Self Driving Cars Course

Taxonomy of Driving

The driving task is composed of 3 sub-tasks:

- Perception
- Motion Planning
- Controlling the vehicle

There is 6 levels to classify a driving system automation:

- Level 0: No automation
- Level 1: Driving assistance (longitudinal or lateral control)
- Level 2: Partial driving automation (longitudinal and lateral control)
- Level 3: Level 2 + OEDR (Object and Event Detection and Reponse)
- Level 4: Level 3 + Fallback(Handles emergency: the driver can entirely focus on other tasks)
- Level 5: Level 4 + Unlimited ODD (Operational Design Domain) can operate under any condition

Actually, the maximum available in the industry is the level 3, however Waymo has deployed vehicle for public transport with the level 4 in a defined geographic area.

Requirements for Perception

Perception is making sense of the environment and ourselves in the environment.

The main tasks for perception are detecting and assessing various types of static and dynamic objects and agents in the environment and the task of making sense of how the vehicle is moving trough the environment.

Driving Decisions and Actions

There is differents types of planing: long term, short term and immediate planning.

For this, we have 2 differents planning approaches: Reactive and predictive planning. Obviously, the most we can use predictive planning, better it is, however, reactive planning is sometimes unavoidable.

Sensors and Computing Hardware

2 types of sensors:

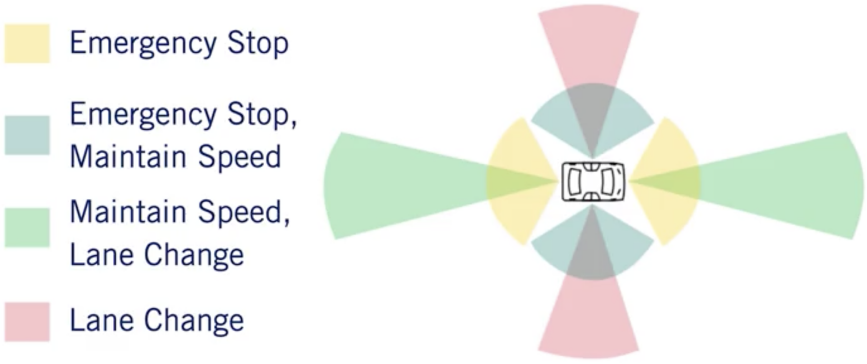
- exteroceptive: relating to, being, or activated by stimuli received by an organism from outside
- proprioceptive : of, relating to, or being stimuli arising within the organism

Among the sensors, we spoke about camera, LIDAR, RADAR, ultrasonics, GNSS, IMU and wheel odometry.

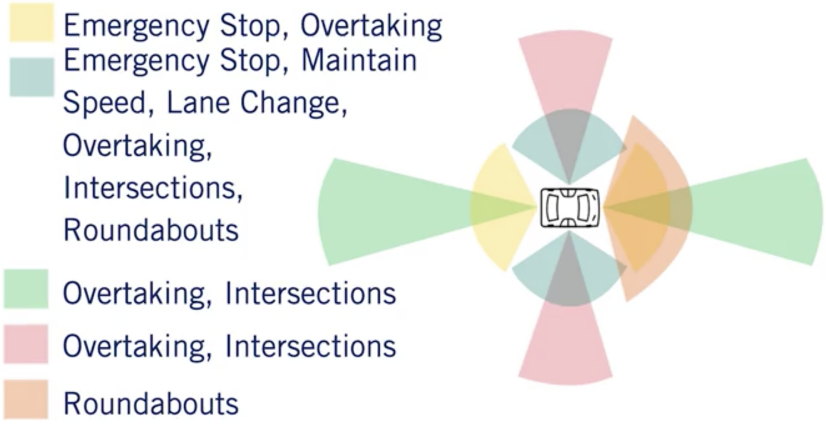
To computing hardware, we need: - a self-driving brain - Image processing, Objects detection, Mapping - Synchronization hardware

