JPL Open Source Rover Code

This repository contains the code that runs on the Raspberry Pi (RPi) and Arduino to control the JPL open source rover (osr). This includes the Arduino code that controls the LED matrix. The rover runs on ROS1 and its code currently is agnostic of the specific distribution (tested on Kinetic, Melodic, Noetic). All Python code is currently Python2.

Status: actively developed. Please refer to the issues tab in GitHub for an overview of ongoing work.

Internals & structure

Please refer to README files associated with each folder for insight in how components work and what they do. This is also the place to look when you have modifications on your rover that require the code or parameters to be changed.

- The ROS overview gives an overview of the setup related to ROS and links to specific implementations such as how the drive and corner commands are being calculated
- The Arduino readme details the code that runs on the Arduino, used to control the LED screen.

1. Flashing the Arduino code

In this section we will be flashing the code that runs on the arduino to control the LED matrix in the head. The following steps should be performed on your laptop or development machine (not the raspberry pi)

1. Install the Arduino IDE used for loading code onto the arduino
2. Download the Arduino code:
   1. Navigate to the repo and click the green “Clone or download” button. Choose “Download ZIP”.
   2. Unzip/extract and open the downloaded zip file. Then, select the Arduino folder and create a new zip file of just that Arduino folder. Name it OsrScreen.zip
3. Load the sketch onto the Arduino
   1. Unplug the Arduino shield JST cable so the Arduino isn’t powered by the control board
   2. Connect the Arduino to your development machine with USB cable
   3. Open Arduino IDE
   4. Select Sketch -> Include Library -> Add .Zip Library
   5. Select the OsrScreen.zip folder created previously
   6. Click the Upload button in the Sketch Window
4. To load the example in the Arduino IDE: File -> Examples -> OsrScreen -> OsrScreen

2. Setting up the Raspberry Pi (RPi)

In this section, we’ll go over setting up the Raspberry Pi (RPi) and setting up all the code that will run the rover. Our rover uses ROS (Robotic Operating System), which will be installed on your choice of operating system; we will set these up below.

These instructions should work for both the RPi 3 and 4. You are free to use other versions of RPi, ROS, or OS, but setting these up is not covered here and it is not guaranteed that those will work.

2.1 Installing the operating system

The first step is to install the Ubuntu (Recommended), or Raspbian OS on your Raspberry Pi.

2.1.1 Ubuntu 18.04 (recommended)

Download Ubuntu 18.04 from here for your RPi version. Go for the 64 bit version. **Note that as of**

7/30/2020, the link for 18.04 64bit for rpi4 is broken, and points to the rpi3 version. You can grab the correct rpi4 version from this direct link.

Follow the instructions on the download page for preparing the image for the RPi. Namely:

- Flash Ubuntu onto your microSD card. There are instructions for doing this on Ubuntu, Windows, and Mac
- Attach a monitor and keyboard to the RPi
- Insert the flashed SD card in the RPi
- Power it on
- Login, using "ubuntu" for the username and password

You should now be logged in to your newly minted copy of Ubuntu 18.04 server!

2.1.2 Raspbian (not recommended)

To install Raspbian, we recommend following the "Getting started with Raspberry Pi" tutorial below.

NOTE: These instructions were tested and verified using Raspbian Stretch. Newer versions of Raspbian (such as Buster) may cause complications with these instructions. We therefore recommend installing Raspbian Stretch instead of the latest/default version of Raspbian, which the tutorial below will recommend. Do not flash NOOBS onto your card; instead download the Raspbian Streth image below and skip the NOOBS step in the tutorial.

In addition, we have created a pre-built image with both Raspbian Stretch and ROS already installed. You can download this image as a last resort if you can't get Raspbian and ROS installed and working on your own, but please note that you will miss out on some the learning opportunities that come with debugging a software installation by doing so.

- Tutorial for installing the latest version of Raspbian on a new Raspberry Pi
- Direct downloadable Raspbian Stretch image
- Custom JPL image with both Raspbian Stretch and ROS pre-built

NOTE: If you install the Custom JPL image with Raspbian Stretch and ROS pre-built, skip to step 2.4.

Once you finish the above tutorial and install Raspbian Stretch, you should be able to boot your Raspberry Pi and see a desktop!

2.2 Installing ROS

2.2.1 ROS Melodic on Ubuntu 18.04

Use these instructions if you followed subsection 2.1.1 above.

The below steps are based off of these instructions. Consult them for more details.

