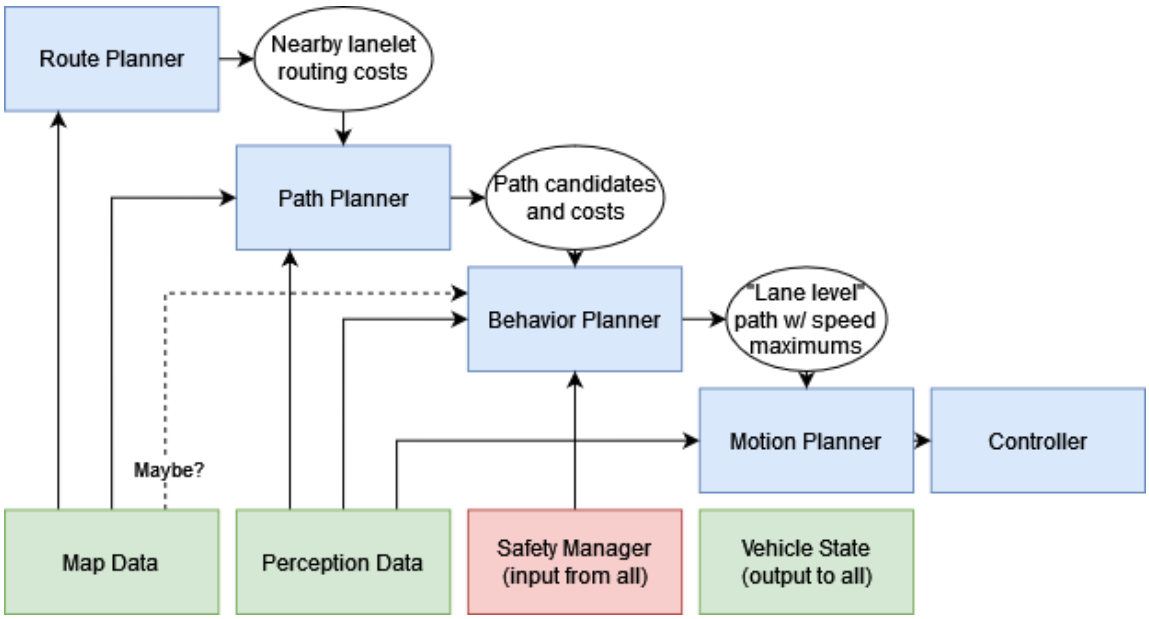
Overview

Components

Caution: some names are used for different things between this proposal and other places.

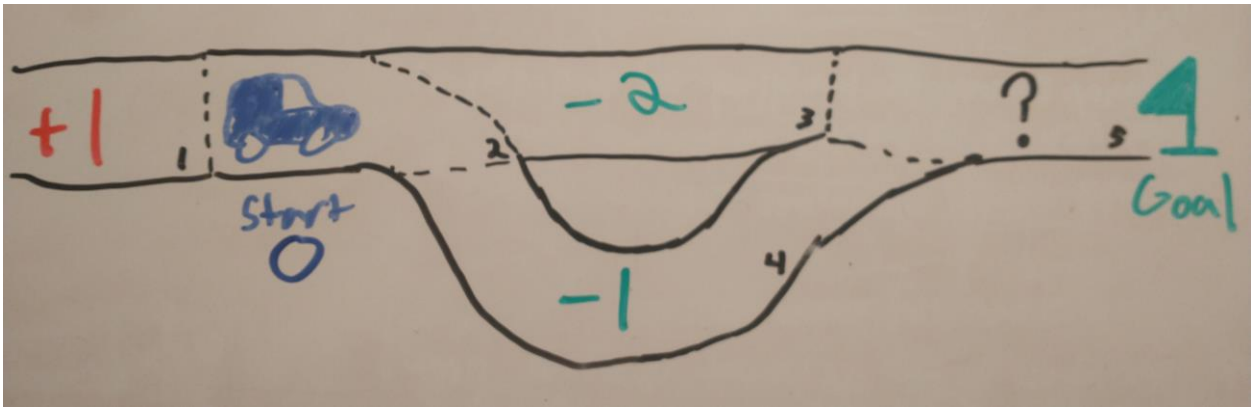
- **Route Planner:** Map-level navigation from origin to destination. Evaluates nearby lanes to determine routing cost (typically distance-to-destination).
- **Path Planner:** Generates and evaluates all lane-level candidate paths for safety and comfort cost, similar to Will's proposal and dissimilar to the current path planner.
- **Behavior Planner:** Selects lane-level path candidate to use as goal using the safety, comfort, and routing costs. Evaluates conditions to determine car state (FSM).
- **Motion planner:** Generates the immediate trajectory for the vehicle that avoids obstacles and meets physical constraints. Identical to the current meaning of the path planner.
- **Controller:** Executes the Motion Planner's trajectory.

