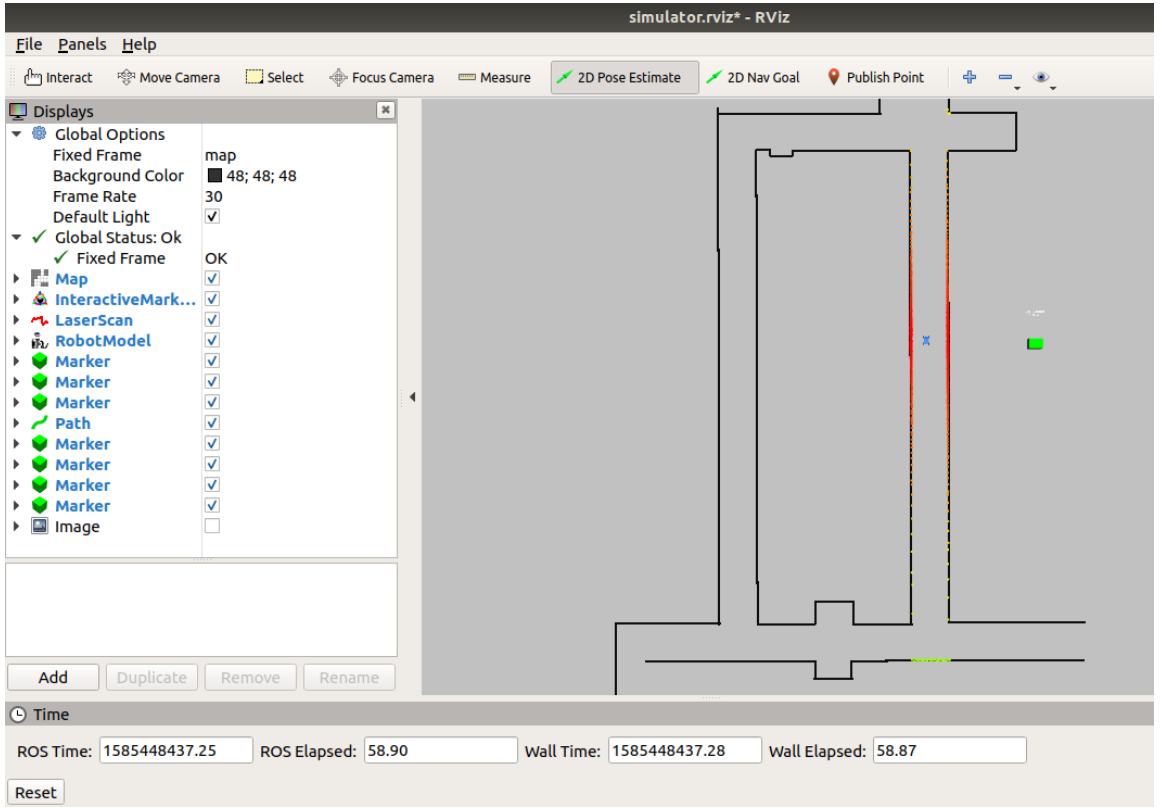
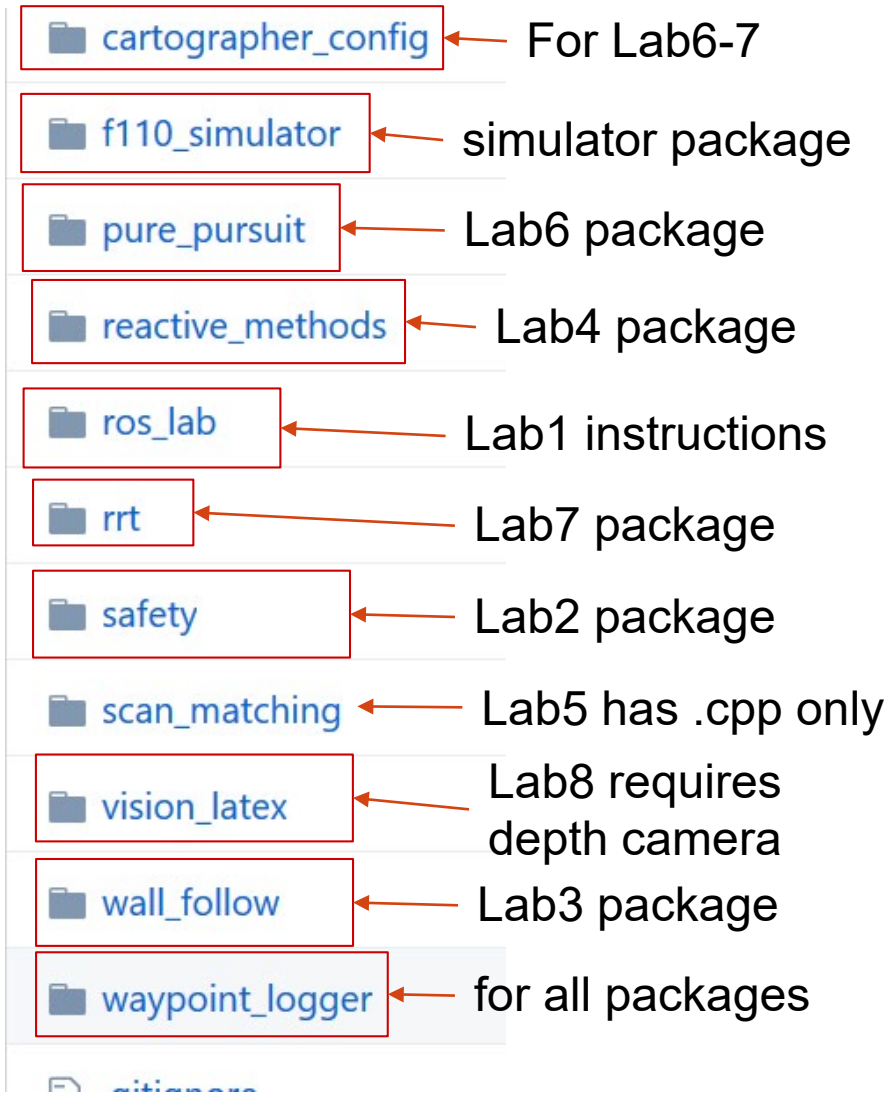
Module B: Reactive Methods



