Open Source Rover

Software Install Instructions

Jet Propulsion Laboratory
California Institute of Technology

Contents

1 Code Overview

Next I'll go through briefly how the different threads and functions communicate with each other. On boot of the RPi two processes start, one of them is the test.py from rover soon to change to main.py, and the other is screen.py which updates what the LED matrix displays.

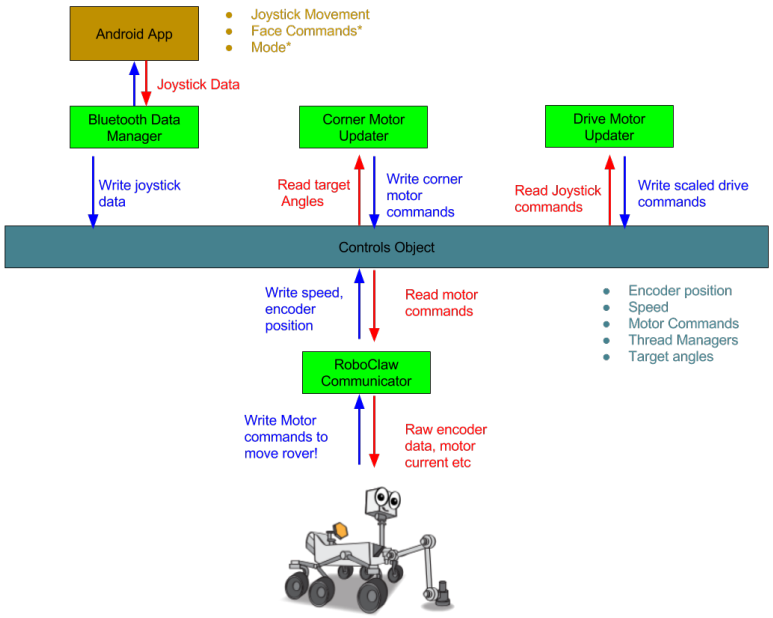

Figure 1: Software Architecture

The remote will kick off a thread to listen for joystick data from the app, a thread monitoring the positions of each of the corners, a thread that handles all the communication with the RoboClaws, and another that will send commands to the LED display. The controls class contains information such as the position of the absolute encoders, signals to send to the motors, and thread management variables. For more in-depth explanation of the software refer to the comments in the source code.

2 Folder/file hierarchy

First I'll go over the folder structure and what each of the files in our system to. Refer to Figure 2 as reference for this section.

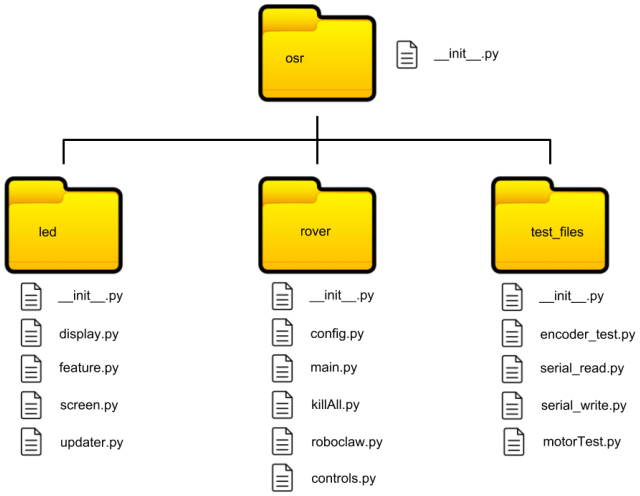

Figure 2: Folder structure

The rover folder contains the code necessary to run the robot and works as following:

- config.py contains the calibration scalings as well as geometric distances necessary for calculations of speed and turning angles
- main.py runs on boot from /etc/rc.local and will look for bluetooth connection, and start the necessary threads to control the rover, and using a unix socket send commands to the thread running the LED screen as well.
- killAll.py is a command line version of the killMotors function, in case something wrong happens and you need to stop the rover from terminal (we used this in testing purposes as a 2nd fail safe and decided to include it. When writing your own driving functions make sure to build in fail safes to shut down motors as well)

- roboclaw.py is the file provided from the Ion Motion website that communicates to the RoboClaw motor controllers
- controls.py has all the functions that do the driving and calculations, as well as communication to the motor controllers