Hardware Configuration Design

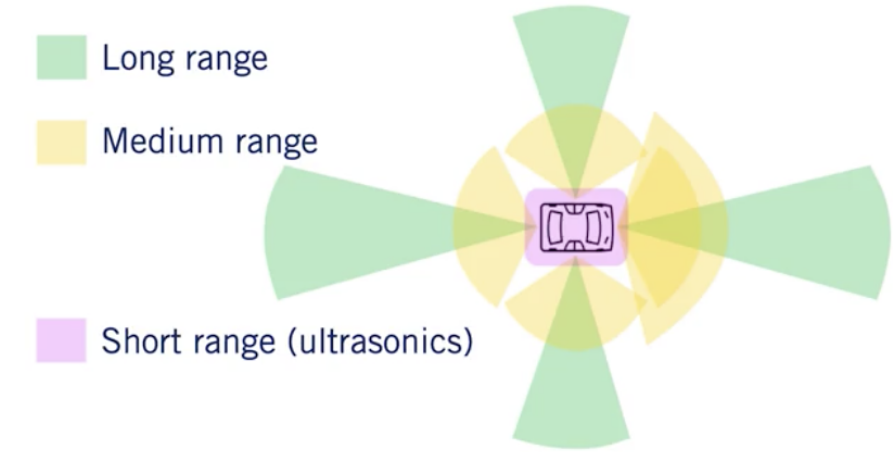
For highways, we need sensors being able to complete 4 tasks:



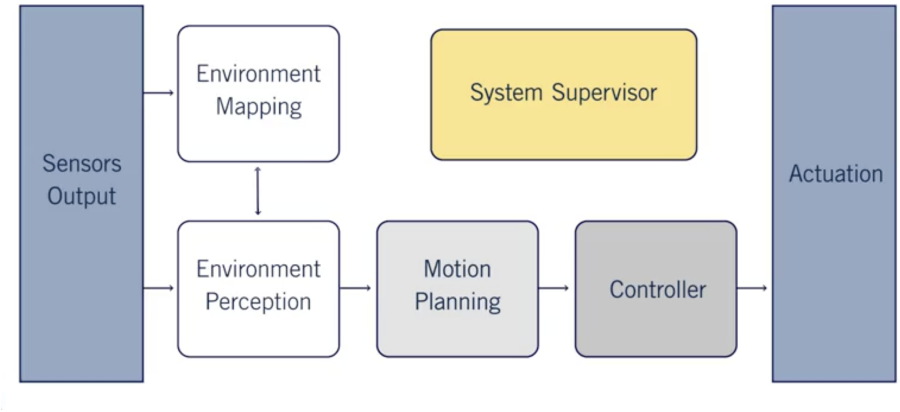
For urban analysis, we need sensors being able to complete 6 tasks:



For recap, we need 3 types of sensors:

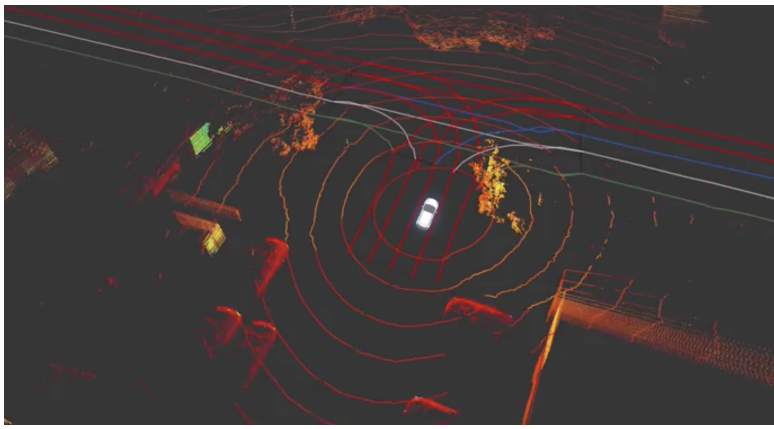
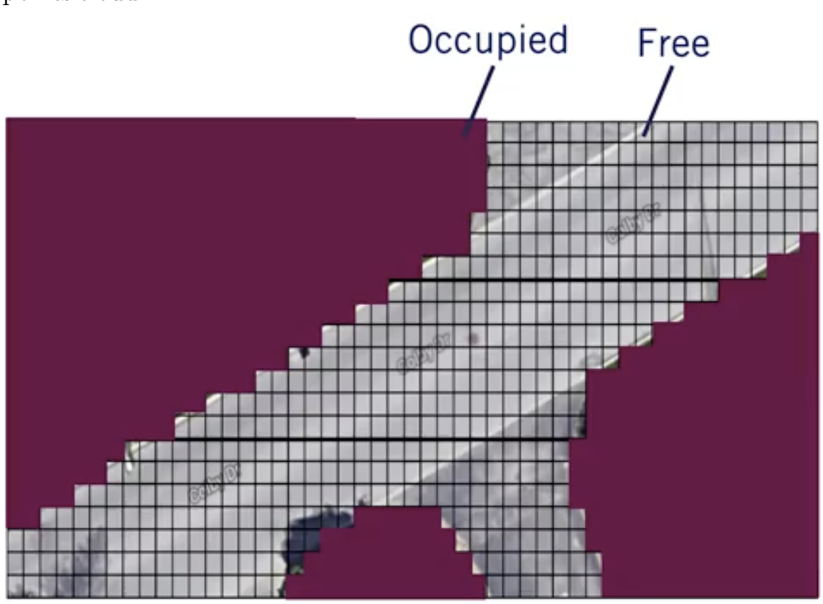