We will cover:

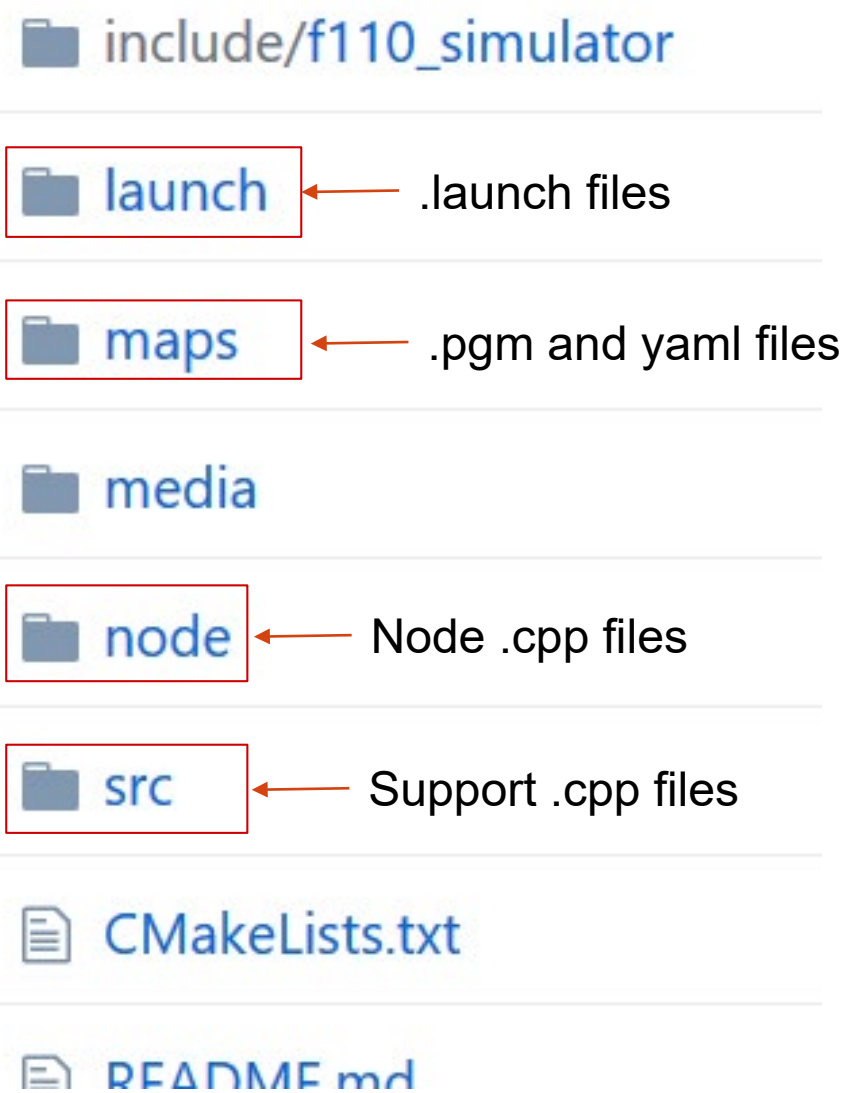
1. Understanding and Working with the f110 simulator
2. Race 1 Rules and Strategies

