
Dwight Look College of ENGINEERING
TEXAS A&M UNIVERSITY

# Team 51: Radio Mobile Foxbot
# Bi-Weekly Update 3

**Brady Lagrone**
**Sophia Panagiotopoulos**
**Miguel Segura**

**Sponsor: Kevin Nowka**
**TA: Fahrettin Ay**

# Project Summary

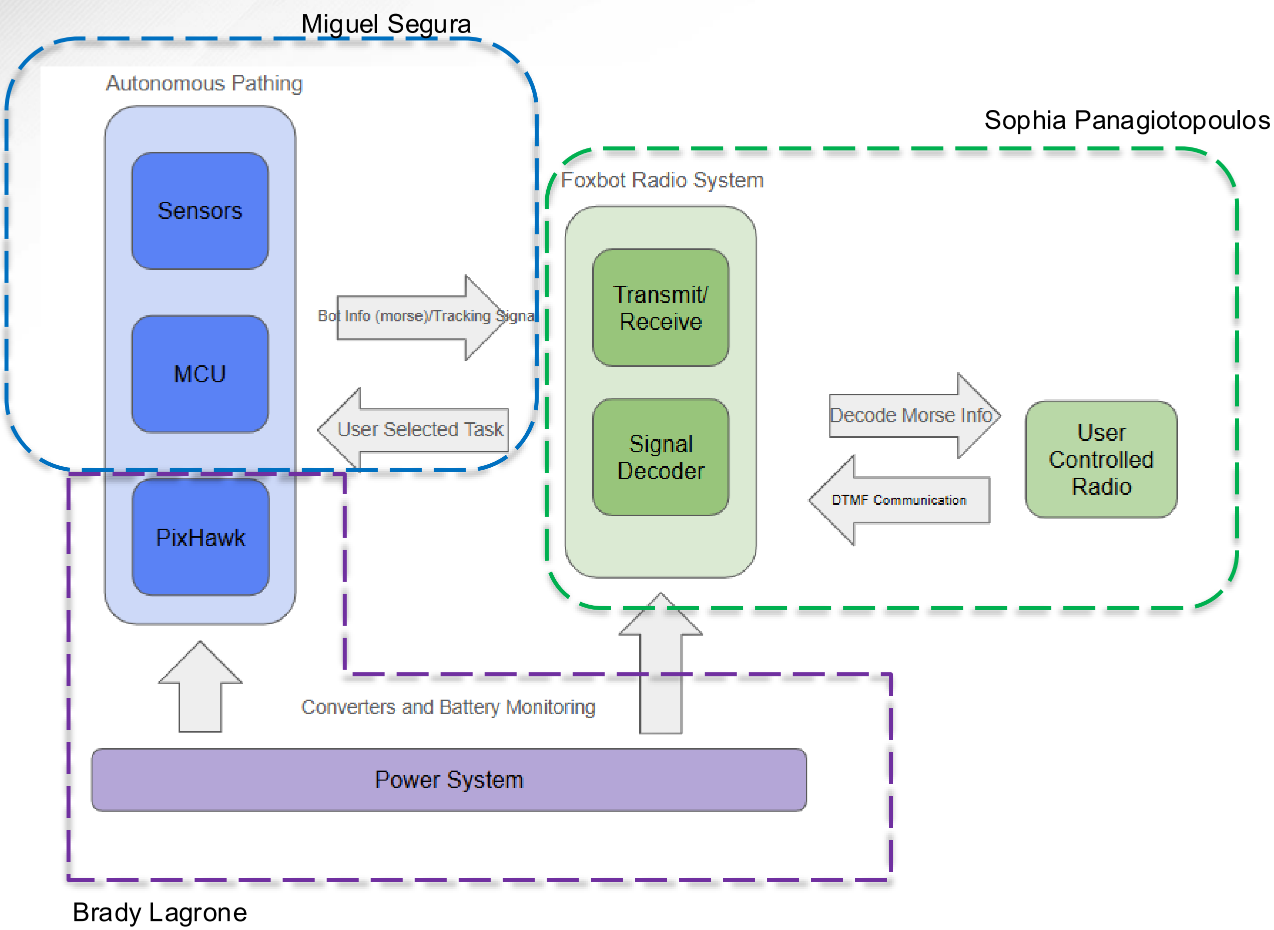
Purpose:

- Amateur Radio Directional Finding (ADRF) is a sport in which an individual tries to locate a hidden transmitting device. Our goal is to “hide” from an adversary Houndbot, who will be trying to locate our robot. Additionally, our robot will increase the potential of ARDF training and aid in the engineering of directional antennas.

