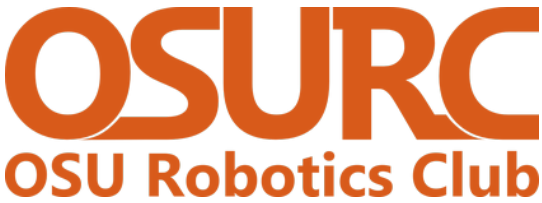


CS Capstone Final Report

12th June 2018

OSU Robotics Club Mars Rover Ground Station

Prepared for

OSU Robotics Club

Nick McComb

Prepared by

Group 30

Ground Station Software Team

Kenneth Steinfeldt
Christopher Pham
Corwin Perren

Abstract

This document contains all of the useful documentation about the OSURC Mars Rover ground station project as it has progressed throughout the year. This includes all technical documents (with revisions), weekly blog posts, code documentation, and lessons learned.

CONTENTS

1 Introduction

1.1 Who Requested the Project?

This project was requested by the Oregon State University Robotics Club's (OSURC) Mars Rover team.

1.2 Why Was the Project Requested?

This project was requested due to the Mars Rover team's need for a complete and functional ground station in a time frame that the existing software team may not have been able to accomplish. Additionally, new requirements for the ground station meant that the increased complexity was most likely going to require a more dedicated team to complete than the volunteers the club normally has working on the Mars Rover project.

1.3 What is the Importance of the Project?

This project solves some of the most common problems for the Mars Rover team in previous competition years. Historically, the team is allowed a 10-15 minute set up time during competition where the Rover must be turned on, ground station connected, and core systems tested before actually competing in a task. In previous years, the Rover's ground station has normally been a laptop with a handful of random pieces of hardware plugged into it over USB. Many times, this hardware has been damaged during transit and caused problems during the setup phase. Additionally, previous software has often been command line driven and extremely specialized meaning the software is difficult to use, difficult to teach, and has ended up getting re-written with each new Rover competition year.

1.4 Our Client and Their Role

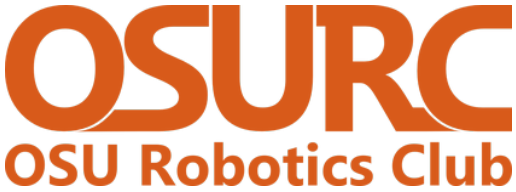
Our client was Nick McComb, team lead and electrical lead of the Mars Rover team for the 2017-2018 competition year. His role was mainly supervisory, helping our group set up the initial requirements for the software, and occasionally making requests for additional features as the year progressed.

1.5 Our Team Members and Roles

- Chris Pham
  - Rover mapping systems
- Ken Steinfeldt
  - Rover status systems
- Corwin Perren
  - Team Manager
  - Rover video systems
  - Program structure / startup

2 Requirements Document

2.1 Original Document



CS CAPSTONE REQUIREMENTS DOCUMENT

NOVEMBER 4, 2017

OSU ROBOTICS CLUB
MARS ROVER GROUND STATION

PREPARED FOR

OSU ROBOTICS CLUB

NICK MCCOMB

PREPARED BY

GROUP 30

GROUND STATION SOFTWARE TEAM

KENNETH STEINFELDT
CHRISTOPHER PHAM
CORWIN PERREN

Abstract

This document covers the design requirements for the OSU Robotics Club Mars Rover Team's Ground Station Software. It begins with details about why the software is necessary, and a general overview of what it needs to accomplish. We then go into further detail about what the specific functional requirements will be, how they relate to each other in terms of dependencies, and then end with stretch goals if there is extra time to complete them.

CONTENTS

1 INTRODUCTION

1.1 Purpose

The purpose of this document is to formally define the characteristics of the software to be implemented. These requirements can then be used to determine if the software is complete once the project is done, while also serving as a reference during development.

1.2 Scope

We will be creating the "Mars Rover Ground Station" software package. This software will allow Rover team members to interact with the Oregon State University Robotics Club's Mars Rover for the purpose of a competition taking place in Utah during June of 2018. The software will provide Rover systems monitoring as well as control of drive and arm systems. During competition, the software will be run on a remote base station operated out of the Mars Desert Research Station or the back of a truck, allowing users to tele-operate the Rover during manned controlled missions, or to monitor progress during autonomous missions. Properly functioning and fully featured ground station software will be a major factor in the success of the Mars Rover during this competition.

1.3 Definitions / Acronyms / Abbreviations

- ROS - Robot Operating System
- QT - A multi-platform GUI and application framework
- GPS - Global Positioning Satellite
- User - Person operating rover from ground control station
- GUI - Graphical User Interface
- RTT - Round-trip Time, the length of time for a signal to be sent plus the length of the time it takes for an acknowledgement of the signal to be received.
- Bogie - The individual wheels on the Rover, each with the ability to be driven independently.
- IMU - Inertial Measurement Unit
- FPS - Frames Per Second
- CAD - Computer Aided Design

1.4 Overview

The remainder of this document consists of 4 sections and will describe the requirements that must be met in order for this project to be considered fully complete. The next section gives a description of the process. The fourth section describes the ground control software's functions, user characteristics, constraints, and assumptions and dependencies. The fifth section describes the specific requirements that must be met for the software to be a success. Finally, stretch goals in the sixth section will describe hopeful, but not mandatory goals for the software.

2 DESCRIPTION

2.1 Product Perspective

This project interacts with and requires the Robotics Club's Mars Rover robotics vehicle in order to be useful. If the Rover is not connected, the software will still be able to launch, but will not perform any useful functions until it has

established a connection to the Rover. In order to accomplish interaction with the rover, the ground station software will use ROS for primary functionality and feedback from the rover.

3 PRODUCT FUNCTIONS

The ground control software will primarily accomplish two tasks:

- Provide Rover control to the user. Direct rover control will be accomplished with two joysticks.
- Enable and present rover feedback to the user. The rover will send health and status information to the ground control station.

3.1 User Characteristics

Users of the Rover Ground Station software will comprise of Rover team leads and members.

3.1.1 User Stories

- User should be able to vary bandwidth as needed to adjust for latency at the event
- User should be able view ground station UI on two HD monitors so that valuable competition time will not be spent switching between screens on a single monitor
- User should be able to add way-points because way-points are needed to find useful points in the competition field and are necessary for autonomous navigation
- User should be able to remove way-points because way-points may become unneeded or need to be reset for a new competition event
- User should be able to edit way-points because there's a chance that the User may enter way-point details incorrectly and will need to change them
- User should be able to change the order of way-points because we might need to traverse back to a location, or fix improperly entered way-points
- User should be able to control the Rover by joysticks because that is the preferred way of controlling the Rover
- User should be able to select video streams because the Rover has more cameras than can be streamed at any one time, and being able to view any camera on demand will help the User drive the Rover and manipulate the arm
- User should be able to view the pitch, yaw, roll of the Rover because we want to know if the rover is on an incline, decline, or about to tip over
- User should be able to switch on and off autonomy because there are competition events that require autonomy and others that require manual control
- User should be able to tell that autonomy is enabled because that alerts the User that they can no longer manually control the Rover
- User should be able to tell that the rover has reached the final way-point after autonomy is enabled because that signals to the judges that the Rover has completed that competition event
- User should be able to view how the arm joints are bent because that can show us if the arm is performing correctly
- User should be able to control the arm because competition events require the use of a Rover arm
- User should be able to view live logs from the Rover because that allows for debugging of hardware systems

- User should be able to set the Rover speed limit so that the User can more easily drive the Rover over difficult terrain
- User should be able to view the CPU usage because the User can use this information to tell if there is a software problem on the Rover or if the Rover is trying to process too much information
- User should be able to view the memory usage because the User does not want the Rover to crash or slow down to unusable levels due to running out of memory
- User should be able to view the drive usage because the User does not want the Rover to crash if the drive is full
- User should be able to see the speed of the Rover because it can be used to determine if the Rover is driving at a speed that is reasonable for the current terrain
- User should be able to see how accurate the GPS positioning is because at the competition there is a requirement to be a certain distance from an object
- User should be able to see what heading the Rover is at because it allows for the User to know what direction to go
- User should be able to see the maximum throughput of the network because it will allow for video stream quality to go up or down depending on the connection quality
- User should be able to see the current latency because it will tell to the User or viewer how long a command will take to become action
- User should be able to confirm connected devices because this will allow for debugging and checking of other connected devices

3.2 Constraints

Due to the nature of the Mars Rover project, there are multiple hardware and software constraints to ensure that the Ground Station software can properly communicate with the Rover. Additionally, the following constraints will help ensure future re-usability of the finalized software, which is a key goal of the Mars Rover team.

The software language allowed is the primary constraint for this project. Python 3 is required by the Mars Rover team as it maintains consistency with the software written on the Rover itself. The team also requires Python as it more easily allows future team members to reverse engineer code to be reused in future projects.

Use of the QT application framework is another constraint placed by the Rover team in order to facilitate rapid prototyping of the user interface for this project. It will also allow for faster and easier modification both by future members of the team, and during competition as that has previously been necessary.

The Rover internally uses ROS to handle routing and interpreting control and status information as well as video data. To be able to view this video data, status information, and to be able to send the Rover control information, this ground station software must also incorporate ROS.

At a hardware level, the Rover team will be providing an Intel NUC desktop computer running Ubuntu 16.04 that the Ground Station software will need to be able to run on. They will also provide two HD monitors, two USB joysticks, as well as a keyboard and mouse that will be used to view and interact with the software.

4 SPECIFIC REQUIREMENTS

4.1 User Interfaces

Upon initial launch, the application should be in full-screen mode across both of the monitors. The left-hand screen will show navigation, Rover status, and miscellaneous Rover controls. The right hand monitor will show the three live video streams from the Rover. In the case that the software is started without a Rover to connect to, the software will display as-such and show placeholder information until such a connection is made.

4.2 Functional Requirements

4.2.1 Statuses / Informational Readouts

1) Rover Connection Status

   - *Description:* This indicator will show a binary state of whether or not the Rover is currently connected.
   - *Rationale:* The Rover must be connected in order for most of the software aspects to function. It is also useful to tell when full network dropouts happen.
   - *Dependencies:* None

2) Joystick Connection Statuses

   - *Description:* Status indicators will visually show binary states as to whether each of the two joysticks are currently connected.
   - *Rationale:* Both joysticks must be connected to drive the Rover.
   - *Dependencies:* Rover Connection Status

3) Rover Battery Status

   - *Description:* This indicator will show the percentage of battery remaining on the Rover.
   - *Rationale:* The User should know how much battery is left to be able to determine remaining runtime and to diagnose other Rover errors.
   - *Dependencies:* Rover Connection Status

4) Rover Voltage Statuses

   - *Description:* These indicators will show the major power rail voltages for the Rover.
   - *Rationale:* The User can use this information to diagnose faulty hardware and to determine cause if Rover systems become non-responsive.
   - *Dependencies:* Rover Connection Status

5) Bogie Connection Statuses

   - *Description:* These indicators will show whether each of the six bogies are connected to the Rover.
   - *Rationale:* This will let drivers of the Rover know if a wheel has failed, which is a common occurrence.
   - *Dependencies:* Rover Connection Status

6) Arm Connection Status

   - *Description:* This indicator will be a binary state of weather or not the mechanical arm is connected or not.

- *Rationale:* This will show the User whether the arm is connected, and therefore whether they will be able to use the joysticks to control the arm.
- *Dependencies:* Rover Connection Status

7) Arm Joint Positions

- *Description:* These indicators will show Rover Arm joint positions in terms of degrees or via visual relationships.
- *Rationale:* These indications will help ensure the User knows where the Arm actually is, and whether it is being moved within its operating limits.
- *Dependencies:* Rover Connection Status, Arm Connection Status

8) Camera Presence Statuses

- *Description:* This indicator will show which cameras on-board the Rover and connected and functional.
- *Rationale:* The User can use such indicators to verify a failed camera for easy troubleshooting.
- *Dependencies:* Rover Connection Status

9) IMU Status

- *Description:* This will indicate the pitch, yaw, and roll of the Rover.
- *Rationale:* This can be used by the User to help navigate uneven terrain and to avoid flipping the Rover.
- *Dependencies:* Rover Connection Status

10) Map Visualizer

- *Description:* This will display a map of the current Rover location and surrounding areas.
- *Rationale:* This can be used to see terrain maps of the area and to place useful way-point and autonomous navigation markers.
- *Dependencies:* Rover Connection Status

11) Rover Computer Statuses

- *Description:* These indicators will show the current CPU, memory, and disk usage of the Rover's on-board computer.
- *Rationale:* These can be used to ensure the Rover's processing computer is not overloaded. It can also show potential software failures while serving as a useful indication that the connection to the Rover is stable.
- *Dependencies:* Rover Connection Status

12) Current Radio Signal Strength

- *Description:* This indicator will display the strength of the connection between the Rover and ground station.
- *Rationale:* The User can use this information to determine if the Rover is driving out of range to ensure a connection is not fully lost.
- *Dependencies:* Rover Connection Status

13) Rover Network Potential Throughput

- *Description:* This indicator will show the maximum theoretical throughput in Kbps between the Rover and this software as provided by the Ubiquity routers.
- *Rationale:* The User can use the information to help tune video bit-rates and resolutions to ensure the connection is not overloaded.
- *Dependencies:* Rover Connection Status, Current Radio Signal Strength, Rover RTT

14) Rover RTT

- *Description:* This indicator will show how long in milliseconds the round trip time for a packet to and from the Rover takes.
- *Rationale:* The User can use this information to help determine if delays in controls are due to network latency or other software error.
- *Dependencies:* Rover Connection Status, Current Radio Signal Strength

15) Estimated Rover Distance

- *Description:* This indicator will show the distance from the Ground Station radio to the Rover radio, as calculated by the Ubiquiti radio systems.
- *Rationale:* The User can use this information to help ensure they do not drive out of communications range.
- *Dependencies:* Rover Connection Status, Current Radio Signal Strength

16) Rover Speed Limit

- *Description:* This display will show the Rover's speed limit in terms of 0 to 100 percent.
- *Rationale:* During manual driving operations it is often useful to limit the max speed of the Rover. A visual indication of this limit will help avoid confusion if the User forgets that a limit has been set, or if the limit is accidentally changed.
- *Dependencies:* Rover Connection Status, Rover Bogie Status

17) Display Data Collection Sensors

- *Description:* Provide data displays for any scientific sensors attached to the Rover.
- *Rationale:* During some competition phases, scientific sensors will be placed into the soil to gather scientific data. This data needs to be sent back to the Ground Station so it can be recorded.
- *Dependencies:* Rover Connection Status

4.2.2 Rover Control

1) Rover Joystick Driving Control

- *Description:* When both joysticks are connected and the Rover is also connected, the joysticks can be used to drive the Rover.
- *Rationale:* The joystick control is needed to be able to drive the Rover.
- *Dependencies:* Rover Connection Status, Joystick Connection Statuses, Bogie Connection Statuses

2) Rover Joystick Arm Control

- *Description:* When both joysticks are connected, the Rover is connected, and the Arm is connected, the joysticks can be used to manipulate the Rover arm.
- *Rationale:* During some competition events, the arm will need to be manipulated by the User to complete competition tasks.
- *Dependencies:* Rover Connection Status, Joystick Connection Statuses, Arm Connection Status

3) Rover Speed Limit Control

- *Description:* When both joysticks are connected and the Rover is also connected, buttons or levers on a joystick may be used to adjust the max speed limit of the Rover.
- *Rationale:* This quick adjustment is often needed to allow for finer control of the Rover over difficult terrain or when manipulating the Rover arm.
- *Dependencies:* Rover Connection Status, Joystick Connection Statuses, Bogie Connection Statuses

4.2.3 Navigation

1) GPS Status

- *Description:* This will show whether the GPS on the Rover is connected and whether it has GPS lock.
- *Rationale:* This is needed to check to see if GPS is working and functioning properly.
- *Dependencies:* Rover Connection Status

2) Rover Speed

- *Description:* This indicator shows how fast the Rover is moving in meters per second via the GPS.
- *Rationale:* This will help the User ensure they are driving within the mechanical limits of the Rover.
- *Dependencies:* Rover Connection Status, GPS Status

3) Rover Heading

- *Description:* This indicator will show what bearing the Rover is heading.
- *Rationale:* The User can use this information to help navigate the competition course.
- *Dependencies:* Rover Connection Status, GPS Status

4) GPS Satellites Connected

- *Description:* This indicator will show how many GPS satellites are currently active.
- *Rationale:* This can help tell the User if the Rover is in a location that is bad for GPS reception, allowing them to drive to a better location before the GPS lock is lost.
- *Dependencies:* Rover Connection Status, GPS Status

5) GPS Accuracy

- *Description:* This indicator will how accurate the current GPS location is in meters.
- *Rationale:* This will allow the User to judge how much the GPS location of the Rover can be trusted, as well as giving useful information about how well the Rover might perform during the autonomous challenge.
- *Dependencies:* Rover Connection Status, GPS Status, GPS Satellites Connected

6) Way-point Placement

- *Description:* Give the ability to place way-points on a map by GPS location, by current Rover location, or via manual GPS co-ordinate entry and add this way-point to a queue. These way-points will also need to show up visually on the map.
- *Rationale:* This will used to map out the course during the autonomous challenge as well as to place points of interest during manual driving events.
- *Dependencies:* Rover Connection Status, GPS Status, GPS Satellites Connected, Map Visualizer

7) Way-point Editing

- *Description:* Allow for editing any way-point in the queue by drag and drop of the way-point, or via manual editing of the GPS co-ordinates.
- *Rationale:* This will allow for easy adjustment of way-points if they were placed incorrectly.
- *Dependencies:* Rover Connection Status, GPS Status, GPS Satellites Connected, Way-point Placement, Map Visualizer

8) Way-point Re-Ordering

- *Description:* Allow for changing the order of way-points.
- *Rationale:* This will allow the User to skip unneeded way-points or fix incorrectly ordered way-points.
- *Dependencies:* Rover Connection Status, GPS Status, GPS Satellites Connected, Way-point Placement, Map Visualizer

9) Way-Point Navigation Path

- *Description:* This should display the expected Rover navigational path when autonomous mode is active.
- *Rationale:* This is useful to follow the progress of the Rover during the autonomy portion of the competition.
- *Dependencies:* Rover Connection Status, GPS Status, GPS Satellites Connected, Way-point Placement, Map Visualizer

10) Active Way-point Selection

- *Description:* Allow for the selection of a way-point from the queue as currently active. Setting the way-point active will update a ”Heading To” indicator to help a User drive to the desired way-point.
- *Rationale:* This is useful to make navigating to competition provided co-ordinates easier.
- *Dependencies:* Rover Connection Status, GPS Status, GPS Satellites Connected, Way-point Placement, Map Visualizer

4.2.4 Autonomy

1) Autonomy Switch

- *Description:* A toggle to enable and disable the Rover’s autonomous mode operation.
- *Rationale:* This is necessary to place the Rover into autonomous mode for that portion of the University Rover Challenge competition.
- *Dependencies:* Rover Connection Status

2) Autonomy Indicator

- *Description:* An alert or indicator for when the Rover has reached its final way-point or has decided to stop.
- *Rationale:* A University Rover Challenge requirement deems that this indicator is necessary to show to the judges when the Rover has completed its autonomy phase.
- *Dependencies:* Rover Connection Status, Autonomy Switch, Way-point Placement

4.2.5 Video / Cameras

1) Video Stream Displays

   - *Description:* The software should show three video displays in terms of a primary, secondary, and tertiary display. These displays will show streamed video feeds from cameras on the Rover and should be able to be individually disabled.
   - *Rationale:* The User will need video feeds from the Rover to navigate the competition course and to actuate the Rover arm.
   - *Dependencies:* Rover Connection Status, Camera Presence Status

2) Camera Source Selection

   - *Description:* For the primary, secondary, and tertiary video displays provide controls to switch which Rover camera is currently active.
   - *Rationale:* During competition, the User will frequently be changing the active cameras to best navigate the competition course and interact with objects.
   - *Dependencies:* Rover Connection Status, Camera Presence Status

3) Camera Quality Adjustments

   - *Description:* For the primary, secondary, and tertiary video displays provide controls to adjust the camera resolution and/or bit-rate.
   - *Rationale:* As the Rover gets farther away from the ground station, connection quality will diminish and require that the video stream qualities be lowered to ensure smooth Rover operation.
   - *Dependencies:* Video Stream Displays

4) Video Stream FPS Counters

   - *Description:* These indicators will show the quality of the video streams by use of FPS counters for all three video streams.
   - *Rationale:* The User would want to know what the video stream FPS values are to determine if camera quality needs to be adjusted.
   - *Dependencies:* Video Stream Displays, Rover Connection Status, Rover RTT

4.2.6 Miscellaneous

1) Offline Functionality

   - *Description:* The ground station software must function without a connection to the Internet during competition. Software elements that require the Internet such as maps must be able to be cached for use offline.

- *Rationale:* The competition is in a remote location with no access to an Internet source.
- *Dependencies:* None

2) Log File Viewing

- *Description:* From within the software, the User must be able to view verbose logs of useful data.
- *Rationale:* The User might want to access this stream to see any data that might help them troubleshoot problems such as faulty controls.
- *Dependencies:* Rover Connection Status

3) UI Dark Theme

- *Description:* The software must be visually ”dark” themed.
- *Rationale:* The dark theme will help ease User eye strain and was also a client request.
- *Dependencies:* None

4) Getting Started Document

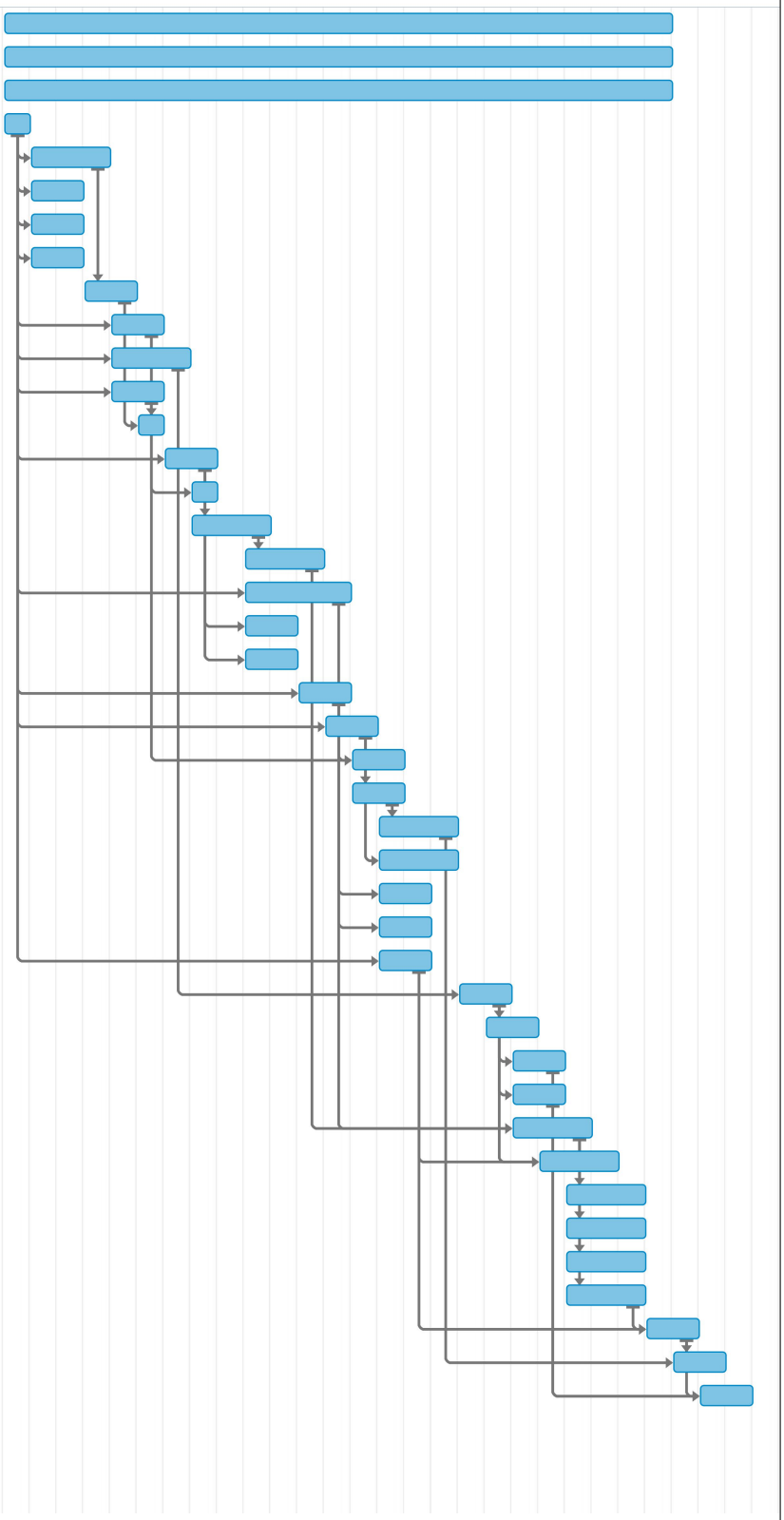
- *Description:* A 1-2 page document describing general useful details for future Rover teams on where to begin for reusing the software.
- *Rationale:* This was requested by the client.
- *Dependencies:* All Prior Requirements