Labs at https://github.com/f1tenth/f110_ros



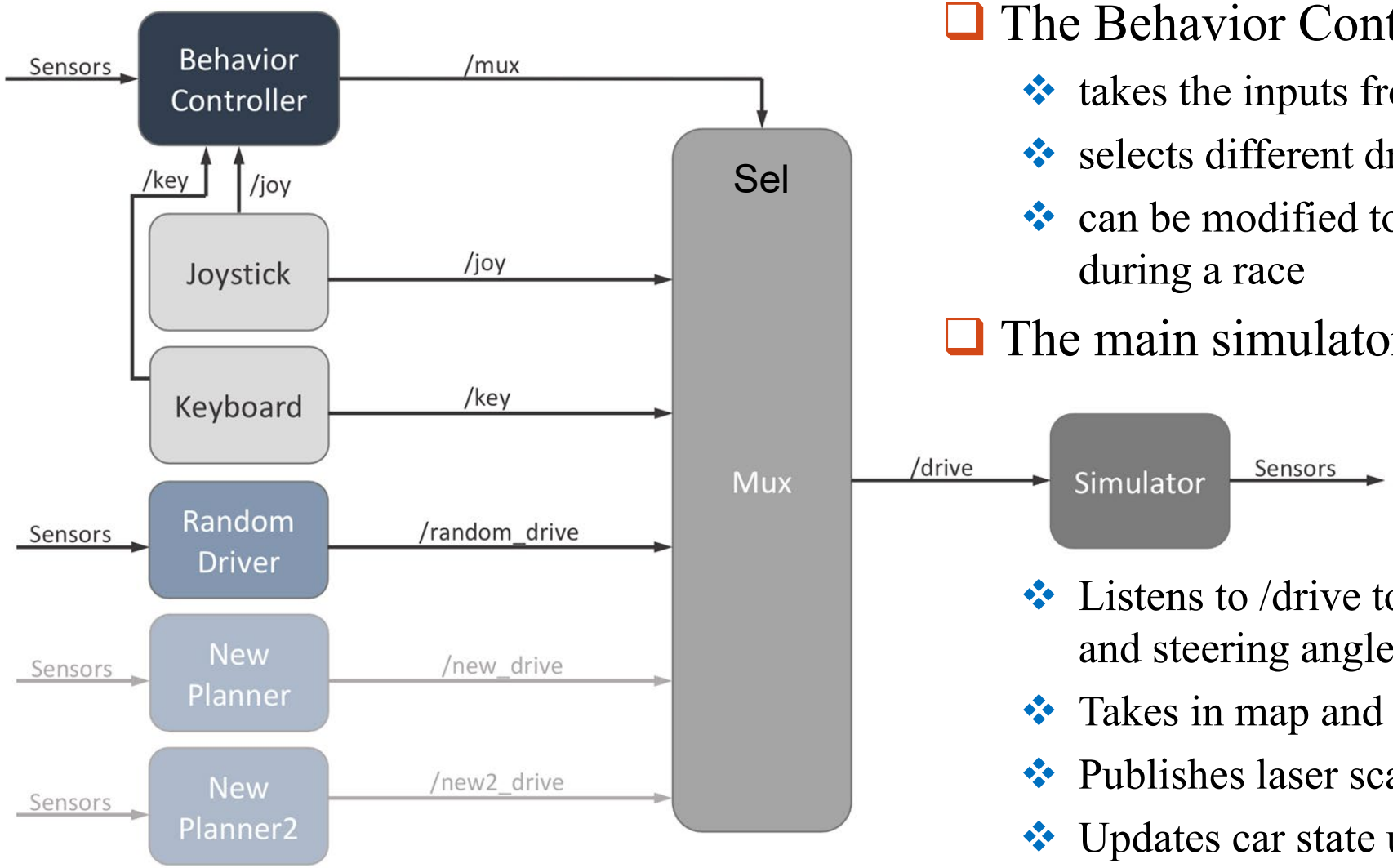
- The git folder contains the main simulator and several lab packages;
- Follow the Readme.md inside the f110_simulator folder to install dependencies and catkin_make;
- Except ros_lab, each lab is a separate package, working along with the f110_simulator package.
  - All required package.xml and CMakeLists.txt are provided;
  - Nodes for each package are written in the scripts or src folder in each lab folder, usually one node/file
  - .py template files are provided for all labs except scan_matching; Avoid having multiple files of the same node name in one package.
  - To run a new package, $rosrun packagename nodename.py

