OSURC Mars Rover Ground Station Quickstart

Corwin Perren, Chris Pham, and Ken Steinfeldt | June 4, 2018

This document will provide a brief overview of the structure, layout, tools, environment, and any other pertinent overview details about the Mars Rover ground station software. This document aims to make reusability in the future easier, and is a good starting reference point for Rover software teams both for running the software and modifying it. Please note that modifying this software is not for the faint of heart, and it should be assumed that anyone working on this is comfortable with Python, Qt, and ROS.

Environment

In order for the software to be launched, a properly set up computer must be provided. This will most likely be the quick deployment ground station box that was built, but bare minimum requirements will be listed here in case it must be set up on a separate system. This software was written and tested on an intel NUC small form factor PC, but any intel/amd based computer should work so long as it has sufficient processing power and at least two display outputs. The computer runs Ubuntu 16.04 LTS along with ROS Kinetic and is connected to two 1080p monitors. Two is needed as the software is designed to launch across both screens. All python written for this project was done in version 2.7 as that is a limitation of ROS Kinetic.

To get the software running, ROS must first be installed and the catkin workspace created. Note that we changed our catkin workspace name from “catkin_ws” to “catkin_workspace”. After this, create a folder called “Github” in the root of the logged in user’s home directory. Clone the Rover repo into this directory, then run the script under the software directory called “ground_station_setup.sh”. This will symbolically link the appropriate packages from the “ros_packages” folder into the catkin_workspace and runs catkin_make to begin building the projects. Once complete, you should be ready to launch the software.

Running the Ground Station

First, ensure that all necessary hardware is plugged in. In our version of the ground station, this includes a joystick, spacenav mouse, mouse, keyboard, and the two 1080p monitors. Also make sure that the radio is connected to the POE port on the right hand side of the ground station and that the radio shows connection to the Rover. At this point, open a terminal and run the following command, “roslaunch rover_main ground_station.launch”. If the rover is connected and running its appropriate software, the ground station should start up in full screen across both monitors. Live video should be seen in the right hand monitor. Pressing the thumb button on the joystick and then moving the y-axis should cause the wheels on the Rover to turn. If this is working, you’ve set everything up correctly. Ctrl-q will quit the application once desired.

Making Changes

Project Structure

The ground station code can be found under the Rover_2017_2018/software/ros_packages/ground_station/src github folder. “ground_station.py” is the main file used to launch the software. There is a script in ground_station/scripts that is used when launching it from ROS to get things set up correctly. The ROS launcher for the ground station can be found in the rover_main package under the launch directory. This file starts the ground station as well as the topic transport layers needed to send and receive data from the Rover over the radios.

Back in the ground_station/src directory, the Resources directory contains all static assets needed by the ground station at launch time. This includes images and the raw UI files that define how the GUI is laid out. The Framework directory contains nested python packages for each logical sub-system of the GUI. This includes packages such as MappingSystems and VideoSystems. Most of these packages contain one or more threaded classes that the main launcher file for the program instantiates, spins up, and keeps running. Any time new features need to be added to the GUI, they should either be turned into their own python sub-package, or added to an existing package if applicable.

UI Elements

To make changes to the UI files (named left.ui and right.ui in Resources/UI), qt designer is needed and can be installed via apt. Gui elements can then be dragged in and rearranged in a WYSIWYG fashion. Make sure and uniquely name all GUI elements you wish to access programmatically, as their names reside in the global scope of the window, so no two can be the same.

Threaded Classes

First off, unless you’re very comfortable with Qt, I’d just recommend copying an existing threaded class and modifying it to suit your needs. The minimum methods that must be kept are __init__, run, connect_signals_and_slots, setup_signals, and on_kill_threads_requested__slot. These are consistent across all threaded classes so coordinated startup and shutdown of the application can be done without crashing. Be sure to keep any appropriate class variables in __init__ as needed as well. Once you’ve copied/created a new package, import the class in “ground_station.py” and add your own call to self.__add_thread under the __init__ method of the GroundStation class. If you did everything correctly, whatever code is being run in your new class should be working, and the program should shut down gracefully with Ctrl-q. Keep in mind that your code should not block for extended periods of time otherwise the program will freeze on shutdown.