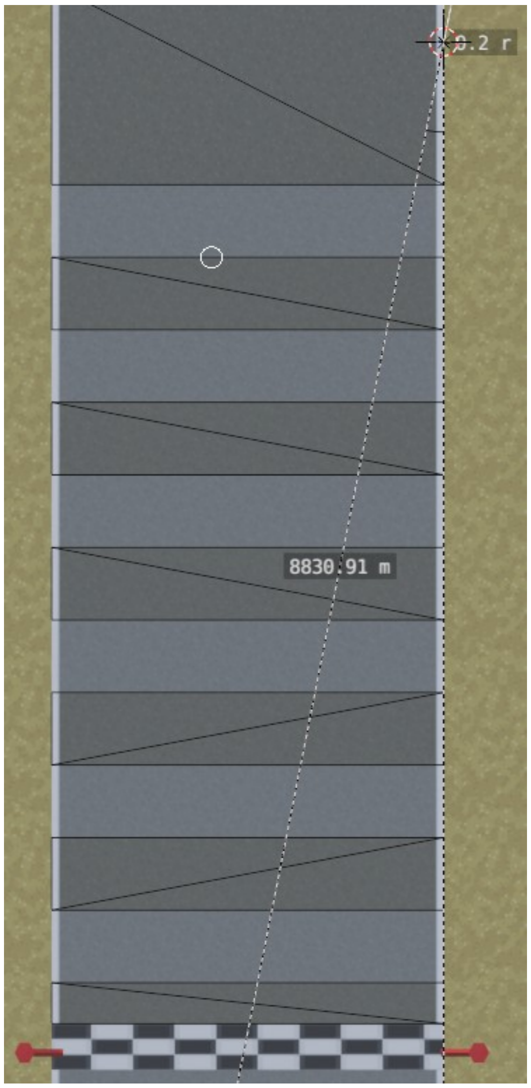


Figure 3.10: Exact length of the first straight

seconds for our vehicle to accelerate from $0\frac{u}{f}$ to its maximum boost speed of $111.98\frac{u}{f}$. Following the first 37.326 frames, we have 32.674 more frames remaining in our boost which implies our vehicle will travel at $111.98\frac{u}{f}$ for another 0.545 seconds.

However, once that time ends, our vehicle does not continue to stay traveling at $111.98\frac{u}{f}$. It begins to decelerate at a rate of $3\frac{u}{f^2}$ down to our maximum non-boost speed of $95.393\frac{u}{f}$. Once again using arithmetic we know this will take 16.590 frames or 0.277 seconds.

After that deceleration, we do not need to worry about any more changes in speed for the first straight as we will just be in a wheelie until the first turn starts. However, in order to calculate this distance we first need to determine exactly how far our vehicle travels during the aforementioned periods of acceleration. We can do this using integration.

Lets begin by creating an equation for modeling our distance D_1 given our acceleration and time t_1. We can do this by double integrating our acceleration of 3 $\frac{u}{f}$.

$$D_1 = \iint_0^{t_1} 3\,[dt_1]^2$$
$$D_1 = \int_0^{t_1} 3t_1\,dt_1$$
$$D_1 = \tfrac{3}{2}t_1^2$$
$$t_1 = 37.326$$
$$D_1 = 2089.845 \text{ units}$$

For the next stage, it is actually quite simple. Our vehicle travels a constant velocity over a set time, so to calculate D_2 we just have to multiply them together.

$$D_2 = 111.98t_2$$
$$t_2 = 32.674$$
$$D_2 = 3658.835 \text{ units}$$

As for the distance traveled during deceleration D_3, we have to do integration similar to how we calculated D_1 except we actually have a non-zero starting speed at $t_3 = 0$. We can start by making an equation for the velocity v_3 of our vehicle, and then integrate it to get distance traveled.

$$v_3 = 111.98 - 3t$$
$$D_3 = \int_0^{t_3} v_3\,dt_3$$
$$D_3 = \int_0^{t_3} 111.98 - 3t_3\,dt_3$$
$$D_3 = 111.98t_3 - \tfrac{3}{2}t_3^2$$

$$t_3 = 16.590$$
$$D_3 = 1857.748 - 412.842$$
$$D_3 = 1444.906 \text{ units}$$

To get our total distance remaining D_4, we can sum together all distances already traveled and subtract that value from the total distance our vehicle needs to travel before the first turn.

$$D_4 = D - (D_1 + D_2 + D_3)$$
$$D_4 = 8830.910 - (2089.845 + 3658.835 + 1444.906)$$
$$D_4 = 1637.324 \text{ units}$$

Now that we have our final distance and have established our vehicle will travel at a fixed velocity of $95.39\frac{u}{f}$, we can calculate how much time (t_4) our vehicle travels in this segment by using basic arithmetic.

$$D_4 = 95.39t_4$$
$$1637.324 = 95.39t_4$$
$$t_4 = 17.165 \text{ frames}$$

We actually have 2 final segments to calculate before we can say we finished calculating the entire first straight. We need to determine how long the hop over the offroad takes. Our fifth segment will be when the vehicle decelerates, and the sixth will be when it is going a constant speed.

To calculate our deceleration we need to take some new values into account. The maximum speed for our vehicle when not in a wheelie or boost is 82.950, and the deceleration is $3\frac{u}{f}$. Using these values and arithmetic we can determine it decelerates for 4.148 frames or 0.0691 seconds. As for our sixth segment, we will have to calculate it similarly to how we calculated the fourth segment above, where calculating the previous segment distances will determine our remaining distance and then simple arithmetic will determine how much time it takes to travel the final distance at a constant speed.

Here is how we can calculate the beginning deceleration.

$$v_5 = 95.393 - 3t$$
$$D_5 = \int_0^{t_5} v_5 \, dt_5$$
$$D_5 = \int_0^{t_3} 95.393 - 3t_5 \, dt_5$$
$$D_5 = 95.393t_5 - \tfrac{3}{2}t_5^2$$
$$t_5 = 4.148$$
$$D_5 = 395.690 - 25.809$$
$$D_5 = 369.881 \text{ units}$$

As before, we can now determine the remaining distance to be 980.719u. Using this value and our velocity v_6, we can solve for the amount of time it takes to complete the hop.

$$D_6 = v_6 t_6$$
$$980.719 = 82.950t_6$$
$$t_6 = 11.823 \text{ frames}$$

To find our total time traveled t on the first straight, we just simply have to sum all our times together for each segment (t_{1-6}).

$$t = t_1 + t_2 + t_3 + t_4 + t_5 + t_6$$
$$t = 37.326 + 32.674 + 16.590 + 17.165 + 4.148 + 11.823$$
$$t = 119.726 \text{ frames}$$
$$\text{or}$$
$$t = 1.995 \text{ seconds}$$

With that, we not only have our optimal route for the first straight but calculations for how long it would take to travel as well. Our next step is to calculate how the path for our first turn.

3.3 Optimizing the first turn on the track

Now that we have an actual point of entry here into the turn, we can start to think of how exactly we want to take the turn. Immediately, we have several questions we have to ask about the turn and how we want to take it. To simplify that, we will put them into a list and answer each of them one at a time.

- Are there any obstacles we need to avoid?
- How many miniturbos can we get on this turn?
- At what point do we want to release our miniturbo(s)?
- Where do we want to align out of the miniturbo(s)?

While this may not seem particularly exhaustive, it does give us a good start for determining an overall picture for how we want to take this turn. Lets try to answer each of these questions, then calculate an optimal route.

3.3.1 Are there any obstacles we need to avoid?

Unfortunately on this turn there is a particularly annoying obstacle: the pipe. At the end of the turn, there is a pipe ever so perilously placed in the offroad. While this may not seem like an issue, it actually presents quite the conundrum when verifying it with our hitbox. Since when our character drifts the hitbox rotates (or rolls for those more technically inclined) in the direction it drifts. That means while our vehicle may be driving on the road, it actually is leaning over the offroad. As you can imagine, this means our vehicle may actually collide with the pipe while it is on the main road! This means that should we attempt to go close enough to that pipe in any optimal route, we will need to determine if any collision would occur and adjust our path accordingly.

3.3.2 How many miniturbos can we get on this turn?

This question is actually quite nuanced, but to start we should determine exactly how these miniturbos are created. Our vehicle travels in the path of an arc, and this arc can very in length and shape depending on the controller inputs. figure 2.5 *Diagram of the joystick and its input range* gives a good idea for exactly what inputs can be used, but may not illustrate how these arcs the vehicles travel vary. One such estimated representation can be seen in the figure below in context of the first turn.

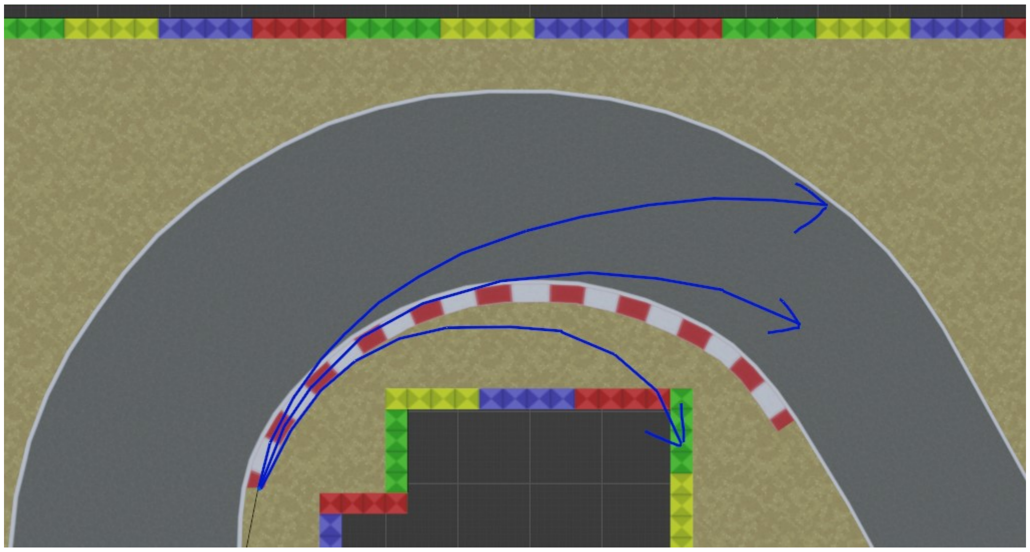


Figure 3.11: Estimations of different arcs from drifting

For example, the topmost line is an example for what would happen if the stick was permanently held the opposite direction of the drift, in this case left. Not only does it take the character very wide, but as discussed earlier in Section 2.3.4 *Drifting and Mini-turbos*, it also charges our miniturbo very inefficiently. This means that it would take much more time to charge than if the stick was held farther to the right. We can see an example of holding the stick ever so slightly to the right as the middle line in the above figure. Not only does it charge the miniturbo at an optimal rate, but it also is a significantly shorter path than the top arc. However, it does struggle to stay as tight as possible, which is where our other arc comes in to play. It drives significantly tighter to the edge of the road, while also charging the miniturbo at the same optimal rate that the middle arc does. However, it goes very tight to the turn and ultimately drives into the offroad which is certainly much slower.

A combination of all these different arcs will be used, and there is no restriction for holding the same stick position on the drift. For example, you can start the turn holding your stick in the opposite direction and then afterwards hold it to the other side. Something we can use to quantify this however is not only the distance traveled during these drifts to give a possible range, but also the difference in alignment out of them. To do this we can test it in the game to get the positional data, then perform analysis.

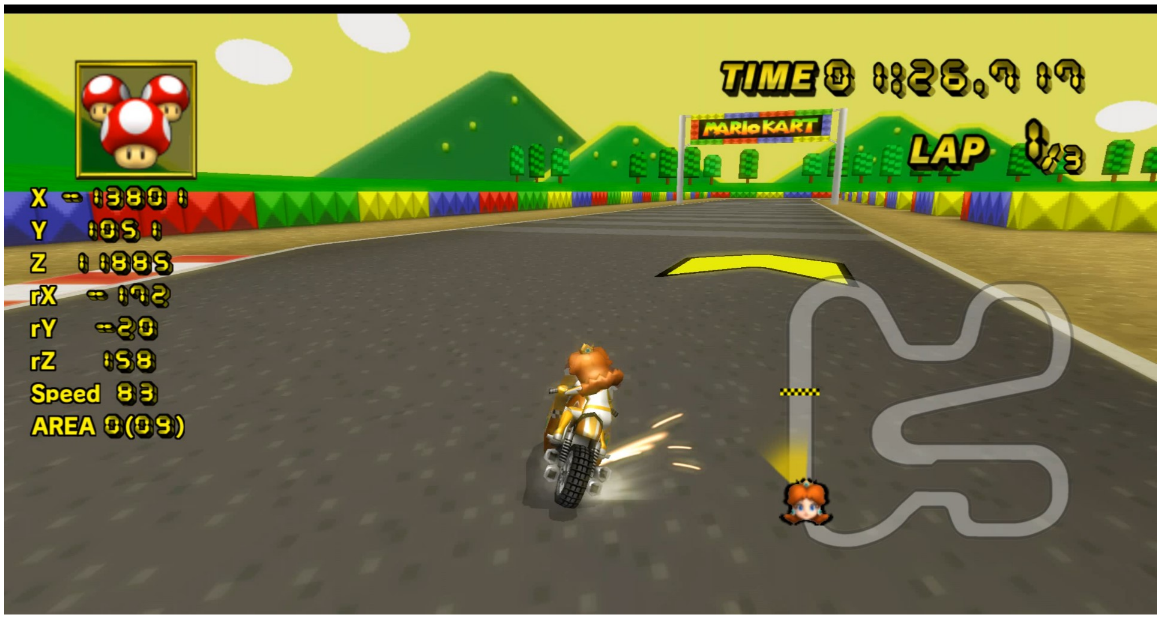
3.3.3 Calculating Angular Rotation of a "Fail Drift"

While we are going to do the process with the 3 different drift types (stick positions at -7, 3, and 7) we will start by calculating when the stick is in the opposite direction. This means if the vehicle is turning right, the stick has an x-value of -7, and if the vehicle is turning left it has an x-value of 7. Since we are keeping the stick position the same (it is constant), we can assume that the arc representative of our vehicle's path is smooth and continuous.