The led folder contains the code necessary to run the actual rover and works as following:

- display.py manages a matrix that that corresponds to each LED color, built from the features
- feature.py creates the object to be used and placed on the LED matrix
- updater.py contains the code that deals with turning on/off individual LEDs in the LED matrix
- screen.py decides which objects to turn on/off based on inputs from unix socket

The test_files folder contains code to help test and make sure that the pieces are all independantly working:

- encoder_test.py allows you to test the encoder scalings before turning the rover on, making sure they read the right angles
- serial_read.py tests the serial read on RX GPIO pin 15
- serial_write.py tests the serial write on TX GPIO pin 14
- motorTest.py is a unittest framework that allows you to test the system. It will test roboclaw serial connections, encoder readings, and that the motor and encoder directions are correct with each other.

3 Control System

There are two systems we have to look at when designing the controls mechanics for the rover. One is how fast each of the drive motors must move, and the other is where each of the corners need to point in order for the rover to be able to make turns.

3.1 Drive Motor calculations

The 6 wheel rover design employs Ackerman steering, which can be seen in Figure 3. The rover steers about a point which lies on the line that passes through the two center wheels as they are both fixed. All 6 wheels are lie perpendicular to the vector from that point to the corner wheel, which we can see causes a more drastic angle in steering for the wheels closer to the point than further away.

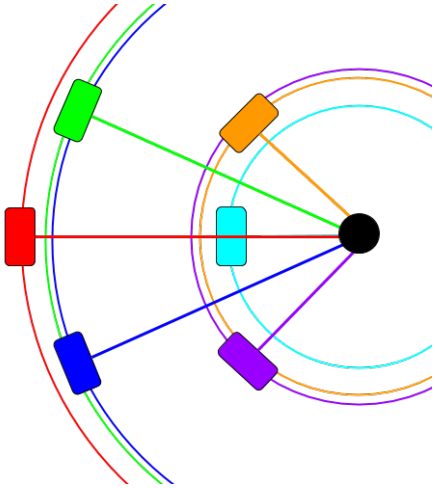

Figure 3: Ackerman Steering

Lets go step by step though how to calculate the speed of each of the drive wheels. To start we will look at the rover which will be turning about a point P at radius r, shows in Figure 4

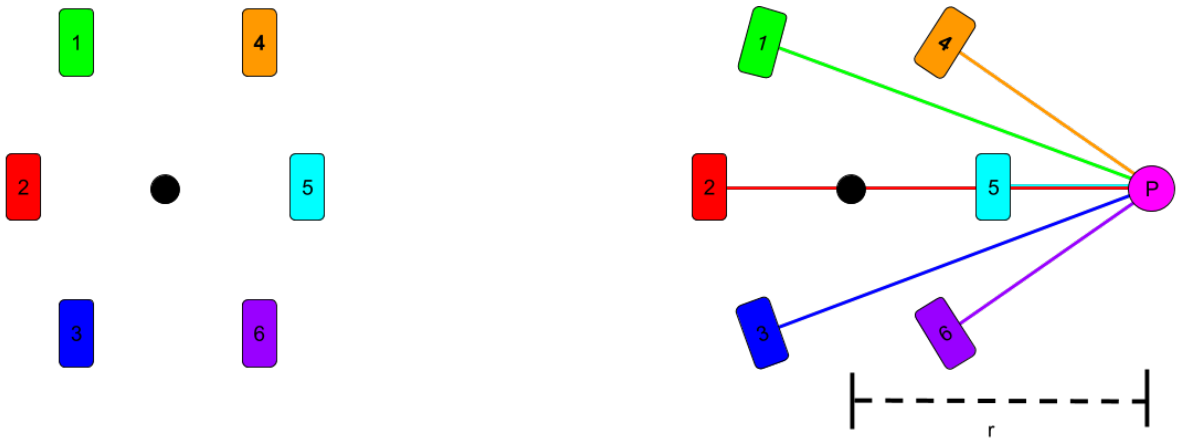

Figure 4: Rover turning about point p

From the controller we will send a signal for the rover to drive forward. Lets call that speed X, and assume we sent a speed of 100. By convention this will send tell the middle of the rover it should move at the max speed of 100. If however we are turning the wheels will need to distribute that speed differently to each wheel. In order for the middle of the rover to move at it's max speed we will distribute the max speed to the furthest out wheel, in this case that is wheel #2 (it will always be the middle wheel furthest from the turning point). So we now choose that the speed 100 is distributed to wheel 2 shows in Figure 5.