5) Quick Deployment

- *Description:* This requirement is to package all items necessary for the Ground Station to function in a quickly deployable physical case that can be set-up and software started in less than two minutes during competition setup time.
- *Rationale:* Fast setup times will allow the User to spend extra time testing Rover communications during the setup phases of competition.
- *Dependencies:* All Prior Requirements

5 GANTT CHART

Mars Rover Ground Station	Sub Tasks	Task Prog.
0	Full-Screen Mode	0%
1	Dark UI Theme	0%
2	Two Monitors Minimum	0%
3	Rover Connection	0%
4	Arm Connection	0%
5	IMU Visualizer	0%
6	Rover Computer	0%
7	Display Data Collection Sensors	0%
8	Arm Joint Positions	0%
9	Joystick Connection	0%
10	Camera Presence	0%
11	Rover Battery	0%
12	Rover Joystick Arm	0%
13	GPS Status	0%
14	Rover Voltage	0%
15	GPS Satellites Connected	0%
16	GPS Accuracy	0%
17	Map Visualizer	0%
18	Rover Speed	0%
19	Rover Heading	0%
20	Bogie Connection	0%
21	Current Radio Signal Strength	0%
22	Rover Joystick Driving	0%
23	Rover RTT	0%
24	Rover Network Potential Throughput	0%
25	Estimated Rover Distance	0%
26	Rover Speed Limit	0%
27	Rover Speed Limit Control	0%
28	Autonomy Switch	0%
29	Video Stream Displays	0%
30	Camera Source Selection	0%
31	Camera Quality Adjustments	0%
32	Video Stream FPS Counter	0%
33	Way-point Placement	0%
34	Autonomy Streams Overlay	0%
35	Way-point Editing	0%
36	Way-point Re-Ordering	0%
37	Way-point Navigation Path	0%
38	Active Way-point Selection	0%
39	Autonomy Indicator	0%
40	Log File Viewing	0%
41	Getting Started Document	0%
42	3D Rover Arm Visualization	0%
43	Quick Deploy Case	0%
44	Automatic Video Stream Quality Adjustment	0%



6 STRETCH GOALS

1) Autonomy Streams / Overlay

   - *Description:* This would display a specialty video stream either independently or via an overlay showing the course obstacles and/or features the Rover is detecting during autonomous mode navigation. This would depend on the computational overhead remaining on the Rover processing computer.
   - *Rationale:* This would help the team judge autonomous navigational performance of the Rover.
   - *Dependencies:* Rover Connection Status, Autonomy Switch

2) 3D Rover Arm Visualization

   - *Description:* Represent the arm joint positions using a 3D CAD model of the arm.
   - *Rationale:* A 3D representation of the arm and its positioning will make it easier for the user to precisely use the arm to manipulate objects.
   - *Dependencies:* Rover Connection Status, Arm Connection Status

3) Automatic Video Steam Quality Adjustment

   - *Description:* The software would automatically detect poor video quality through use of low FPS counts and/or high network latency and use this information to adjust video resolution and bit-rates to increase video stream FPS to reasonable levels.
   - *Rationale:* Automatic control of video stream quality would help the User avoid spending time adjusting values during valuable competition time.
   - *Dependencies:* Video Stream Displays, Camera Quality Adjustments, Video Streams FPS Counters

2.2 Additions

2.2.1 Final Revision Document

Oregon State University
College of Engineering

OSURC
OSU Robotics Club

CS Capstone Requirements Document

April 26, 2018

OSU Robotics Club
Mars Rover Ground Station

Revision 1.1

Changed requirements to match competition change and slow hardware development from Rover team

Signatures

4/26/18

4/26/18

4/26/18

4/26/18

Prepared for

OSU Robotics Club

Nick McComb

Prepared by

Group 30

Ground Station Software Team

Kenneth Steinfeldt
Christopher Pham
Corwin Perren

Abstract

This document covers the design requirements for the OSU Robotics Club Mars Rover Team's Ground Station Software. It begins with details about why the software is necessary, and a general overview of what it needs to accomplish. We then go into further detail about what the specific functional requirements will be, how they relate to each other in terms of dependencies, and then end with stretch goals if there is extra time to complete them.

CONTENTS

1 INTRODUCTION

1.1 Purpose

The purpose of this document is to formally define the characteristics of the software to be implemented. These requirements can then be used to determine if the software is complete once the project is done, while also serving as a reference during development.

1.2 Scope

We will be creating the ”Mars Rover Ground Station” software package. This software will allow Rover team members to interact with the Oregon State University Robotics Club’s Mars Rover for the purpose of a competition taking place in Drumheller, Alberta, Canada during August of 2018. The software will provide Rover systems monitoring as well as control of drive and arm systems. During competition, the software will be run on a remote base station primarily operated off the back of a truck, allowing users to tele-operate the Rover during manned controlled missions, or to monitor progress during autonomous missions. Properly functioning and fully featured ground station software will be a major factor in the success of the Mars Rover during this competition.

1.3 Definitions / Acronyms / Abbreviations

- ROS - Robot Operating System
- QT - A multi-platform GUI and application framework
- GPS - Global Positioning Satellite
- User - Person operating rover from ground control station
- GUI - Graphical User Interface
- RTT - Round-trip Time, the length of time for a signal to be sent plus the length of the time it takes for an acknowledgement of the signal to be received.
- Bogie - Groups of two individual wheels on the Rover, each with the ability to be driven independently.
- IMU - Inertial Measurement Unit
- CAD - Computer Aided Design

1.4 Overview

The remainder of this document consists of 4 sections and will describe the requirements that must be met in order for this project to be considered fully complete. The next section gives a description of the process. The fourth section describes the ground control software’s functions, user characteristics, constraints, and assumptions and dependencies. The fifth section describes the specific requirements that must be met for the software to be a success. Finally, stretch goals in the sixth section will describe hopeful, but not mandatory goals for the software.

2 DESCRIPTION

2.1 Product Perspective

This project interacts with and requires the Robotics Club’s Mars Rover robotics vehicle in order to be useful. If the Rover is not connected, the software will still be able to launch, but will not perform any useful functions until it has established a connection to the Rover. In order to accomplish interaction with the rover, the ground station software will use ROS for primary functionality and feedback from the rover.

3 PRODUCT FUNCTIONS

The ground control software will primarily accomplish two tasks:

- Provide Rover control to the user. Direct rover control will be accomplished with joystick(s) and/or a SpaceNav mouse along with a keyboard and mouse.
- Enable and present rover feedback to the user. The rover will send health and status information to the ground control station.

3.1 User Characteristics

Users of the Rover Ground Station software will comprise of Rover team leads and members.

3.1.1 User Stories

- User should be able view ground station UI on two HD monitors so that valuable competition time will not be spent switching between screens on a single monitor
- User should be able to add way-points because way-points are needed to find useful points in the competition field and are necessary for autonomous navigation
- User should be able to remove way-points because way-points may become unneeded or need to be reset for a new competition event
- User should be able to edit way-points because there's a chance that the User may enter way-point details incorrectly and will need to change them
- User should be able to change the order of way-points because we might need to traverse back to a location, or fix improperly entered way-points
- User should be able to control the Rover by joystick(s) because that is the preferred way of controlling the Rover
- User should be able to select video streams because the Rover has more cameras than can be streamed at any one time, and being able to view any camera on demand will help the User drive the Rover and manipulate the arm
- User should be able to view the pitch, yaw, roll of the Rover because we want to know if the rover is on an incline, decline, or about to tip over
- User should be able to switch on and off autonomy because there are competition events that require autonomy and others that require manual control
- User should be able to tell that autonomy is enabled because that alerts the User that they can no longer manually control the Rover
- User should be able to view how the arm joints positions as that can show us if the arm is performing correctly
- User should be able to control the arm because competition events require the use of a Rover arm
- User should be able to set the Rover speed limit so that the User can more easily drive the Rover over difficult terrain
- User should be able to view the Rover CPU usage because the User can use this information to tell if there is a software problem on the Rover or if the Rover is trying to process too much information
- User should be able to view the Rover memory usage because the User does not want the Rover to crash or slow down to unusable levels due to running out of memory
- User should be able to view the Rover drive usage because the User does not want the Rover to crash if the drive is full

- User should be able to see the speed of the Rover because it can be used to determine if the Rover is driving at a speed that is reasonable for the current terrain
- User should be able to see how accurate the GPS positioning is because at the competition there is a requirement to be a certain distance from an object
- User should be able to see what heading the Rover is at because it allows for the User to know what direction to go
- User should be able to see the maximum throughput of the network because it will allow for video stream quality to go up or down depending on the connection quality
- User should be able to confirm connected devices because this will allow for debugging and checking of other connected devices

3.2 Constraints

Due to the nature of the Mars Rover project, there are multiple hardware and software constraints to ensure that the Ground Station software can properly communicate with the Rover. Additionally, the following constraints will help ensure future re-usability of the finalized software, which is a key goal of the Mars Rover team.

The software language allowed is the primary constraint for this project. Python 2.7 is required by the Mars Rover team as it maintains consistency with the software written on the Rover itself. The team also requires Python as it more easily allows future team members to reverse engineer code to be reused in future projects.

Use of the QT application framework is another constraint placed by the Rover team in order to facilitate rapid prototyping of the user interface for this project. It will also allow for faster and easier modification both by future members of the team, and during competition as that has previously been necessary.

The Rover internally uses ROS to handle routing and interpreting control and status information as well as video data. To be able to view this video data, status information, and to be able to send the Rover control information, this ground station software must also incorporate ROS.

At a hardware level, the Rover team will be providing an Intel NUC desktop computer running Ubuntu 16.04 that the Ground Station software will need to be able to run on. They will also provide two HD monitors, one or two USB joysticks, a SpaceNav mouse, as well as a keyboard and mouse that will be used to view and interact with the software.

As the Rover project is highly volatile where designs change frequently and hardware development often falls behind, any systems that cannot be explicitly implemented must at least have placeholders in both the GUI frontend and code backend to make eventual development easier when the hardware becomes available.

4 Specific Requirements

4.1 User Interfaces

Upon initial launch, the application should be in full-screen mode across both of the monitors. The left-hand screen will show navigation, Rover status, and miscellaneous Rover controls. The right hand monitor will show the three live video streams from the Rover. In the case that the software is started without a Rover to connect to, the software will display as-such and show placeholder information until such a connection is made.

4.2 Functional Requirements

4.2.1 Statuses / Informational Readouts

1) Rover Connection Status

   - *Description:* This indicator will show a binary state of whether or not the Rover is currently connected.
   - *Rationale:* The Rover must be connected in order for most of the software aspects to function. It is also useful to tell when full network dropouts happen.
   - *Dependencies:* None

2) Joystick(s) Connection Status

   - *Description:* Status indicators will visually show binary states as to whether joystick(s) are currently connected.
   - *Rationale:* Joystick(s) must be connected to drive the Rover.
   - *Dependencies:* Rover Connection Status

3) SpaceNav Mouse Connection Status

   - *Description:* Status indicators will visually show binary states as to whether the SpaceNav mouse is connected.
   - *Rationale:* SpaceNav mouse must be connected to control the Rover arm.
   - *Dependencies:* Rover Connection Status

4) Rover Battery Voltage

   - *Description:* This indicator will show the Rover's battery voltage.
   - *Rationale:* The User should know what the battery voltage is to be able to determine remaining runtime and to diagnose other Rover errors.
   - *Dependencies:* Rover Connection Status

5) Wheel Connection Statuses

   - *Description:* These indicators will show whether each of the six wheels are connected to the Rover.
   - *Rationale:* This will let drivers of the Rover know if a wheel has failed, which is a common occurrence.
   - *Dependencies:* Rover Connection Status

6) Arm Connection Status

   - *Description:* This indicator will be a binary state of weather or not the mechanical arm is connected or not.
   - *Rationale:* This will show the User whether the arm is connected, and therefore whether they will be able to use the joysticks to control the arm.
   - *Dependencies:* Rover Connection Status

7) Arm Joint Positions

   - *Description:* These indicators will show Rover Arm joint positions in terms of degrees or via visual relationships.
   - *Rationale:* These indications will help ensure the User knows where the Arm actually is, and whether it is being moved within its operating limits.

- *Dependencies:* Rover Connection Status, Arm Connection Status

8) Camera Presence Statuses

- *Description:* This indicator will show which cameras on-board the Rover and connected and functional.
- *Rationale:* The User can use such indicators to verify a failed camera for easy troubleshooting.
- *Dependencies:* Rover Connection Status

9) IMU Status

- *Description:* This will indicate the pitch, yaw, and roll of the Rover.
- *Rationale:* This can be used by the User to help navigate uneven terrain and to avoid flipping the Rover.
- *Dependencies:* Rover Connection Status

10) Map Visualizer

- *Description:* This will display a map of the current Rover location and surrounding areas.
- *Rationale:* This can be used to see terrain maps of the area and to place useful way-point and autonomous navigation markers.
- *Dependencies:* Rover Connection Status

11) Rover Computer Statuses

- *Description:* These indicators will show the current CPU, memory, and disk usage of the Rover's on-board computer.
- *Rationale:* These can be used to ensure the Rover's processing computer is not overloaded. It can also show potential software failures while serving as a useful indication that the connection to the Rover is stable.
- *Dependencies:* Rover Connection Status

12) Current Radio Signal Strength

- *Description:* This indicator will display the strength of the connection between the Rover and ground station.
- *Rationale:* The User can use this information to determine if the Rover is driving out of range to ensure a connection is not fully lost.
- *Dependencies:* Rover Connection Status

13) Rover Network Potential Throughput

- *Description:* This indicator will show the maximum theoretical throughput in Kbps between the Rover and this software as provided by the Ubiquity routers.
- *Rationale:* The User can use the information to help tune video bit-rates and resolutions to ensure the connection is not overloaded.
- *Dependencies:* Rover Connection Status, Current Radio Signal Strength, Rover RTT

14) Estimated Rover Distance

- *Description:* This indicator will show the distance from the Ground Station radio to the Rover radio, as calculated by the Ubiquiti radio systems.

- *Rationale:* The User can use this information to help ensure they do not drive out of communications range.
- *Dependencies:* Rover Connection Status, Current Radio Signal Strength

15) Rover Speed Limit

- *Description:* This display will show the Rover's speed limit in terms of 0 to 100 percent.
- *Rationale:* During manual driving operations it is often useful to limit the max speed of the Rover. A visual indication of this limit will help avoid confusion if the User forgets that a limit has been set, or if the limit is accidentally changed.
- *Dependencies:* Rover Connection Status, Rover Bogie Status

16) Display Data Collection Sensors

- *Description:* Provide data displays for any scientific sensors attached to the Rover.
- *Rationale:* During some competition phases, scientific sensors will be placed into the soil to gather scientific data. This data needs to be sent back to the Ground Station so it can be recorded.
- *Dependencies:* Rover Connection Status

4.2.2 Rover Control

1) Rover Joystick Driving Control

- *Description:* When both joysticks are connected and the Rover is also connected, the joysticks can be used to drive the Rover.
- *Rationale:* The joystick control is needed to be able to drive the Rover.
- *Dependencies:* Rover Connection Status, Joystick Connection Statuses, Bogie Connection Statuses

2) Rover SpaceNav Mouse Arm Control

- *Description:* When the SpaceNav mouse is connected, the Rover is connected, and the Arm is connected, this mouse can be used to manipulate the Rover arm.
- *Rationale:* During some competition events, the arm will need to be manipulated by the User to complete competition tasks.
- *Dependencies:* Rover Connection Status, Joystick Connection Statuses, Arm Connection Status

3) Rover Speed Limit Control

- *Description:* When both joysticks are connected and the Rover is also connected, buttons or levers on a joystick may be used to adjust the max speed limit of the Rover.
- *Rationale:* This quick adjustment is often needed to allow for finer control of the Rover over difficult terrain or when manipulating the Rover arm.
- *Dependencies:* Rover Connection Status, Joystick Connection Statuses, Bogie Connection Statuses

4.2.3 Navigation

1) GPS Status

- *Description:* This will show whether the GPS on the Rover is connected and whether it has GPS lock.

- *Rationale:* This is needed to check to see if GPS is working and functioning properly.
- *Dependencies:* Rover Connection Status

2) Rover Speed

- *Description:* This indicator shows how fast the Rover is moving in meters per second via the GPS.
- *Rationale:* This will help the User ensure they are driving within the mechanical limits of the Rover.
- *Dependencies:* Rover Connection Status, GPS Status

3) Rover GPS Heading

- *Description:* This indicator will show what bearing the Rover is heading.
- *Rationale:* The User can use this information to help navigate the competition course.
- *Dependencies:* Rover Connection Status, GPS Status

4) GPS Satellites Connected

- *Description:* This indicator will show how many GPS satellites are currently active.
- *Rationale:* This can help tell the User if the Rover is in a location that is bad for GPS reception, allowing them to drive to a better location before the GPS lock is lost.
- *Dependencies:* Rover Connection Status, GPS Status

5) GPS Accuracy

- *Description:* This indicator will how accurate the current GPS location is in meters.
- *Rationale:* This will allow the User to judge how much the GPS location of the Rover can be trusted, as well as giving useful information about how well the Rover might perform during the autonomous challenge.
- *Dependencies:* Rover Connection Status, GPS Status, GPS Satellites Connected

6) Way-point Placement

- *Description:* Give the ability to place way-points on a map by GPS location, by current Rover location, or via manual GPS co-ordinate entry and add this way-point to a queue. These way-points will also need to show up visually on the map.
- *Rationale:* This will used to map out the course during the autonomous challenge as well as to place points of interest during manual driving events.
- *Dependencies:* Rover Connection Status, GPS Status, GPS Satellites Connected, Map Visualizer

7) Way-point Editing

- *Description:* Allow for editing any way-point in the queue by drag and drop of the way-point, or via manual editing of the GPS co-ordinates.
- *Rationale:* This will allow for easy adjustment of way-points if they were placed incorrectly.
- *Dependencies:* Rover Connection Status, GPS Status, GPS Satellites Connected, Way-point Placement, Map Visualizer

8) Way-point Re-Ordering

- *Description:* Allow for changing the order of way-points.

- *Rationale:* This will allow the User to skip unneeded way-points or fix incorrectly ordered way-points.
- *Dependencies:* Rover Connection Status, GPS Status, GPS Satellites Connected, Way-point Placement, Map Visualizer

9) Active Way-point Selection

- *Description:* Allow for the selection of a way-point from the queue as currently active. Setting the way-point active will update a ”Heading To” indicator to help a User drive to the desired way-point.
- *Rationale:* This is useful to make navigating to competition provided co-ordinates easier.
- *Dependencies:* Rover Connection Status, GPS Status, GPS Satellites Connected, Way-point Placement, Map Visualizer

4.2.4 Autonomy

1) Autonomy Switch

- *Description:* A toggle to enable and disable the Rover’s autonomous mode operation.
- *Rationale:* This is necessary to place the Rover into autonomous mode for that portion of the University Rover Challenge competition.
- *Dependencies:* Rover Connection Status

2) Autonomy Indicator

- *Description:* An alert or indicator for when the Rover is in autonomous mode.
- *Rationale:* This indicator will make it easy to tell whether the joystick(s) can presently be used to drive the Rover.
- *Dependencies:* Rover Connection Status, Autonomy Switch, Way-point Placement

4.2.5 Video / Cameras

1) Video Stream Displays

- *Description:* The software should show three video displays in terms of a primary, secondary, and tertiary display. These displays will show streamed video feeds from cameras on the Rover and should be able to be individually disabled.
- *Rationale:* The User will need video feeds from the Rover to navigate the competition course and to actuate the Rover arm.
- *Dependencies:* Rover Connection Status, Camera Presence Status

2) Camera Source Selection

- *Description:* For the primary, secondary, and tertiary video displays provide controls to switch which Rover camera is currently active.
- *Rationale:* During competition, the User will frequently be changing the active cameras to best navigate the competition course and interact with objects.
- *Dependencies:* Rover Connection Status, Camera Presence Status

3) Camera Quality Adjustments

- *Description:* For the primary, secondary, and tertiary video displays provide controls to adjust the camera resolution and/or bit-rate.
- *Rationale:* As the Rover gets farther away from the ground station, connection quality will diminish and require that the video stream qualities be lowered to ensure smooth Rover operation.
- *Dependencies:* Video Stream Displays

4.2.6 Miscellaneous

1) Offline Functionality

   - *Description:* The ground station software must function without a connection to the Internet during competition. Software elements that require the Internet such as maps must be able to be cached for use offline.
   - *Rationale:* The competition is in a remote location with no access to an Internet source.
   - *Dependencies:* None

2) UI Dark Theme

   - *Description:* The software must be visually "dark" themed.
   - *Rationale:* The dark theme will help ease User eye strain and was also a client request.
   - *Dependencies:* None

3) Getting Started Document

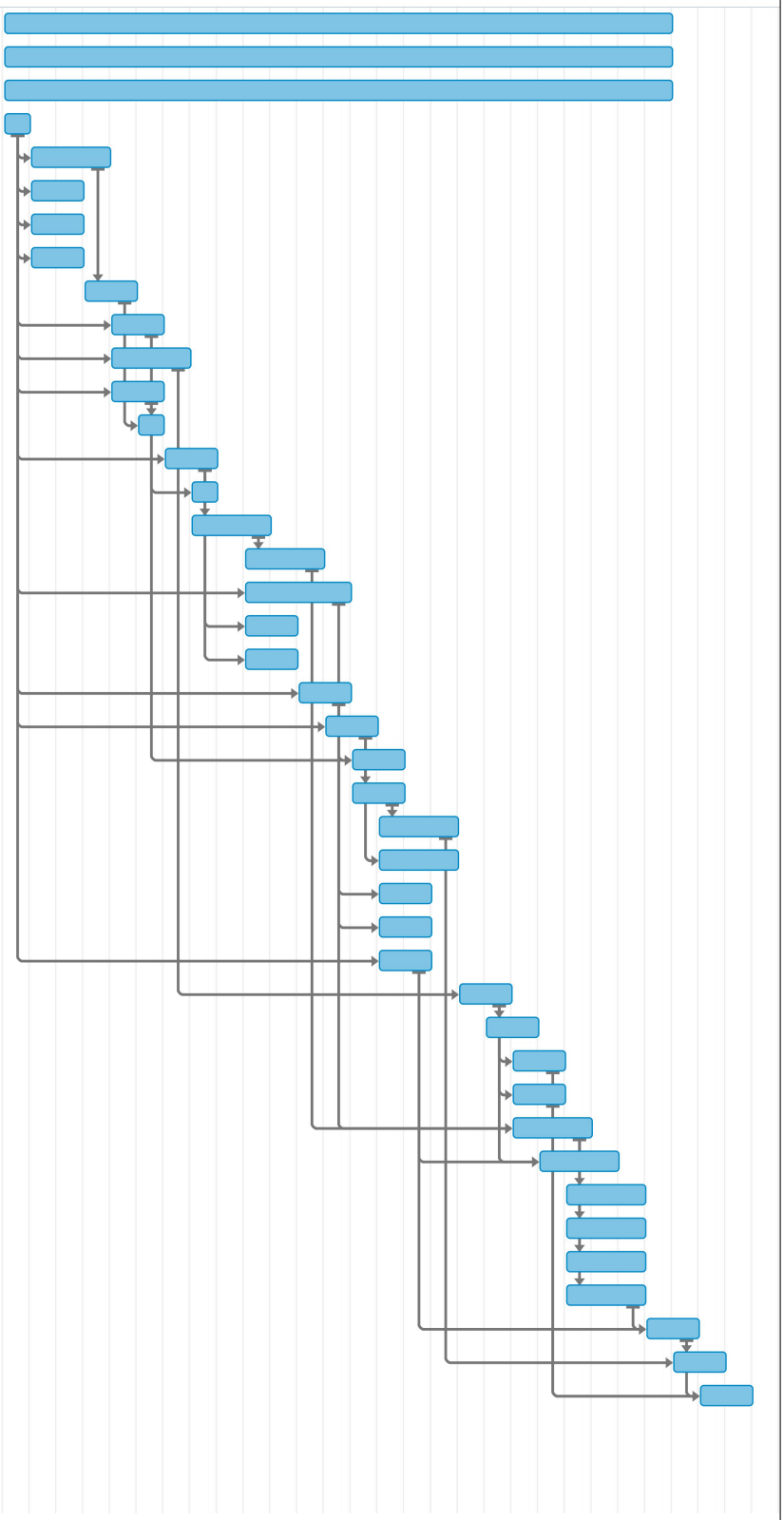
   - *Description:* A 1-2 page document describing general useful details for future Rover teams on where to begin for reusing the software.
   - *Rationale:* This was requested by the client.
   - *Dependencies:* All Prior Requirements

