UT AUTOmata

Reference Manual v1.0.1

Joydeep Biswas, Jarrett Holtz, Sadegh Rabiee
{joydeepb, jaholtz, srabiee}@cs.utexas.edu

Lab Computer First Time Setup

Update your .profile file

Add the following lines to the bottom of `~/.profile`:

```
source /opt/ros/melodic/setup.bash
export ROS_PACKAGE_PATH=$ROS_PACKAGE_PATH:~/f1tenth_course
export ROS_PACKAGE_PATH=$ROS_PACKAGE_PATH:~/projects/f1tenth_starter
```

After adding these lines you will need to either relog into the computer or run:

```
source ~/.profile
```

Clone and Build Course Code

From your home directory run:

```
git clone https://github.com/ut-amrl/f1tenth_course.git
cd f1tenth_course && make -j
```

Verify everything is working

In one terminal start roscore and leave running:

```
roscore
```

In another start the simulator:

```
cd ~/f1tenth_course && ./bin/simulator
```

Finally open the visualizer webpage in google-chrome (or your browser of choice):

```
google-chrome ~/f1tenth_course/web_rviz.html
```

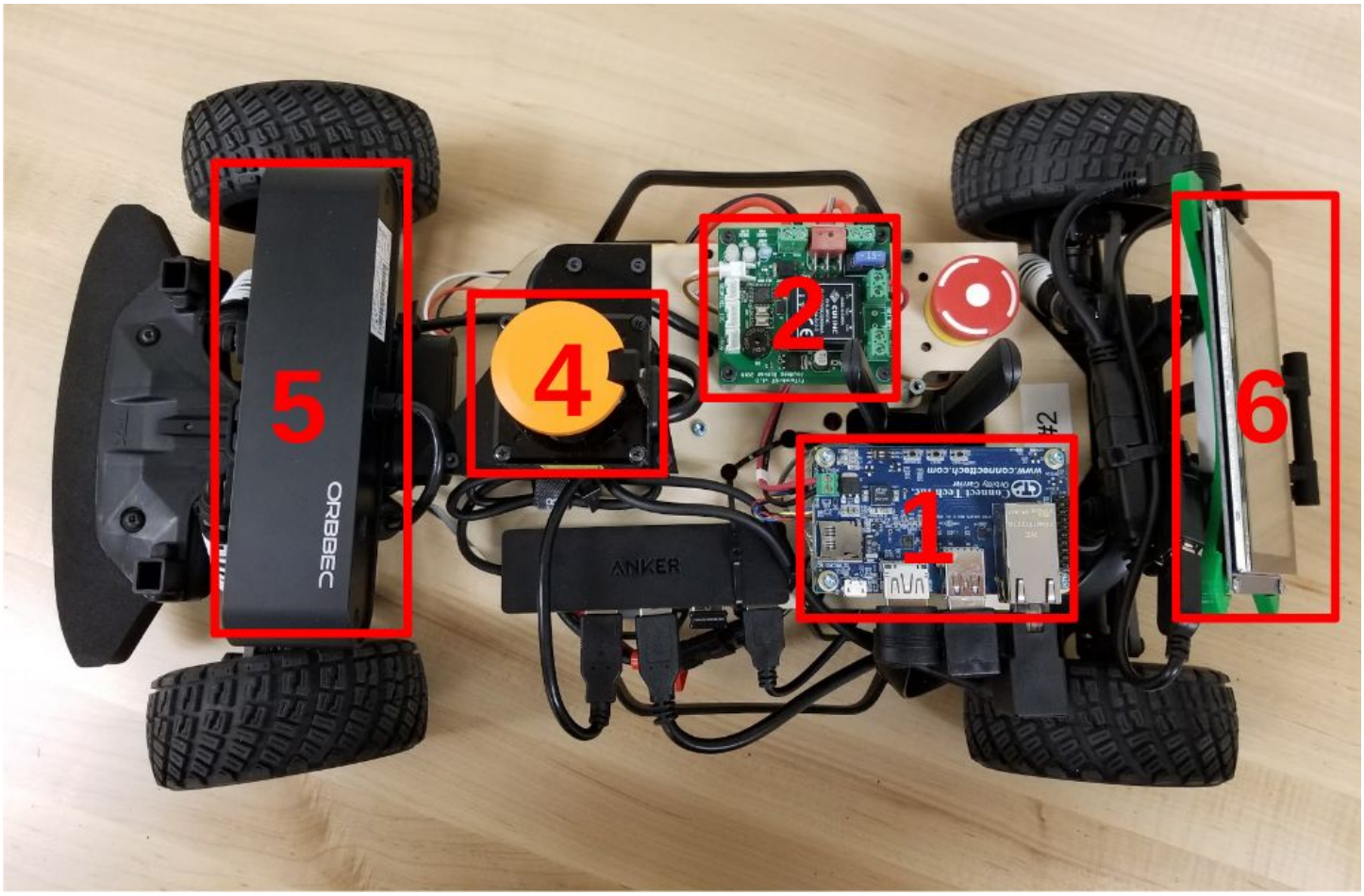
Updating the code

If the instructors make changes to the f1tenth_course repository you may need to update your code. To do this:

1. `cd f1tenth_course`
2. `git pull`
3. `make -j`

F1/10 Car Hardware

Specifications



The car has the following boards and sensors:

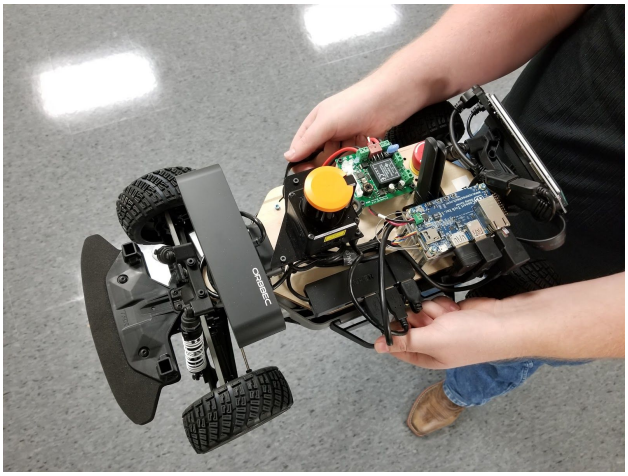
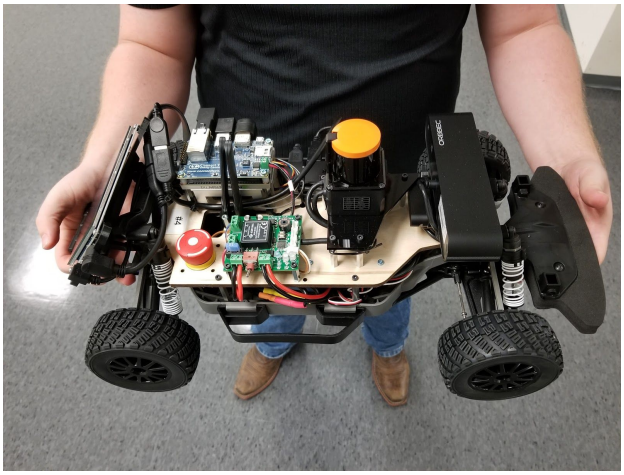
1. Jetson TX2: The mini computer that is the main brain of the car and you will run your programs on.
2. Power board: Receives the the battery voltage as input and provides regulated voltage output for all boards and sensors on the car. It also has a battery voltage level indicator LED that goes from green to red as the battery drains as well as a buzzer that beeps to remind you that the car is on. The board continuously monitors the battery voltage, and will trigger an audible alarm if the battery is drained.
3. Motor Driver (located under the car's midplate): It converts your velocity commands to actual electrical signals that drive the motors.
4. Hokuyo 10lx: A 2D lidar with 270 degrees field of view and 30m range.
5. Orbbec Astra Pro: A depth camera with 8 meters of range.
6. Touch screen: It allows you start the car via a graphical interface and monitor the status of the sensors and motors.

Handling the Robot

Safety

There exist exposed electronics on the car, hence it is important for you to be mindful of them when handling and using the robot to prevent any potential damage to the car's hardware. To keep you car functional and avoid hazards please abide by these safety rules:

- No food or drink while using the car.
- Do NOT put metallic objects in contact with boards.
- Do NOT jostle wires.
- Don't lift the car by PCBs or sensors.
- Hold the car by either the two side handles or the front and back bumpers as shown in the following figure.
- **Before turning on the car**, make sure it is in a **stable position and on the ground**. Do NOT turn on the car while it is on a table. This is to prevent accidental damage in case unintended drive commands are sent to the motor.

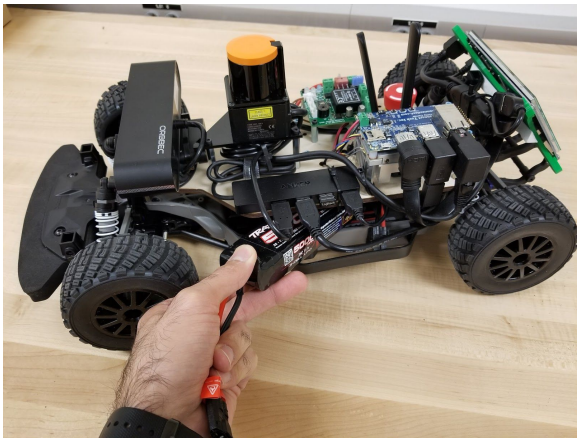
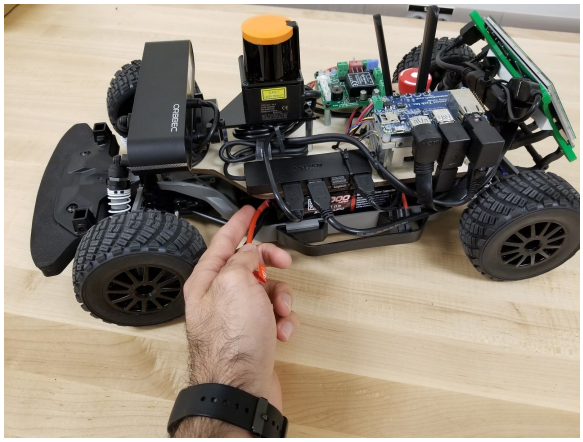



Each team is provided two LiPo batteries to power up their car. Although LiPo batteries have the benefit of a high power density which makes them ideal for mobile devices and robots, they could be very dangerous if used incorrectly. **Misuse or incorrect handling of LiPo batteries can cause fires**. Please make sure to follow these safety rules to prevent potential hazards:

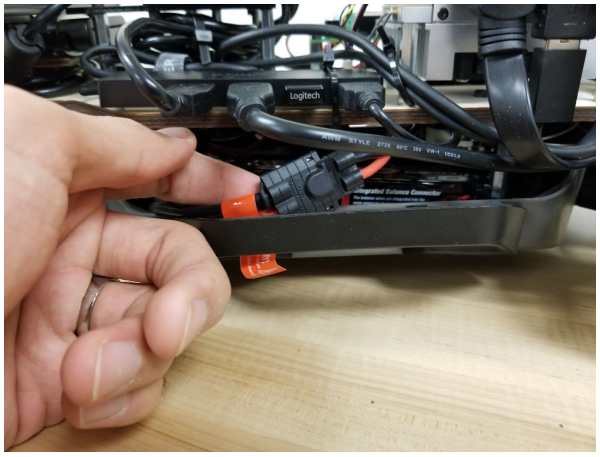
- Be careful not to short the battery leads.
- Never discharge your batteries under 10V. Over-discharging LiPo batteries causes permanent damage to the batteries.
- If the **battery alarm** goes off on the car, kill all the processes, turn it off immediately and charge the battery.
- Never charge your battery unattended. Regularly check if the battery is getting warm or starts to swell, if so stop charging immediately.
- Pick up a LiPo battery by its body, not the leads – the wires could be pulled off from the fragile solder joints.
- Do NOT leave the battery in the sun or next to a heater.
- Never use or charge a damaged battery. Don't charge if it is swollen (puffy) or has any other visible signs of damage. Ask the TA's to check your battery if you see such signs.