The f110_simulator package



- The main simulator has many nodes, all written by .cpp
  - Original simulator was MIT's racecar simulator, so many files have been taken from that git.
  - Main developers in Upenn are grad students (Matthew O'Kelly, Joseph Auckley, and Billy Zheng, etc.)
- The package consists of several folders + parameter files.
  - /node folder contains the major nodes, such as mux, random_walker, keyboard, behavior_controller, etc.
  - /src folder contains support files, such as distance transform, kinematics, (lidar) scan simulator, etc.

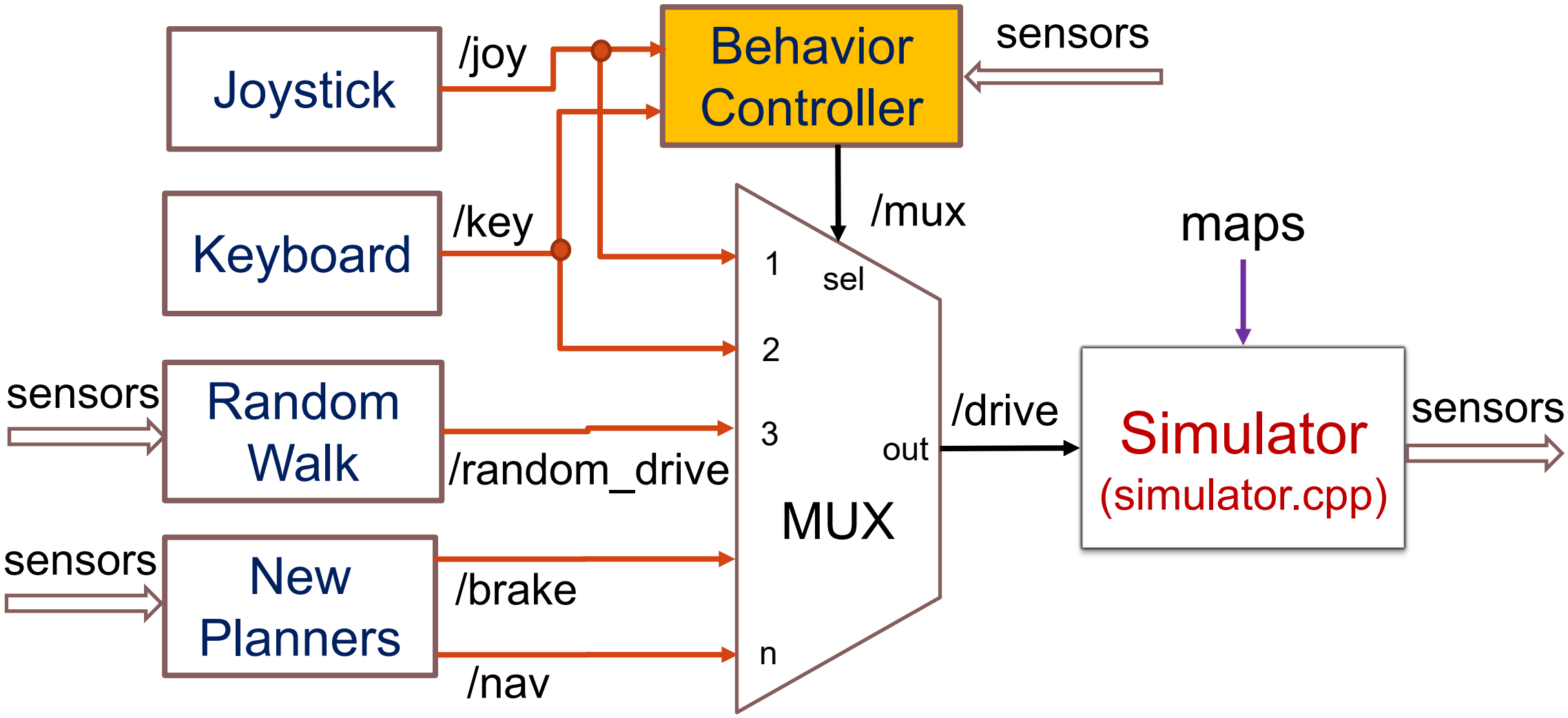
The F110 Simulator Package



- The Behavior Controller:
  - takes the inputs from joystick or keyboard
  - selects different drive mode via MUX
  - can be modified to switch between planners during a race
- The main simulator node:
  - Listens to /drive topic for commanded velocity and steering angle
  - Takes in map and compute sensor data
  - Publishes laser scan and odometry messages
  - Updates car state using bicycle model dynamics

Figure modified from UPenn course ESE680

The F110 Simulator Package



Joystick Controller Node

Check the button indices by echoing the /joy topic. They vary by different types of joysticks

The Keyboard Controller

The main simulator terminal

```
k[ INFO] [1585448500.569347616]: Keyboard turned on
MUX:
0
1
0
0
0

was[ INFO] [1585448510.309024646]: Collision detected
MUX:
0
0
0
0
0

[ INFO] [1585448539.091073808]: Setting pose: 4.980 0.534 -2.789 [frame=map]
[ INFO] [1585448547.796949713]: Setting pose: 4.880 0.485 -1.819 [frame=map]
b[ INFO] [1585448564.562559312]: Emergency brake turned on
k[ INFO] [1585448570.176290089]: Keyboard turned on
MUX:
0
1
0
0
0

s[ INFO] [1585448601.719845089]: Collision detected
```

- Press keys in the main simulator terminal which displays the status.
- Manually toggle driver modes
  - B – AEB enable/disable
  - J – joystick on/off
  - K – keyboard drive on/off
  - R – random walk on/off
  - N – Navigation on/off

  K or J can be combined with other modes
- Keyboard drive:
  - W – forward
  - S – backward
  - A – left turn
  - D – right turn

