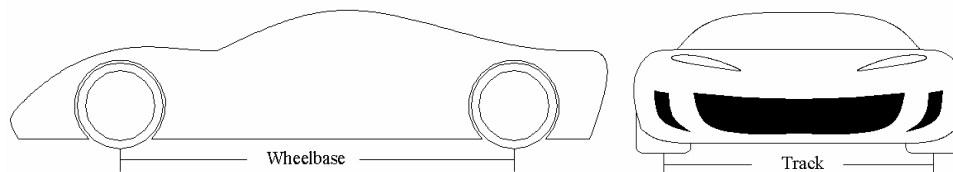


The Effects of Wheelbase and Track on Vehicle Dynamics

Automotive vehicles move by delivering rotational forces from the engine to wheels. The wheels push in the opposite direction of the motion of the car, which contracts and stretches the tires to deliver the engine force to the ground. Under acceleration the car shifts its weight in the direction opposite the motion. The weight transfer occurs because of the inertia of the car, which tries to keep the car at a constant velocity. Such weight transfer acts as a pivotal characteristic of the car's ability to be agile and responsive.

Two factors that affect how much weight is transferred during acceleration are the wheelbase and track of a car. The wheelbase is the distance between the front and rear axles, while track is the distance between the left and right wheels of an axle. These two factors are illustrated in the figure below. Another factor that affects weight transfer is the height of the center of gravity, but I am more interested on the effects of wheelbase and track.



It is generally accepted that, when the center of gravity is lower, less weight transfer occurs. I hypothesize that increased length of the wheelbase and track will result in even less weight transfer and will increase the acceleration and agility of the car. There are several ways to test my hypothesis, such as using cars with varying wheelbases and track. But there are drawbacks to this method. Measuring the results of such experimentation requires sophisticated equipment that costs quite a bit. Also controls are

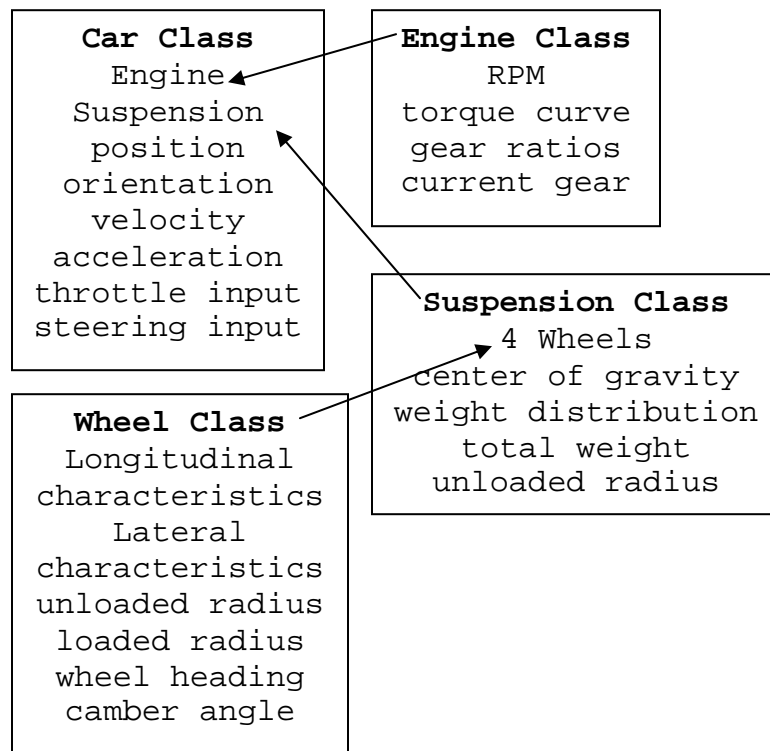
hard to maintain in such experimentation. The two variables of this experiment are wheelbase and track only, thus all other conditions must remain constant. Unfortunately it is nearly impossible to find multiple cars with the same characteristics except for wheelbase and track. Different models typically have different engine and handling characteristics, thus no controls may exist.

Instead of attempting to seek out nearly identical cars and expensive equipment, I plan to build a computer simulation to act as the environment and the car. This gives me a stable controlled environment in which I can manipulate hold everything constant except for the variables. The simulation would also cost much less. So I will a computer simulation to test the effects of different wheelbases and track on the traction of the car.

Before I start programming the simulation, I must first develop a plan for it. In order to build a structured program, the rules of object-oriented programming must be followed. Object-oriented programs are mainly driven by separate entities known as objects. One of the rules of object-oriented programming is encapsulation, which is keeping the functions and characteristics of an entity together in one class. To enforce encapsulation I shall use several classes to represent the different entities of the simulation. A class in computer programming is a collection of related functions and information. Classes in computer programming have member functions and member variables. Member functions perform a certain task or returns a value while member variables store information about the instance of the class. An instance is a copy of the class. Classes are like blueprints of a car. The classes are used to build actual objects that can be used by a program, but the class itself cannot be used as an object. Classes also

contain a method called a constructor where all its member variables are assigned some initial value [5].