3.3.3.1 Attempt 1: Calculating angular rotation using 3 consecutive coordinate pairs

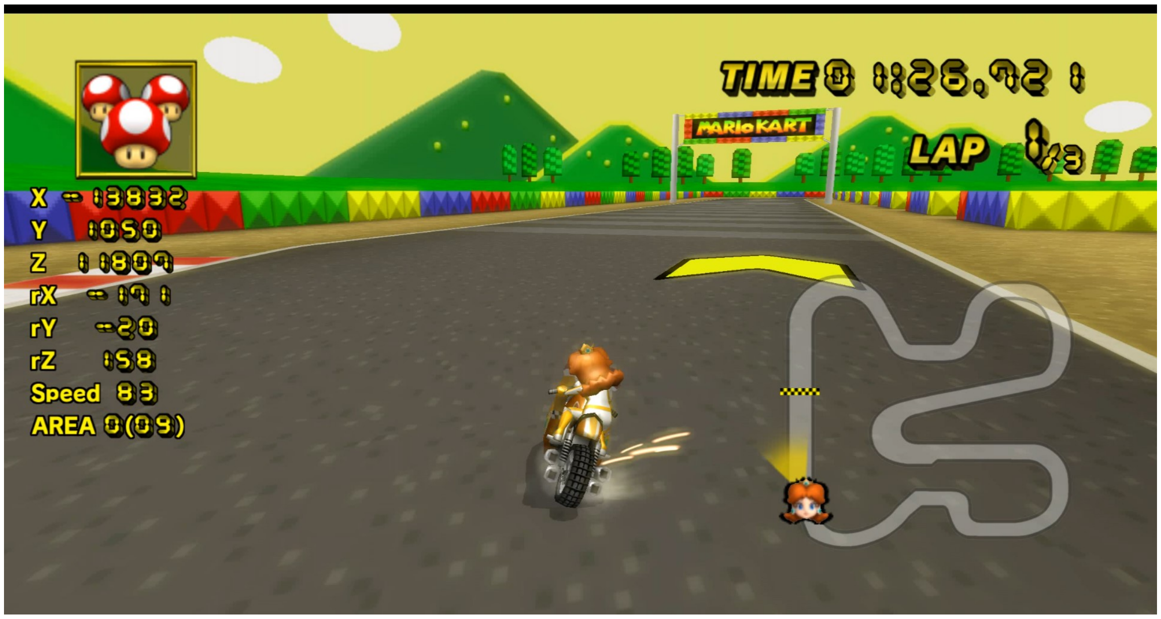
With this assumption, all we need is 3 coordinate pairs from 3 consecutive frames to calculate change in angle. From this, we can use the miniturbo charge values to determine how many frames the stick would have to be held in that position for the miniturbo to charge, and that gives us our arc distance. Here are the 3 images:

From these 3 images, we now have 3 coordinates. We will denote them as follows:



```
X     -13801
Y      1051
Z     11885
rX     -172
rY      -20
rZ      158
Speed    83
AREA 0(09)
```

Figure 3.12: Frame 1 of -7 stick position



```
X     -13832
Y      1050
Z     11807
rX     -171
rY      -20
rZ      158
Speed    83
AREA 0(09)
```

Figure 3.13: Frame 2 of -7 stick position

$$p_n = (x, y)$$

$$p_1 = (-13801, 11885)$$

$$p_2 = (-13832, 11807)$$

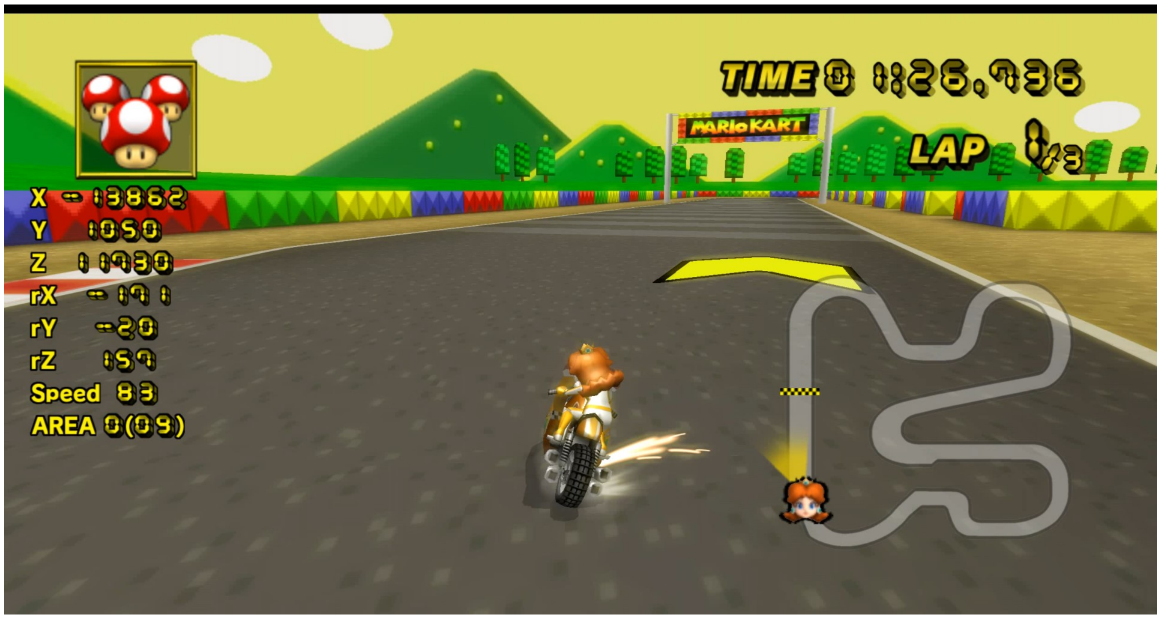


Figure 3.14: Frame 3 of -7 stick position

$$p_3 = (-13862, 11730)$$

Now lets verify these are correct by calculating the distance between each. The distance should be equivalent to how much the vehicle travels for 1 frame traveling at its fastest non-boost/wheelie speed, or 82.950 units.

$$\begin{aligned}
D_1 &= \sqrt{(p_{1_x} - p_{2_x})^2 + (p_{1_y} - p_{2_y})^2} \\
D_1 &= \sqrt{7045} \\
D_1 &= 83.934 \\
D_2 &= \sqrt{(p_{2_x} - p_{3_x})^2 + (p_{2_y} - p_{3_y})^2} \\
D_2 &= \sqrt{7045} \\
D_2 &= 82.638
\end{aligned}$$

While it is true that $D_1 \neq D_2$, we can ultimately chalk this off to precision error. This is because we can only record positioning to the nearest whole unit, which means we will have some slight error. In light of this, it would be poor practice to use this data due to the relative margin of error. Instead, we can model our arc in a different manner using distance covered and starting/ending positions.

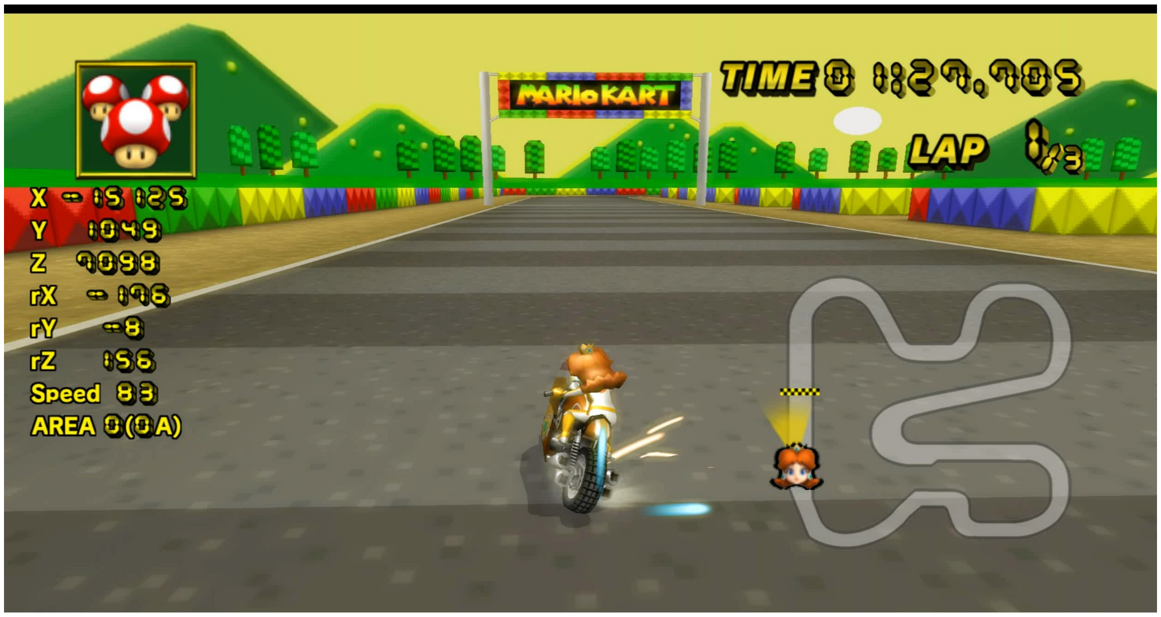
3.3.3.2 Attempt 2: Calculating angular rotation using 3 distant coordinate pairs

It is important to note that due to floating point values, there still will be some level of error present in these calculations. That being acknowledged, it would be substantially less than the above method. In this example where we drift to the right, fail drifting by holding our stick at the -7 position charges a miniturbo at 2 units of charge per frame. We also know we can use the miniturbo at 270 units of charge (While technically we can only use it at 271, for simplicity sake we will use 270). This means we would be holding the drift for 135 frames, or 2.250 seconds. Since we know we are traveling at a velocity of $82.950\frac{u}{f}$, we can calculate the length L_1 of our arc as:

$$L = (135)(82.950)$$
$$L = 11198.250$$

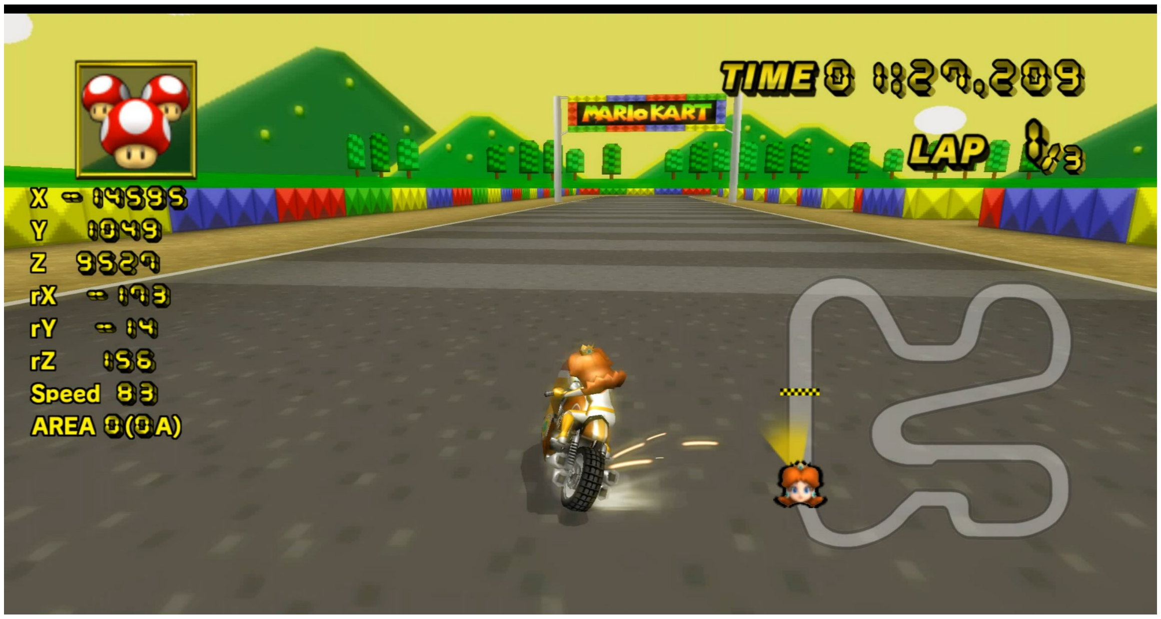
From here, we can then find the starting/ending point. While there will still be error in these values, it will be significantly less than our other method as this method does not propagate the error repeatedly. On a minor note, due to some bounces the drift in the game actually does not start entirely smoothly, so we will be positioning our starting point on the 7th frame after landing. This means we can still use figure 3.12 as our starting position. As for our new ending position, we can use figure 3.15.

We also need a midpoint to determine the height of this arc. Since our velocity relative to our position and rotation remains constant, we can determine the apex of this arc is in the middle of travel, or on frame 31. Hence, we collect more data for frame 31.



X -15125
Y 1049
Z 7098
rX -176
rY -8
rZ 156
Speed 83
AREA 0(0A)

Figure 3.15: Frame 61 of -7 stick position



X -14595
Y 1049
Z 9527
rX -173
rY -14
rZ 156
Speed 83
AREA 0(0A)

Figure 3.16: Frame 31 of -7 stick position

Now we can start the math. We can start by finding the Cartesian distance between our end points.

$$p_1 = (-13801, 11885) \text{ our starting point}$$

$$p_2 = (-14595, 9527)$$

$$p_3 = (-15125, 7098) \text{ our ending point}$$

$$C = \sqrt{(P_{1_x} - P_{2_x})^2 + (P_{1_y} - P_{2_y})^2}$$

$$C = \sqrt{24668345}$$

$$C = 4966.724$$

We ultimately need to calculate the height of the arc above this line segment. One such way we could do this is to make an equation of a line that represents our line segment, then make a line perpendicular to it that runs through the apex of the arc. If we determine the length then of that line segment, we determine the height of the arc and ultimately the angle. For starters, let us make an equation.

$$m = \frac{P_{1_z} - P_{2_z}}{P_{1_x} - P_{2_x}}$$

$$m = \frac{-4787}{-1324}$$

$$m = 3.616$$

$$y - P_{1_z} = m(x - P_{1_x})$$

$$y - 11885 = 3.616(x - [-13801])$$

$$y = 3.616x + 61783.329$$

Now we want to make a perpendicular line to it, and eventually determine how long it is between intersecting that line and the arc.

$$m_p = -\frac{1}{m} = -\frac{1}{3.616} = -0.277$$

$$P = (-14595, 9527)$$

$$y - 9527 = -0.277(x - [-14595])$$

$$y = -0.277x + 5490.772$$

Using both of our new equations we can find where they intersect using a basic system of equations.

$$y = 3.616x + 61783.329$$
$$y = -0.277x + 5490.772$$
$$0 = 3.893x + 56292.557$$
$$x = -14459.942$$
$$y = 3.616[-14459.942] + 61783.329$$
$$y = 9496.176$$
$$p_i = (-14459.942, 9496.176)$$

Our last step is simply to calculate the final Cartesian Distance between the apex of the arc and the point we just calculated.

$$D = \sqrt{(p_{2_x} - p_{i_x})^2 + (p_{2_y} - p_{i_y})^2}$$
$$D = \sqrt{(-14595 - [-14459.942])^2 + (9527 - 9496.176)^2}$$
$$D = \sqrt{(-14595 - [-14459.942])^2 + (9527 - 9496.176)^2}$$
$$D = \sqrt{[-135.058]^2 + 30.824^2}$$
$$D = \sqrt{19190.782}$$
$$D = 138.531$$

From here we can actually set up a system of equations to determine the radius of the circle our arc is on. This will ultimately help us represent the path of the arc mathematically. Below is a visual representation of our system of equations.

Immediately we note that our radius is already represented by the combination of two line segments, x and D. This gives us a relatively simple equation for two unknowns, x and our radius r. It may seem difficult to setup another equation, but there is actually a pretty elegant solution for this. Fortunately we already know the length of the chord C, which we can use in our final equation. We can form a right triangle with our line segment x which happens to be perpendicular to our chord C. From this we can set up our right triangle with legs of x and $\frac{C}{2}$ and a hypotenuse of r. Since $\frac{C}{2}$ is a known constant, this equation also has the same two unknowns of our previous equation, which means we have a valid system! We can represent this system as:

$$r = n + D$$
$$r^2 = n^2 + (\tfrac{C}{2})^2$$

Now we can begin solving this system to determine our radius r.

$$r = n + D$$
$$r^2 = n^2 + (\tfrac{C}{2})^2$$
let us square the first equation
$$r^2 = n^2 + 2Dn + D^2$$
Use elimination to cancel out the r^2 terms
$$r^2 - r^2 = n^2 - n^2 + (\tfrac{C}{2})^2 - 2Dn - D^2$$
$$0 = (\tfrac{C}{2})^2 - 2Dn - D^2$$
Substitute in our constant values
$$0 = (6167086.823) - 2(138.531)n - 19190.838$$
$$0 = 6147895.985 - 277.062n$$
$$n = 22189.604$$
Substitute back in to solve for r
$$r = n + D$$
$$r = 22189.604 + 138.531$$
$$r = 22328.135$$

And with that we finally have calculated our radius. Using this value, our next step is to try and create an equation for our circle, so we need to find the center of this circle. To do this we can add our radius r to our point p_2 in the direction of p_i.

Convert x into a vector
$$m_p = -0.277$$

$$r = 22328.135$$

Let our sides be of length a and $3.610a$

$$22328.135 = \sqrt{a^2 + (3.610a)^2}$$

$$22328.135 = \sqrt{14.033a^2}$$

$$22328.135 = 3.746a$$

$$a = 5960.528$$

This means our sides are of length -5960.528 and 21517.506

Combine to create our circle center $C_{x,y}$

$$c_{x,y} = p_2 + \hat{x}$$

$$c_{x,y} = (-14595 + 21517.506, 9527 + [-5960.528])$$

$$c_{x,y} = (6922.506, 3566.472)$$

Now we have our circle center, we can finally create the equation for our circle so we can begin to solve for curvature.

$$r^2 = (x - x_1)^2 + (y - y_1)^2$$

$$22328.135^2 = (x - 6922.506)^2 + (y - 3566.472)^2$$

Our final equation for the circle in Cartesian form can be written as:

$$498545612.578 = (x - 6922.506)^2 + (y - 3566.472)^2$$

While technically speaking it is not necessary to solve for the equation of this circle, it helps a lot in illustrating the larger picture for the reader. From here we just need to calculate how our vehicle's angle changes over the 60 frames it traveled. Since we know a circle is 360 degrees, and our circle is uniformly round, we can conclude that the angle of rotation is proportional (in fact it is identical) to our vehicle's rotation in its path. Therefore we can just find our angle of rotation to determine the vehicle's rotation.

