**MPC = Model Predictive Control**

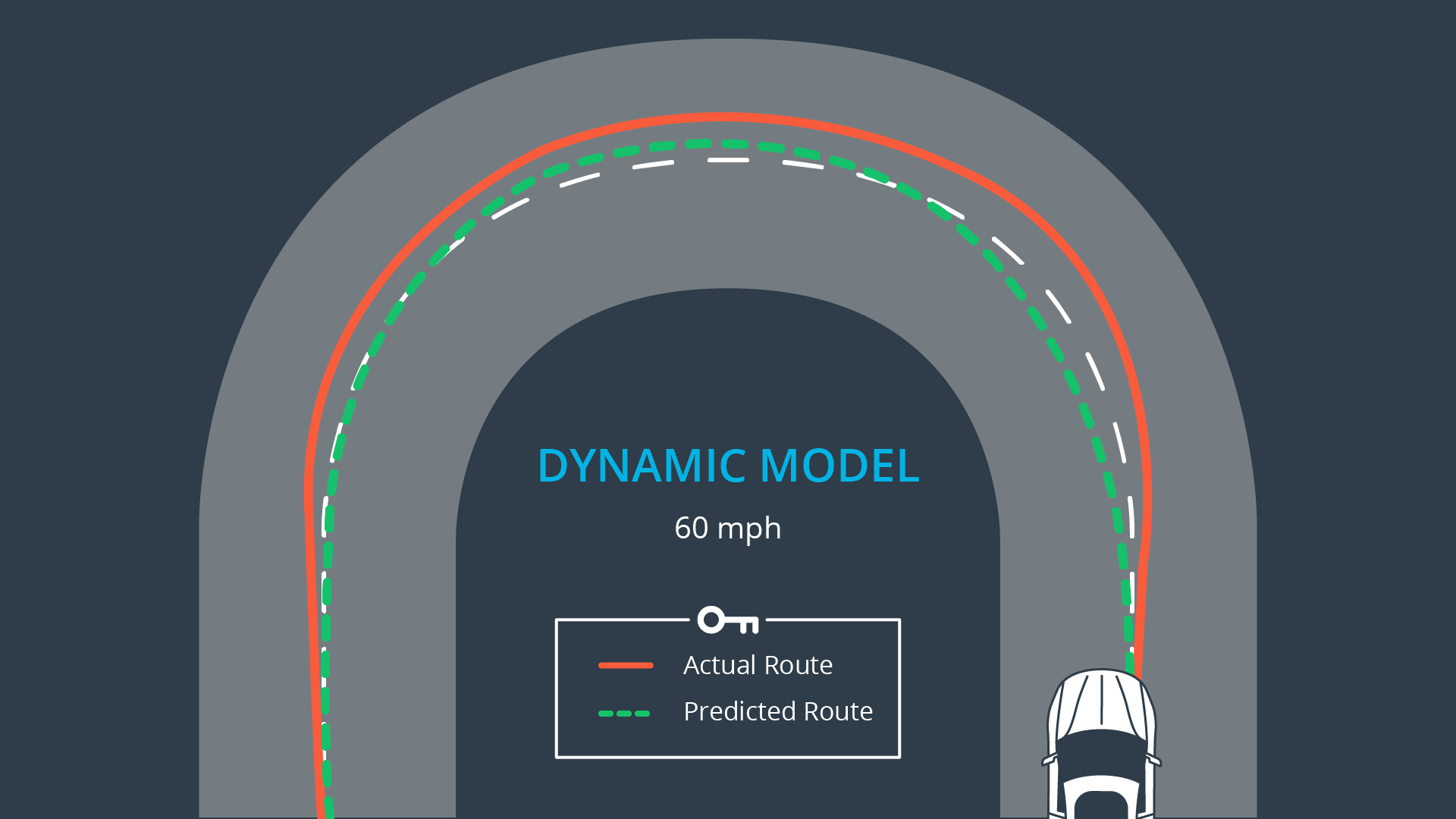
**LESSON 18.2 Vehicle Models – Dynamic Models – MOST ACCURATE**

Dynamic models aim to embody the actual vehicle dynamics as closely as possible.

They might encompass tire forces, longitudinal and lateral forces, inertia, gravity, air resistance, drag, mass, and the geometry of the vehicle.

Not all dynamic models are created equal! Some may consider more of these factors than others.

Advanced dynamic models even take internal vehicle forces into account - for example, how responsive the chassis suspension is.