Turning On the Car

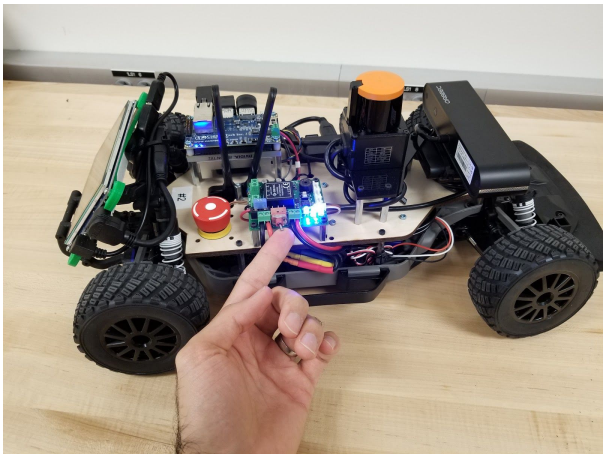
1. Slide in a charged battery in the orientation shown in the following figure.

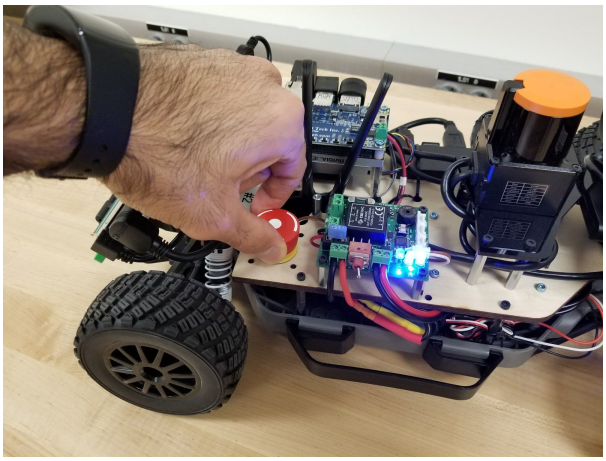
2. Connect the battery connector to the battery connector on the left side of the car.
   Make sure that you correctly align the red/black colored wires from the battery and connector
   The connecter won't engage in the incorrect polarity, **DO NOT FORCE IT**



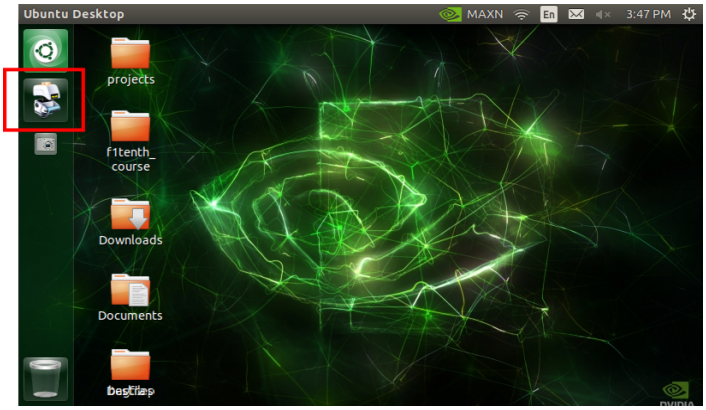
3. Flip the red switch on the power board to turn on the car.



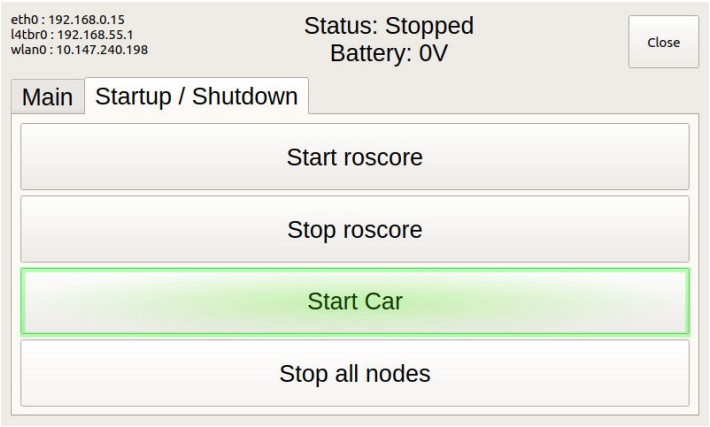
4. Make sure the emergency stop is not engaged by rotating it clockwise.