Proposal:

- We have created a radio system coupled with a mobile robot that receives DTMF signals and preforms various functions based on the decoded signal.

# Project/Subsystem Overview




**Subsystem Goal Overview:**

**Autonomous Pathing System:**

- Robot can arrive at user decided destination while avoiding objects by using sensors and Pixhawk.
- MCU will send analog signal to radio transmit while controlling PTT function.
- MCU will receive and control batteries properties

**Power System:**

- Lippo batteries will be converted for component ratings.
- Battery monitor will send alert to MCU when voltage ratings are low.

**Radio System:**

- User can send DTMF tone to Foxbot which can be decoded to determine user selected settings (transmit for houndbot or battery health).
- Radio will transmit the formulated signal by the MCU. If the user has requested battery health, morse will be decoded.

# Project Timeline

| | | | | | | |
|---|---|---|---|---|---|---|
| Subsystem Designs and Testing (completed 01/27) | Integration of power and radio subsystem PTT (completed 2/9) | Integration of MCU and Motor Driver (to complete by 2/14) | Integration of power and radio ADC (to complete by 2/14) | All systems have been integrated (to complete by 3/7) | System Testing (to be completed 3/17) | Demo and Report (to complete by 4/14) |

# MCU Subsystem

**Miguel Segura**

| Accomplishments since last update ~17hrs | Ongoing progress/problems and plans until the next presentation |
|---|---|
| • ESP32 code developed to arm PixHawk and verified to work with tonal system<br>• MavLink autonomous base design coded to connect ESP32 to PixHawk to control Motor Controllers | • Continued issues integrating the GPS system of PixHawk with ESP32 (Unable to parse GPS coordinates from MavLink)<br>• Finalize integration with ESP32 with PixHawk (one successful design path)<br>• Integrate the PixHawk with Radio/Power Subsystems |

# MCU Subsystem

**Miguel Segura**

```c
void send_arm_disarm(bool arm) {

    mavlink_message_t msg;
    uint8_t buf[MAVLINK_MAX_PACKET_LEN];

    mavlink_msg_command_long_pack(
        ESP_SYSTEM_ID, MAV_COMP_ID, &msg,
        MAV_SYSTEM_ID, MAV_COMP_ID,
        MAV_CMD_COMPONENT_ARM_DISARM, // MAVLink command to ARM/DISARM
        0,  // Confirmation
        arm ? 1 : 0,  // Param1: 1 to arm, 0 to disarm
        0, 0, 0, 0, 0, 0  // Unused parameters
    );

    uint16_t len = mavlink_msg_to_send_buffer(buf, &msg);
    uart_write_bytes(UART_NUM, buf, len);

    ESP_LOGI(TAGG, "Sent MAVLink %s command", arm ? "ARM" : "DISARM");
}
```

## Arming Code

We arm the PixHawk through the ESP32 with the assistance of UART. With this we are able to integrate the key componenets of the radio subsystem to the control of the PixHawk