The largest class of the simulation is the Car class. This represents the car entity of the simulation. It brings together several subclasses and manages the cooperation between these classes. The Car class also contains the general information about the car under testing. This information will likely be loaded from data files and transferred to the respective subclasses. The following diagram illustrates the class structure of the simulation and each class's member variables. The member functions of each class are too numerous for this diagram.



One of the subclasses, the Engine class represents the engine of the virtual car. Its chief function is to determine the current torque output of the engine. It will also track

the progress of the current engine speed in revolutions per minute (RPM). Since the engine and transmission are not the focus of this experiment, they will not be treated with much precision. Thus to simplify the program I will combine the engine and transmission entities together into a single `Engine` class. The transmission part of the `Engine` class will simulate the functions of the transmission by multiplying the torque output of the engine part with the respective gear ratio.

The other class in the `Car` class, the `Suspension` class, represents the whole suspension system of the car. It accounts for the springs, control arms and all other parts that constitute a car's suspension system. The main function of the `Suspension` class is to determine the weight transfer of the car under different circumstances. At constant velocity the weight of the car is evenly distributed on the wheels of the car, but during acceleration, which is a change of speed or direction, the car's inertia resists the motion and some of the sprung weight of the car shifts to the opposite direction of the acceleration. Velocity is a vector quantity with the speed as its magnitude, so when it is constant both the speed and the direction of the object stay the same. The sprung weight of a car is the amount of its weight that sits on the suspension system. The suspension class also contains the four `Wheel` objects of the car. Thus the `Suspension` class determines how much of the car's weight each wheel receives and also determines the total force generated by the individual `Wheel` objects. The two variables of this experiment, wheelbase and track, have most of their influence in this class [3].

The four `Wheel` classes represent each of the wheel-tire combinations. These objects determine how much traction each wheel obtains based on the amount of weight they receive, how much torque the engine delivers and the current speed of the car. The

tire delivers a backward force on the ground by bending and flexing in order to propel the car forward. But if the wheel receives too much torque the wheel loses traction and the car will understeer or worse lose control. Understeering is a phenomenon where loss of traction causes the car to slide towards the outside of the curve. Oversteering may also occur, where the car slides towards the inside of the curve. Both are undesired results, which may cause the driver to lose control of the car. Thus the `Wheel` object uses data from the other subclasses of the `Car` class to determine the longitudinal and lateral forces generated by the tires [1, 2].

Now I must implement the planned classes into an actual simulation. The first class of the simulation to be implemented is the `Engine` class. An engine creates torque output by firing cylinders and causing the crankshaft to spin. A torque is a spinning force that revolves around a certain point. This rotational motion of the crankshaft is then sent through the transmission, which multiplies the engine's torque output. Then the rotational motion finally travels through the axle of the car to the wheels causing them to spin. The speed the engine is currently spinning at can be measured in the number of revolutions per minute (RPM). The engine has different torque outputs for different engine speeds. The torque output can be modeled as a function of the engine speed. This graph is called a torque curve and provides a graphical description of the engine's torque output. The simulation program must simulate the torque curve in order for the values of acceleration to be at least slightly accurate. The `Engine` class has a member variable that tracks the engine speed. A member function named `updateRPM` is used to determine the current engine speed. This class also has member variables that hold the gear ratios, which are loaded from a text file and a variable that keeps track of the current gear of the car.

The `updateRPM` function updates the current engine speed based on the throttle input. Thus this function takes in the `throttle` as a parameter. Parameters are information sent by the function that invokes this function. The function that invokes this function is known as the client. The engine of the car is not the focus of the project and will not receive great amounts of detail. The throttle input from the client is a decimal value that is clamped to range of [-1.0, 1.0]. In the throttle parameter 1.0 means a fully depressed accelerator pedal, -1.0 means that the brake pedal is fully depressed and 0.0 means that neither pedal is being depressed. For the virtual car in my simulation I use the following formula to model the change in the engine speed:

$$S_{Ef} = S_{Ei} + (g * 3000 * t_{delta})$$

where S_{Ef} is the new engine speed in RPM, S_{Ei} is the former engine speed in RPM, g is the throttle parameter from the client and t_{delta} is also a member of the `Engine` class that is a constant that has been initialized in the `Engine` class constructor. It is the time between each execution of this function. Thus the engine speed increases or decreases by 3000 RPM per second depending on whether the gas or brake pedal is depressed and how much the particular pedal has been depressed. Once the current engine speed has been determined, the engine must also determine the current transmission gear. A car's transmission is used to multiply the torque output of the engine. At lower gears the transmission multiplies the engine's torque at the expense of speed by spinning gears slower than the engine. At higher gears the transmission gears spin faster to produce higher speeds at the expense of the torque output. The car needs high torque output to start the car, but it needs more speed as it spins faster. Thus to simulate the gear shifts of my virtual car the engine speed must be kept on a particular range. In my virtual car I