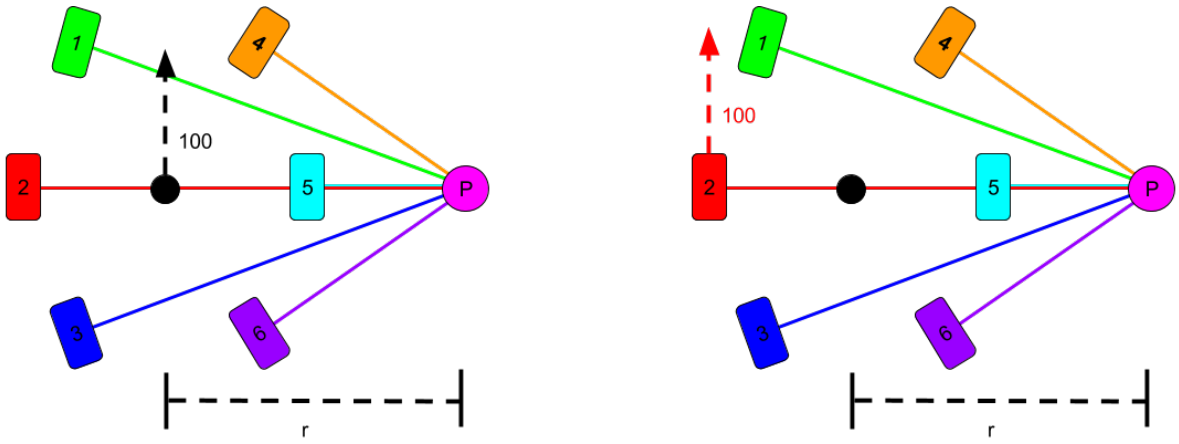

Figure 5: Rover distributing speed to fastest wheel

Now we look at how to distribute speed to the other wheels. We will make use of arclengths of the circles the wheels turn about, and the relation in Equation 1

$$S = R\theta \tag{1}$$

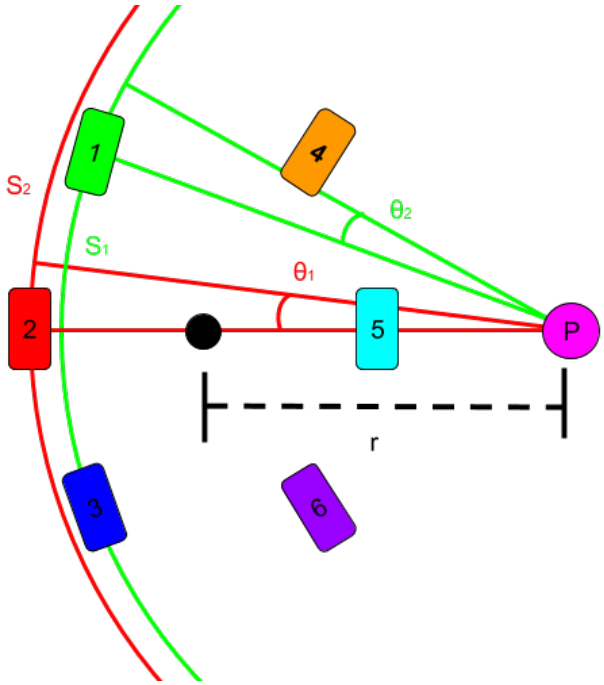

Figure 6: Arclengths of driving curves

In order for the entire rover to drive cohesively each wheel must travel the same number of degrees from Equation 1 in the same time, so we can solve for the relation between each of the wheels now.

$$\theta_1 = \frac{S_1}{R_1} \qquad \theta_2 = \frac{S_2}{R_2} \tag{2}$$

$$\theta_1 = \theta_2 \qquad S_1 = S_2 \frac{R_1}{R_2} \tag{3}$$

From Equation 3 we can see that the ratio of the distance traveled, and thus the speed necessary, of each of the wheels is going to be given by the ratio of the radius of the arc each wheel is turning about. The last step is to figure out the radius of each arc.