Diagram



Route Planner

The route planner oversees map level navigation. Rather than generating a single path, it evaluates all nearby lanelets: Lanelets close to the destination are given a low cost, while lanelets far from the destination are given a high cost. The route planner may internally keep track of a single route, but this does not need to be provided.



In this example, all lanes near the car have been assigned a route cost based on the start and goal:

- Moving from the current lane to itself (lane 2 to lane 2) does not impact distance from route, so it is assigned a zero
- Moving away from the goal (lane 2 to lane 1) incurs a cost
- Moving from lane 2 to lane 3 makes progress to the goal, which means it has a negative cost.
- Moving from lane 2 to lane 4 also makes progress to the goal, but it is longer, meaning it has a greater cost. If the length of lane 4 were to increase, it could potentially incur positive cost.
- Lane 5 is not within the horizon of the car and does not need to be assigned a cost at this time.

The Lanelet2 library already includes everything necessary for this node, including determining what lanes are within a horizon of the current position.

Inputs:

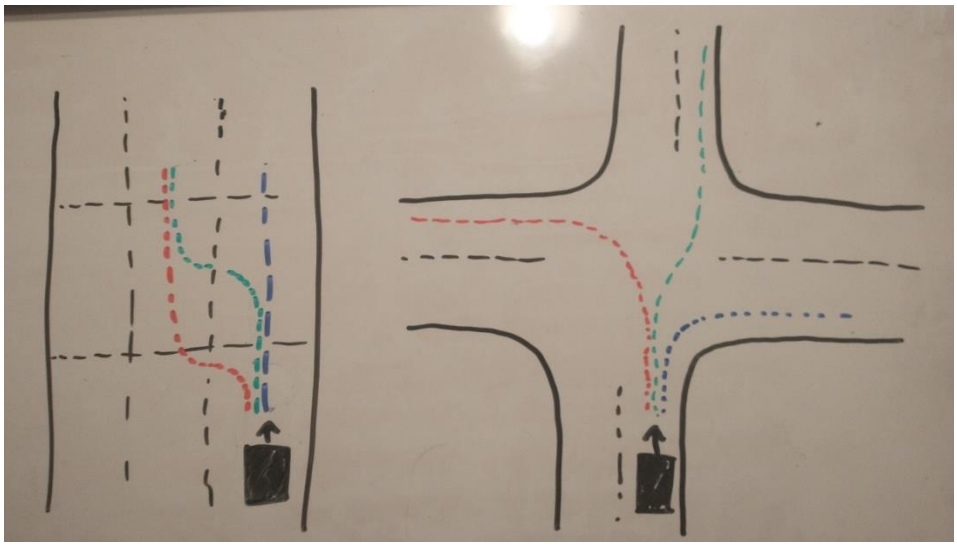
- **Static map data:** lanelet binary, as already implemented
- **Current pose:** Localized position of the vehicle on the map.
- **Destination:** Goal pose (or lanelet containing goal)

Outputs:

- <Lanelet ID, Routing cost>[]: Gives the routing cost of nearby lanes, where lower cost indicates the preferable route. All lanes within a horizon that **matches or exceeds the path planner's horizon** must be included.