Let our angle of rotation be $\angle a$, then using the triangle we used earlier to calculate the distance from the chord to the circle's center (it solved for the variable x), we can solve for $\angle\frac{a}{2}$.

$$\cos(\tfrac{\angle a}{2}) = \tfrac{n}{r}$$

$$\cos(\tfrac{\angle a}{2}) = \tfrac{22189.604}{22328.135}$$

$$\cos(\tfrac{\angle a}{2}) = 0.993796$$

$$\tfrac{\angle a}{2} = 6.38571729907$$

$$\angle a = 12.771 \text{ degrees}$$

Finally we have managed to calculate the angle our vehicle turns over 60 frames. However, we want to find how much is rotates each frame for our future calculations. Using basic arithmetic, we know that this value $\angle r = 0.213$ degrees per frame.

While we have calculated this now for a specific position (holding it the opposite direction of the drift), we still need to calculate this value for soft drifting and hard drifting. Since we have already demonstrated our method for calculating this value, we will forgo much of the explanatory text in between the calculations.

3.3.4 Calculating Angular Rotation for "Hard Drift"

We can once again start by displaying our data points for the hard drift calculations.

From these images we get the following points:

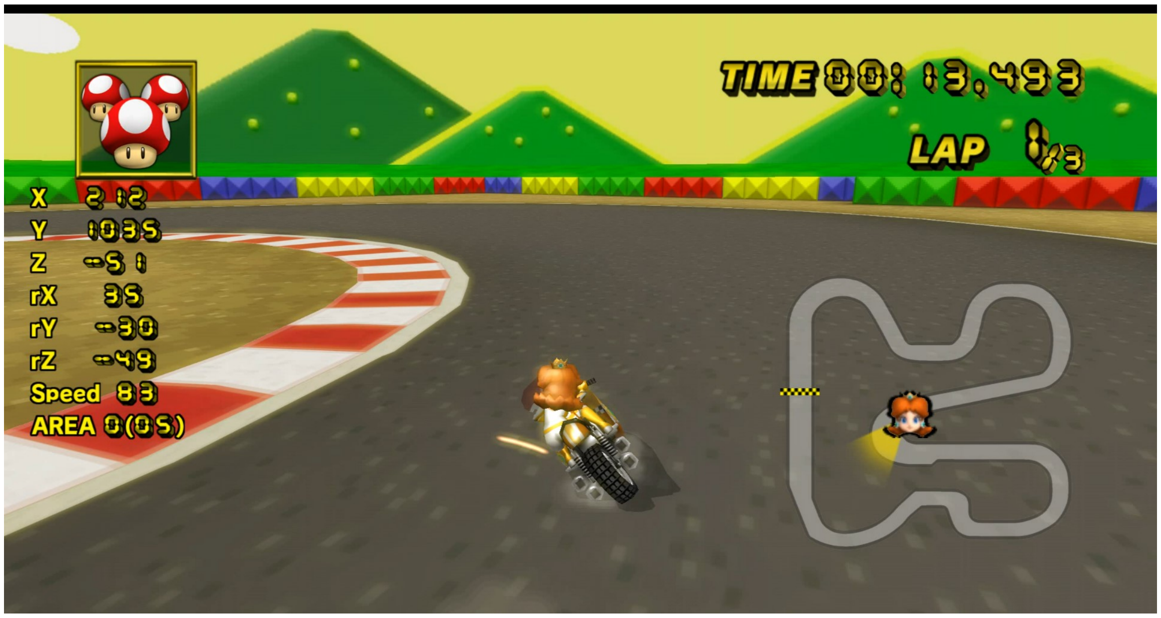
$$p_1 = (212, -51)$$

$$p_2 = (-610, 2231)$$

$$p_3 = (465, 4399)$$

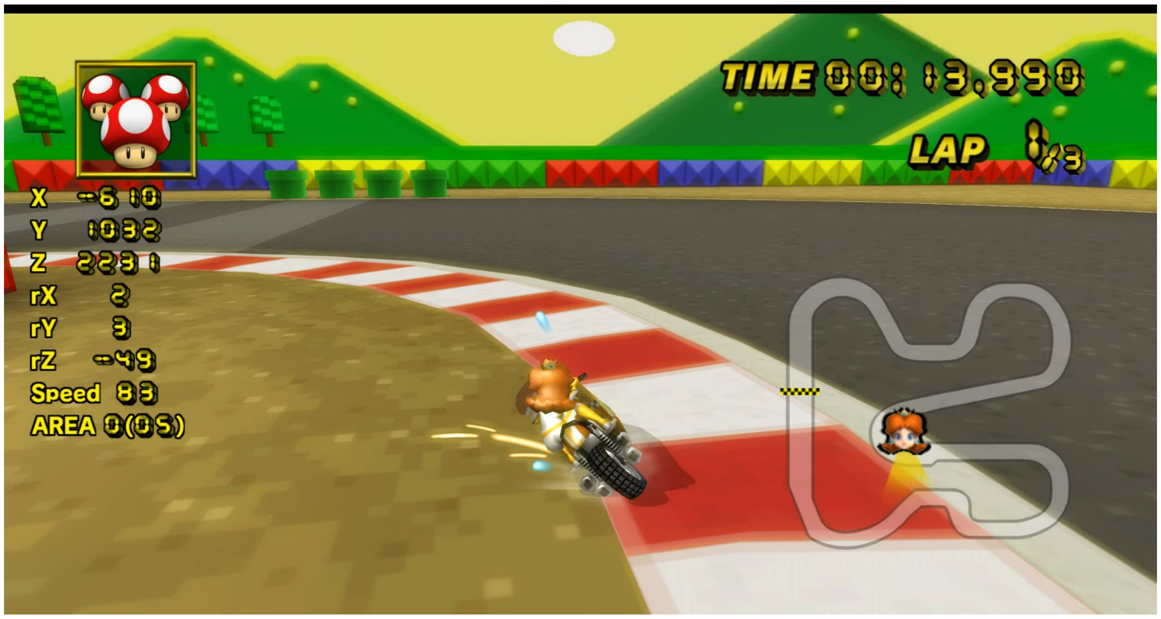
Using these points we can begin to calculate the radius of our circle that our vehicle's path arc lays atop. We start by making an equation for a line that passes through p_1 and p_3 and calculate that distance.

$$C = \sqrt{(p_3(x) - p_1(x))^2 + (p_3(y) - p_1(y))^2}$$



X 212
Y 1035
Z -51
rX 35
rY -30
rZ -49
Speed 83
AREA 0(05)

Figure 3.17: Frame 1 of hard drift



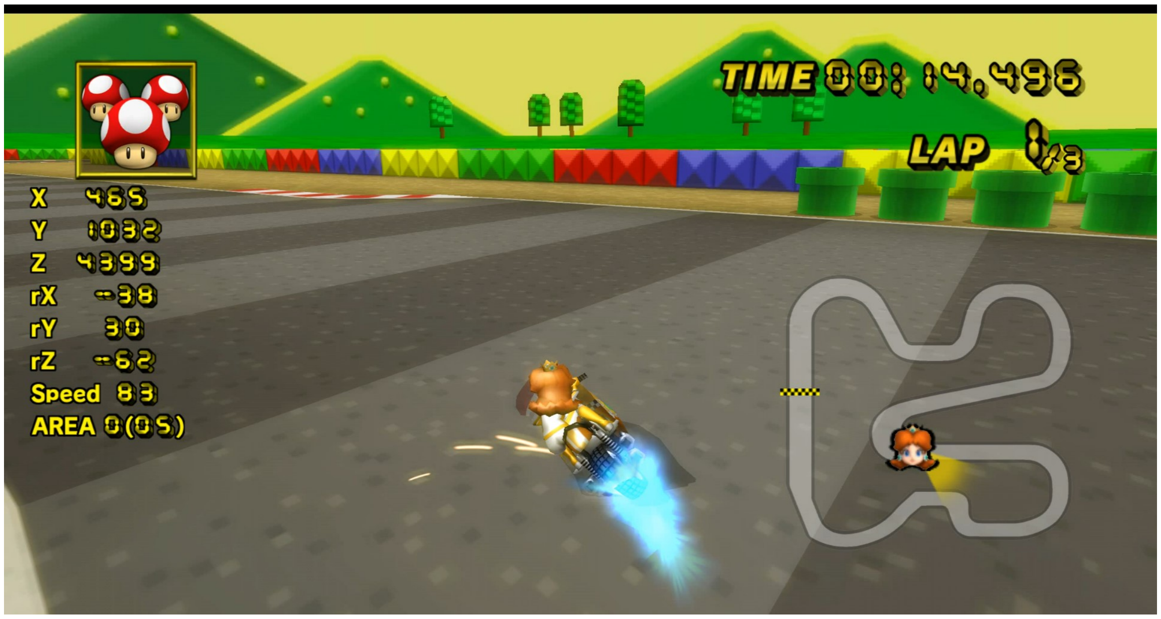
X -610
Y 1032
Z 2231
rX 2
rY 3
rZ -49
Speed 83
AREA 0(05)

Figure 3.18: Frame 31 of hard drift

$$C = \sqrt{(465 - 212)^2 + (4399 - [-51])^2}$$

$$C = \sqrt{253^2 + 4450^2}$$

$$C = 4457.186$$



X 465
Y 1032
Z 4399
rX -38
rY 30
rZ -62
Speed 83
AREA 0(05)

Figure 3.19: Frame 61 of hard drift

$$m = \frac{4450}{253}$$

$$m = 17.589$$

$$y - y_1 = m(x - x_1)$$

$$y - (-51) = 17.589(x - 212)$$

$$y = 17.589x - 3779.854$$

Now that we have our chord formed, we need to determine the equation and distance of the line segment perpendicular to the chord and passing through p_2.

$$m_p = -\frac{1}{m}$$

$$m_p = -0.0569$$

$$y - y_2 = m(x - x_2)$$

$$y - 2231 = -0.0569(x + 610)$$

$$y = -0.0569x + 2196.319$$

Set up a system of equations to determine where our two lines intersect

$$y = 17.589x - 3779.854$$
$$y = -0.0569x + 2196.319$$
$$0 = 17.646x - 5976.173$$
$$17.6459x = 5976.173$$
$$x = 338.672$$
$$y = -3779.854 + 17.589(338.672)$$
$$y = 2177.049$$
$$p_i = (338.670, 2177.049)$$

Next we calculate the distance between the point of intersection and p_2.

$$D = \sqrt{(p_2(x) - p_i(x))^2 + (p_2(y) - p_i(y))^2}$$
$$p_2 = (-610, 2231)$$
$$D = \sqrt{([-610] - 338.670)^2 + (2231 - 2177.049)^2}$$
$$D = \sqrt{948.670^2 + 53.951^2}$$
$$D = \sqrt{902885.479}$$
$$D = 950.203$$

Similar to how we solved the radius of the soft drift circle, we now use the same system of equations to solve for the radius of the hard drift circle.

$$r = n + D$$
$$r^2 = n^2 + (\tfrac{C}{2})^2$$
$$r^2 = n^2 + 2Dn + D^2$$
$$r^2 - r^2 = n^2 - n^2 + (\tfrac{C}{2})^2 - 2Dx - D^2$$
$$0 = (\tfrac{C}{2})^2 - 2Dn - D^2$$
$$0 = (2228.593)^2 - 2(950.203)n - (950.203)^2$$
$$0 = 4966626.760 - 1900.406n - 902885.479$$
$$0 = 4063741.281 - 1900.4061n$$

$$n = 2138.354$$

Substitute back in to solve for r

$$r = n + D$$

$$r = 2138.354 + 950.203$$

$$r = 3088.557$$

Using our earlier equation for the line perpendicular to the chord, and the length of our radius r, we can solve for the center of our circle.

Let side lengths be of length a and $17.575a$.

$$3088.557 = \sqrt{a^2 + (17.575a)^2}$$

$$3088.557 = \sqrt{309.870a^2}$$

$$3088.557 = 17.603a$$

$$a = 175.456$$

This shows us we have side lengths of 3083.644 and 175.456

$$c_{x,y} = p_2 + \delta x, \delta y$$

$$c_{x,y} = ([-610] + 3083.644, 2231 - 175.456)$$

$$c_{x,y} = (2473.644, 2055.544)$$

Now we have all the pieces to make our equation of a circle which can be written as the following:

$$r^2 = (x - c_x)^2 + (y - c_y)^2$$

$$9539184.342 = (x - 2473.644)^2 + (y - 2055.544)^2$$

Our last step for this is to calculate angular rotation, which we can do by discovering how far in degrees our vehicle traveled around the center of our circle.

Let our angle of rotation be $\angle a$, then using the triangle we used earlier to calculate the distance from the chord to the circle's center (it solved for the variable n), we can solve for $\angle \frac{a}{2}$.

$$\cos(\tfrac{\angle a}{2}) = \tfrac{n}{r}$$

$$\cos(\tfrac{\angle a}{2}) = \tfrac{2138.354}{3088.557}$$

$$\cos(\tfrac{\angle a}{2}) = 0.69235$$

$$\tfrac{\angle a}{2} = 46.184$$

$$\angle a = 92.368 \text{ degrees}$$

Angular rotation is 1.540 degrees per frame.

3.3.5 Calculating Angular Rotation of "Soft Drift"

Once again we are going to do practically the same thing with a different set of coordinates which were observed following the same methods as previously, except the stick position is now in place for a soft drift.

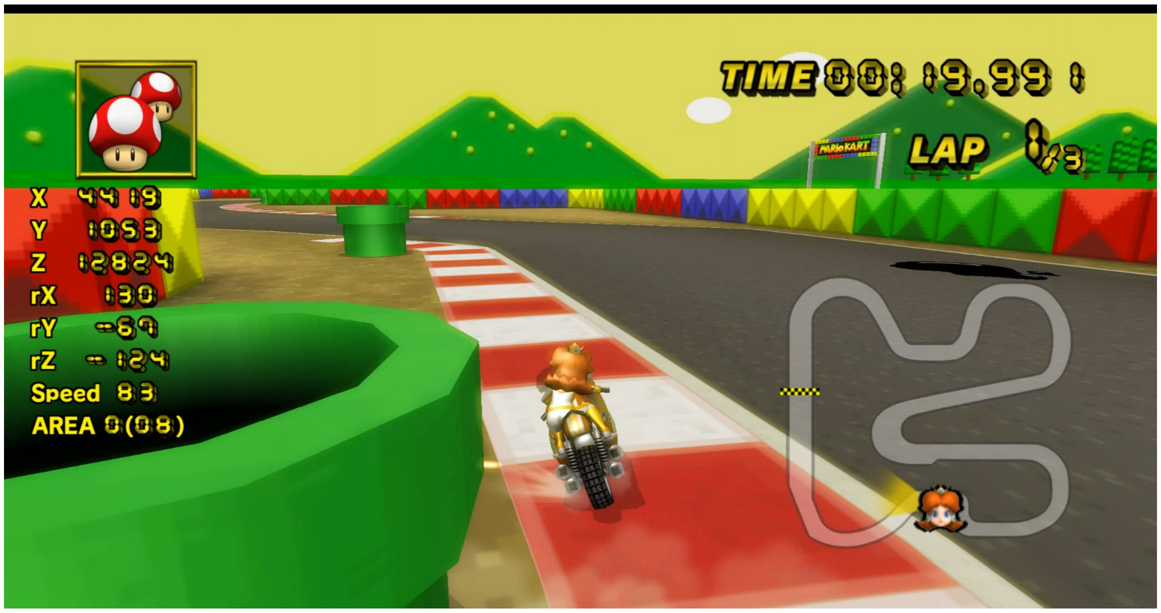


Figure 3.20: Frame 1 of a soft drift

From these images we get the following points:

$$p_1 = (4419, 12824)$$

$$p_2 = (2689, 12706)$$

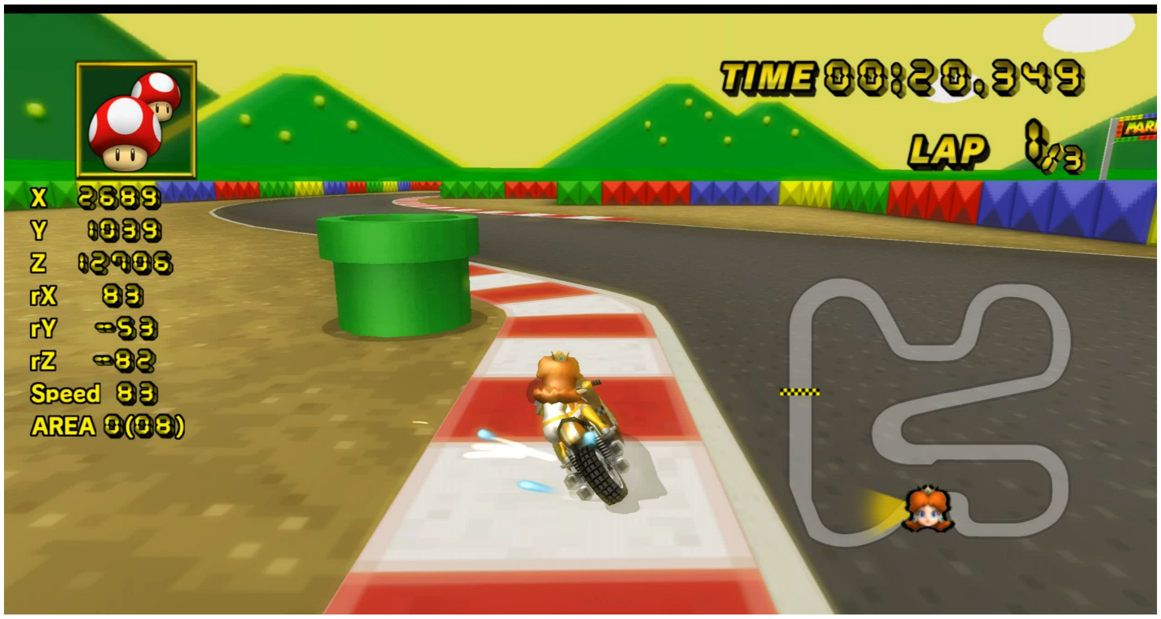


Figure 3.21: Frame 21 of a soft drift

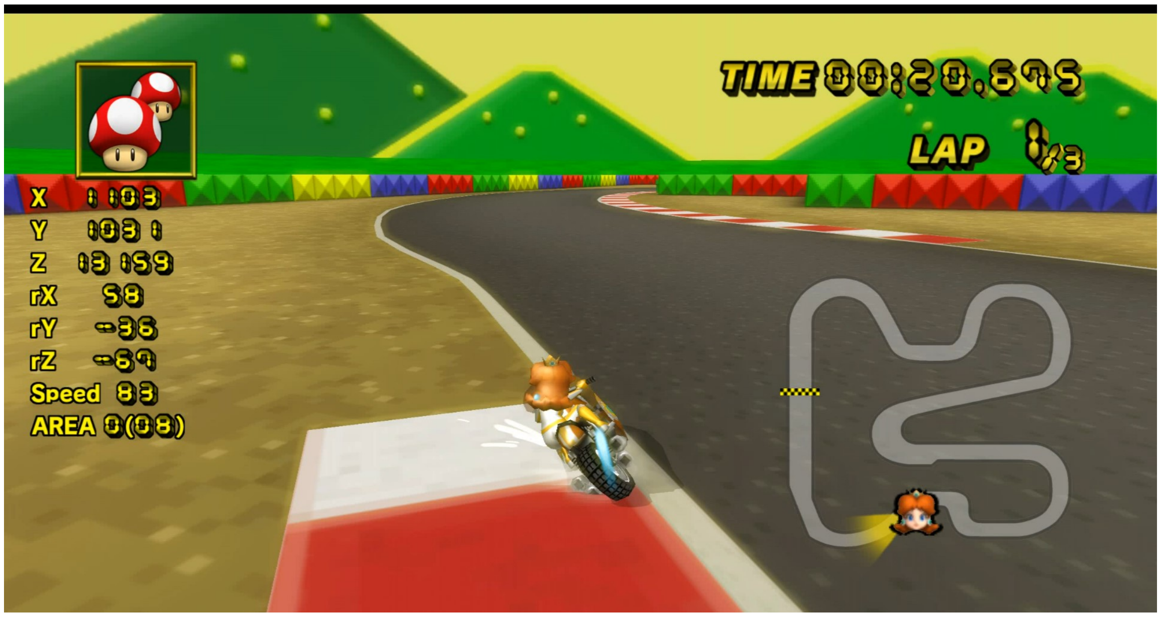


Figure 3.22: Frame 41 of a soft drift

$$p_3 = (1103, 13159)$$

Using these points we can begin to calculate the radius of our circle that our vehicle's path arc lays atop. We start by making an equation for a line that passes through p_1 and p_3 and calculate that distance.

$$C = \sqrt{(p_3(x) - p_1(x))^2 + (p_3(y) - p_1(y))^2}$$
$$C = \sqrt{(1103 - 4419)^2 + (13159 - 12824)}$$
$$C = \sqrt{(-3316)^2 + 335^2}$$
$$C = \sqrt{10995856 + 112225}$$
$$C = \sqrt{11108081}$$
$$C = 3332.879$$
$$m = \frac{p_3(y)-p_1(y)}{p_3(x)-p_1(x)}$$
$$m = \frac{335}{-3316}$$
$$m = -0.101$$
$$y - y_1 = m(x - x_1)$$
$$y - 12824 = -0.101(x - 4419)$$
$$y = -0.101x + 13270.431$$

Now that we have our chord formed, we need to determine the equation and distance of the line segment perpendicular to the chord and passing through p_2.

$$m_p = -\frac{1}{m}$$
$$m_p = 9.899$$
$$y - y_2 = m_p(x - x_2)$$
$$y - 12706 = 9.899(x - 2689)$$
$$y = 9.899x + -13912.411$$

Set up a system of equations to determine where our two lines intersect

$$y = -0.101x + 13270.431$$
$$y = 9.899x + -13912.411$$
$$0 = -10.000x + 27182.842$$

$$10.000x = 27182.842$$

$$x = 2718.284$$

$$y = -0.101(2718.284) + 13270.431$$

$$y = -274.547 + 13270.431$$

$$y = 12995.884$$

$$p_i = (2718.284, 12995.884)$$

Next we calculate the distance between the point of intersection and p_2.

$$p_2 = (2689, 12706)$$

$$D = \sqrt{(p_2(x) - p_i(x))^2 + (p_2(y) - p_i(y))^2}$$

$$D = \sqrt{(2689 - 2718.284)^2 + (12706 - 12995.884)^2}$$

$$D = \sqrt{[-29.284]^2 + [-289.884]^2}$$

$$D = \sqrt{857.553 + 84032.733}$$

$$D = \sqrt{84890.286}$$

$$D = 291.359$$

Similar to how we solved the radius of the soft drift circle, we now use the same system of equations to solve for the radius.

$$r = n + D$$

$$r^2 = n^2 + (\tfrac{C}{2})^2$$

$$r^2 = n^2 + 2Dn + D^2$$

$$r^2 - r^2 = n^2 - n^2 + (\tfrac{C}{2})^2 - 2Dn - D^2$$

$$0 = (\tfrac{C}{2})^2 - 2Dn - D^2$$

Substitute in our constant values

$$0 = (1666.440)^2 - 2(291.359)n - (291.359)^2$$

$$0 = 2777020.607 - 582.718n - 84890.067$$

$$0 = 2692130.540 - 582.718n$$

$$n = 4619.954$$

Substitute back in to solve for r

$$r = n + D$$

$$r = 4619.954 + 291.359$$

$$r = 4911.313$$

Using our earlier equation for the line perpendicular to the chord, and the length of our radius r, we can solve for the center of our circle.

Let side lengths be of length a and $9.899a$.

$$4911.313 = \sqrt{a^2 + (9.899a)^2}$$

$$4911.313 = \sqrt{98.990a^2}$$

$$4911.313 = 9.949a$$

$$a = 493.63$$

This means our side lengths are 493.63 and 4866.444 respectively.

$$p_2 = (2689, 12706)$$

$$c_{x,y} = p_2 + 493.63, 4866.444$$

$$c_{x,y} = (2689 + 493.63, 12706 + 4866.44)$$

$$c_{x,y} = (3182.63, 17592.444)$$

Now we have all the pieces to make our equation of a circle which can be written as the following:

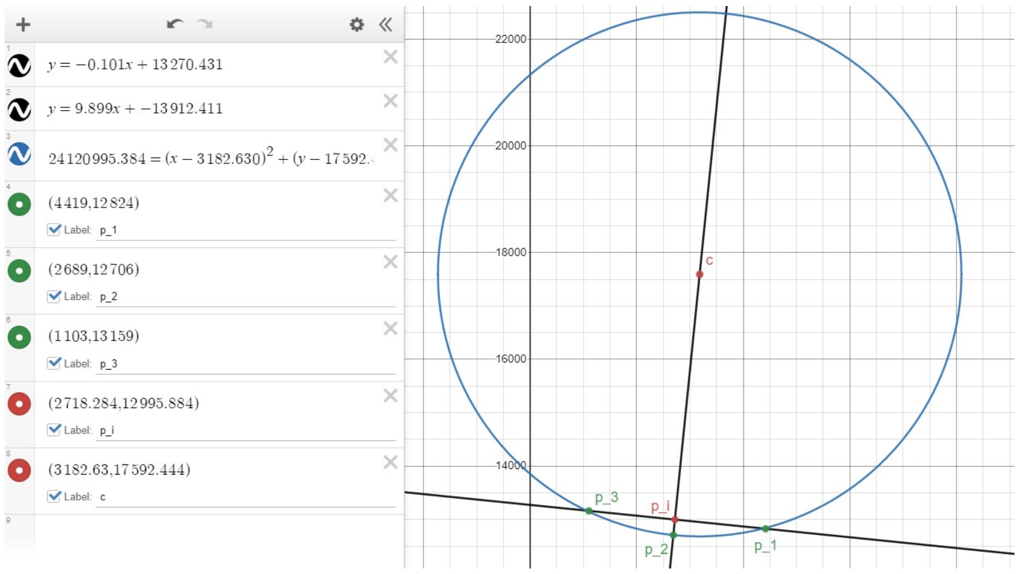
$$r^2 = (x - c_x)^2 + (y - c_y)^2$$

$$4911.313^2 = (x - 3182.63)^2 + (y - 17592.444)^2$$

$$24120995.384 = (x - 3182.63)^2 + (y - 17592.444)^2$$

To try and illustrate exactly what we have constructed, here is a diagram which accurately depicts our calculations.

Our last step for this is to calculate angular rotation, which we can do by discovering how far in degrees our vehicle traveled around the center of our circle.



1 $y = -0.101x + 13270.431$

2 $y = 9.899x + -13912.411$

3 $24120995.384 = (x - 3182.630)^2 + (y - 17592.$

4 (4419,12824) Label: p_1

5 (2689,12706) Label: p_2

6 (1103,13159) Label: p_3

7 (2718.284,12995.884) Label: p_i

8 (3182.63,17592.444) Label: c

9

Figure 3.23: Graphical representation of soft drift calculations

Let our angle of rotation be $\angle a$, then using the triangle we used earlier to calculate the distance from the chord to the circle's center (it solved for the variable n), we can solve for $\angle\frac{a}{2}$.

$$\cos(\tfrac{\angle a}{2}) = \tfrac{n}{r}$$

$$\cos(\tfrac{\angle a}{2}) = \tfrac{4619.954}{4911.313}$$

$$\cos(\tfrac{\angle a}{2}) = 0.94068$$

$$\tfrac{\angle a}{2} = 19.83461579$$

$$\angle a = 39.66923158 \text{ degrees}$$

angular rotation is 0.992 degrees per frame

3.3.6 Comparing Drift Paths

We have now determined exactly how our vehicle can travel in drifts. It can rotate its trajectory by any of these amounts when starting or in a drift:

- hard Drift: 1.540 $\frac{d}{f}$ (degrees per frame)
- soft Drift: 0.992 $\frac{d}{f}$

- fail Drift: 0.213 $\frac{d}{f}$

From this information, along with the relevant circle radii calculated from earlier, we can begin to visualize our vehicle's capability on the first turn.

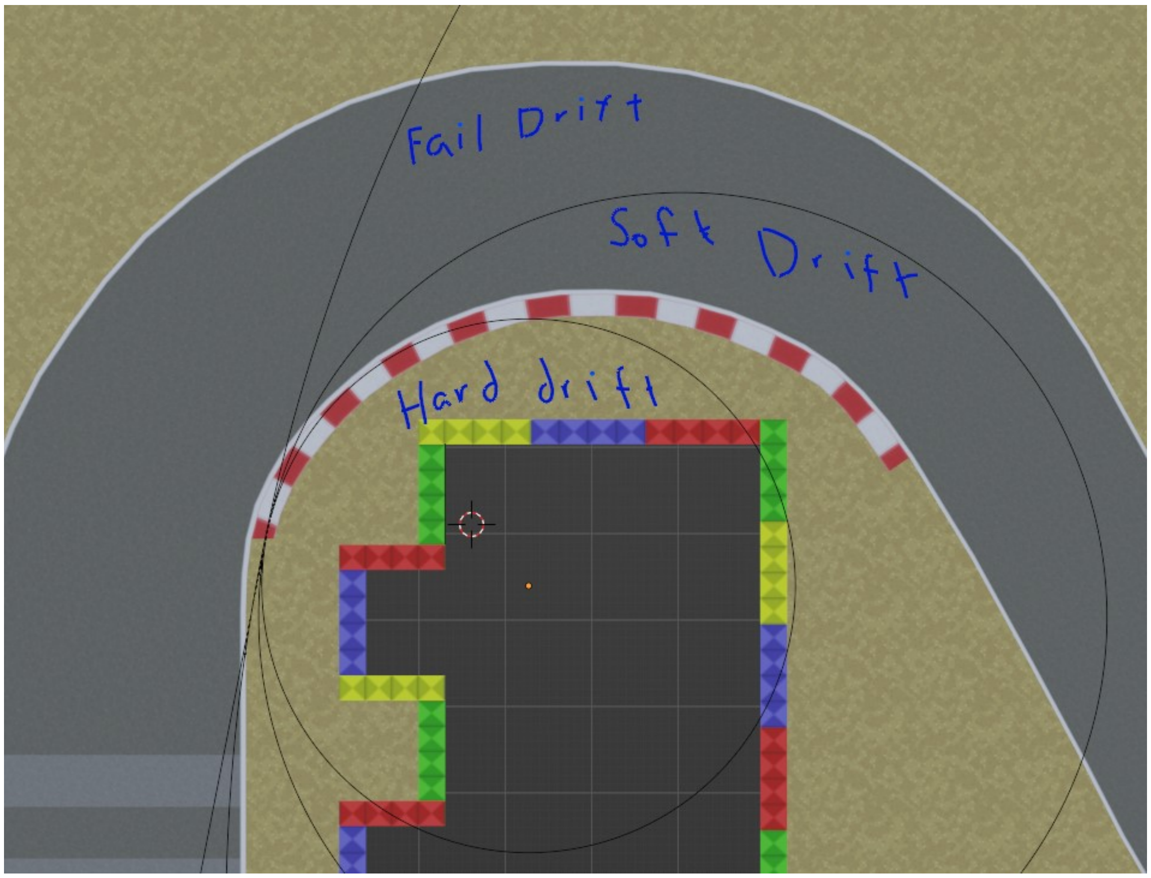


Figure 3.24: Diagram of vehicle paths with corresponding stick positions

Looking at this first turn now begs the question, what stick positions will we be using for this first turn? While we certainly will be using hard and soft drift, it is uncertain at this stage whether we would use any fail drifting. To determine exactly how long we can fail drift and still stay on course, we need to determine 3 important factors:

- How long will we be in our drift for?

- How many drifts can/will we do on the turn? (One way of mathematically thinking about this is over the total distance traveled, how many times will we be able to reach a miniturbo charge of 270)?
- Where do we plan to end our final drift and start our alignment into the next turn?

Out of all 3 of these, the second is most easy to solve. We can solve with a rather simplistic proof.

3.3.7 Calculating the quantity of drifts on the first turn

On the first turn charging only 1 miniturbo is strictly faster than any other quantity of miniturbos.

Proof. Let us assume by contradiction that you can charge 2 or more miniturbos on the first turn and it is faster. This means the turn must allow our vehicle to reach 270 charge (c) twice over its drift. We know there are 2 different ways to increase charge depending on stick position:

- Stick is held in a fail drift (Input ranges -7 to 2) which increases charge value by 2 every frame drift.
- Stick is held in a hard or soft drift (Input ranges 3 to 7) which increases charge value by 5 every frame of the drift.