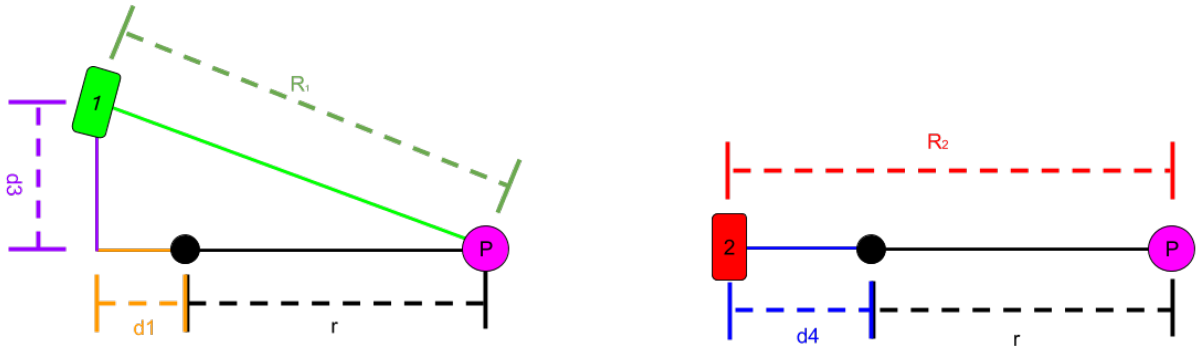

Figure 7: Radius of arc for turning

$$R_1 = \sqrt{{d_3}^2 + (d_1 + r)^2} \qquad R_2 = d_4 + r \tag{4}$$

These distances d_1, and d_3 are geometric relations based on the mechanical lengths of the robot itself. Putting this all together lets look at an example speed and turning distance.

Variable	Physical description
d_1	Horizontal distance between middle of rover and the corner wheels
d_2	Verticle distance between the middle of rover and back corner wheels
d_3	Verticle distance between the middle of the rover and the front corner wheels
d_4	Horizontal distance between the middle of the rover and the center wheels

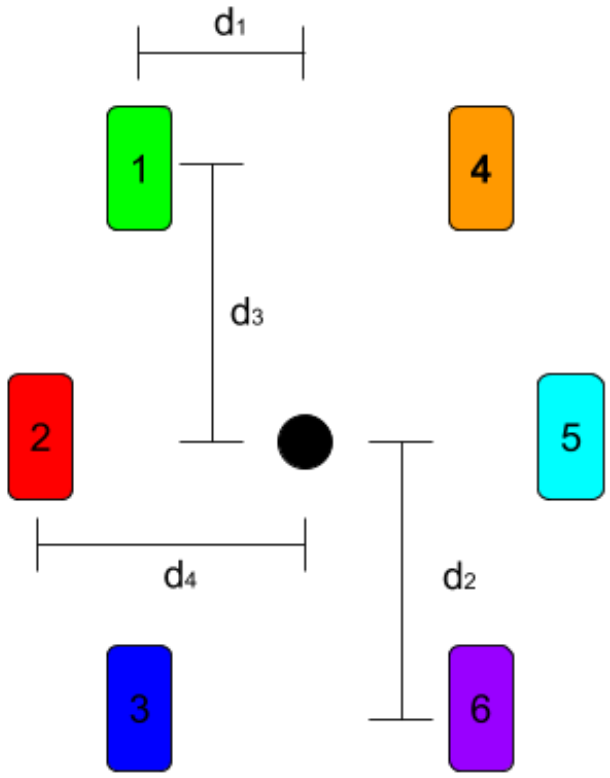

Figure 8: Physical distance of wheel geometry

$$d_1 = 7.254in \qquad d_2 = d_3 = 10.5in \qquad d_4 = 10.073in \tag{5}$$

For this example we will have a controller signal of max speed, being 100, and a turning radius of 30 inches (by convention the positive turning radius is turning to the right and negative is turning to the left), depicted by Figure 9

