**Detailed Design Documentation**

**Robo-Ops Competition**

**Controls and Communication**

**Prepared By: Daniel Bittner, Edward Carter, Jun He, Matthew Ng, Dakota Pollitt, Zhentao Zhong**

**Software Engineering Fall 2014**

**Rowan University**

**Instructor: Dr. Adrian Rusu**

**Contents**

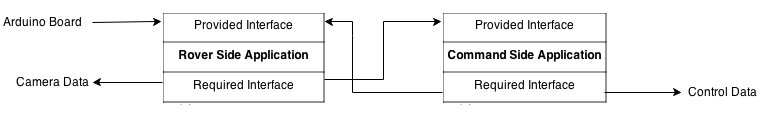
1. Introduction ………………………………………………………………………………………………………………….3
2. Module Design ……………………………………………………………………………………………………………...3  
   2.1 Command Side Application ……………………………………………………………………………………..3  
    2.1.1 Command Communication ……………………………………………………………………….5  
    2.1.2 Control ……………………………………………………………………………………………………..7  
    2.1.3 Camera Processing ………………………………………………………………………………….12  
    2.1.4 GUI ………………………………………………………………………………………………………….13  
   2.2 Rover Side Application …………………………………………………………………………………………..14  
    2.2.1 Rover Communication ……………………………………………………………….……………15  
    2.2.2 Rover Command Processing ………………………………………………………………..….19  
    2.2.3 Rover Camera Processing ………………………………………………………………………..20
3. Abstract Data Types …………………………………………………………………………………………………….21
4. COMPRISES Diagram …………………………………………………………………………………………………...21
5. USES Diagram …………………………………………………………………………………………..………………...22

Integration Test Plan …………………………………………………………………………………………………...22

1. **Introduction**

This is the official design document for the NASA’s RASC-AL Robo-Ops planetary exploration rover competition rover. This document is based on the requirements explained by the SRS document. The system will allow the following requirements:

* Allow the user to wirelessly control the Rover while at the station located in NJ.
* Allow the user to see live camera feeds coming from the rover and switch between cameras at will.
* Allow the user to control the arm of the rover to pick up rocks as well as other basic controls such as move forward, backward and turn left and right.



**Figure 1.1 High-level architecture**

This design will allow the user to quickly send data wirelessly. A Graphical User Interface will also be implemented which will allow the user to easily control the cameras on the rover. A joystick will allow the user to easily control the rovers movements. These features will help the user control the rover and its respective cameras as quickly and fluidly as possible.

The rest of the document is organized in the following way: Chapter two contains detailed descriptions of each module found in the design, including the purpose, rationale and interfaces of each module. Chapter three contains a list of Abstract Data Types(ADT’s) used in the application. Chapter four will illustrate a COMPRISES diagram. Chapter five will illustrate a USES diagram, this will lay out the diagram in more detail. Lastly chapter six will describe a detailed test plan for the application.

1. **Module Design**

**2.1 Command Side Application Module**

**Purpose**

The purpose of this module is to provide the user with wireless control of the rover from afar. The module will allow the user to turn the rover, move the rover forward, backward, and control the arm of the rover. It will also allow the user to switch between camera feeds streamed live from the rover. This module also contains the GUI for the whole system, allowing the user to view information from the rover’s cameras.

**Rationale**

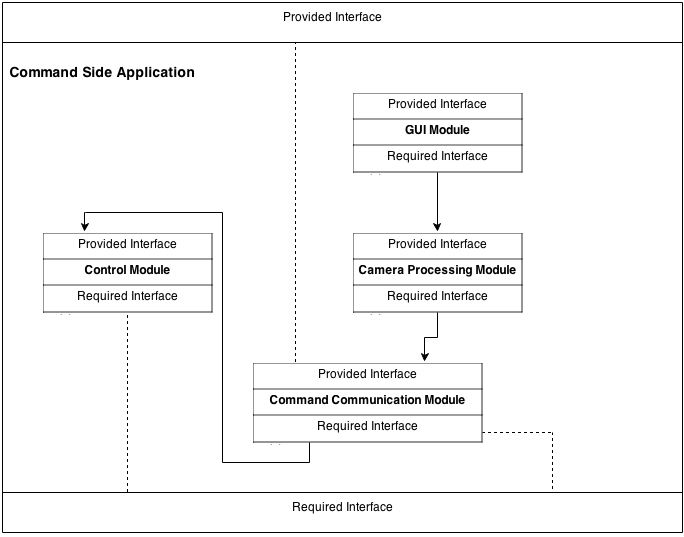
The rationale of this module is to give the user complete wireless control of a rover from a distance in order to compete in the NASA’s RASC-AL Robo-Ops planetary exploration rover competition.