Immediately this presents us with two cases: one where the stick is permanently held in fail drift, and the other where it permanently soft drift. With these two cases, we can determine the total angle they must rotate to align properly for the next turn. If this angular rotation is less than or equal to the vehicle's rotation through each respective process of charging 2 miniturbos or more, we can assume that it is possible to charge 2 miniturbos with that method.

Using the value calculated back in section 3.2.2, we can assert that our starting angle is 11.173 degrees. Let $\angle a$ be the angle of our vehicle after releasing its final miniturbo. This means the total angular rotation during our vehicle's drifts would be $p - 11.173$, since we know that p must be a larger positive value than 11.173 (for confirmation view figure 3.24).

If we are to claim that 2 miniturbos may be charged over this turn, we can calculate the total angular rotation of our vehicle in both our aforementioned cases. We can do this using the charge values referenced above, along with the calculated angular rotation of each drift type in sections 3.3.3-5. With this in mind, we can analyze our cases.

- Case 1: The stick is permanently held in a fail drift. Using our calculated values, we can further analyze and determine our total angular rotation θ when charging two miniturbos (neglecting miniturbo boost as this is a lower bound and that would only increase the total angular rotation).

$$\theta = \frac{270(c)}{2(\frac{c}{f})} * 0.213(\tfrac{d}{f})$$

$$\theta = 135(f) * 0.213(\tfrac{d}{f})$$

$$\theta = 28.755 \text{ degrees}$$

  This means that if we were to charge 2 miniturbos, our vehicle would rotate $2 * 28.755 = 57.51$ degrees. This would mean that $p \geq 68.683$, which based on figure 3.24 seems quite reasonable. However, something else the figure highlights is that if we were to fail drift the entire time we would drive into the offroad on the opposite side, which is certainly slower. Hence, this case is invalid.

- Case 2: The stick is permanently held in a soft drift. Once again, similar to case 1, we can calculate total angular rotation θ as follows:

$$\theta = \frac{270(c)}{5(\frac{c}{f})} * 0.992(\frac{d}{f})$$

$$\theta = 54(f) * 0.992(\frac{d}{f})$$

$$\theta = 53.568 degrees$$

Following a similar logic, 2 miniturbos would rotate our vehicle at least $2 * 53.568 = 107.136$ degrees, making $p \geq 118.309$. Once again, while it is slightly closer it does appear that p could very well satisfy this value. That being said, referencing figure 3.24 again, we note that holding a soft drift the entire time would take the vehicle incredibly wide in comparison to combining a soft and hard drift to try and stay as tight to the turn as possible. Furthermore, we would have to delay releasing our wheelie substantially, and this simply is not viable for a faster method.

- Now let us imagine a Case 3 where the stick positions are some combination between hard and soft drift where each type of drift is used. We could assume that $\theta > 107.136$ since hard drift has a larger coefficient of angular rotation ($1.540 \frac{d}{f}$) than soft drifting ($0.992 \frac{d}{f}$). Ultimately, even if we are able to take the turn optimally tight, not only are we unable to ensure that the inequality $p \geq 11.173 + \theta$ would hold, but we would certainly delay our wheelie. Using data collected earlier in section 2.3 we know it is faster to instantly release our wheelie, hence why any Case 3 is also invalid.

- Finally, while this may seem redundant, lets assume a Case 4 where you do not actually release a miniturbo (or technically you charge 0 miniturbos) because you do not charge it all the way. Assuming you take the tightest path, it is actually impossible not to charge it as $p \geq 11.173 + 54(f) * 1.540(\frac{d}{f}) = 94.333$, which is a very reasonable value for p.

Since there is no possible case where charging 2 miniturbos would be faster than charging 1 miniturbo, we can assert that it is optimal to only charge 1 miniturbo on the first turn. □

With this in mind, we can now safely assume we not only need to charge 1 miniturbo on the first turn, but we are also at no rush to do so as we are guaranteed to have it completely charged before we rotate to the angle of 94.333 degrees.

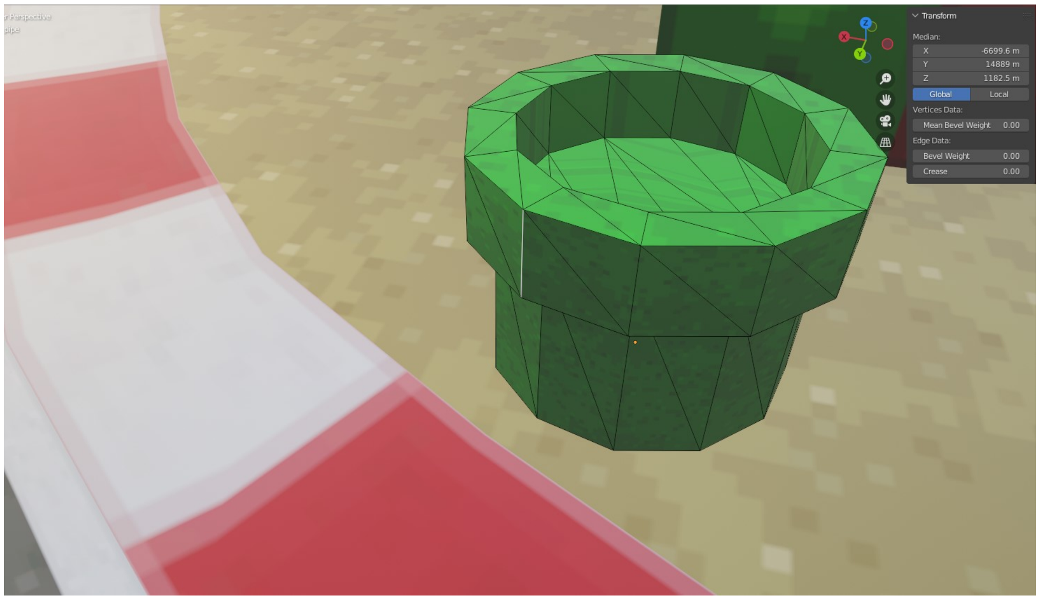
3.3.8 Calculating where to release our miniturbo

Now that we know only to charge 1 miniturbo, we need to decide where to release our miniturbo. Wherever we decide to release our miniturbo, our angular rotation is reduced to 0 and we are forced to travel in a straight line until we begin the next drift. With this in mind, there are 2 important factors to calculate this.

- How close can we maneuver around the pipe at the end of the first turn and beginning of the second turn?
- Where do we want to begin our next drift?

While the first part of this is somewhat mundane, the second certainly is not. In order to calculate this accurately, we would need to repeat the calculations in section 3.3.7 for our second turn. Due to some extraneous calculations, we know the exact entrance to our third turn so working backwards we can calculate the exit of the second turn, and then determine our exact entrance to the second turn. However, as that would require substantial amounts of computation and would essentially require us to calculate the second turn entirely before we could finish the first, we opt to make a few assumptions we cannot necessarily prove without further calculations about this second turn. Let us assume this turn only allows for 1 single miniturbo. With this simple assumption (which we could surely prove with a substantial amount of work), we can make more conclusions based on where we want to start our drift into the second turn.

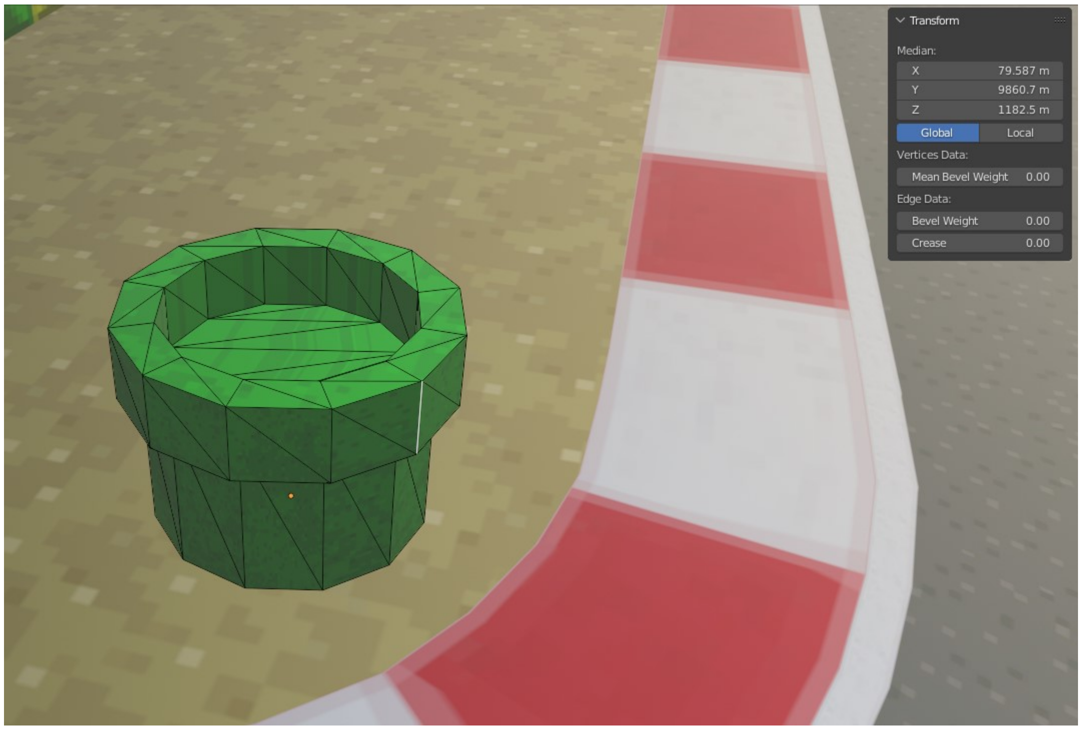
For now however, let us go back to the pipes. For us we only need to worry about the section of each pipe that is closet to the road and ensure we do not hit it. For the first pipe, the closest edge is at the coordinates (-6699.59, 14889.30) and the second pipe's closest edge is at (79.59, 9860.72). While it may seem easy to just path our vehicle to slightly avoid these pipes, we actually have to account for something quite difficult: hitboxes.



Median:
X -6699.6 m
Y 14889 m
Z 1182.5 m
Vertices Data:
Mean Bevel Weight 0.00
Edge Data:
Bevel Weight 0.00
Crease 0.00

Figure 3.25: First pipe and its most protruding edge

For reference as to what the hitbox looks like, figure 2.13 is a model representation. The largest sphere out of all 5 present in our hitbox is the one at the front of our vehicle which has a radius of 65 units. While it seems simple to make our release point for our miniturbo then only 65 units away from the pipe, it is important to note that our vehicle rotates vertically when it drives (technically called roll, similar to an airplane). This means we need to make sure our vehicle does not rotate into the pipe, so to do that we need to calculate exactly how far it rotates to the side in a soft drift and hard drift. That being said, if we drive directly by the pipe in a wheelie we can ignore this rotation and drive a fair bit closer.


Transform

Median:

```
X    79.587 m
Y    9860.7 m
Z    1182.5 m
[✓ Global] Local
```

Vertices Data:

```
Mean Bevel Weight    0.00
```

Edge Data:

```
Bevel Weight    0.00
Crease          0.00
```

Figure 3.26: Second pipe and its most protruding edge

From data we collected from the game we know that in a hard drift our vehicle's axis of rotation is 30 degrees towards whichever side it is leaning. However, there is no data or known way to know exactly how far it rotates in a soft drift. From our limited observations it appears the rotation is linearly proportional to the stick position such that if you multiply the stick value by $\frac{30}{7}$ you are left with the vehicles rotation with any given stick position. Using this logic, in a soft drift our vehicle would rotate $3 * \frac{30}{7} = \frac{90}{7} = 12.857 degrees$. Using this value and the dimensions of our current hit box we can determine the length at which our hitbox extends when in a soft drift. It is important to note that the axis of rotation lies directly below the bike's wheels as well.

Looking at the above figures, it is evident that our vehicle's maximum width in a soft and hard drift is 91.48 and 124.5 units respectively. This means that we can make 2 equations for circles, one for the closest point to the road on each pipe, that my vehicle cannot enter without leaning into it. This would allow us to determine suitable points for where to release our miniturbo and where to align. Lets start by making equations for pipe 1 (p_1) and pipe 2 (p_2) for both soft (s) and hard (h) drifting.

$$p_{1s} : 91.48^2 = (x + 6699.59)^2 + (y - 14889.30)^2$$
$$p_{1h} : 124.5^2 = (x + 6699.59)^2 + (y - 14889.30)^2$$
$$p_{2s} : 91.48^2 = (x - 79.59)^2 + (y - 9860.72)^2$$
$$p_{2h} : 124.5^2 = (x - 79.59)^2 + (y - 9860.72)^2$$

Again, we face a conundrum involving the second pipe. Since we have not proved the drift will start with a soft drift or a hard drift, it is really challenging to choose which equation we should use to determine where to start our second drift. Because of this, let us assume the drift starts as a soft drift and we can use soft drift circle equation to properly estimate our vehicle's valid path options. Furthermore, lets also assume a drift on the second turn would start as soon as the vehicle is adjacent to the pipe. While this may seem like a stretch, it is actually quite realistic based on historic performances on this track so such an assumption has some credibility.

To properly illustrate what this represents graphically, figure 3.27 outlines not only the soft drift circle equation but the prior turn as well.

However, in this diagram we do not exactly know what circle equation to use for the first pipe. For example, if we are able to charge a miniturbo before reaching the pipe it would be substantially faster to release it right next to the pipe in such a way that you would not be in a drift when you pass it so you can drive tighter next to it. This would result in the circle with radius of 65 which just so happens to be the one in Figure 3.25. However, if the miniturbo cannot be charged by then we would need to use a circle with a larger radius, which would be a bit slower as you would need to go wider.

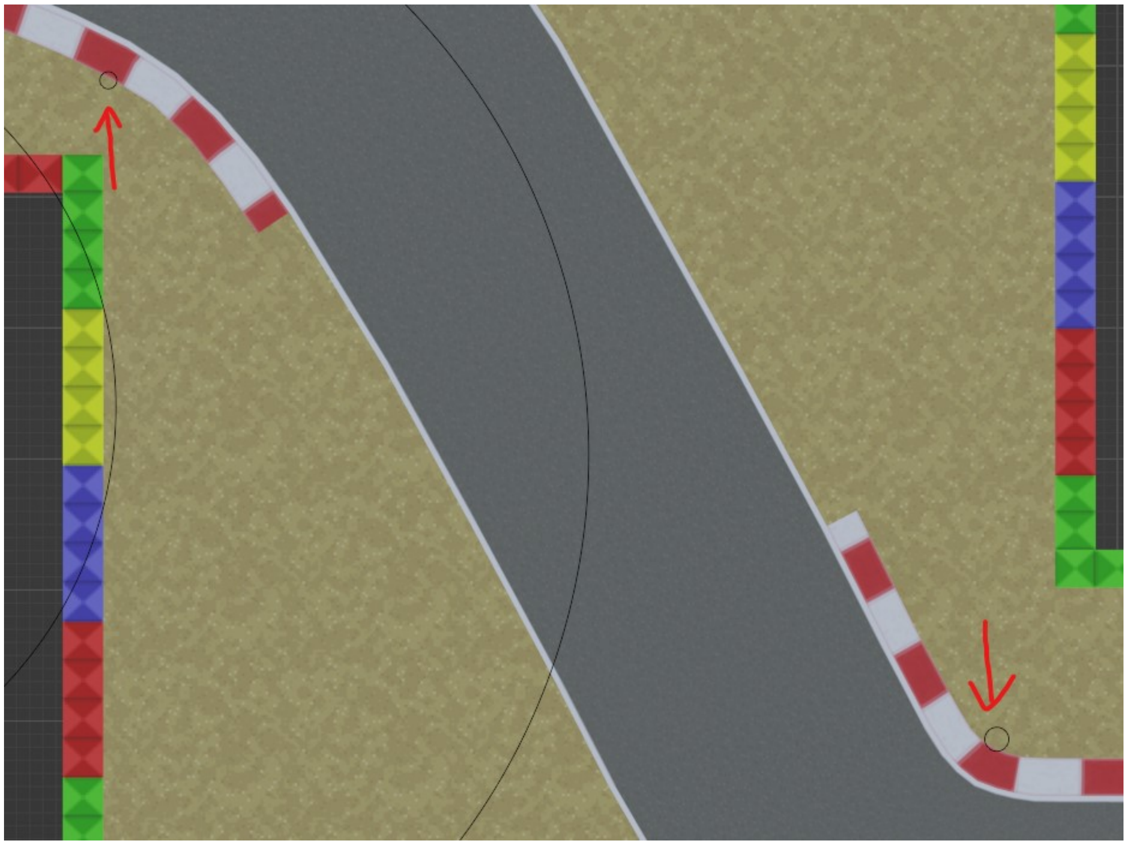


Figure 3.27: Model of second straight with pipe collision boundaries

Since we know we are able to charge a miniturbo very on in this turn due to our earlier calculations in section 3.3.7, we can safely assume that our miniturbo would be charged by the time we reach the first pipe. With this in mind, we can begin to calculate the direction we will need to travel on the second straight, as we need that to determine the angle at which we release the second miniturbo. To determine this angle, we need to find a line that is tangent to each circle on the path our vehicle would take.

