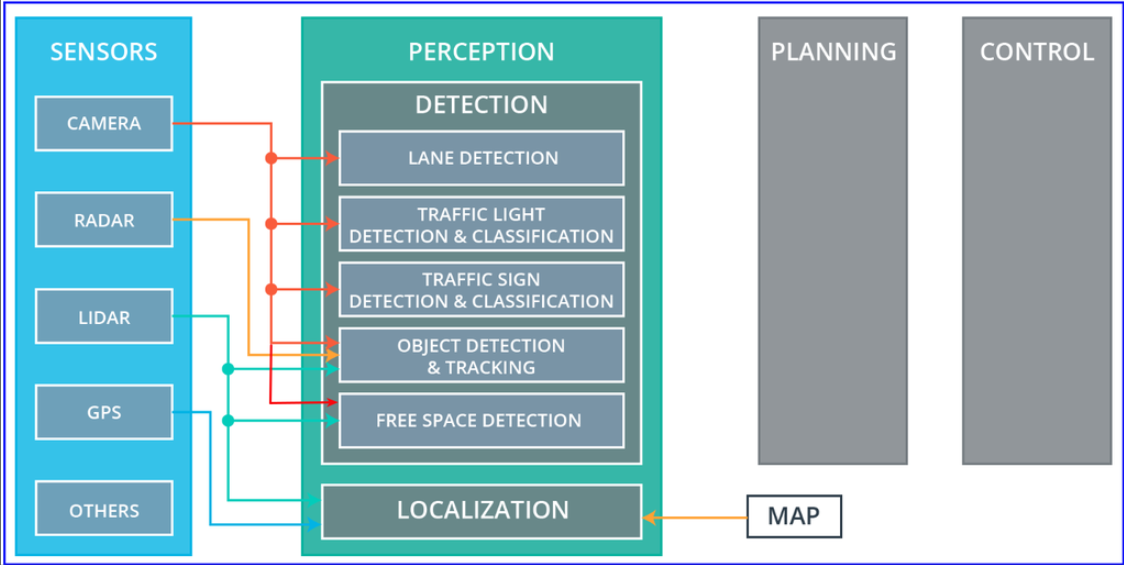
AD Architecture



Planning:

**Route planning**

The route planning component is responsible for high-level decisions about the path of the vehicle between two points on a map; for example which roads, highways, or freeways to take. This component is similar to the route planning feature found on many smartphones or modern car navigation systems.

**Prediction**

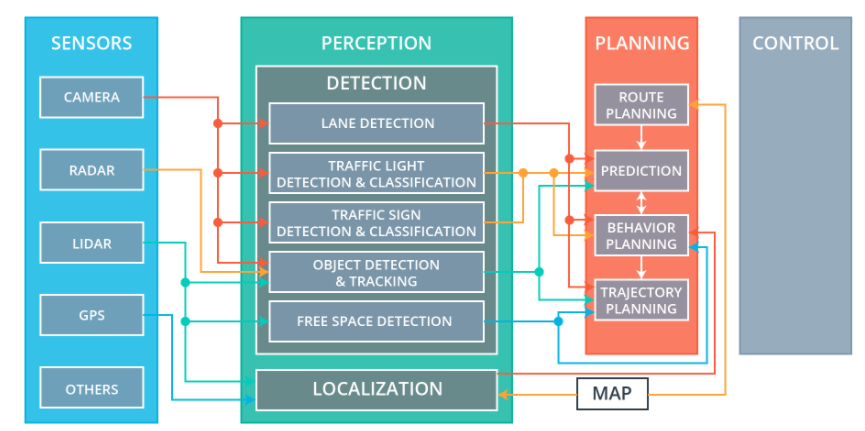
The prediction component estimates what actions other objects might take in the future. For example, if another vehicle were identified, the prediction component would estimate its future trajectory.