4) Quick Deployment

   - *Description:* This requirement is to package all items necessary for the Ground Station to function in a quickly deployable physical case that can be set-up and software started in less than two minutes during competition setup time.
   - *Rationale:* Fast setup times will allow the User to spend extra time testing Rover communications during the setup phases of competition.
   - *Dependencies:* All Prior Requirements

5 GANTT CHART

Mars Rover Ground Station	Sub Tasks	Task Prog.
0	Full-Screen Mode	0%
1	Dark UI Theme	0%
2	Two Monitors Minimum	0%
3	Rover Connection	0%
4	Arm Connection	0%
5	IMU Visualizer	0%
6	Rover Computer	0%
7	Display Data Collection Sensors	0%
8	Arm Joint Positions	0%
9	Joystick Connection	0%
10	Camera Presence	0%
11	Rover Battery	0%
12	Rover Joystick Arm	0%
13	GPS Status	0%
14	Rover Voltage	0%
15	GPS Satellites Connected	0%
16	GPS Accuracy	0%
17	Map Visualizer	0%
18	Rover Speed	0%
19	Rover Heading	0%
20	Bogie Connection	0%
21	Current Radio Signal Strength	0%
22	Rover Joystick Driving	0%
23	Rover RTT	0%
24	Rover Network Potential Throughput	0%
25	Estimated Rover Distance	0%
26	Rover Speed Limit	0%
27	Rover Speed Limit Control	0%
28	Autonomy Switch	0%
29	Video Stream Displays	0%
30	Camera Source Selection	0%
31	Camera Quality Adjustments	0%
32	Video Stream FPS Counter	0%
33	Way-point Placement	0%
34	Autonomy Streams Overlay	0%
35	Way-point Editing	0%
36	Way-point Re-Ordering	0%
37	Way-point Navigation Path	0%
38	Active Way-point Selection	0%
39	Autonomy Indicator	0%
40	Log File Viewing	0%
41	Getting Started Document	0%
42	3D Rover Arm Visualization	0%
43	Quick Deploy Case	0%
44	Automatic Video Stream Quality Adjustment	0%



6 STRETCH GOALS

1) Autonomy Streams / Overlay

   - *Description:* This would display a specialty video stream either independently or via an overlay showing the course obstacles and/or features the Rover is detecting during autonomous mode navigation. This would depend on the computational overhead remaining on the Rover processing computer.
   - *Rationale:* This would help the team judge autonomous navigational performance of the Rover.
   - *Dependencies:* Rover Connection Status, Autonomy Switch

2) 3D Rover Arm Visualization

   - *Description:* Represent the arm joint positions using a 3D CAD model of the arm.
   - *Rationale:* A 3D representation of the arm and its positioning will make it easier for the user to precisely use the arm to manipulate objects.
   - *Dependencies:* Rover Connection Status, Arm Connection Status

3) Automatic Video Steam Quality Adjustment

   - *Description:* The software would automatically detect poor video quality through use of low FPS counts and/or high network latency and use this information to adjust video resolution and bit-rates to increase video stream FPS to reasonable levels.
   - *Rationale:* Automatic control of video stream quality would help the User avoid spending time adjusting values during valuable competition time.
   - *Dependencies:* Video Stream Displays, Camera Quality Adjustments, Video Streams FPS Counters

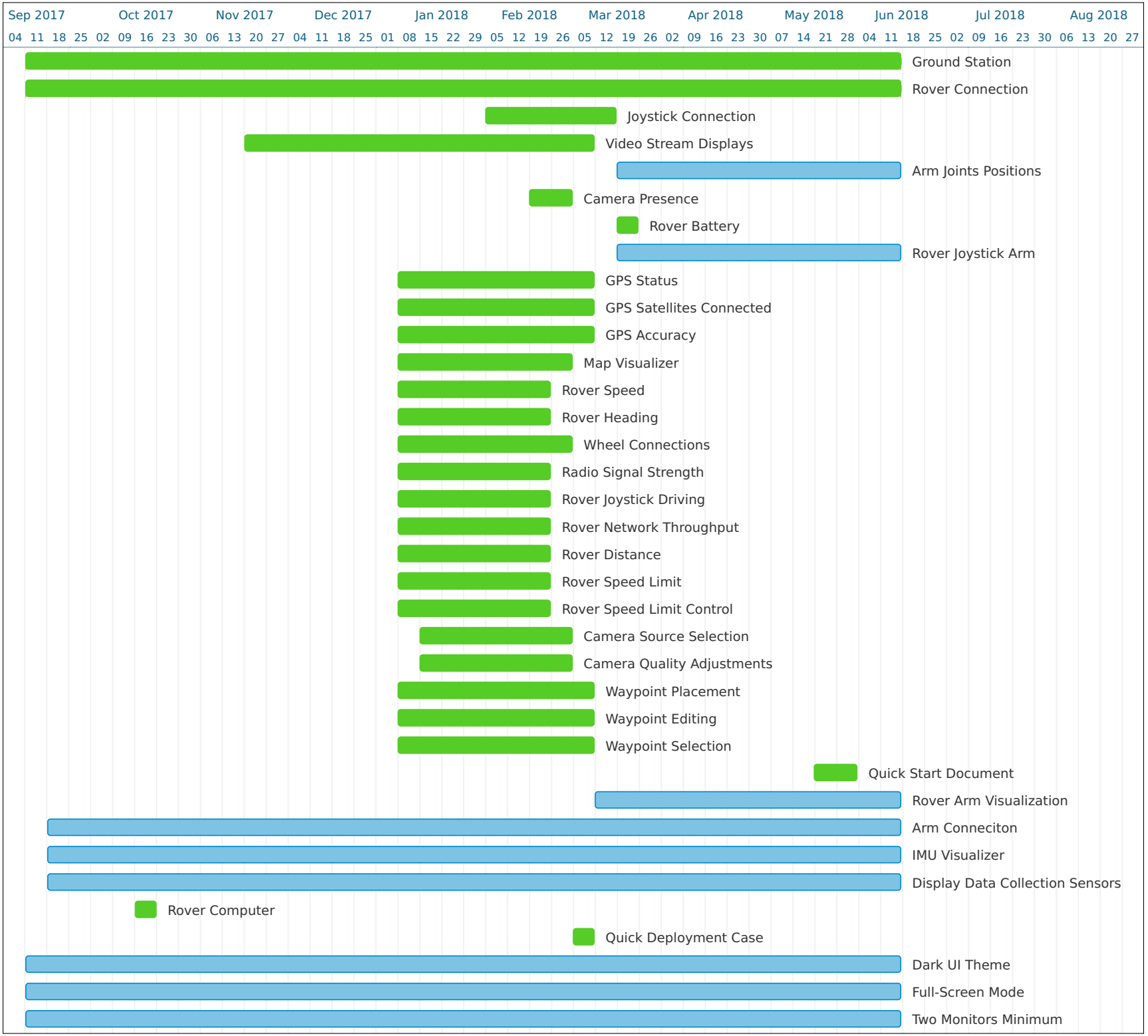
2.2.2 Description of Changes

Version 1.1 of document was approved by all team members and stakeholder on 4/26/2018.

- Edited descriptions of competition from University Rover Challenge to Canadian International Rover Challenge
  - The Mars Rover team changed competitions, so we updated the document to reflect this
- Removed user story for manual video quality adjustment
  - Our team implemented automatic quality adjustment, so manual adjustment was no longer necessary
- Removed user story about easily seeing when an autonomous run has completed
  - The new Rover competition does not require this display element
- Removed user story about needing to view live logs
  - Log viewing was determined to not be needed as there would be too many logs to sort through in a short amount of time
- Removed user story about seeing network latency (round trip time)
  - The network connection percentage was determined to be correlated with this value enough that this redundant element was not needed
- Added constraint that the GUI software must have placeholder for GUI elements and software features that can not be completed due to waiting on Rover team progress
  - Our team encountered many situations where we could not continue development due to lack of hardware/software on Rover
  - This new constraint allows us to still meet requirements if the bottleneck is the Rover team
- Changed functional requirement for joystick statuses so that the SpaceNav mouse had its own requirement
  - The team changed to a SpaceNav mouse earlier in the year and it was determined that an individual readout for this input device would be convenient
- Changed functional requirement for battery status to battery voltage
  - The battery status value was determined to be most useful as a voltage, rather than a percentage estimate
- Removed functional requirement for Rover voltage statuses
  - The Rover design changed so that only the battery voltage is measured
- Changed functional requirement for bogie connection statuses to be individual wheel statuses
  - The Rover software team got individual status information for each wheel, rather than a two wheel bogie group, and requested this change
- Removed functional requirement for Rover network round trip time
  - The network connection percentage was determined to be correlated with this value enough that this redundant element was not needed

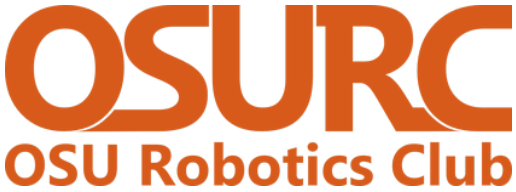
- Changed functional requirement for Rover arm joystick control to SpaceNav mouse control
  - The team changed to a SpaceNav mouse over a second joystick for arm control
- Removed functional requirement for way-point navigation path
  - Expectations for how autonomy on the Rover would work would not easily give our team this information to display
- Changed functional requirement for the autonomy indicator so that it shows whether autonomy is active rather than when autonomy is finished
  - Changing to the Canadian International Rover Challenge made the old requirement unnecessary
- Removed functional requirement for video FPS counters
  - FPS counters would have been useful for manual video quality adjustment, but are no longer needed with automatic quality adjustment
- Removed functional requirement for log file viewing
  - Log files were too large to usefully sort through in a short amount of time

2.3 Final Gantt Chart


Sep 2017
Oct 2017
Nov 2017
Dec 2017
Jan 2018
Feb 2018
Mar 2018
Apr 2018
May 2018
Jun 2018
Jul 2018
Aug 2018
04 11 18 25 02 09 16 23 30 06 13 20 27 04 11 18 25 01 08 15 22 29 05 12 19 26 05 12 19 26 02 09 16 23 30 07 14 21 28 04 11 18 25 02 09 16 23 30 06 13 20 27
Ground Station
Rover Connection
Joystick Connection
Video Stream Displays
Arm Joints Positions
Camera Presence
Rover Battery
Rover Joystick Arm
GPS Status
GPS Satellites Connected
GPS Accuracy
Map Visualizer
Rover Speed
Rover Heading
Wheel Connections
Radio Signal Strength
Rover Joystick Driving
Rover Network Throughput
Rover Distance
Rover Speed Limit
Rover Speed Limit Control
Camera Source Selection
Camera Quality Adjustments
Waypoint Placement
Waypoint Editing
Waypoint Selection
Quick Start Document
Rover Arm Visualization
Arm Conneciton
IMU Visualizer
Display Data Collection Sensors
Rover Computer
Quick Deployment Case
Dark UI Theme
Full-Screen Mode
Two Monitors Minimum

3 DESIGN DOCUMENT

3.1 Original Document



CS CAPSTONE DESIGN DOCUMENT

JANUARY 25, 2018

OSU ROBOTICS CLUB
MARS ROVER GROUND STATION

PREPARED FOR

OSU ROBOTICS CLUB

NICK MCCOMB

PREPARED BY

GROUP 30

GROUND STATION SOFTWARE TEAM

KENNETH STEINFELDT
CHRISTOPHER PHAM
CORWIN PERREN

Abstract

This document describes the design of the Mars Rover Ground Control software for the purpose of conveying design decisions and information to the project's stakeholders. The Software Design Description (SDD) supplies a clear picture of the software's future implementation and provides the development group with a clear road-map.

CONTENTS

1 INTRODUCTION

1.1 Purpose

The purpose of this software design document is to cover the design for the Oregon State University Mars Rover Team's ground station software. It will cover the details about how the software will function, how we will implement that functionality, and the reasoning behind design decision choices that were made. The ground station software for the Mars Rover project is the single point of contact between the team-built competition Rover and the users operating or viewing the rover.

1.2 Scope

This project will consist of a software package running from a remote control base station in order to remotely operate a competition-ready robot built by the Oregon State University Robotics Club's Mars Rover team. The project must be completed by May 31st, 2018 in order to be ready for the University Rover Challenge taking place in Hanksville, Utah. The controlling software must be able to do at minimum, the following:

- Provide the capability to remotely drive the Rover via joystick(s) connected to the ground station computer.
- Allow for viewing of up to three video feeds from the Rover, and be able to change what video streams are viewed.
- Via a user input device, allow for manipulation of the Rover arm.
- Provide visual feedback about the state of the joint positions of the arm as it is being moved.
- Show the user a map of the competition area, allowing the user to zoom, view the Rover's location, and place navigation and landmark way-points.
- Allow the user to add way-points and then place the Rover in autonomous mode, wherein the Rover will drive under its own control along the way-points provided.
- Upon completion of an autonomous path, provide an easy to read notification that the navigation is complete.
- Provide a myriad of status information about the Rover's current condition including, but not limited to, the following:
  - Connection statuses
  - Rover component statuses
  - On-board sensor readings
  - Battery charge level
  - System errors

Additionally, the client has requested a document be written providing a starting guide on how to re-use the finalized software package for future Rover teams to ease development in future years.

1.3 Context

Each year the Mars Rover software team writes ground station software from scratch in order to meet the changing requirements of both the Rover itself and competition rule changes. As this software is the main point of contact between the users and the Rover, the success of the team during competition often hinges on how well this software performs. In the past, lack of modularity and abstraction has made re-use of the ground station code near impossible.

1.4 Summary

This rest of this document will provide the details about how our team will implement the ground station software for the Mars Rover team. We will start by covering design considerations such as the environment our software will be running in, as well as the specific concerns of our stakeholder. Immediately following will be a breakdown of the individual software components we will be writing. These will include descriptions of what, how, and why we will be writing each component the way we are. The document ends with a visual breakdown of the software UI, and rationale for why each component looks the way it does.

2 References

[1] University rover challenge rules 2018. [Online]. Available: http://urc.marssociety.org/files/University%20Rover%20Challenge%20Rules%202018.pdf

3 Glossary

3.1 Acronyms and Abbreviations

- ROS - Robot Operating System
- GPS - Global Positioning Satellite
- OSU - Oregon State University
- OSURC - Oregon State University Robotics Club
- NASA - The National Aeronautics and Space Administration

3.2 Definitions

- Ubuntu - A popular open source Linux distribution based on the classic Debian distribution.
- Rover - A robotic, remote controlled vehicle designed and built by OSURC to compete in the NASA Mars Rover Competition held in Hanksville, Utah.
- QT - An open source application framework developed by the QT Company.
- Python - A popular scripting language known for it's easy readability and rapid deployment abilities.

4 Design Considerations

4.1 Assumptions

From the descriptions and restrictions provided by the client, there are a few things that we need to assume to assure that the software will function correctly. Our team is going to assume that the software/OS on the rover is going to be a combination of ROS Kinetic on Ubuntu 16.04. The rover will have all of its components working correctly such as GPS and drive systems. The rover needs to correctly interact with the data that is provided by the station. The rover needs to be connected to the same network as the rover.

4.2 General Constraints

- *Time Frame:* The software must be completed on or before May 31st, 2018.
- *URC Requirements:* The software must not violate any rules provided by the competition host. [1]

4.3 System Environment

- Hardware
  - 1 x Intel NUC
  - 2 x 24″ 1080p Monitors
  - 1 x Keyboard
  - 1 x Mouse
  - 1-2 x USB Joystick
  - 1 x SpaceMouse Pro
  - 1 x Ubiquiti Rocket M2 Wireless Router
- Software / OS / Libraries
  - Ubuntu 16.04 LTS
  - ROS Kinetic
  - Python 2.7
  - Additional Python Libraries
    - PyQt5
    - inputs
    - PyGame
    - spnav
    - cv2
    - Pillow
    - qimage2ndarray

4.4 Stakeholder Concerns

4.4.1 Reliability

The ground control software must be as reliable as possible. The rover operator must be able to complete all competition tasks, assuming all hardware on the Rover is functioning properly. If unable to do so, the software cannot be considered a success.

4.4.2 Robustness

The University Rover Challenge requires that the rover operate successfully in a variety of environments. For example, it is expected that as the rover advances through the course, range and obstacles will negatively effect the latency and bandwidth of the wireless Ethernet connection used to communicate with the rover. In the above example, the rover must be able to downgrade the video feed in order to accommodate the lowered communication abilities of the rover.

4.4.3 Rapid Prototyping

In order to fulfill the necessary reliability and robustness requirements of the client, the team must be able to rapidly prototype the software. Rapid prototyping allows the team to maximize testing time.

4.4.4 Documentation

The client has requested that documentation be a primary concern of the project. The ground control software will be used as a foundation for future rover competitions, therefore it is important that the software is easy to use and the code is easy to understand.

5 COMPONENT DESIGN

5.1 Human Interface Device Integration

5.1.1 Overview

During use of this ground station software, the user will need to be able to interact with a joystick(s) and SpaceNav mouse to control ground station software elements, to drive the Rover, and to manipulate the Rover arm. The systems that integrate with these HID devices will be some of the most active parts of the ground station software.

5.1.2 Design Concerns

- *Control Latency:* Latency for these control inputs must be kept to a minimum to ensure that the Rover responds quickly to control commands and that motion is fluid.
- *Reliability:* As these control inputs can directly control the Rover, the code must be reliable, robust, and be able to recover from errors such as a disconnect and reconnect of a joystick.
- *Flexibility:* It is hard to determine the best possible control scheme for the Rover via these control inputs until the team has had a chance to physically drive the robot, so having the flexibility to easily change the control structure is important. Additionally, these input devices may change completely if they are determined to be inadequate in real-world tests later on.

5.1.3 Design Elements

- The HID integrators will use the `inputs` or `pygame` libraries for joysticks or the `spnav` library for the SpaceNav mouse.
- Each integrator will be housed within a QThread class that will monitor the state of the HID devices and poll the devices for changes.
- Upon a device change, the class will broadcast the updates using QSignals so that other parts of the program may use the data.
- Upon an HID failure, such as on accidental disconnect, the class will attempt to reconnect and broadcast an error state until it is resolved.

5.1.4 Design Rationale

One of the main goals of this project is to be able to rapidly prototype the software so the Rover team can spend more time testing, and less time waiting for code to be written. By using off the shelf libraries for reading in joystick and SpaceNav mouse input, our team's development time can be better put to use making the control systems robust and handling error cases. As these are commonly used and tested libraries, there is also a high likelihood that their implementations are more reliable and documented than if our design team were to try and implement equivalent libraries/classes ourselves. The use of QT's QSignals will also help us in this regard by minimizing the design time needed to write custom inter-thread communication protocols.

5.2 Drive Coordinator

5.2.1 Overview

This sub-system will handle taking in raw joystick(s) control information, and transforming them into usable drive control commands. This will also handle the interpretation of button presses on the joystick to control GUI functions, or for example, artificially limiting the Rover max speed via the current state of the joystick throttle lever.

5.2.2 Design Concerns

- *Reliability:* This node must be incredibly reliable. If the software runs off and sends continuous drive commands, for example, the physical Rover will be driving out of control.
- *Speed:* As this node is what is sending drive commands, any major processing delays here will make driving the Rover a choppy, unresponsive, and unpleasant experience.
- *Pause State Responsiveness:* When the pause button is pressed, the Rover will need to stop all movement quickly.

5.2.3 Design Elements

- The coordinator will take in joystick control information via QSignals.
- The coordinator will send Rover drive commands via a ROS topic using `rospy`.
- The coordinator will artificially limit the max drive speed sent to the Rover through limiting with the throttle lever on the joystick.
- The coordinator will stop sending drive commands when it detects a joystick button press putting it in the paused state. It will then start sending commands again when brought out of the pause state.
- If using two joysticks to drive, instead of one, the coordinator will calculate the joystick differential to determine and send the correct drive command.

5.2.4 Design Rationale

The use of QT's QSignals for receiving of the joystick control commands will help alleviate inter-thread communication design time, allowing our team to focus on the more important aspects of the coordinator. Sending drive commands via the `rospy` package allows us to natively integrate with the ROS control stack. This is ideal because it allows the Rover Software team to use native navigation and motion planning packages to control the Rover, and the ground station software will fit the expected protocols that those packages use. By having the drive coordinator handle limiting the max speed allowed to be sent to the Rover, we can guarantee that the Rover will never drive faster than intended. If we had instead made the coordinator send two commands, one with the drive command, and one that was a speed limit, there is the possibility that the speed limit command could get lost in transit to the Rover. This same idea can also be applied to the pause state in that simply stopping commands from being sent is more reliable than sending a remote pause command to the Rover.

5.3 Mapping System

5.3.1 Overview

The mapping sub-system will handle the storing and loading of maps of the competition area as well as plotting important landmarks as provided by the user.

5.3.2 Design Concerns

- *Offline Use:* As this software will primarily be used in environments where there are no Internet connections, the mapping sub-system will need to be able to store local maps of the desired competition areas in advance.
- *Zoom Options:* During competition, the user may desire to zoom in or out on any particular area. The software will have to accommodate this by either digitally zooming into an the desired area, or by loading newer high-resolution images at a adjusted zoom level.
- *Location Services:* The mapping system must accurately show where the rover is on the map so that the operator is able to keep track of it. Furthermore the mapping must also allow the setting of way-points. These way-points allow the operator to set a location that the rover will travel to once set.
- *Reliability:* The mapping system will be one of the most used aspects of the ground station software as it will be used for properly navigating the Rover to competition way-points. If the mapping system fails, it may be near impossible for the user to determine where the Rover should be driven.
- *Responsiveness:* Outside of the drive and video sub-systems, the mapping system will be one of the most frequently updated and used features on the ground station software. In order for it to be useful to the user the map updates must be fast and responsive so that the user is not spending valuable competition time waiting for the map to load.

5.3.3 Design Elements

- The mapping system will load satellite imagery of the desired area via the Google Maps API.
- The Google Maps tiles will be cached so they can be used offline.
- Individual image tiles will be stitched together into one large image to be shown in the GUI using either OpenCV or Pillow.
- The map will have icons and/or numbers overlaid on the map image corresponding to the Rover location, navigation way-points, and landmark way-points.
- The aforementioned markers will be accurate on the map based on their GPS positions.
- The map will show a trail of the Rover's previous driving path that fades away over time.
- The map will have a faint grid showing latitudinal and longitudinal divisions.

5.3.4 Design Rationale

The choice of the Google Maps API allows us to use the most up-to-date and readily available maps of the competition area easily. Google Maps has some restrictions placed upon it's use in robots, but do not apply to this system. OpenCV and Pillow are both fast and well-documented frameworks for dealing with image data, and are especially good at fast image stitching. The other design aspects of the mapping system should make the map view easy and intuitive to use, meaning less training will be necessary before a user will understand it.

5.4 Way-points Coordinator

5.4.1 Overview

The way-point coordinator will handle the storing, editing, and displaying of the way-points that the user will be placing for the rover.

5.4.2 Design Concerns

- *Accuracy:* As the coordinator will be controlling the rover when a user is not presently using the HID's. The accuracy is important to reach some points correctly and within a certain distance.
- *Queue Order:* As the coordinator is sending out the values to the drive coordinators, if the order of way-points are off, an incorrect path might be taken.
- *Queue Editing:* The way-point coordinator needs some way of editing values to allow for user mistakes. Humans are imperfect and might enter something wrong or have done something wrong and might need to fix it.
- *Queue Deletion:* Users might need to remove a way-point from a queue to allow for more user flexibility and sudden changes to the system.

5.4.3 Design Elements

- The system will access the Mapping system to control and place any way-points.
- The system will likely be built using a linked-list of nodes that contain information for the rover.
- Adding a way-point to the queue can be like clicking on the visual map in the program or entering GPS coordinates via pop-up or input space in the GUI
- Editing a way-point could be clicking on the way-point and then editing the values that are displayed.
- Deleting a way-point might be clicking a visual list of points and then confirming the action.

5.4.4 Design Rationale

The design of this coordinator revolves around the idea of a linked list version of a queue. The rover must be traveling in the order of way-points created. This is important for the competition in May where way-points are used to control the rover in a few events.

5.5 Arm Visualizer

5.5.1 Overview

The arm visualizer will allow the user to quickly and easily view and understand where the joints on the Mars Rover arm are at any given time. As the arm is moved, these visual indicators will update to show the new arm positions.

5.5.2 Design Concerns

- *Viewing Simplicity:* The indicators need to be visually displayed in such a way that the user instinctively understands what the data means.
- *Responsiveness:* In order to be useful, the indicators will need to update quickly. During competition, the user will use these indicators to position the arm, and major delays in updating these will make them unpleasant to use.
- *Clutter:* As there are many competition events where the arm will not be needed, these visualizations should be able to be disabled so that there is not unnecessary clutter and information on-screen when it is not needed.

5.5.3 Design Elements

- The arm visualizer will take in position information via a ROS topic using `rospy`.
- The arm visualizer will, in the initial version, show this information in native QT GUI elements such as a QGraphicsView or QPixmap embedded in a QLabel.