5. Once the car has booted up and you can see the Ubuntu desktop on its LCD, you can tap the startup icon to start the UT-AUTOmata GUI:

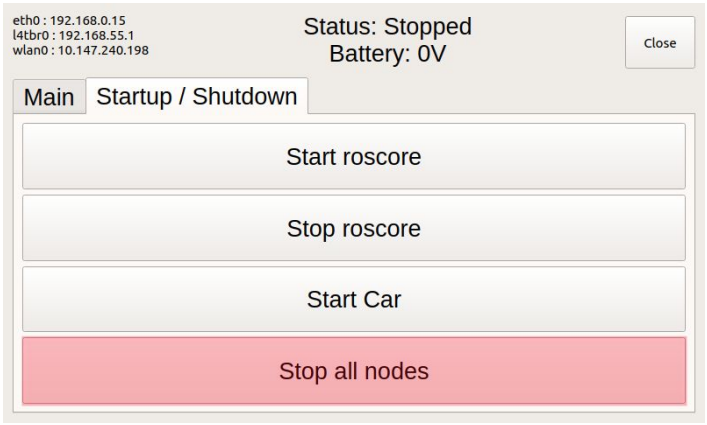


6. Once in the GUI you can then navigate to the Startup/Shutdown tab and start all required ROS nodes by pressing the button shown below:



Turning Off the Car

1. Kill all the processes using the GUI. Also kill any processes that you might be running on the car.



2. Shutdown the Jetson either using the Ubuntu desktop or by running `sudo poweroff` while you are ssh'd into the car.
3. Flip the powerboard switch to cut off power to the car.
4. Unplug the battery.

Charging the Batteries

You have a battery charger that can charge one battery at a time. Plug in the charger's power cord to a power outlet and connect the battery to the charger as shown in the following figure.

- The charger should automatically detect the battery type as LIPO (the green LED next to the BATTERY TYPE button should light up).

- Select “BALANCE” as the LiPo charge mode (out of the three different modes of BALANCE, FAST, and STORE) by pressing the “LIPO CHARGE” button.
- To **start charging**, **press and hold** the “START/STOP” button. The charger will make a beeping sound and start charging the battery. The charging status LED (the green LED at the top of the charge rate section) will blink to indicate that charging is in progress.
- If you want to **stop charging**, you can **press (NOT HOLD)** the “START/STOP” button and then disconnect the battery.
- Once the battery is charged, the charger will make a beeping sound and the charging status LED will be steady green. Press the “START/STOP” button and unplug the battery.

Steering Offset Calibration

If your car drifts to the left or right when being commanded to drive straight, you may need to calibrate the steering offset.

1. Using the joystick, drive the car forwards, and see if it curves to the left or right.
2. Adjust the steering offset calibration value `steering_angle_to_servo_offset` in the file `~/f1tenth_course/config/vesc.lua`
3. The value should be approximately 0.5, lower if the car is swerving to the left, higher if swerving to the right.
4. Kill the vesc driver node, and re-start it:
   ```
   killall vesc_driver && ./bin/vesc_driver &
   ```
5. Re-test the car (step 1)

Motor Speed Gain Calibration

If your car's odometry consistently reports that the car has traversed a greater (or smaller) distance than the actual distance traversed, you may need to calibrate the motor speed gain.

1. Kill and restart the motor driver node:
   `killall vesc_driver && ./bin/vesc_driver &`
2. In a different terminal, start monitoring the odometry using `rostopic echo /odom`. It should show odometry values such as:

```
---
header:
 seq: 443
 stamp:
   secs: 1582385508
   nsecs:  35240352
 frame_id: "odom"
child_frame_id: "base_footprint"
pose:
 pose:
   position:
     x: 0.0
     y: 0.0
     z: 0.0
   orientation:
     x: 0.0
     y: 0.0
     z: 0.0
     w: 1.0
 covariance: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.
0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0]
twist:
 twist:
   linear:
     x: 0.0
     y: 0.0
     z: 0.0
   angular:
     x: 0.0
     y: 0.0
     z: 0.0
 covariance: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.
0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0]
---
```

3. Note the starting position.
4. Using the joystick, drive the car forward along a measuring tape.
5. Note down the actual distance traversed, using the measuring tape: x
6. From the terminal, note down the odometry-reported distance traversed: y
7. Compute the ratio of the two: r = x/y
8. Repeat this procedure a number of times, and verify that the ratio r is consistent.

9. If r is close to 1 (e.g. 1.02 or 0.98), then your car's speed calibration is fine. If not (e.g. r=0.9 or 1.1), continue the calibration procedure:
10. Adjust the speed gain calibration value `speed_to_erpm_gain` in the file `~/f1tenth_course/config/vesc.lua`
11. The value should be approximately 5356. Let the value in the config file be z.
12. Replace the calibration value with r * z
13. Kill the vesc driver node, and re-start it:
    ```
    killall vesc_driver && ./bin/vesc_driver &
    ```
14. Re-test the car (step 1)