$$p_1 : 65^2 = (x + 6699.59)^2 + (y - 14889.30)^2$$

$$p_{2s} : 91.48^2 = (x - 79.59)^2 + (y - 9860.72)^2$$

$$|-6699.59a + 14889.30 + c| = 65 * \sqrt{a^2 + 1}$$

$$|79.59a + 9860.72 + c| = 91.48 * \sqrt{a^2 + 1}$$

Used a systems of equations calculator, 4 solution pairs

$$a = 0.713414, c = -10029.9$$

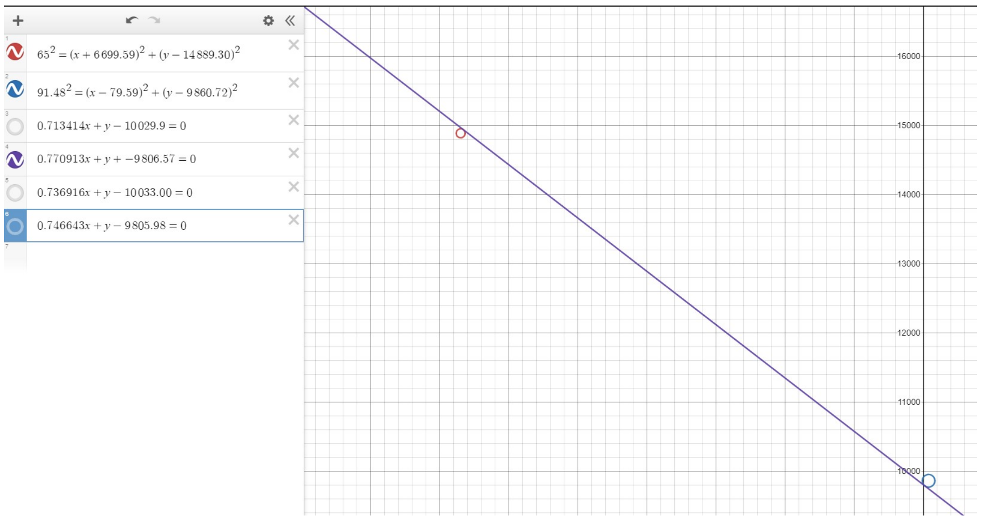
$$a = 0.770913, c = -9806.57$$

$$a = 0.736916, c = -10033.00$$

$$a = 0.746643, c = -9805.98$$

Find which solution is the proper tangent line

$$y = -0.770913x + 9806.57$$



1 $65^2 = (x + 6699.59)^2 + (y - 14889.30)^2$

2 $91.48^2 = (x - 79.59)^2 + (y - 9860.72)^2$

3 $0.713414x + y - 10029.9 = 0$

4 $0.770913x + y + -9806.57 = 0$

5 $0.736916x + y - 10033.00 = 0$

6 $0.746643x + y - 9805.98 = 0$

Figure 3.28: Graphical representation of tangent line to both circles

Finally our last job here is to convert this slope into an angle $\angle\theta$ such that we can have a final angle for our vehicle to be at when it releases its miniturbo.

$$m = -0.770913$$

Create a triangle with legs of length a and b

$$a = 1, b = 0.770913$$

$$tan(\theta) = \frac{1}{0.770913}$$

$$\theta = 52.37 \text{ degrees}$$

This angle however is from the bottom, so to determine the relative angle to our original starting position we need to subtract it from 180

$$\angle a = 180 - \theta = 127.63 \text{ degrees}$$

Now that we have the angle to release our miniturbo, we need to calculate the exact position at which to do so. This position is the one we calculate as tangent to the pipe circle because it is the closest to the offroad we can release our miniturbo and align for the next drift.

$$y = -0.770913x + 9806.57$$

$$65^2 = (x + 6699.59)^2 + (y - 14889.30)^2$$

Using the graph I already made, I can see the intersection quite easily

$$(-6660.245, 14941.039)$$

3.3.9 Determining our sequence of inputs for the First Turn

To start let us review our information. We have the starting position for our first turn *A* (-12773, 12942) and our starting angle (11.173 degrees). We also have our ending position *B* (-6660.245, 14941.039) and ending angle (127.63 degrees). Furthermore, we have each of the angular rotation values for each type of drift (hard 1.540 $\frac{d}{f}$, soft 0.992 $\frac{d}{f}$, fail 0.213 $\frac{d}{f}$) along with their corresponding charge values. Using all of this information, we are going to attempt to optimize the shortest path from the start to end point such that the vehicle rotates perfectly throughout its path to end with the correct angle to exit the turn.

Immediately we can construct a circle that passes through both the starting and ending points and is tangent to the tangent line calculated above. To do this we can construct a perpendicular bisector between our two points and another perpendicular line through the point touching the tangent line. Wherever these two perpendicular lines meet must be the center of the circle. From there we could calculate the radius with ease and we would have our circle, along with a distance.

First we construct our perpendicular bisector between points *A* and *B* using a calculator.

$$m = \frac{14941.039 - 12942}{-6660.245 - -12773}$$

$$m = 0.33$$

$$m_p = -3.06$$

$$y = -3.06x - 15791.3372$$

Now we calculate a perpendicular line to our tangent line through B.

$$y = 1.297x + 23580.464$$

Combine the equations in a system to get the point

$$(-9036.447, 11860.192)$$

Finally calculate the distance between the center point and A to get the radius

$$r = 3890.005$$

With this radius we can attempt to calculate the distance of the arc in between A and B. With this arc length we will have an upper bound for shortest distance and time to take the turn meeting the conditions about exit point and trajectory.

Start by calculating the interior angle between the center point $\angle C$ and points A and B

$$\angle C = 111.496 \text{ degrees}$$

Next we can calculate the circumference

$$C = 2\pi r = 7,780.01\pi = 24441.622$$

We now determine the portion of the circumference covered by our vehicle path to determine the overall distance traveled.

$$\frac{111.496}{360} * 24441.622 = 7569.842$$

Finally we divide out the distance traveled by the velocity to determine how long it takes our vehicle to travel on its path (t).

$$t = \frac{7569.842}{82.950} = 91.258 \text{ frames}$$

Fortunately we do not need to check if this is between hard and soft drift because we know based on figure 3.25 that it lies between both of those arcs, hence this is a valid path.

3.3.10 Optimizing our turn

Originally when I intended to do this step, I was going to run several simulations using these equations to try and optimize a shortest theoretical route. In its current state, it is really challenging to calculate distances traveled in increments considering the path is curved. Something like a Fourier series could potentially work but would require a large amount of setup. My last ditch effort was to attempt to model distance traveled using kinematics.

We can start this by attempting to write out another massive system of equations with the intent of optimizing one regarding distance.

Let a, b, and c be a real number representing the amount of frames our vehicle is in a hard, soft, and fail drift respectively. Also let f be the total amount of frames that elapse over the turn.

$$f = a + b + c$$

$$127.63 = (a * 1.540) + (b * 0.992) + (c * 0.213) + 11.173$$

One major issue with this however is that the order of when you want to have your stick in certain positions on the turn matters greatly. For example, if you start a hard drift and then follow it up with a soft drift for the same length of time, you will end up at a different position than if you were to start in a soft drift and then go to a hard drift. That being said, regardless of order your angle will always be the same, therefore the second equation above stands.

One potential modification we can attempt to do is to remove the fail drift portion of the equation, as it would probably be unused anyways due to the sharpness of the turn. That leaves us with a much more simplified setup:

$$127.63 = (a * 1.540) + (b * 0.992) + 11.173$$

$$127.63 - 11.173 = 1.540a + 0.992b$$

$$116.457 - 0.992b = 1.540a$$

Using this system above and the equation for our baseline we could begin iterations and optimization attempts, but some of the mathematics required for it are incredibly complex and would take a substantial amount of computational power or high level mathematical understanding.

3.4 Higher Level Math

Most of the mathematics outlined above is rather fundamental, even if it is used in rather unorthodox ways. This is largely due to the fact that most upper level math surrounding shortest paths is incredibly complex, and much of it remains either too difficult to utilize or even comprehend for an undergraduate level, or entirely unsolved.

That being said, if we wished to attempt to utilize more advanced mathematics there are two main routes we could attempt to follow. One of which would define a minimum link path within a simple polygon, and the other for purely shortest euclidean path. While neither of these are truly the most optimal, they do provide more potential insight.

3.4.1 Minimum Link Path

We start this by prefacing the idea of a minimum length path. Imagine there is some set of points $s(start), d(destination)$ which lie within some simple polygon. We define a simple polygon as a polygon which does not intersect itself or have holes. As seen in figure _, we have a simple polygon with a start and destination point. However, there is no direct line between the two such that the line is entirely contained within the simple polygon. The main goal of the minimum link path would be to secure a polygonal path linking points *s* and *d* such that it lays within the bounds of our simple polygon *P* and has the minimum amount of edges or vertices.

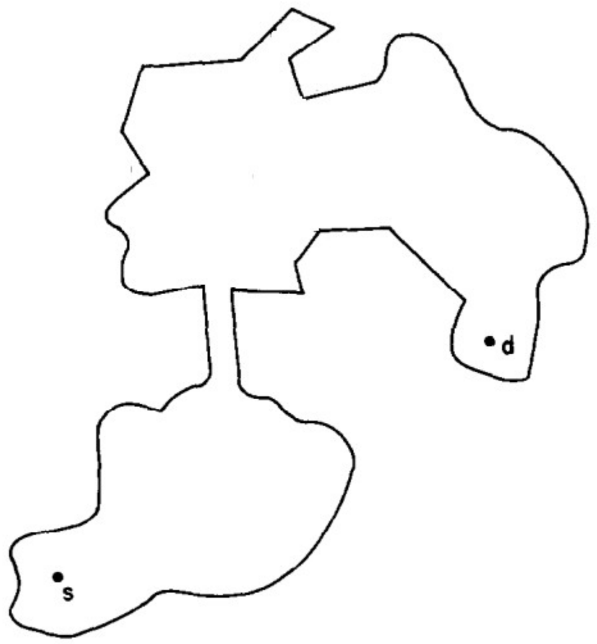


Figure 3.29: Example of starting and destination points on a simple polygon [10]

Ultimately this problem is quite complex. It's application lends itself quite nicely to game mechanics, since it is optimal for our vehicle to travel in straight lines any minimum link path by definition would optimize our time in a wheelie. That being said, these paths may also lead us wide and prove difficult to accurately follow due to limitations in game movement. With all that accounted for, this could still be incredibly valuable for determining an optimal path.

To begin solving the minimum link path we deal with the concept of *visibility* polygons. For each of these, we imagine some sort of light that would expand out from a certain vertex and try to describe the region it would cover within P. We would then denote the boundaries of this visibility polygon as V. To mathematically represent this, we would claim that for some point $x \in P$, the visibility polygon denoted by $V(s)$ is the set of points visible from point s. We would also claim that for all points within $V(s)$, a line can be created within $V(s)$ as follows:

$$V(s) = \{z \in P | sz \bigcap P = sz\}$$

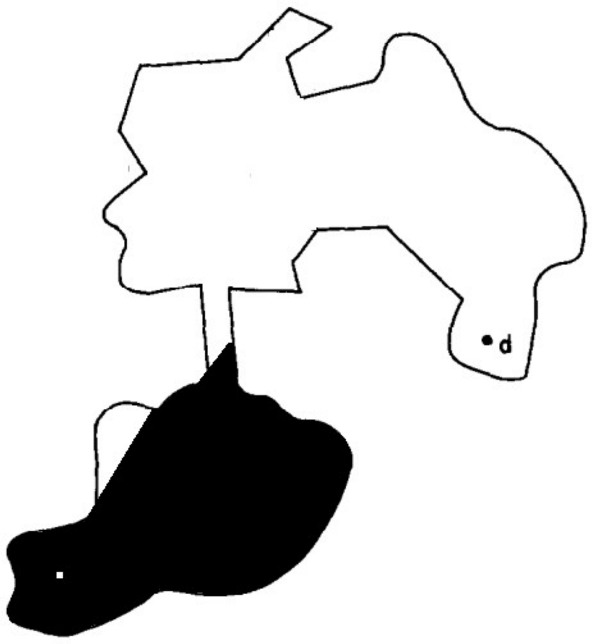


Figure 3.30: Visibility polygon $V(s)$ describing our white point s [10]

Once this has been established, we can attempt to expand our definition of visibility polygons from vertices to edges for more broad use later. For some edge e, we can describe $V(e)$ as follows:

$$V(s) = \{z \in P, y \in e \text{ s.t. } zy \bigcap P = zy\}$$

We will use this understanding to expand upon our path as the problem progresses. Imagine we have a hypothetical line segment to use in our minimum link path. To start identifying the next one, we would need to use this definition to construct a visibility polygon from the line segment so we could determine the next possible segment.

We can begin to make assertions about $V(e)$, and claim certain properties must be withheld. One such lemma from Suri's paper claims that $V(e)$ must overlap with an edge of P or it must be a chord separating P into 2 sub-polygons. With our example shown in Figure 3.30, we can see if you exclude the northwestern area outside $V(e)$ that the 2nd part of the lemma would hold, therefore describing $V(e)$ as a subdivided polygon of P. However since that area remains it is clear $V(e)$ intersects an edge of P.

Now that we have established a classification for $V(e)$ we need to find applications for it. This leads into the idea of *triangulation* from Melhourn's 1984 paper on multi-dimensional searching [6]. He defines triangulation as "A triangulation of a vertex set $\{v_1, v_2, ., v_n\}$ is a maximal set of non-intersecting straight line segments between points in this set. A triangulation of polygon P is a triangulation of its vertex set such that all the triangulating edges lie in P." With this definition we can construct a dual graph G. We could link the vertices of G if and only if two triangles within T share a side. Let the triangles that contain s and d be known as t_s and t_d respectively, and any path linking these two as $X_{s,d}$. $T_{\{}1, k\}$ represents the list of all triangles within T.

If we were to actually construct a graph of G as outlined above we would create a fully functional tree. The idea from here would be to use this tree to reach our endpoint d as there must exist at least one path in our tree G that starts in our triangle containing s and ends within d. With this method alone we have limited it down to a finite set of possible solutions (albeit a large quantity), from what was once seen as a problem with infinite possibilities. This is because assuming each path can only pass through a specific triangle t_i once, this leaves a finite amount of possible paths through a finite quantity of triangles from our triangulation.

We could start this by computing the minimum number of links in a path for each triangle $t_n \in T_{\{1,k\}}$. Eventually since $t_d \in T_{\{1,k\}}$ we would compute the minimum links, but we can show that these minimum links calculated have properties that can be shared through a term defined as a window. A window $w(e)$ we define similarly to a visibility polygon except for an entire edge and shows the farthest triangulation in $T_{\{1,k\}}$ that e can go through or onto. Say we have an edge e as our first link in our minimum link path, it is then necessary to determine each triangle t_n that can be reached from this link.

With this in mind we can create an algorithm for determining a minimum link path based on any regular polygon P, outlined in Suri's paper. His algorithm goes as follows:

1. Initialize our Visibility Polygon for our starting point $V(s)$
2. Determine window for e_1 and farthest triangulation in $T_{\{1,k\}}$
3. Loop following two steps until our ending point $d \in e_i$
4. Compute $V(e_i)$
5. $e_i < -w(e_{i-1})$
6. Let $d_i = d_{i+1}$
7. Loop from i to 1
8. $d_j < -$ a point on e_j such that $d_j + 1$ lies within $w(e_i)$

To simplify this algorithm, the idea is to lay edges such that they reach the triangle closest to d within the visibility polygon of s_i. Once a complete link path is formed, it becomes refined by traveling in the reverse direction in an attempt to skip over any potential extra links.