## Basic Autonomous Code (MavLink)

```c
void gps_only_autonomous_task(void *pvParameters) {
    int current_wp = 0;
    waypoint_t current_position = {37.7740, -122.4200};  // Placeholder; normally derived from GPS

    while (1) {
        // In a real implementation, update current_position from incoming GLOBAL_POSITION_INT messages

        // Compute distance to next waypoint
        float distance = compute_distance(current_position, waypoints[current_wp]);
        ESP_LOGI(TAG, "Distance to waypoint %d: %.4f", current_wp, distance);

        // If close enough, advance to next waypoint
        if (distance < 0.0005f) {
            current_wp = (current_wp + 1) % NUM_WAYPOINTS;
            ESP_LOGI(TAG, "Advancing to waypoint %d", current_wp);
        }

        // Compute desired velocity command (here, a fixed forward velocity)
        // In a real implementation, compute heading and project velocity vector accordingly.
        float vx = 0.5;  // 0.5 m/s forward
        float vy = 0.0;
        float vz = 0.0;

        send_mavlink_velocity_command(vx, vy, vz);

        vTaskDelay(pdMS_TO_TICKS(1000));
    }
}
```

Base Autonomous Design case to be configured after GPS signals are turned into available values.

```c
void receive_mavlink_task(void *pvParameters) {
    uint8_t data[512];
    int len;
    bool inside_packet = false;
    int packet_start = -1;
    int expected_length = 0;
    int received_bytes = 0;

    while (1) {
        len = uart_read_bytes(UART_NUM_1, data, sizeof(data), 100 / portTICK_PERIOD_MS);

        if (len > 0) {
            ESP_LOGI("UART", "Received %d bytes:", len);

            for (int i = 0; i < len; i++) {
                printf("0x%02X ", data[i]);

                // Detect MAVLink header
                if (data[i] == 0xFE || data[i] == 0xFD) {
                    packet_start = i;
                    inside_packet = true;
                    received_bytes = 1;  // We have 1 byte (the header)
                    expected_length = 8; // Minimum MAVLink v2 packet length
                    ESP_LOGI("UART", "MAVLink Header Found at index %d", packet_start);
                    continue;
                }
```

```c
                // Track MAVLink packet length
                if (inside_packet) {
                    received_bytes++;

                    // Payload length is the 2nd byte in MAVLink packet
                    if (received_bytes == 2) {
                        expected_length += data[i];  // Adjust expected length
                    }

                    // MAVLink messages should not exceed 280 bytes
                    if (received_bytes >= expected_length) {
                        ESP_LOGI("UART", "Full MAVLink packet detected (%d bytes)", received_bytes);

                        // Log entire MAVLink packet
                        printf("\nMAVLink Packet: ");
                        for (int j = packet_start; j < packet_start + received_bytes; j++) {
                            printf("0x%02X ", data[j]);
                        }
                        printf("\n");

                        // Try to parse MAVLink message
                        mavlink_message_t msg;
                        mavlink_status_t status;
                        bool parsed = false;

                        for (int j = packet_start; j < packet_start + received_bytes; j++) {
                            if (mavlink_parse_char(MAVLINK_COMM_0, data[j], &msg, &status)) {
                                ESP_LOGI("MAVLINK", "Parsed MAVLink Message ID: %d", msg.msgid);
                                process_mavlink_message(&msg);
                                parsed = true;
                            }
                        }
```

```c
                        if (!parsed) {
                            ESP_LOGE("MAVLINK", "MAVLink packet detected but could NOT be parsed");
                        }

                        // Reset tracking variables
                        inside_packet = false;
                        packet_start = -1;
                        received_bytes = 0;
                        expected_length = 0;
                    }
                }
            }
            printf("\n");
        } else {
            ESP_LOGW("UART", "No MAVLink Data Received");
        }

        vTaskDelay(pdMS_TO_TICKS(500));
    }
}
```

GPS Parsing Code – PixHawk and ESP32 are connected through UART. GPS signals are being sent to the ESP32 and when specific Hex values are detected ("0xFE" or "0xFD"). Begins the parsing detection system with MavLink open source code functions.