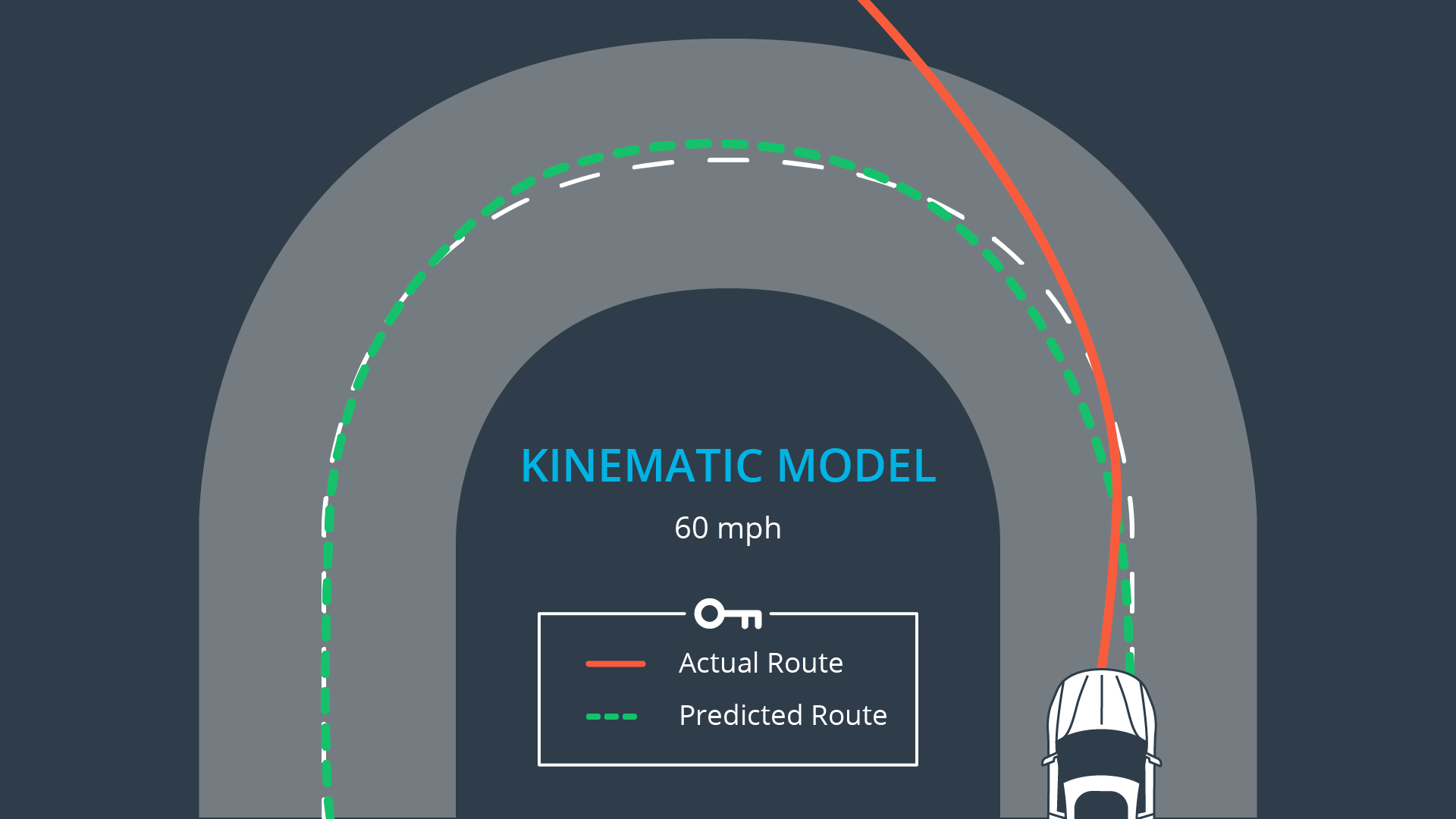


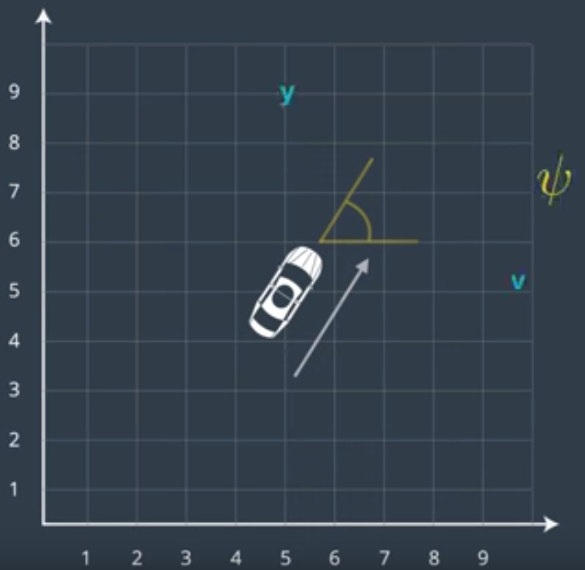
**LESSON 18.2 Vehicle Models – Kinematic Models – IGNORES FORCES**

Kinematic models are simplifications of dynamic models that ignore tire forces, gravity, and mass.

This simplification reduces the accuracy of the models, but it also makes them more tractable.

At low and moderate speeds, kinematic models often approximate the actual vehicle dynamics.

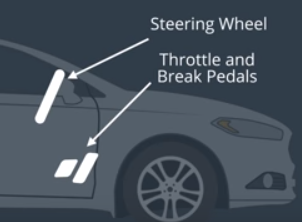
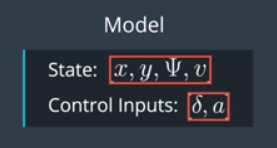
**LESSON 18.3 State of a car**

Four state variables define the state of a moving car:



**LESSON 18.4 Kinematic Model**

Two actuators: Steering Wheel and Throttle/Break Pedals



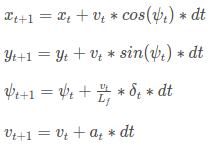
**LESSON 18.4: Building a KINEMATIC MODEL (without forces)**

Kinematic model defines the state, actuators and how the state changes over time based on the previous state and current actuator inputs:

State Vector: [x, y, psi, v] : x-,y-locations, angle, velocity

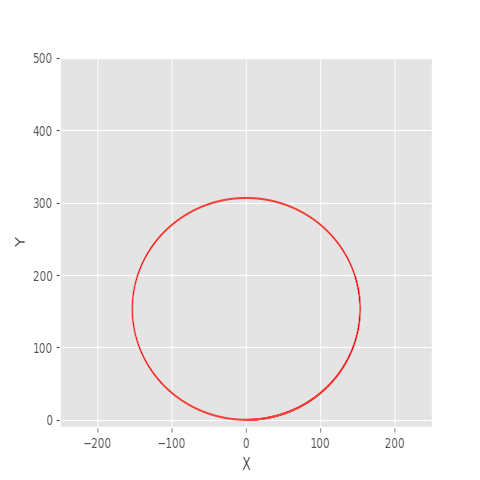
Actuators (Control Inputs): [*δ* , a] : steering angle, acceleration (throttle and break)

Kinematic Model:



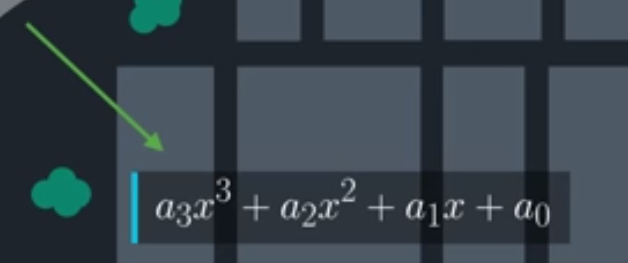
where Lf measures the distance between the front of the vehicle and its center of gravity. The larger the vehicle, the slower the turn rate. If you've driven a vehicle you're well aware at higher speeds you turn quicker than at lower speeds. This is why *v* is the included in the update On the topic of running a vehicle around in a circle, this is actually a good way to test the validity of a model! If the radius of the circle generated from driving the test vehicle around in a circle with a constant velocity and steering angle is similar to that of your model in the simulation, then you're on the right track.

This is how the value of *L*f was chosen for the project!



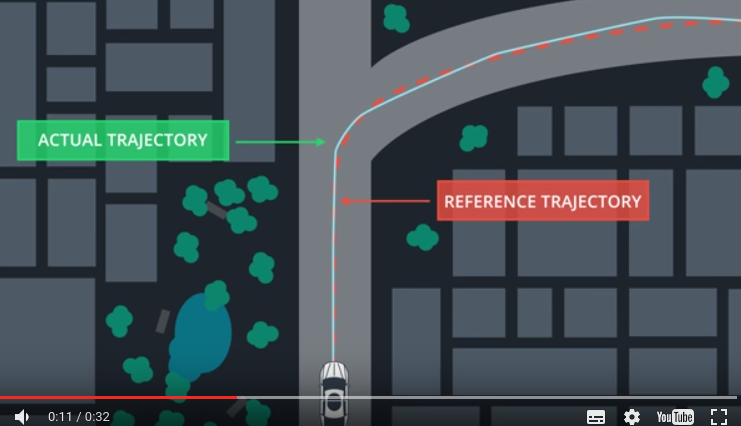
In the above image the vehicle started at the origin, oriented at 0 degrees and was then simulated driving with a *δ* value of 1 degree and *Lf* value of 2.67.

**LESSON 18.6: Following Trajectories**

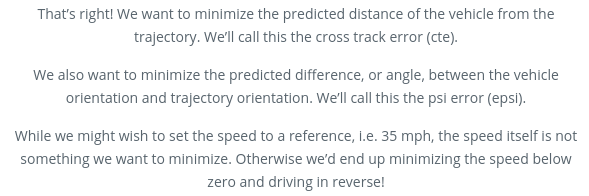


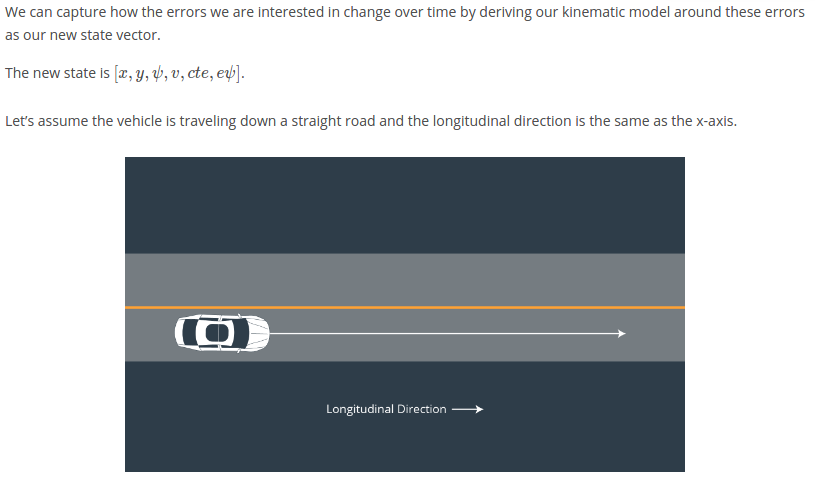
Path of trajectory is commonly given as 3rd degree polynomials.