Adding a New Planner inside Simulator

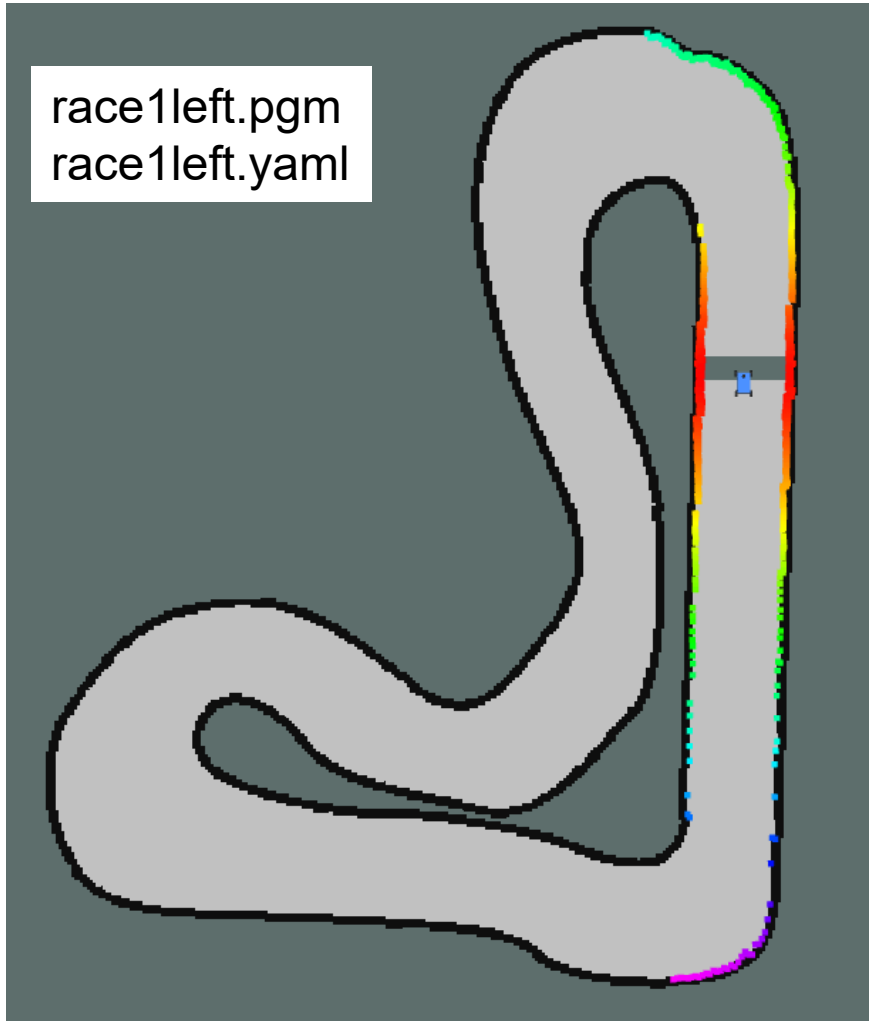
- Method 1: use the /nav topic, add the node in simulator.launch.
- Method 2: Add a new channel in Mux

params.yaml

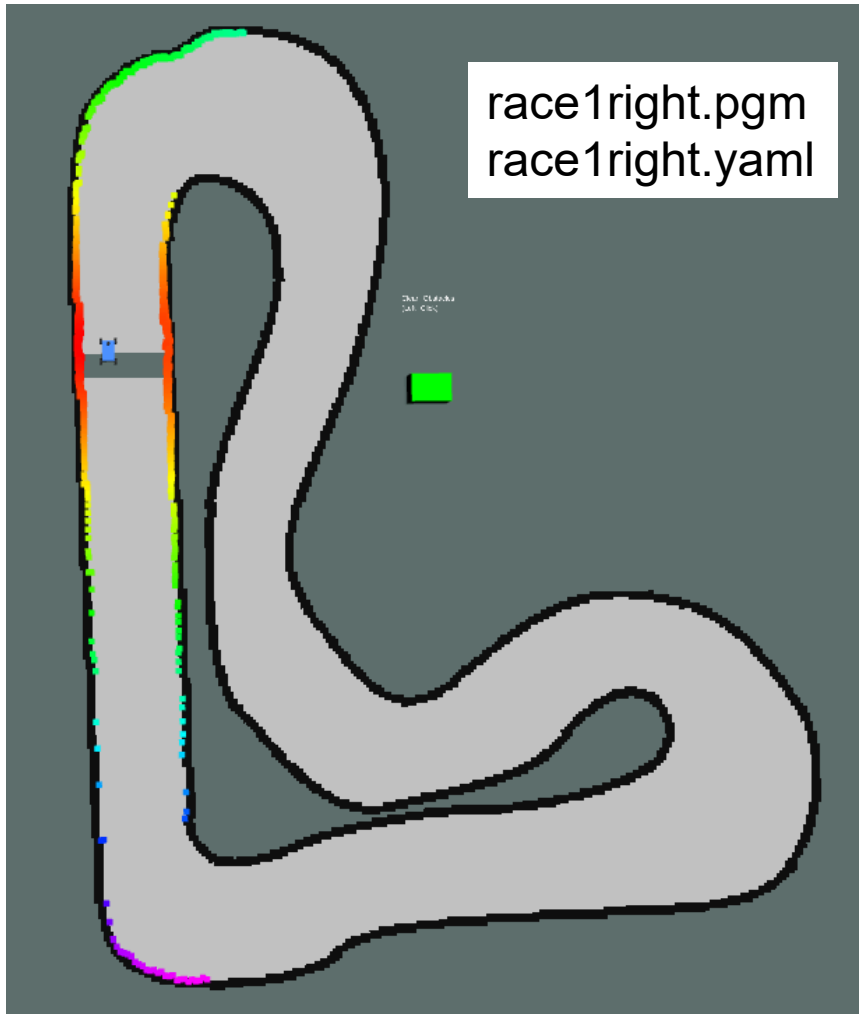
```
72  # Keyboard characters for toggling mux
73  joy_key_char: "j"
74  keyboard_key_char: "k"
75  brake_key_char: "b"
76  random_walk_key_char: "r"
77  nav_key_char: "n"
78  # **Add button for new planning method here**
79  new_key_char: "z"
```

