NO HANDS

CS 3892: FINAL PROJECT

Neelasha Bhattacharjee, Rachel Hawkins, Amina Irizarry, Katherine Oung

OVERVIEW

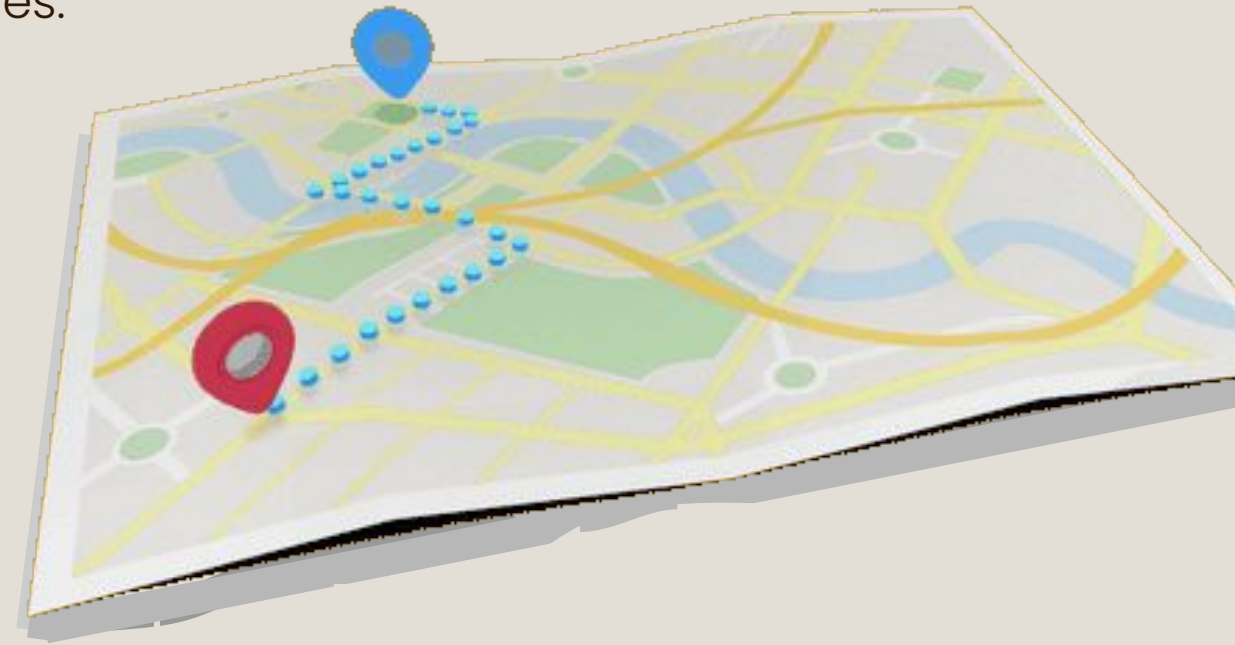
GOALS & EXPECTATIONS

- Adhere to speed limits, stop signs, and lane boundaries
- Successfully follow a path to “pick up zones” and “drop off zones”
- Avoid collisions with other cars

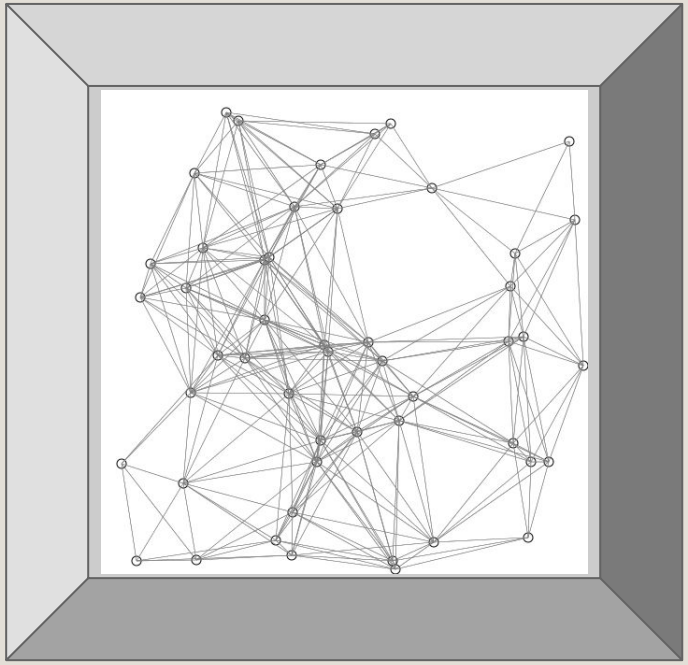
ROUTING

Determines an **optimal path** for the vehicle to travel towards its target destination while adhering to traffic rules.

This provides the overall directive for motion planning.



Finding Our Route

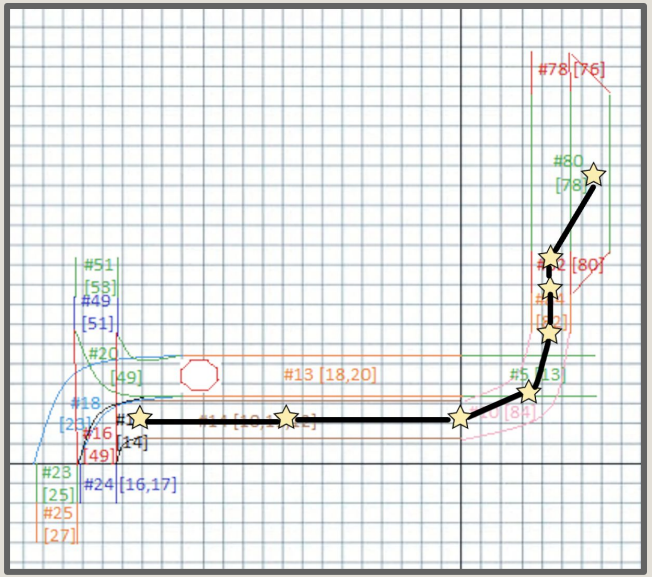


Use Dijkstra’s shortest path algorithm to find an efficient way to reach target segments

MOTION PLANNING

Create polyline based on the road segments in the map

- If the curvature>0 or the length of the segment >80, add a middle point between these values
- If its a loading zone, get the loading center, so the car will turn into it
- Store if the segment has a stop sign at its end



MOTION PLANNING

Implement pure pursuit controller

- A pure pursuit controller picks a best next point to get the car to, hoping to match the given polyline
- Added a modification to “try” three points on the polyline as the next_best location, depending on which one is closest to the lookahead distance
- Error adjustment is mainly shown through zig-zagging

