**Sr Design II – Communications Final Report**

*An organized zipped directory of all cautions, logins/passwords, device IP addresses, Arduino .INO files, dependencies, board pinouts for the custom communications shield, Python programs, phone application backup, previous (uncut) communications reports, and various how-to documents have been compiled and should be provided to anyone continuing this project. The user must read and familiarize themselves with all documentation before using or modifying any programs/devices.*

**Table of Contents –**

1. **Communications Overview**
2. **Equipment & Radiating Devices**
3. **Programs Required**
4. **MQTT Systems Overview**
5. **M2 Radio Overview**
6. **CAN Overview**
7. **Timeline Review & Reflection**
8. **Communications Appendix**

**List of Figures –** *(for EECOM)*

**EECOM Figure 1:** list of radiating devices

**EECOM Figure 2:** Communications data flow chart

**EECOM Figure 3:** ESP32 communication controller pinout ESP32 Comms motor controller pinout.

**EECOM Figure 4:** IoT MQTT Panel Pro application for android. Custom designed for remote readouts and control during testing.

**EECOM Figure 5:** AK motor control python shell for any system. "val" takes 6 inputs as position or velocity based on mode setting for both drive motors but is easily set for arm by changing to the arm topic. Caution of position and velocity when operating arm.

1. **Communication Overview**

The communication system works as designed and all components communicate to satisfy essential functionality. All components include the radios, IP camera, GPS, IMU, various remote-control methods and MQTT methods used are stable. There are two concerns with communications, one is with the CAN motor arrays. The method used to split the CAN bus which is parallel instead of series. Measurements of line impedance provide no problems, but results are inconsistent when testing where readouts or controls can be delayed, not respond, or they could work as intended. Wiring is believed to be the problem and requires further investigation at the time of reporting. The computer science team developing the automated system in ROS2 was incompatible with Ubuntu released later than 18.04, which is the latest operating system developed for the onboard Nvidia Jetson TX2. A container was needed, but access to the network within the container has yet to be bridged. Access to communications for manual functionality is unimpeded by this as all communications are directly exchanged with the respective microcontroller subscribed to the topic. Communications seeks to fix autonomous communications before the end of the term but in the last couple weeks focus is on the inconsistent motor response and radio functionality as we get the ability to place all components on the final design with the battery pack. Reliable manual controls are of the highest importance so autonomous functionality may not be available for demonstration.

1. **Equipment & Radiating Devices**

Timeline

Description automatically generated

Figure 1: List of radiating devices.

The final equipment used specific to communications was the originally planned Ubiquiti Rocket M2 with the OEM antennas and amended for short range missions with the generic 8dBi antennas. The SparkFun ZED F9P was the GPS of choice instead of the Titan XA1110 due to immediate accuracy and increased potential accuracy if RTK capabilities are configured on future rovers. An overview of the radiating devices in this list is provided in figure 1. This list will be useful when reporting for FCC compliance when entering the URC competition.

Diagram

Description automatically generated

Figure 2: Comm data flow chart.

The implemented communication network is visualized in figure 2. The ESP32 devices were the final choice for controllers to reduce wiring, ease servicing, increase communication speed to the devices, reduce restart times, allow for dynamic number of control devices/platforms, and ease the limitations of ROS autonomous communication with MCU devices. A communications shield was developed to ensure secure connections of all devices interacting with the ESP32 devices. The motor shield is depicted in figure 3 with the pinouts used. The sensors use a motor shield in which the SPI does not properly work, but the motors’ shields work as designed. The Altium files are included in the program files which should be available for future teams.

Map

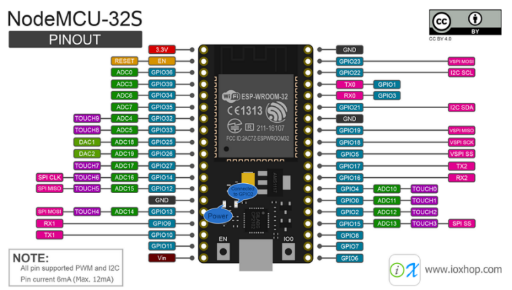
Description automatically generated with medium confidence

Figure 3: ESP32 Comms motor controller pinout. Similar pinout for sensor comm controllers but their SPIs DO NOT work due to being an earlier prototype.

1. **Programs Required**

Multiple programs are used for testing the equipment using the MQTT communication system. One of the most useful methods of control for testing and controller dashboard reference is the phone application. The application is MQTT based, allowing for mode and value control of the motors while providing feedback from the AK60 motors, IMU, ultrasonic sensors, and GPS. The application blinks due to connection errors, but this phenomenon only occurs within the application and not the entire MQTT network. This application is more stable on DHCP networks for testing. Figure 4 depicts the dashboard views available to the controller from the phone application. The use and setup of widgets is relatively intuitive therefore is not explained in detail within the reference material. Download the IoT MQTT Panel application and restore it from the .JSON file provided in the project backup provided.