Software Architecture



Environment Representation

There is three types of maps commonly used in self-driving:

- The localization map with points cloud 
- The occupancy grid map 

- The detailed road map [](Course_1/

detailed_road.png)

Safety Assurance for Self-Driving Vehicles

First, we see some of the autonomous driving accidents.

Safety terms:

- Harm : Refer to the physical harm to a living thing.
- Risk : Probability that a event occurs + severity of the harm of the event

Major Hazard Sources:

- Mechanical
- Electrical
- Hardware
- Software
- Sensors
- Incorrect planning or decision making
- Fallback of the human
- Cyber-attack

The National Highway Transportation Safety Administration or NHTSA, has defined a twelve-part safety framework to structure safety assessment for autonomous driving.

Industry Methods for Safety Assurance and Testing

According to industry methods, in California, currently , Waymo had driven 563 000km with 63 disengagements, it's 1 disengagement every 9000km. GM (General Motor) had driven 210 000km with 105 disengagement, it's 1 disengagement every 2000km.

The question is to know: How many miles (in year) would autonomous vehicles have to be driven to demonstrate with 95% confidence their failure rate to within 20% of the true rate of 1 fatality per 146 millions km (= Human capabilities)

The answer is : " It would take at least 400 years to do so with a fleet of 100 vehicles traveling 24/7"

Safety Frameworks for Self-Driving

There exists simple analytic framework: - Fault trees and probabilistic fault trees - Failure modes and effect analysis

And functional safety framework: - FuSa HARA : Safety requirements trough risk analysis - SOTIF : Behavior risk assessment

Introduction

In this course, we're going to dig into the other side of the self-driving problem and learn about different types of sensors and how we can use them for state estimation and localization in a self-driving car. State estimation is the process of determining the best value of some physical quantity from what are typically noisy measurements.

For self-driving, one of the most important types of state estimation is localization, which is the process of determining where the car is in the world and how it's

moving.Thy is why state estimation is so important.

Squared Error Criterion and the Method of Least Squares

To approximate the value of a parameter, usually, we base our approximation to minimize the squared error criterion. To this, we use the Method of Least Square, which come to use the arithmetic mean. However, in Self-Driving car, we may want to trust certains measurements more than the others due to better sensors.

This is why we prefere to use Weighted Least Squares. Here is a recap of the difference between the two methods:

	Least Squares	Weighted Least Squares
Loss / Criterion	$\mathscr{L}_{\text{LS}}(\mathbf{x}) = \mathbf{e}^T\mathbf{e}$	$\mathscr{L}_{\text{WLS}}(\mathbf{x}) = \mathbf{e}^T\mathbf{R}^{-1}\mathbf{e}$
Solution	$\hat{\mathbf{x}}_{\text{LS}} = (\mathbf{H}^T\mathbf{H})^{-1}\mathbf{H}^T\mathbf{y}$	$\hat{\mathbf{x}}_{\text{WLS}} = (\mathbf{H}^T\mathbf{R}^{-1}\mathbf{H})^{-1}\mathbf{H}^T\mathbf{R}^{-1}\mathbf{y}$
Limitations	$m \geq n$	$m \geq n$ $\sigma_i^2 > 0$

Recursive Least Squares

To use the precedent Least Square's Methods, we made a assumption: that we have a batch of data. This is assumption is regurlarly false in Self Driving Cars. So, we use the Recursive Least Squares in order to produces a 'running estimate' of parameter(s) for a stream of measurements. It's a recursive linear estimator that minimizes the variance of the parameters at the current time.

