Racecar Info

1 ROS Environment Setup

1. Follow steps in the ROS Environment Setup to get your environment setup.
   (a) We are using the **Kinetic** ROS distrobution.
   (b) Your course machine should already have ROS installed, so you can skip **Step 1**. If you'd like to install it on your own machine, you're free to do so.
   (c) The robot will already have a workspace, but you will need to create one on your lab machine
   (d) Be sure to add the following lines to your `~/.bashrc` if not already there, and source it: `source /opt/ros/kinetic/setup.bash`
2. If not already there, download the following base driver packages to your catkin workspace on your lab machine to get started (these packages should already be loaded onto the car). If you discover missing packages when compiling, they can be installed according to ROS Installation.
3. Compile the workspace: `$ cd ~/catkin_ws; catkin_make`
4. There are two important environment variables used for communication between ROS nodes:
   (a) `ROS_IP`: This can be found via `$ ifconfig eth0` (or whatever interface is connected to the UW network) and taking the `inet addr` value:
   ```
   eth0      Link encap:Ethernet  HWaddr 12:34:56:78:9a:bc
             inet addr:xxx.xxx.xxx.xxx  Bcast:...  Mask:...
   ```
   `$ export ROS_IP=xxx.xxx.xxx.xxx`.
   On the car, the IP is a function of the car number: `172.16.77.XX` where `XX` is your car number.
   (b) `ROS_MASTER_URI`: This is dependent on where `roscore` is running. Use the same technique as above to get the corresponding IP address. **The default port is 11311** `$ export ROS_MASTER_URI=http://<ROS Master IP>:11311`. **Note this is a URI, not just an IP**.

2 Connecting to the Car

In order to run our code on the physical car we need to setup access. Once you can SSH into the car you should clone your repository onto the car.

1. Plug in the battery packs (one for the TX2 and one for the VESC). See Figure 1. **You must plug in both batteries before turning the TX2 on or you may have issues with the VESC.**
2. Turn on the TX2. See Figure 2.
3. Use the following instructions the first time you use the car. `<car-ip>` is the IP of your car: `172.16.77.XX`, where `XX` is the number on the body of the car. The password is `nvidiaXX`. `<car-name>` (The hostname) is the name on the side of the car.
   (a) To setup `ssh` via host name use the script in `lab0/scripts`: `$ sudo ./add-car/hostname.sh <car-name> <car-ip>`. Now you should be able to `ssh` and `scp` just by hostname. *This has only been tested on Ubuntu and Mac, you may have to use a different setup configuration if you want to ssh via another OS.*

(b) Use `$ ssh-copy-id <car-name>` to copy your local public key to the car. After doing this you should no longer have to type in the password to ssh to the car.

4. Now `ssh` into the car: `$ ssh <car-name>`
5. Start teleoperation: `$ roslaunch racecar teleop.launch`

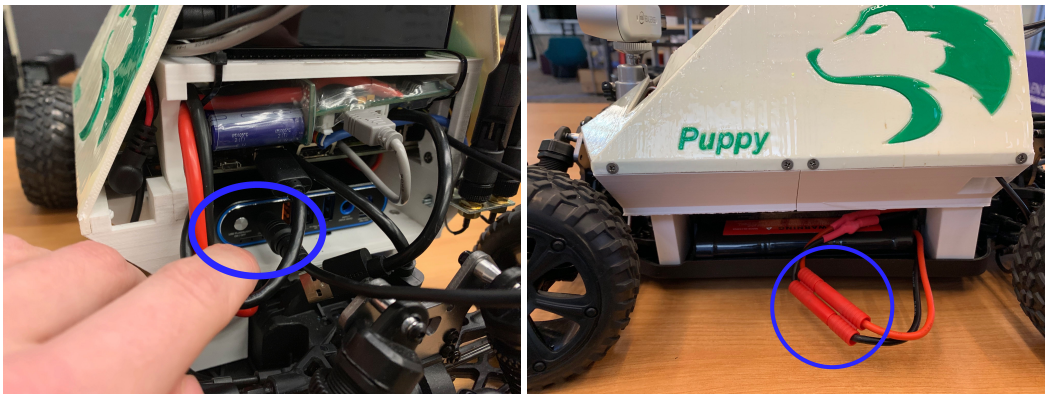

Figure 1: Left: TX2 power plugin. Push the button until the lights are solid green. Right: VESC battery plugin.