- Steps for Method 2 add a new channel in Mux (not required for Race 1)
  - In `params.yaml`, add the new drive topic name, new mux index, new single keyboard letter, new joystick index
  - Add launch command in simulator.launch
  - Add a new Channel instance at the end of Mux() constructor in mux.cpp
  - Add to joystick and keyboard callbacks in behavior_controller.cpp a few member variables: see template on
    - Line287 – 320 inside joy_callback()
    - Line 323-354 inside key_callback()

Race 1 Setup: Replace race maps



Map for Left Follow



Map for Right Follow

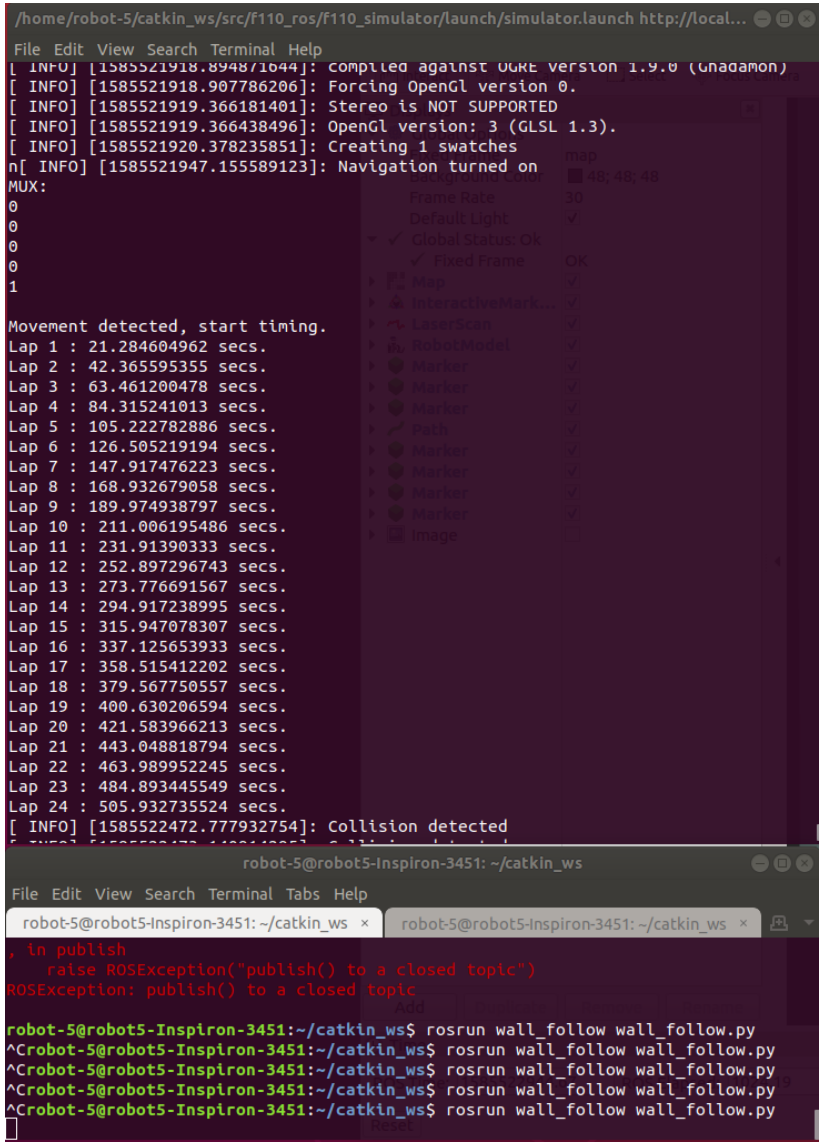
- Two maps are mirror images of each other.
- Modified by TA from an existing porto map used in one real race
- To replace the map in simulator.launch on line 7:
  - Left wall following: change Levine.yaml to race1left.yaml
  - Right wall following, change Levine.yaml to race1right.yaml