One basic example we have of this can be seen below. We have triangulated our polygon P by using only vertices of P, and from the dual graph we have created a corresponding tree. After this, we number the triangulations counting upwards for each order away from D (for example, the farthest right triangle has order 2 as it is 2 triangles away from D).

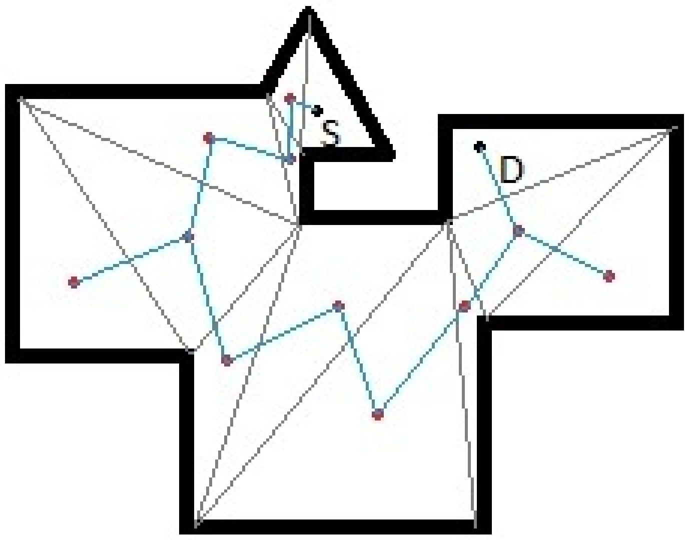


Figure 3.31: Minimum Link Example Part 1

With the tree now made, we must recognize the visibility polygon from s, $V(s)$. With this we recognize that within $V(s)$ we have the window $w(e)$ outlined by the green line. We determine this to be the window as it enters the lowest order triangulation and is a chord of P within $V(s)$.

We then repeat this step with the end point of $w(e)$. While this may seem arbitrary we actually choose it because our lowest order triangulation T_d lies within the visibility polygon centered at that point seen in the figure below.

Finally, we connect the final window to d and we have formed our path. We now work backwards to determine the points that define our minimum link path.

This still does leave questions about how to calculate distance of triangles from d along with algorithms for calculating $V(e_i)$ or $W(e_i)$. These methods are rather complex and would be drawn out to explain but can be seen in Suri's paper [10].

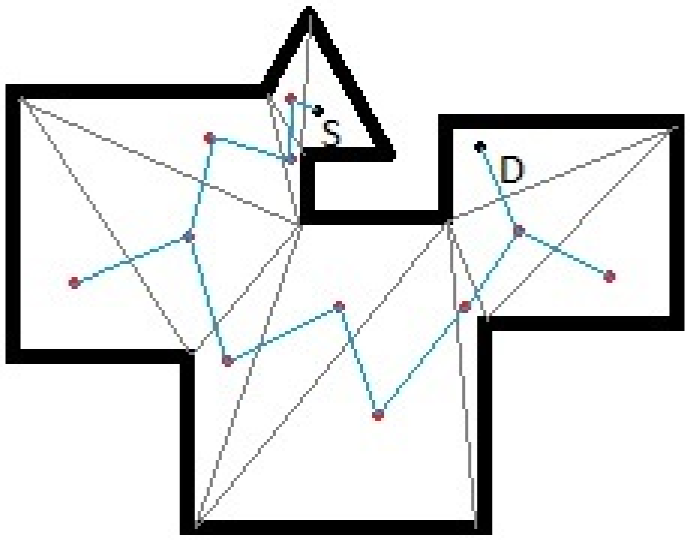


Figure 3.32: Minimum Link Example Part 2

He outlines an additional algorithm for calculating $V(e_i)$ along with details of triangulation ordering withing $T_{\{1,k\}}$.

We could apply this to our first turn by creating a triangulation of our entire course as P and compute $V(s)$ from our vehicle's starting point. However, looping through all of the steps along with hundreds of calculations for windows and visibility polygons would make this very exhaustive, but could be a next step for this project.

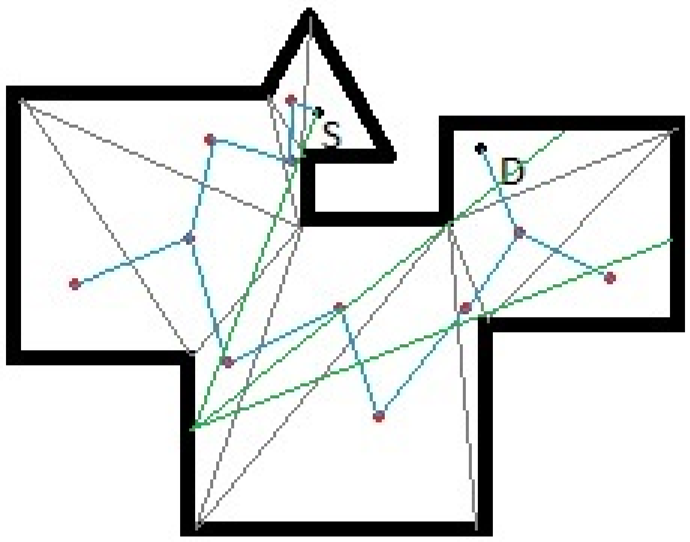


Figure 3.33: Minimum Link Example Part 3

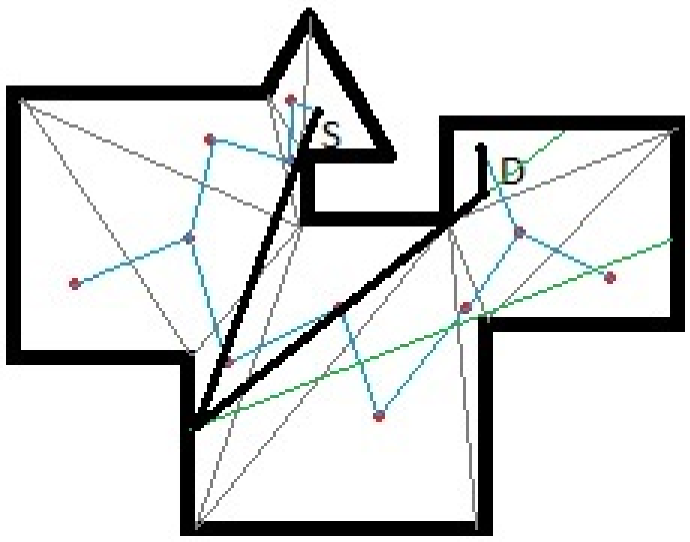


Figure 3.34: Minimum Link Example Part 4

CHAPTER 4

Conclusion of Mathematical Modeling

4.1 Continuation of Modeling

Unfortunately due to the scope of this project I was unable to really explore each of the following subsequent turns appropriately. We were also unable to truly optimize it properly due to a lack of time to create the proper computer simulations. That being said, I was successful in creating a formula that would allow for any turn to be optimized. The steps would be as follows:

1. Calculate how many miniturbos can be charged feasibly. For each, long would the distance traveled be to charge said miniturbo?
2. Identify the coordinate where the drift following the one you wish to calculate will begin.
3. Determine the most optimal time to release miniturbo to properly align for the next drift.
4. Using the start and end point of the drift, along with the tangent line to the beginning of the next drift, create an equation for the circle the arc is on and determine the distance traveled on that arc.

5. Through Monte Carlo or reinforcement learning determine a large finite set of possible paths and determine which has the shortest/least weighted path..

While this process is a fair bit complicated, it ultimately should not vary much between any of the turns. We can start by giving a brief visual overview for all the turns in a sequence, then we can show how each of the turns would vary in terms of computation in comparison to our 1st turn:

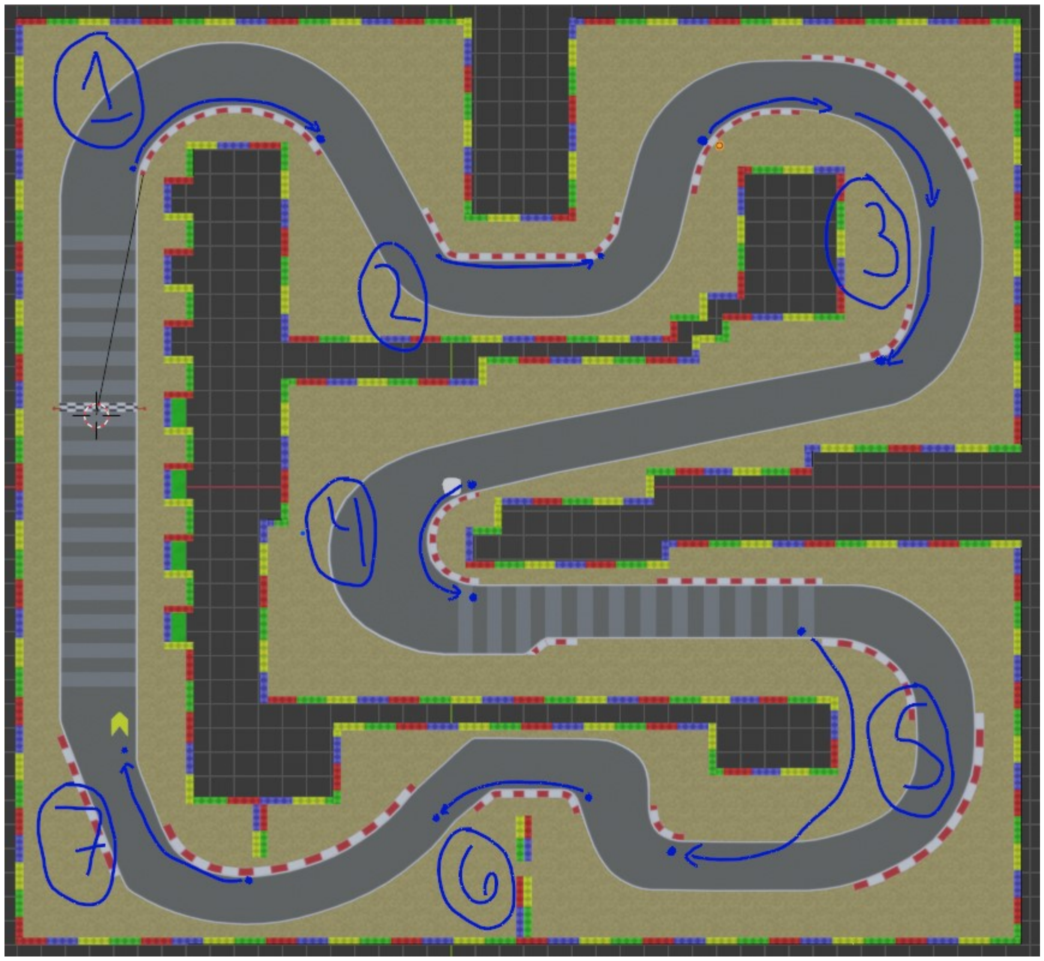


Figure 4.1: Overview of All Turns in a Sequence

- Turn 2: This turn is almost exactly identical to Turn 1, however there is a pipe at the beginning and end of the turn. All this means is that we would need to slightly vary our start and end point to ensure that whatever our calculated

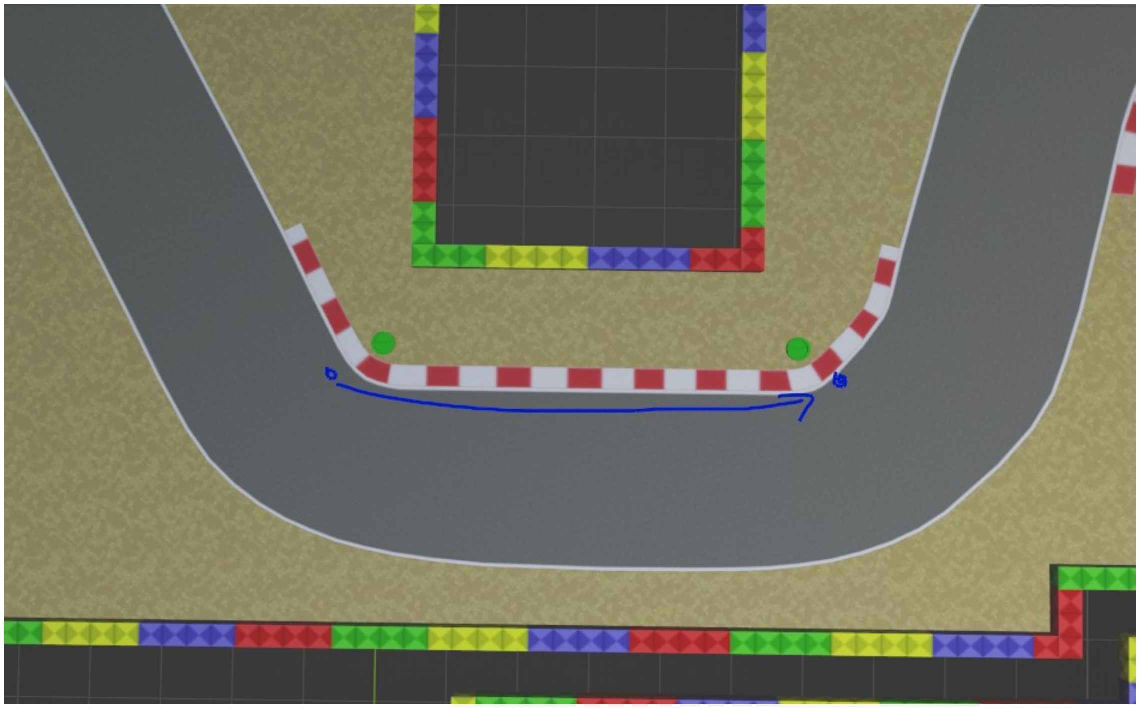


Figure 4.2: Overview of Turn 2

path is would not drive through any of the pipes, rather as tight to them as possible. We could do this simply be imagining a remodel of the track where we remove all parts of the road where our vehicle could hit either pipe, and then calculate the optimal path for the turn with the remaining road.

- Turn 3: This would largely be a difference in terms of the amount of miniturbos. This turn it is possible to get 3 miniturbos out while remaining optimally tight to the turn due to how long the turn is. This means that we would need to calculate the change in velocity for each of the released miniturbos when factoring in the distance our vehicle travels on the turn.

- Turn 4: This turn is incredibly tight, which means our stick position will be all the way to the left for the entire duration of the drift to try and take it as optimally as possible. This means using a hypothetical trajectory we would then have to calculate only the position of the starting and ending points, which would be relatively easy in comparison to any of the other turns.