MOTION PLANNING

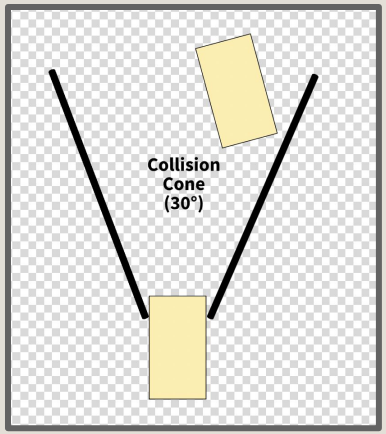
Road requirements

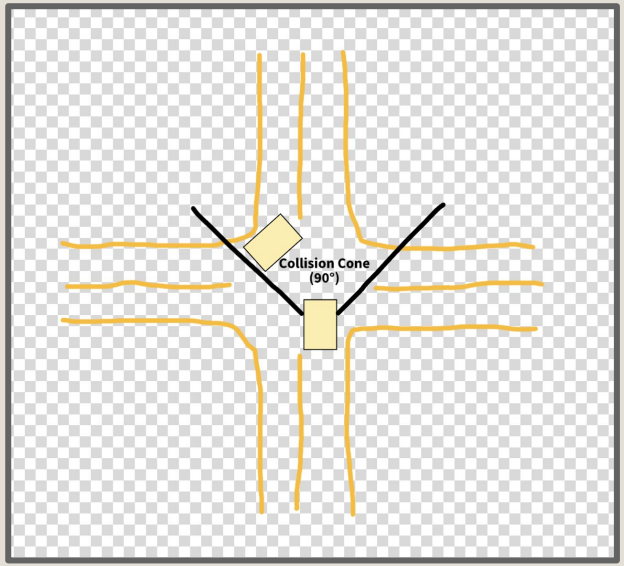
- The car detects if there is a stop sign in at the end of the current segment in decision_making. It will slow down in target_velocity to a stop as it nears the stop sign, then start up again when the found_stop_sign flag is removed
- The car slows down at curved segments
- When the car is at the target segment loading zone, it slows to a stop before getting a new route

MOTION PLANNING

Collision avoidance

- Implement collision avoidance as a separate thread/channel
- Stop if there is a vehicle <30 away in front of ego
- Only care about vehicles in front of you (create a 30 degree cone)
- At an intersection, collision is more likely, so we increase the cone to 90 degrees





LOCALIZATION

Estimates **where** the car is located on the map, **how fast** it is moving, and **what direction** its facing.

This estimate is necessary for decision making and route following.

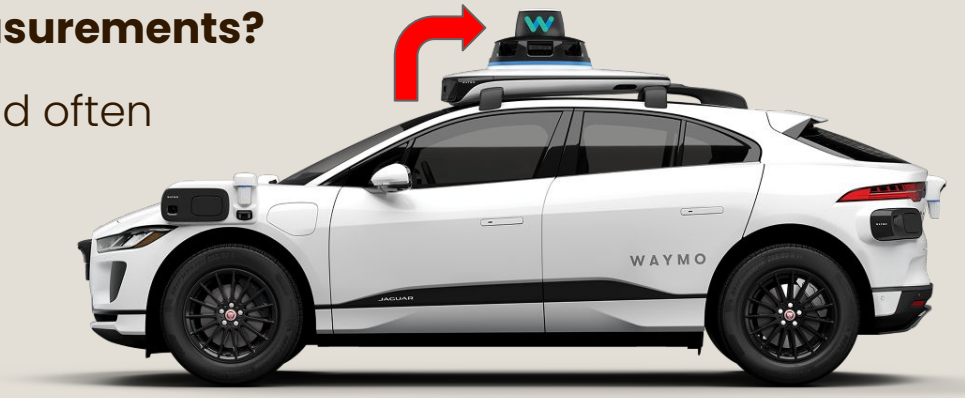
How do we localize?

GPS –> Latitude, longitude, heading

IMU –> Linear/angular velocity

Why can't we just use the GPS measurements?

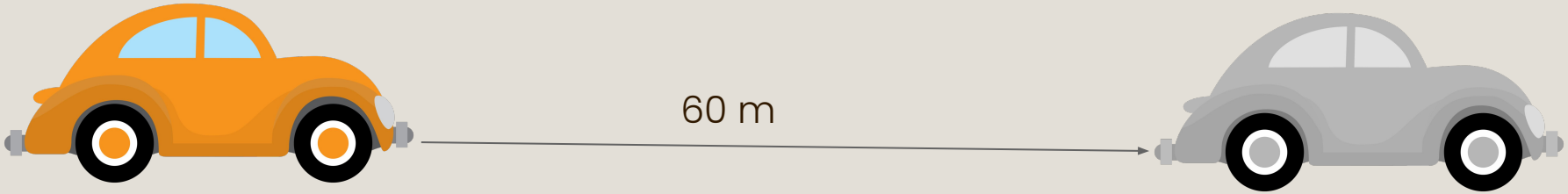
GPS measurements are **too slow** and often **unreliable**.



Extended Kalman Filter

Step 1: Predict

- Given last known state, how fast we were going, and how much time has passed since then, where should we be now?



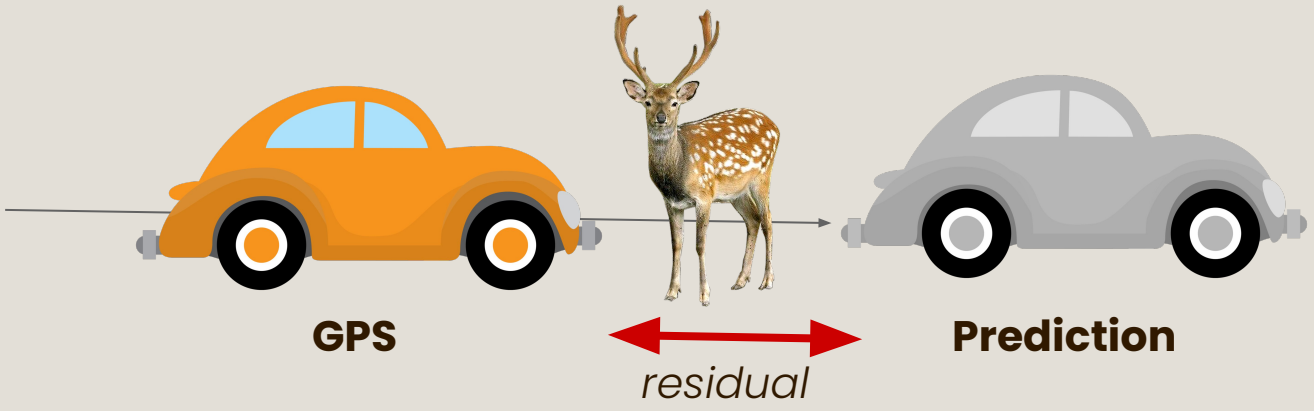
t = 0, velocity = 60 m/s

t = 1

Extended Kalman Filter

Step 2: Correct

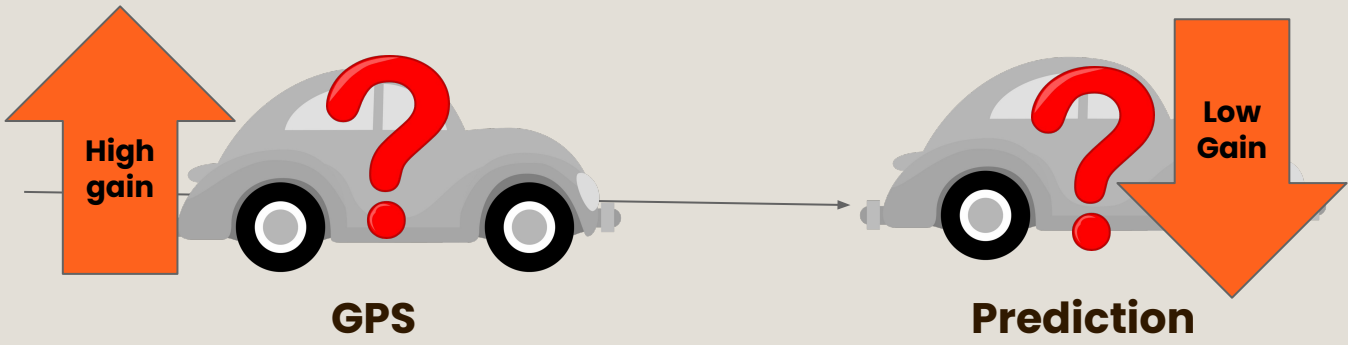
- Where does our GPS measurement say we are? How far off is that from our prediction?