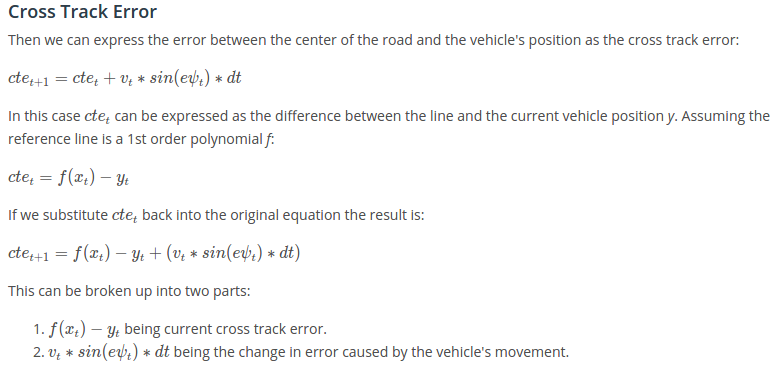
**Lesson 18.8: Errors**

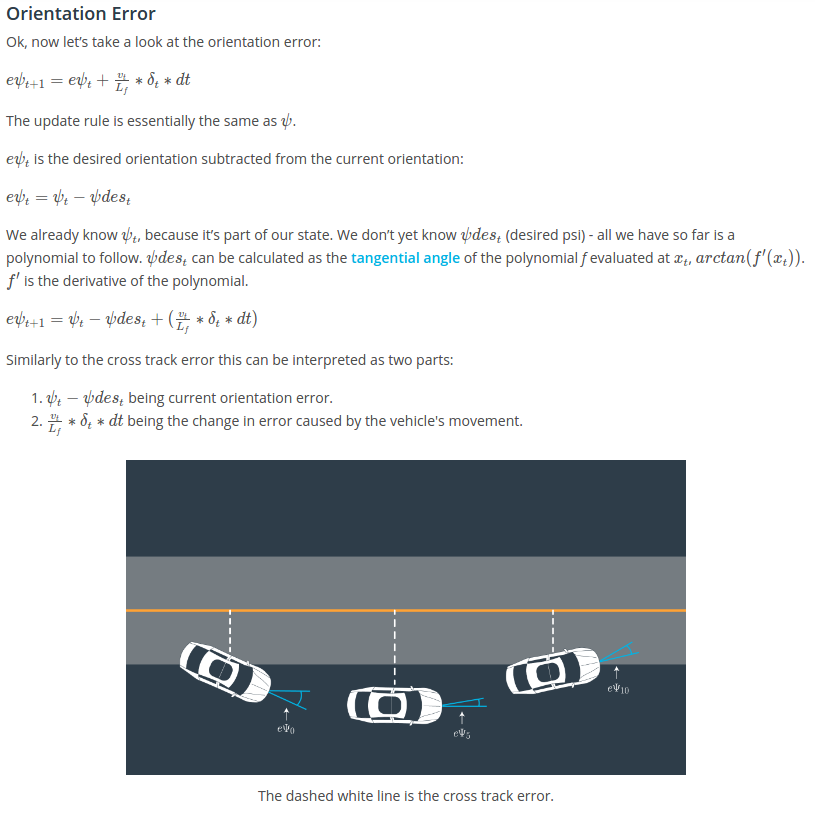


The KINEMATIC model is used to predict the future state, and then adjust the actuators (Input Controls) to minimize the error between actual trajectory and reference trajectory.

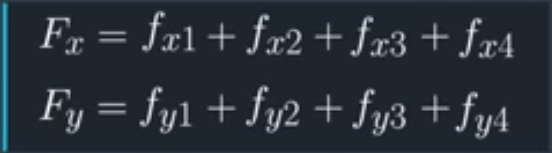
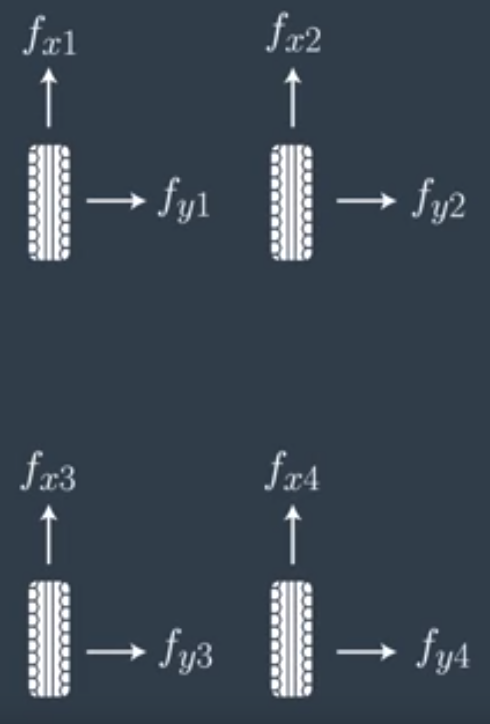
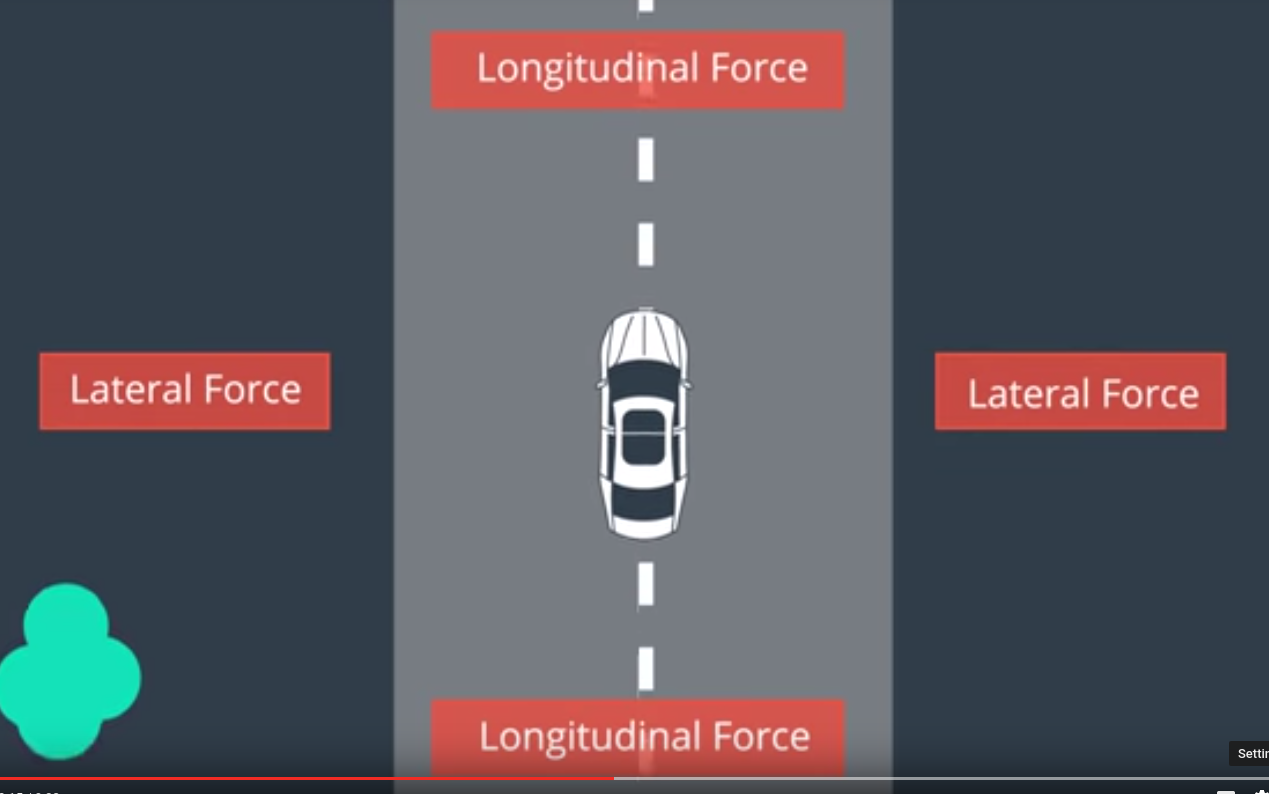
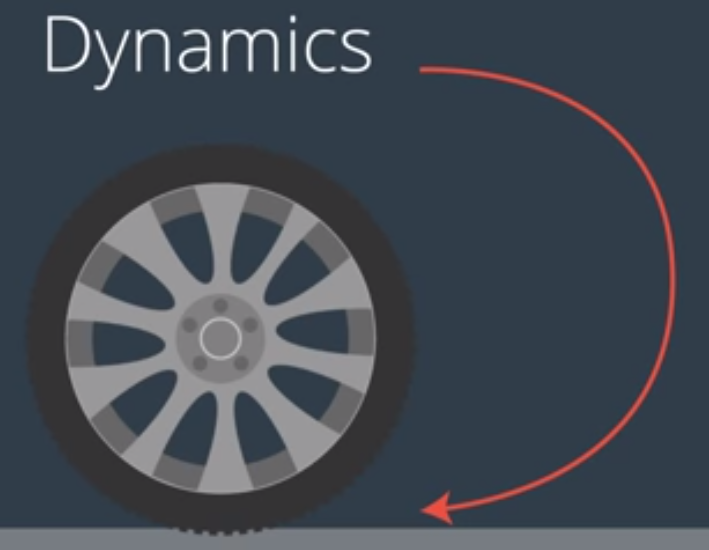