- The arm visualizer will black out the unneeded visual elements when the arm is not presently attached to the Rover.
- In the event there is spare time, the previous visualizations will be replaced with a 3D visualization of the Rover through ROS' RVIZ tool, where the position changes accurately correspond to movement changes on the model.
- If the previous option is not attainable, the visualizer will attempt to be replaced with a custom OpenGL widget containing the 3D view of the Rover arm, and like previously, will update the model to match the current arm position.

5.5.4 Design Rationale

By taking in position information through ROS topics with `rospy`, we will be able to avoid having to write a custom network message system to provide and interpret this data from the Rover, which would have been a tedious and error prone process. Using native QT widgets, at least for the initial version, will mean that we can more quickly get the pertinent information into a display format that can be understood. The QGraphics view widget in particular will let us draw a scene in a painting environment, and then manipulate the scene to show a arrow moving around a circle for example, with relative ease.

5.6 Arm Coordinator

5.6.1 Overview

This sub-system will handle taking in raw SpaceNav mouse control information and transforming them into usable arm control commands. This will also handle the interpretation of button presses on the SpaceNav mouse to control GUI functions, for example, panning the map around.

5.6.2 Design Concerns

- *Reliability:* This node must be incredibly reliable. If it runs off and sends continuous arm commands, for example, the physical Rover arm will also be out of control.
- *Speed:* As this node is what is sending arm commands, any major processing delays here will make controlling the Rover arm a choppy and unpleasant experience.
- *Pause State Responsiveness:* When the pause button is pressed on the joystick, the Rover will need to stop all movement quickly.

5.6.3 Design Elements

- The coordinator will take in SpaveNav control information via QSignals.
- The coordinator will send Rover arm commands via a ROS topic using `rospy`.
- The coordinator will stop sending arm commands when it detect a joystick button press putting it in the paused state. It will then start sending commands again when brought out of the pause state.

5.6.4 Design Rationale

The use of QT's QSignals for receiving of the SpaceNav control commands will help alleviate inter-thread communication design time, allowing our team to focus on the more important aspects of the coordinator. Sending arm commands via the `rospy` package allows us to natively integrate with the ROS control stack. This is ideal because it allows the

Rover Software team to use native arm movement and motion planning packages to control the Rover arm, and the ground station software will fit the expected protocols that those packages use. By having the arm coordinator node handle stopping sending control commands to the Rover, we can guarantee that the arm won't continue moving if commands stop getting sent, such as in the pause state. If the Rover itself had to determine whether to move or not via a sent pause command, the arm could potentially still move if that external pause command were lost.

5.7 Video Coordinator

5.7.1 Overview

The video coordinator will handle taking in video stream data from the Rover, and displaying that data on the GUI. It will also handling telling the Rover what resolution and potentially bit-rate the ground station would like from a particular camera. Lastly, it will handle switching which video streams are showed in each of the three video display areas on the GUI, including the ability to switch off the secondary/tertiary displays.

5.7.2 Design Concerns

- *Reliability:* The live video streams are a necessity to be able to remotely operate the Rover for most competition events. If the video streams are not functioning, the Rover will do very poorly in competition.
- *Processing Time:* As the video streams will be used the drive the Rover remotely, processing delays should be kept to an absolute minimum so that the driving experience feels more fluid.
- *Resolution Change Detection:* The user should not have to manually change resolutions if a video that was on the primary video window is switched to the secondary or tertiary window. This does not apply if the user overrides the resolution settings in order to get smoother video during times of poor radio reception.

5.7.3 Design Elements

- The video coordinator will use `rospy` to retrieve compressed video data streams from the Rover.
- The video coordinator will format the video data in a way that it can be displayed on a QLabel as an embedded QPixmap.
- The video coordinator will automatically send resolution adjustment commands to the Rover over a ROS topic as the video streams are switched between the primary and secondary/tertiary video windows.
- The video coordinator will automatically handle adjusting the video data so it can be properly shown in the QLabel, even after a resolution adjustment.
- The video coordinator will handling blanking out the secondary/tertiary video displays and stop processing the underlying video if a user requests it.

5.7.4 Design Rationale

By using ROS' built-in compressed video streams, and the accompanying `rospy` libraries for interpreting this video data, our team can focus on the logistics of showing these videos on the GUI. By handling automatic resolution adjustment, we are simplifying the tasks the user will have to perform during competition. The automatic adjustments also help alleviate unnecessary network bandwidth for streams that would not benefit from the higher resolution. Automatic adjustments will both lower latency and allow the Rover to have to process less data, which are both positives in a remote, battery-powered environment.

5.8 Statuses Coordinator

5.8.1 Overview

The statuses coordinator will be tasked with handling all the miscellaneous messages being sent to the ground station from the Rover. It will take in these messages and route them to GUI elements for display. This will include information such as Rover battery level, raw GPS data for the mapping sub-system, and whether sub-components of the Rover are connected, such as the arm.

5.8.2 Design Concerns

- *Minimal Overhead:* As most of these messages are non-critical, it is important for this coordinator to keep its resource usage low so that the processing power is available for other ground station functions.
- *Fast Navigational Updates:* Since navigational updates will be part of these messages, it will be important for these messages to be prioritized for updates to GUI elements such as the compass and speed indicators.
- *Adequate Warnings:* It will be important that this coordinator provides noticeable warnings, potentially through methods such as flashing indicators, when there are problems with Rover systems that it has received a message about. Depending on the warning, a low battery for example, it could be the difference between the Rover winning or losing a competition event.

5.8.3 Design Elements

- The statuses coordinator will take in Rover status messages via ROS topics using `rospy`.
- The statuses coordinator will update any necessary GUI elements such as QLabels with their pertinent formatted information as received via the ROS messages.
- The statuses coordinator will prioritize any updates to GPS and navigation messages so that the user can more easily drive the Rover.
- The statuses coordinator will brightly flash GUI elements that it feels the user needs to pay attention to, and allow the user to cancel the warning by clicking on the GUI element.

5.8.4 Design Rationale

By using ROS topics to handle messages from the Rover to the GUI, our team is able to spend that time working on quickly showing these updates instead of creating custom network message protocols. Prioritizing updates to the navigation GUI elements when performing updates will help the user driving the Rover more reliably understand what the Rover is doing movement-wise, which is more important than secondarily important messages such as battery voltage. By flashing GUI elements in bright colors, and allowing users to cancel these warnings, we are bringing the proper amount of attention to messages that need it while not being obnoxious and having these warning continue indefinitely.

5.9 Logging / Recording Coordinator

5.9.1 Overview

Recording and logging information is very important for monitoring the rover's status and re-run how a rover was performing during some set period.

5.9.2 Design Concerns

- *Storage Space:* With all the video/recording the system is going to do, the space might be limited on the system or on the rover.
- *Video Bitrate:* The video should be recorded in a quality that is independent of the streaming video stream to allow full video quality when retrieved
- *Storage Format:* With the variable size of log files, some settings might be necessary to control logging location, size, or type.

5.9.3 Design Elements

- The coordinator will take any information from the rover as a stream of information.
- The coordinator will store any video stream that the rover sends back to save back on overhead on the rover.
- The rover will not be saving any files to allow for overhead and processing.

5.9.4 Design Rationale

The rover is going to be overloaded with calculations and trans-coding, so the ground station should be able to offload any extra computation onto it. The ground control system will take care of storing information like log files in session logs. There is an inbuilt system using ROS that makes a bag that can be used to log everything.

6 User Interface Design

6.1 Left Screen

6.1.1 Mock-up

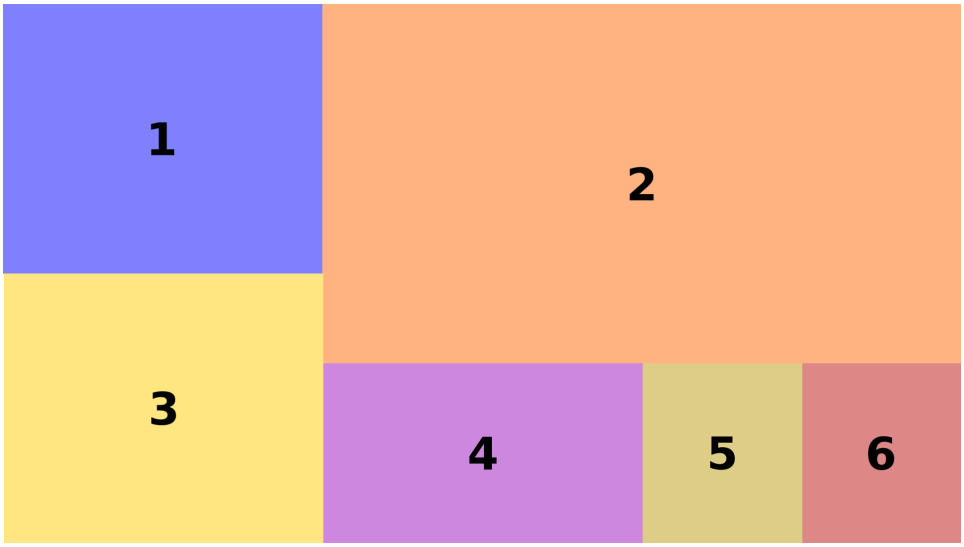


Fig. 1: Left Screen Mock-up

6.2 Zones

6.2.1 1: System Statuses / Sensor Readings

This section will display all status and sensor data both from the Rover and about the ground station itself. This includes items such as radio signal strength, GPS accuracy, and whether the Ground Station has connection to the drive joystick/s and SpaceNav mouse device.

6.2.2 2: Map Display

The map display will display a satellite view of the competition areas with navigation and land-mark way-points, the Rover, and the Rover movement trail overlaid.

6.2.3 3: Recording / Logging / Settings

This section will be a tabbed grouping that by default will show the controls for starting ROS bag recordings, but will also have a second tab for live logs and a third tab for settings that adjust items such as which map is being shown.

6.2.4 4: Way-point Entry / Autonomy Controls

This block will allow for manual entry of GPS coordinates, as are provided by the competition at the beginning of some events. The user will be able to choose whether the entry is being added as a navigation or landmark way-point as well as using the entry fields to edit the way-points from pre-entered points.

6.2.5 5: Navigation Way-points Listing

This area will list, in order, the navigation way-points that the system is currently set to follow, both for manual driving and for the autonomy portion.

6.2.6 6: Landmark Way-points Listing

This will list, in the order they were added, the landmark way-points that the user has entered to make important points on the competition field.

6.3 Layout Rationale

This display shows most of the information regarding the Rover that does not need to be viewed while the Rover is being actively driven, or while the arm is being moved. When the software starts and is first connecting to the Rover, the user will spend some time studying system statuses, entering way-points, changing settings, and checking logs if needed. Once they are done with these initial checks and entries, most of the rest of the control experience will take place on the right hand monitor. In the case that the user wants to quickly glance at the map while they are driving, the map will come in to view easily as it is as close to the right hand monitor as it can be. As the logging and setting views will hopefully not need to be viewed very often, combining them onto a tabbed GUI element helps reduce used space while still leaving them accessible when needed.

6.4 Right Screen

6.4.1 Mock-up

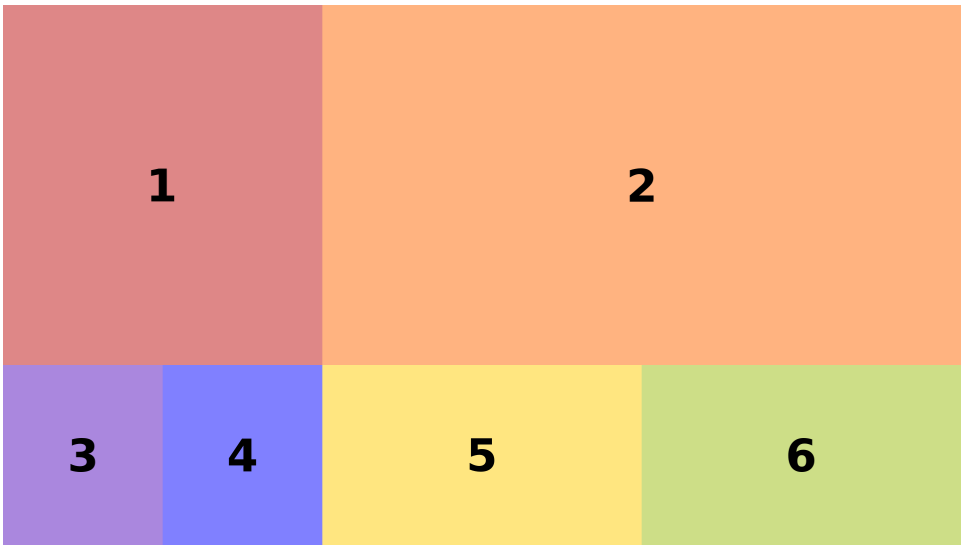


Fig. 2: Right Screen Mock-up

6.5 Zones

6.5.1 1: Arm Visualization

This visualization area will show the joint positions of the Rover arm when the arm is attached. In its simplest form, the visualization will be a simple line drawn in a reference frame, that adjusts its position as the arm is moved. For this version, each joint would have its own visualization box. If there is enough time, our team will attempt to integrate with RVIZ, the ROS visualization package, to show a 3D view of the arm moving. If the RVIZ solution does not seem feasible and there is still extra time, we may alternatively try to implement a custom OpenGL view of the arm.

6.5.2 2: Primary Video Display

This will show the stream from the camera on the Rover is currently selected as the primary video stream.

6.5.3 3: Heading Compass

This heading compass will dynamically rotate to match the current heading of the Rover. It will also mark on the compass edge the currently active way-point, making it easy for the user driving the Rover to determine which direction they need to turn to line up with a way-point marker.

6.5.4 4: Speed and Speed Limit Display

This area will show the current Rover speed is meters per second as is reported by the Rover GPS. Additionally, it will also show the current speed limit of the Rover from zero to one hundred percent as has been limited by the user via the joystick throttle lever.

6.5.5 5: Secondary Video Display

This area will show the stream from the camera on th Rover that is currently selected as the secondary video stream. This stream has the capability to be disabled and show a placeholder image if the team needs to save on radio bandwidth.

6.5.6 6: Tertiary Video Display

This area will show the stream from the camera on th Rover that is currently selected as the tertiary video stream. This stream has the capability to be disabled and show a placeholder image if the team needs to save on radio bandwidth.

6.6 Layout Rationale

This screen will be showing the most commonly looked at data for the user driving the Rover. During a normal competition, the user will start by entering way-points to drive the Rover to, and then switch to the joystick(s) and SpaceNav mouse. Once they start driving, the user will be heavily focused on monitoring video data to make sure they do not run the Rover into anything. While driving to the way-points to get to an end location, the user will be able to easily see the compass indicator, with a marker for the direction they should be heading. After arriving at an ending way-point, the user can then seamlessly transition to moving the Rover arm (for competition challenges where this is the case) and watching the visualization update on the same screen as the video of the arm moving. By laying out these particular elements on this screen, the user will be less likely to have to look at the left screen except when the Rover is not moving.

7 Conclusion

Throughout this document, we have covered the design considerations, component design, and GUI design for the Mars Rover ground station software that our team will be implementing. By following the design guidelines laid out in this document over the next five months, we are confident that our team will be able to accomplish the desired state of the Mars Rover ground station software. In doing so, not only will we be delivering to our client a useful and robust front-end to the Mars Rover for the competition next June, but also hopefully a base to work off of for many years into the future.

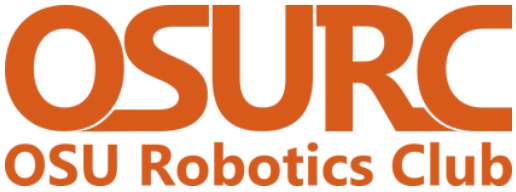
3.2 Additions

There were no additions to our original design document.

4 Tech Review Document

4.1 Original Documents

4.1.1 Chris Pham



CS Capstone Technology Review

June 12, 2018

OSU Robotics Club
Mars Rover Ground Station

Prepared for

OSU Robotics Club

Nick McComb

Prepared by

Group 30

Ground Station Software Team

Christopher Pham

Abstract

This document examines the differences between three technologies in three different subjects. Python GUI frameworks are going to be used on the project to show and abstract information coming from the rover towards the end user. The arm and graphical visualizers will be used to show how an object or the rover's state. Mapping software is going to be needed for any autonomy or any view of dangers on the map. This document goes over these different subjects and reviews technologies that will be suited for the system.

CONTENTS

1 Introduction

My section of the technology review document is going to revolve around the ”graphical” side of the ground station software.

2 Python GUI Frameworks

2.1 Overview

A graphical user interface (GUI) is used by a program to obscure and allow the user to access and interact with objects instead of a command line interface system. The GUI allows for people to interact with the system even if they are not trained. A GUI framework is a pack of tools that will build a GUI using a commands and bindings and will allow for bindings across many systems like Linux, OS X, Windows, Android, and iOS. A framework also comes with an added benefit of being made and maintained by the public or a company instead of being made in-house.

2.2 Criteria

Given the restrictions at the request of our client, I’ve chosen to focus more towards Python based implementations of graphical user interfaces. Some potential choices can be used in other languages but that is not a main focus point of this project. The interface needs to be able to go into full-screen and needs to be able to change the colors of the GUI itself to a darker theme. It also needs to support many video streams.

2.3 Potential Choices

2.3.1 PyQT

PyQt is a set of bindings for Python for the GUI framework Qt developed by The Qt company. [1] PyQt can run on any platform that can support Qt like Windows, OS X, Linux, iOS, and Android. Qt is not just a GUI toolkit, but also comes with ”abstractions of network sockets, threads, Unicode, regular expressions, SQL databases, SVG, OpenGL, XML, a fully functional web browser, a help system, a multimedia framework, as well as a rich collection of GUI widgets.” [1] Qt also includes a designer called Qt Designer and PyQt is able to generate Python code from it.

2.3.2 Tkinter

Tkinker is GUI framework that is included in any Python 2 or 3 installation package. Tkinker itself is a wrapper on top of the Tcl/Tk package from ActiveState. [2] Tkinker supports ”most Unix platforms, as well as on Windows systems”. [2] Tcl/Tk comes with a BSD-like license that allows for commercial use of their framework.

2.3.3 Kivy

Kivy is an open source framework for Python that is under MIT license. [3] Kivy runs on Linux, Windows, OS X, Android, iOS and can use the same code for all platforms. It can natively use most inputs and devices including ”WM_Touch, WM_Pen, Mac OS X Trackpad and Magic Mouse, Mtdev, Linux Kernel HID, TUIO”. [3] Kivy also includes hardware acceleration using OpenGL ES 2.

2.4 Comparison

With all these frameworks, Tkinker/(Tcl/Tk) and PyQt/(Qt) are also possible on other languages using some other wrappers, unlike Kivy. However, unlike the other frameworks, which are built in C or C++, Kivy is natively built in Python. All the frameworks support OpenGL contexts (windows/frames) only PyQt and Kivy support OpenGL acceleration on the menus which might be necessary for how many video streams we are pushing. Kivy in contrast to the other frameworks also supports multi-touch enabled devices like phones, and tablets unlike the other frameworks. Tkinker however is included in all builds of Python released, making it very easy for quick and lightweight programming and prototyping compared to other frameworks. Qt requires payment for any commercial product, unlike the others.

2.5 Decision

My suggestion for this project would be PyQt. PyQt's fee that Qt asks for is not applied to this project. PyQt and Qt in general have good support and documentation because its longevity, numerous tutorials, and support by the community. Kivy would work well for the project if it had more documentation but its OpenGL rendering would be fantastic for design ideas we have. Kivy is also built more towards multi-input systems which is not our focus. Tkinker would work well for small projects but the lack of OpenGL acceleration might be questionable for the all the video streams that might be handled on the system.

3 ARM/GRAPHICAL VISUALIZATION

3.1 Overview

For this project, one important aspect is going to be visualizing how the arm is currently moving in respect to the rover itself and showing how the user input changes the arm visually. The need comes from the integration between our project and the other senior project that revolves around the rover arm. Without the visualization, the other senior project can not test to see if their arm works correctly on the rover and for later competition usage.

3.2 Criteria

If this was to be made using the command line or via text, it would be very confusing. Per requests, the arm visualizer needs to resemble the arms. The other requirements would be needing to able to run on Linux (Ubuntu) and easy integration with the ROS (Robot Operating System) on the Rover.

3.3 Potential Choices

3.3.1 ROS Visualization

Inbuilt into ROS, there is a package called rvis which allows for the system to visualize objects using displays. The included displays in rvis are [4]:

Axes	Effort	Camera	Grid
Grid Cells	Image	Interactive Marker	Laser Scan
Map	Markers	Path	Point
Pose	Pose Array	Point Cloud (two types)	Polygon
Odometry	Range	RobotModel	TF
	Wrench	Oculus	

The package is also very well documented with sample code and projects to show how some interactions happen.

3.3.2 OpenGL visualization

We can build a system using one of our GUI frameworks to give us access to an OpenGL window. In that window, we can then write to it using some OpenGL and Python code to form the arm. We can use any shapes we want like line pieces or even importing the arm's 3D model and modifying it from there. Inverse kinematics is needed to get the joints working correctly if the arm is moving from point to point, and normal kinematics to get the final location from arm segment movements. We would be able to control how fast the video/render would be refreshing, drawing, or rendering size which would allow for granularity control to reduce CPU or GPU usage.

3.3.3 Moveit Simulator

We could also use simulator software that can take the signals from the inputs/joysticks and then send them both to the rover and simulator. The simulator can emulate what the arm SHOULD do and can be imported from the other senior project that is already simulating the arm using the same software. The video can be captured using OpenCV or open source prorgrams like Istanbul to stream the video to the GUI.

3.4 Comparison

Compared to ROS and OpenGL, Moveit actually just emulates the arm while the others calculate how the arm should move using inverse kinematics. ROS has all built-in displays in which many other things like mapping software and waypoint systems would be much simpler than building our own system for the project. OpenGL provides flexibility compared to the others but does not include any built-in packages or anything of the sort.

3.5 Decision

I think that the visualization should be a mix of ROS and some custom OpenGL. The combination of the two allow for access to packages that are included and built around ROS while also including the flexibility that is included with OpenGL. The arm can be visualized using the included packages like rvis, but other things like the IMU (Inertia Measurement Unit) might be hard to visualize in rvis and would be possible with custom OpenGL code.

4 Mapping Software

4.1 Overview

Another part of the project is the mapping system. In competition, the Robotics team wishes to see where the rover is currently on the map. We are going off the assumption that the way-point system needs to be built, but we can use a package to deal with the GPS location and mapping.

4.2 Criteria

A requirement would be ease of integration between the rover and ground station. Another would be the ability to use the mapping software with a reasonable amount of delay or no Internet at all.

4.3 Potential Choice

4.3.1 ROS Packages

There are inbuilt packages for GPS and GPS location parsing in the ROS subsystem. Using packages like rvis or mapviZ can display the map and transform the map image to display the correct location. The location can be calculated on the rover or on the ground control system.

4.3.2 *Google Maps/Earth*

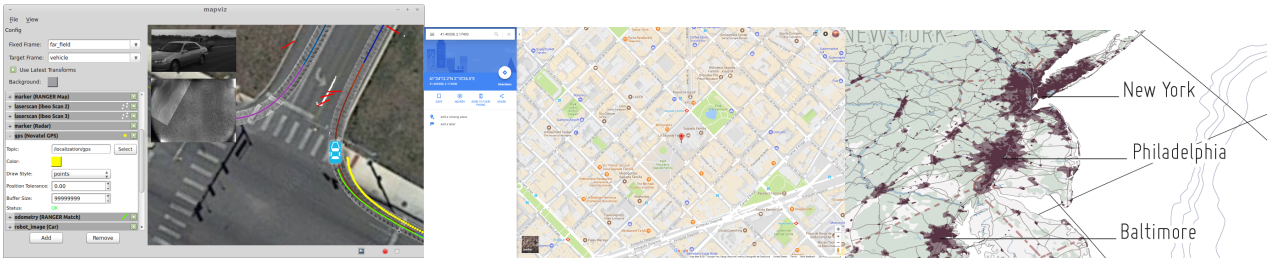
Using the same packages as above, instead of parsing the location on the robot or ground control system, the rover/ground station can parse the information given to it by sending it to Google. This would remove any need for calculating location at all and the map would be generated by Google.

4.3.3 *Custom Mapping/Packages*

We could build a system using our own polling and parsing of the GPS information. This can be represented by texture location using an external package like Kartograph [6] for Python. Kartograph can then return an SVG (Scalable Vector Graphics) which can be displayed on the GUI.

4.4 Comparison

ROS and Google are very similar with the only differentiating thing about the two would be where the information being sent, locally or via ground control to a remote server. ROS and the custom software are very similar but uses in-house code or open source code. The only difference between all of these is how the GPS data is analyzed and texture map is generated and then put onto the screen.



