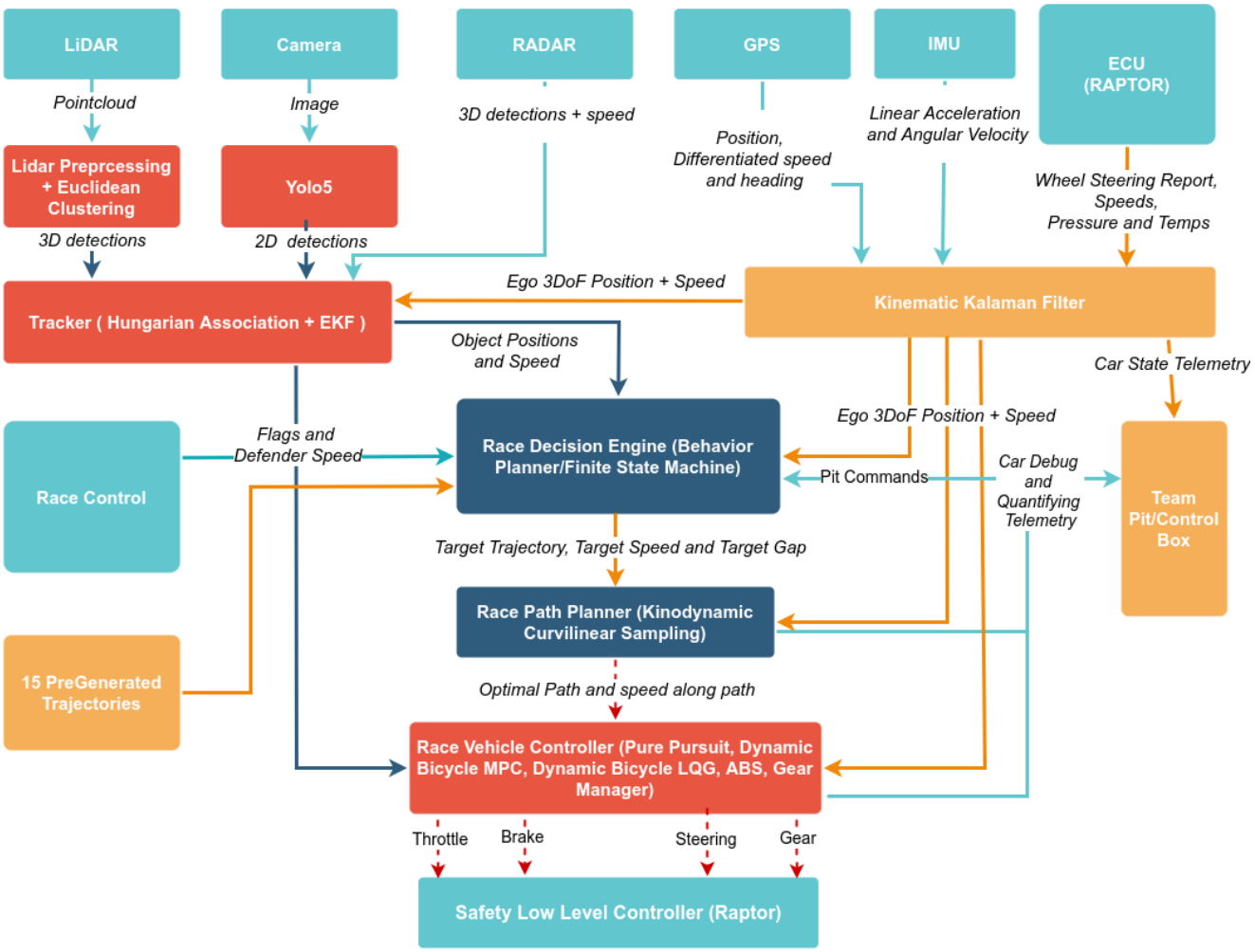
Race Architecture



Perception

Explained in Perception Architecture of our stack

Localization

Currently there is a kinematic kalman filter that implements a constant acceleration point mass model. We use the GPS ENU position and speed from our GNSS systems and linear acceleration and angular velocity as input measurements. The state vector is `[x, y, yaw]` where each variable is update as follows

$$x = x + (v * cos(yaw) * t) + (a_x * \frac{t^2}{2})$$
$$y = y + (v * sin(yaw) * t) + (a_y * \frac{t^2}{2})$$
$$yaw = yaw + (yaw\_rate * t)$$

The speed was sent out unfiltered

Behavior planning (Race Decision Engine)

The Race Decision Engine (RDE) takes the the pit station commands, race control, sensory input (position, joystick and actuator faults) and rival position and formulates what is the speed limit and target line the stack should follow. It has a choice of up to 15 lines to choose from (generally not all are generated) for racing. In practice, due to the current race format we only generate an inner line and outer line for racing. There is additionally a pitting line which we use to autonomously pit in. The triggers are as follows

- Red flag: Controlled Stop
- Yellow Flag: Drive at yellow speed (set by pit station)
- Green Flag: Drive at green speed on defensive line (set by pit station)
- Waiving Green Flag: Transition to attacker line (set by pit station)
- Black Flag: Transition to pit line when close to the pit entry
- Purple Flag: Emergency Stop

There are also watchdogs built into this module that keeps track of sensory inputs and if there is anything going wrong it commands a controlled stop (in case of joystick failures in autonomous mode, loss of race control and position errors) or emergency stop (in case of joystick failures in manual mode or actuator faults)

Path Planning (Race Path Planner)

The Race Path Planner (RPP) takes a target speed and target path from the decision engine and generates it's own path. This is currently done using kinodynamic curvilinear planning. This means that the path planner generates paths with respect to a time horizon and that the length of the path may vary based off speed. It generates lateral and longtitudinal paths using a quintic polynomial planner that takes our current state and our desired state and generates a path b/w them. These lateral and longtitudinal paths are generated in the Frenet frame which makes it possible to independently generate them and combine later. The combined paths are then ranked by their costs (based off jerk, progress on the path, and speed). Paths that collide with obstacles/bypass feasibility constraints (too high acceleration, above speed limit, too much curvature etc) are filtered out. The selected path is converted back to Cartesian Frame and outputted

Path Tracking (Race Vehicle Controller)

The Race Vehicle Controller is responsible for taking in the final path and tracking it as best as possible based off the vehicle dynamics. It generates throttle, steering, brake and gear shift commands for DBW systems. It can also publish in just speed, brake and steering for low level controllers based off speed controls. It current implements a PID for speed (from RDE/RPP) to throttle conversion and PID, Pure Pursuit and Dynamic MPC with linear tire model (taken from Autoware) as lateral controllers

Offline Trajectory Generation

We have simple pipelines to take reference lines/track boundaries and offset them to generate inner and outer lines. We also have a minimalistic optimal line generator that uses lookup tables to formulate vehicle dynamics and determine the path that takes the minimum time based off a greedy algorithm