SoRo3

There are currently 2 mission control (MC) computers and 1 rover computer that handles 3 microcontrollers (not all microcontrollers have to be plugged in for the system to work). MC1 was for controlling the wheels, while MC2 was for controlling the drill or the arm. Xbox controllers were plugged into the mission control computers, which used the controller’s inputs to build a message, which was to eventually be delivered to an Arduino. Each Arduino microcontroller controlled a different subsystem, and therefore all had to receive different data. There is also the GPS module that connects to the rover router and communicates through what is basically UDP.

Here is a map of the way information travels in last year’s (17-18) design:

Mission control 1

Rover computer (Cameras, Autonomous)

Drive Arduino

UDP

UDP

MC Router (antenna)

Rover Router

Arm Arduino

UDP

Antennas work as a network bridge

Rover computer (UDP hub)

Science (drill) Arduino

Serial

UDP

Mission control 2

**Two types of communication: serial and UDP**

Serial is over a usb cable, and for our purposes it will always be between a computer and an Arduino (or other microcontroller). On Linux, the serial ports are called /dev/ttyUSB0, /dev/ttyUSB1, etc. *We currently use 9600 Baud for the communication rate.*

UDP is over the internet, and for our purposes will be between the mission control computers and one of the rover’s onboard computers. We use UDP instead of TCP because you do not need to establish a handshake to send data, as there are no hosts/clients. You just send out information to a certain IP address and the computer on that IP will receive it if it can. This is useful because we do not need to restart any programs if the signal is lost momentarily, even if a router were to be reset (as long as the IP’s do not change).

Mission control 1: Sends wheel control messages to rover computers

Mission control 2: Sends arm control messages to rover computers

*Currently, no data messages are actually sent back to mission control, but they will have to in the future.*

Rover (UDP hub): receives from all other computers, currently does not send data to other computers with UDP (it will have to for the science data)

Rover (Cameras/Autonomous): Sends data to rover UDP hub for autonomous control and sends video stream (via udp) to MC computers

The messages sent will have the same format and content regardless of the type of communication. Usually a message is sent from a mission control computer to a rover computer over UDP, which is then passed though to the correct Arduino over serial.

**Message Formats:**

Every communication is a string of individual bytes, each carrying a different piece of information. A single byte is a number that can range from 0 to 255 if unsigned, or -127 to 126 if signed. There are no commas or any other characters that separate these bytes.

The Arduino of each subsystem has a unique ID so that the rover computer knows which Arduino to send data to.

There are a few types of messages that can be sent.

**Message to control a subsystem:**

The Arduinos that control the wheels, arm, and drill receive these. They can be created by the MC computers (in the case of human operation) or by the rover computer (autonomous operation). If the rover computer receives these messages it will pass them through to the correct Arduino without modifying the message.

Format: **[start transmission: control][device id][data (multiple bytes)][hash]**

* Start transmission
  + Value is -127 *(signed 8 bit integer – signed char or int\_8t)*
  + Lets the Arduino/computer know this is the beginning of a new message
* Device id
  + unsigned 8 bit integer – unsigned char or uint\_8t
  + Unique ID of the Arduino the message is intended for
  + Arduinos will ignore the message if the ID does not match, and will respond with their correct ID (this message is detailed below)
  + Wheels/Drive = 0
  + Arm = 1
  + Science = 2
* Data
  + Multiple bytes
  + Different length and format for each subsystem (see below)
* Hash
  + signed 8 bit integer – signed char or int\_8t
  + The average value of all the data bytes
  + Is used to verify the data. The Arduino calculates the average then compares it to this byte to make sure the data is not corrupted. The Arduino will ignore the message if they don’t match.