Where are we actually?

The EKF algorithm knows that both our predictions and the actual GPS/IMU measurements are **predictably unreliable.**

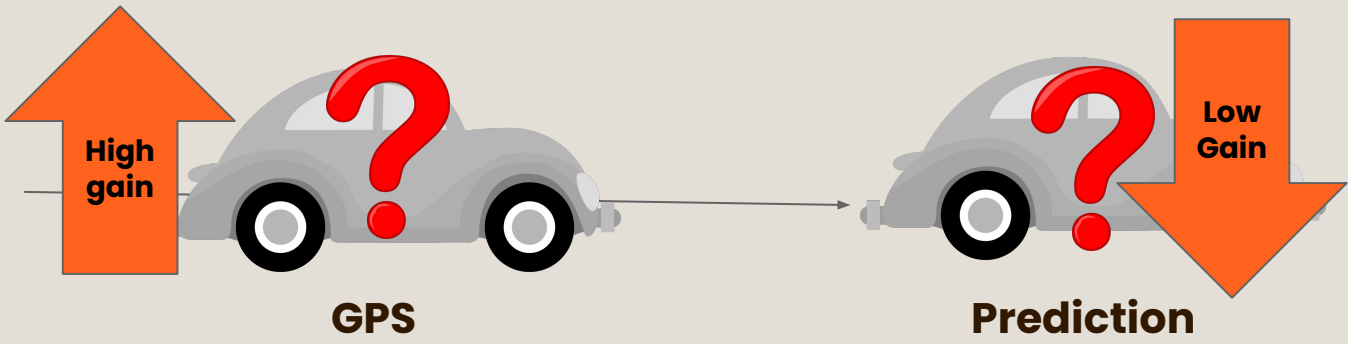
With each iteration, EKF calculates a **Kalman Gain**, which balances between trusting the new measurement or trusting the previous, predicted state



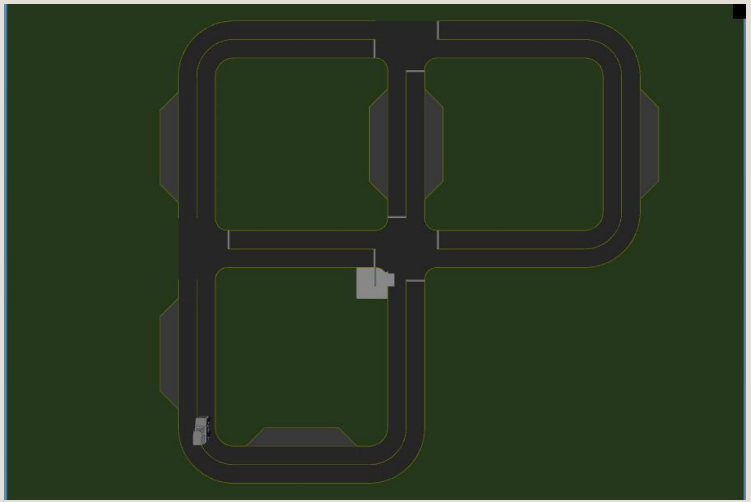
Where are we actually?

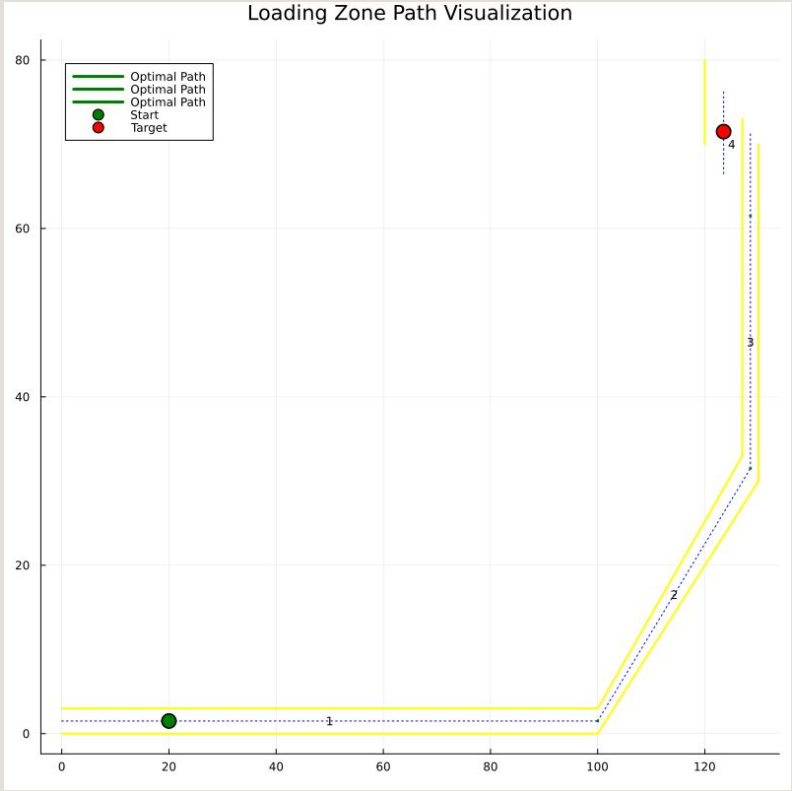
If GPS says something very different, but we think it's noisy → we only shift our state a little