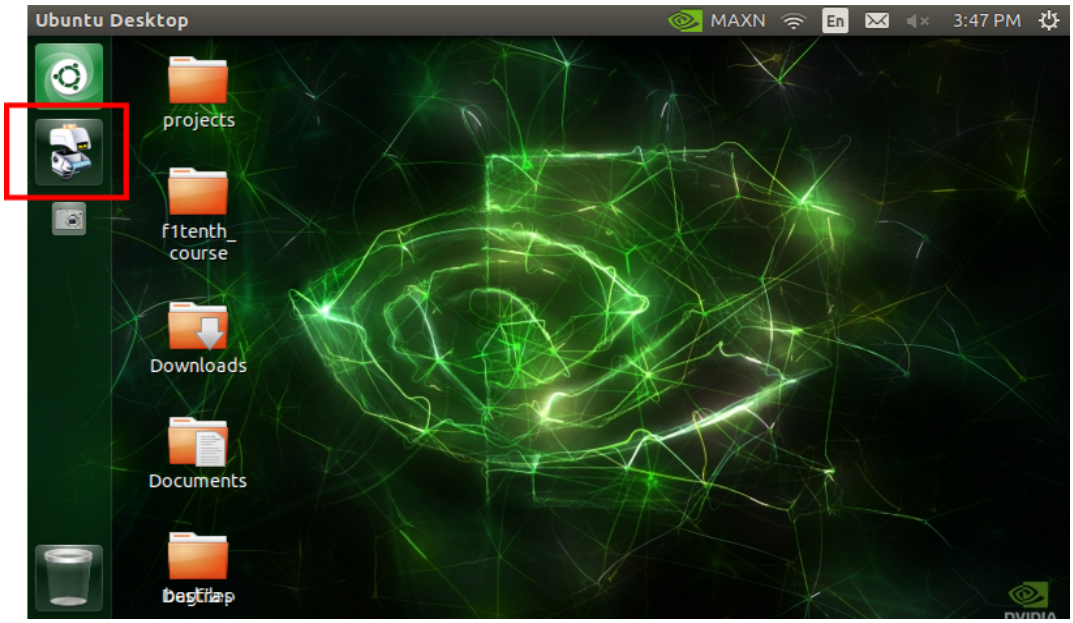
Using the Cars

Hardware Startup

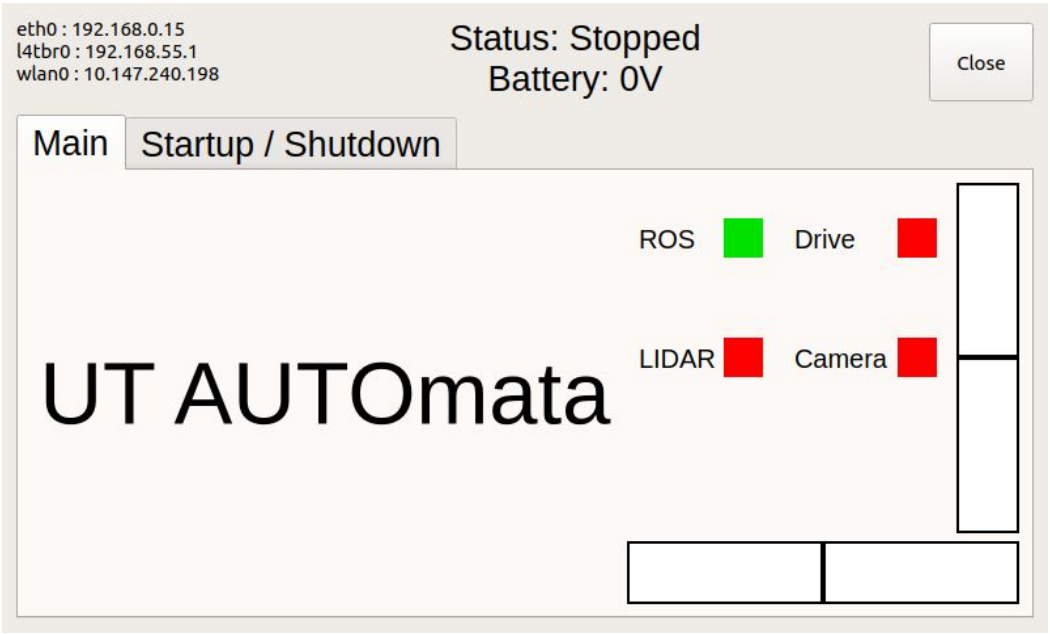
Refer to the section on F1/10th Hardware.

Software startup

After the car is powered on the initial screen you will see is the desktop below. To startup all necessary components for running the car you begin by pressing the icon shown in the red box in the image below.