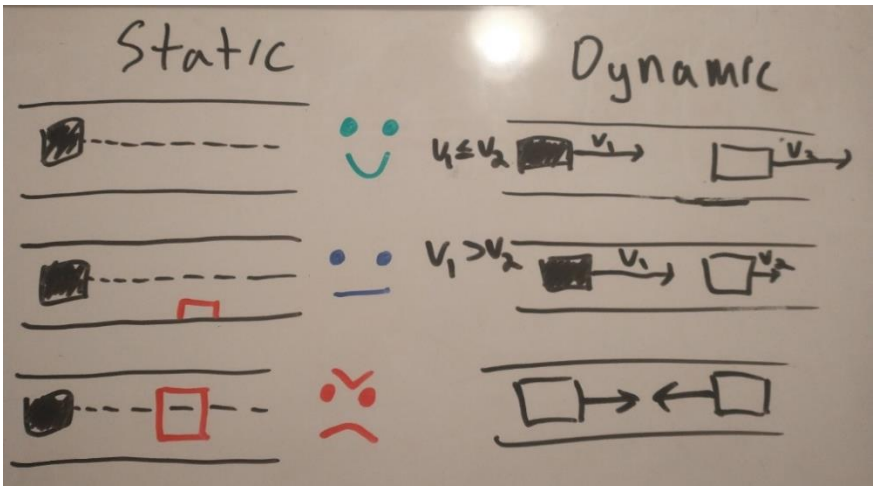
Path Planner

The Path Planner deals with “lane-level” candidate paths. What this means: For each adjacent and succeeding lane within a horizon, a space-curve is generated that would represent the cars path to that lane. What this looks like in different scenarios (note: color doesn’t mean anything in these diagrams):



Each of these paths is assigned a cost for safety and comfort. Safety costs should consider the following at minimum:

- **Static obstacles:** If a lane/path is completely blocked by a static obstacle it may be discarded; if it is only partially blocked (i.e. a trash bin by the side of the road), its cost can be increased, and avoidance can be left to the motion planner.
- **Dynamic obstacles:** Moving obstacles should increase the cost of paths. Avoidance can be left up to the motion planner, but obstacles that may conflict with the cars path should increase the cost.
- **Lane changes, left turns:** More complex operations such as a lane change or a left turn are riskier and should therefore have a cost.



Dynamic obstacles will likely be postponed for Demo 2

Paths might cross multiple lanelets depending on the horizon. In these cases, a path may contain only one lane change: it is bad driving practice to change multiple lanes at once and not something we need to consider for the vehicle.

Routing: The routing costs are folded in with the safety and comfort costs in this node. The route cost of a path is the same as the route cost of the last lanelet in the path. I don't know if routing cost should be kept separate from or combined into comfort costs: see "Comparing Costs" at the bottom of the document.

Inputs:

- **Static map data:** lanelet binary, as already implemented
- **Current pose:** Localized position of the vehicle on the map.
- **Current velocity (?):** Current velocity may be required pre-motion planner to evaluate dynamic obstacles
- **Routing Costs:** from the Route planner
- **Detected obstacles** and other perception data

Outputs:

<**Path**, **Safety Cost, Routing Cost**>[] where:

- a **Path** is an array of **<X, Y, ~~Theta~~>** poses. This is similar to Will's proposal, but instead of selecting the lowest cost here that is done in the behavior planner.

Behavior Planner

The behavior planners' job is to select the path candidate that the car will follow. It should not need to access obstacle information as much: this should already be embedded in the cost information for each path. It is also the behavior planners' job to interpret the routing costs and reconcile them with the safety and comfort cost.

Why we feed the paths to the Behavior Planner instead of selecting the lowest cost at the path planner:

- Path stability: Selecting the lowest cost path could cause rapid switching between different paths of similar but jittery costs. Feeding the paths through the state machine eliminates this behavior.
    - The rapid switching would likely take the form of decision paralysis rather than changing lanes back and forth, which would be prevented by the lane change cost. However, the costs may switch many times before the car can execute either path: the car *wants* to make a lane change, and then doesn't, and then does, many times a second. It would not have time to act on either path, which is the issue.
- Behavior continuity: certain actions, like signaling for and then taking a turn, require time to execute and should not be aborted due to fluctuations in path costs.
- Flexibility: The finite state machine is more flexible than the path planner and can consider wider circumstances easily. For instance, suppose in the future we run into a problem where the car occasionally lights on fire. We would need to pull over, but it's hard to tell the path planner to change costs due to the car being on fire. The adaptability of the FSM lets us make these more nuanced decisions.