(a) Map Visualization Using Mapviz Package [5]

(b) Google Maps using Decimal Degrees

(c) Kartograph Render [6]

Fig. 1: Map Visualization

4.5 Decision

I think that the project should use the ROS packages because all of those are open source and packages are constantly made for problems like this. ROS also has integrations between other packages like rvis and mapping software. And in the environment where the competition is going to take place, the speed and the Internet quality might be sub-par and any external requests would make the system react very slowly or be completely off. Another problem with using Google would be that we can not use their mapping system to do anything autonomous as stated from their ToS (Terms of Service) and we would need to use some other system like OpenSourceMaps. If we had to build a custom solution to this project, Kartograph would be good. Kartograph produces SVG files which Qt takes for but the lack of documentation might make building the project longer than alloted.

REFERENCES

[1] "What is PyQt." Riverbank Computing Limited. November 2017

[2] "Tkinker – Python interface to Tcl/TK" Python Software Foundation. November 2017

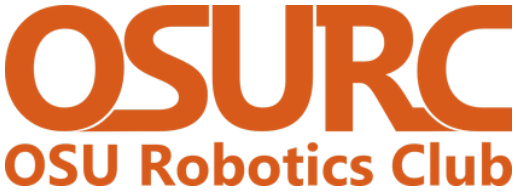
[3] ”Kivy.” Kivy Organization. November 2017

[4] ”rvis/DisplayTypes” David Gossow. August 17, 2013

[5] ”Mapviz.” swri-robotics, November 21, 2017

[6] ”Kartograph - A Simple and lightweight frameowkr for createing interavtive vector maps”

4.1.2 Ken Steinfeldt



CS CAPSTONE PROBLEM STATEMENT

NOVEMBER 22, 2017

OSU ROBOTICS CLUB MARS ROVER GROUND STATION

PREPARED FOR

OSU ROBOTICS CLUB

NICK MCCOMB

PREPARED BY

GROUP 30

KENNETH STEINFELDT

Abstract

In order to successfully compete in the Mars Rover competition, the ground station software must run from an actual ground station. This means that the ground control software must run on hardware that will make up the physical ground control station before it can be delivered to the robotics club. In order to accomplish this we must determine various levels of technology. In this document I research different solutions that can be used for this purpose. The technologies discussed in this document are the host operating system, the computing hardware that will make up the host, and how that hardware will be physically packaged for quick deployment at the competition.

CONTENTS

1 Operating System

1.1 Overview

The ground control software must run from a central host in order to be controlled by the operator. Therefore, an operating system must be chosen on which the software will run. This is an important technology decision as it is integral to the foundation of the software, and will likely be unchanged by future robotics team developers. This section describes the research and process behind deciding the best operating system on which to run the ground control software.

1.2 Criteria

The chosen operating system must meet several points of criteria. It should be well supported in robotics in order to ease development work, ensure robust software, and enable future developers. It should be acquired at minimum cost in order to best allocate robotics resources now and in the future. There should be a reasonable expectation that development on the operating system is familiar to this team as well as future teams doing development work on the ground control software.

1.3 Linux

Going into this research Linux was the preferred choice for this technology. Linux meets all basic criteria for this choice.

- Linux is the primary choice of operating systems in robotics. Because of this it is best supported by the community and will offer developers the greatest amount of support going forward. Of note, the Robot Operating System (ROS) [2], which is the foremost robotics operating system and academically supported, only runs on the Linux operating system. Furthermore, ROS is technology under review by this team, for this project. The use of ROS would require that Linux be chosen.
- All robotics team developers are students of Oregon State University, and as such have been routinely exposed to developing on Linux. Computer science students of OSU are required to develop on Linux for a majority of their programming courses. Because of this there is a reasonable expectation that all developers, future and present, will be familiar with Linux development. It is true that this team is most familiar with Linux development. In fact, it is likely that a large majority of these developers are more familiar with Linux development than development on any other operating system. We expect that this familiarity will greatly increase the quality of the software as well as the pace at which this software can be built and maintained.
- Linux is famously free and open sourced. This means that procurement of this operating system is exceedingly easy and cost free. This is greatly beneficial to the team as resources are limited. In acquiring this technology freely, the robotics team is free to allocate more resources to other parts of the project, such as the rover itself, or other ground station costs.
- Linux is famously light-weight and modifiable, allowing for a less powerful hardware solution.

1.4 MacOS

MacOS is Apple's Unix operating system designed for Apple computers. MacOS meets some but not all base criteria for this choice.

- MacOS is a Unix operating system [1], and as such shares many traits and tools with Linux. Because of this ROS runs and is supported on MacOS [2].
- Because of its similarity to Linux, it is also fair to say that there is a reasonable expectation that developers, present and future, will be familiar with the development environment it provides. Therefore it is reasonable to expect that MacOS can provide a stable, robust foundation on which the ground station can be built.
- Unfortunately, MacOS is only legally permitted to be run on Apple hardware. This requirement causes significant additional cost to the project and restricts development environment for anyone working on the ground station.

1.5 Windows

Microsoft Windows is the most popular PC operating system in the world. It is vastly different from Linux in both design and philosophy and represents a significant choice for this project.

- ROS is not supported on the Windows platform [2], which means that selecting the Windows operating system restricts us greatly. Moreover, Windows generally does not have the support of the general robotics environment in academics. Finding a Windows specific solution could be a significant hindrance to this team.
- The standard path through computer science at OSU does not include much, if any, development experience on the Windows platform. Therefore it is not reasonable to expect that current or incoming developers will be able to meaningfully impact the project without a significant, upfront, learning curve. It is also fair to say that the unfamiliarity of developing on Windows and the rapid deployment model that this project will take is likely to create a more mistake prone environment that will likely lead to less robust and reliable software.
- Windows is a licensed operating system that runs on nearly all PCs. However, a license can be procured through the University at no cost. Windows would not be a significant monetary burden if selected.

1.6 Summary

Clearly each operating system has different traits and attributes. MacOS and Linux are both Unix operating systems and share many ideas and offer a similar environment. Windows is a drastic change from the previously mentioned two, but is the world's most popular PC operating system, and is professionally supported.

1.7 Conclusion

Linux is the clear choice here as it offers the project everything that is needed in an operating system and affords the greatest amount of flexibility on future choices. Linux is also the most familiar operating system to the developers and can be modified as needed in order to meet the exact definitions required for the project.

2 Computing Hardware

2.1 Overview

As the ground station controls the rover, it must be a self-contained computer. This affords us a few choices. This team can choose to install the ground station software on either a laptop, an Intel NUC mini-PC, or a full-sized workstation PC. It is important to note that the ground station must be transported a significant distance to the competition and must be setup and torn down quickly.

2.2 Criteria

The computing hardware for the project must fulfill specific requirements. It must have enough processing power to drive the software. It must be small, portable, and easy to setup. It must not incur a significant cost. Due to the nature of the ground control software, all three of these requirements must be met.

2.3 Laptop

A laptop may seem like the most obvious choice and offers many advantages. They are already designed to be portable and offer enough computing power for the base station. They can be developed on just the same as any other computer. To use a laptop, one will have to be purchased, causing significant cost to the project.

2.4 Intel NUC

An Intel NUC is a miniature computer designed for similar use cases as this. It provides more than enough computing power and its small size will afford easy transportation and setup. The client prefers the use of an Intel NUC and is providing one for the project.

2.5 Full-Sized PC

A full-sized PC offers significant processing power but at the cost of size. Transporting and setting up one of these will be laborious and slow. The extra computing power afforded by a full workstation is unnecessary and wasted on the task. A workstation such as this will have to be purchased, causing significant cost to the project.

2.6 Summary

All three options can adequately fill the need of the project. A laptop is more portable and as powerful as a NUC, and a workstation is significantly more powerful than both. However, this nature of the project requires that the most efficient choice be made.

2.7 Conclusion

While a laptop may represent the most desirable choice, it is not the most efficient choice to complete the task. An Intel NUC is being provided to the group at no cost, and it has been indicated by the client that this is the preferred hardware choice. the NUC provides more than enough computing power than is required with almost the same portability as a laptop.

3 Quick-Deploy Package

3.1 Overview

Transportation and set up are important to the Mars Rover competition, as it is a timed trial. In order to be successful, a packaging solution must be thoroughly considered. For example, does a solution have to be created, or does a solution already exist that can be purchase? Should the packaging be totally self contained or require external set up materials?

3.2 Criteria

The ground station will have to easily transported and unpacked. The competition is in the desert and reliability is paramount is it is also important that the transportation solution is adequate to protect the equipment while in transit and also in use. Also required is that the ground station must be setup with software launched in 2 minutes or less.

3.3 Self-Contained Case

This case contains all equipment necessary for the ground station to function, already connected, missing only power. Logistically this is the nicest solution, however, such a case will require some customization from the team, as such a case has not yet been found ready-made. Also the equipment will have to be carried approximately 50 feet from the transportation vehicle to the setup location, so it cannot be too heavy to be carried.

3.4 Separately Packaged Components

In this solution, all components of the station are packed individually and will have to be connected and powered separately once transported to the setup location. In contrast to the self contained case, this does not require any modification or customization, which means it is less resource intensive. However, moving the station from the transport vehicle to the setup location will require more than a single trip. This can introduce error as the client's rover team moves quickly to meet the two minute deadline.

3.5 Pre-Packed Case

Unlike the previous two solutions, in this case all necessary components are fit into a backpack or case, but not connected. This is different from the previous option because components are packaged together as opposed to separately, but are not connected. This allows for a single trip to unload the vehicle, but still requires that each component is connected and powered on in two minutes. Again this could lead to errors or troubleshooting in the two minutes allowed for setup, but may prove to be more feasible than the fully contained and connected ground station.

3.6 Summary

Each solution carries pros and cons. In this decision the team must determine how much value to impose on time. Is two minutes enough time for a full setup?

3.7 Conclusion

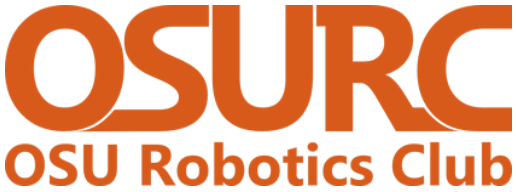
Though it is more resource heavy, it is better to have the entire ground station pre-connected in a hard case. This solution will cost both time and money, but will greatly increase the chance of success for the Mars Rover team during the competition. Moreover the solution can be reused in years to come, which is an important condition from our client. Packaging the ground station together unconnected provides the second most favorable option.

REFERENCES

[1] The Open Group. The open group official register of unix certified products.

[2] ROS Wiki. Introduction.

4.1.3 Corwin Perren



CS CAPSTONE TECHNOLOGY REVIEW

NOVEMBER 21, 2017

OSU ROBOTICS CLUB
MARS ROVER GROUND STATION

PREPARED FOR

OSU ROBOTICS CLUB

NICK MCCOMB

PREPARED BY

CORWIN PERREN

Abstract

This document will cover my personal team role in developing the OSU Robotics Club's Mars Rover Ground station software. Additionally, it will cover different options for robotics frameworks, radio systems, and joysticks that the Rover team and specifically the Ground Station software team may end up using to complete our software package.

CONTENTS

1 Personal Role

My role on the Mars Rover Ground Station team will primarily be as a programmer writing code to interface with the team's Mars Rover. As part of this interfacing, the software I help write will allow for remote control of the Rover, the showing of video streams broadcast from the Rover, and the showing of Rover status and mapping information. An additional role I will have will be as a bridge between the rest of our capstone team and the Mars Rover group, as needed. In the past, I have both been a member of the Mars Rover team and the electrical team lead, so my knowledge of the competition as well as those in charge of the Rover team will help facilitate easy communication.

2 Goals

The goals of the rest of this document are to cover three separate technological aspects of the Mars Rover Ground Station that have multiple viable solutions. Evaluation of three of these options will then culminate in choosing one of one to be used on the Ground Station. The first technology to be looked at will be the robotics framework the software will be interfacing with. As the Mars Rover is a complicated robot, choosing the right option here will affect how difficult it will be to interface with the Rover's systems. The second technology will cover the radio system options the Ground Station could use to communicate with the Rover. Selection of this system will determine how much and how fast data can be transmitted between the Ground Station and Rover, both of which are major factors in how usable the Rover is during competition. The final technology will be the options for USB joysticks that will be used to drive the Rover remotely using this Ground Station software.

3 Robotics Frameworks

3.1 Overview

When working with large robotic systems there are often many sensors, actuators, computers, and cameras that all need to work in unison to make the platform perform its desired task. In order to facilitate this intercommunication most robots use off the shelf robotics frameworks so that programmers can focus on implementing higher level features. The result is that less time is spent on lower leveling coding for things like sending messages between robot system nodes and more time can be spend implementing features such as obstacle avoidance. As the Mars Rover Ground Station software is interfacing with a complicated robot, it is definitely important to consider different options for these frameworks to choose the one best suited to the project.

3.2 Criteria

- *Feature Coverage*: Ideally the framework should take care of as many lower level tasks as possible including, but not limited to, inter-node communications, simplified computer vision, localization, motion-planning, and built-in simulators.
- *Hardware Compatibility*: The framework would preferably have pre-built nodes/libraries/drivers for common hardware like GPS and cameras.
- *Documentation*: In order to facilitate rapid development, ample and well-maintained documentation is a must.
- *Community / Resource Availability*: To supplement official documentation, the framework should have a well-developed community to allow for easy answering of framework programming questions when encountered as well as example code when needed.

- *Development Time*: Due to the limited time constraints, the framework used should allow for fast development of the software. This component will take into account all the previous criteria.

3.3 Potential Options

3.3.1 Custom Framework

This option involves writing a framework from scratch in-house. Due to its popularity within the Mars Rover team, Python would most likely be used as the language of choice to develop such a system. A custom framework would have to incorporate writing nodes for handling GPS localization, camera processing, computer-vision based obstacle avoidance, drive system control, arm system control, and many others with little to no starting code. And important aspect of this system, with respect to the ground control software, is that a custom system would need specialty sending and receiving nodes for handling transmission of control, status, and video data.

- *Feature Coverage*: Complete coverage is possible.
- *Hardware Compatibility*: Complete hardware support is possible.
- *Documentation*: Only pre-made documentation for libraries is available.
- *Community / Resource Availability*: Reliant on availability for any libraries used.
- *Development Time*: Development will take an incredibly long amount of time.

3.3.2 Mobile Robotics Programming Toolkit (MRPT)

The Mobile Robotics Programming Toolkit is a robotics framework developed to handle the difficult aspects of robot localization, mapping, computer vision, and motion planning. While this does not cover all of the modules needed for the Mars Rover team, nor the ground station, it would greatly simplify some of the design work for the more challenging aspects of the Rover. Custom nodes would still have to be made for drive and arm system control as well as custom messaging nodes.

- *Feature Coverage*: Partial coverage. Still requires writing complex nodes for some features.
- *Hardware Compatibility*: Partial hardware support. Custom additions would be needed to support the rest of the robot hardware.
- *Documentation*: A reasonable number of official tutorials are available.
- *Community / Resource Availability*: Help can be found on stackoverflow.com with the search tag ”mrpt”.
- *Development Time*: Development will take a moderate amount of time. This option still requires a significant amount of custom node development.

3.3.3 Robot Operating System (ROS)

ROS is the most popular robotics framework in the world, and it shows in terms of the complete package it provides. Nodes can be found, both official and unofficial, for most features the Robotics Club’s Mars Rover might need. This includes localization and mapping, computer-vision, obstacle avoidance, motion planning, arm and drive system nodes, and and robust messaging framework. For any nodes that are not complete, they often provide a great starting point so we would at least not be starting from scratch.

- *Feature Coverage*: Excellent coverage. Will require some modification of some feature nodes, but most are pre-built and ready to use.

- *Hardware Compatibility*: Excellent hardware support. ROS comes with support for many robotic hardware tools, and abstracted nodes also for easy modification for custom hardware when needed.
- *Documentation*: Excellent documentation and tutorials on ROS's website.
- *Community / Resource Availability*: Many forums and sites provide help with ROS, as well as many Youtube video series dedicated to ROS.
- *Development Time*: Development time will be greatly reduced due to pre-built nodes and great documentation.

3.4 Discussion

In terms of feature coverage, ROS seems to be most complete. However, depending on how difficult implementing the remaining nodes might be, MRPT might not be too far off in terms of complexity. A custom solution has the benefit of having complete coverage once the code is written, but would take considerably longer to write overall.

With regards to hardware compatibility, it's a similar spread as with feature coverage. ROS is most compatible with the least amount of effort from the start. MRPT would require a fair bit of work to get working with a lot of the hardware that is not directly related to mapping, localization, and computer-vision. The custom solution again has the benefit of having complete hardware support once done, but could require a very large amount of time to get all hardware working.

Documentation-wise, the custom solution is the worst option. Any development will depend on general programming documentation for the language used, and any that exists for libraries in that language. MRPT has some documentation, but it's mostly limited to tutorials and provides less in-depth information for programmers wanting to adjust MRPT to a custom application. ROS, on the other hand, has excellent documentation and many tutorials.

The community and resource availability for the custom solution is essentially non-existent. Any community and resources would have to be internal to the Rover team or Ground Station software team. MRPT at least has a small community and is active on stackoverflow to answer questions when needed. ROS's community and availability is excellent by contrast. There are a multitude of forums, but official and unofficial providing support, and ample third party resources providing examples and videos.

Lastly, when it comes to development time ROS appears to win in this category. The combination of many pre-built nodes plus excellent documentation and community support helps make development much speedier than other options. MRPT is next best, with at least a starting framework to begin with and build off of. The custom solution would be a nightmare in terms of development time and would likely not be complete before competition.

3.5 Conclusion

For the choice of a robotics framework ROS definitely seem to be most fully-featured and quickest to implement framework and wins out in every single category compared to MRPT or a custom solution. Additionally, the fact that ROS has been actively developed for ten years shows in its mature and robust nodes and documentation that should allow the Rover team and our Ground Station software team to more quickly develop the desired application.

4 Radio System

4.1 Overview

During the University Rover Challenge competition, most of the events that take place are remote control events where the user must remotely operate their Rover from a remote base station. During both the remotely operated and

autonomous events, data will need to be sent back and forth between the Rover and its base station. As such, the radios used make a huge difference in how well the Rover performs. Here, we'll look into a few different options for radio systems.

4.2 Criteria

- *Cost*: The radio must be as low cost as possible while still providing all needed functions.
- *URC Radio Rules*: The radio must meet the URC competition radio requirements.
- *Data Throughput*: The radio must allow for high enough data throughput to allow for comfortable remote driving of the Rover.
- *Range*: The radio must be able to reach at least 1km away per URC requirements.

4.3 Potential Options

4.3.1 RockBLOCK Mk2 Iridium SatComm

This satellite modem would allow for remote communications with the Rover anywhere the Rover has line of sight to the open sky. While more difficult to come by, and more difficult to interface with, these radios would near guarantee consistent communication between the Rover and ground station due to the open nature of the Utah desert where the competition will take place.

- *Cost*: The radio is expensive at $250, and also has costs per message.
- *URC Radio Rules*: This radio would meet URC requirements with prior approval.
- *Data Throughput*: The data throughput of this radio would be abysmal at one message every 10 seconds.
- *Range*: Range is essentially unlimited in the open desert of the competition area.

4.3.2 LoRa 433MHz Radio

These small serial radios are cheap and easy to use, but have low power output and require an amateur radio license to operate. These radios are quite easy to acquire and provide a simple abstract serial-over-radio interface that is easy to use.

- *Cost*: The radio is incredibly cheap at $20 each.
- *URC Radio Rules*: This radio would meet URC requirements with prior approval and licensing.
- *Data Throughput*: The data throughput of this radio would be quite slow at 19.2 kbaud per second, but enough to drive the Rover and receive status messages.
- *Range*: Line of sight these radios are rated at 2km.

4.3.3 Ubiquity Rocket M2

The Rocket M2 radios are 2.4GHz long-range WiFi radios that provide a remote Ethernet link between two stations. These radios are reasonably easy to come by, and while requiring some networking knowlege to set up, provides a simple-to-use interface that natively will integrate with Robotics Frameworks such as ROS.

- *Cost*: The radio is expensive at $250 each.
- *URC Radio Rules*: This radio meets URC requirements as it is on the pre-approved radios list.

- *Data Throughput*: The data throughput of these radios are up to 100MBit per second, enough to do control and status data as well as Rover video.
- *Range*: Line of sight, these radios have been tested at up to 1.25km by Rover team members.

4.4 Discussion

From a cost perspective, the LoRa radios definitely win out at a mere $20. The Rocket M2 and RockBLOCK are matched in price, but barring being beyond the range limit of the M2 radios, the RockBLOCK is not a very good contender in terms of throughput for price.

All of these radios are valid by URC competition rules, although both the satellite radio and LoRa radio would require express prior permission from the URC judges.

In terms of data throughput, the Rocket M2 is an easy winner, especially when considering the fact that it's fast enough to also send video data over rather than having to have a second radio for video transmission. The LoRa radio would be reasonable if only control and status information was being sent, but the satellite RockBLOCK would basically be useless for remote operation. To use the RockBLOCK you'd have to send the Rover way-points and let the Rover drive on its own.

Range-wise the satellite modem is the best option, but overkill for what the requirements of the competition are. LoRa is the next best option with 2km line of sight, but it is worth noting that the M2 radios also have a range greater than what is needed during competition.

4.5 Conclusion

Not surprisingly the RockBLOCK is not a reasonable option for this project. However, both the LoRa and M2 radios have enough data throughput and range to make it through the competition happily. With this in mind, the Rocket M2 radios are the optimal radio to use. While they are expensive at $250, they have enough bandwidth to transmit video data as well as control data meaning the team can save the costs of a secondary radio system for video.

5 JOYSTICKS

5.1 Overview

The ground station software's primary function is to allow for remote control of the Rover via the use of joysticks. Therefore, it is not surprising that the choice of joystick is important as it is the primary point of contact between a driver and the Rover itself. Below we'll compare a few different choices for USB joysticks.

5.2 Criteria

- *Cost*: Due to cost restrictions imposed by the competition rules, keeping the cost as low as possible is a priority.
- *Control Feature Set*: The joystick chosen should have pitch, roll, a four position hat stick, at least five buttons, and a throttle slider.
- *Ambidextrous Grip*: As two joysticks will be bought, one to use with the left hand and one with the right, it must be one capable of being used with either hand from the factory.
- *Reviews*: User reviews for the joystick must imply that the joystick is well-made and is unlikely to fail during competition.

5.3 Potential Options

5.3.1 Logitech Saitek X52

The Saitek X52 Flight Control joystick has a large airplane style throttle control as well as a normal ambidextrous joystick. The throttle unit and joystick are separate pieces that are connected via a cable. [1]

- *Cost*: Expensive at $142.
- *Control Feature Set*: Exceeds minimum feature set.
- *Ambidextrous Grip*: Yes
- *Reviews*: This joystick has slightly better than average reviews, but complaints seem to focus on build quality issues where the joystick wears out prematurely.

5.3.2 Logitech Saitek Evo

The Saitek Evo joystick is a simple single-piece unit that has nine buttons, an 8-way POV hat, and a single throttle control. This is a pretty generic joystick. [2]

- *Cost*: Very cheap at $45.
- *Control Feature Set*: Exceeds minimum feature set.
- *Ambidextrous Grip*: Yes
- *Reviews*: This joystick gets average reviews, with major complaints being issues with the joystick not staying centered as it gets worn in, causing small amounts of drift on the control axes.

5.3.3 Hercules Thrustmaster T16000M

The Hercules Thrustmaster T16000M is a plain looking, but fully featured joystick with 16 buttons, one 8-way POV hat, a throttle slider, and a special hall-effect axis sensor system that provides greater precision than standard analog joysticks. [3]

- *Cost*: Low-mid cost at $65.
- *Control Feature Set*: Exceeds minimum feature set.
- *Ambidextrous Grip*: Yes
- *Reviews*: This joystick gets better than average reviews with most complaints focusing on the joystick feeling cheap while performing admirably.

5.4 Discussion

From a cost perspective, the Saitek Evo wins out, but only barely. The Thrustmaster is a close second, while the X52 is overly expensive at the cost of both of the previous joysticks put together.

All of the joysticks exceed the needed feature sets and are all ambidextrous, so in these categories they're all on an even playing field.

When reviews are factored in, the Thrustmaster is well-regarded whereas the other two get average reviews that aren't too spectacular.

5.5 Conclusion

Once you combine the excellent reviews with the reasonably cheap costs of the Thrustmaster T16000M, it comes out as the clear winner for the joystick category. It is definitely the best performance per dollar. The Saitek Evo is a close second, but the the X52 should not even be considered as an option.