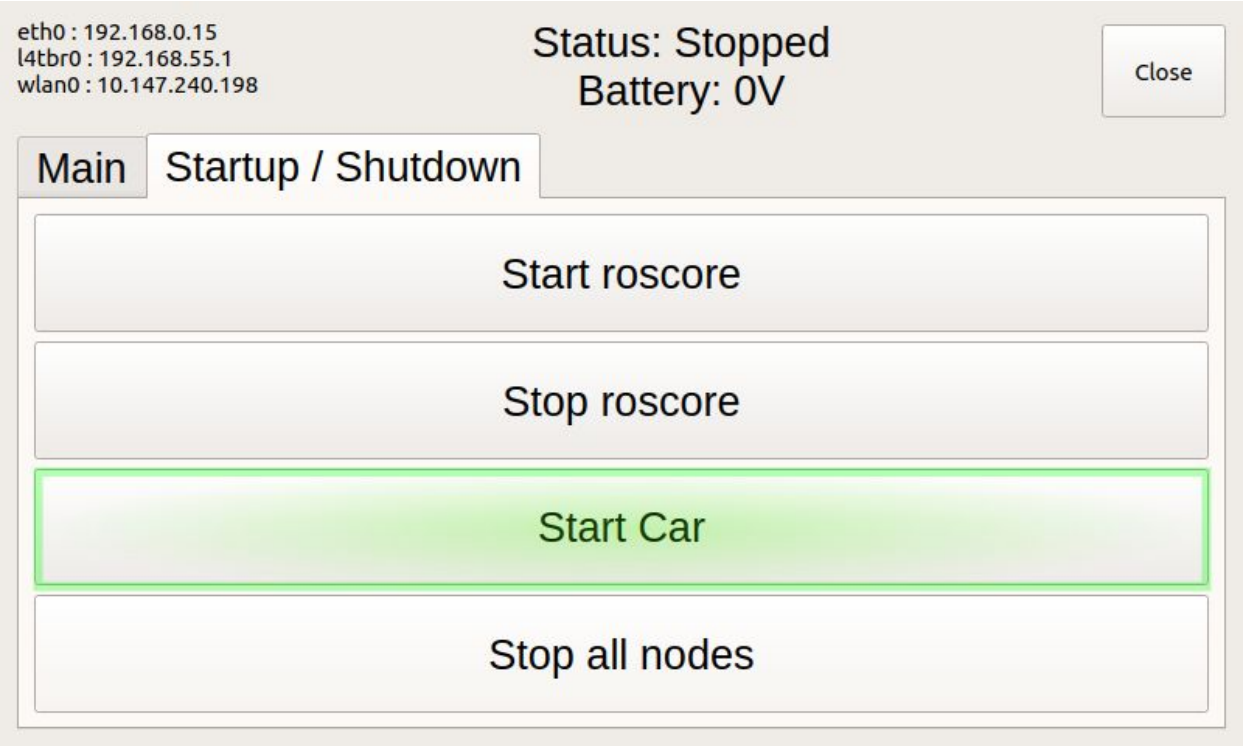


This will open the UT AUTOmata GUI to the main screen. This screen displays useful information, such as the IP addresses, robot control status, battery voltage, the state of the four core nodes (green for running and red for off), as well as the steering angle and velocity.

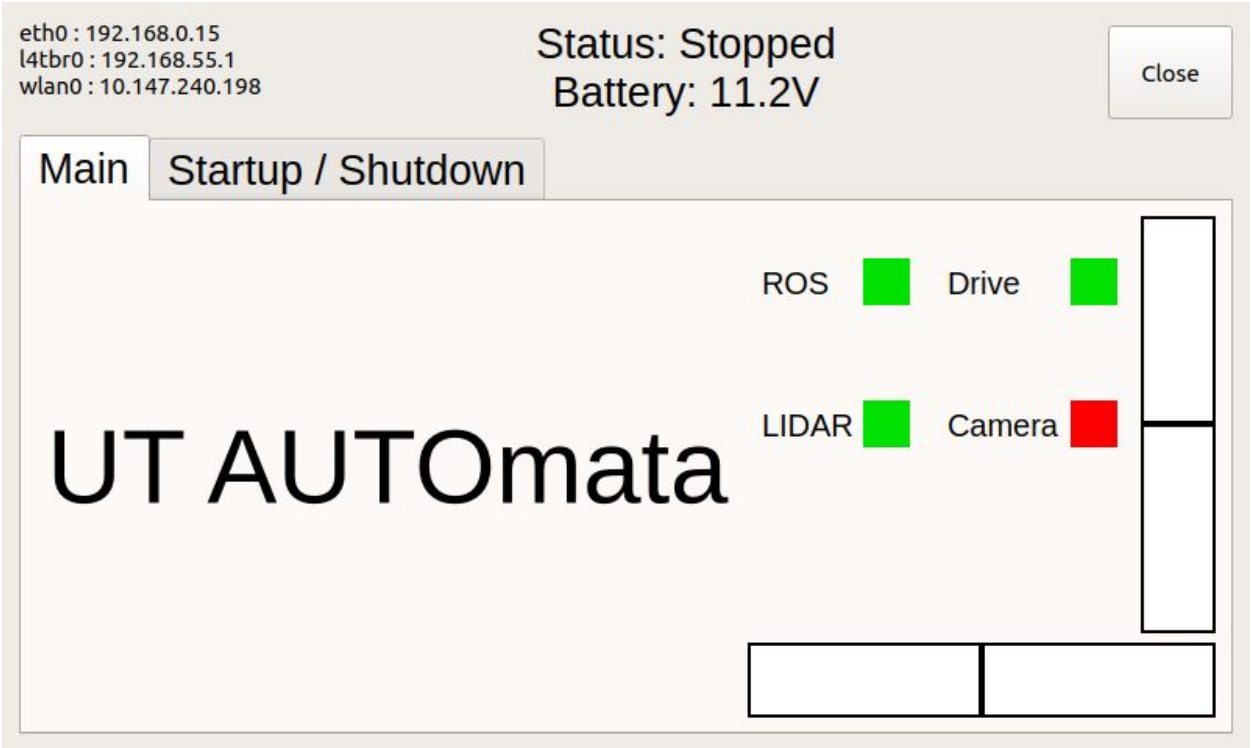


To start the nodes used by the car you will first navigate to the Startup / Shutdown tab of the GUI. In this tab you can stop and restart roscore, as well as start and stop all of the nodes for the car.

To start the required nodes, select the button highlighted in green in the picture below. Then navigate back to the main tab of the GUI.



The minimal necessary nodes for running the cars should now be started. You can check this by looking to see that the boxes next to ROS, Drive, and LIDAR are green. The camera node will not be necessary for this course.

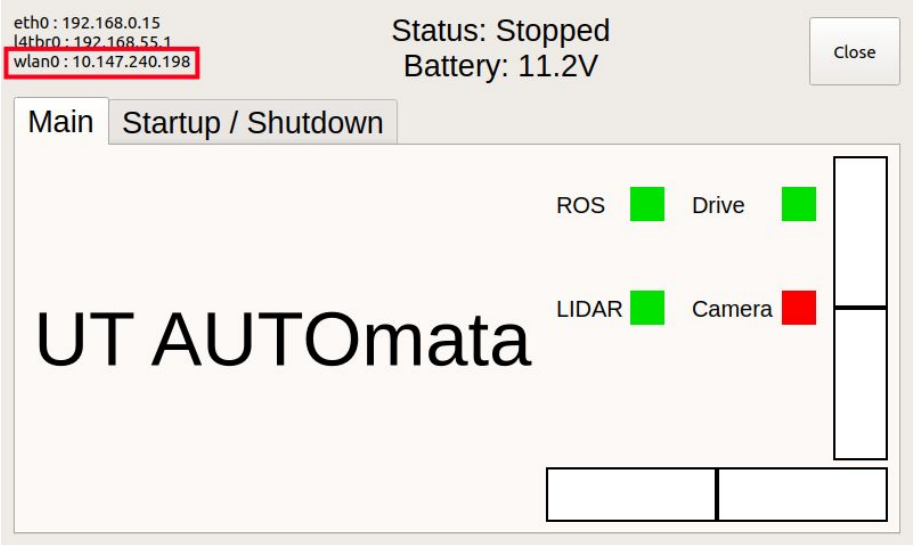


At this point the core ros nodes should be started, and you are ready to run your own code on the car.