Setup

```
# Setup your computer to accept software from packages.ros.org
sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc) main" >
/etc/apt/sources.list.d/ros-latest.list'

# set up keys
sudo apt-key adv --keyserver 'hkp://keyserver.ubuntu.com:80' --recv-key
C1CF6E31E6BADE8868B172B4F42ED6FBAB17C654
```

Installation

```
# make sure your Debian package index is up-to-date
sudo apt update
```

```
# desktop-Full Install: (Recommended) : ROS, rqt, rviz, robot-generic libraries, 2D/3D
simulators and 2D/3D perception
sudo apt install ros-melodic-desktop-full
```

Dependencies for building packages

```
# install 'em
sudo apt install python-rosdep python-rosinstall python-rosinstall-generator python-
wstool build-essential

# initialize rosdep
sudo apt install python-rosdep
sudo rosdep init
rosdep update

# sensor msgs dependencies
rosinstall_generator sensor_msgs --rosdistro melodic --deps --wet-only --tar >melodic-
sensor_msgs-wet.rosinstall
wstool init src melodic-sensor_msgs-wet.rosinstall
```

2.2.2 ROS Kinetic on Raspbian

Use these instructions if you followed subsection 2.1.2 above.

If you do not download the pre-built Raspbian Stretch / ROS image, you will need to build ROS yourself.

First, make sure you update all the packages on your pi:

```
sudo apt-get update
sudo apt-get upgrade
```

(The above commands may take a few minutes to run.)

Next, we will install ROS (and specifically, the version called 'Kinetic').

NOTE: Follow the ROS installation directions in the link below CAREFULLY! Do not ignore warnings, as they will lead to issues later in the installation. Some known "gotchas" to note before you get started are:

- Make sure you install the Kinetic "desktop full" version of ROS, not the desktop ros-comm, or other versions.
- ROS changed their GPG keys as of June 7 2019. The ROS install instructions do not mention this as of July 2019 and will not work unless you update your GPG keys. See here for instructions on how to update your GPG keys.
- Make sure you follow directions for KINETIC, not Indigo, Melodic, or any other ROS version.

Once you familiarize yourself with the above issues, go ahead and complete your ROS installation by following these instructions

2.3 Setting up serial communication on the RPi

2.3.1 Ubuntu 18.04

You'll need to create dev files for serial0 and serial1 and update permissions:

```
sudo ln -s /dev/ttyS0 /dev/serial0
sudo ln -s /dev/ttyAMA0 /dev/serial1

# change the group of the new serial devices
sudo chgrp -h tty /dev/serial0
sudo chgrp -h tty /dev/serial1

# The devices are now under the tty group. Need to add the ubuntu user to the tty
group:
sudo adduser ubuntu tty
```

```
# also need dialout for ttyS0 and ttyAMA0
sudo adduser ubuntu dialout

# update the permissions for group read on the devices
sudo chmod g+r /dev/ttyS0
sudo chmod g+r /dev/ttyAMA0
```

todo: should update instructions in this subsection for making this setup permanent

2.3.1 Raspbian

In this project we will be using serial communication to talk to the motor controllers. Serial communication is a communication protocol that describes how bits of information are transferred between one device and another. You can find more information on serial communication here

Run the following commands on the Pi to setup/enable the serial communication for the RoboClaws:

```
sudo raspi-config
```

In the raspi-config menu, set the following options:

- Interface Options –> Serial
- Would you like a login shell to be accessible over serial? –> No
- Would you like the serial port hardware to be enabled? –> Yes
- Would you like to reboot now? –> Yes

Once the Pi reboots, open up a terminal again and look at the serial devices:

```
ls -l /dev/serial*
```

Make sure that this shows `serial0` -> `ttyS0` . If it does not, ensure that you have followed every step in this tutorial in order. Next, edit the `/boot/cmdline.txt` file:

```
sudo nano /boot/cmdline.txt
```

Change ONLY the part with `console = ...` to read `console=tty1` and remove any other instance where it references console. The first bit of that line should look similar to the below:

```
dwc_otg.lpm_enable=0 console=tty1 root=/dev/mmcblk0p7
```

See here if you need additional help with above steps.

Once you’ve completed the above steps, reboot the Pi.

```
sudo reboot
```

2.4 Setting up ROS environment and building the rover code

2.4.1 Setup ROS build environment

First we’ll create a workspace for the rover code.

On the Raspberry Pi, open up a terminal `(ctl + alt + t)` and then type the following commands:

```
# Create a catkin workspace directory, which will contain all ROS compilation and
# source code files, and move into it
mkdir -p ~/osr_ws/src && cd ~/osr_ws
# Build a basic, empty catkin project
catkin_make

# If this doesn’t report any errors, check if there are two new directories:
ls ~/osr_ws/build
ls ~/osr_ws/devel
# the above commands should not report 'No such file or directory'
# Source your newly created ROS environment
# EITHER, if you installed ROS melodic
source /opt/ros/melodic/setup.bash
# OR, if you installed ROS kinetic
source /opt/ros/kinetic/setup.bash
```

2.4.2 Clone and build the rover code

In the newly created catkin workspace you just made, clone this repo:

```
cd ~/osr_ws/src
git clone https://github.com/nasa-jpl/osr-rover-code.git

# install the dependencies
rosdep install --from-paths src --ignore-src
catkin_make

# add the generated files to the path so ROS can find them
source devel/setup.bash
```

The rover has some customizable settings that will overwrite the default values. Whether you have any changes compared to the defaults or not, you have to manually create these files:

```
cd ~/osr_ws/src/osr-rover-code/ROS/osr_bringup/config
touch physical_properties_mod.yaml roboclaw_params_mod.yaml
```

To change any values from the default, modify these files instead so they don't get tracked by git. The files follow the same structure as the default.