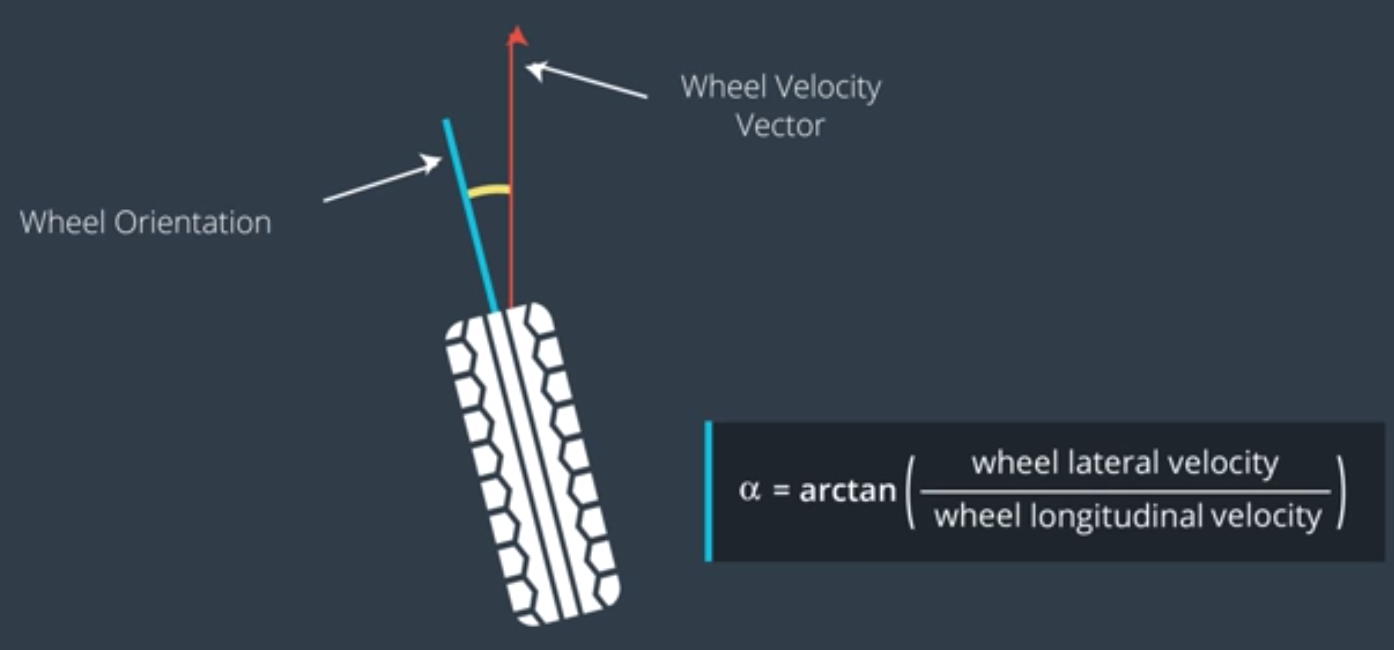




**Lesson 18.9: Dynamics Models – Forces acting between car and road (Tires!)**



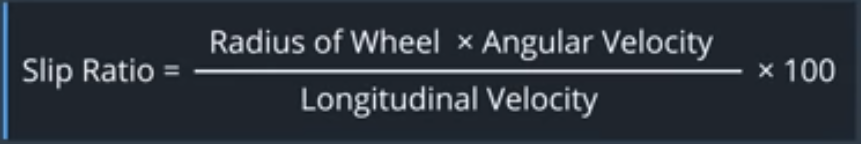
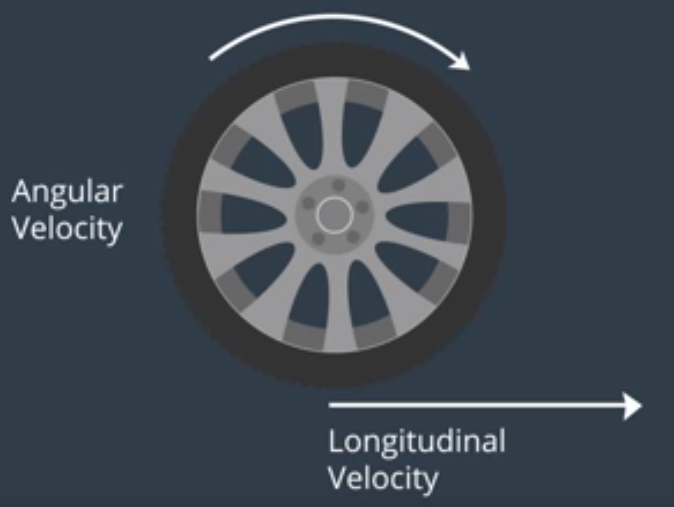
**Lesson 18.10: Dynamic Models – Tire Slip Angle**



The slip angle is needed to generate lateral (sideways) force → Makes a car turn!

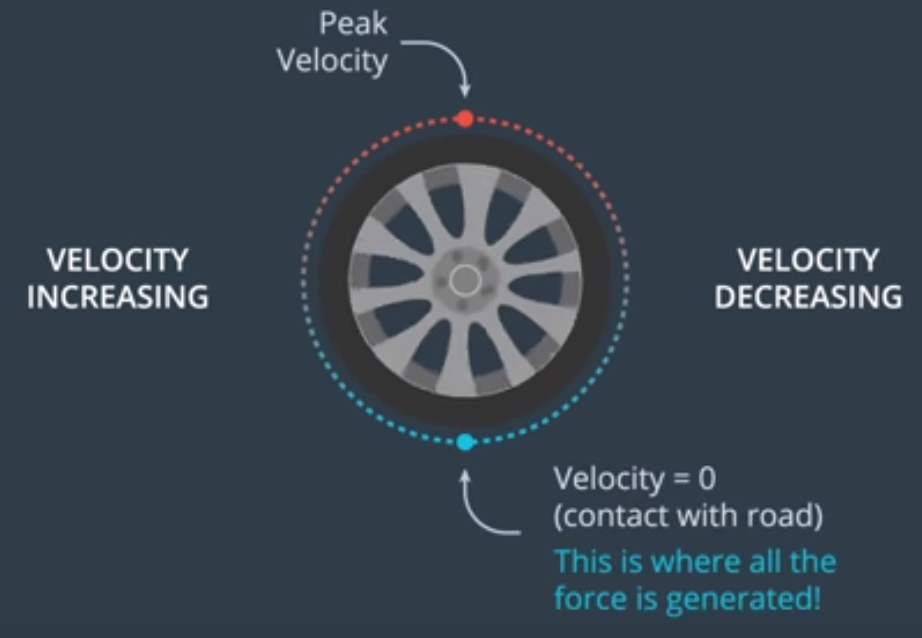
Race tires generates more force from the same slip angle than conventional tires.