Running your code on the cars

To run your code on the car you will be using ssh to connect to the car from the lab computers. Be sure that you have followed the Software Startup steps first.

1. Get the IP Address from the car GUI



2. Start a terminal on your machine
3. ssh to the car with the following command: `ssh amrl_user@<ip>`
    a. Password: **SafetyBeforeSpeed!**
4. Clone your own git repository containing your code inside the projects/ folder.

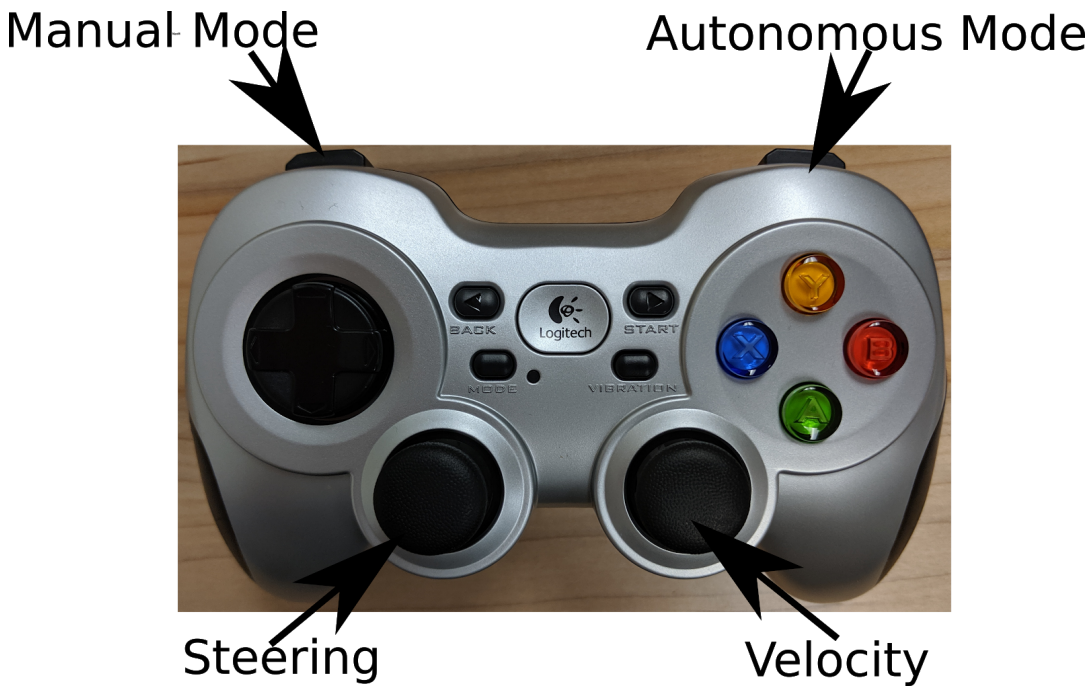
```
cd projects/
git clone <your git repository>
```

5. Add your repo to the package path:
    a. `ROS_PACKAGE_PATH=$ROS_PACKAGE_PATH:~/projects/<codeDirectory>`
    b. You may want to add this line to the end of the ~/.bashrc on the car.
6. Build your code: `cd <pathToCode> && make -j`
7. Run your code: `./bin/<executable>`
8. Refer to the section on Using the Joystick below to get the car to move.

Using the Joystick

The joystick is used with the robots in two modes. Autonomous and manual modes. The joystick is started as part of the normal startup process for the robot detailed above.

For the joystick to work the button labeled mode needs to be off (NO green light).



Manual Mode

In manual mode you can drive the robot with the controller. To activate manual mode you need to continually hold down the **LEFT** bumper on the controller. Steering is then controlled with the left analog stick, and velocity with the right analog stick. The car will stop executing joystick commands when you release the left bumper.

Autonomous Mode

Autonomous mode uses a dead man's switch for safety reasons. This means that for your car to drive autonomously based on commands published as messages you must continually hold down the **RIGHT** bumper on the controller. The car will stop executing messages when you release the right bumper.

Remote Visualizations

As part of the course repository we have provided an interactive web visualization tool that receives a specific message type sent from your code. You can use this message, as well as the tools in f1tenth_starter/src/visualization, to assist you with visualization and debugging throughout the course.

`f1tenth_courseVisualizationMsg.msg:`

```
std_msgs/Header header
# Namespace
string ns

# Provided by particle filter
Pose2Df[] particles

# Provided by navigation
# Assumes that the last one in the list is the best option.
PathVisualization[] path_options

# Generic visualization types.
ColoredPoint2D[] points
ColoredLine2D[] lines
ColoredArc2D[] arcs
```