**Provided Interface**

The Command Side Application Module’s provided interface is comprised of the interface provided by the Communication Module.

**Required Interface**

The Command Side Application Module’s required interface is comprised of the interface required by the Communication Module and the input data provided by the joystick.



**Figure 2.1 High-level Design for Command Side Application Module**

**2.1.1 Command Communications Module**

**Purpose**

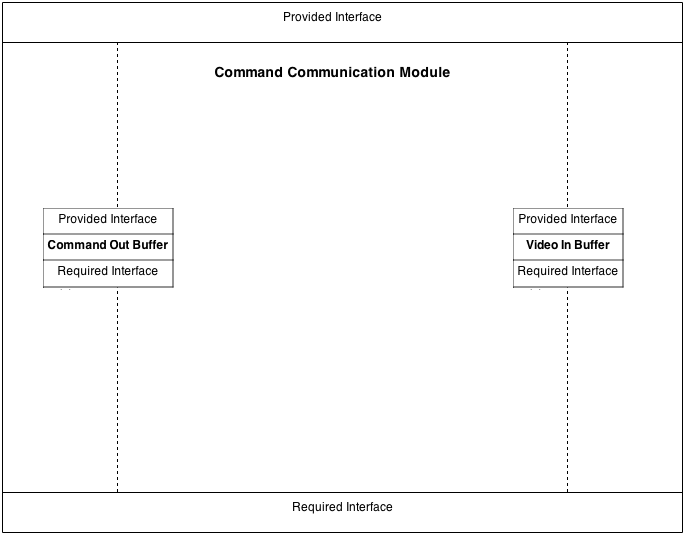
The purpose of this module is to manage sending commands and receiving video feeds over network connections between the rover’s onboard PC and the command PC.

**Rationale**

This module was created in order to enable the Mission Control and the Rover to be able to communicate over a long distance. Also it eases out the onboard PC’s computation by handling the video streaming.

**High Level Module Design**

The module is broken down as shown below.



**Figure 2.3 High Level Architecture of Communication Module**

**2.1.1.1 Command Out Buffer Module**

**Purpose**

The purpose of the buffer module is to hold command information as it waits to be sent across the network to the Rover Communication Module, where it will be held until the Rover Command Processing Module is ready to process it.

**Rationale**

The rover will not be able to immediately process any command as it is sent

across the network so a buffer is needed to hold a queue of commands to be processed in order as the rover command processing module handles them.

**Provided Interface**

**boolean addCommands(BufferUnit buffer, byte command)**

Definition: adds command into a buffer unit

Parameters: BufferUnit, byte command

Returns: boolean value of the action

Throws: none

**byte getCommand(BufferUnit buffer)**

Definition: gets command from the buffer unit

Parameters: Buffer Unit

Returns: returns the byte command

Throws: none

**boolean removeCommand(BufferUnit buffer)**

Definition: removes command from the buffer unit

Parameters: Buffer Unit

Returns: returns boolean value of the action

Throws: none

**Required Interface**

**Control Module**

Command getCommand()

**2.1.1.2 Video In Buffer Module**

**Purpose**

The purpose of this module is to receive encoded transferrable video packets across the internet from the rover.

**Rationale**

This is needed so that the GUI can receive the real time video feed from the

rover cameras and provide drivers the ability to see what the rover is doing.

**Provided Interface**

**boolean addVideo(BufferUnit buffer, transferableVideo video)**

Definition: adds pakced videos into the Buffer unit

Parameters: Buffer unit, transferableVideo video

Returns: boolean value of the action

Throws: none

**transferableVideo getvideo(BufferUnit buffer)**

Definition: gets video from the buffer unit

Parameters: Buffer Unit

Returns: returns the transferable Video

Throws: none

**boolean removeVideo(BufferUnit buffer)**

Definition: removes video from the buffer unit

Parameters: Buffer Unit

Returns: returns boolean value of the action

Throws: none

**Required Interface**

**Rover Communication Module**

Packet getVideoPacket()

**2.1.2 Control Module**

**Purpose**

The purpose of this module is provide the connection controllers and the local PC. So that when the users create actions on controllers, this module can figure out which controller is sending actions, and send the signal to the local PC, then to the Robot PC, Finally to the Robot’s motors and arm. This module can handle signals and send to connection module.