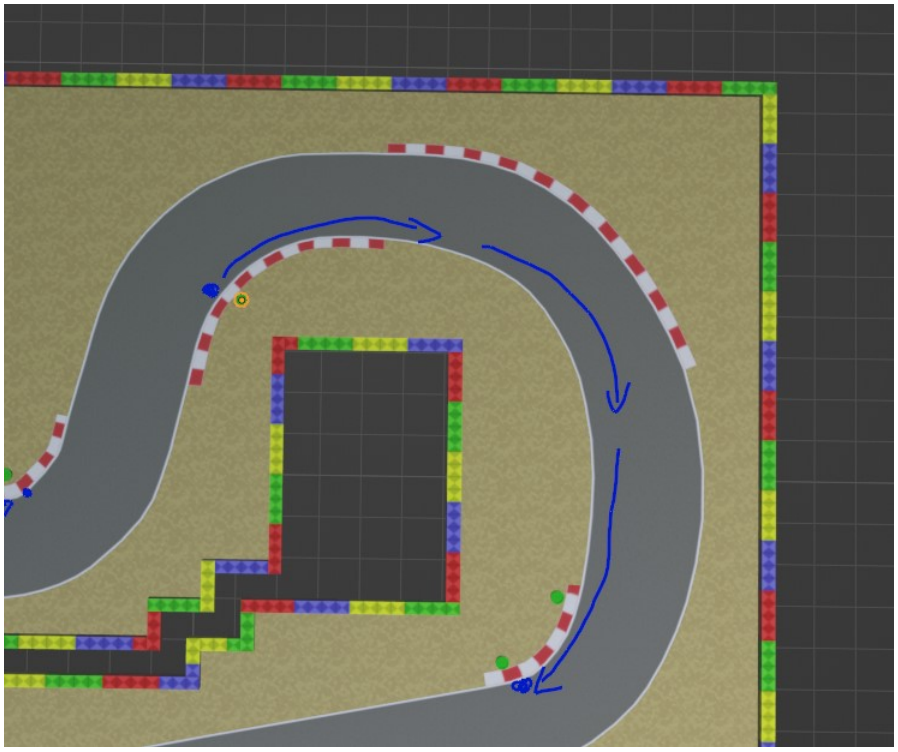


Figure 4.3: Overview of Turn 3

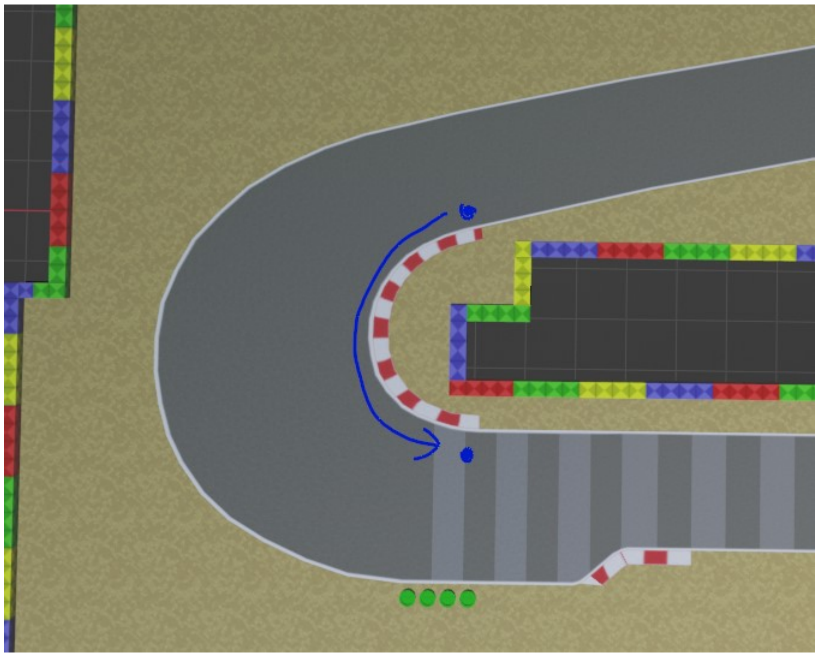


Figure 4.4: Overview of Turn 4

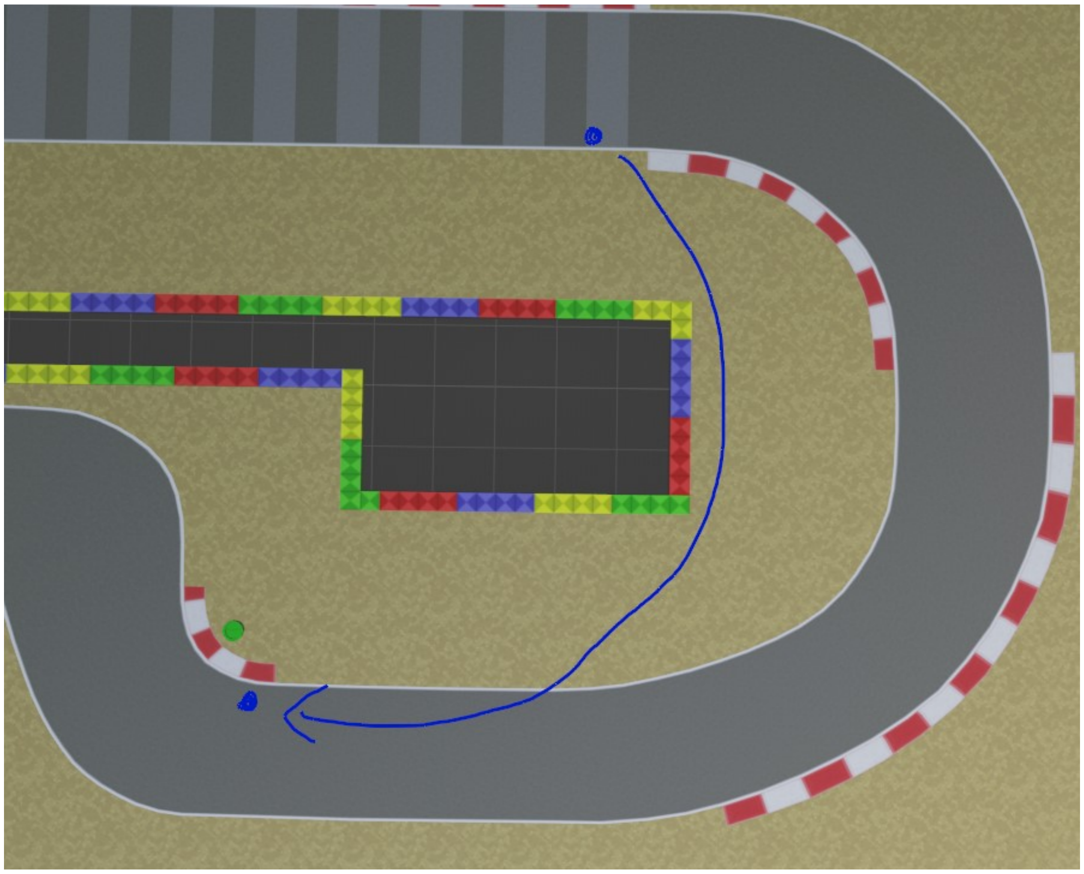


Figure 4.5: Overview of Turn 5

- Turn 5: We would have to recalculate how fast our vehicle turns here as for a large duration of the turn it would be in a boost (or more precisely a mushroom boost). This means we would need to calculate acceleration and distance traveled in the shroom as well. Miniturbo calculations would not be a problem, similar to turn 1, as the turn is so long we can easily charge one. While it would be possible to charge two miniturbos, it would be impractical because we would have to release one while we are in a boost, and since accelerations do not stack this would be pointless.

- Turn 6: This is very similar to Turn 2 but with a twist. There are pipes on both the beginning and back of the turn where we would remodel a road for the vehicle, but also we would be primarily soft drifting due to the relatively small

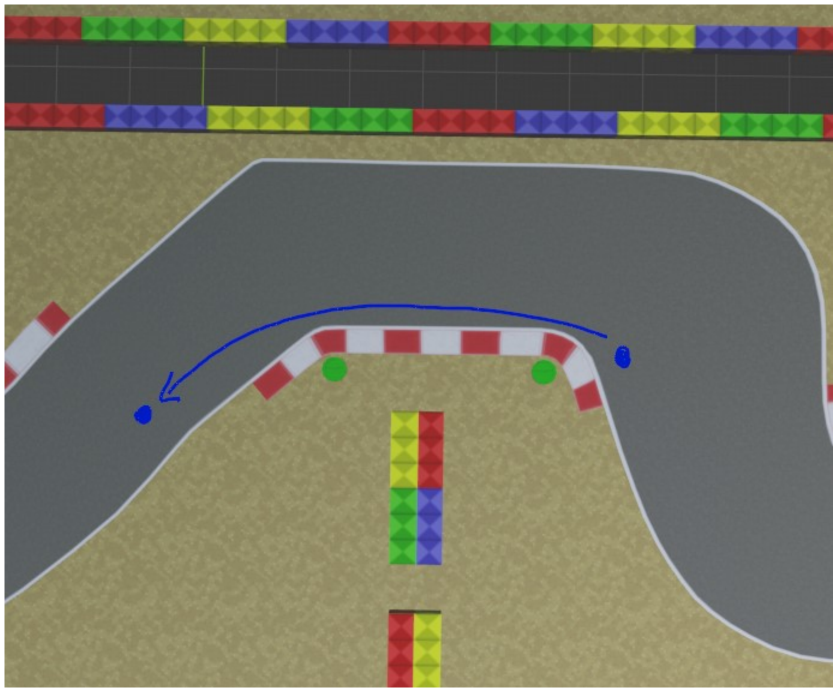


Figure 4.6: Overview of Turn 6

change in angle for the turn. This means our start and end points would be determined by how far the vehicle travels while charging a miniturbo with a soft drift, unlike any of the above problems. We would only need to calculate the start point, the ending point would be derived naturally from that.

- Turn 7: Our final turn sort of combines it all together. There is a pipe which we must avoid, an incredibly tight part of the turn at the beginning, along with a boost panel at the end which applied the boost. While we can certainly just remodel again to compensate for the positioning of the pipes, along with assume the drift to be a hard drift, calculating a start and end point proves challenging. We want to contact the boost panel relatively early as it greatly accelerates us, but we would need to calculate how far we would need to go out of the way to hit it optimally. To discover this, we would need to calculate the turn angle of our vehicle when in a boost and wheelie, and then determine how much it can change its angle for the duration of the boost so it can properly

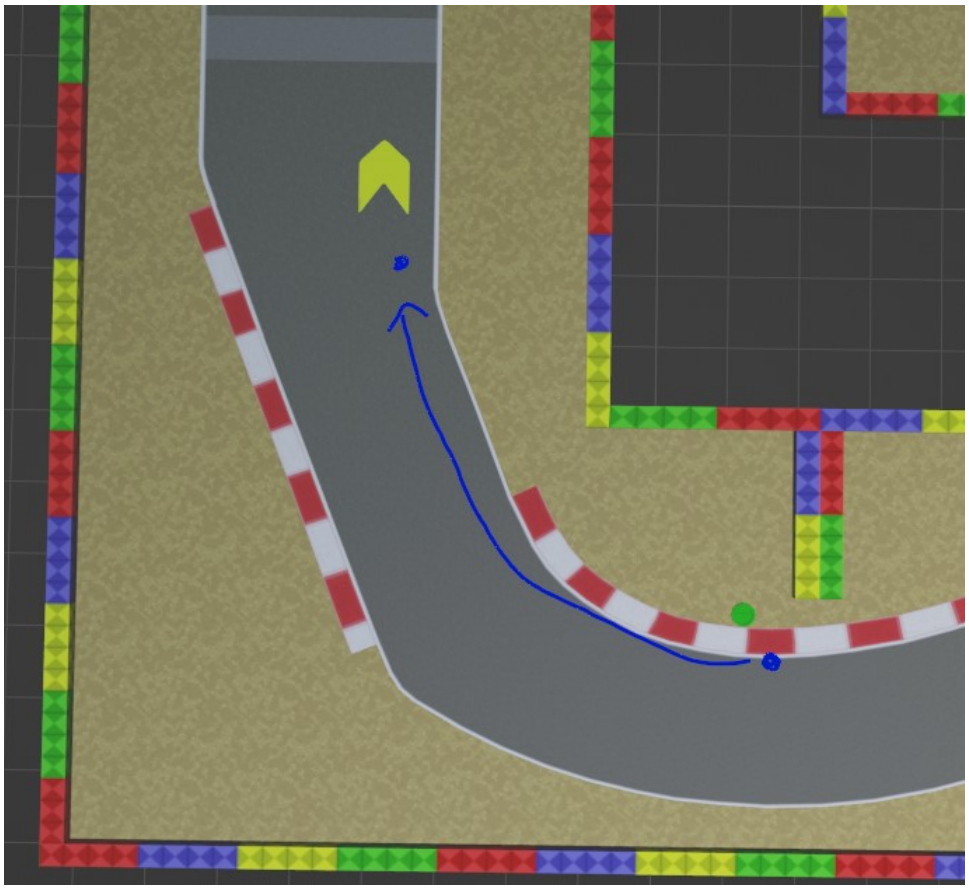


Figure 4.7: Overview of Turn 7

aim at the initial point for Turn 1 on lap 2. We would also combine it with the distance between the initial point on the path and any point that touches the nearest edges of the boost panel. This would likely involve another non-linear system, but ultimately should not be too challenging to calculate.

With this laid out anyone can calculate this and find our total time to complete one lap, however it would take a very large amount of time that the scope of this Independent Study project was just unable to accommodate.

4.2 Modeling other race tracks

There are 32 total tracks in Mario Kart Wii, and for this project we only focused on one of them. The main reason we chose to focus on this one was because it is relatively flat and simplistic, without a lot of worrying for an added dimension. The removal of a dimension for elevated ground (commonly referred to as a z coordinate) makes the project much more simple by itself, and that is one of the several features which we would need to begin accounting for when we start our calculations. That being said, the remaining 31 tracks are all much more complex, some more so than others, but due to this complexity several issues arise that would require substantially more complex mathematics and calculations.

One such example is elevated edges. This is very prominent on Luigi's Circuit, which can be seen in the figure below. These elevated edges are graphically represented by extruding the faces along the edge of the road, and then beveling them partly. Not only would this add another dimension of calculations, but it would also require us calculating distances along faces of complex 3 dimensional polyhedrons.

Another example is tricking. Every other course in the game has tricking involved, which certainly complicates things. Not only would we have to calculate even more changes in velocity and angular momentum, but also we would have to calculate air time. All tricks in the game result in air time ranging from a couple seconds to several milliseconds. During this time we would have to approximate our position function as a 3 dimensional vector, and to compound this another layer of deceleration occurs for vehicles in midair.

Those were just two of the most basic examples that are pretty widespread across each of the other courses in Mario Kart Wii, there are certainly many more not mentioned here that an attentive reader can imply from some of what was earlier written in Chapter 2.

4.3 Final Thoughts

This thesis strayed well from what was originally designed, but ultimately I am very happy with the outcome of it. While there was not necessarily a large amount of in-depth mathematics, there was a very substantial amount of computational work that was done to make application of the mathematical theory possible. There certainly is great room potential for continuation of this work, and I hope to persevere through this project after graduation.

References

1. Davood Asadi and Ella Atkins. Multi-objective weight optimization for trajectory planning of an airplane with structural damage. *Journal of Intelligent Robotic Systems*, 91, 09 2018. doi: 10.1007/s10846-017-0753-9. 6

2. Danny Z. Chen, Kevin S. Klenk, and Hung-Yi T. Tu. Shortest path queries among weighted obstacles in the rectilinear plane. In *Proceedings of the Eleventh Annual Symposium on Computational Geometry*, SCG '95, page 370–379, New York, NY, USA, 1995. Association for Computing Machinery. ISBN 0897917243. doi: 10.1145/220279.220319. URL https://doi.org/10.1145/220279.220319. 3

3. EjayB and Citrinitas. Discord conversation with ejayb and citrinitas, Oct 2022. 10, 12, 17

4. John Hershberger and Subhash Suri. An optimal algorithm for euclidean shortest paths in the plane. *SIAM Journal on Computing*, 28(6):2215–2256, 1999. doi: 10.1137/S0097539795289604. URL https://doi.org/10.1137/S0097539795289604. 3

5. Kierio. Everything there is to know about mkwii tas, Jul 2020. URL https://docs.google.com/document/u/1/d/e/2PACX-1vRZObe4loAptsyRU5Ba-k0WdNQPnT_1DhG_r4H7wKZFm6BsKs28aPREV_649xTRT2cPZdz26GOB3zbR/pub. 12, 14, 15

6. K Melhourn. a multi-dimensional searching and computational geometry. *EATCS Monographs on Theoretical Computer Science*, 3, 1984. 74

7. Joseph S. B. Mitchell and Christos H. Papadimitriou. The weighted region problem: Finding shortest paths through a weighted planar subdivision. *J. ACM*, 38(1):18–73, jan 1991. ISSN 0004-5411. doi: 10.1145/102782.102784. URL https://doi.org/10.1145/102782.102784. 5

8. A. Scheuer and C. Laugier. Planning sub-optimal and continuous-curvature paths for car-like robots. 1:25–31 vol.1, 1998. doi: 10.1109/IROS.1998.724591. 6, 7, 8

9. Eilon Solan and Nicolas Vielle. Deterministic Multi-Player Dynkin Games. Discussion Papers 1355, Northwestern University, Center for Mathematical Studies in Economics and Management Science, September 2002. URL https://ideas.repec.org/p/nwu/cmsems/1355.html. 9

10. Subhash Suri. A linear time algorithm for minimum link paths inside a simple polygon. *Computer Vision, Graphics, and Image Processing*, 35(1):99–110, 1986. doi: 10.1016/0734-189x(86)90127-1. 72, 73, 76