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)

Drive Arduino

Serial

Serial

UDP

Mission control

MC Router (antenna)

Rover Router

UDP

Slave Arm Arduino

UDP

UDP

Antennas work as a network bridge

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**