**Rationale**

Without this module, the PC wouldn’t able to send orders to the rover. Rover will not able to function for action.

**Provided Interface**

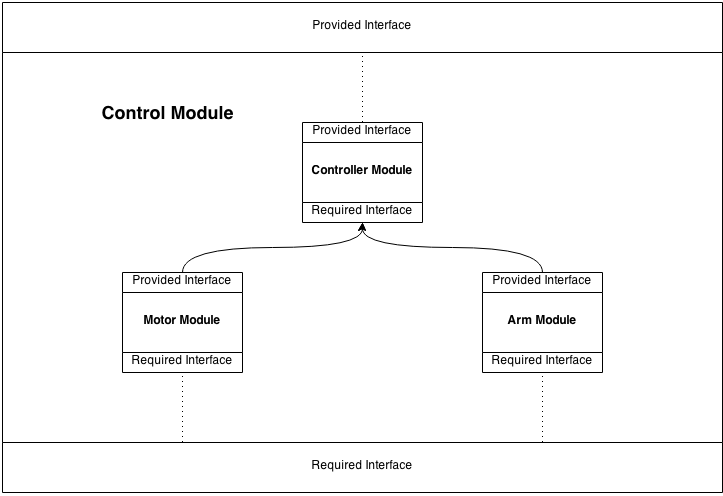
The Control Module provides command packets for the communication module to send out. These packets include data that tells the rover’s motors what to do.

**Required Interface**

The Control Module requires the joystick to function. The joystick inputs the commands to the Control Module.

**High Level Module Design**

The module is broken down as shown below.



**Figure 2.2 High Level Architecture of Control Module**

**2.1.2.1 Motor Module**

**Purpose**

The purpose of this module is to receive inputs from the motor joystick, specify actions for the robot and send actions to the robot motors. This allows the motors to move forward, backward and turn.

**Rationale**

Without this module users cannot send motor actions to the robot. Without motor actions, the robot will not able to move.

**Provided Interface**

**void getData()**

Definition: Override super class, get data input that joystick provide, change to useful

information for motor

Parameters: none

Returns: none

Throws: none

**void getSpeed()**

Definition: use data from joystick transfer to speed of robot

Parameters: none

Returns: none

Throws: none

**void getRotation()**

Definition: uses data from joystick transfer to rotation of robot wheels

Parameters: none

Returns: none

Throws: none

**Required Interface**

Handled by the super class.

**2.1.2.2 Arm Module**

**Purpose**

The purpose of this module is to receive inputs from the arm joystick, specify actions for the robot and send actions to the robot arm. The arm can move based on its degrees of freedom.

**Rationale**

Without this module users cannot create arm actions to the robot. Without arm actions, the robot will not able to move the arm and will not be able to pick up objects.

**Provided Interface**

**void getData()**

Definition: Override super class, get data input that joystick provide, change to useful

information for arm

Parameters: none

Returns: none

Throws: none

**void jointSwitch()**

Definition: switch between shoulder, elbow, and wrist

Parameters: none

Returns: none

Throws: none

**void getMovement()**

Definition: provide order of movement for arm

Parameters: none

Returns: none

Throws: none

**void getRotation()**

Definition: provide order of rotaton for arm

Parameters: none

Returns: none

Throws: none

**Required Interface**

Handled by the super class.

**2.1.2.3 Controller Module**

**Purpose**

The purpose of this module is receive inputs from Arm and Motor modules and do the command calculations for these components before passing on the information to the Command Out Buffer.

**Rationale**

Without this module motor module and arm controller module have to code the same thing twice. It will take more time and the code will not look clean.