Once the header is found, the parsing begins. The current issue is the loss of information through the GPS in UART. There is active UART Hexbytes being sent to the ESP32, however, these bytes are invalid and are not properly being parced through the "if (mavlink_parse_char(MAVLINK_COMM_0, data[j], &msg, &status))

```c
void process_mavlink_message(mavlink_message_t *msg) {
    if (msg->msgid == MAVLINK_MSG_ID_GLOBAL_POSITION_INT) {
        mavlink_global_position_int_t gps;
        mavlink_msg_global_position_int_decode(msg, &gps);

        float lat = gps.lat / 1e7;
        float lon = gps.lon / 1e7;
        float alt = gps.alt / 1000.0; // mm to meters

        ESP_LOGI(TAGG, "GPS: Lat: %.7f, Lon: %.7f, Alt: %.2f m", lat, lon, alt);
    }
}
```

Processing MavLink data messages

# Power Subsystem

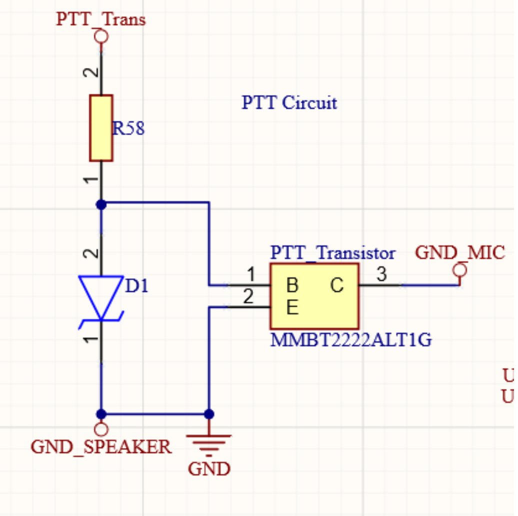
**Brady Lagrone**

| Accomplishments since last update<br>30 hrs of effort | Ongoing progress/problems and plans until the next presentation |
|---|---|
| • ESP receives and responds to walkie talkie dial tone in FEDC lab (~50 ft)<br>• ESP GPIO PTT works turning off and on when GPIO is triggered<br>• Removed static noise from DAC morse through radio<br>• Integration between Radio and ESP system complete (breadboard)<br>• On going integration with Pixhawk and ESP/Radio System | • Finish soldering and test PCB for full functionality<br>• Integrate with Pixhawk system with ESP<br>• Begin testing functionality of system (transmitting, driving and transmitting, stopping and going) |

## Dial Tone flag

```c
void vTaskMonitorBits(void *pvParameters) {
    while (1) {
        if(bit_action_flag == -1){

            // Read the state of each GPIO pin and calculate the decimal value
            for (int i = 0; i < NUM_PINS; i++) {⋯

            // Print the binary and decimal values
            printf("Binary: %d%d%d%d, Decimal: %d\n",
               gpio_get_level(pins[3]),  // MSB (Bit3)
               gpio_get_level(pins[2]),  // Bit2
               gpio_get_level(pins[1]),  // Bit1
               gpio_get_level(pins[0]),  // LSB (Bit0)
               decimal_value);

            // Check if the current value matches the previous value
            if (decimal_value == prev_decimal_value) {
                count++; // Increment the counter
            }
            // Else reset the count to 1
            else {
                count = 1; // Reset the counter if the value changes
            }

            // Update the previous value
            prev_decimal_value = decimal_value;

            // if condition that checks to see if bit_action_flag needs to be set
            // checks if the previous flag is not the same as current, dont do action on zero, and make sure there was a count of 3
            if(bit_prev_flag != decimal_value && decimal_value !=0 && (count >=3)){
                // set bit flag to the new decimal value
                bit_action_flag = decimal_value;
            }
```