**Lesson 18.11: Dynamic Models – Tire Slip Ratio**



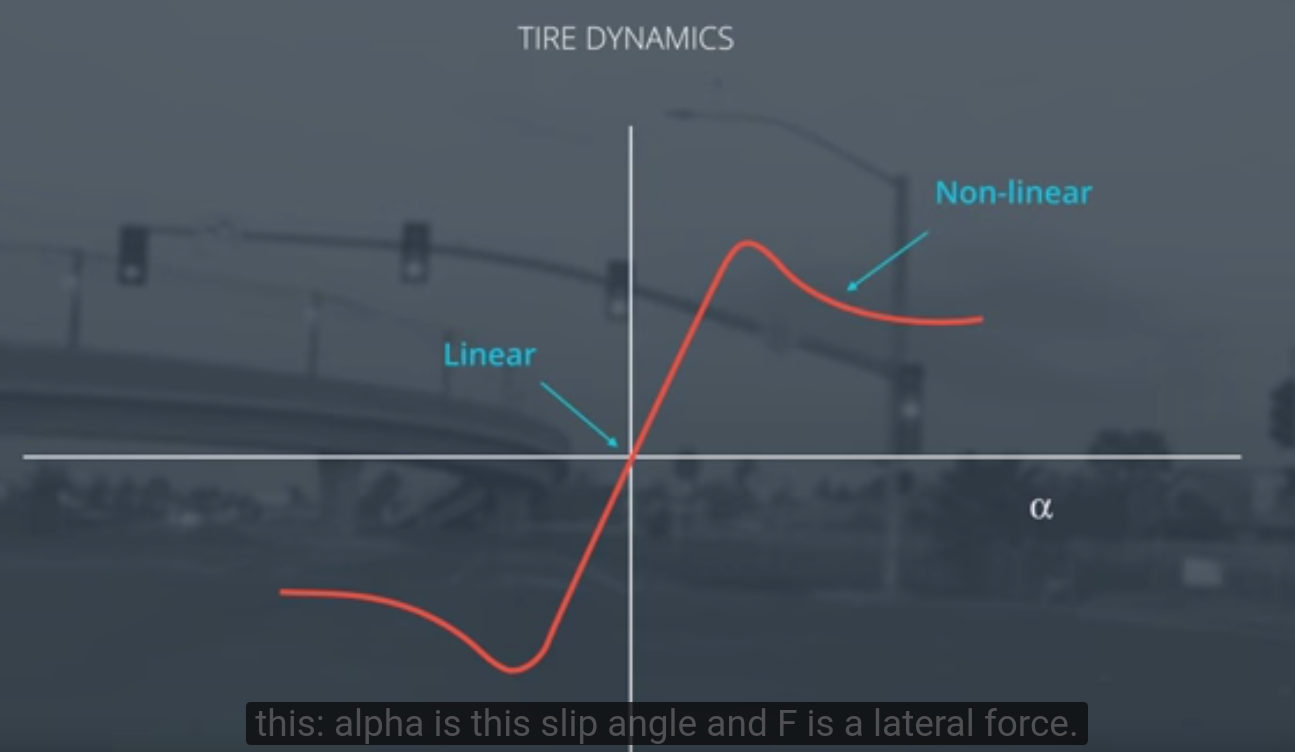
The slip ratio is a mis-match between tire angular velocity and it’s longitudinal velocity.

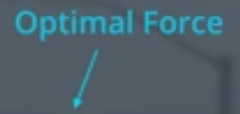
The slip ratio is needed to generate the longitudinal force → Makes a car accelerate or decelerate!



**Lesson 18.12: Dynamic Models – Tire Models**

Example: Lateral force as function of slip angle:





The most popular model is the Pacejka tire model, also known as the Magic Tire Formula.

## **Additional Reading**

* [**Pacejka Tire Model**](http://www.theoryinpracticeengineering.com/resources/tires/pacejka87.pdf)
* [**This paper**](http://www.me.berkeley.edu/~frborrel/pdfpub/IV_KinematicMPC_jason.pdf) presents a comparison between a kinematic and dynamic model. (MPC!)

**Lesson 18.13: Actuator Constraints**

The actuators always have constraints. For example, a vehicle can’t have a steering angle of 90 degrees.

A model that is using steering constraints is called: non holonomic

This is done by setting lower & upper bounds for the actuators, for example:

* delta = [-30, 30] degrees
* a = [-1,1] m/sec

**Lesson 18.14: Cost function**

Kinematic and Dynamic vehicle models, together with the actuator constraints are the foundations of Model Predictive Control (MPC).

We have build a model that describes the future trajectory of the vehicle.

To build an optimal controller, we need to define a **cost function** that captures the errors we want to minimize.

The cost function will require to use the model to predict where the vehicle will go in the future.

**The cost** is the difference between where you want the vehicle to go and where you predict it will go based on that model.

**Lesson 19.1: Model Predictive Control is an optimization problem**

MPC involves simulating variations of actuator inputs in time, and minimize a cost function.

The cost function is build with several considerations:

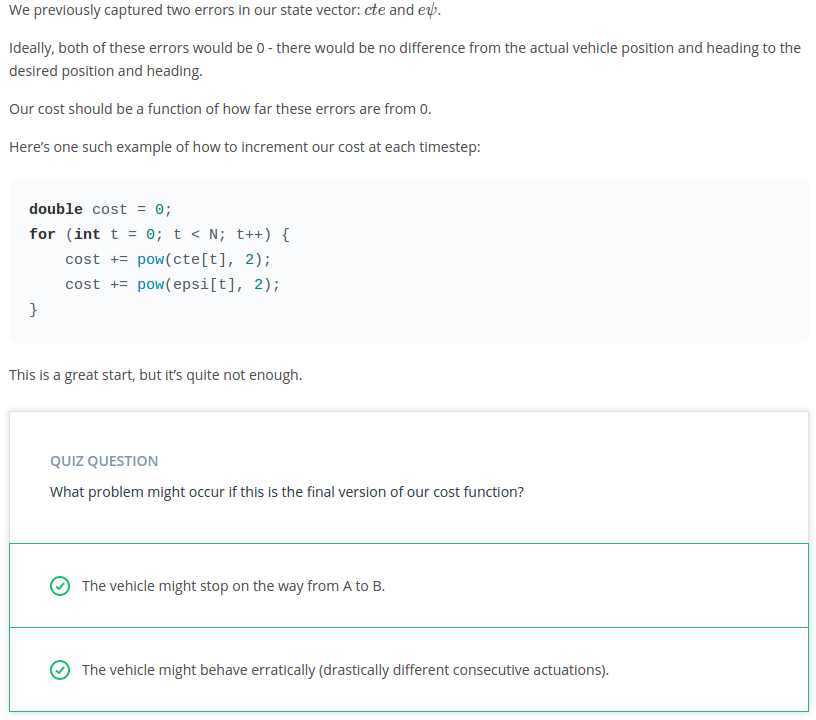
* Follow trajectory
* Drive smoothly
* Maintain a certain target speed
* etc.

The simulation works like this:

* Calculate actuator inputs in time that predict an optimal future trajectory
* Move forward on time-step
* Throw away calculated actuator inputs and take current state. We do this because the predictive model is inaccurate.
* Repeat

Since we are continuously calculating future trajectory, it is sometimes called ‘Receding Horizon Control’.