keep the RPM between 1000 and 6000 RPM. If the engine speed reaches 6000 RPM, then the transmission will shift up one gear, which causes the engine speed to drop. If the engine speed reaches 1000 RPM, then the gear will be shifted down in order to raise the RPM. The gear change mechanism is much more complex in a real car, but since this is not the focus of this experiment, it will not receive more attention to detail.

The next function, `getEngineTorque`, can now determine the torque output of the engine based on the value computed from `updateRPM`. I made a text file to with the torque output for every 50 RPM between 1000 and 6000 RPM. The simulation uses this text file to look up the torque output at the current RPM. If the current RPM is between two torque values, then the function averages the two closest values to determine the torque output.

The `getWheelTorque` function acts as the virtual car's transmission and determines the torque output at the wheel. As mentioned earlier the transmission multiplies the engine's torque output, and thus this function uses the following equation:

$$T_W = T_E * G_{CUR} * G_{FINAL}$$

where T_W is the torque output at the axle, T_E is the engine's torque output, G_{CUR} is the current gear ratio and G_{FINAL} is the final drive ratio of the axle. The engine and transmission are again much more complex, but since they are not the focus of this simulation, they do not require much detail [2].

The next class of the simulation program is the `Suspension` class. This class determines how much of the weight is distributed to each wheel. This is where the wheelbase and track (the variables of this experiment) have the greatest effect. As stated earlier in this report the shifting of weight during acceleration is due to the inertia of the

car. More weight is shifted to the wheels opposite the direction of the acceleration. Thus when the car accelerates forward, weight is shifted from the front wheels to the rear wheels. If the car is turning right, then the two left wheels receive more weight than the two right wheels. In this scenario centripetal force is acting upon the car. Centripetal force is a force that keeps an object moving in a circular motion. For example, if a pebble is attached to the end of the string, and this is spun in a horizontal circle, what prevents the pebble from traveling in a straight line is the centripetal force, which is acting as the tension on the string in this example. Centripetal force always acts towards the center of the circle and keeps an object moving in a circular manner by changing the direction of its velocity but not its speed. On a car going taking a corner what makes the car turn is the centripetal force, which in this case is a frictional force acting on the tires. Frictional forces resist the sliding of two surfaces against each other, such as the tire and the ground. Most cars do not have weight equally distributed over the four wheels even at constant velocity. To express the weight distribution of such cars the F/R notation is used, where F is the percentage of the car's weight that rests on the front axle and R is the percentage that rests on the rear axle at constant velocity. To maintain simplicity in my simulation, my virtual car will have an ideal 50/50 weight distribution. Also the springs of the suspension system can be safely ignored. The springs resist shifting of weight during acceleration, but this is a complex system to simulate and requires great processor speeds from the computer. The results of the executions of the simulation program are used for comparison with each other. Since the springs do not affect any of the results and acts as a constant, it can be canceled out of the equation.

This class contains member variables that describe the suspension system. It contains variables that describe weight distribution at constant velocity, the overall weight of the car and most importantly the wheelbase and the track. The `Suspension` class also has a reference to an `Engine` object. A reference is a member variable that allows objects to communicate with each other, such as the `Suspension` and `Engine` classes. The `Suspension` class also has four `Wheel` objects that represent the wheels of the car. The implementation of the `Wheel` class will be discussed later in this report.

The member function, `determineForce`, computes the amount of the car's weight that is distributed to each axle and thus plays the central role of this class. First this function uses its reference to an `Engine` object to determine the force that the all the wheels deliver to the ground. The torque force of the axle is converted to a linear force by the wheel when it pushes backwards on the ground, which pushes the car forward. This is true according to Newton's Third Law of Motion, which states that for every force there is a force with equal magnitude but in the opposite direction. The following equation is used to determine the force that the wheel delivers on the ground:

$$F_w = T_w / r$$

where F_w is the force that the wheel exerts on the ground, T_w is the torque output at the wheel by the member function of the `Engine` object, `getWheelTorque` and r is the radius of the tire-wheel combination. This presents the longitudinal and lateral components of the force generated by the car and so is the magnitude of the force. Longitudinal is the direction that runs lengthwise through the car (forward and backward), while lateral is the direction the runs widthwise (left and right) through the car. Another parameter to this method is the steering angle, which is how much the