6 References

References

[1] Logitech g saitek x52 flight control system. [Online]. Available: https://www.amazon.com/Logitech-Saitek-Flight-Control-System/dp/B01LY285ZH/ref=pd_day0_147_4?_encoding=UTF8&psc=1\refRID=050PFE0S4QS28T91V2J7

[2] Saitek cyborg evo joystick. [Online]. Available: https://www.amazon.com/Saitek-102989-Cyborg-Evo-Joystick/dp/B0000AW9P1

[3] Thrustmaster t-16000m flight stick. [Online]. Available: https://www.amazon.com/Hercules-2960706-Thrustmaster-T-16000M-Flight/dp/B001S0RTU0

5 Individual Blog Posts

5.1 Chris Pham

5.1.1 Fall

- Week 1
  - Activities
    - Talked to the client to see if I can get into the OSURC Project
  - Summary
    - I went ahead and looked through projects and then found a project that I liked, and then I emailed the stakeholder, Nick McComb, into adding me into his project.
- Week 2
  - Activities
    - Emailed client when deciding to meet
  - Problems / Solutions
    - Client happened to be in California
  - Summary
    - After being given our projects, we emailed our client and talked to our client over Slack to decide when to do our meeting about the project. We decided on Sunday for meeting up, while we think about the time schedule for the next assignments.
- Week 3
  - Activities
    - Finishing Problem Statement
    - Started on Python Mastery 1
  - Summary
    - I started working on my personal Problem Statement and was told from our client that we should do the Python Masteries. The client wanted to use it to prove that someone has enough understanding of the subject.
- Week 4
  - Activities
    - Finished Problem Statement
  - Problems / Solutions
    - Python Mastery 1 is taking forever
    - Looking into alternatives to Code Academy.

  - Summary
    * We worked on our collaborated Problem Statement and I started work on the Python 1 Mastery as suggested, but the actual website is expecting us to be completely new to coding. So in the end, I stopped because I was suffering while doing it.

- Week 5
  - Activities
    * Start on rough draft of the Requirements Document
    * Finished the draft, and it is uploaded to GitHub
  - Problems / Solutions
    * Ended up being sick
    * Could not finish the mastery program yet
  - Summary
    * We worked on our rough draft of the Requirements Document and that is now submitted to GitHub. Client was wanting some changes on the Problem Statement, but we changed the document to reflect those changes.

- Week 6
  - Activities
    * Finish the final draft of the Requirements Document
    * Sent to client for approval
  - Problems / Solutions
    * Issues with making the Gantt chart
    * Might need to start coding soon because the Mechanical team needs code
  - Summary
    * We finished the Requirement Document and now I am thinking we are going to start on the project soon because we need to build parts of it for the mechanical engineering capstone group.

- Week 7
  - Activities
    * Divide up the section of the Tech Review Document into coherent sections
  - Summary
    * After submitting our assignment last week about the Requirements document, we went ahead after our meeting to build our skeleton for the tech review. After build that, I could not do anything else because was finishing other homework and plan on starting more of the assignment during the weekend.

- Week 8

  - Activities
    * Finish up rough draft of Tech Review Document
    * Gave our document to other people to review
  - Summary
    * On Monday, I went ahead a my rough draft for the personal tech review. Then on Tuesday, we went ahead and reviewed someone else's document and got ours back on Thursday. Will probably finish final draft on Sunday.

- Week 9
  - Activities
    * Finished final draft of Tech Review Document
    * Design document going to be started soon after
  - Summary
    * On Monday, went and finished most of my tech review substance wise. On Tuesday, went ahead and tried to make it more readable.

- Week 10
  - Activities
    * Finish working on Design Document in parts
    * Finish working on recording
  - Problems / Solutions
    * Everything ended up being due on Friday
    * Had to save this for last
  - Summary
    * We started working on our design document on Thursday and then completed most of it during Friday because of lack of time and things like that. The video was done around that time too, couldn't do it in my house early in the day because of noises.

5.1.2 Winter

- Week 1
  - Activities
    * Created a working schedule because we have to work on campus for direct access
  - Problems / Solutions
    * When thinking about implementing my area, was thinking of live tiling, or pre-stitched image after.
  - Summary

        * After everyone got back from break, after class on Tuesday, we make a work schedule for us to work on the Rover. I was assigned the Mapping functionality of the ground station and some of the design decisions are up to me to choose.

- Week 2
    - Activities
        * Start on turning last year's code into Object-oriented instead of procedural for ROS
    - Problems / Solutions
        * Issues with big functions and converting them into stateful functions
        * Issues with downloading and speed of combining images into a single image
    - Summary
        * I started on working the mapping core but I need to make the UI element and then try to make some speed improvements against PIL
- Week 3
    - Activities
        * Working on adjusting maps in ways (Zoom/Scaling, Movement)
        * A smaller image can be cut out of the bigger image now
    - Problems / Solutions
        * PIL zoom does not work like I expected, might have to use QT's zoom
    - Summary
        * Mapping is working mostly now with now a moving bounding box that goes in the big image and cuts what is needed.
- Week 4
    - Activities
        * Map stitching works up to 3 miles before skew happens
        * Got items to place correctly onto the map and display correctly
    - Problems / Solutions
        * Somehow got issues with converting latitude and longitude into x, y coordinates gave me extreme values but got solved by pasting it again with a different name...
        * I was not able to do much work during the week because of the issue above
    - Summary
        * In the end, I got mapping to work correctly to around 3 miles before my projection becomes skewed enough to become really inaccurate. I also got a way to map waypoints to a system, however that is my next problem because I actually have no way of removing them from the map itself. I think I might

make another image to lay on top of the map and then clear out the box by doing the same command but making it transparent instead of coloring it.

- Week 5
  - Activities
    - Implemented my mapping to work with the Qt interface
  - Problems / Solutions
    - Had issues with signals and slots for my listener because it would stall out when trying to paste the image, had to create another thread to handle that.
    - Had slight issues with trying to merge my branch because the repository's folder structure had changed so much
  - Summary
    - This week I was trying to setup my class to properly implement with my Qt Application. That was real questionable but I think I can fix it using some type of class constructor call in an Qobject instead of instantiating it from the map object itself. Another thing I did was integrated my fork/branch into the main branch and it looks way better now. Feels pretty good actually.
- Week 6
  - Activities
    - Worked on displaying the map correctly without massive hangups
    - Worked on our capstone paper and video
  - Problems / Solutions
    - Issues with scheduling for working, but in the end we worked on our own mostly and then worked online when we needed to.
  - Summary
    - Worked on the poster, video, and displaying images on the GUI. Productive I say.
- Week 7
  - Activities
    - Nothing
  - Problems / Solutions
    - Was way too busy with midterms
  - Summary
    - I did nothing at all. Feels bad man.
- Week 8

  - Activities
    * Worked on some GPS/Waypoint handling for the map
    * Helped with recording for the club
  - Problems / Solutions
    * Overlay not working correctly, need to fix paste functionality
  - Summary
    * Worked on the GPS and waypoint system, and for the club itself, I helped with filming and recording for the system reviews report needed.

- Week 9
  - Activities
    * Worked on an correctly producing my indicator
    * Fixed on pasting the overlay image
  - Problems / Solutions
    * Was busy during Monday-Wends with plumber issues needing me to be home
  - Summary
    * Fixed up my mapping and started working on integrating a waypoint system

- Week 10
  - Activities
    * Built a intermediate system to handle button presses for way-point lists.
  - Problems / Solutions
    * QT is really finicky with some function names and changes from Qt4 to Qt5 made some things harder to find
  - Summary
    * During the start of the term, I personally did not work on much because of issues with registration However, we did work out when we are expecting to work and meet up this term, which would be on Thursdays around 10 o'clock in the morning.

5.1.3 Spring

- Week 1
  - Activities
    * Meet up with the group to plan meetup times
  - Problems / Solutions

    - Issues in trying to get into classes
  - Summary
    - After meeting up and trying to find
- Week 2
  - Activities
    - Finish integrating the GUI elements like buttons
  - Problems / Solutions
    - Issues with DMS conversions
  - Summary
    - This week, I worked on completing the integration of the GUI. It turned out last term, that the East/West portion of the DMS conversion was not being set correctly. I spent a bit of time trying to get the correct float value to go down but then my minutes was completely off to the calculated DMS of the GPS location. Turns out that the box that contained the DMS location, would be limited to 0, 90, which actually needs to be 180 because it needs to span a half circle. Once that value was changed, the values were being set correctly.
- Week 3
  - Activities
    - Finish up linking up buttons and interactions in the GUI
  - Problems / Solutions
    - More issues with trying to get into the negative values of the DMS portion
  - Summary
    - Turns out that I forgot to set the sign of the longitude and latitude to their corresponding North/South and East/West values. That was easily done using with many possible implementations like equalities, sign function in python, and other things like that.
- Week 4
  - Activities
    - Finish up and submitting the poster
    - Edit documents to fit new the new competition
  - Summary
    - This week, I was too busy to actually do much, and instead we worked on updating our documents to the new competition that we're going to now. And when that was done, we all just signed it and gave it to our client to sign. Also finished poster and it was sent to student media services.

- Week 5
  - Activities
    - Finish working on multithreading and inputs.
    - Work on Midterm Report
  - Problems / Solutions
    - Was getting rate limited, needed to use a semaphore to limit thread usage.
  - Summary
    - During Week 5, I worked on two important things, multi-threading the downloading of the image files from Google and integration of inputs into my mapping system. I was hitting issues with trying to use a pool but could not do it, in the end I needed to use processes. This in the ended up needing to be limited with semaphores so downloads could be regulated.
    - I also have started on integrating the GPS data that we are getting and applying that onto the map system that I've created. One problem is I forget how the data is formatted in the system and I need direct access to view the data that is being sent by the rover to parse correctly.
- Week 6
  - Activities
    - Did not plan to do anything for Capstone at all.
  - Summary
    - Did not do anything pertaining to Capstone at all.
- Week 7
  - Activities
    - Present at Expo
    - Hunt for items that we need for Expo
  - Problems / Solutions
    - Someone actually destroyed some motor nodes, so only 4 or the 6 wheel positions only move.
    - Corwin in the end, converted the connectors to USB and then connected those via USB.
  - Summary
    - We met up on Thursday to find out what we actually need for Expo and what will be provided for Expo or not. It turned out that a Electrical team member blew a series of driver nodes, and had to build something to replace it. One method was making an emulator, or just run a newer number of nodes if the complete system cannot boot. It only turned out the controller boards only got fried, and so it had to get rigged to run off of USB. The day of Expo, we went to our position, and realized that we only had one table shared with the other Mars Rover Team Capstone project, and had to find the

coordinator to get us another table. That was resolved, and then we went ahead and got all the stuff from the club room, and brought it to our table and presented.

- Week 8
  - Activities
    - Nothing at all
  - Problems / Solutions
    - With nothing working correctly, because of the blown electronics, other interfaces could not be used like the GPS information.
  - Summary
    - Nothing could be done, with the lack of integrating parts from the rest of the rover team.
- Week 9
  - Activities
    - Nothing at all
  - Summary
    - Did nothing, because of other homework assignments taking my time.
- Week 10
  - Activities
    - Started and finished the final video report and started on the final paper.
  - Summary
    - We started our video on Thursday and finished on Friday, and then started working on this paper over the weekend. We are expecting to finish this whole document around Tuesday afternoon.

5.2 Ken Steinfeldt

5.2.1 Fall

Week	Day	Post
1	Tuesday Wednesday Friday	Learned about capstone. Considered capstone projects. Decided on a capstone project.
	Summary	Introduction to Capstone. I spent most of the week getting my feet wet and understanding the project ahead of me.
2	Tuesday Sunday	Learned of project assignment. Met group members. Got the project I wanted. Very excited. Met with client 2:30 - 3:30 to discuss requirements and scope of project.

	Summary	Struggled with LaTeXmakefile. Still don't really understand how to make it go. Big part of this week was learning about capstone project assignment. Met with client for first time.
3	Monday	Wrote and submitted project problem statement document.
	Summary	Wrote Problem Statement. It was difficult to grasp the entire project so early but with help from the client and group members it went smoothly enough. Good learning experience for getting to know the project and what we were getting ourselves into. Going forward we will need to become more acquainted.
4	Summary	Weekly meeting set for Tuesdays at 3:50 starting next week.
5	Tuesday	First meeting with TA Andrew Fielding at 3:50.
6	Tuesday	Weekly meeting with TA Andrew.
7	Tuesday	Weekly meeting with TA Andrew.
	Summary	Requirement document individual assignments made and agreed upon.
8	Tuesday	Weekly meeting with Andrew.
	Summary	Worked on requirements document rough draft.
9	Tuesday	Weekly meeting with Andrew.
	Summary	This week is defined by the requirements document final draft.
10	Tuesday Wednesday Thursday Friday	Weekly meeting with Andrew. Short meeting, we don't have much to discuss. Everything is well. Design document is created on Overleaf for group to work on. Work on design document, research and some writing. Finish up design document.
	Summary	This is the final week of classes. Thursday class is an optional workshop. On Friday the design document is due. Team works on document all day. The document feels rushed and we're a little overwhelmed with it. We also have a difficult time figuring out roles and ownership as no code has yet been written.

5.2.2 Winter

Week	Day	Post

1	Tuesday	First day of class. First time group meets since winter break • We meet in our lab • Discuss current state of code base • Discuss rover progress – Current state – Issues, etc. • We set group work times for the term – Tuesday and Thursday all day (more or less) – First group work day will be Thursday 1/11/18 • Group needs to complete ROS tutorial before next work day • Winter term TA meeting time has not yet been determined – Will likely be on Tuesday or Thursday as group will be together on those days
	Summary	First week of class after winter break. Group reviews state of current code base for the project. Group meeting times are determined by group. These are Tuesdays and Thursdays.
2	Tuesday	Meet with group to work on project all day. Begin writing StatusCore module that will grab system status information from the rover and display it to the user. The rover will create publishers for the necessary statuses. StatusCore module creates subscribers that read from the rover publishers. Minimal processing should be done on the information coming from the rover.
	Summary	Week spent primarily getting started with coding. Met with group at OSURC club meeting on Saturday.
3	Tuesday Thursday Friday Saturday	Met team at 10:00AM to work on system statuses We reach out to project leader Nick McComb for approval of our capstone documents. Nick responds later in the day with approval. Hear from TA Andrew to setup this terms weekly meetings. Corwin responds with the preferred group meeting times which are 9-11 and 1-4 on Tuesdays. Unable to make the OSURC Mars Rover team meeting this Saturday due to not feeling well. Work on system statuses from home. I am unable to finish SensorCore.py this week like I had hoped. However, I get a basic mockup going and generally make progress. Repository is cleaned up and reorganized for clarity between different rover teams. This is much needed and well done but causes me difficulty as my branch becomes significantly diverged from the main branch. I am unable to quickly fix the divergence and will have to do so next week.

	Summary	Goal is to finish system statuses by the end of this week. Not feeling well this week.
4	Tuesday	TA meeting with Andrew Learn about PyQt framework in order to create UI layout. It is more complicated than I thought it was.
	Thursday	Implemented clock into StatusCore and tested that it worked - this was one of the ways I attempted to learn the PyQT Began working on the stopwatch module for the ground station. This is not as trivial as I thought it would be.
	Saturday	Went to Corvallis for OSURC Rover team meeting. Worked with team in lab after meeting. Got stop watch working as an individual module, but not integrated with StatusCore. Includes push buttons, but needs to be left - right mouse clicks for the final product.
5	Tuesday	TA meeting with Andrew Midterms galore this week. I manage to design a very basic visual framework for the UI.
	Thursday	Visual framework is shown to team members and is not liked. I agree with the criticism and the UI for SensorCore is redesigned with the group. Instead of being text based it is now boxed labels that are either red, orange, or green depending on the values that are given. For simple on/off alerts, such as various module connectivity, the box is simply red or green. Red - something is wrong - usually 80% and greater (if applicable) or a False boolean value Orange - warning, approaching read usually 70%-80% (if applicable) Green - everything is great or a True boolean value
6	Tuesday	TA meeting with Andrew. Meet with team in Corvallis to work in lab. Group picture is taken and poster rough draft is created. We are happy with the poster and don't foresee a large redesign in the future.
	Thursday	Group works on midterm report for the class. Report is written together and videos are recorded. Videos are then stiched together.
7	Tuesday	TA meeting with Andrew. Nothing of note • Team gives status update • Nothing of concern • Team gets back to work on ground station software • Code base cleanup
	Thursday	A little stymied by system status module on the rover side. I help out with some random things while waiting for rover team. It's not a big deal.

	Summary	This week is primarily dominated by the need to produce the SAR video for the competition. If this video doesn't get done the rover cannot compete. This takes up a lot of the group's work time but work still gets done on the ground station. Work is done on the joystick and Corwin integrates the nimbro packages to cutdown on latency between the ground station and rover.
8	Tuesday	TA meeting - All is well. Project is progressing. Work with rover team to get system values from rover StatusCore values (green, orange, red) decided.
	Thursday	Meet team in Corvallis to work in lab. Corwin has created scripts to ease the build process of the ground station. I sit down with Corwin to learn how to use these new scripts. Work on StatusCore now that rover statuses are available for final integration.
	Summary	A lot happens in week 8 • Repository is reorganized • New startup script is written for ground station – This greatly helps development • Create some more placeholders for statuses in the UI • Rover drives!
9	Tuesday	TA meeting. All is well and progressing. Do not work from lab, will meet with team on Thursday. StatusCore: • Work on callback functions – pickup rover statuses, evaluate their return functions • New system statuses added to rover – System statuses are still likely to change • Needs to ask for update from rover on startup – Peripheral systems, such as cameras, will not automatically update until a change has happened. I will need to poll the rover at startup to get initial status – This requires a subscriber on rover and a publisher in StatusCore

	Thursday	Met with team in lab. Begin push to meet Beta by week 10. StatusCore work: • Finalize UI look in QT designer • Properly lable/organize UI elements • Prepare for main branch integration
	Summary	I make some serious progress on StatusCore this week, plan to fully integrate into ground station next week to meet beta requirement. • StatusCore look is (somewhat) finalized • Rover team gets their status module sending signals I can use • I clean up StatusCore in preparation for integration next week
10	Tuesday	Worked late to finally integrate StatusCore with main branch. StatusCore module hit beta status and, with the help of Corwin, integrated into the project. Some bugs were noticed, namely some flickering in the status alerts. The source of the bug was identified but not fixed. • Problem: Did not think StatusCore would require the use of Slots • Solution: StatusCore needs to use QT slots in order to refresh nicely
	Thursday	Last class of the term After class we worked in the lab. I hooked up the clock and fixed the StatusCore bugs that were identified on Tuesday
	Summary	Final week of the term • No TA meeting this week • Finally got StatusCore integrated into ground station – Very successful, only a few bugs • Some bugs exist in StatusCore and are squashed (as far as we know) • Ground station hits beta! All in all a very exciting week. Everyone is happy with our progress.

5.2.3 Spring

Week	Day	Post
1	Tuesday	Code review of Spring break work on Rover and Ground Station.
	Wednesday	Class Meeting.
	Thursday	Team meets and goes over project. Change in venue discussed and new meeting times and requirements are discussed and agreed upon.

	Summary	This is the first week of Spring term. First class meeting. First week after Spring break so the team meets to go over the status of the project and any of the changes that occurred during the last two weeks. The Rover team is no longer going to compete in the University Mars Rover Challenge, but instead will now compete in the Canadian International Rover Challenge. Change in competition means that that some minor changes in requirements are needed, these changes are discussed. The team also reviews the general status and progress of the project. New meeting times are discussed and agreed upon for the coming term.
2	Tuesday Thursday	Code review. Team meets. Changes to design document are discussed. Work will need to be finalized and signed off on by TA Andrew. StatusCore module changes will be small. Mostly subscribing to new publishers. Will need to wait for Rover team to get publishers up and running. May be most beneficial for me to do as much as I can to help Rover team.
	Summary	Group deals with EXPO bureaucracy. This entails registering, requesting necessary equipment/power and signing the proper release forms. Changes in competition requirements again discussed as they will require that the design document be amended to match the changes in software. The changes are minor but are changes none the less. Changes to StatusCore module look to be minor.
3	Tuesday Wednesday Thursday	Code review. Rover and Ground Station. Work on stubbing out subscribers. Wired Article due. Team meets. Rover team expected to be ready to broadcast new statuses by the end of the week. Will work on them on Saturday.
	Summary	Beginning of week three the Rover team is still not ready for new statuses. This is not unexpected considering the quick change of design requirements. Despite the lack of broadcasted statuses I can and do begin working on the foundational subscribers, etc. so that when the Rover team is ready I can pick them up in StatusCore. Some Statuses are also deprecated as they will no longer be used. This includes Bogies. New subscribers are stubbed out. WIRED! Article is due this week.
4	Tuesday	Code Review. StatusCore work.

	Thursday	Team meets. EXPO poster is finalized, approved, and submitted. Final changes to requirements document made and approved.
	Summary	Continue to work on changes to StatusCore. Final poser is due this week so it is finalized and submitted after being approved. By the end of this week we deem software to be EXPO ready. Amendments to project requirements are made, agreed upon, and signed-off on. This includes all team members as well as client/stakeholder.
5	Wednesday Thursday	Class meets. Team meets, works on Spring midterm report/presentation.
	Summary	This is week 5 and is mid-term season. Every member of the team is swamped with work. I am not able to get very much done on the ground station software. Team meets and works on Spring midterm report. Report and presentation are finished by the end of the week (of course).
6	Summary	Not much capstone work. Midterms and projects in other classes.
7	Wednesday Friday	Class meeting to prepare for EXPO. EXPO! Goes great. Our team does really well and has a lot of fun.
	Summary	Expo happens this week.
8	Summary	Did not work on ground station.
9	Summary	Did not work on ground station.
10	Thursday Friday	Final video work begins. Final video presentation completed and submitted.
	Summary	Team is all busy with finals and final projects. Final presentation is due and completed by the team.

5.3 Corwin Perren

5.3.1 Fall

- Week 1
  - Week 1.3
    - Looked at and selected projects for my preference submission.
  - Summary
    - Looked through potential projects and submitted project preferences. I also contacted and got a positive confirmation from Nick McComb, the stakeholder for the OSURC Mars Rover capstone project, that myself and Christopher Pham were requested to work on the project via an email to McGrath.
- Week 2
  - Week 2.2
    - Took class notes

  - Week 2.3
    * Group wrote an email to send to our client, Nick
    * Sent said email to Nick
  - Week 2.7
    * Had a meeting with Nick McComb, Kenneth, and Chris to covered the design guidelines document
  - Summary
    * Met with group members Chris and Kenneth, where we exchanged contact information and coordinated contacting our client to set up a meeting. We made contact and set up a video conference call for Sunday to cover project details. We also were added to the rover team's slack, which we'll use for primary communication in the future. For future use, we were also added to the club's GitHub account.

- Week 3
  - Week 3.1
    * Worked on and completed the rough draft problem statement
    * Did some initial research on the tools and frameworks our team will be using for this project
  - Week 3.2
    * Started communication as a group with our TA to set up a weekly meeting time.
  - Summary
    * After communicating with our client on Sunday, I took Sunday night and Monday to work on and complete the problem statement document. I also began to look more into the tools and frameworks we will be using on this project. Our group also started emailing our TA, Andrew, to set up a weekly meeting time, which is still not set in stone as Andrew is having a hard time scheduling so many groups at once. So far, it looks like they'll probably be on Tuesday afternoons as a remote session.

- Week 4
  - Week 4.4
    * Made revisions to the problem statement document
  - Week 4.5
    * Uploaded the final version of the document to the Github rep
    * Emailed client asking for revisions, and an approval email once ready
    * Embedded PDF in the group OneNote
  - Summary
    * Our group used overleaf to group edit a join problem statement document, made revisions, and then emailed our client the final version. We're currently waiting for the approval email. I also embedded our PDF into the group OneNote per the follow-up email from Kirsten

- Week 5
  - Week 5.2
    - Had TA Meeting
  - Week 5.4
    - Worked on design requirements document with Chris and Ken
  - Summary
    - We had a general meeting with our TA, the first of the term. Covered what the TA's role will be in our group. Covered that we seem to have a good plan on what our group will be doing. On Thursday, worked on the design requirements rough draft.
- Week 6
  - Week 6.2
    - Had an update meeting with the TA
  - Week 6.3
    - Worked on design requirements document with Chris and Ken
  - Week 6.4
    - Continued working on design requirements document with Chris and Ken
    - Sent the semi-final draft to the client for potential edits
  - Week 6.5
    - Made client edits to requirements document
    - Sent approval request email to client
  - Summary
    - We had a general meeting with our TA and otherwise spent the week working on our requirements document. We sent a semi-final draft to our client Nick, who requested some minor edits, which we made and then sent him a final draft for approval.
- Week 7
  - Week 7.2
    - Decided on projects to work on
    - Had TA meeting
  - Week 7.4
    - Set up the ground station hardware in Graf 306
  - Summary

        - Last week, we had a short quick meeting with our TA and then worked with each other to determine who would take what technologies for the tech review document. On Thursday, I set up the hardware our team will be running the ground station on set up in Graf 306. This included an Intel NUC, two joysticks, and two 1080p monitors. For now, it's connected via and ethernet switch to a desktop emulating the Rover.