**Lesson 19.2: Cost function: terms to follow trajectory, using state variables**



**Lesson 19.3: Cost function: avoid stopping, using state variables**

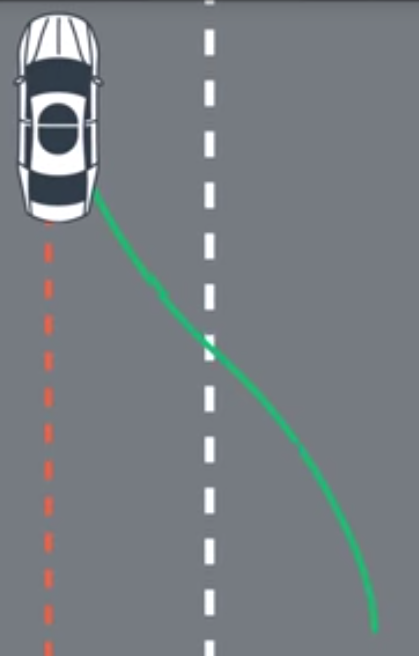
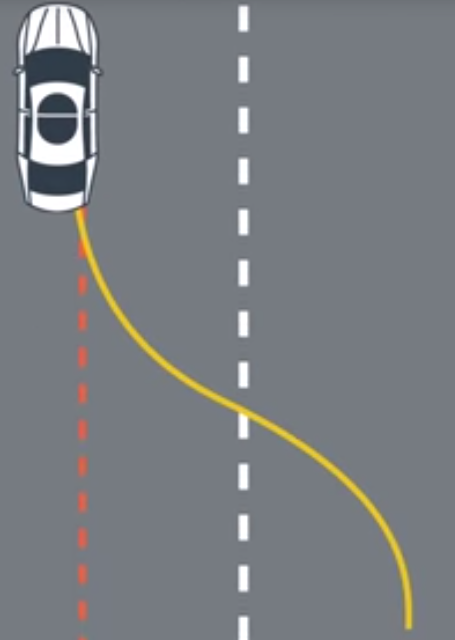
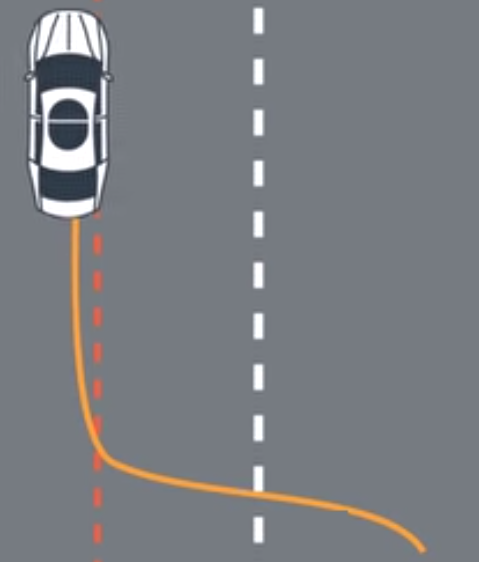
Option 1: Insert a reference velocity into the cost function that penalizes a different velocity:

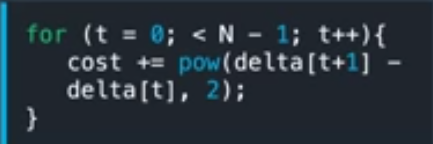


Option 2: Measure distance to destination and add this to the cost function.

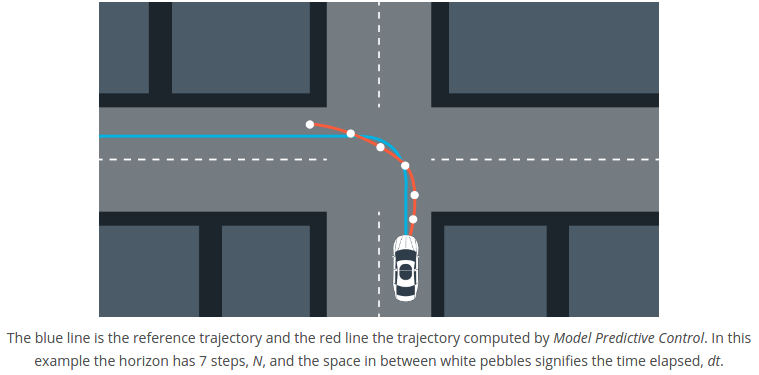
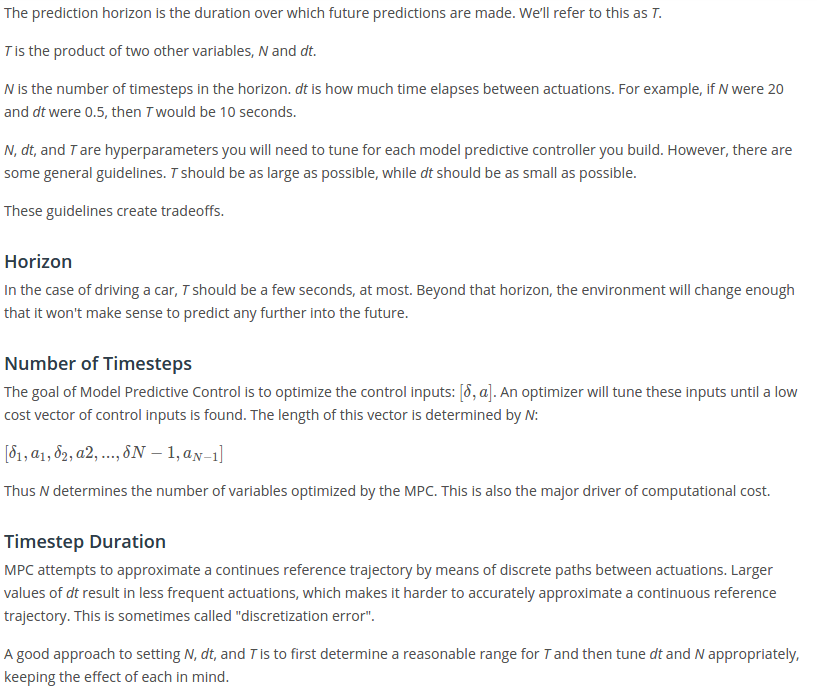
**Lesson 19.4: Cost function: additional considerations, using non-state variables**

It is possible to include the control input into the cost function, for example to enforce smooth lane changes one can penalize the steering angle or the change-rate of steering angle:

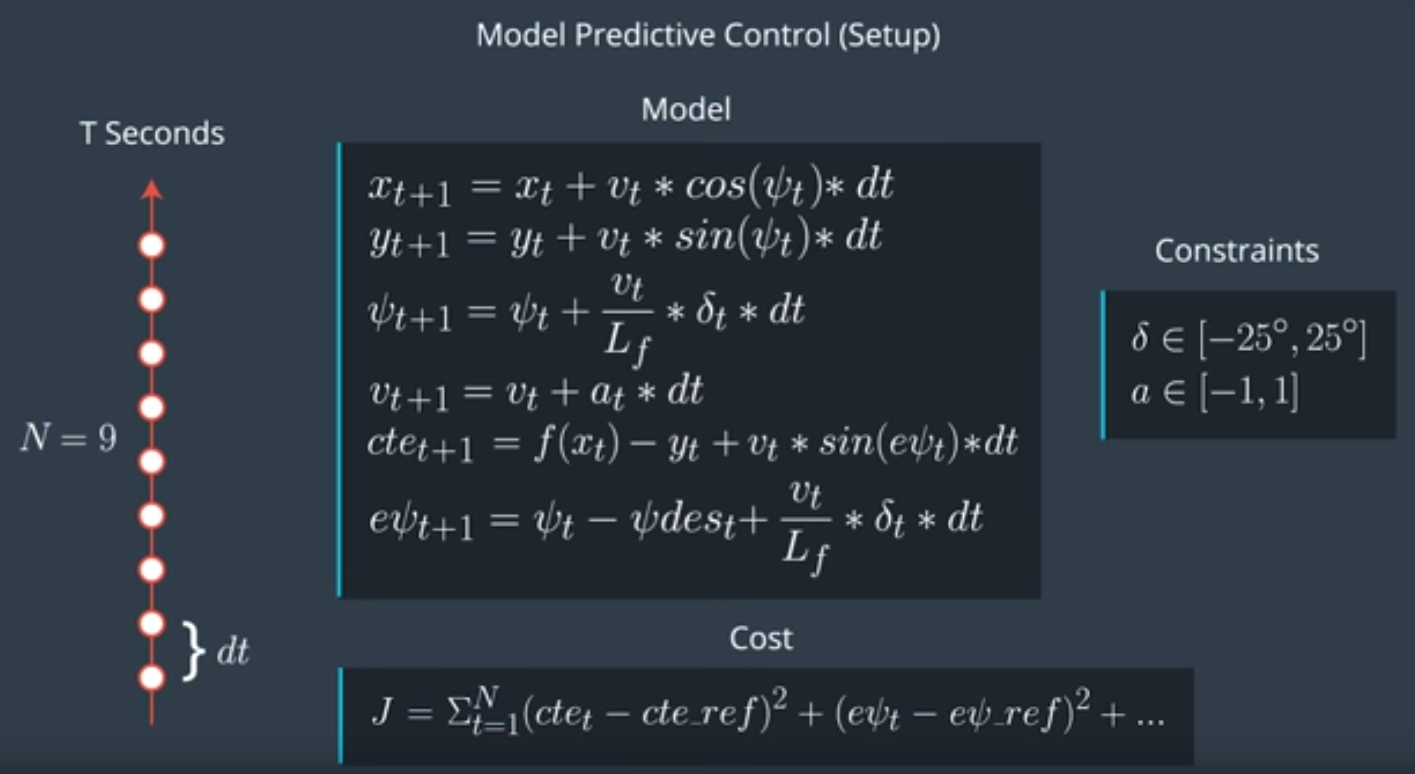


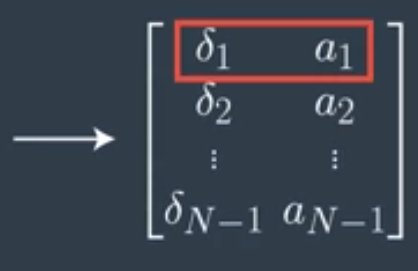
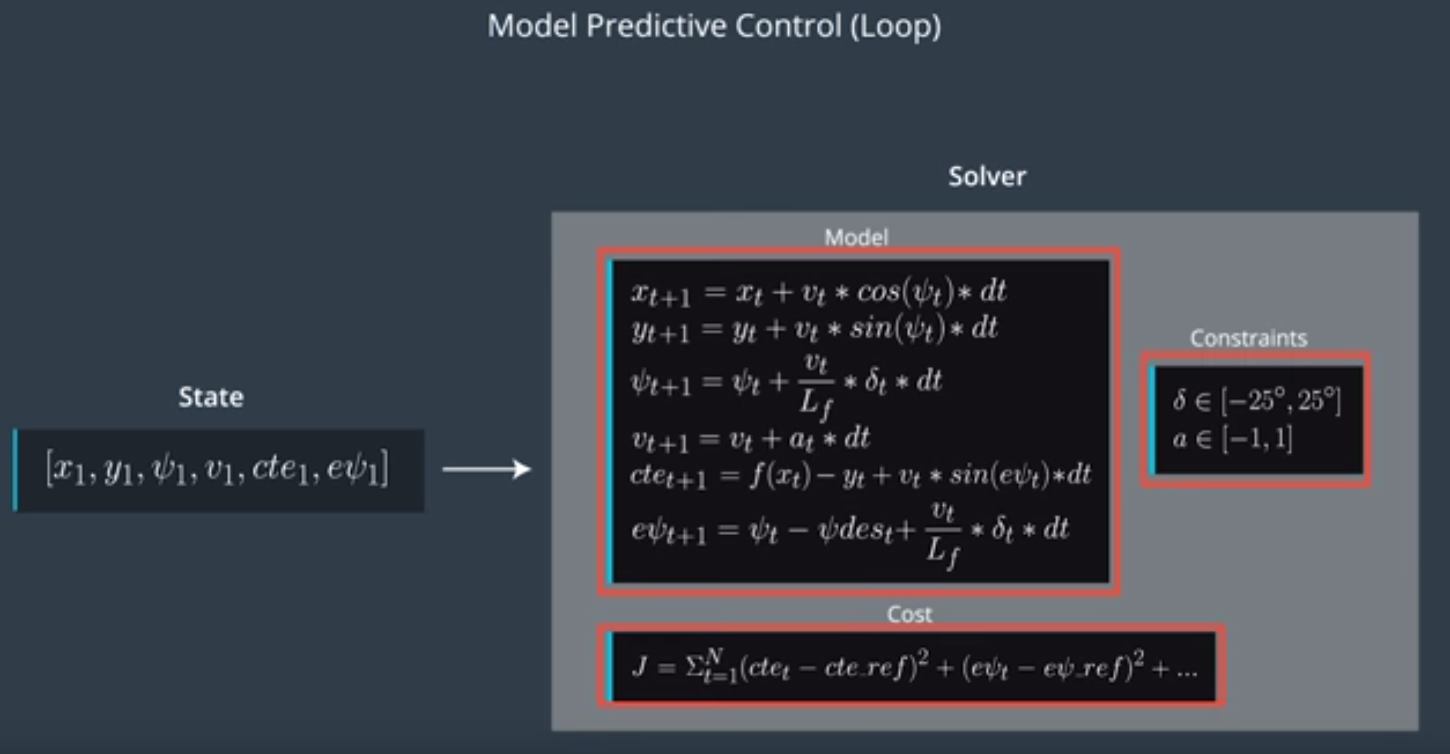


**Lesson 19.5: Length and Duration**



**Lesson 19.6: Implementation details – Use of an optimization solver**

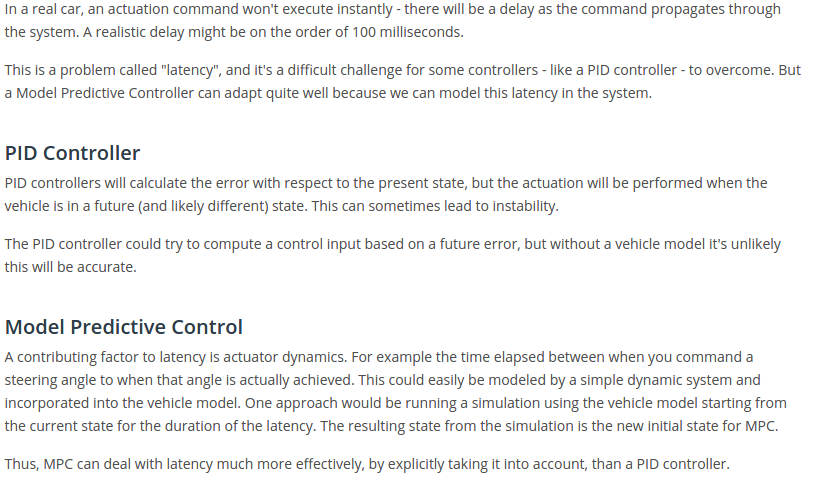




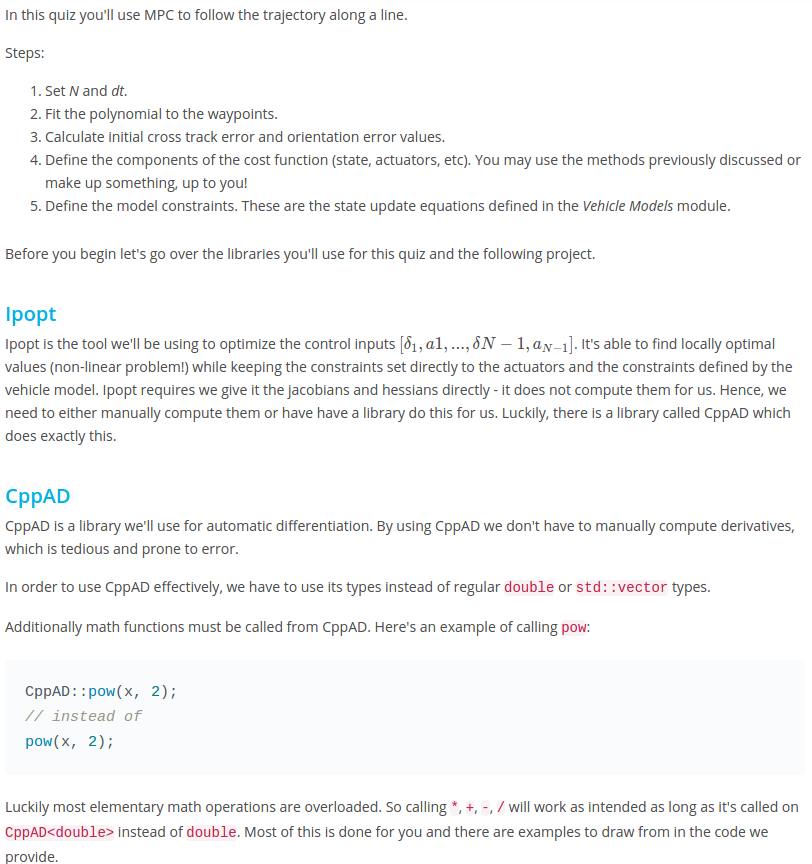


Apply optimized control inputs delta\_1 and a\_1. Then reset & repeat.