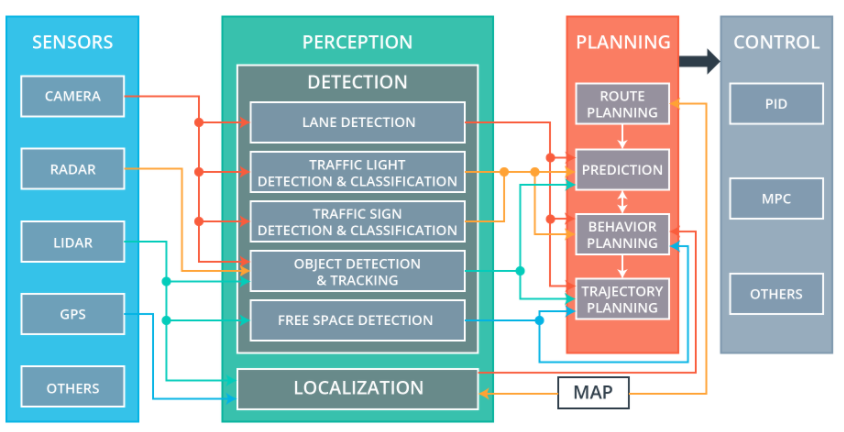
**Behavioral planning**

The behavioral planning component determines what behavior the vehicle should exhibit at any point in time. For example stopping at a traffic light or intersection, changing lanes, accelerating, or making a left turn onto a new street are all maneuvers that may be issued by this component.

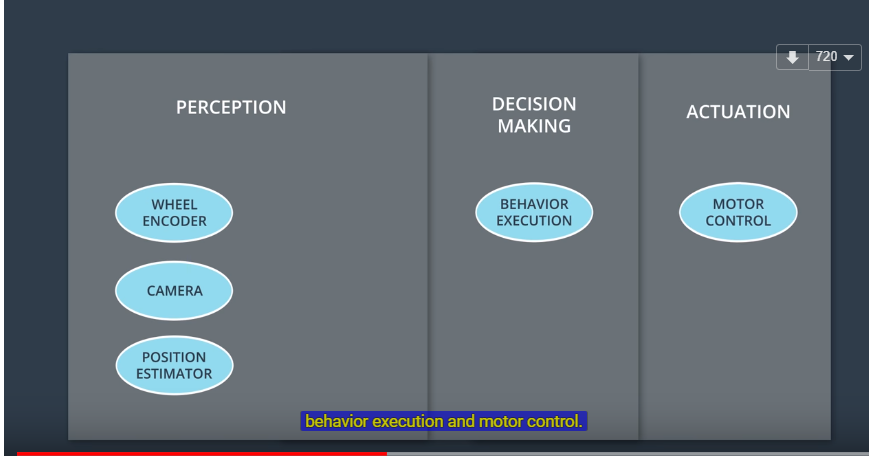
**Trajectory planning**

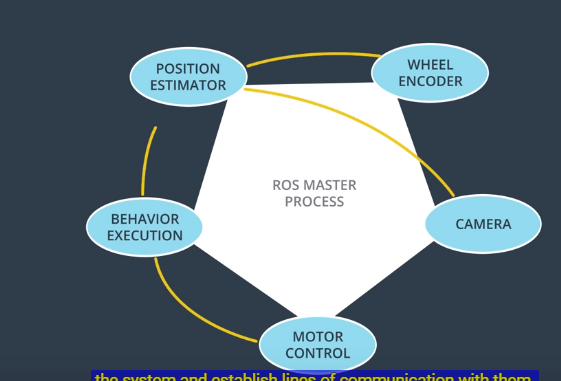
Based on the desired immediate behavior, the trajectory planning component will determine which trajectory is best for executing this behavior.