If GPS says something reasonable and we're unsure of ourselves → we shift more

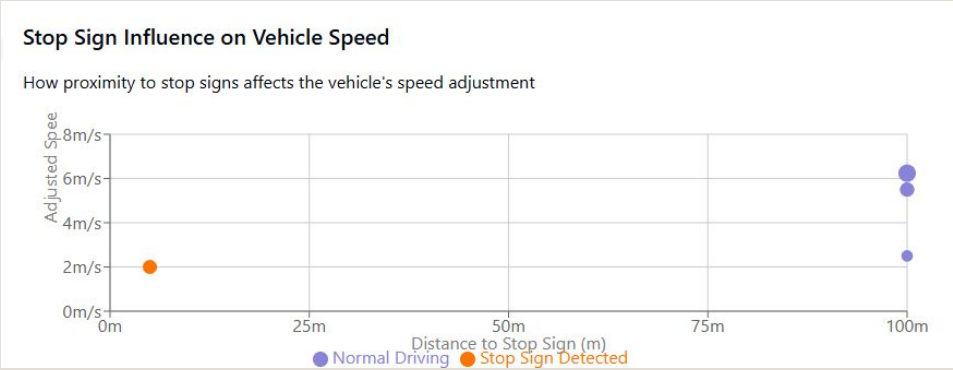


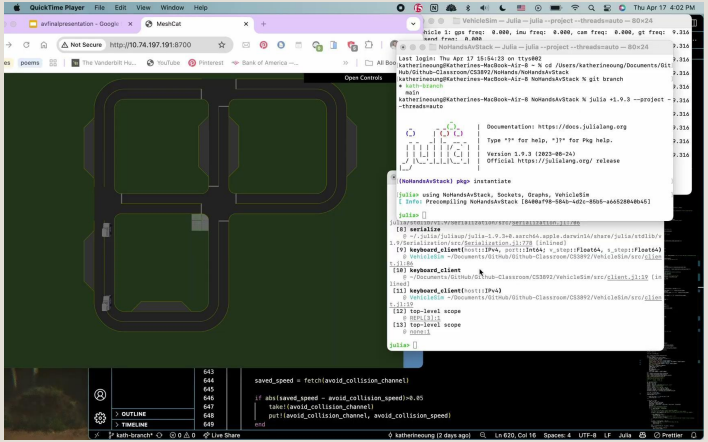
TESTING - ROUTING

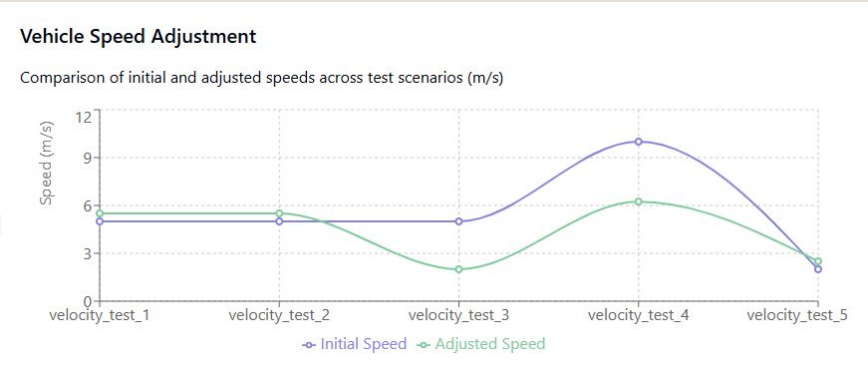




TESTING - MOTION PLANNING

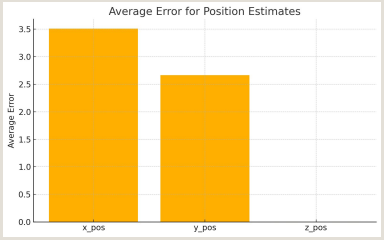






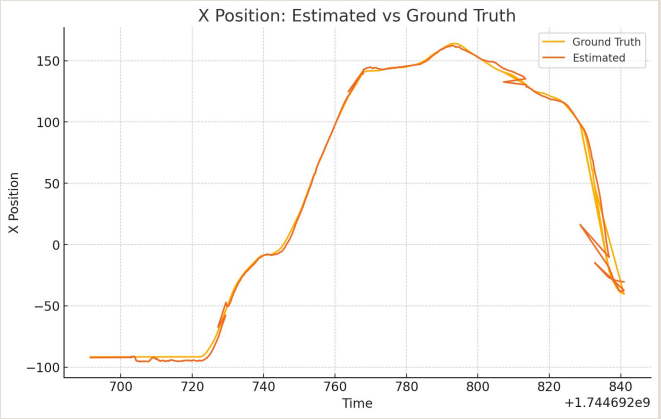
TESTING – LOCALIZATION

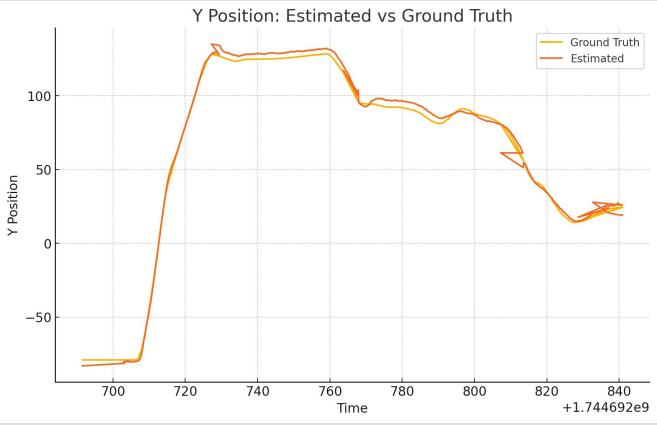
Positional error

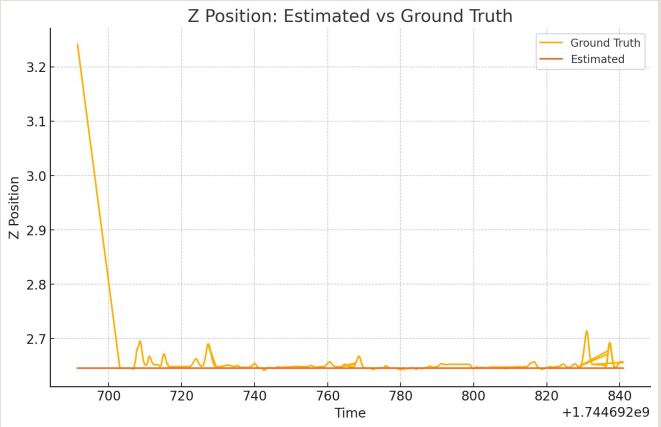


Average Position Errors

	Position Axis	Average Error
1	x_pos	3.508589353652426
2	y_pos	2.663504009068498
3	z_pos	0.005363020645858058





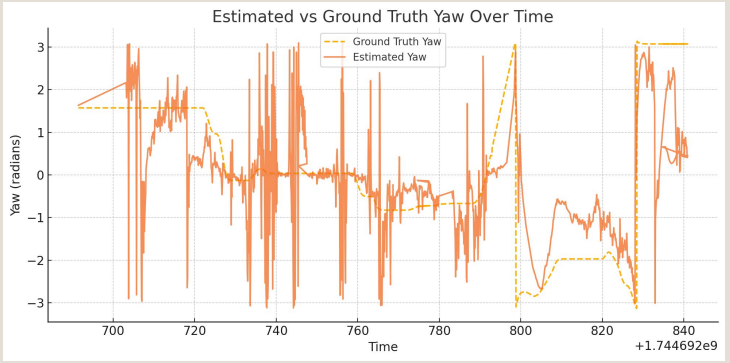


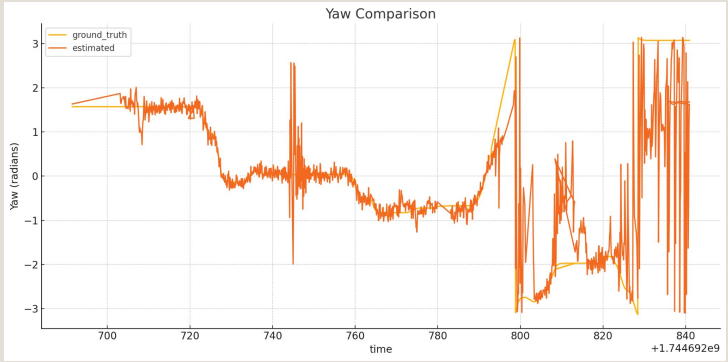
TESTING – LOCALIZATION

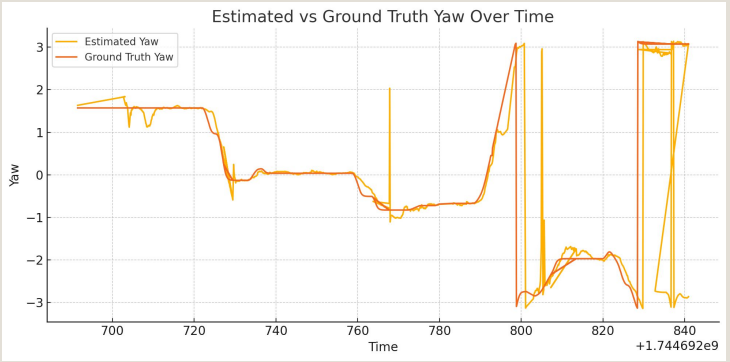
Lowering orientation error by adjusting process and measurement noise