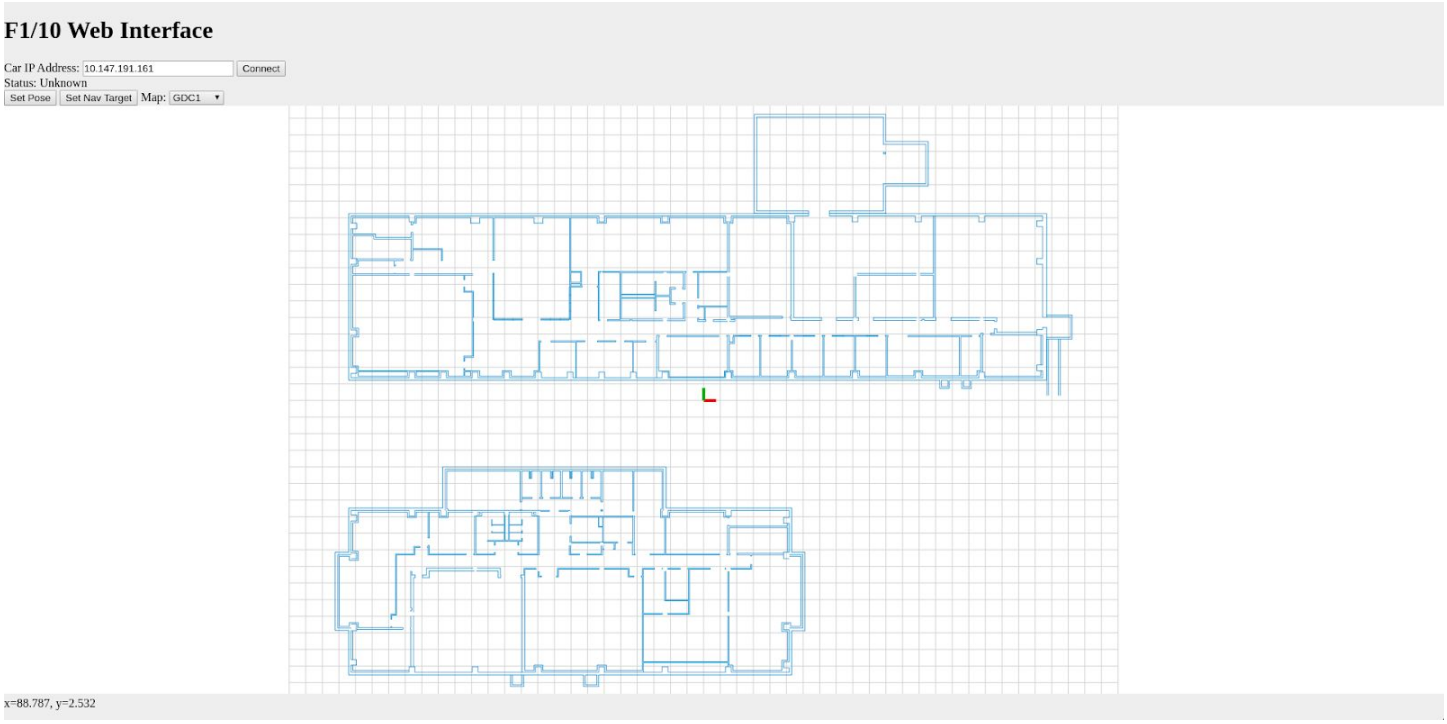
In this message the `particles` and `path_options` are fields used for visualizing later parts of the project.
`particles`: allows visualizing the particles as poses.
`path_options`: displays the possible paths the robot could display from a list. The last path is treated as the best option, and draws the clearance for it.
`points`: displays a list of colored points on the map, can be used for any desired visualization.
`lines`: displays a list of colored lines on the map, can be used for any desired visualization.
`arcs`: displays a list of colored arcs on the map, can be used for any desired visualization.

Using the Web Tool

To view the web visualization tool you must first open the webpage on your local machine

```
google-chrome ~/f1tenth_course/web_rviz.html
```

This will open a web page similar to the below:



This visualization tool is used as follows:

1. Publish VisualizationMessage from running robot software.
2. Open the visualizer `google-chrome ~/f1tenth_course/web_rviz.html`
3. Set the ip address in the visualizer to the ip of the robot and press connect.
4. Select the correct map from the drop down box.
5. You should see the robot visualized at a location on the map (not initially the correct location).
6. The set pose and set nav target buttons will do nothing until you have written the required navigation and localization software as part of the course.

Appendix A: ROS

The code for this course is using features from the Robot Operating System (ROS). We will describe some of the ros commands that will be useful to you below, but if you are interested in learning more, there are a number of ros tutorials as well.

Important Note: There is a lot more information in these tutorials than you will need for this course, and we are NOT using the catkin build system. ROS tutorials

Ros Commands of Interest

Complete information about each command and usage instructions can be found by following the links to the ROS wiki:

1. roscore: Runs a collection of nodes and programs that are prerequisite for running ROS code.
2. `rostopic` : Displays information about currently subscribed or published topics.
3. `rosnode` : Displays information about and controls currently running ROS nodes.
4. `rosmsg`: Displays information about message types.
5. `rosbag`: Records and plays back data from rostopics

Appendix B: tmux

The standard workflow for this class requires you to ssh to the robot in order to run your code.
tmux is a tool that can make this process easier. tmux is a terminal multiplexer, which has two major features that are important:

1. tmux allows you to open multiple terminal panes or tabs within a single ssh session making it easier to run multiple commands and view files.
2. Tmux allows you to open a terminal run commands, and then detach from the session leaving everything as you left it and commands running. You can then reattach these sessions at any time to pick up where you left off.

Using tmux can initially have a learning curve, but a number of cheat sheets exist on the internet that can be used as references if you want to explore these features.

The core of using tmux is as follows:

1. First start a tmux session with the command: `tmux`
2. In a tmux session you can use the terminal as normal, and you can enter tmux commands with special key combinations. All tmux commands require you first press `ctrl + b` followed by another key combination:
    a. Open a new pane: vertical `ctrl + b, %` , horizontal `ctrl + b, “`
    b. `Navigate between panes:` `ctrl + b, arrow key`
    c. `Open a new tab:` `ctrl + b, c`
    d. `Navigate between windows:` `ctrl + b, n`
    e. `Detach session:` `ctrl + b, d`
    f. `List all sessions:` `tmux ls`
    g. `Attach session:` `tmux a -t <session_name>`