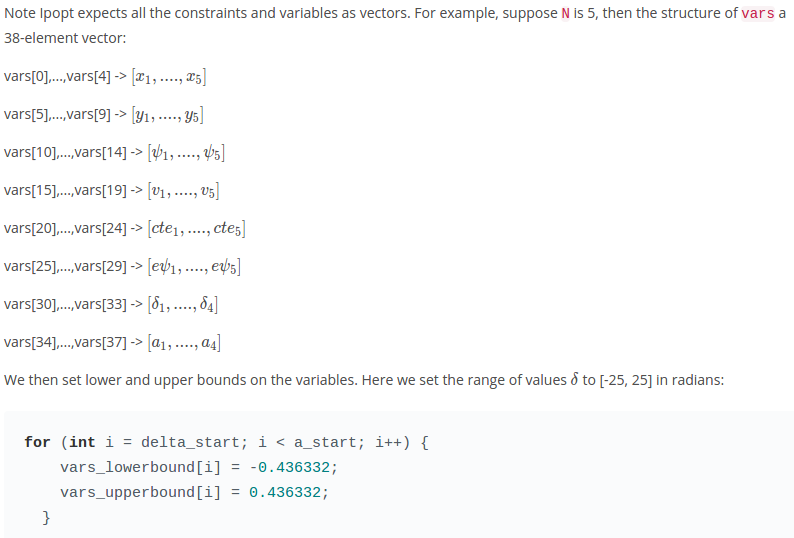
**Lesson 19.7: Latency → realistic delay is ~ 100 milliseconds!**

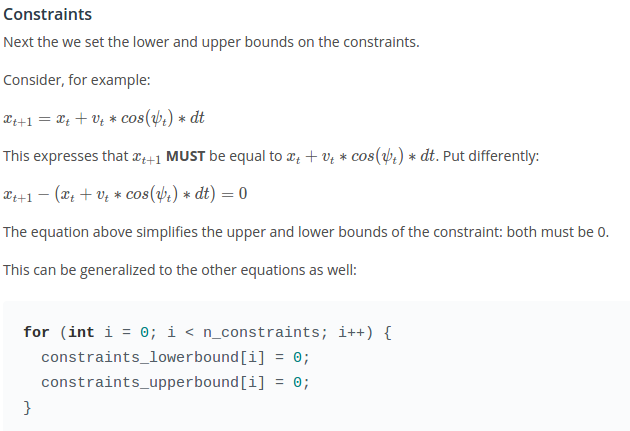


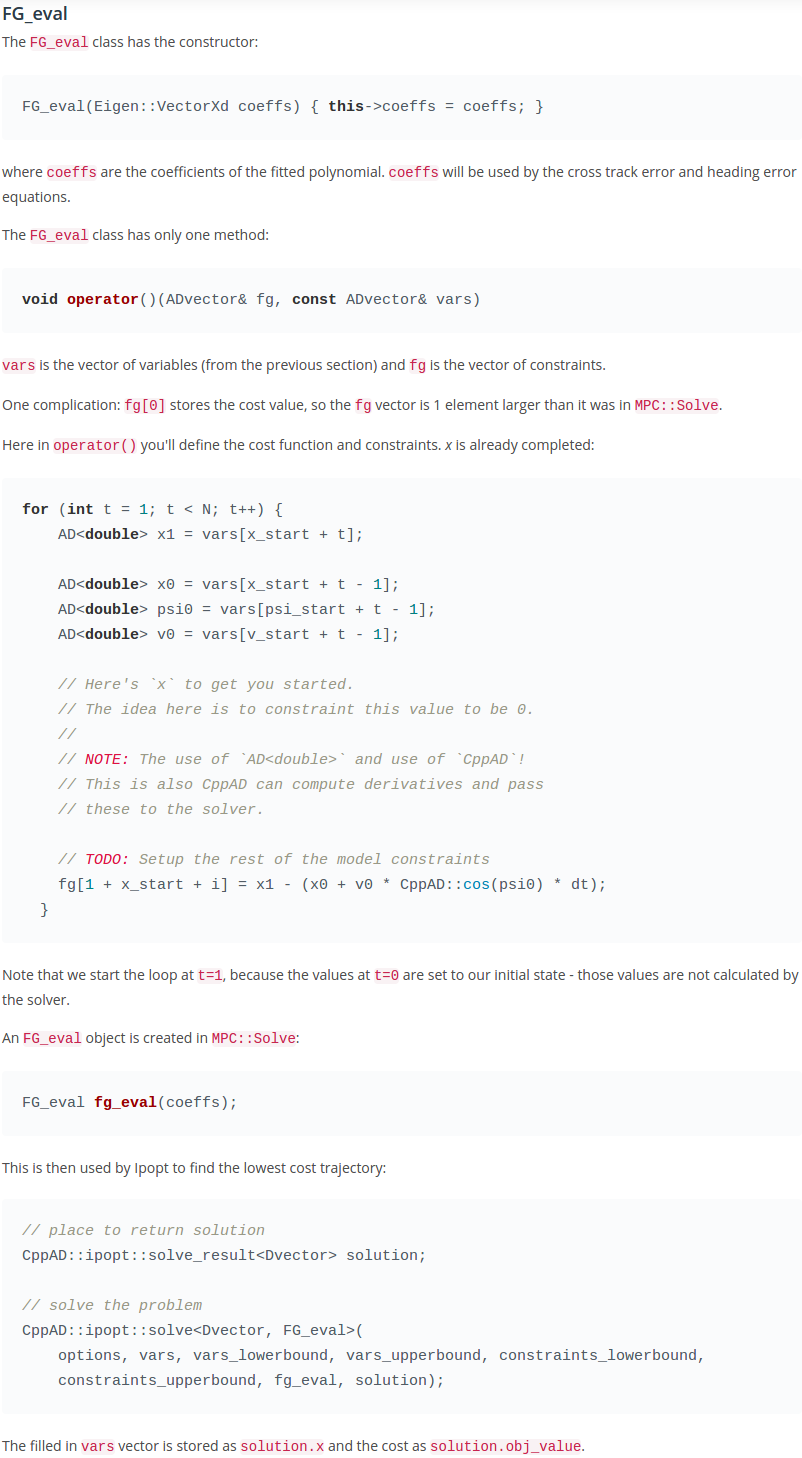
**Lesson 19.8: Follow the trajectory along a line (C++ quiz to implement MPC)**











For Project:

Conversion from global (\_g) to car (\_c) coordinates

x\_g = x\_car\_g + x\_c\*cos(psi\_g) - y\_c\*sin(psi\_g)

y\_g = y\_car\_g + x\_c\*sin(psi\_g) + y\_c\*cos(psi\_g)

Conversion from global (\_g) to car (\_c) coordinates

→ x\_g - x\_car\_g = x\_c\*cos(psi\_g) - y\_c\*sin(psi\_g)

y\_g - y\_car\_g = x\_c\*sin(psi\_g) + y\_c\*cos(psi\_g)

→ x\_c = ( x\_g – x\_car\_g)\*cos(psi\_g) + (y\_g – y\_car\_g)\*sin(psi\_g)

y\_c = -( x\_g – x\_car\_g)\*sin(psi\_g) + (y\_g – y\_car\_g)\*cos(psi\_g)

See also:<https://gamedev.stackexchange.com/questions/79765/how-do-i-convert-from-the-global-coordinate-space-to-a-local-space>