(**Feedback requested**): Once the planner has selected the path to follow, it has one more job to do. The behavior planner but still needs to handle things like yielding and stopping. It will do so by assigning maximum/ideal velocities to the path. These velocities are nonbinding and need not follow the physical constraints of the vehicle: instead, they indicate whether the vehicle needs to stop or slow down by a certain point.

Alternately: We saw during the Waymo presentation that they used virtual obstacles to help control vehicle behavior, like putting one up in the intersection when a car was about to run a red light. We could implement a similar system here.

Inputs

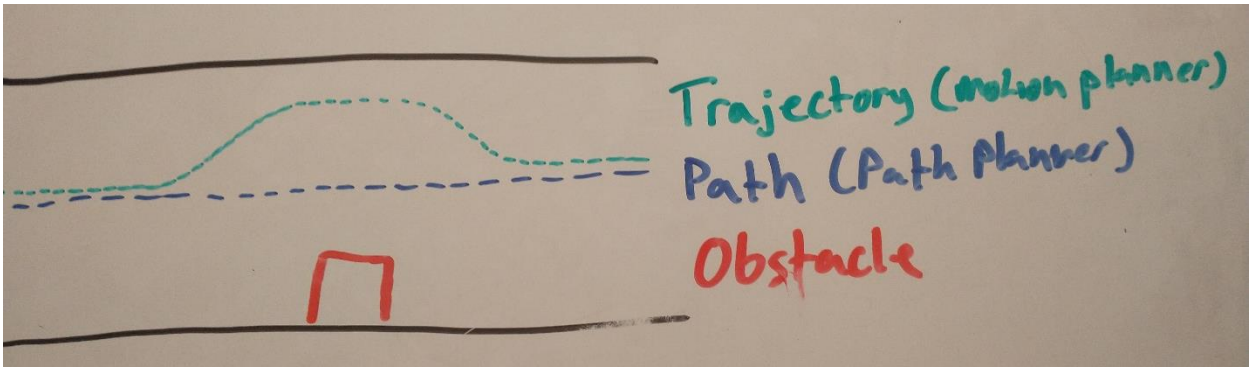
- **Path candidates:** The <Path, Costs> array from the path planner
- **Routing costs:** The routing costs for adjacent lanelets. These can then be associated with the paths that end in them.
- **Safety manager status**: For emergency stops
- Other miscellaneous (i.e. stoplight status)

Outputs

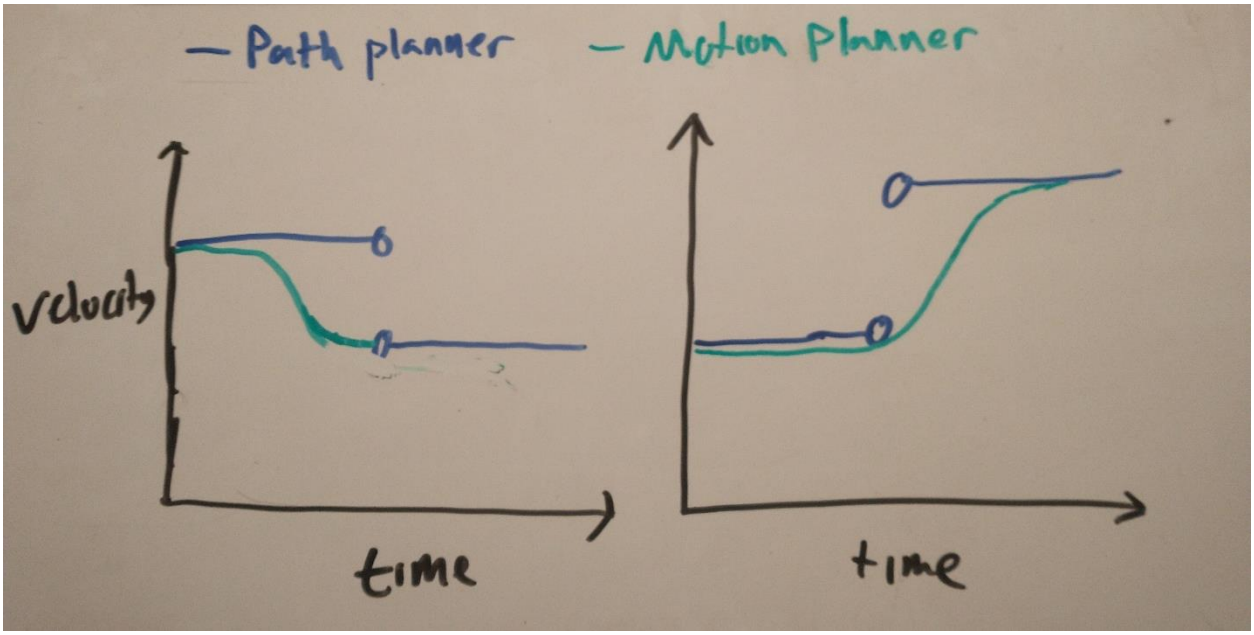
- **Path with velocity:** Final selected path with the associated ideal velocity. Array of triplets (X, Y, Speed)