- Week 8
    - Week 8.1
        - Worked on rough draft for the tech review document
    - Week 8.2
        - Did peer revisions on tech review documents in class
        - Turned in tech review
        - Met with TA
    - Week 8.5
        - Worked on setting up the ROS framework on ground station software
        - Got simple rover drive control data sending to the rover
        - Got three video streams showing on the gui from the rover
    - Summary
        - Last week, I finished my peer review, did an in class peer review, and turned it in. Our group also had another short meeting with our TA to make sure we were doing fine. On Friday, I began writing some initial ground station code. I got simple rover drive data sending over the network to the Rover. I also got three camera streams displaying in the GUI.
- Week 9
    - Week 9.1
        - Worked on the final draft for the tech review
    - Summary
        - I finished the final draft of my tech review on Monday night and submitted it mid-day Tuesday.
- Week 10
    - Week 10.2
        - Meeting with TA
    - Week 10.3
        - Created latex template doc on overleaf
        - Added initial content
    - Week 10.4

        * Worked on final draft of the design document
    - Week 10.5
        * Worked on final draft of the design document
        * Turned in the final draft of the design document
    - Summary
        * I worked on and finished the final draft of our group design document and turned it int. We also had a 60 second meeting with our TA this week.

5.3.2 Winter

- Week 1
    - Week 1.2
        * We went and checked out our lab space in Graf 306
        * A work schedule was made to work at least five hours on Tuesday, Thursdays, and Saturdays, with the Saturday work overlapping with the Mars Rover weekly meetings
        * Planned to start actual development work Thursday
    - Week 1.4
        * I filled out Asana with the core tasks that our software need to produce
        * Initial tasks were divvied up
        * I began continuing my work on processing and displaying video data
    - Summary
        * Chris, Ken, and I met up after class on Tuesday so I could show them our lab space in Graf 306. We then came up with a work schedule for the term where we work on Tuesdays, Thursdays, and Saturdays for five hours a day at minimum. Then on Thursday, we all began actual work starting with creating tasks to complete and assigning initial tasks to everyone. I then continued working on the video display systems. I made the processing and display code slightly more efficient, and also added stream labels and fps counters directly on the video stream. On Saturday, I plan to add in auto adjustment of the stream quality based on whether the fps values are too low. Conversely, if the fps is pegged at the maximum, the quality will be increased until it starts to drop.
- Week 2
    - Week 2.2
        * Continued working on video systems
    - Week 2.4
        * Began refactoring video receiving class after verifying that the core class worked well.
    - Week 2.6

    - Began refactoring the core of the ground station launcher file and restructured the program folder to have a cleaner layout.

  - Summary
    - Over that week, I mainly worked on the video receiving class, testing it to make sure it was reliable and could show enough video streams at a smooth frame rate. Once that was verified, I began refactoring the class into a new version that would be able to include turning on and off video streams and adjusting what streams were shown in what display box. Later in the week, I began refactoring the ground station launcher file and folder structure so we'll have a cleaner base to work with as the project progresses.

- Week 3
  - Week 3.2
    - Finished refactor of the launcher for the ground station
    - Added in check for if the rover is on the network when the software starts
    - Continued refactor of the video classes
  - Week 3.4
    - Worked on adding advanced features of the video class refactor.
    - Added sleeping of video streams
    - Made number of cameras and resolutions dynamic
    - Can handle smart changing of what video displays are in each GUI element
  - Summary
    - Over this week, I finished the refactor of the launcher for the ground station software. This included an addition to keep the program from launching if the Rover is not present on the network. Also, there is now a shared object that can be passed to all the important classes in the program to allow for easy interfacing with signals from other classes and the GUI. I also made significant progress on the Video class refactoring. At the moment, the GUI can now display a dynamic number of video streams, and also allows for dynamically changing which stream is displayed in each video display element. I've also added the ability to disable video streams where it now actually stops pulling in the video data to lower bandwidth usage. This includes "sleeping" video streams that aren't explicitly disabled, but are not actively being shown. Lastly, I finished the core of the GUI layout using QT Designer so the core widget modules now match the layouts defined in our documents from last term.
- Week 4
  - Week 4.2
    - Continued work on video systems
    - Assisted main software team with their GPS and IMU firmware, as well as creating a ROS node to broadcast said data. Goal is to get GPS data sooner so Chris has that data to work with.
  - Week 4.4

        * Assisted with firmware programming for drive control micro-controllers
        * Assisted on work on the motor controllers drive nodes for ROS so our team can have a node to send drive commands to.
    - Summary
        * This week was mostly spent helping with development on software packages that will help support the ground station software on the Rover. Our group is getting to the point where having working ROS nodes on the Rover and hardware to control is getting more important, so I decided to help speed up the process. I worked on firmware for the micro-controllers that processed raw GPS and IMU data as well as the motor controller. I then also helped write the ROS nodes to interact with these micro-controllers so they could be controller using normal ROS topics.
- Week 5
    - Week 5.2
        * Took a break from capstone software to assemble revision 1 of the Mars Rover Iris control board
        * Spent 10 hours assembling Iris
    - Week 5.4
        * Spend 7 hours testing and fixing electrical problems with the Iris board
        * Ending board had full functionality on the 12 communication ports
    - Summary
        * This week was pretty light for capstone as I had midterms. Additionally, I decided to take some time to help the Mars Rover electrical team, and assembled their Iris central control board. Further development of the ground station will be greatly helped by having missing hardware finally connected to the Rover to talk to, so I figured spending some extra time helping this to happen would be useful for the team overall, and then by proxy, our capstone team. Assembly, testing, and modifications took 17 hours. I also wrote a simple motor driver node for ROS so the motor driver board could be tested using control from ROS topics.
- Week 6
    - Week 6.2
        * Worked on and completed rough draft of expo poster
        * Helped Chris with mapping integration
        * Wrote simple joystick control file to be run on ground station
        * Added UI elements
    - Week 6.4
        * Worked on midterm report
    - Summary

        * This week was spent mostly working on administrative stuff for the capstone project. A rough draft of the expo poster was made, and the later part of the week was spent working on the midterm report and video. However, I also did create a simple test file that takes in joystick data on the ground station, and broadcasts it to a ROS topic, allowing for remote driving of the motor connected to the Rover through Iris.

- Week 7
    - Week 7.4
        * Helped with SAR work
        * Made multi-threaded controller code.
        * Tested nimbo_sender / receiver
    - Summary
        * This week I helped out getting things ready for the Rover systems acceptance review, one of the pivotal turning points for the rover team. This involved cleaning up code and getting remote drive systems working. As part of this, I re-wrote the joystick control test file to be multi-threaded and also got drive working with nimbro_transport, which allows for UDP sending / receiving of ROS topics for packet loss allowable topics such as drive and video.
- Week 8
    - Week 8.2
        * Continued work for SAR deadline
        * Wrote drive sender so that sent control data matched the refactored drive_receiver on the Rover
        * Joystick control speed limiting using throttle axis, shows speed limit and tank output on GUI.
        * Tuned drive sender until remote drive over radio was smooth
        * Helped with refactor of whole GitHub structure
        * Needed for easy access to shared ROS packages due to nimbro addition
        * Made major updates to GUI design, moved closer to matching design document.
        * Arm joint position placeholders
        * Rover status placeholders
        * Added pause feature to drive controller, starts in pause by default
        * Added mock compass image to the GUI, for show during SAR video
        * Tested remote drive and video with no tether! Drove around parking lot from ground station.
    - Week 8.3
        * Helped create videos for SAR demo
    - Summary
        * Due to the SAR deadline being March 2, I put in extra time this week to make systems work as well as they could for the demo video made on Wednesday. For the demo, we wanted at minimum remote Rover drive working, video systems buttoned up and working well. On this note, wrote a

joystickcontrolsender class to match the new rover_drive system implemented on the rover. This code was the finalized version of the testing joystick code I'd written. I also added the ability to use the throttle stick on the joystick to act as a speed limit for the drive commands sent. So a setting of 50% would scale the full joystick range to only 50% speed. Additionally, a pause feature was added so that the rover cannot be accidentally controlled when the GUI first starts up. A button on the joystick disables pause state, and can be used to toggle the state while the gui is running. Also, gui elements on the right monitor were updated to show the speed limit value and the current power output to each of the sides of the rover in tank drive form. After making these changes, I tuned update rates and nimbro transport rates to get driving smooth again.

Due to the addition of nimbro_network, I refactored the GitHub software repository to make it easier to share packages between the rover and ground station software. This was needed so that both had access to custom designed message formats in ROS. I also worked a lot this week to update the look of the GUI to match our design document. Placeholders images or Qt GUI elements for arm joint positions, rover statuses, and the compass were all added.

On Tuesday night, I and the team's mechanical lead, Ben, tested actual real remote drive of the Rover. We placed the rover in the Graf parking lot where he followed it with a kill switch tether, and I drove the rover over the remote radio link from the 3rd floor of Graf with the antenna sticking out the window. Tests showed that overall everything worked well! Driving was mostly smooth and video responsive enough to feel comfortable driving. Then on Wednesday, we remotely drove the rover on the catwalk of 3rd floor Graf and took videos of both the Rover driving, and of me controlling it at a now largely flushed out looking ground station.

- Week 9
  - Week 9.2
    - Tuned some parameters of the drive controller so that driving was a bit smoother after remote drive testing the previous week.
    - Rewrote video_receiver and coordinator to include new topic to control which video resolution is enabled, side effect of nimbro
    - Added auto resolution adjustment of video_receivers
    - Tested new end effector video stream
  - Week 9.4
    - Added joystick control sending and receiving for selection of cameras in GUI
    - Helped Chris integrate full map system into master branch
    - Added the ability to move compass, just spinning never-ending in one direction.
  - Summary
    - After testing real remote driving the previous week, I tuned parameters on the rover and the ground station to get remote drive working just a little bit smoother and with no dropouts. After that, I worked on rewriting my video_receiver and video_coordinator threads so that instead of simply opening the

designed video stream, the ground station would tell the rover which stream to send. This was a side effect of using the UDP nimbro_network transports. Because UDP continually spams data at the ground station, it was in effect sending all the different resolution steams to the ground station all the time. This extra bandwidth was definitely something the team didn't want. During the rewrite, I also added automatic resolution adjustment of the video_receivers based on averages of the received frame rates.

I also tested and integrated receiving and showing the fourth and final video stream from the end_effector, which worked as intended. Receiving of rover_statuses was also integrated into the roslaunch file and I tested receiving those statuses from command line. I ended up adding and additional manual update message for that package on the rover so the ground station could request and initial state when the software first starts up, and tested this via rostopic publish from the command line.

As I'd gotten the joystick controlclass cleaned up the previous week, I also wrote and tested joystick control of which video streams were shown on the three gui elements while I was rewriting the receivers and video coordinators. Two sets of two buttons each on the joystick now cyclically change which element is selected via an orange ring around the outside as well as what camera stream is current shown. The trigger on the joystick disables the currently selected stream.

I also helped integrate Chris' mapping system into the master branch of the rover software before ending the week by writing the threaded class needed to make the compass spin. I ended by making the compass spin in one direction indefinitely.

I sat down in the middle of the week and helped the mechanical team lead determine the final design for the quick deployment ground station box, which he then began fabricating.

- Week 10
  - Week 10.2
    * Made nice new compass image asset as vector image in Inkscape, exported as PNG
    * Compass now can respond to heading updates (once we get them). For now, left/right clicks change heading
    * Heading and next goal heading update
    * Arm joint positions spin/stop when clicked. Placeholders.
    * Help Ken integrate his system statuses with the GUI
    * Placeholders for waypoint entry/editing
    * Wrote test code to get info from the M2 radios, set channel
    * Refactored M2 radio test code into one module to set radio channel from GUI, added GUI button and setting. Tested and works.
    * Helped Ken debug thread context issues updating GUI
    * Added pitch and roll indicators for Rover for when we get that data
  - Week 10.4
    * Made minor changes to statuses so that it looked nicer
    * Added OSURC logo to top left corner of GUI per Nick's request

    - Made numerous UI changes to help clean up look.
    - Refactored joysticksystems to inputsystems
    - Added spacenavsystems to read in spacenav mouse data

  - Summary

    - This final week was spent mostly dealing with non-critical updates to the look and feel and glue logic for the GUI. I made a brand new compass asset in inkscape that looks like a real compass and exported it as a high resolution PNG to be imported in the GUI. I then finished off the compass class allowing for the compass to rotate to a heading if one is received from the Rover. As the rover is not currently broadcasting this data, left and right clicking on the GUI element causes it to add or subtract five degrees from a mock heading, whose update is then shown. Additionally, I also made the textual readouts show the current heading to prove we have control of them. I did something similar with the arm joint positions by making them all spin when left clicked and stop spinning when right clicked. These are placeholders until we get arm data later next term.

      Ken got his system statuses class ready to integrate into the main GUI and launcher, so I helped him do that and make the minor changes needed to make his thread start and stop properly. I also helped him debug a thread context issue that was causing weird flickering on the status display elements. Afterwards, I added the gui elements for manually entering GPS coordinates for adding to the waypoints listings.

      I wrote test code to interact with the Rocket M2 radios we're using to get the status data we'll end up needing, such as signal strength, channel, and throughput. I also got test code working to set the 2.4GHz channel that the radios are set to. Afterwards, I wrote a class to handle the channel updates, made gui elements to control it, added this class to the GUI, and tested it. Now, a channel from 1-11 can be set from the GUI. After pressing the button, radios are reconnected and data returns after 4-10 seconds.

      Near the end of the week, I made mostly graphical changes to the GUI such as adding and OSURC logo to the top left corner, and adjusting spacing and layouts to be more pleasing. I ended this week by refactoring the joysticsystems package into inputsystems before adding the spacenavsysystem class. I currently have this class reading in and storing SpaceNav mouse data in a class variable, waiting to be broadcast to appropriate systems once they're ready to accept the control data.

      The mechanical team also showed us the assembled quick-deployment box, which should be ready for painting and then hardware install in the next few days.

5.3.3 Spring

- Week 1

  - Week 1.2

    - Talked with group mates over slack about what needs to be completed this term
    - Generally talked about the fact that the team was changing competitions
    - Didn't work on the project as everyone was busy

  - Summary
    * Chris, Ken, and I chatted over slack about what needs to be done over the term. Not much is left, and a lot of the work is waiting on Rover hardware. Also, we talked about the fact that the Rover team has switched to the Canadian International Rover Challenge.

- Week 2
  - Week 2.4
    * Did a work day on Thursday, I worked on SpaceNav mouse integration.
    * I registered the team for EXPO
    * I also dealt with sending in expo release forms
    * We made a list of all tasks left to be completed
    * Went over design requirements and found a few that need to be modified due to Rover design changes over the year.
  - Summary
    * We all met on Thursday for a work day. I worked on continued integration of the SpaceNav mouse, getting values normalized and broadcasting to Chris could use it to pan the map. I also registered the team for EXPO and dealt with sending in the media release forms for our team. I also went over the design requirements document with Chris and Ken and found some elements that need to be changed as they're either no longer pertinent, or have changed enough that it would be better to update their descriptions. We'll be making these changes in the following week. Part of this was looking at the differences in the rules for University Rover Challenge vs the Canadian competition the team is now going to.

- Week 3
  - Week 3.4
    * Worked on Rover side software to help get more statuses for Ken
    * Made new and worked on new rover package rover_arm to get rover arm integrated with ROS
  - Summary
    * This week I worked on Rover code rather than ground station code as we're now mostly waiting on features of the Rover to be finished before we can implement the appropriate ground station side code. This involved me getting rover battery status monitoring working and sending to the ground station as well as helping another member get GPS statuses doing the same. I also made and began work on the Rover's rover_arm package to interface with the IONI motor controllers used to move the arm. I got a simple package compiling with both the ROS packages and simplemotion library and got an arm motor moving with it.

- Week 4
  - Week 4.2

        - Finished final draft of poster
    - Week 4.6
        - Worked on and submitted a revised version of our design requirements document.
    - Summary
        - This week, I worked with my team to make the final changes to our poster for expo. I then submitted the poster for printing. I also worked with my team to make modifications to our design requirements document to reflect the team's change to the Canadian International Rover Challenge. We got the revised document signed off by all group members and the stakeholder and emailed it to the professor, TA, and stakeholder.
- Week 5
    - Week 5.6
        - Worked on midterm progress report
    - Summary
        - This was a busy midterm week, so I just worked on the midterm report and presentation.
- Week 6
    - Summary
        - No work was done this week
- Week 7
    - Week 7.5
        - Went to expo!
    - Summary
        - Our team was at expo! Things went well! The demo was smooth.
- Week 8
    - Week 8.6
        - Got the ground station controlling the pan/tilt node
    - Summary
        - I finished the firmware for a prototype pan/tilt node and wrote the code for the GUI to control it using the joystick. Tested and worked!
- Week 9
    - Summary
        - No work was done this week

- Week 10
  - Week 10.2
    - Started final report document
  - Week 10.4
    - Worked on final video presentation.
    - Recorded my portion of presentation.
  - Summary
    - This week, I've worked on the final video and started the final report document to finish off capstone.

6 POSTER

OSURC Mars Rover Team

- The OSU Robotics Club is a sponsored student organization at OSU with over 150 members working on numerous projects.
- The Mars Rover Team of the Robotics club has requested a system to control and monitor their mock mars rover from afar.
- This system's goal is to allow for fast prototyping of the actual rover and relieve pressure from the software side of the rover team.

ENGINEERING REQUIREMENTS

- Reliability/Robustness: Needs to reliably function in harsh environments where network connectivity is not guaranteed.
- Control: Software must allow for remote driver and control of the Rover.
- Video Feedback: Software must show video feeds from the Rover, with bandwidth adjustment and camera selection capabilities.
- System Status: Software must display Rover system status information.
- Navigation/Mapping: Software must provide GPS navigation, mapping, localization, and waypoint setting.
- Autonomy Control: Software must allow for users to enable and disable autonomy on the rover.
- Documentation: Team must provide thorough documentation to serve ground station developers in coming years.

Oregon State University

OSU ROBOTICS CLUB
MARS ROVER GROUND STATION

Remote control and monitoring software for a competition-bound mock mars rover robot.

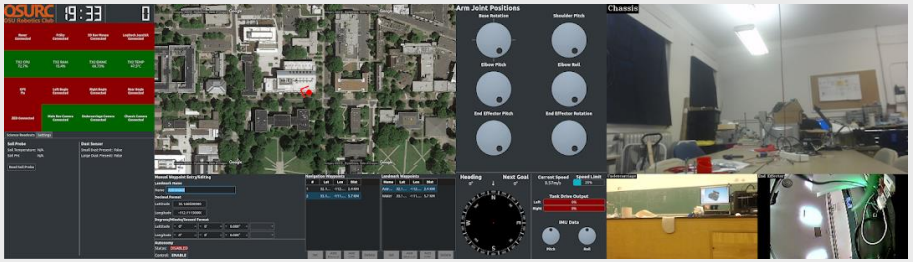


Figure 1: Screenshot of Ground Station Software

OVERVIEW

The OSU Robotics Club's Mars Rover team is developing a robot to compete in the Canadian International Rover Challenge, a robotics competition taking place in in August, 2018.

Competition tasks include astronaut assistance, equipment servicing, science/soil collection, extreme terrain traversal, and includes extra points for autonomous navigation. Teams must be able to remotely operate their Rover from a central ground station located up to a 1km away as well as have the ability to enable the autonomous navigation and self-driving mode.

The club's team of undergraduates builds the mechanical, electrical, and software components each year and has often requested capstone teams to help.

PROBLEM

While the OSURC Mars Rover team is developing the actual rover, it is the responsibility of our team to develop the ground station that will serve as the rover's base of operations.

The ground station is not only software but also consists of a computer, monitors, long-range radio, and additional input hardware such as joysticks that will serve as the primary point of contact between the Rover and the operating user.

The ground station software will be completely rewritten by our team for this year's competition and will serve as the foundation for years to come.

Due to the nature of competition it is vital that ground station software precisely fits the needs of the rover pilot.

SOLUTION

The Mars Rover team is utilizing the Robot Operating System (ROS) to facilitate communications between hardware on the Rover and the ground station. This allows for control and monitoring of the Rover via our software.

Our team is using Python 2.7 with the Qt framework via PyQt5 to display our data in a dual monitor full screen command station display setup.

Our video data streams select the best resolutions and quality settings automatically and can rotate between the possible video streams via Joystick control. Mapping is done with Google Static Images which are stitched together into a single cohesive map.

Additional feedback information such as system statuses and Rover arm states are displayed and updated frequently.

The software reads in joystick data and broadcasts it via ROS topics to allow the software to remotely drive the Rover and control its robotic arm.

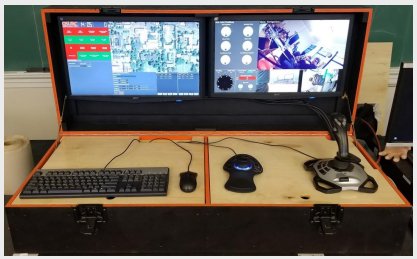


Figure 2: Fully Assembled Ground Station Quick Deploy Unit

PROJECT TEAM

Corwin Perren | Chris Pham | Ken Steinfeldt

perrenc@oregonstate.edu
phamchr@oregonstate.edu
steinfek@oregonstate.edu

PROJECT SPONSOR

OSURC
OSU Robotics Club

Contact: OSURCofficers@engr.oregonstate.edu

The OSU Robotics Club would like to thank their sponsors.

OSURC INTERNAL SPONSORS

College of Engineering
College of MIME
College of Business

OSURC EXTERNAL SPONSORS

Protocase
Boeing
Tattu
GitHub
Asana
Dominos

7 DOCUMENTATION

7.1 How the Project Works

7.1.1 Overview

This project works by combining the power of Robot Operating System (ROS), the Qt framework, and Python to make a simple (for what it is), fast, easy to modify, and easier to understand piece of software to act as the front end for the Mars Rover. PyQt is used to load and show the UI and give the software programmatic access to GUI elements. It also provides intelligent multi-threading with easy to use cross-thread communication pipelines. The use of python has allowed our team to write this code in a fraction of the time it would have taken in another language such as C++. ROS is the framework that the Rover is running and is based around the concept of abstracting everything into ROS topics. These topics allow for easy sending and receiving of control and status information in a way that is standardized instead of having to use a custom network control protocol.

7.1.2 Low bandwidth communications

As the Rover is going to be at long range at many times, having as small of communication packets as possible is a priority. This helps keep control, video, and status information smooth and consistent. In order to facilitate this, the ground station and Rover both are using a package called nimbro_network that allows for compression of ROS topics and also allows for us to choose between a TCP and UDP transport layer. Non-critical data that we want fast gets sent with compression over UDP and includes video data from the Rover and live drive commands from the ground station. Important and/or infrequent data is sent with or without compression depending and using the TCP transport. Data sent using TCP includes status information from the Rover and, in the future, waypoints and autonomy control.

7.1.3 Program Logical Structuring

To make the ground station software easier to modify and understand, it has been broken down into logical hierarchies. At the core of the ground station source folder are the Framework and Resources folders. The Resources folder contains all of the static assets for the application such as image files and the UI files that determine how the GUI visually looks. The Framework folder contains all of the logical sub-systems of the software such as VideoSystems, InputSystems, and MappingSystems. By separating all systems out like this, it makes it easy to know which files need to be looked at when fixing or adding to the ground station. At the root of the source folder is ”ground_station.py”, the launcher file for the whole application. Under normal conditions, this file only needs to be modified if a new threaded class is being added for additional functionality. This file handles launching of the software, displaying the GUI, and management of the startup and shutdown of all running threads.