There is a different data format for each subsystem, each being a different amount of bytes. **These will be different for the new design, and I will document them as we go.**

* Drive/Wheels (also controls the moveable camera):
  + **[ [overdrive][left wheels][right wheels][gimble tilt][gimble pan] ]**
  + Overdrive = 1 or 0. Not used, so its always 0 *(unsigned char/uint8\_t)*
  + Left wheels = [-90, 90] speed of the left wheels *(signed char/ int8\_t)*
  + Right wheels = [-90, 90] speed of the right wheels *(signed char/ int8\_t)*
  + Gimble tilt = [-5, 5] speed at which to tilt the camera *(signed char/ int8\_t)*
  + Gimple pan = [-5, 5] speed at which to pan the camera *(signed char/ int8\_t)*
  + **Entire message example:**
    - [-127][0][0][90][70][0][0][36]
    - Drives forward with the right wheels spinning faster than the left. The camera is not moving
    - You would send this data to the drive Arduino serial port at 9600 Baud
* Arm:
  + **[ [shoulder][elbow][wrist 1][wrist 2][claw left][claw right] ]**
  + Shoulder = [40, 135] length of the shoulder actuator *(unsigned char/uint8\_t)*
  + Elbow = [40, 135] length of the elbow actuator *(unsigned char/uint8\_t)*
  + Wrist 1 = speed to change the pitch of the wrist *(signed char/ int8\_t)*
  + Wrist 2 = speed to change the roll of the wrist *(signed char/ int8\_t)*
  + Claw left = [0, 180] angle of the left claw servo *(unsigned char/uint8\_t)*
  + Claw right = [0, 180] angle of the right claw servo *(unsigned char/uint8\_t)*
* Science/Drill: (I am unsure about some of these)
  + **[ [move direction][move speed][spin speed][overdrive][fan speed] ]**
  + Move direction = Direction to move the drill actuator vertically – 2 is not moving, 0 is down, 1 is up *(unsigned char/uint8\_t)*
  + Move speed = [240] speed to move the drill actuator vertically *(unsigned char/uint8\_t)*
  + Spin speed = [-90, 90] speed to spin the drill bit *(signed char/ int8\_t)*
  + Overdrive is probably not used *(signed char/ int8\_t)*
  + Fan speed = [34] speed of the fan, which is the power of the vacuum *(signed char/ int8\_t)*

**Message to set the device’s ID:**

Each serial device (Arduino) has a unique ID, and each control message says the ID of the device to send it to. However, the rover computer initially does not know which serial device has which unique id, so it has to be told. This is a message created by an Arduino to be sent over serial to the rover computer. The Arduino does this when it receives a control message that does not have the correct id. Currently, the rover computer initially has some arbitrary number for the ID of each serial device and will send every control message to all of those devices (which will be ignored by the device) until the ID’s are corrected.

Format (pretty simple): **[start transmission: ID][device ID]**

* Start transmission
  + Value is -126
  + Lets the rover computer know that this message is correcting the ID of the device that sends it
* Device ID
  + The correct ID of the device that sent this message
  + Wheels/Drive = 0
  + Arm = 1
  + Science = 2

**Message to record data:**

The science package sent serial data back to the rover computer that needed to be recorded. It literally just prints each line of a .csv file, one line at a time, ASCII format. Every time the rover computer received serial data that was not one of the message types above, it would assume it was a data message and print it to a file.

More message types were planned, such as sending accelerometer data back to MC, but they were never used.

SoRo4

There will *probably* be only one mission control computer (MCC) with two sessions (one screen, keyboard, mouse for each session). We *might* abandon the router on the rover and use a switch instead. There are two Rover computers (RC1, and RC2). RC1, the UDP Hub, is responsible for receiving UDP messages from mission control or RC2 and sending them to the correct subsystem controllers (Arduinos) to move the rover. In autonomous mode, RC2 will send UDP messages to RC1 to operate the rover. RC2 is connected to the cameras and an Arduino that sends back serial data about the autonomous sensors (planar LIDAR and other stuff), and is responsible for autonomous mode and streaming video to mission control. The GPS module is connected to the rover network and is capable of sending data to MCC or either of the RC’s.