Least Squares and the Method of Maximum Likelihood

A self-driving car will have to deal with many, many sources of error, some of which are very difficult to model. However, the central limit theorem tells us that when combining all of these errors together, they can reasonably be modeled by a single Gaussian error distribution. We would like to model our system probabilistically and yet maintain simplicity in calculations. If our errors

are Gaussian, then the best maximum likelihood estimate of the parameters of interest is exactly the least squares solution we're already familiar with, easy.

Warning: Outliers can significantly affect our estimate value!

The (Linear) Kalman Filter

While Recursive least squares updates the estimate of a static parameter, Kalman filter is able to update and estimate of an evolving state. It use two steps to take a probabilistic estimate of the state and update it in real time: prediction and correction.

In this way, we can think of the Kalman filter as a technique to fuse information from different sensors to produce a final estimate of some unknown state, taking into account, uncertainty in motion and in our measurements.

Kalman Filter and The Bias BLUEs

During this lesson, we have seen that given our linear formulation, and zero-mean , white-noise: The Kalman Filter is unbiased. We have also seen that it is consistent.

If we have white , uncorrelated zero-mean noise, the Kalman filter is the best unbiased estimator that uses only a linear combination of measurements. So, we call it the BLUE (Best Linear Unbiased Estimator)

Going Nonlinear - The Extended Kalman Filter

The classic Kalman Filter is linear. However, in reality, there is no full linear model. So, we discover the Exttended Kalman Filter.

The EKF is designed to work with nonlinear systems, and it's often considered one of the workhorses of state estimation It uses first-order linearization to turn a non-linear problem into a linear one.

Linearization works by computing a local linear approximation to a nonlinear function using a first-order Taylor series expansion about an operating point.

An Improved EKF - The Error State Extended Kalman Filter

True state: ex : True position Nominal State: Calculate position Error state: True position - Calculate position

Instead of doing Kalman filtering on the full state which might have lots of complicated non-linear behaviors, we're going to use the EKF to estimate the

error state instead, and then use the estimate of the error state as a correction to the nominal state. It's called the ES EKF (Error State Extended Kalman Filter)

Limitations of the EKF

If the dynamics of the system being modeled are highly non-linear or the linearization error is large, the filter may diverge. This means that linearization error can cause our estimator to be overconfident in a completely wrong answer. Also, the EKF requires Jacobian matrices to be computed which is often a tedious and error-prone process.

An Alternative to the EKF - The Unscented Kalman Filter

Intuition of the Unscented Transform: "It's typically much easier to approximate a probability distribution, than it is to approximate an arbitrary nonlinear function"

UKF uses the Unscented Transform to adapt the Kalman Filter to nonlinear systems. The Unscented Transform works by passing a small set of carefully chosen samples through a nonlinear system and computing the mean and covariance of the outputs, and it often does a much better job of approximating the output distribution than the local analytical linearization technique used by the EKF for similar computational cost.

Summarize

	EKF	ES-EKF	UKF
Operating Principle	Linearization (Full State)	Linearization (Error State)	Unscented Transform
Accuracy	Good	Better	Best
Jacobians	Required	Required	Not required
Speed	Slightly faster	Slightly faster	Slightly slower

3D Geometry and Reference Frames

Vector quantities can be in different reference frames trough rotations and translations. First, we talk about different representation of rotations, with their advantages and drawbacks, let's have a recap here:

	Rotation Matrix	Unit quaternion	Euler angles
Expression	$\mathbf{C}$	$\mathbf{q} = \begin{bmatrix} \cos\frac{\phi}{2} \\ \hat{\mathbf{u}}\sin\frac{\phi}{2} \end{bmatrix}$	$\{\theta_3, \theta_2, \theta_1\}$
Parameters	9	4	3
Constraints	$\mathbf{C}\mathbf{C}^T = \mathbf{1}$	$\lvert\mathbf{q}\rvert = 1$	*None**
Singularities?	No	No	*Yes!*

For localization, ECEF (Earth-Centered Earth-Fixed Frame), ECIF (Earth-Centered Inertial Frame) and Navigation frames are important.