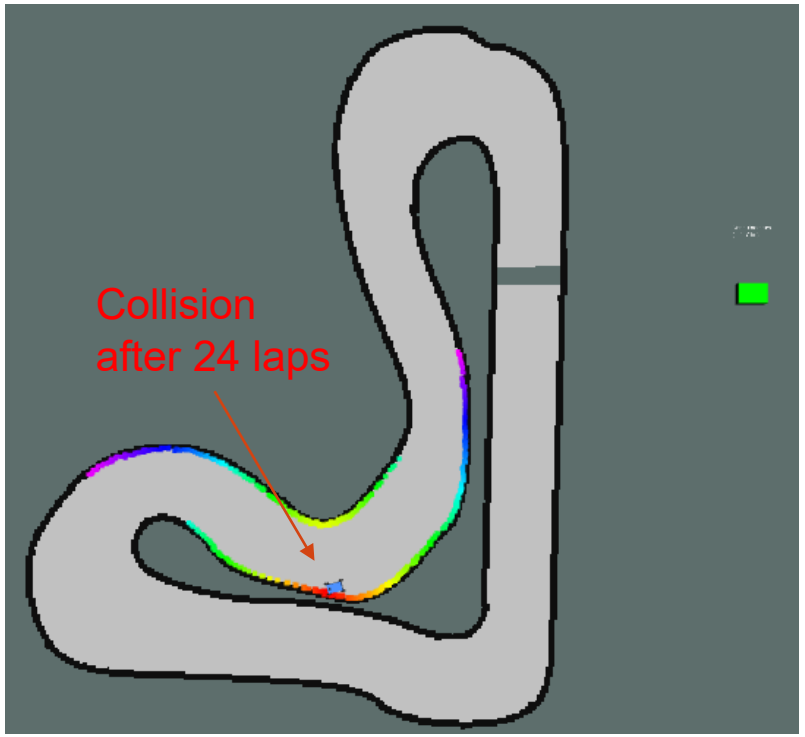
Race 1 Running on Three Terminals

1. Run f110_simulator in terminal1
2. Run wall follow node in terminal2
3. Select ‘n’ in terminal 1 to enable navigation

The demo node achieves 27 s/lap

Race 1 Running by simulator.launch





- Download timer.py and put it in wall_follow/scripts folder. Modify simulator.launch by adding 3 lines.
- Run wall_follow.py in another terminal when adjusting parameters. When you are done with the PID tuning, add the wall_follow node in the launch file too.

```
<!-- Launch the timer node -->
  <node pkg="wall_follow" name="Timer_node" type="timer.py" output="screen">
      </node>

<!-- Launch the wall_follow node -->
  <node pkg="wall_follow" name="nodename" type="filename.py" output="screen">
      </node>
```

Race 1 Grading and Strategies

- Race 1 Team Grade:
  - Simulator runs successfully with race track map → 20 points
  - Finish 3 laps without collision → 50
  - Finish additional laps in 120 seconds without collision → each additional lap adds 2.5 points
  - Shortest time to finish 3 laps: race winner, earn 10 points
  - Longest time to finish 3 laps: ranks #10 and earns 1 point.
  - Quality of report: 15 points
- Individual grade: adjusted by peer evaluation and class participation.

- Race File Submission:
  - Project lead: Entire simulator package under your workspace, zipped together
  - Subscriber: Short report 2-5 pgs on
    - How you tried to tune the parameters for the race and their corresponding results.
    - Experience gained through the process. Best results you obtained on your computer.
    - Show evidence such as screenshots or video.
  - Peer evaluation form with some details about individual effort in the comments.
- Race Strategies w/ wall_follow:
  - Choose desired distance, velocity, etc.
  - Fine tune PID parameters
  - Reduce computational complexity