Yaw Error Table

	Component	Average Error
1	Yaw	0.584

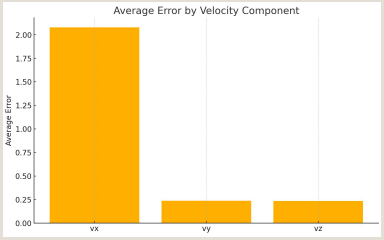






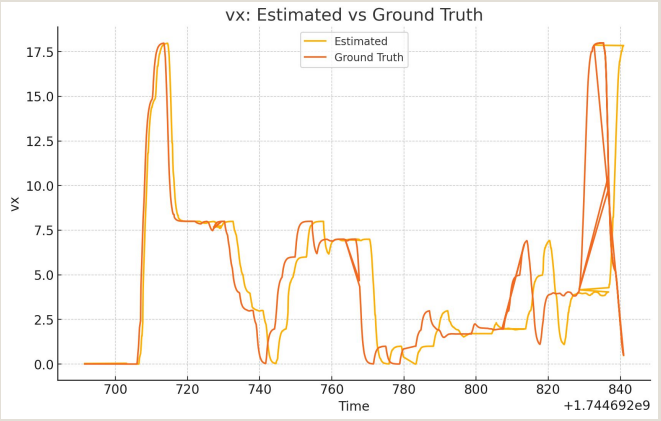
TESTING – LOCALIZATION

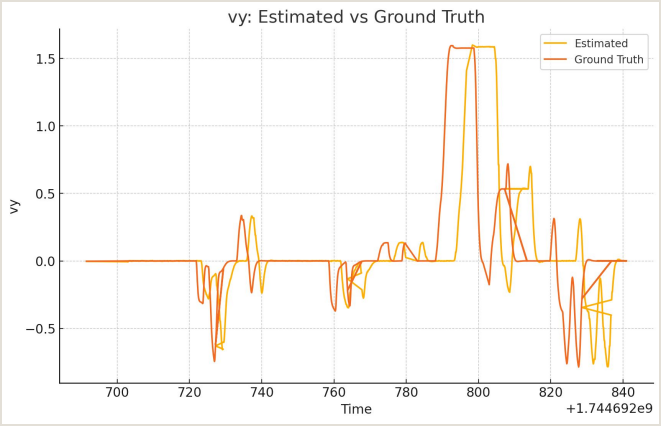
Positional velocity error

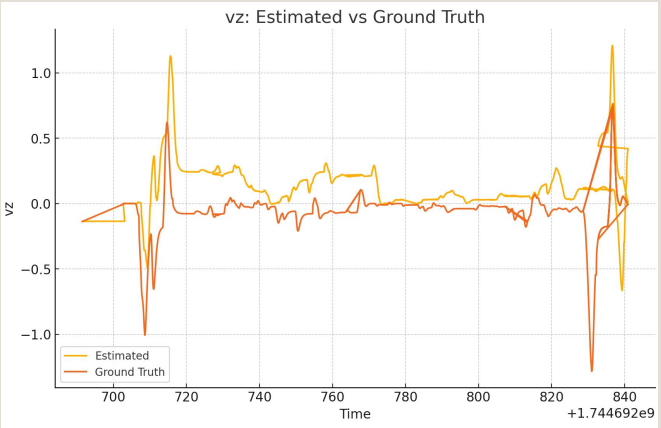


Velocity Component Errors

	Component	Average Error
1	vx	2.078
2	vy	0.236
3	vz	0.234

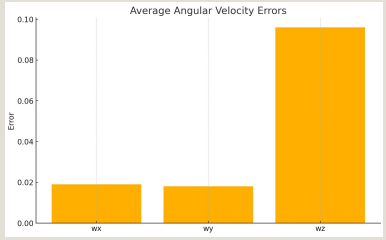






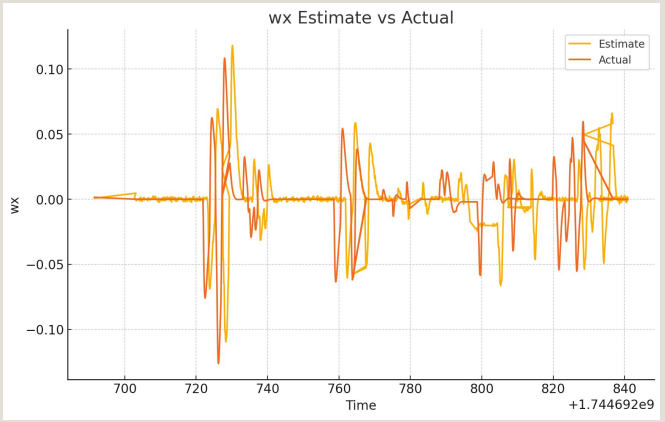
TESTING – LOCALIZATION

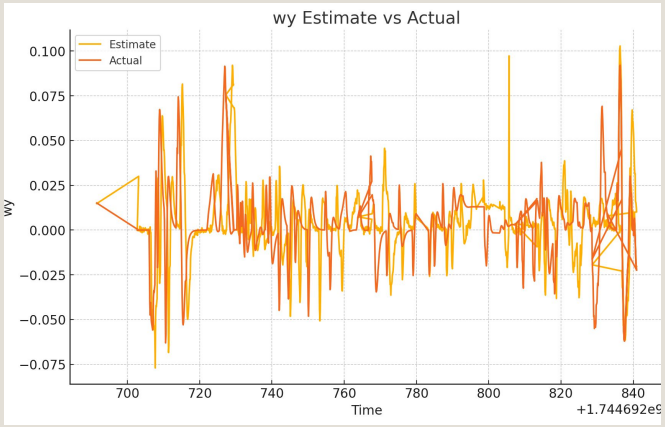
Angular velocity error

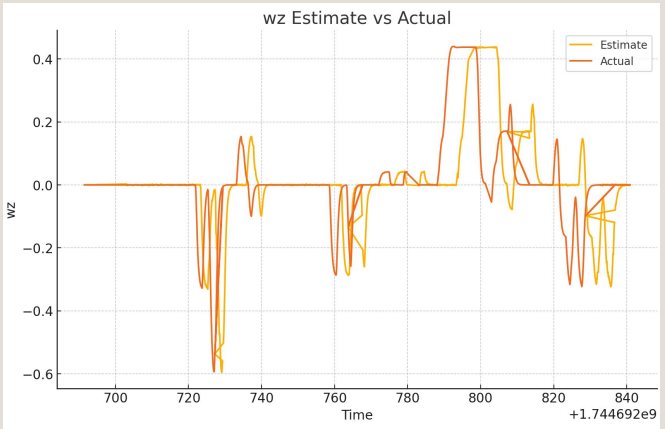


Angular Velocity Errors

	Component	Average Error
1	wx	0.019
2	wy	0.018
3	wz	0.096







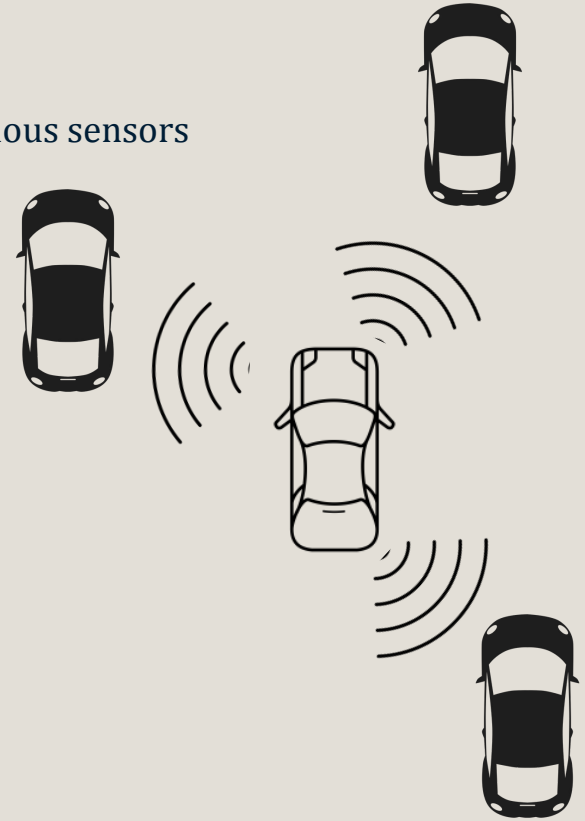
PERCEPTION

perception: vehicle's ability to interpret its surroundings using various sensors

Perception answers the following questions:

1. How many cars are around me?