**Provided Interface**

**int getIndex()**

Definition: gets the port Index

Parameters: none

Returns: int of port index

Throws: none

**void getData()**

Definition: gets data input that joystick provides

Parameters: none

Returns: none

Throws: none

**void searchForControllers()**

Definition: searches for available joystick controllers

Parameters: none

Returns: none

Throws: none

**boolean isEmpty()**

Definition: checks if joystick controller is empty

Parameters: none

Returns: boolean whether joystick is empty

Throws: none

**Required Interface**

**Controllers[] getControllers()**

Definition: provides list of controllers in PC

Parameters: none

Returns: an array of controllers

Throws: none

**Type getType()**

Definition: provides type of the controllers

Parameters: none

Returns: a type of the controller

Throws: none

**getComponents()**

Definition: provides input that the controller sends

Parameters: none

Returns: an array of componenets contain datas input of the joystick

Throws: none

**2.1.3 Camera Processing Module**

**Purpose**

The purpose of the cameras is to allow the user to see the rover’s environment and choose the most optimal path for the rover. This module will decode video from the camera and pack the uncoded video data for transmission via the Communication Module.

**Rationale**

The rationale of this module is to prepare video packets for use by other modules in the software.

**Provided Interface**

**void unpack(packedVideo data)**

Definition: unpacks up video data to be sent out

Parameters: DecodedVideo

Returns: none

Throws: none

**DecodedVideo Decode(Video data)**

Definition: decodes video from camera

Parameters: Video

Returns: none

Throws: none

**Required Interface**

**Command Communication Module**

transferableVideo getVideo()

**2.1.4 GUI Module**

**Purpose**

Provide interface GUI to the users. It will receive the image and display in the interface. It will provide user the ability to switch the cameras sets of the rover and send signal to the rover to change the activate cameras. It also display information of the system, tells users any necessary information.

**Rationale**

Without a GUI the user will not be able to interact with the rover efficiently nor will the user be able to see the camera feed.

**Provided Interface**

**void display()**

Definition: displays the necessary interface

Parameters: none

Returns: none

Throws: none

**void actionPerformed(ActionEvent e)**

Definition: monitors and listens to actions performed on the GUI

Parameters: ActionEvent e

Returns: none

Throws: none

**void switchCameraFeeds(Camera camera)**

Definition: perform actions for switching between cameras

Parameters: Camera camera

Returns: none

Throws: none

**void displayCameraFeeds(transferableVideo video)**

Definition: display camera feeds on the GUI

Parameters: transferableVideo video

Returns: none

Throws: none

**Required Interface**

**Action Listener from java library**

void actionPerformed(ActionEvent e)

**Camera Processing Module**

DecodedVideo Decode(Video data)

**2.2 Rover Side Application**

**Purpose**

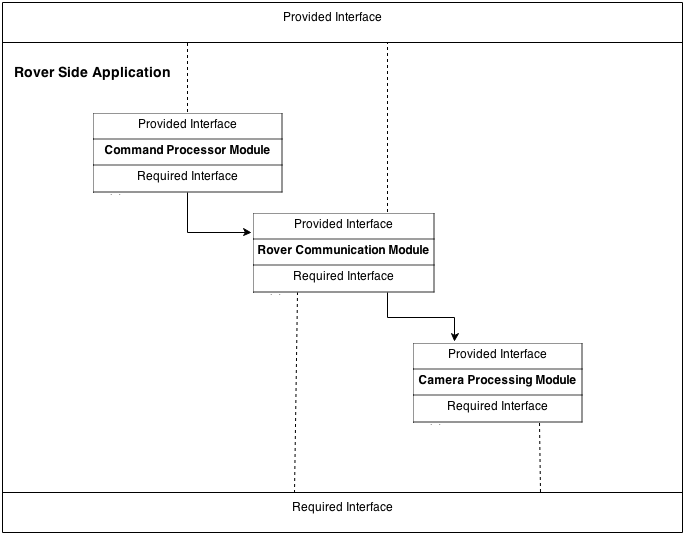
The purpose of the rover onboard module is to manage the flow of information between the command PC and rover motors, arm and camera. A buffer holds commands issued from the command PC. The commands are continually processed from the buffer and converted into a format that the Arduino systems can used to control the command motors. The video sent from the camera will be processed by the onboard rover PC so it can be sent back to the communications module in an efficient and useable format. The rover will also be able to receive commands to only return certain sets of the video feeds.

**Rationale**

This module is necessary to serve as an interlink between the command PC and the motors so that there is a way to control and command the rover remotely.

**High Level Module Design**

The module is broken down as shown below.