A screenshot of a computer screen

Description automatically generated with medium confidenceGraphical user interface, application

Description automatically generated with medium confidenceA screenshot of a computer screen

Description automatically generated with medium confidenceA screenshot of a computer screen

Description automatically generated with medium confidence

Figure 4: IoT MQTT Panel Pro application for android. Custom designed for remote readouts and control during testing.

One of the most dynamic is a Python shell developed for testing the motor arrays remotely where simply exchanging the topics inside the corresponding script will allow for various motor controls such as arm or any other control device as a useful template for future devices. The current design, given the MCU design, takes an array of 6 values representing each motor for synchronous motor controls. The true value of the shell is that it works on any computer platform supporting Python. The terminal interface with all current options for the Python shell program is provided in figure 5. The Linux gamepad controller code is best, and safest to use with the arm. This code uses low-speed velocity settings since position control may lead to damage if the position is accidentally zeroed or improperly initialized.

Text

Description automatically generatedText

Description automatically generatedText

Description automatically generated

Figure 5: AK motor control python shell for any system. "val" takes 6 inputs as position or velocity based on mode setting for drive motors.

1. **MQTT Systems Overview**

The MQTT system works as the backbone of the communication system. It has simplified ROS communications which is offered for very limited MCUs, reduced wiring hassle/time, and provided the ability to use a range of programs for monitoring and testing the devices and components on the network. A bridge template was produced for the computer science team which included two headers and four lines of code including network connection setup, a six-value array for motor values and a callback header for the acknowledgement and sensor data. Unfortunately, accumulating all topics and functions would be too time consuming, but the four INO scripts will provide all available topics in the subscribe function and locations which publish the topics in these files. The Mosquitto MQTT broker provides parallel communication with control and sensor updates of 20Hz and greater realized during testing. This meets the response time requirement outlined by the URC guidelines for manual controls.

1. **Ubiquiti Radio Overview**

The Ubiquiti radios have worked as intended for all testing. The largest concern with the network is testing with the battery as the supply, and the radio shares the 24V power bus where power has agreed to supplement an extra device for current limiting and voltage regulation to dissipate power before it reaches radio. This will prevent burning out the radio and ensuring a stable connection as the motors and electronic switching within the system can disrupt the radio. Another concern is if NLOS does not meet requirements, future rovers are encouraged to explore building a system to forward all port activity to a second 900MHz radio system capable of 14MBPs bidirectional and full duplex communications. In this case, video data may require reduced framerate or resolution to maintain speed due to reduced bandwidth. This is likely unnecessary but would provide a more reliable radio system if realized. Line of sight (LOS) has provided stable 720p video connection with 5dB loss at 0.33km with maximum EIRP gain (36dBm to 53dBm based on antenna gain and amplifier ratio) set, MCS2, and maximum distance set. When referencing the estimate from the second communications report 20dB LOS of loss was expected, providing three times more loss than measured. This should ensure a kilometer LOS minimum and depending on obstacles for NLOS, should meet competition requirements given data despite the ability to test conditions like the competition grounds.

All devices on the Ubiquiti network have IP addresses 192.168.42.XXX for XXX is 0-255. A list of current IP addresses assigned is provided with the project file package. It is essential that each device has its unique address set to prevent cross-communication. Connecting to the network requires static IP assignment for all devices connected. All devices effectively connect to the network with rare interruptions. Some devices may require reset if network errors occur, which can be done from the application where full functionality for manual functions can be achieved in 30 seconds. It should be noted that the rover radio is the AP (access point) so that the ESP32 devices may wirelessly connect to the network as the STA (station) radio does not have this capability. The control side uses the STA radio which requires LAN connections for local devices to utilize the Ubiquiti network. Given research and testing, the configurations provided appear to be the most reliable, high-performance configuration for the network.

1. **CAN Overview**

CAN communications have proven to be the most difficult communications to realize. The CAN data was inaccessible for the NEO550 but was realized for the AK60-6 motors for both position and velocity with full readouts. More functionality with the AK60-6 using CAN is possible but unnecessary. Using the OEM wires, the communication works as designed. The line uses a daisy chain method, bussing all motors in series. This is not reasonable for our project as the center mount would require a set of data wires fed in and out where a shared node is more realizable, less wiring, and reduced risk of tangle. The characteristic line impedance is 120 as required for a stable CAN bus, but communications have had minor stability issues. During installation some communications would be delayed, will only send sensor data, or all will work as intended. Clamping junction node changed responsiveness, indicating wiring issues to investigate despite line impedance showing the correct value at each node.

1. **Timeline Review & Reflection**

The timeline for communications has been achieved as expected until final assembly stage. Power and physical components for mounting are not available, so final incorporation of all devices in the network is not reportable. Communication wanted power data available for the dashboard, but power distribution systems were not tested in time to incorporate the internal readings from the BMS. Communications can be improved as explained in section V, but it is suggested to use the Ubiquiti system in conjunction with any improvements. Given the results provided in section V this radio system will be more than sufficient for all remotely controlled missions, it’s manageable to set up given the resources provided, and not complex to develop within for future rover teams.