$$r = 30in \qquad X = 100 \tag{6}$$

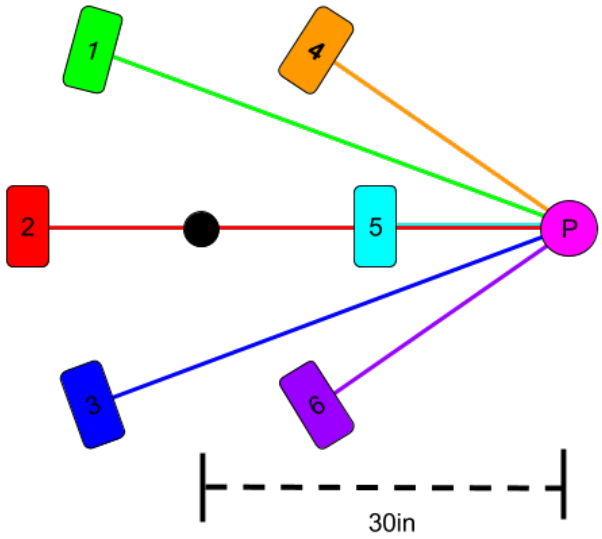

Figure 9: 30 Inch Turning radius

Using Equations 3 and 4 we get the following relations for the speed distribution. We subsititue Equation 4 into 3, which solves for the arc length distance. This is analagous to the speed of each wheel, as speed is just the $speed = \frac{distance}{time}$.

$$V_1 = X\frac{R_1}{R_2} \qquad = X\frac{\sqrt{d_3^2 + (d_1 + r)^2}}{r + d_4} \tag{7}$$

$$V_2 = X \tag{8}$$

$$V_3 = X\frac{R_3}{R_2} \qquad = X\frac{\sqrt{d_2^2 + (d_1 + r)^2}}{r + d_4} \tag{9}$$

$$V_4 = X\frac{R_4}{R_2} \qquad = X\frac{\sqrt{d_3^2 + (r - d_1)^2}}{r + d_4} \tag{10}$$

$$V_5 = X\frac{R_5}{R_2} \qquad = X\frac{r - d_4}{r + d_4} \tag{11}$$

$$V_6 = X\frac{R_6}{R_2} \qquad = X\frac{\sqrt{d_2^2 + (r - d_1)^2}}{r + d_4} \tag{12}$$

Substituting in the values of the speed X and distances it leads to the motors spinning at the following speed % (out of 100).

Motor	Motor speed % of max (100)
1	96.59
2	100
3	96.59
4	62.52
5	49.73
6	62.52

We can see that at a small turning radius such as 30 inches we will see a drastic difference between the motor speeds. As the turning radius increases we will see less and less difference between the speeds, up until the point where the calculated turning radius is 250 inches or greater. This is the point in which the encoders on the corners no longer have spacial resolution to differentiate between 1 °and 0 °and thus we say that anything above a turning radius of 250 inches is not turning at all and it gives all motors the same speed.

3.2 Corner Motor Positions

The rover will take input from the controller which will be in the range of -100 to 100 inclusive. This is the percent of turning radius the wheels should be at, with negative numbers turning to the left, positive numbers to the right, and the higher the magnitude the number the tighter turning radius, ie. -100 means the tightest turn possible to the left side. This "tightest" turning radius is determined by a combination of the geometric distances

of the rover as well as the physical hard stop limitations of the corner motors. Each of the corner motor is allowed to turn up to 45 degrees before hitting a physical hard stop, shows in Figure 10.