Motion Planner

A node like our current path planner. Determine a vehicle trajectory that follows the path, avoids obstacles, and meets vehicle physical constraints. Must account for both dynamic and static obstacles and has some discretion to leave the path (which should be reported to safety). For instance, in a narrow lane where a trash bin is by the side of the road:



When it comes to velocities, it is more important to not exceed the path velocity than it is to match it. When conflicts arise, the slower speed wins:



* Note: I labeled this path planner, but they really come from the behavior planner.

The current implementation of many splines can already accommodate these requirements.

Inputs

- Path with velocities from the behavior planner
- Unchanged: Obstacles and perception data, car pose and velocity.

Outputs

TrajectoryPoint[] making up the vehicle's trajectory.

- Each element has a Point and a Twist

Controller

The requirements for the controller are unchanged: Translate the trajectory into efforts and execute.

Obstacle Avoidance Philosophy

The burden of avoiding obstacles lies primarily with the motion planner, but the Path Planner and Behavior Planner have their parts as well. The goal of the Path/Behavior Planner is to select the safest option at an intermediate level of abstraction. This level makes determinations such as “There is a car accident in front so I will take a right” or “I am in a construction zone and should slow down.” It is their role to select the safest option at this level of abstraction.

Not all obstacles can be avoided at this level of abstraction, though. The safest option from the Path/Behavior planner’s perspective may still require driving other than following the lanes centerline at optimal velocity. This is where the Motion Planner comes in. Examples include shying to one edge of a lane to avoid a wide vehicle or encroaching obstacle and slowing down when a car is likely to pull out into our lane.

(Summary) Between a higher-level abstract view from the Path/Behavior planner and a low-level view from the Motion Planner, obstacles should be safely avoided. The Behavior Planner will guide the car into zones where obstacles are fewer or easily avoidable and the Motion Planner will do the rest.

Comparing Costs

The tradeoff between comparing different costs can be arbitrary, but before a head-to-head comparison can be done the costs need to represent some standardized unit. No determination of these units has been made anywhere in this paper: all costs mentioned so far have only been required to be relative to each other.

Why this doesn’t work: Suppose path A has a safety cost of 10 and a routing cost of 3, while path B has a safety cost of 15 and a routing cost of 1. We may determine that the difference of 5 in safety cost represents a significant cost or outweighs the routing cost difference of 2, and select route A. However, since no standard values have been assigned to either safety or routing, only the relative costs are assured: this means that the costs A = (1, 300) and B = (2, 1) with equal validity, just in a different system of units. In this case, briefly it looks like route B is less costly.

Without a standard unit system cross-category cost comparison cannot be made. Cost units must be agreed upon through the entire system but don’t need to be the same for each category. An example of what these units could be for routing costs:

- Meters (when routes are measured in distance)
- Seconds (when routes are measured in time)
- Number of left turns or lane changes
- Some nameless but consistently defined metric
- Some compound metrics made of the previous