wheels have been rotated from the default bearing. With this angle the two components of the force can be calculated, but first the angle must be converted to radians according to the following equation:

$$\text{theta} = (90 - \text{steering}) * \text{PI} / 180$$

where theta is the angle in radians, Steering is the steering angle in degrees and PI is the constant 3.14159. steering is subtracted from 90 because the normal bearing of the steering is on the positive y-axis. After the angle has been calculated then the following formula can be used to determine the separate components of the force:

$$F_x = F * \cos(\text{theta})$$

$$F_y = F * \sin(\text{theta})$$

where F_x is the lateral component of the force, F_y is the longitudinal component of the force and F is the magnitude of the force. Now that the components of the force have been determined the following equations are used to determine the weight distribution in the longitudinal direction caused by the longitudinal component of the net force.

$$W_f = f * G - F_y * h / R$$

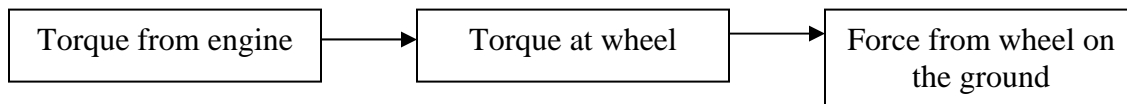
$$W_r = r * G + F_y * h / R$$

where W_f is the weight on the front axle, W_r is the weight on the rear axle, f is front weight distribution at constant velocity, r is the rear weight distribution at constant velocity, G is the total weight of the car, F_y is the longitudinal force acting on the car, h is the height of the center of gravity and R is the wheelbase. This is where the wheelbase, one of the variables of the experiment, has its effect. From this equation I can infer that the longer the wheelbase is, the less weight is transferred during acceleration. The same rules apply to the later component:

$$W = 0.5 * G +- F_x * h / T$$

where W is the amount of weight that is on the wheel, G is the total weight of the car, h is the height of the center of gravity and T is the length of the track of the car. The additional weight is either added or subtracted based which the car is turning.

After it has determined the weight distribution, the function `determineForce`, sends the weight on each wheel to the respective `Wheel` object along with the current velocity, which is calculated from the ideal force. The `Wheel` objects then return how much force they produce and push against the ground with. This function adds all of the results from the four wheels and returns the total force to the client function [1, 4]. The following figure illustrates the flow of data so far in the simulation.



The last major class in this simulation is the `Wheel` class, which determines how much force each wheel generates based on the weight on them. The tire generates forces by pushing and pulling on the ground and thus needs to “stick” to the ground. How much the tires stick to the ground is called traction. The simple friction formula,

$$F_A \leq \mu_s * F_N$$

cannot be used to simulate a realistic physics engine. When a car reaches its maximum friction it starts to lost traction before it actually reaches peak frictional force. Also this model does not apply to a car that has both longitudinal and lateral forces acting on it at the same time. The amounts of longitudinal and lateral forces acting at the same time are inversely proportional to each other. Thus when the car is fully accelerating, it cannot

turn. Trying to do so would result in skidding of all the wheels. Tire dynamics is a vague area of study due to trade secrecy by tire manufacturers and ongoing research, but in my simulation I try to simulate them as much as possible.

The `Wheel` class contains all the information about the wheel of the virtual car. It has the radius of the wheel-tire combination and the longitudinal and lateral characteristics of the tire.

This class computes the longitudinal and lateral forces separately. First the `getLongitudinalWheelForce` function determines the longitudinal force generated by the tires. The longitudinal characteristics for my virtual car are as follows:

b_0	1.65	dimensionless	b_6	0	$1/(\text{Kilo Newton})^2$
b_1	0	1/MegaNewton	b_7	0	1/KiloNewton
b_2	1688	1/Kilo	b_8	-10	dimensionless
b_3	0	1/MegaNewton	b_9	0	1/KiloNewton
b_4	229	1/Kilo	b_{10}	0	dimensionless
b_5	0	1/KiloNewton			

Longitudinal tire characteristics. Source: [1]

The nature of these characteristics is still under research and since the suspension is the primary concern of this research project, tire dynamics shall not be dealt with in great detail. These characteristics are used in the following formula to determine the longitudinal force:

$$\begin{aligned}
 F_x(\sigma, F_r) &= D \sin\left(b_0 \tan^{-1}\left\{SB + E\left[\tan^{-1}(SB) - SB\right]\right\}\right) + S_x \\
 D &= \mu_{xy} F_r = (b_1 F_r + b_2) F_r, \quad B = (b_3 F_r + b_4) \exp(-b_5 F_r) / b_0 (b_1 F_r + b_2) \\
 E &= (b_6 F_r^2 + b_7 F_r + b_8), \quad S = (100 \sigma + b_9 F_r + b_{10}), \quad S_x = \text{constant}
 \end{aligned}$$