The Inertial Measurement Unit (IMU)

An IMU is composed of:

- gyroscopes (measure angular rotation rates about three separates axis)
- accelerometers (measure accelerations along three orthogonal axes)

IMUs are tricky to calibrate and drift over time so we'll use the modern system of global navigation satellites to periodically correct our posed estimates.

The Global Navigation Satellite Systems (GNSS)

Global Navigation Satellite Systems work by combining pseudoranges from at least four satellites to determine a 3D position.

It exist different improvents of the GPS, let's see:

Basic GPS	Differential GPS (DGPS)	Real-Time Kinematic (RTK) GPS
mobile receiver	mobile receiver + fixed base station(s)	mobile receiver + fixed base station
no error correction	estimate error caused by atmospheric effects	estimate relative position using phase of carrier signal
~10 m accuracy	~10 cm accuracy	~2 cm accuracy

Light Detection and Ranging Sensors

LIDAR sensors use lasers pulses and time-of-flight to measure distances to objects along a specific direction. For 2D or 3D LIDARs work by sweeping the laser pulses in many directions across the whole environment. Considering the operation of a LIDAR, we should be aware of some sources of measurement noise:

- Uncertainty in determining the exact time of arrival of the reflected signal.
- Uncertainty in measuring the exact orientation of the mirror.
- Interaction with the target (ex: surface absorption)
- Variation of propagation speed.

Also, for a moving vehicle, each point in a scan is taken from a slightly different place. We need to take this into account, otherwise, for a quickly moving vehicle, the motion distorsion can become a problem.

LIDAR Sensor Models and Point Clouds

We will need to do 3 operations on Point Clouds:

- Translation
- Rotation
- Scaling

For this, we will use a data structure like this:

We assign an index to each of the points, say point 1 through point n, and store the x, y, z coordinates of each point as a 3 by 1 column vector. We stack them side by side into a matrix that we'll call big P. Doing it this way make it easier to work with the standard linear algebra libraries, like the Python NumPy library, which lets us take advantage of fast matrix operations.

For doing basic and advanced operations on point clouds, we have the open-source Point Cloud Library (PCL) built with C++ but it exists unofficial python binding available.

Pose Estimation from LIDAR Data

To determine the motion of a self-driving car by aligning points clouds from LIDAR, we use ICP (Iterative Closest Point) algorithm. It works by iteratively minimizes the distance between points in each point clouds.

Moving objects violate the stationary world assumption that ICP is based on, so it can be a problem. These outlier measure can be mitigated by using Robust Loss Functions which assign less weight to large errors than the usual squared error loss.

State Estimation in Practice

In practice, state estimation typically fuses data from multiple sensors like IMUs, LiDAR, cameras, and GPS or GNSS receivers. For a correct sensor fusion, the calibration is very important. We need to consider the relative positions and orientations of all the sensors and any differences in polling time. We need to think about how to safely cope with localization failures and aspects of the world that do not conform to our assumptions such as moving objects.

Multisensor Fusion for State Estimation

For vehicular state estimation, we use EKF(Extended Kalman Filter). In order to fuse GNSS with IMU and LIDAR measurements.

For this we made some assumption:

- LIDAR provides position in the same reference frame as GNSS.
- IMU has no biases.
- State initialization is provided.
- Our sensors are spatially and temporally aligned.

Sensor Calibration - A Necessary Evil

Sensor fusion is impossible without calibration, for this, we have to deal with 3 types of calibatrion:

- Intrinsic Calibration: sensors specific parameters.
- Extrinsic Calibration: how the sensors are positioned and oriented on the vehicle.

- Temporal calibration: Time offset between different sensors measurements

Loss of One or More Sensors

We have seen different examples of car crashing because of a sensor malfunction and a bad management of this failure. For this, multiple sensors are crucial to robust localization in varied environment.

Visual Perception for Self-Driving Cars

The goal of the perception stack is to provide the motion planning stack enough information about the driving environment so that the latter can achieve its mission in a safe and efficient way.