2.4.3 Add ROS config scripts to .bashrc

The `source...foo.bash` lines above are used to manually configure your ROS environment. We can do this automatically in the future by doing:

```
# use "melodic" or "kinetic" below, as appropriate
echo "source /opt/ros/melodic/setup.bash" >> ~/.bashrc
echo "source ~/osr_ws/devel/setup.bash" >> ~/.bashrc
```

This adds the `source` lines to `~/.bashrc`, which runs whenever a new shell is opened on the RPi - by logging in via ssh, for example. So, from now on, when you log into the RPi your new command line environment will have the appropriate configuration for ROS and the rover code.

3 Testing serial comm with the Roboclaw motors controllers

Run the roboclawtest.py script with all of the motor addresses:

```
cd ~/osr_ws/src/osr-rover-code/scripts
python roboclawtest.py 128
python roboclawtest.py 129
python roboclawtest.py 130
python roboclawtest.py 131
python roboclawtest.py 132
```

Each of these should output something like, within a very short execution time:

```
(1, 'USB Roboclaw 2x7a v4.1.34\n')
(1, 853, 130)
```

If the script seems to hang, or returns only zeros inside the parantheses, then you have a problem communicating with the given roboclaw for that address. Some troubleshooting steps in this cases:

- Make sure you followed the instructions in the "Setting up serial communication" section above, and the serial devices are configured correctly on the RPi.
- Also make sure you went through the calibration instructions from the main repo and set the

proper address, serial comm baud rate, and “Enable Multi-Unit Mode” option for every roboclaw controller (if multi-unit mode isn’t enabled on every controller, there will be serial bus contention.). If you update anything on a controller, you’ll need to fully power cycle it by turning the rover off.
- If you’re still having trouble after the above steps, try unplugging every motor controller except for one, and debug exclusively with that one.

4 Manual rover bringup

In a sourced terminal (`source ~/osr_ws/devel/setup.bash`), run

```
roslaunch osr_bringup osr.launch
```

to run the rover.

Any errors or warnings will be displayed there in case something went wrong. If you’re using the Xbox wireless controller, command the rover by holding the left back button (LB) down and moving the joysticks.

5 Automatic bringup with init script

todo: need to update this stuff to reflect changes from this PR: https://github.com/nasa-jpl/osr-rover-code/pull/107

Starting scripts on boot using ROS can be a little more difficult than starting scripts on boot normally from the Raspberry Pi because of the default permission settings on the RPi and the fact that that ROS cannot be ran as the root user. The way that we will starting our rover code automatically on boot is to create a service that starts our roslaunch script, and then automatically run that service on boot of the robot. Further information on system service scripts running at boot.

There are two scripts in the ”Software/Init Scripts” folder. The first is the bash file that runs the roslaunch file, and the other creates a system service to start that bash script. Open up a terminal on the raspberry Pi and execute the following commands.

```
cd /home/pi/osr/Init\ Scripts
sudo cp LaunchOSR.sh /usr/bin/LaunchOSR.sh
sudo chmod +x /usr/bin/LaunchOSR.sh
sudo cp osr startup.service /etc/systemd/system/osr startup.service
sudo chmod 644 /etc/systemd/system/osr startup.service
```

Your osr startup service is now installed on the Pi and ready to be used. The following are some commands related to managing this service which you might find useful:

Description	**Command**
Start service	sudo systemctl start osr startup.service
Stop service	sudo systemctl stop osr startup.service
Enable service (runs on boot of RPi)	sudo systemctl enable osr startup.service
Disable service (doesn’t run on boot of RPi)	sudo systemctl disable osr startup.service
Check status of service	sudo systemctl status osr startup.service
View live service list	sudo journalctl -f

Note: We do not recommend enabling the service until you have verified that everything on your robot runs successfully manually. Once you enable the service, as soon as you power on the RPi it will try and run everything. This could cause issues if everything has not yet been fully tested and verified. Additionally, if you are doing development of your own software for the robot we suggest disabling the service and doing manual launch of the scripts during testing phases. This will help you more easily debug any issues with your code.

Once you have fully tested the robot and made sure that everything is running correctly by starting the rover code manually via `roslaunch osr bringup osr.launch`, enable the startup service on the robot with the command below:

```
sudo systemctl enable osr startup.service
```

At this point, your rover should be fully functional and automatically run whenever you boot it up! Congratulations and happy roving!!

6 Other stuff

For getting ssh to start before login: https://askubuntu.com/a/681768