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

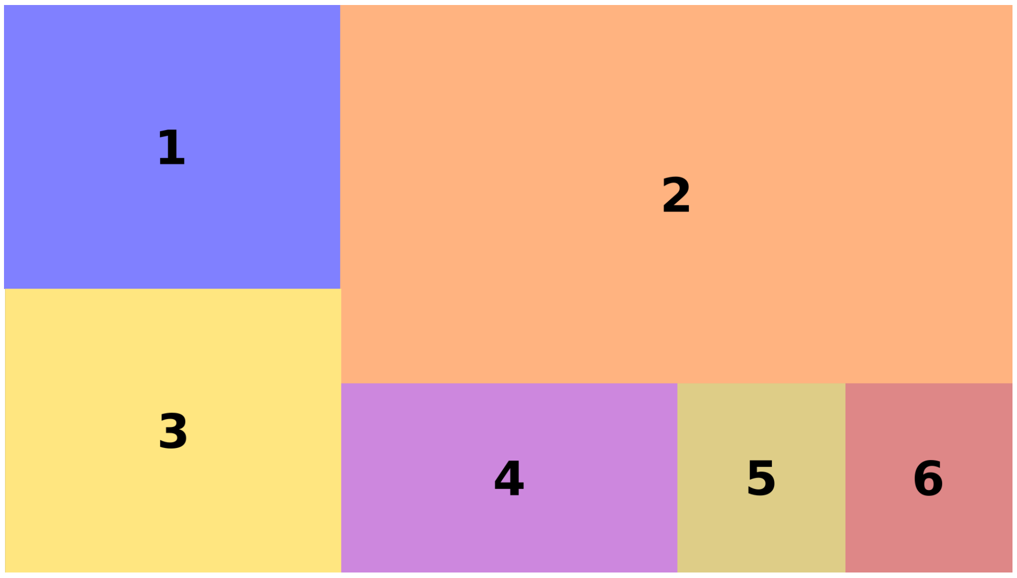

Fig. 1: Left Screen Mock-up

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

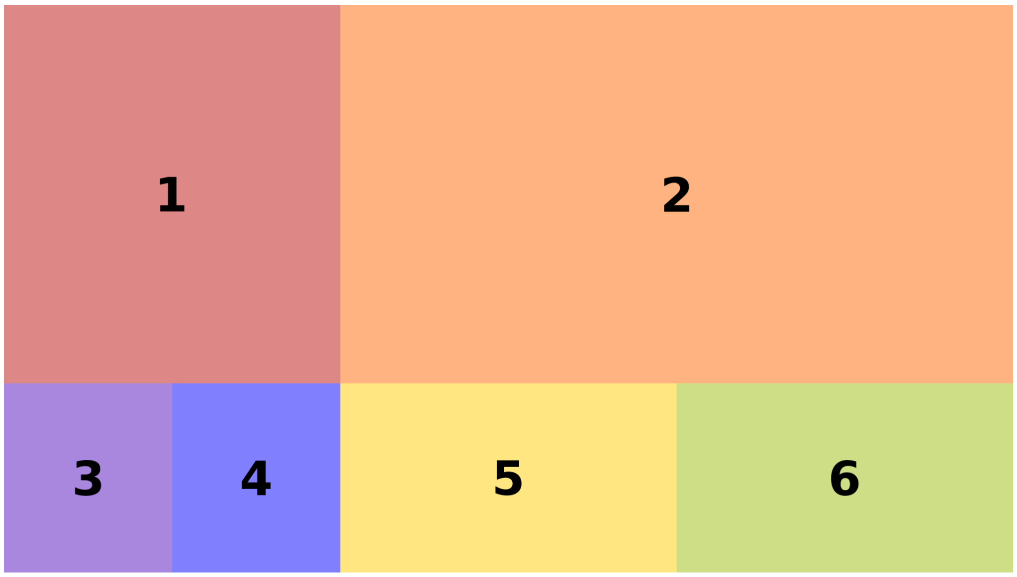

Fig. 2: Right Screen Mock-up

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.