**Figure 2.4 High-level Architecture of Rover Side Application Module**

**2.2.1 Rover Communications Module**

**Purpose**

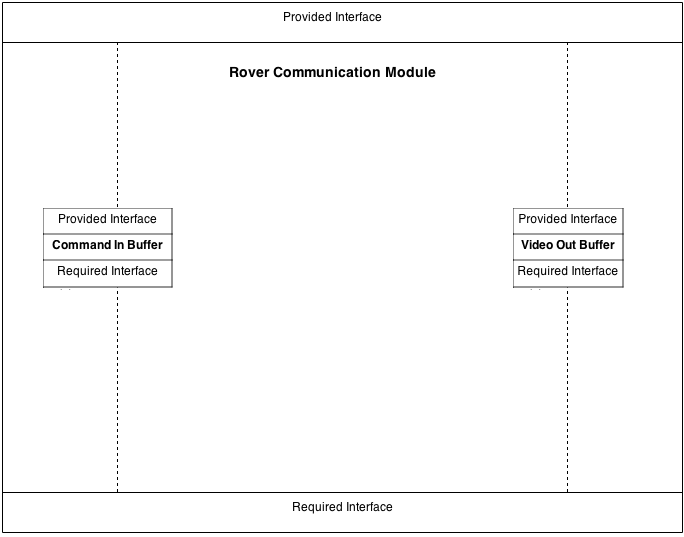
The purpose of this module is to manage sending commands and receiving video feed backs over network connections between the rover’s onboard PC and the command PC.

**Rationale**

This module was created in order to enable the Mission Control and the Rover to be able to communicate over a long distance. Also it eases out the onboard PC’s computation by handling the video streaming.

**High Level Module Design**

The module is broken down as shown below.



**Figure 2.3 High Level Architecture of Communication Module**

**2.2.1.1 Command In Buffer Module**

**Purpose**

The purpose of the buffer module is to hold command information as it is

received across the network from the Communications module. It will hold this information until the Rover Command Processing module is ready to process it.

**Rationale**

The rover will not be able to immediately process any command as it is sent

across the network so a buffer is needed to hold a queue of commands to be processed in order as the rover command processing module handles them.

**Provided Interface**

**boolean isReady()**

Definition: Checks to see if the buffer is ready to receive information.

Parameters: None

Returns: A boolean representing whether or not the information is ready.

Error: IOException

**int commandCount()**

Definition: Counts and returns the number of commands currently in the buffer.

Parameters: None

Returns: An integer representing the number of BufferUnits that the buffer is

currently holding

Error: None

**boolean hasCommands()**

Definition: Checks whether or not there are currently any commands in the buffer

Parameters: None

Returns: A boolean representing true if there is a command in the buffer and false

if there is not

Error: None

**byte processByte()**

Definition: Pulls a command packet byte from the Rover wireless port

buffer and returns it

Parameters: None

Returns: A byte of command information

Error: None

**BufferUnit parseCommand()**

Definition: Pulls bytes of information from the Rover wireless port looking for

midpoint stop bytes to indicate the end of the command itself and a stop byte indicating the end of destination motor bytes

Parameters: None

Returns: A BufferUnit representing an entire command including bytes which

represent how much the motor needs to turn and bytes representing the destination motors for the command

Error: None

**void addCommand(BufferUnit command)**

Definition: Adds the processed command into a queue of stored commands waiting

for the Rover Command Processing to be ready to access it

Parameters: A BufferUnit representing the complete information of the command to

be added to the queue

Returns: None

Error: None

**BufferUnit processCommand()**

Definition: Returns a command from the queue that is ready to be processed by the

command processing module

Parameters: None

Returns: A BufferUnit representing a complete command ready to be processed

Error: None

**Required Interface**

**Command Communications Module**

Packet getCommand()

**2.2.1.2 Video Out Buffer Module**

**Purpose**

The purpose of this module is to send processed video feed across the internet

back to the command PC.

**Rationale**

This is needed so that the GUI can receive the real time video feed from the

cameras.

**Provided Interface**

**boolean openConnection()**

Definition: Opens up an outbound connection with the command PC

Parameter: None

Returns: A boolean, representing whether or not a connection was

successfully opened with the command PC.

Error: IOException

**void sendVideoFeed(transferrableVideo videoFeed)**

Definition: Sends the transferrable from the network to the Communication

Module.