2. Where exactly are all of them?
3. What is their velocity?
4. Which way are they heading

Perception helps with decision making & collision avoidance.

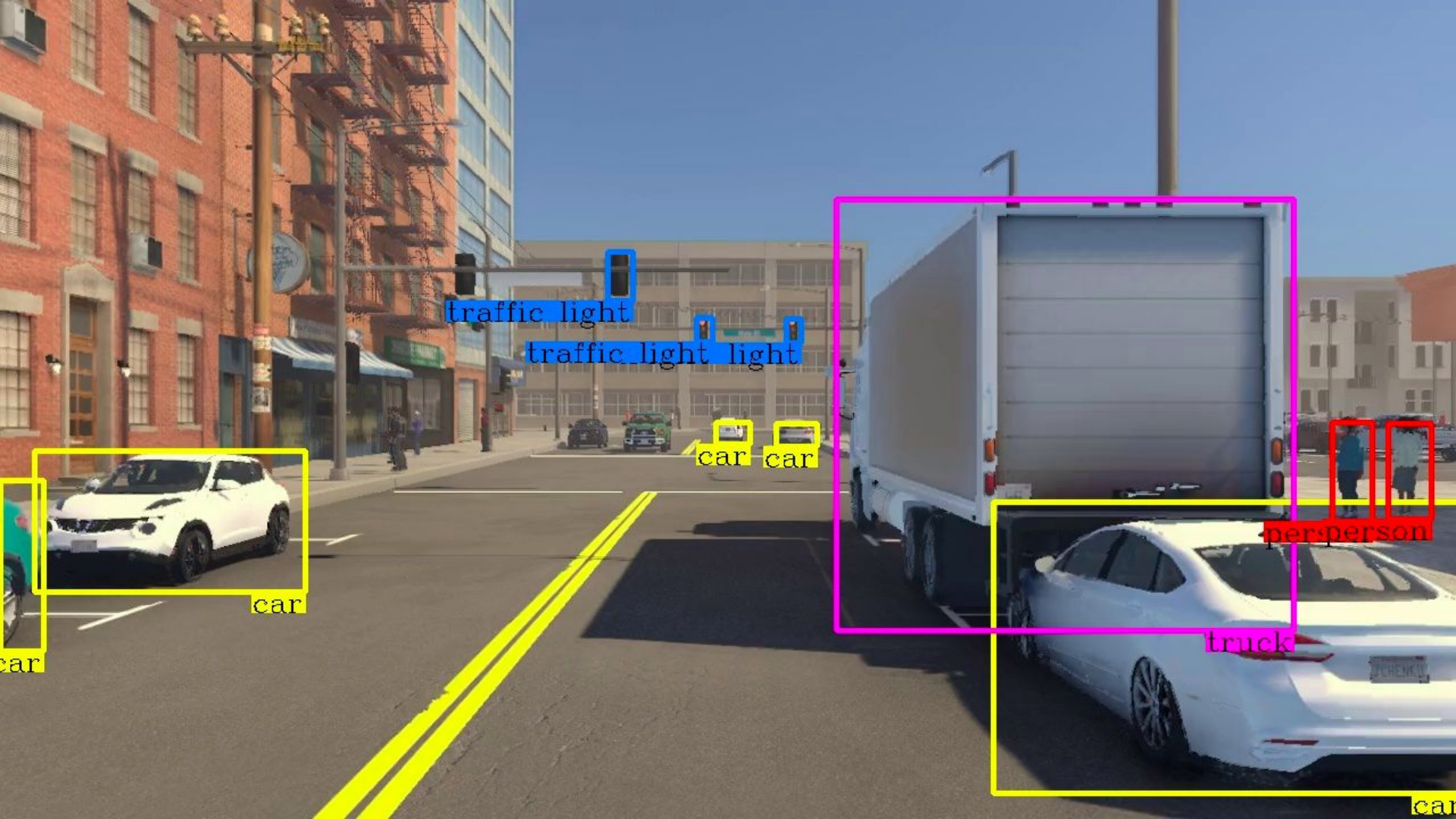


PERCEPTION

Keeps Tracks Of: All previously recorded objects and their related information

```
struct MyPerceptionType
    time::Float64
    next_id::Int
    tracked_objs::Vector{TrackedObject}
end
```

```
struct TrackedObject
    id::Int
    time::Float64
    pos::SVector{3, Float64}  # x, y, z
    orientation::SVector{4, Float64}
    vel::SVector{3, Float64}  # vx, vy, vz
    angular_velocity::SVector{3, Float64}
    P::SMatrix{13,13,Float64} # covariance matrix
end
```


traffic light
traffic light
car
car
car
car
truck
person
person

PERCEPTION

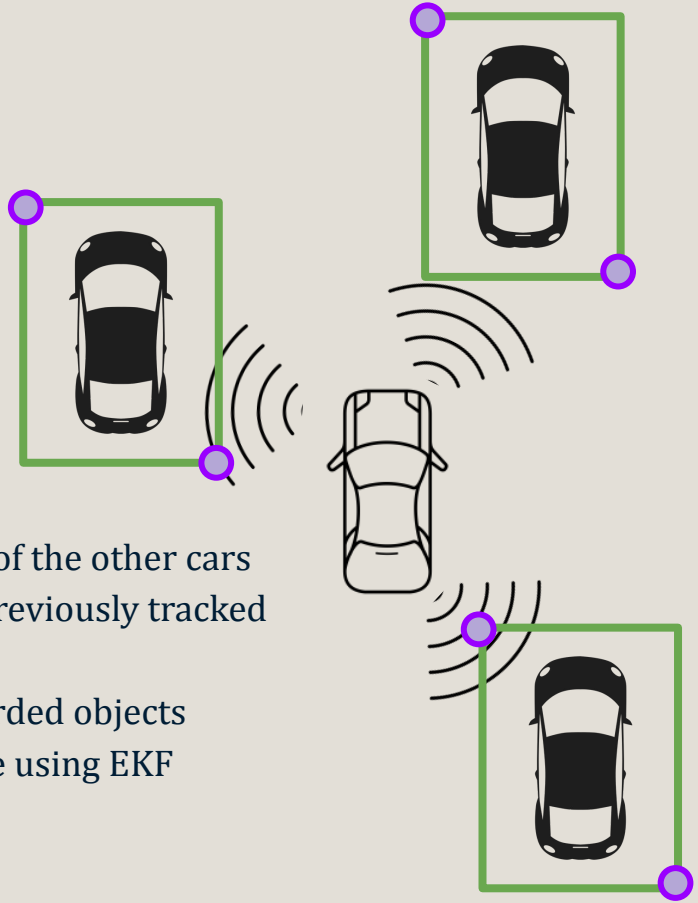
How do we get this information regarding the other vehicles?