Longitudinal force formula. Source: [1]

The two parameters to this formula are the weight on the tire and the longitudinal slip ratio. This is determined by the following formula:

$$\sigma = \frac{\omega}{\omega_0} - 1 = \frac{\omega}{V/R_e} - 1 = \frac{\omega R_e - V}{V}$$

Longitudinal slip ratio formula. Source: [1]

where V is the current speed of the car and

$$\omega = V/R \text{ and } \omega_0 = V/R_e$$

Loaded and free-rolling angular velocity: Source [1]

where R is the loaded radius of the wheel. During acceleration the driving wheels come under heavy stress and they bend. Thus the radius of these wheels decrease and are smaller than R_e , which is the unloaded radius of the tire-wheel combination [1, 2].

Most of the rules apply to get `LateralWheelForce` function, which determines the lateral wheel force. This function uses the following lateral tire characteristics:

a_0	1.799	dimensionless	a_7	1	dimensionless
a_1	0	1/MN	a_8	0	dimensionless
a_2	1688	1/Kilo	a_9	-6.111/1000	Degree/KN
a_3	4140	N	a_{10}	-3.224/100	Degree
a_4	6.026	KN	a_{11}	0	1/MN - Degree
a_5	0	1/Degree	a_{12}	0	1/Kilo Degree
a_6	-0.3589	KN	a_{13}	0	1/Kilo
			a_{14}	0	N

Lateral tire characteristics. Source [1]

This function uses the following formula:

$$F_y(\alpha, F_z, \gamma) = D \sin\left(\alpha_0 \tan^{-1}\left\{SB + E\left[\tan^{-1}(SB) - SB\right]\right\}\right) + S_v$$

$$D = \mu_{\text{eff}} F_z = (\alpha_1 F_z + \alpha_2) F_z, \quad B = a_3 \sin\left[2 \tan^{-1}(F_z / a_4)\right] (1 - \alpha_5 |\gamma|) / \alpha_0 (\alpha_1 F_z + \alpha_2) F_z$$

$$E = \alpha_6 F_z + \alpha_7, \quad S = \alpha_{\text{degrees}} + \alpha_8 \gamma_{\text{degrees}} + \alpha_9 F_z + \alpha_{10}$$

$$S_v = \left[(\alpha_{11,1} F_z + \alpha_{11,2}) \gamma_{\text{degrees}} + \alpha_{12} \right] F_z + \alpha_{13}$$

Lateral force formula. Source: [1]

This formula has three parameters: the slip angle, the weight on the wheel and the camber angle respectively. The camber angle, γ , is assigned a value of 0.0 for this experiment, since it does not deal with camber angle. The slip angle is computed by the following formula:

$$\alpha = \tan^{-1}\left(\frac{L_y}{L_x - W_x}\right)$$

Slip angle formula. Source: [1]

where W is the velocity of the contact patch with respect to the hub of the wheel. The contact patch is the area of the tire that is touching the ground and the hub is the center of the wheel. L is the slip velocity and is defined by the formula:

$$L = V + W$$

where V is the velocity of the car.

The final function of the `Wheel` class is the `getWheelForce` function, which combines the lateral and longitudinal force formulas with the following formula:

$$\rho = +\sqrt{s^2 + a^2}, \quad F_x^*(s, a) = \frac{s}{\rho} \Phi_x(\rho), \quad F_y^*(s, a) = \frac{a}{\rho} \Phi_y(\rho)$$

Combination grip formula. Source: [1].

With this formula the `Wheel` class returns the force delivered by each wheel [1].

Finally the `Car` class combines all the classes mentioned above into one entity. It contains an `Engine` and `Suspension` objects. It contains a loop, which simulates the progression of time through the simulation. Inside this loop it first invokes the `updateRPM` function of the `Engine` class and then invokes the `determineForce` function of the `Suspension` class. From the resulting force it can calculate velocity and position.

Now the program is complete and I can execute some testing scenarios. The first test I run is the acceleration test for 10 seconds with a wheelbase of .259 meters, which is about the wheelbase of a race car, and then with 2.56m and 2.62m wheelbases. Track does not affect longitudinal acceleration. Remember that these results are only used for comparison with each other and so are not very accurate if compared with real world examples. The first value of each entry is the lateral component and the second is the longitudinal component [1]. The following are results from the simulation, where distance is measured in meters and velocity is measured in meters per second.