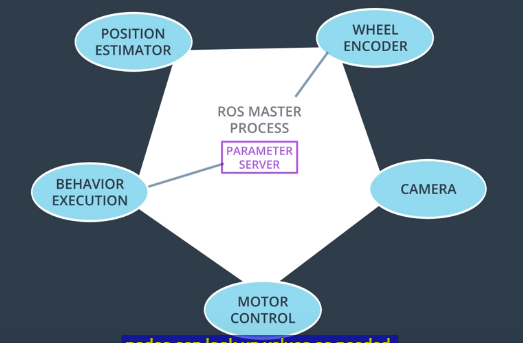


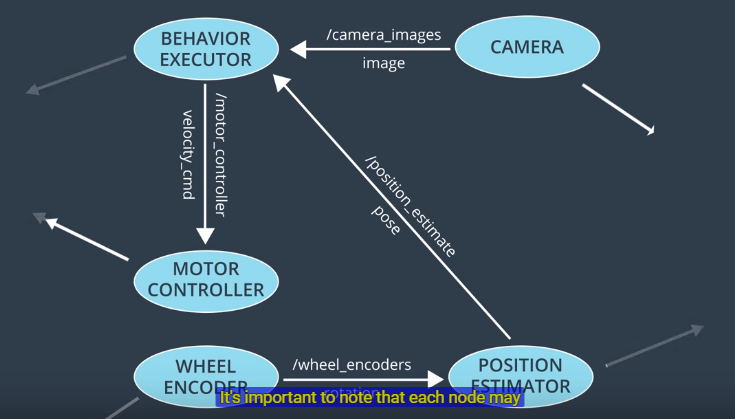


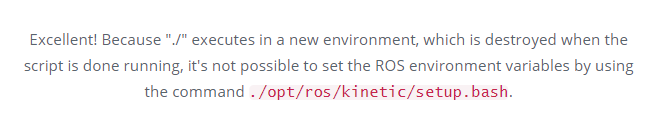
**ROS**











======================= catkin workspace ==========================

***Catkin\_make***

Must run catkin\_make directly under catkin\_ws\_oroca which is the **workspace home directory**

!!



Catkin\_ws\_oroca/ build/

src/

devel/

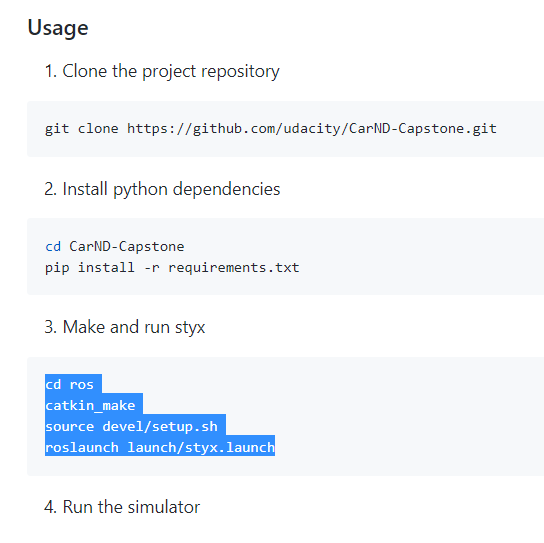
This setup.bash script must be sourced before using the catkin workspace

***Source devel/setup.bash***

***Roslaunch***

roslaunch allows you to do the following

* Launch ROS Master and multiple nodes with one simple command
* Set default parameters on the parameter server
* Automatically re-spawn processes that have died



***Rosdep***

The rosdep tool will check for a package's missing dependencies, download them, and install them.

To check for missing dependencies in the simple\_arm package:

$ rosdep check simple\_arm

===========conda===========

Bash:

# added by Anaconda2 installer

export PATH="/home/byron/anaconda2/bin:$PATH"

============== docker ================

Sudo apt install docker.io

============= git pull fork remote repo=====================

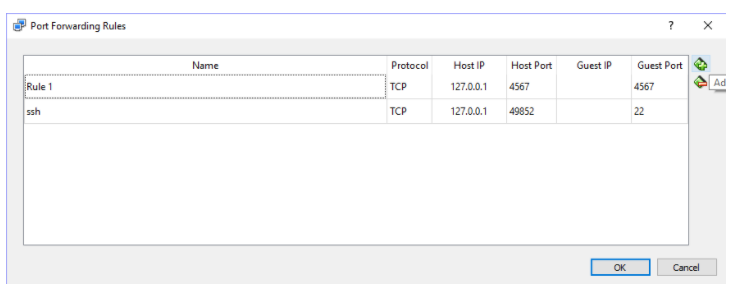


=================== simulator V1.3======================================

Virtualbox Netoword Adaptor prot forwarding:

Bridge: mutual with port forwarding, do NOT need enable

NAT: manually add port 4567:



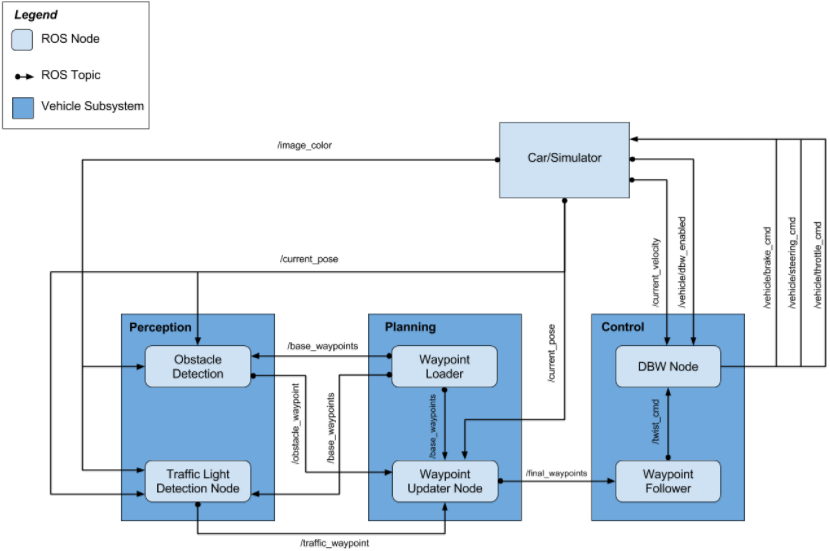
<https://classroom.udacity.com/nanodegrees/nd013/parts/40f38239-66b6-46ec-ae68-03afd8a601c8/modules/0949fca6-b379-42af-a919-ee50aa304e6a/lessons/f758c44c-5e40-4e01-93b5-1a82aa4e044f/concepts/16cf4a78-4fc7-49e1-8621-3450ca938b77>

the first track has a toggle button for camera data. Many students have experienced latency when running the simulator together with a virtual machine, and leaving the camera data off as you develop the car's controllers will help with this issue.

**Controller development: switch off Camera data !**

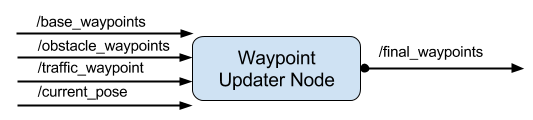
all values used within the project code are use the metric system (m or m/s), including current velocity data coming from the simulator : **m/s convert to MPH in display !**

**=========== project plan ===============**

****

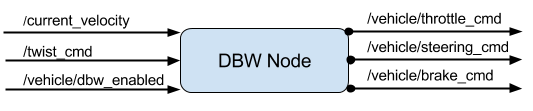
**1, Waypoint Updater Node (Partial):**

Complete a partial waypoint updater which subscribes to /base\_waypoints and /current\_pose and publishes to /final\_waypoints.

****

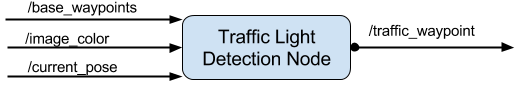
**2.a, DBW Node**

Once your waypoint updater is publishing /final\_waypoints, the **waypoint\_follower**node will start publishing messages to the/twist\_cmd topic. At this point, you have everything needed to build the **dbw\_node**. After completing this step, the car should drive in the simulator, ignoring the traffic lights..

****

**2.b Traffic Light Detection**

* Detection: Detect the traffic light and its color from the /image\_color. The topic /vehicle/traffic\_lights contains the exact location and status of all traffic lights in simulator, so you can test your output.
* Waypoint publishing: Once you have correctly identified the traffic light and determined its position, you can convert it to a waypoint index and publish it.



You will build both a traffic light detection node and a traffic light classification node. Traffic light detection should take place within **tl\_detector.py**, whereas traffic light classification should take place within **../tl\_detector/light\_classification\_model/tl\_classfier.py.**

**3, Waypoint Updater (Full):**

Use /traffic\_waypoint to change the waypoint target velocities before publishing to /final\_waypoints. Your car should now stop at red traffic lights and move when they are green.