1. **Communications Appendix**

*This data is also available as notepad readmes in the program zip file.*

**STATIC IP ADDRESSES:**

* TX2: 192.168.42.33 (May need to reset to this value)
* Rover M2 (AP): 192.168.42.10
* Base M2 (STA): 192.168.42.11
* Drive ESP32: 192.168.42.93
* Arm ESP32: 192.168.42.83
* GPS ESP32: 192.168.42.45
* Sensors ESP32: 192.168.42.63
* IP Camera: 192.168.42.90

**PASSWORDS:**

* Ubiquiti Gateway: user “ubnt” pass “LynxRobo”. If unable to access, contact Ian Dwyer or attempt to reset network settings as outlined in communications report 2.
* Ubiquiti Wifi: “LynxRobo”
* MQTT Broker: None, insecure 1883 port for unimpeded performance
* TX2: “yahboom”

**COMMS/CONTROLS CAUTIONS:**

* Under penalty of law, do not exceed maximum EIRP power for 2.4GHz unlicensed band when adjusting gain on radio through the gateway or appending non-OEM antennas to the radio.
* Caution of operating arm/claw given the slip ring was the wrong kind and cords at shoulder were not considered given the mechanics of the joints. The arm can only rotate within a 2pi radian range or cords may tug at the shoulder.
* Caution operating claw as only 5 rotations of the NEO motor are required to open and close the claw where CAN positional data was not accessible given the data from the manufacturer of the controller.
* The radio should be placed on a separate current controlled device parallel to the motors to choke power kicked back from the motors since it shares the same 24V bus. Changes in motors will blackout radio if not properly configured.
* Devices at the control side are connected to the STA, therefore only LAN connections are possible without another router, which is not suggested.
* Caution using high kp/kd values on the AK60 motors.
* Do not use large steps in velocity or position when testing AK60 motors on a power supply which does not have a regulating unit to dissipate power kicked back from reverse torque.
* It is not suggested to do steps larger than pi radians with the AK60 motors.
* Caution using the pullup resistors on the esp32 comms control boards as improper or unnecessary use will prevent communication of I2C.
* Do not use SPI with the esp32 communication boards that are connected to the sensors. These ports are not properly configured as it was an initial prototype. Also, caution using the GPIO ports on these boards without ensuring that power and ground are bridges as to circumvent another design defect.
* Caution configuring controller for analog control as response of motors is spontaneous due to blocking in event class required for python controller script. Arduino controller should be designed as a buffer between the pad and computer with a python script that checks these values to get X and Y coordinates to use the analog controls properly and safely.
* Caution using high power or long-distance settings on the Ubiquiti when in close range.
* Do not use AirMax as it will block non-Ubiquiti devices from joining the network.
* All devices must be set up using manually defined static IP settings for each device otherwise connections will be either unstable or will not connect. Currently only Windows appears to use DHCP protocols capable of joining without setting the IP but connection is slow.
* Do not place GPS antenna near Ubiquiti antenna as it will act as an asynchronous waveguide and disrupt signals.

**DEPENDANCIES:**

*Python:*

* Paho MQTT
* evdev on linux (note that every new install of evdev requires remapping of controller buttons)
* json

*Arduino:*

* Mosquitto MQTT broker (type mosquitto -v to check if running)
* Arduino Uno Wifi Dev Ed Library by Arduino
* Adafruit BNO055 by Adafruit
* Adafruit BusIO by Adafruit
* Adafruit Unified Sensor by Adafruit
* ArduinoJson by Benoit Blanchon
* SparkFun I2C GPS Reading and Control by Sparkfun
* SparkFun u-blox GNSS v3 by Sparkfun
* TinyGPSPlus by Mikal Hart
* Autowp-mcp2515 by autowp
* Wifi\_logins.h (custom credentials file placed in libraries or with local ino file)

*Place the following links in preferences before installing esp32 and pubsubclient:*

* https://github.com/knolleary/pubsubclient/releases/tag/v2.8
* https://raw.githubusercontent.com/espressif/arduino-esp32/gh-pages/package\_esp32\_index.json
* ESP32 board installed after including git repositories above.
* PubSubClient by Nick O’Leary (install after adding git repository above)

*Other Applications:*

* IoT MQTT Panel for Android (use json file provided in zip to load setup. Read all cautions before use of any controller program.)
* CubeMars rLink for setting CAN IDs (download, set english bottom left. See official manuals and video to familiarize with software before use for safety of the motors and other equipment)

**GAMEPAD MAPPING:**

* Home: arm mode
* Touchpad: drive mode
* Select: exit mode and reset

*Drive Mode:*

* Dpad up/down: move forward and back
* O: increase speed
* square: decrease speed
* Trigger Lower Right: Forward right
* Trigger Lower Left: Forward left
* Trigger Upper Right: Reverse right
* Trigger Upper Left: Reverse left

*Arm Mode:*

* *Dpad right/left: arm forward back*
* *Dpad up: arm forward back*
* *Dpad down: arm down*
* *O: Rotate claw right*
* *square: Rotate claw left*
* *triangle: elbow up*
* *X: elbow down*
* *Trigger Right: Tighten Claw*
* *Trigger Left: Loosen Claw*