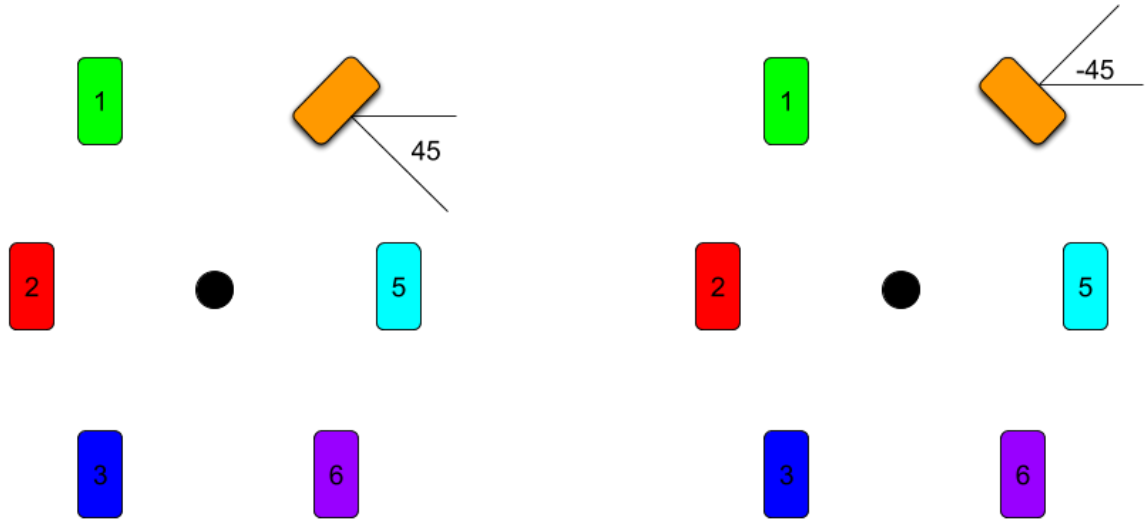

Figure 10: Corner angle limitations

In order to figure out the tightest turning radius we will need to solve for the closest point the corner wheel can be perpendicular to at its 45°limitation in Figure 11.

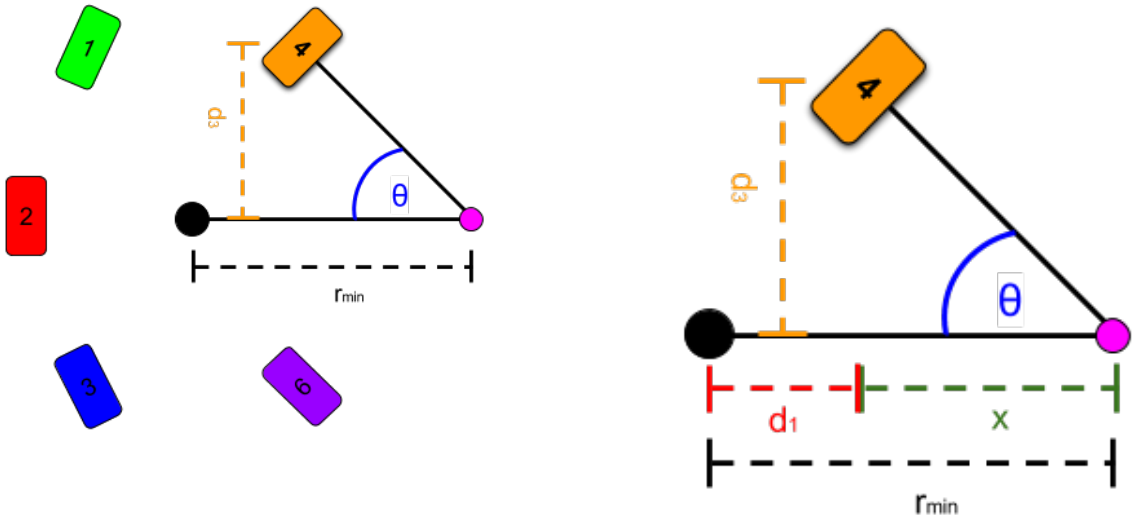

Figure 11: Minimum Turning Radius calculations

$$tan(\theta) = \frac{d_3}{x} \qquad r_{min} = d_1 + x \tag{13}$$

$$r_{min} = d_1 + \frac{d_3}{tan(\theta)} \qquad r_{min} = 7.254 + \frac{10.5}{tan(45°)} \tag{14}$$

$$r_{min} = 17.754 inches \tag{15}$$

Here we can see that the minimum turning radius to each side will be at 17.754 inches. In order to not push the system to the physical hard stops in the software it is set to be at 20 inches to either side. As mentioned before the "minimum" turning radius is at 250 inches to either side, as this is the location in which the encoder no longer can differentiate between it being at 0°and 1°. In order to calculate what target angle each corner should be at given the turning radius input from the controller we will work backwards through Equation 13.

$$tan(\theta) = \frac{d_3}{x} \qquad \theta = tan^{-1}(\frac{d_3}{R - d_1}) \tag{16}$$

Where R is the input turning radius from the controller.