Receive:

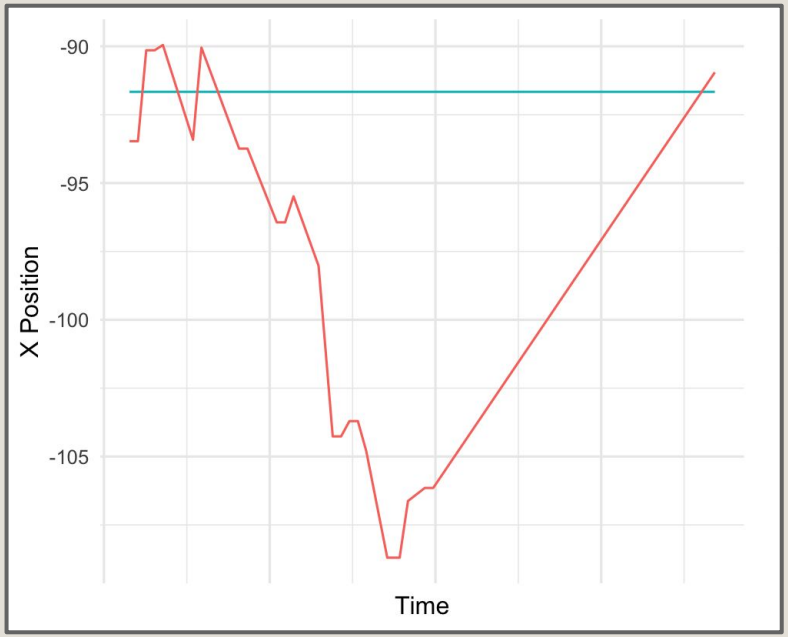
1. The TR and BL pixels of the bounding boxes
2. The current position of the ego vehicle

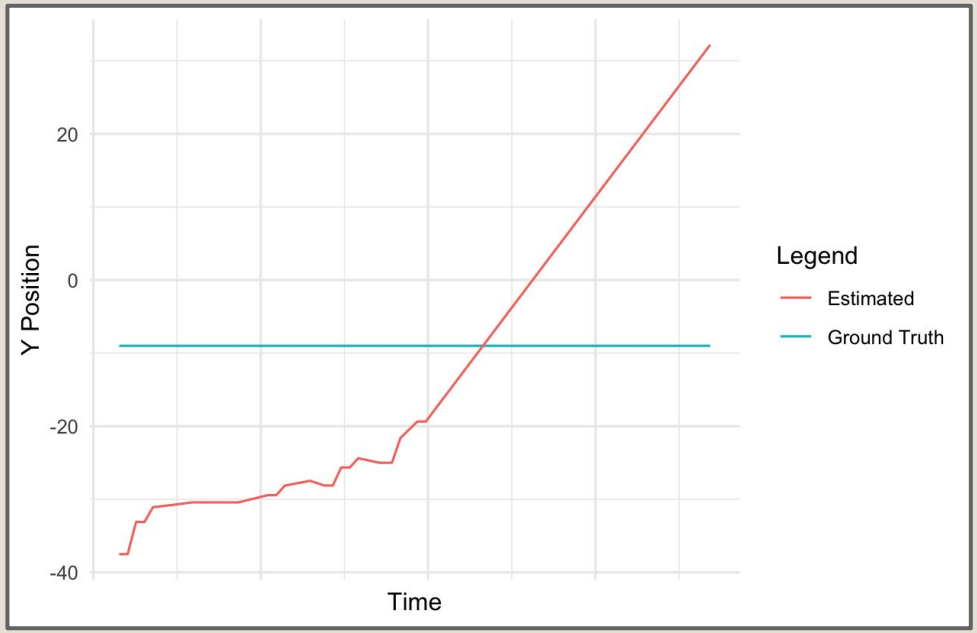
Calculates:

- Use camera information to estimate real world positions of the other cars
- Using the estimate determine: Is this a new vehicle or a previously tracked vehicle?
    - If a new vehicle: Estimate its track and save to recorded objects
    - If tracked vehicle: Update position and track vehicle using EKF

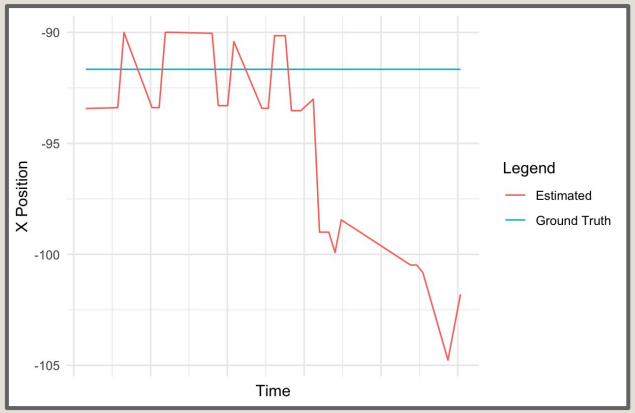


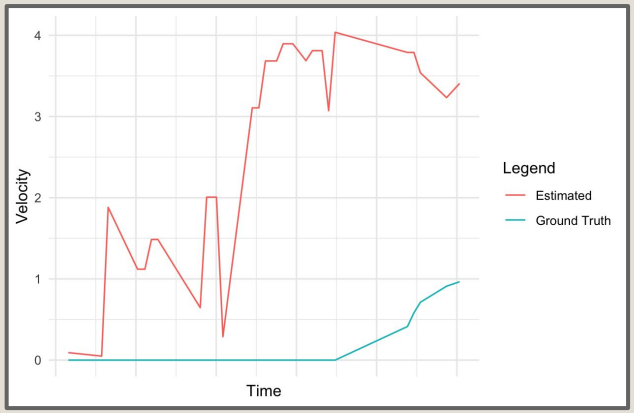
Perception Test 01 - Ego approaches stationary vehicle