Here is a map of the way information travels:

Swift GPS Module

Autonomous Sensors Arduino

Master Arm Arduino

Rover computer 2 (Cameras, Autonomous)

UDP

Serial

Serial

Drive Arduino

UDP

Mission control

MC Router (antenna)

Rover Router

Antennas work as a network bridge

UDP

Slave Arm Arduino

UDP

Rover computer 1 (UDP hub)

Science Arduino

**Message to control a subsystem:**

The Arduinos that control the wheels, arm, and science package will receive these. They can be created by the MC computers (in the case of human operation) or by the camera rover computer (autonomous operation). If the udp hub rover computer receives these messages (through UDP) it will pass them through to the correct Arduino (through serial) without modifying the message.

Format: **[start transmission: control][device id][data (multiple bytes)][hash]**

* Start transmission
  + Value is -127 *(signed 8 bit integer – signed char or int\_8t)*
  + Lets the Arduino/computer know this is the beginning of a new message
* Device id
  + unsigned 8 bit integer – unsigned char or uint\_8t
  + Unique ID of the Arduino the message is intended for
  + Arduinos will ignore the message if the ID does not match, and will respond with their correct ID (this message is detailed below)
  + Wheels/Drive = 0
  + Arm = 1
  + Science = 2
* Data
  + Multiple bytes
  + Different length and format for each subsystem (see below)
* Hash
  + signed 8 bit integer – signed char or int\_8t
  + The average value of all the data bytes
  + Is used to verify the data. The Arduino calculates the average then compares it to this byte to make sure the data is not corrupted. The Arduino will ignore the message if they don’t match.

There is a different data format for each subsystem, each being a different amount of bytes.

* Drive/Wheels (also controls the moveable camera):
  + **[ [booleans][left wheels][right wheels][gimble tilt][gimble pan] ]**
  + Booleans = each bit is a different boolean *(unsigned char/uint8\_t)*
    - 20 bit: center axle break on(1)/off(0)
    - 21 bit: front wheels modifier. If 1, then only the front wheels will spin
    - 22 bit: rear wheels modifier. If 1, then only the rear wheels will spin
      * If the front and rear wheel modifiers are both 1, then the front and rear wheels will spin in opposite directions (for unfolding the rover)
    - 23 bit: reset camera gimbal. If 1, then the arduino will set the angles of the gimbal servos to default positions. (Gimbal will have to be rebuilt without continuous rotation servos)
  + Left wheels = [-90, 90] speed of the left wheels *(signed char/ int8\_t)*
  + Right wheels = [-90, 90] speed of the right wheels *(signed char/ int8\_t)*
  + Gimble tilt = [-5, 5] speed at which to tilt the camera *(signed char/ int8\_t)*
  + Gimple pan = [-5, 5] speed at which to pan the camera *(signed char/ int8\_t)*
  + **Entire message example:**
    - [-127][0][0][90][70][0][0][36]
    - Drives forward with the right wheels spinning faster than the left. The camera is not moving
      * You would send this data to the drive Arduino serial port at 9600 Baud
* Arm
  + **[ [base pos][shoulder pos][elbow pos][wrist pitch][wrist roll CCW][wrist roll CW][close hand][open hand] ]**
  + Base position = [0,255] Comes from a potentiometer, 0 corresponds to 0 degrees, 255 corresponds to 270 degrees *(unsigned char/uint8\_t)*
  + Shoulder position = [0,255] Comes from a potentiometer, 0 corresponds to 0 degrees, 255 corresponds to 270 degrees *(unsigned char/uint8\_t)*
  + Elbow position = [0,255] Comes from a potentiometer, 0 corresponds to 0 degrees, 255 corresponds to 270 degrees *(unsigned char/uint8\_t)*
  + Wrist pitch = [0,255] Comes from a potentiometer, 0 corresponds to 0 degrees, 255 corresponds to 270 degrees *(unsigned char/uint8\_t)*
  + Wrist roll CCW = is a button. If 1, the wrist will be rotating counter clockwise
  + Wrist roll CW = is a button. If 1, the wrist will be rotating clockwise
  + close hand = is a button. If 1, the claw will be opening
  + open hand = is a button. If 1, the claw will be closing
  + It will probably be the exact same message coming from the master arm going through all the computers to the slave arm
  + **Potential change: make all the buttons singular bits of one byte**