Parameter: transferrableVideo, a representation of video that can be sent across the

network to the Communication module

Returns: None

Throws: None

**boolean receivedCommand()**

Description: This function will returns the boolean value of whether it receives the command from the command PC.

Parameters: none

Returns: boolean

Throws: none

**Required Interface**

**Rover Camera Processing Module**

transferableVideo prepareVideo(encodedVideo videoFeed)

**2.2.2 Rover Command Processing Module**

**Purpose**

The purpose of the rover command process module is to take commands that

have been received in the rover buffer, process them into a Arduino useable format, and send them to the Arduino motor controls.

**Rationale**

A module is needed to coordinate handling of all the different commands that are

received in the buffer. The commands need to be sent to the proper place in usable format.

**Provided Interface**

**void sendCommandToMotor(byte commandValue, int motorNumber)**

Defintion: Sends a command value that the rover has received from the

command PC on to a specific motor as identified by motor a motor number.

Parameters: A byte, representing a value between 0-255 for how far the motor

needs to turn

Returns: none

Errors: IndexOutOfBoundsException

**void processBufferUnit(BufferUnit command)**

Definition: Unpacks the command that has been received over the network

into a command that can be used by the Arduino.

Parameter: A BufferUnit, representing command information and intended

destination that was being held in the Buffer module.

Returns: none

Errors: none

**BufferUnit processBuffer()**

Definition: Processes a command that has been stored in the Buffer Module queue.

These commands should include the command itself and the intended

target of the command whether it be motor or camera control.

Parameters: None

Returns: A BufferUnit which holds bytes of command info and a destination

motor.

Error: NoSuchElementException

**Required Interface**

**Rover Communication Module**

BufferUnit processCommand()

**2.2.3 Rover Camera Processing Module**

**Purpose**

The purpose of this module is to take the encoded video file received from the

camera and prepare it to be sent to the command PC.

**Rationale**

This module is needed in order to make sure that the encoded video is in a

format that can actually be sent through the rover communications module. Otherwise the video will not be able to be sent back to the command PC.

**Provided Interface**

**transferableVideo prepareVideo(encodedVideo videoFeed)**

Definition: Takes encoded video receieved from the camera and prepares it

for transmission across the internet to the command PC.

Parameter: encodedVideo, a representation of the format that the camera video

is being sent in

Returns: transferrableVideo, a representation of the format that can be

transmitted across the network to the command PC.

Error: None

**Required Interface**

Raw camera input data

1. **Abstract Data Types**

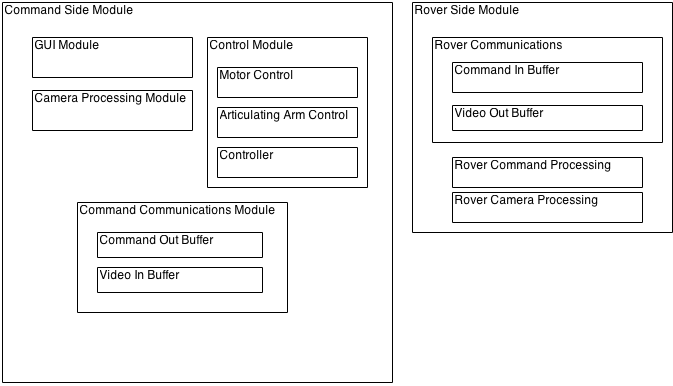
BufferUnit {

ArrayList<byte> commands

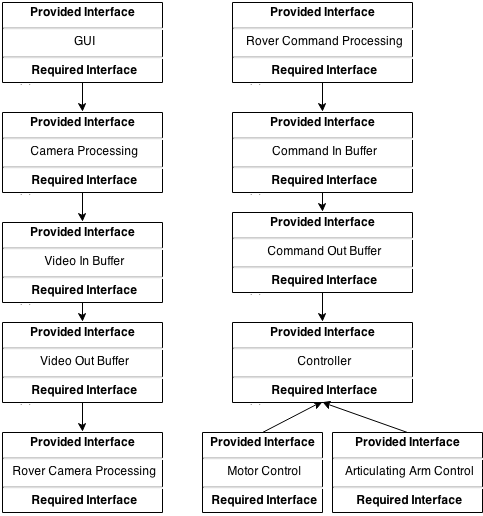
ArrayList<byte> intendedDestinations

}

1. **COMPRISES Diagram**



1. **USES Diagram**



1. **Integration Test Plan**

**Approach**

Our team will be using a bottom up approach. More testing will be done as more

motors and technology become available to us. We will start tests to make sure

our joystick works. Then we will move on to sending commands to another

computer. Eventually making our way to sending commands back and fourth

between the Rover on-board CPU.