Time elapsed: 1 seconds
VELOCITY: 0.0, 3.1227665735955186
POSITION: 0.0, 3.1227665735955186

Time elapsed: 2 seconds
VELOCITY: 0.0, 9.588698992223321
POSITION: 0.0, 12.71146556581884

Time elapsed: 3 seconds
VELOCITY: 0.0, 21.76344299924207
POSITION: 0.0, 31.35214199146539

Time elapsed: 4 seconds
VELOCITY: 0.0, 46.63389363370485
POSITION: 0.0, 68.39733663294692

Time elapsed: 5 seconds
VELOCITY: 0.0, 100.68518265239092
POSITION: 0.0, 147.31907628609576

Time elapsed: 6 seconds
VELOCITY: 0.0, 235.72661735363803
POSITION: 0.0, 336.41180000602895

Time elapsed: 7 seconds
VELOCITY: 0.0, 1352.6019826315903
POSITION: 0.0, 1588.3285999852283

Time elapsed: 8 seconds
VELOCITY: 0.0, 1023.479981177156
POSITION: 0.0, 2376.0819638087464

Time elapsed: 9 seconds
VELOCITY: 0.0, 681.9899769761327
POSITION: 0.0, 1705.4699581532886

Time elapsed: 10 seconds
VELOCITY: 0.0, 240.95760325614617
POSITION: 0.0, 922.9475802322788

Time elapsed: 1 seconds
VELOCITY: 0.0, 3.1363776263659613
POSITION: 0.0, 3.1363776263659613

Time elapsed: 2 seconds
VELOCITY: 0.0, 9.645310098814566
POSITION: 0.0, 12.781687725180527

Time elapsed: 3 seconds
VELOCITY: 0.0, 21.926045031323042
POSITION: 0.0, 31.571355130137608

Time elapsed: 4 seconds
VELOCITY: 0.0, 47.06832259375536
POSITION: 0.0, 68.99436762507841

Time elapsed: 5 seconds
VELOCITY: 0.0, 101.87604345138794
POSITION: 0.0, 148.9443660451433

Time elapsed: 6 seconds
VELOCITY: 0.0, 239.69615177546183
POSITION: 0.0, 341.5721952268498

Time elapsed: 7 seconds
VELOCITY: 0.0, 1471.2828006986874
POSITION: 0.0, 1710.9789524741493

Time elapsed: 8 seconds
VELOCITY: 0.0, 1148.6578225785074
POSITION: 0.0, 2619.940623277195

Time elapsed: 9 seconds
VELOCITY: 0.0, 821.1287314682322
POSITION: 0.0, 1969.7865540467396

Time elapsed: 10 seconds
VELOCITY: 0.0, 430.4820492240519
POSITION: 0.0, 1251.6107806922842

Acceleration with 2.59m wheel base.

Acceleration with 2.56m wheel base.

Time elapsed: 1 seconds
VELOCITY: 0.0, 3.1094672243236356
POSITION: 0.0, 3.1094672243236356

Time elapsed: 2 seconds
VELOCITY: 0.0, 9.533511412483254
POSITION: 0.0, 12.64297863680689

Time elapsed: 3 seconds
VELOCITY: 0.0, 21.60528483029802
POSITION: 0.0, 31.138796242781275

Time elapsed: 4 seconds
VELOCITY: 0.0, 46.21240053384166
POSITION: 0.0, 67.81768536413968

Time elapsed: 5 seconds
VELOCITY: 0.0, 99.53411545579938
POSITION: 0.0, 145.74651598964104

Time elapsed: 6 seconds
VELOCITY: 0.0, 231.92632283894494
POSITION: 0.0, 331.46043829474434

Time elapsed: 7 seconds
VELOCITY: 0.0, 1254.3801130223067
POSITION: 0.0, 1486.3064358612517

Time elapsed: 8 seconds
VELOCITY: 0.0, 918.6671931402193
POSITION: 0.0, 2173.047306162526

Time elapsed: 9 seconds
VELOCITY: 0.0, 561.101467245803
POSITION: 0.0, 1479.7686603860222

Time elapsed: 10 seconds
VELOCITY: 0.0, 32.4554005698933
POSITION: 0.0, 593.5568678156963

Acceleration with 2.62m wheel base.

From these results I can deduce that when the wheelbase is shorter more weight transfer will occur, which will result in quicker acceleration. The next test determines the effect of wheelbase when taking a curve. The throttle input remains the same as last time but now the steering angle 5.0°.