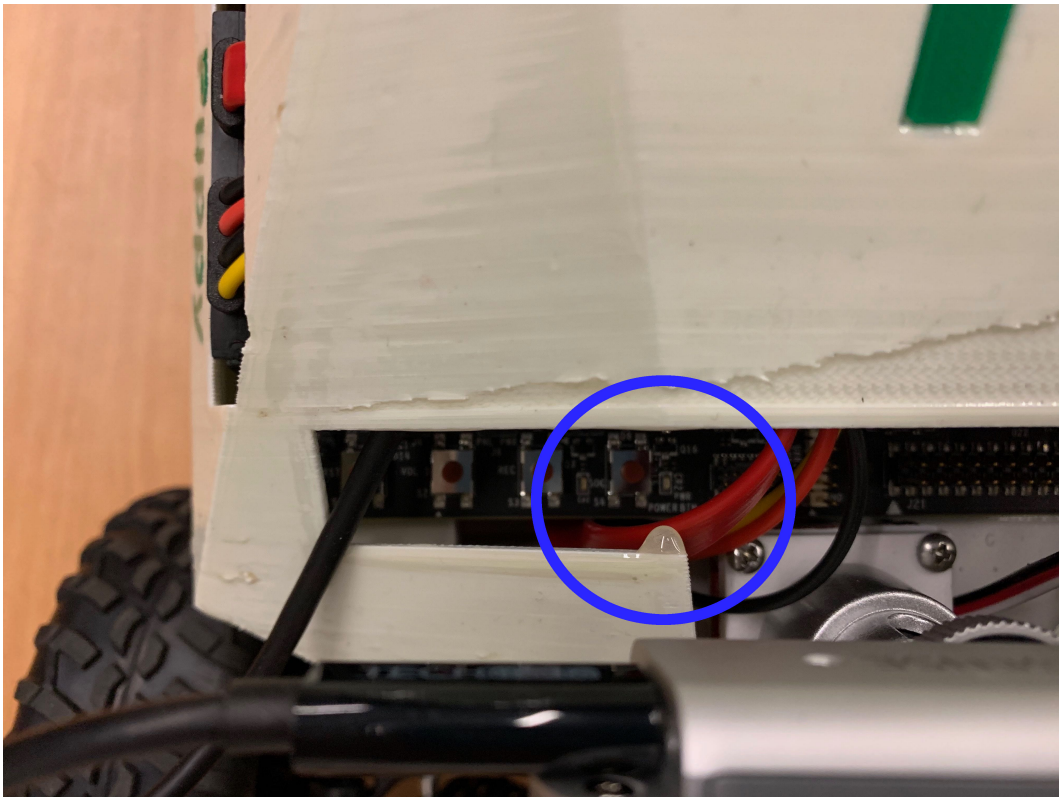

Figure 2: Press the the circled button, labeled `POWER BTN` on the board. The light next to the button will emit green when powered on.

3 Tuning VESC parameters

The Vedder Electric Speed Controller (VESC) is the chip we send commands to to be executed on the car. We want to send a speed and steering angle to the VESC and have it convert to electical RPMs. The speed is converted via:

```
erpm = speed_to_erpm_gain * speed (m/s) + speed_to_erpm_offset
```

The steering angle is converted to a continuous 0 to 1 servo value:

```
servo_value = steering_angle_to_servo_gain * steering_angle (radians) +
                    steering_angle_to_servo_offset
```

These differ on individual cars slightly, so we will start off by tuning these parameters. Once you do this once, you shouldn't have to do it again, but there is no harm in doing it as having suitable parameters will affect the odometry from the VESC.

The steps to tune are as such:

1. Adjust `steering_angle_to_servo_offset` until the car drives sufficiently straight when commanded to. Use the controller to run the car straight down a hallway, use your best judgement to gauge straightness.
2. Adjust `servo_min` and `servo_max` so that the car's minimum turning radius when turning left is similar to its minimum turning radius when turning right. Imagine driving the car in a complete circle turning the car as hard left and as hard right as possible. The radius of both circles should be roughly equal.
3. Drive the car straight forward for a fixed, measured 2 meters. Adjust the `speed_to_erpm_gain` parameter until the odometer roughly matches the true distance travelled (use `$ rostopic echo /vesc/odom` to see odometry)
4. Calibrate turning radius: Open `rviz` and visualize the car's odometry. Then drive the car along the arc of a semicircle (so constant nonzero steering angle). Adjust the `steering_angle_to_servo_gain` until the visualization of the odometry and truth match.

4 Getting Started Information and References

1. Become familiar with Publishers and Subscribers interfaces and functionality. Note that in python, callbacks are executed asynchronously. Be sure to account for this when accessing shared data in different threads
2. ROS programs you will likely interact with a lot in the course:
   (a) `roscore` (Reference) `$ roscore` will start the requisite master server needed for other nodes to run on. You don't *necessarily* need to run this as your first launch will run a master server, but if you exit from the launch, the ROS service will be exited as well, disconnecting all other nodes.
   (b) `rviz` (Reference) `$ rosrun rviz rviz` will start an instance of `rviz`, the visualization tool used to interact with your code. If you're running code on the car you'll want to use rviz on your local desktop and set your `ROS_MASTER_URI` to point to the car's `roscore`.
   (c) `racecar` `$ roslaunch racecar teleop.launch` will start the teleop tool required to run the physical car with the handheld controller and your code.
   (d) `mushr_sim` `$ roslaunch mushr_sim teleop.launch` will start the simulator for the difference components of the car.
   (e) `map_server` (Reference) The map server provides a map to your code. See `lauch/map_server.launch` for a launch file and `maps/` for different maps you can use.
   (f) Your code! Become familiar with `roslaunch` and Launch files

5 Troubleshooting and General FAQ

This section will be updated throughout the quarter with common issues and remedies.

1. Some packages don't handle forced shutdowns gracefully. If you run into a situation where a node won't start, you should restart your TX2 and get it into a better state. On this note,

you should let `roslaunch` run to completion. If you `Ctrl + \` (send a `SIGKILL`) to the `roslaunch` it won't be able to kill the ROS node processes it created.