**Order of Integration**

Below are the various test phases that will be performed as more technology for

the Rover becomes available to us.

|  |  |  |
| --- | --- | --- |
| **Integration Group** | **Driver(s) Needed** | **Stub(s) Needed** |

**Phase 1: Joystick Controls**

|  |  |  |
| --- | --- | --- |
| Move Forward | Joystick Driver | None |
| Move Backward | Joystick Driver | None |
| Move Left | Joystick Driver | None |
| Move Right | Joystick Driver | None |
| Set Speed | Joystick Driver | None |
| Change Camera | Joystick Driver | None |
| Move Arm | Joystick Driver | None |

**Phase 2: Communications Modules**

|  |  |  |
| --- | --- | --- |
| Send Signal | Communications Driver | None |
| Receive Signal | Communications Driver | Signal Stub |

**Phase 3: Motor Control**

|  |  |  |
| --- | --- | --- |
| Motor A Movement | Motor Driver | None |
| Motor A Rotation | Motor Driver | None |
| Motor B Movement | Motor Driver | None |
| Motor B Rotation | Motor Driver | None |
| Motor C Movement | Motor Driver | None |
| Motor D Movement | Motor Driver | None |
| Motor E Movement | Motor Driver | None |
| Motor E Rotation | Motor Driver | None |
| Motor F Movement | Motor Driver | None |
| Motor F Rotation | Motor Driver | None |
| Arm Movement | Motor Driver | None |
| Arm Rotation | Motor Driver | None |

**Phase 4: Video Feeds**

|  |  |  |
| --- | --- | --- |
| Send Video | Video Driver | None |
| Receive Video | Video Driver | Video Stub |

**Driver Definitions:**

**Joystick Driver:** This driver tests the functionality of the joystick. It will allow commands to be given from the joystick and visually display if they are being received by the program.

**Communications Driver:** This driver tests the functionality of our communications. It will create signals to be packaged and sent/received between the Command Station and the Rover.

**Motor Driver:** This driver tests the functionality of our Motors. It will provide necessary signals to cause each motor to move or rotate.