Time elapsed: 1 seconds
 VELOCITY: 1.0451154990407792, 3.1063645933622164
 POSITION: 1.0451154990407792, 3.1063645933622164

Time elapsed: 2 seconds
 VELOCITY: 0.02262153746765172, 9.533181356207432
 POSITION: 1.0677370365084309, 12.639545949569648

Time elapsed: 3 seconds
 VELOCITY: 0.7917542021743128, 21.62472972121006
 POSITION: 1.8594912386827436, 34.26427567077971

Time elapsed: 4 seconds
 VELOCITY: 0.049319301390379544, 46.300984607830394
 POSITION: 1.908810540073123, 80.56526027861011

Time elapsed: 5 seconds
 VELOCITY: 0.6188721160895133, 99.84341888139595
 POSITION: 2.527682656162636, 180.40867916000605

Time elapsed: 6 seconds
 VELOCITY: 0.1239154929824412, 233.06669321382589
 POSITION: 2.6515981491450775, 413.47537237383193

Time elapsed: 7 seconds
 VELOCITY: 0.42472499803844743, 1283.37403066068
 POSITION: 3.076323147183525, 1696.849403034512

Time elapsed: 8 seconds
 VELOCITY: 0.06905201863976468, 949.4446410279014
 POSITION: 3.1453751658232894, 2646.2940440624134

Time elapsed: 1 seconds
 VELOCITY: 1.0451154990407792, 3.1198717506734357
 POSITION: 1.0451154990407792, 3.1198717506734357

Time elapsed: 2 seconds
 VELOCITY: 0.022621537498306976, 9.589313661138716
 POSITION: 1.0677370365390861, 12.709185411812152

Time elapsed: 3 seconds
 VELOCITY: 0.7917542021338583, 21.78581252347548
 POSITION: 1.8594912386729443, 34.494997935287635

Time elapsed: 4 seconds
 VELOCITY: 0.04931930138090945, 46.73084972593198
 POSITION: 1.9088105400538538, 81.22584766121962

Time elapsed: 5 seconds
 VELOCITY: 0.6188721160533888, 101.01927984505681
 POSITION: 2.5276826561072427, 182.24512750627645

Time elapsed: 6 seconds
 VELOCITY: 0.1239154930854457, 236.9616236729086
 POSITION: 2.6515981491926883, 419.20675117918506

Time elapsed: 7 seconds
 VELOCITY: 0.4247249973917274, 1388.577763517981
 POSITION: 3.0763231465844156, 1807.7845146971663

Time elapsed: 8 seconds
 VELOCITY: 0.06905190719294357, 1061.2809924252042
 POSITION: 3.145375053777359, 2869.0655071223705

Accelerating curve with 2.59m wheelbase.

Accelerating curve with 2.56m wheelbase.

Time elapsed: 1 seconds
 VELOCITY: 1.0451154990407792, 3.0931667602642308
 POSITION: 1.0451154990407792, 3.0931667602642308

Time elapsed: 2 seconds
 VELOCITY: 0.022621537437555128, 9.478460132453423
 POSITION: 1.0677370364783343, 12.571626892717653

Time elapsed: 3 seconds
 VELOCITY: 0.7917542022140062, 21.46804677028151
 POSITION: 1.8594912386923403, 34.039673662999164

Time elapsed: 4 seconds
 VELOCITY: 0.04931930139972607, 45.88390731876102
 POSITION: 1.9088105400920665, 79.92358098176018

Time elapsed: 5 seconds
 VELOCITY: 0.6188721161244124, 98.7067797902042
 POSITION: 2.527682656216479, 178.63036077196438

Time elapsed: 6 seconds
 VELOCITY: 0.12391549288296577, 229.33696437233536
 POSITION: 2.6515981490994447, 407.9673251442997

Time elapsed: 7 seconds
 VELOCITY: 0.4247249986558662, 1195.387708383574
 POSITION: 3.076323147755311, 1603.3550335278737

Time elapsed: 8 seconds
 VELOCITY: 0.06905219548162383, 854.7218872853402
 POSITION: 3.145375343236935, 2458.076920813214

Accelerating curve with 2.62m wheelbase

From this set of results I can conclude that longer wheelbase result in understeer, when taking a corner while accelerating. The next test determines the effect of track on cornering ability. The next test is almost like the last test except that the wheelbase remains constant at 2.59m, but the track will differ. The throttle and steering input will be the same as the last test.

Time elapsed: 1 seconds
 VELOCITY: 1.0451154990407792, 3.1063645933622164
 POSITION: 1.0451154990407792, 3.1063645933622164

Time elapsed: 2 seconds
 VELOCITY: 0.02262153746765172, 9.533181356207432
 POSITION: 1.0677370365084309, 12.639545949569648

Time elapsed: 3 seconds
 VELOCITY: 0.7917542021743128, 21.62472972121006
 POSITION: 1.8594912386827436, 34.26427567077971

Time elapsed: 4 seconds
 VELOCITY: 0.049319301390379544, 46.300984607830394
 POSITION: 1.908810540073123, 80.56526027861011

Time elapsed: 5 seconds
 VELOCITY: 0.6188721160895133, 99.84341888139595
 POSITION: 2.527682656162636, 180.40867916000605