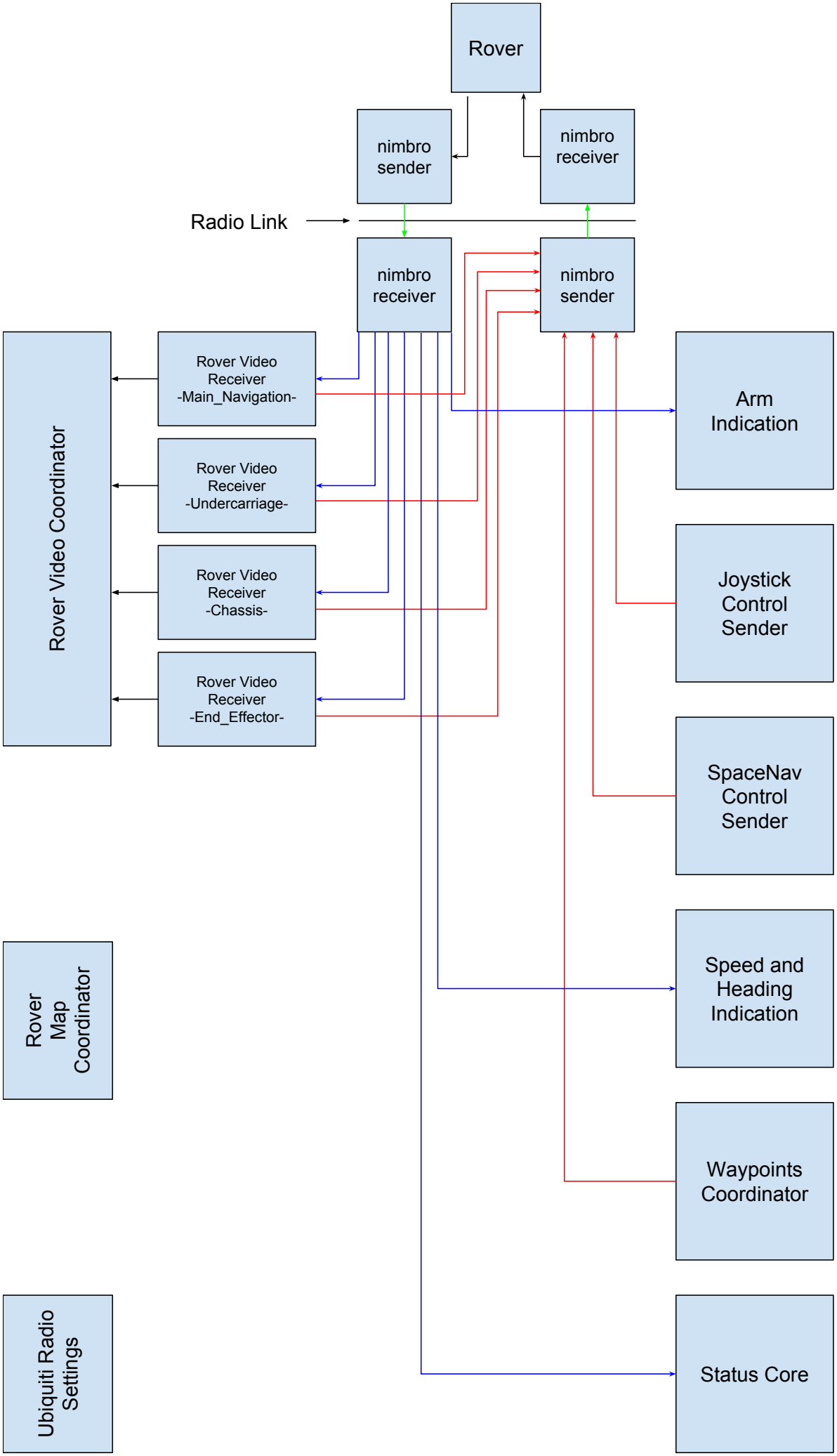
7.1.4 Threading & Adding Classes

As this is a very large program with many systems running concurrently it has required the use of many threaded classes. To abstract away many of the complexities of this, the ground station main launcher file contains a method called add_thread used to spin up a new threaded class from the Framework folder when one is made. This method also handles the graceful shutdown of the software when quit. In order for graceful shutdown, all of the program’s child threads must not be running when the main launcher thread exits. If this is not the case, the application may hang or maintain unwanted connections in the background after it seems like it has exited. Anyone wishing to add new functionality to the GUI in a new file should copy an existing threaded class to use as a baseline.

7.1.5 ROS in Python

ROS is a large part of why this software has been much easier to write than it could have been. In Python, the ROS subsystem can be accessed by importing rospy and calling rospy.init to tell ROS that we have a new node running. At this point, topics can be subscribed to by giving it the topic path and a the data type of the topic. Conversely, to broadcast a new topic message, a publisher is made by providing the topic path, data type, and queue size. To actually broadcast a message, you create a new instance of the data type, fill it with the desired data, and use the publisher's ".publish()" method to send it. In our software, these topics are used for all communications to and from the Rover. Absolutely no custom communication methods are used to interface between the Rover and ground station. An important note about the ROS messages used are that most of these messages are custom defined. In order for rospy and python to know the required information about a custom message, the ROS package containing the message define must be built as part of the local catkin workspace. This is the reason that the ground station's setup script includes many packages other than for just launching the ground station. The extra packages give us access to custom messages needed for understanding status messages and sending drive commands.

7.1.6 ROS Topic / Classes Block Diagram


Rover
nimbro sender
nimbro receiver
Radio Link
nimbro receiver
nimbro sender
Rover Video Coordinator
Rover Video Receiver -Main_Navigation-
Rover Video Receiver -Undercarriage-
Rover Video Receiver -Chassis-
Rover Video Receiver -End_Effector-
Arm Indication
Joystick Control Sender
SpaceNav Control Sender
Rover Map Coordinator
Speed and Heading Indication
Waypoints Coordinator
Ubiquiti Radio Settings
Status Core

7.2 System Requirements

7.2.1 Hardware

- 1x Computer running Ubuntu 16.04 LTS
  - Intel core i5 or i7 equivalent processor
  - 4GB+ Ram
  - Minimum two display outputs
- 2x 1080p Monitors
- 1x USB Joystick
- 1x SpaceNav Mouse
- 1x USB Keyboard
- 1x USB Mouse

7.2.2 Software / OS

- ROS Kinetic
- Python 2.X with the following packages:
  - PyQt5
  - qdarkstyle
  - inputs
  - spnav
  - Pillow
  - paramiko
  - OpenCV 2
  - qimage2ndarray
  - numpy
- Set the computer to an IP address of 192.168.1.15

7.3 How to Install

1) Create and setup a catkin workspace at "~/catkin_workspace"
2) Add "source /home/[username]/catkin_workspace/devel/setup.bash" to the end of your ".bashrc" file, replacing "[username]" with your account's username
3) Create the directory "~/Github" and clone the Rover_2017_2018 repository into it
4) Run the setup script at "~/Github/Rover_2017_2018/software/ground_station_setup.sh" to have the proper packages symbolically linked into your new catkin environment and for catkin_make to be run automatically against it
5) If the catkin_make process at the end of the script completes with no errors, you're ready to launch the ground station

7.4 How to Run

1) Ensure all hardware from the requirements section above is plugged in
2) Ensure there is a network connection between the ground station computer and Rover
3) Open a terminal and run ”roslaunch rover_main ground_station.launch”
4) The ground station should launch in full screen across both monitors
5) Press ”ctrl-q” at any time to quit the application

7.5 User Guides

7.5.1 Client Requested Quickstart Guide

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.

8 Technical Resources

- ROS Documentation
- QT Documentation

9 Conclusions and Reflections

9.1 Chris Pham

9.1.1 Technical Knowledge Gained

From this project, I learned quite a bit of technical knowledge doing this project over the year. I learned how to use Qt and ROS enough to build a system using it, do not think I can do it well though. After working with a large code base, I've learned to link projects in a tried grouped faction, close to the next object. I think I know how to remove very stateful code, and breaking it into helper functions and the main class. With the use of image manipulation, I feel that I know how to exactly how to use PIL, OpenCV, and other image software to stitch a bunch of files.

9.1.2 Non-Technical Knowledge Gained

For this project, I think I gained more of a time regulating skill, where I would have to say, I could not contribute to a project for a period of time, instead of doing half time on both projects or more. Another point, is learning to ask for expectations and goals, and then slightly passing them because extra work would be unnoticed, and/or unused by the team, but too little would produce an incomplete product.

9.1.3 Knowledge Gained of Project Work

From this project, I learned that when you expect that something will take a period of time, expect truly to take double that. Another take from this project is that, with some other projects that rely in other parts by other people, try to make something, and you could always try to fix it later. Goals will always change, and be ready to change on a dime. And in the end, without any type of leader or management, time will always be hard to come by, and everything will possibly be late.

9.1.4 Knowledge Gained of Project Management

For our group, there was not much of a need to police what someone did or did not do. In the end, our biggest problem was the time commitments from having a group member in Hillsboro and only down Tuesday or Thursday which made some activities like weekend meetings near impossible.

9.1.5 Knowledge Gained of Working in Teams

For the group, we all had fun with each other and everyone else in the team and that made communications very easy to do. Everyone was free form, and we did whatever we needed and the others would be accommodating in trying to fit that schedule. When we could not meetup, we could at least use other means of communications like Slack to talk about expectations and code, which allowed for us do to whatever we need to do.

9.1.6 Changes to Make if Starting Over

If we were to start this over, I would like to some things to allow us to succeed further. I think we would all make better plans at the start to allow for missteps of the team, and allow for easier prototyping. That prototyping can allow for delays in the team, and then can be built with expected values instead of waiting for the system to be built. Another thing would be getting testing data that we can use for integrated tests like GPS, Video, and Statuses, so we can just build a system using that data then it should be the same on the complete system. I think for my section, I would actually use OpenGL instead of the PIL mapping system, because translating numbers randomly feels very weird, compared to my graphics history.

9.2 Ken Steinfeldt

9.2.1 Technical Knowledge Gained

I gained a lot of technical knowledge and experience from the project. First and foremost I learned how to use frameworks that I had never experienced before in ROS and PyQT5. Not only had I never used Qt before, but I had never worked with any real UI framework before this project. The ROS framework was also a unique challenge. This project was an example of a very real use case for ROS, doing exactly what it was designed to do. The framework was more complicated than I had anticipated, and required a significant learning curve. Using both ROS and PyQt5 on a project such as this required using the frameworks according to best practice, and was fairly complicated at first, but simpler as time went on.

9.2.2 Non-Technical Knowledge Gained

I think that the most important non-technical knowledge I gained was the experience of working on a large, complicated project. As the ground station was only a small portion of the of the Mars Rover project, it relied on many modules in many different locations. As a group we were forced to manage these problems as they arose. Even the organization of the repository was a challenge and had to be redone multiple times. Development was also a problem as it could be a challenge to set up a functional development and testing environment. This was alleviated with the development of a startup script that Corwin wrote.

9.2.3 Knowledge Gained of Project Work

Software engineering is notorious for delays and missed deadlines, and I am beginning to understand why. At the beginning of the project many modules seemed to have a clear path to completion and simple solutions. We thought that it would be easy for us to learn the two frameworks and then just start working with them, after all, that's what frameworks are for. As it turns out, both ROS and Qt have not insignificant learning curves. Learning how to use these frameworks took me much longer than I thought it would, and that hampered me throughout the process. I also learned to trust my teammates, which is not something that someone expects to learn from a group project. My teammates were great at always came through, even when I was not always able to. By the end of the project I had learned that I could trust them completely.

9.2.4 Knowledge Gained of Project Management

The biggest thing that I learned about project management was how much easier it is to do when the project is well defined and planned at the beginning. The work that we were required to do at the beginning of the year defined

the scope of the project, the goals, the problems, and their planned solutions. Having this already done due to the requirements of the class was hugely beneficial in implementing our solutions and a lesson in project management for the future.

9.2.5 Knowledge Gained of Working in Teams

Our group had great communication which was very beneficial to the team. This open communication helped to make clear what was expected of each group member and helped our group to have fun and get along.

9.2.6 Changes to Make if Starting Over

- Take more time to understand the frameworks before trying to code with them.
- Spend more time with the group personally.
- Spend more time working with the rover team on rover stuff.
- Take the time to create the testing environment before starting.

9.3 Corwin Perren

9.3.1 Technical Knowledge Gained

- Real-world ROS usage
- Multi-monitor PyQt interfaces
- Some nuances of low-bandwidth networking
- Multiple video steam coordination
- Better organization of large software projects

9.3.2 Non-Technical Knowledge Gained

- Better project time management skills

9.3.3 Knowledge Gained of Project Work

- Projects almost always take longer than expected
- Requirements almost always change, even if only a little bit
- Lack of proper organization can make working on a project near impossible

9.3.4 Knowledge Gained of Project Management

- Scheduling times to do things like project work where everyone can show up can be difficult

9.3.5 Knowledge Gained of Working in Teams

- Continuous communication makes working in teams much much easier
- So long as everyone is on the same page teams can get as much done when in the same room as they can separately

9.3.6 Changes to Make if Starting Over

- Make better plans at the beginning for handling delays in the Rover side of the project
- Work out a more empirical way of testing how much bandwidth is needed / can be handled by the radio links so we would have known what we had to work with

10 Appendix 1: Interesting Code Listings

10.1 Drive Test

10.1.1 Code

```
class DriveTest(QtCore.QThread):
    def __init__(self):
        super(DriveTest, self).__init__()

        self.not_abort = True

        self.message = None
        self.publisher = rospy.Publisher("/cmd_vel", Twist, queue_size=10)

        rospy.init_node("test")

    def run(self):
        while self.not_abort:
            self.message = Twist()

            self.message.linear.x = 1.0
            self.message.angular.z = 1.0

            self.publisher.publish(self.message)

            self.msleep(100)
```

10.1.2 Description

This QThread example class starts a ROS publishing node on the "/cmd_vel" topic to send raw drive control commands to the Rover. In this case, it is sending a command to drive the Rover forward and to the right, essentially causing it to drive in a never-ending circle.

10.2 Video Test

10.2.1 Code

```
class VideoTest(QtCore.QThread):
    image_ready_signal = QtCore.pyqtSignal()

    def __init__(self, screen_label, video_size=None, sub_path=None):
        super(VideoTest, self).__init__()

        self.not_abort = True

        self.screen_label = screen_label
        self.video_size = video_size

        self.message = None
        self.publisher = rospy.Subscriber(sub_path, CompressedImage, self.__receive_message)

        self.raw_image = None
```

```
        self.cv_image = None
        self.pixmap = None
        self.bridge = CvBridge()

        self.image_ready_signal.connect(self.__on_image_update_ready)

    def run(self):
        while self.not_abort:
            if self.raw_image:
                self.cv_image = self.bridge.compressed_imgmsg_to_cv2(self.raw_image, "rgb8")

                if self.video_size:
                    self.cv_image = cv2.resize(self.cv_image, self.video_size)

                self.pixmap = QtGui.QPixmap.fromImage(qimage2ndarray.array2qimage(self.cv_image))
                self.image_ready_signal.emit()
            self.msleep(20)

    def __on_image_update_ready(self):
        self.screen_label.setPixmap(self.pixmap)

    def __receive_message(self, message):
        self.raw_image = message
```

10.2.2 Description

This example subscribes to the ROS topic that is passed in under the sub_path argument in order to get video stream data. An example of this topic might be "/cam1/usb_cam1/image_raw/compressed". Inside of the body of the thread, it checks if there is image data, and if so decompresses it into a raw 8-bit image using ROS's OpenCV bridge libraries. Finally, it converts the OpenCV image into a QImage and then into a QPixmap before broadcasting an update signal so the main GUI thread can show the image on the QLabel. It is important to note that any direct GUI updates must happen in the main GUI thread, otherwise the QApplication will crash.

10.3 Joystick ROS Drive Test

10.3.1 Code

```
rospy.init_node("drive_tester")
self.pub = rospy.Publisher("/drive/motoroneandtwo", RoverMotorDrive, queue_size=1)

def __get_controller_data(self):
        if (self.controller_aquired):
            events = self.gamepad.read()

            for event in events:
                if event.code in self.raw_mapping_to_class_mapping:
                    self.controller_states[self.raw_mapping_to_class_mapping[event.code]] = event.state
                    # print "Logitech: %s" % self.controller_states


def __broadcast_if_ready(self):
```

```
drive = RoverMotorDrive()

axis = self.controller_states["left_stick_y_axis"]

drive.first_motor_direction = 1 if axis <= 512 else 0
drive.first_motor_speed = min(abs(self.controller_states["left_stick_y_axis"] − 512) ∗ 128, 65535)

self.pub.publish(drive)
```

10.3.2 Description

These two methods and supporting lines above, taken from the testing class LogitechJoystick, contained in the file joystick_drive_test.py are the core of what is needed to get joystick data and broadcast it to the rover over a ROS topic. These two methods are called on after another in a QThread. __get_controller_data() reacts to motion events from the joystick and stores the current value of all axes and buttons in self.controller_states. Then, in __broadcast_if_ready(), and instantiation of the custom ROS message type, RoverMotorDrive, is made and values set to a scaled version of the raw values provided by the joystick. Finally, this data is published to the motor drive node and causes the ROS receiving node to see the data, send a message to the motor driver, and cause the motor to spin.

10.4 Video Test

10.4.1 Code

```
def toggle_video_display(self):
        if self.video_enabled:
                if self.video_subscriber:
                        self.video_subscriber.unregister()
                self.new_frame = True
                self.video_enabled = False
        else:
                new_topic = self.camera_topics[self.current_camera_settings["resolution"]]
                self.video_subscriber = rospy.Subscriber(new_topic, CompressedImage, self.__image_data_received_callback)
                self.video_enabled = True
```

10.4.2 Description

This very simple snippet is in the VideoReceiver class in VideoSystems. It is a demonstration of what is needed to properly disable the receiving of video data on a stream. Looking at the Subscriber line, you can see that there is an image callback associated with the subscription to a topic in ROS. This means that if you don't actually unsubscribe (or in this case, unregister) from a topic, as can be seen a few lines above, the data will continue being received even if you are not actively using it. Not doing this would cause unwanted bandwidth to be used.

10.5 Ubiquiti Channel Change

10.5.1 Code

```
GET_CURRENT_CHANNEL_COMMAND = "iwlist ath0 channel"
SET_CHANNEL_COMMAND = "iwconfig ath0 channel"

def setup_and_connect_ssh_client(self):
```

```
    self.ssh_client = paramiko.SSHClient()
    self.ssh_client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
    self.ssh_client.connect(ACCESS_POINT_IP, username=ACCESS_POINT_USER, password=ACCESS_POINT_PASSWORD,
                            compress=True)

def apply_channel_if_needed(self):
    if self.channel_change_needed:
        self.show_channel__signal.emit(0)
        self.set_gui_elements_enabled__signal.emit(False)
        self.ssh_client.exec_command(SET_CHANNEL_COMMAND + " %02d" % self.new_channel)
        self.get_and_show_current_channel()
        self.channel_change_needed = False

def get_and_show_current_channel(self):
    channel = 0

    ssh_stdin, ssh_stdout, ssh_stderr = self.ssh_client.exec_command(GET_CURRENT_CHANNEL_COMMAND)
    output = ssh_stdout.read()

    for line in output.split("\n"):
        if "Current Frequency:" in line:
            channel = line.strip("()").split("Channel ")[1]
            break

    self.msleep(500) # From the gui, this helps show something is actually happening

    self.show_channel__signal.emit(int(channel))
    self.set_gui_elements_enabled__signal.emit(True)
```

10.5.2 Description

This code shows how we change the 2.4GHz radio channel on the Rocket M2 radios. In the first method, the class initializes an ssh connection to the access point radio. Then, in the thread's main loop, apply_channel_if_needed runs on a regular basis waiting for a channel change to happen. Once it does, it executes an ssh command to change the channel via command line before running a second command via the third method to get the new channel from the radio, parse it, and show it in the GUI. This ensures that if the channel does not get set, the value shown in the GUI will alert the user to this fact.

10.6 Compass Rotation

10.6.1 Code

```
ROTATION_SPEED_MODIFIER = 2.5

def rotate_compass_if_needed(self):
    heading_difference = abs(int(self.shown_heading) − self.current_heading)
    should_update = False

    if heading_difference > ROTATION_SPEED_MODIFIER:
        self.shown_heading += self.rotation_direction ∗ ROTATION_SPEED_MODIFIER
        self.shown_heading %= 360
```

```
            should_update = True
        elif heading_difference != 0:
            self.shown_heading = self.current_heading
            should_update = True

        if should_update:
            self.current_heading_shown_rotation_angle = int(self.shown_heading)

            if self.current_heading_shown_rotation_angle != self.last_current_heading_shown:
                new_compass_image = self.main_compass_image.rotate(self.current_heading_shown_rotation_angle, resample=PIL.Image.
                    ↪ BICUBIC)
                self.last_current_heading_shown = self.current_heading_shown_rotation_angle

                self.compass_pixmap = QtGui.QPixmap.fromImage(ImageQt(new_compass_image))
                self.show_compass_image__signal.emit()

def update_heading_movement(self):
    current_minus_shown = (self.current_heading − self.shown_heading) % 360

    if current_minus_shown >= 180:
        self.rotation_direction = −1
    else:
        self.rotation_direction = 1
```

10.6.2 Description

This code shows how movement updates to the compass are made. The second method gets called when a new heading change is made, setting whichever rotation direction is shorter to reach the desired goal. Then, in the main loop, the upper method calculates the difference between the current shown heading and the actual heading determining whether it should be moving or not. If it should, the main compass image that was loaded during init is rotated via a bicubic sampling algorithm to maintain image clarity, converted to a QPixmap, before an update signal is broadcast to the main thread to show the image.

11 APPENDIX 2

11.1 Ground Station Final Photos

Figure 1: Finished Ground Station Hardware

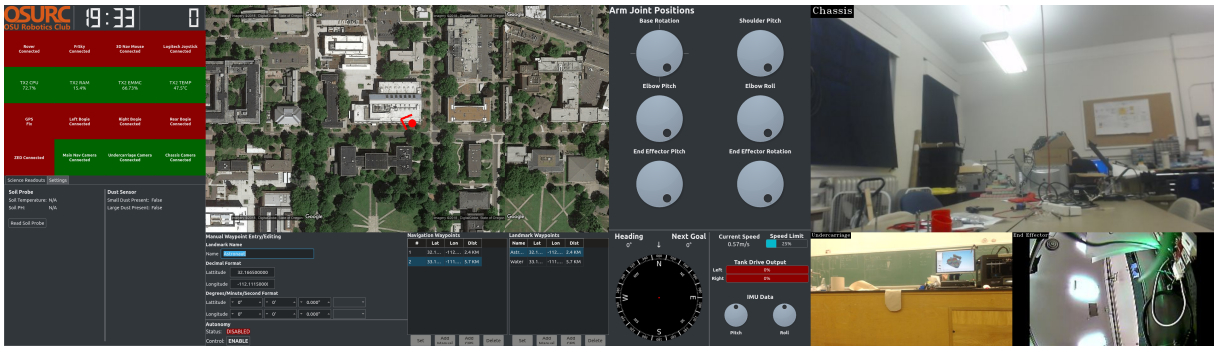


Figure 2: Finished Ground Station Screenshot

Figure 3: Finished Ground Station at Expo

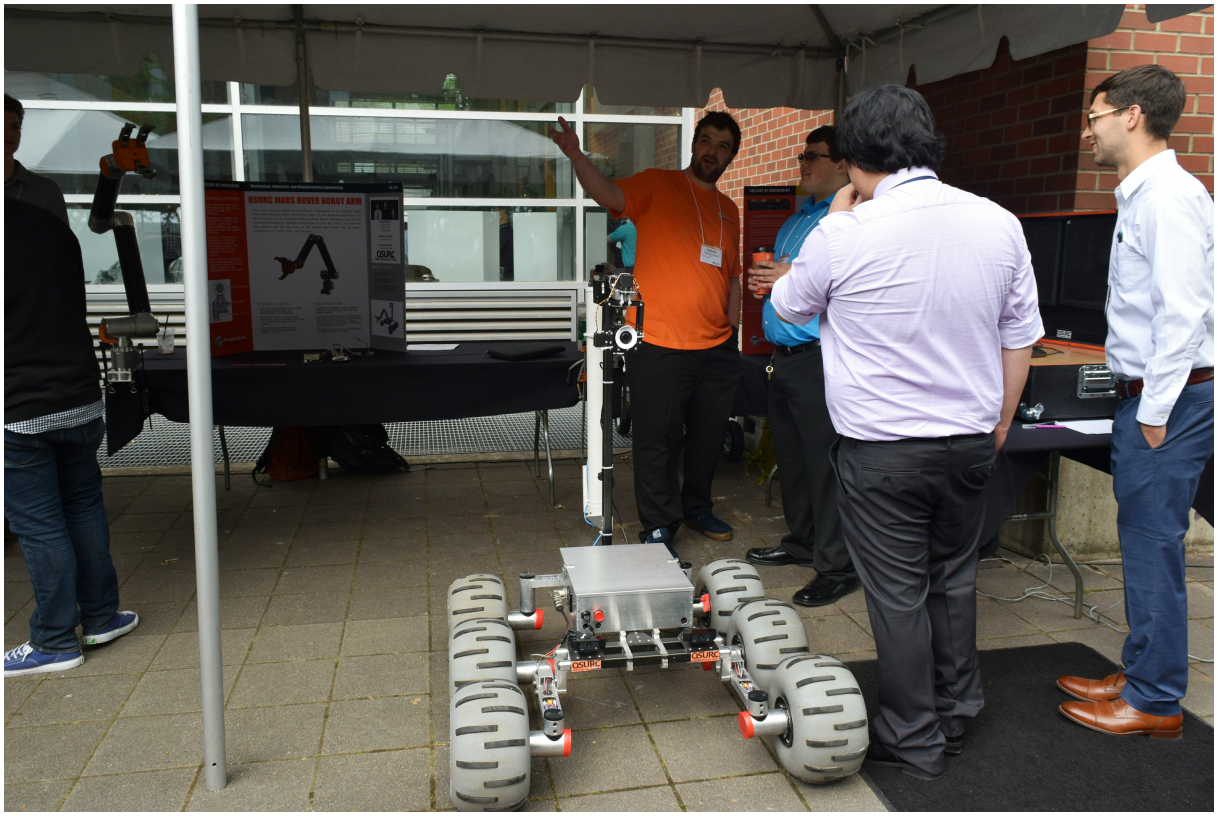


Figure 4: Chatting with TA at Expo