**Video Driver:** This driver tests the functionality of the video streaming capabilities.

**Test Cases:**

**Phase 1: Joystick Controls**

**ID:** Move Forward

**Description:** Test ability to send Move Forward signal to computer.

**Data Input:** None

**Expected Output:** Notification that the correct signal was successfully received.

**Environmental Constraints:** Properly functioning joystick.

**Testing Constraints:** None

**ID:** Move Backward

**Description:** Test ability to send Move Backward signal to computer.

**Data Input:** None

**Expected Output:** Notification that the correct signal was successfully received.

**Environmental Constraints:** Properly functioning joystick.

**Testing Constraints:** None

**ID:** Move Left

**Description:** Test ability to send Move Left signal to computer.

**Data Input:** None

**Expected Output:** Notification that the correct signal was successfully received.

**Environmental Constraints:** Properly functioning joystick.

**Testing Constraints:** None

**ID:** Move Right

**Description:** Test ability to send Move Right signal to computer.

**Data Input:** None

**Expected Output:** Notification that the correct signal was successfully received.

**Environmental Constraints:** Properly functioning joystick.

**Testing Constraints:** None

**ID:** Set Speed

**Description:** Test ability to send Set Speed signal to computer.

**Data Input:** None

**Expected Output:** Notification that the correct signal was successfully received.

**Environmental Constraints:** Properly functioning joystick.

**Testing Constraints:** None

**ID:** Change Camera

**Description:** Test ability to send Change Camera signal to computer.

**Data Input:** None

**Expected Output:** Notification that the correct signal was successfully received.

**Environmental Constraints:** Properly functioning joystick.

**Testing Constraints:** None

**ID:** Move Arm

**Description:** Test ability to send Move Arm signal to computer.

**Data Input:** None

**Expected Output:** Notification that the correct signal was successfully received.

**Environmental Constraints:** Properly functioning joystick.

**Testing Constraints:** None

**Phase 2: Communications Module**

**ID:** Send Signal

**Description:** Test the ability to package and send a signal to a designated IP.

**Data Input:** A signal to mimic those inputted from the joystick.

**Expected Output:** Notification that the signal was successfully sent.

**Environmental Constraints:** Network is functioning properly.

**Testing Constraints:** None

**ID:** Receive Signal

**Description:** Test the ability to unpack and read signal.

**Data Input:** Packaged signal to mimic format of sent signal.

**Expected Output:** Notification that the correct signal was received successfully.

**Environmental Constraints:** Network is functioning properly.

**Testing Constraints:** None

**Phase 3: Motor Control**

**ID:** Motor A Movement

**Description:** Test out the ability of Motor A

**Data Input:** Move forward command

**Expected Output:** Motor will move forward

**Environmental Constraints:** None

**Testing Constraints:** We don’t know when we are actually going to get a

prototype

**ID:** Motor A Rotation

**Description:** Test the rotation ability of the Motor

**Data Input:** Number of rotations

**Expected Output:** Actual number of rotations on motor A

**Environmental Constraints:** None

**Testing Constraints:** We don’t know when we are actually going to get a

prototype

**ID:** Motor B Movement

**Description:** Test out the ability of Motor B

**Data Input:** Move forward command

**Expected Output:** Motor will move forward

**Environmental Constraints:** None

**Testing Constraints:** We don’t know when we are actually going to get a

prototype

**ID:** Motor B Rotation

**Description:** Test the rotation ability of the Motor

**Data Input:** Number of rotations

**Expected Output:** Actual number of rotations on motor B

**Environmental Constraints:** None

**Testing Constraints:** We don’t know when we are actually going to get a

prototype

**ID:** Motor C Movement

**Description:** Test out the ability of Motor C

**Data Input:** Move forward command

**Expected Output:** Motor will move forward

**Environmental Constraints:** None

**Testing Constraints:** We don’t know when we are actually going to get a

prototype

**ID:** Motor D Movement

**Description:** Test out the ability of Motor D

**Data Input:** Move forward command

**Expected Output:** Motor will move forward

**Environmental Constraints:** None

**Testing Constraints:** We don’t know when we are actually going to get a

prototype

**ID:** Motor E Movement

**Description:** Test out the ability of Motor E

**Data Input:** Move forward command

**Expected Output:** Motor will move forward

**Environmental Constraints:** None

**Testing Constraints:** We don’t know when we are actually going to get a

prototype

**ID:** Motor E Rotation

**Description:** Test the rotation ability of the Motor

**Data Input:** Number of rotations

**Expected Output:** Actual number of rotations on motor E

**Environmental Constraints:** None

**Testing Constraints:** We don’t know when we are actually going to get a

prototype

**ID:** Motor F Movement

**Description:** Test out the ability of Motor F

**Data Input:** Move forward command

**Expected Output:** Motor will move forward

**Environmental Constraints:** None

**Testing Constraints:** We don’t know when we are actually going to get a

prototype

**ID:** Motor F Rotation

**Description:** Test the rotation ability of the Motor

**Data Input:** Number of rotations

**Expected Output:** Actual number of rotations on motor F

**Environmental Constraints:** None

**Testing Constraints:** We don’t know when we are actually going to get a

prototype

**ID:** Arm Movement

**Description:** Test the ability to move the Articulating Arm.

**Data Input:** Degrees of movement.

**Expected Output:** Arm to move as expected.

**Environmental Constraints:** None

**Testing Constraints:** Prototype not yet received.

**ID:** Arm Rotation

**Description:** Test the ability to rotate the Articulating Arm.

**Data Input:** Degrees of rotation.

**Expected Output:** Arm to rotate as expected.

**Environmental Constraints:** None

**Testing Constraints:** Prototype not yet received.

**Phase 4: Video Feeds**

**ID:** Send Video

**Description:** Video feed is packed and sent out to a specific location.

**Data Input:** Video feed

**Expected Output:** Notification that the feed was sent successfully.

**Environmental Constraints:** Proper 4G network connection.

**Testing Constraints:** None

**ID:** Receive Video

**Description:** Video feed is received and viewable.

**Data Input:** Packaged video

**Expected Output:** Display video

**Environmental Constraints:** Successful 4G network connection

**Testing Constraints:** None