Time elapsed: 6 seconds
 VELOCITY: 0.1239154929824412, 233.06669321382589
 POSITION: 2.6515981491450775, 413.47537237383193

Time elapsed: 7 seconds
 VELOCITY: 0.42472499803844743, 1283.37403066068
 POSITION: 3.076323147183525, 1696.849403034512

Time elapsed: 8 seconds
 VELOCITY: 0.06905201863976468, 949.4446410279014
 POSITION: 3.1453751658232894, 2646.2940440624134

Time elapsed: 1 seconds
 VELOCITY: 1.0647114118262708, 3.1063645933622164
 POSITION: 1.0647114118262708, 3.1063645933622164

Time elapsed: 2 seconds
 VELOCITY: 0.023045691262270962, 9.533181356207432
 POSITION: 1.0877571030885418, 12.639545949569648

Time elapsed: 3 seconds
 VELOCITY: 0.8065995911724493, 21.62472972121006
 POSITION: 1.894356694260991, 34.26427567077971

Time elapsed: 4 seconds
 VELOCITY: 0.05024403815278988, 46.300984607830394
 POSITION: 1.944600732413781, 80.56526027861011

Time elapsed: 5 seconds
 VELOCITY: 0.6304759664788928, 99.84341888139595
 POSITION: 2.5750766988926737, 180.40867916000605

Time elapsed: 6 seconds
 VELOCITY: 0.12623890807159632, 233.06669321382589
 POSITION: 2.70131560696427, 413.47537237383193

Time elapsed: 7 seconds
 VELOCITY: 0.4326885909572593, 1283.37403066068
 POSITION: 3.1340041979215294, 1696.849403034512

Time elapsed: 8 seconds
 VELOCITY: 0.07034671333881459, 949.4446410279014
 POSITION: 3.204350911260344, 2646.2940440624134

Accelerating curve with 1.63m track.

Accelerating curve with 1.6m track.

Time elapsed: 1 seconds
 VELOCITY: 1.026227872233193, 3.0931667602642308
 POSITION: 1.026227872233193, 3.0931667602642308

Time elapsed: 2 seconds
 VELOCITY: 0.022212714505481568, 9.478460132453423
 POSITION: 1.0484405867386746, 12.571626892717653

Time elapsed: 3 seconds
 VELOCITY: 0.7774453935115206, 21.46804677028151
 POSITION: 1.8258859802501952, 34.039673662999164

Time elapsed: 4 seconds
 VELOCITY: 0.04842798885541055, 45.88390731876102
 POSITION: 1.8743139691056059, 79.92358098176018

Time elapsed: 5 seconds
 VELOCITY: 0.6076876819902636, 98.7067797902042
 POSITION: 2.4820016510958696, 178.63036077196438

Time elapsed: 6 seconds
 VELOCITY: 0.12167605665146974, 229.33696437233536
 POSITION: 2.603677707747339, 407.9673251442997

Time elapsed: 7 seconds
 VELOCITY: 0.4170492464240866, 1195.387708383574
 POSITION: 3.0207269541714257, 1603.3550335278737

Time elapsed: 8 seconds
 VELOCITY: 0.0678043145296619, 854.7218872853402
 POSITION: 3.0885312687010877, 2458.076920813214

Accelerating curve with 1.66m track.

From these results I conclude that wider track has a higher tendency to understeer, while shorter track has a higher tendency to oversteer. Thus shorter and longer track each have benefits and liabilities. Wider track causes the car to go towards the outside of the curve but is more stable. Shorter track causes the car to go towards the inside of the curve, which would give a more optimal line of attack, but it also has a higher likelihood of causing a spinout. Like track different wheelbases have different strengths and weaknesses. Shorter wheelbase results in improved acceleration but also causes oversteer when cornering. Another conclusion I have reached through this experiment is the value of simulations. Again these results can only be compared with other results from my simulation. They may not match real world examples, because I omitted some parts of the car from my simulation in order to keep the level of complexity down. Conducting this experiment in the real world would be nearly impossible, but the computer provides an ideal environment that can be used to find the answer to nearly any kind of physical phenomenon.

Bibliography

- [1] Beckman, Brian. "Physics of Racing Series." Copyright 1991.
<<http://phors.locost7.info/contents.htm>> (17 October 2006).
- [2] Parker, Barry. Isaac Newton School of Driving. The Johns Hopkins University Press, 2003.
- [3] Parker, Jim. Start Your Engines – Developing Driving and Racing Games. Paraglyph Press, 2005.
- [4] Halliday, David, Resnick, Robert, and Walker Jearl. Fundamentals of Physics. John Wiler & Sons, Inc., 2001.
- [5] Liberty, Jesse. Sams Teach Yourself C++ in 24 Hours – Third Edition. Sams Publishing, 2002.