The perception task is (in a way) already solved. We know how to do object recognition, classification, segmentation, even monocular depth estimation. We can do instance segmentation and track dynamic objects as long as they're covered by camera/LIDAR. The only remaining question is what does the motion planner needs ?

This however is still an area of active research. We could look at it in 2 different ways: 1. End-to-end solution, the vision stack outputs the behavior. 1. Output as much info as possible from the vision stack, then find a way to solve motion planning.

In this part, we'll talk about the second solution. Let's assume, for the sake of simplicity, that we need to know 3 things to drive car: 1. We need to know where should we drive, i.e. road and lane lines 1. We should know the position of the other agents (cars, pedestrains, etc.) in the same environment, so that we can drive without colliding. 1. We have to be aware of the regulatory elements in our driving environment (stop signs, crosswalks, etc.)

All of this can be achieved just with an RGB camera input, with just 3 deep learning models.

Segmentation

Image segmentation is the process of classifying every pixel of an image into a given set of labels. In this case, we can think of different types of labels: dynamic object, road, regulatory element, etc.

Object Recognition

Object recognition is the process of recognizing and classifying objects in a driving scene. The ouput is a set of bounding boxes and the corresponding labels

Figure 1: segmentation

Depth Estimation

Depth estimation is the process of estimating a pixel's depth. Depth estimation is usually done with stereovision (2 cameras). However, recent advances in deep learning has made it possible to train unsupervised models to depth estimation.

Putting it all together

If we piece it all together, we get the following:

Motion Planning for Self-Driving Cars

The motion planning task can be grouped into a list of behaviours: * follow lead car * make a U-turn * decelerate to stop * yied * etc.

The motion planner uses a behavioural planner to choose a behaviour according to the driving scenario. And then a local planner computes, for this behavior, the path and velocity profile.

This path have to be feasible and collision free. So the following constraints apply: * Curvature of the path can't exceed a limit * Maximum magnitude of lateral forces on the tires before stability loss * Static obstacles that blocks portions of the drivable space * Dynamic obstacles * Rules of the road and regulatory elements

Behaviour planner

Typically uses one of the 3 following architectures: * Finite state machines. Set of states and their transitions. * Rule-based systems. A set of if conditions. * Reccurent learning based methods.

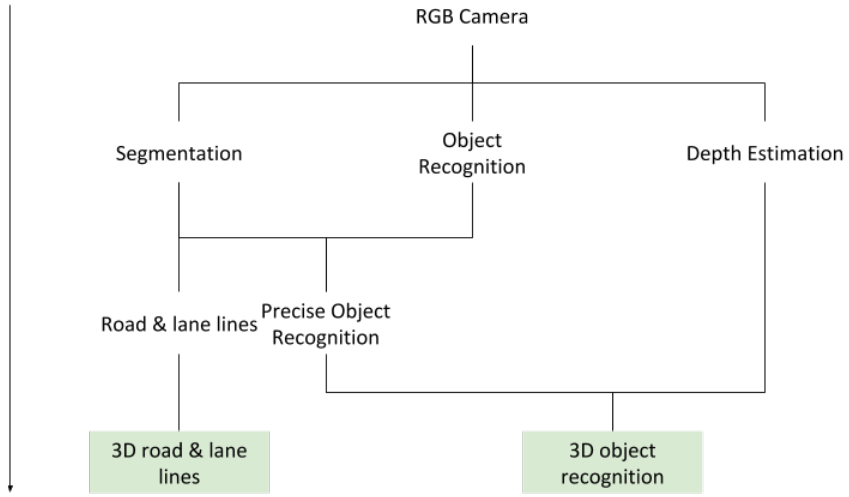


Figure 2: vision

It takes as inputs: * High definiton road map (road map with all the regulatory elements) * Mission path (Start and end of the trip) * Localization * Perception stack output

And outputs: * Manoeuvre (or behaviour) * Set of constraints (optimal path if there is no other agent or obstacle, speed limit, lane boundaries)

Local planner

For the path planning task: * Sampling based planner. Randomly samples the searchspace until a good enough path is found. * Variationnal planner. Optimizes the trajectory according to a cost function. * Lattice planner.

For the velocity profilte generation, we optimize the following values: * Smoothness. Minimize jerk (derivative of the acceleration) * Deviation from reference, i.e. optimal path (which can be unfeasible) * Acceleration limit