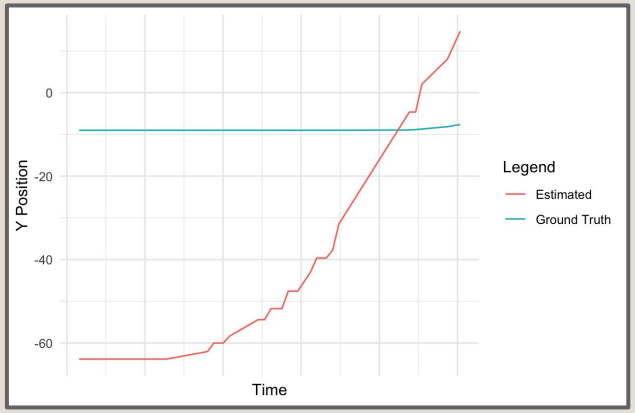


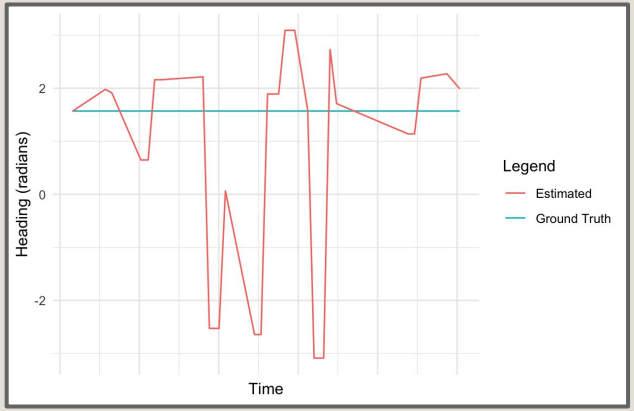


Perception Test 02 - Ego and other vehicle drive in parallel

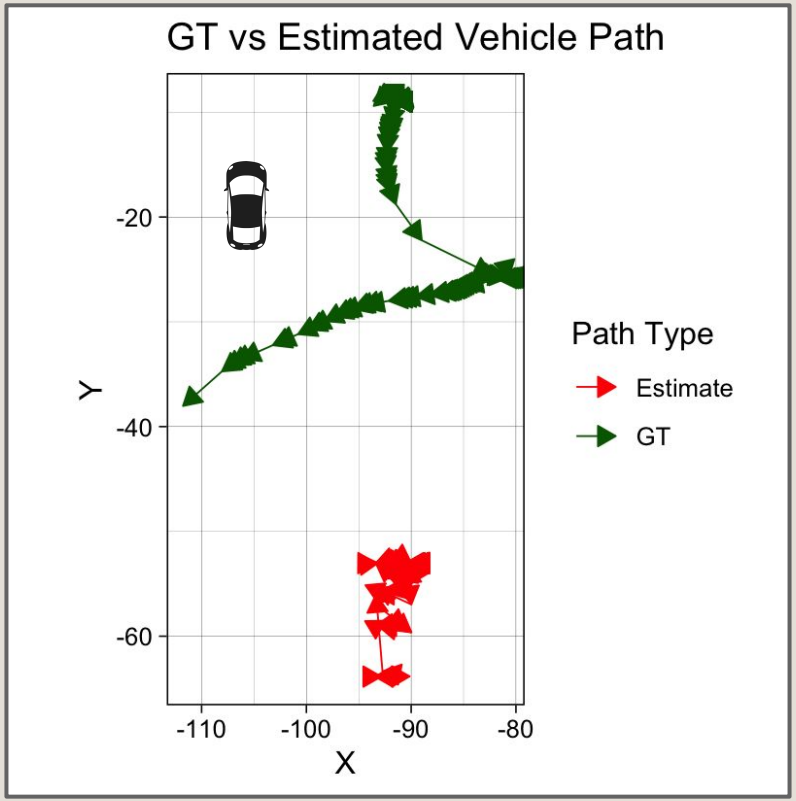




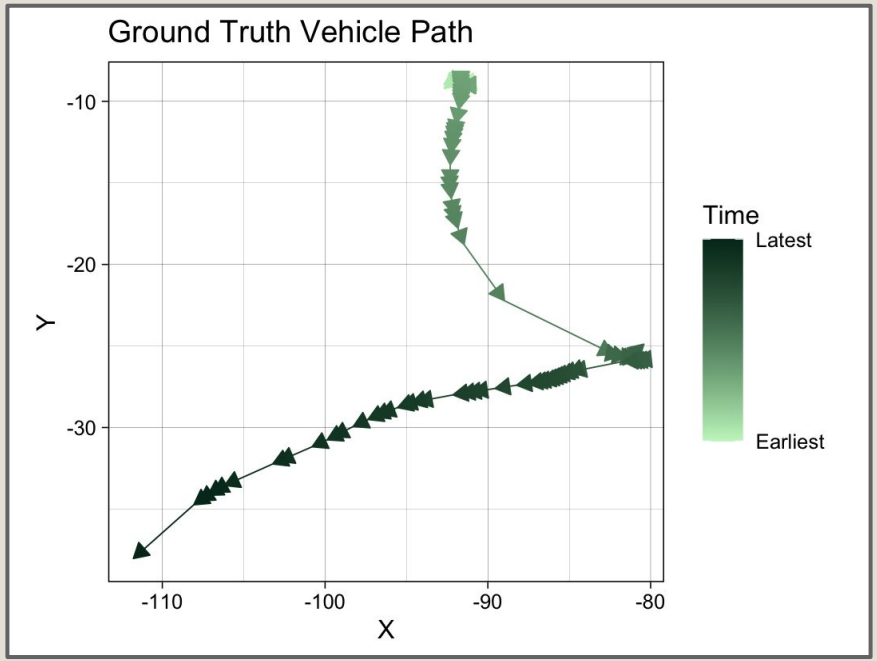


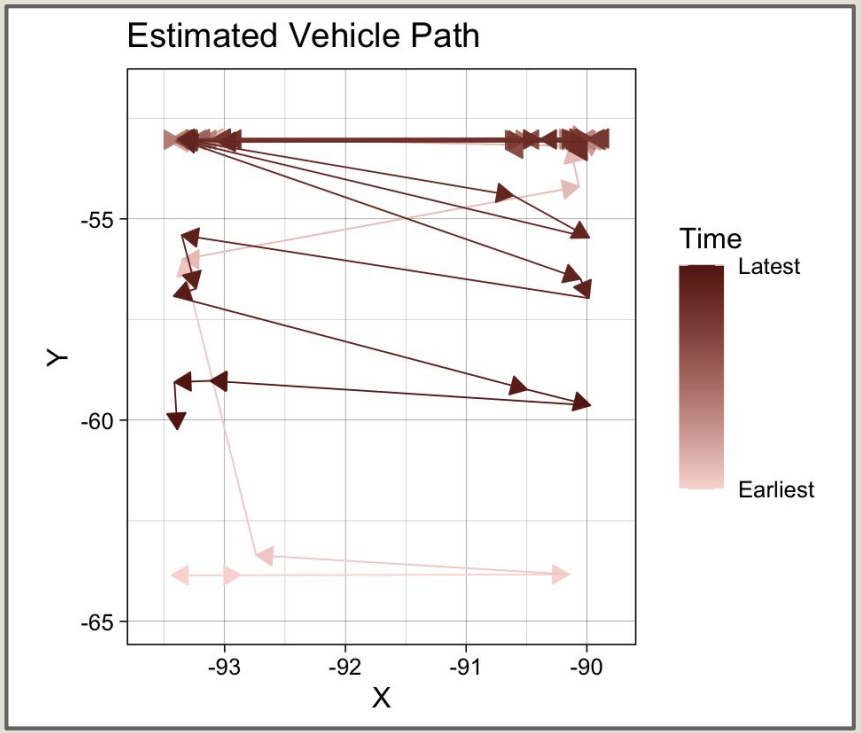


Perception Test 03 - Ego remains stationary while vehicle circles around



Perception Test 03 - Ego remains stationary while vehicle circles around





THANK YOU FOR YOUR ATTENTION <3