## PTT Circuit



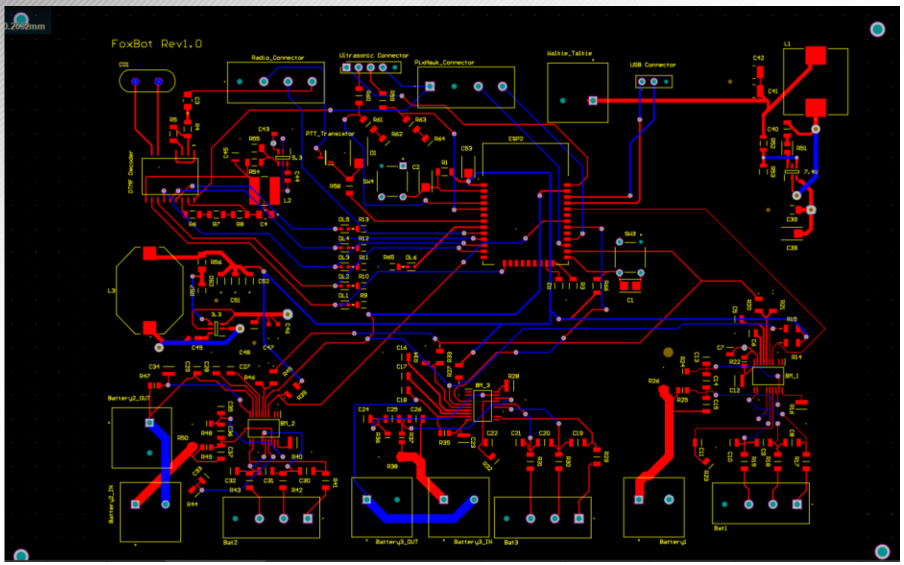

```
Binary: 0001, Decimal: 1
Binary: 0001, Decimal: 1
Binary: 0001, Decimal: 1
Executing Action 1, reading Cell1 Voltage
This is the HI: 0x10, and this is the Lo: 0x33
Cell 1 Voltage: 1.56 V
Letter: 1, Morse Code: .----
Letter: ., Morse Code: .-.-.-
Letter: 5, Morse Code: .....
Letter: 6, Morse Code: -....
  (Word Space)
Letter: V, Morse Code: ...-
Binary: 0001, Decimal: 1
```

## Battery Monitor to Morse Output

# Radio Subsystem

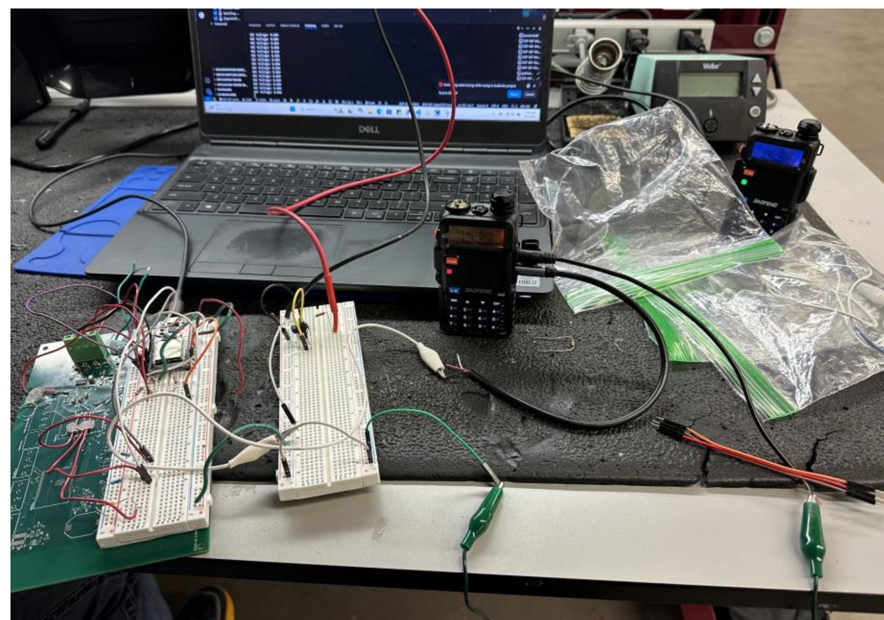
**Sophia Panagiotopoulos**

| Accomplishments since last update<br>30 hours of effort | Ongoing progress/problems and plans until the next presentation |
|---|---|
| - Ordered the integrated PCB and began soldering components.<br><br>Mostly completed integration with the power system:<br>- Debugged Radio/MCU system with Brady, eliminating PTT bug that was causing the radio to always be in PTT.<br>- Fine-tuned functional code block.<br>- Tested for decoding (input) and morse (output) range | - Finish soldering PCB and begin testing all circuits for intended functionality<br>- Any autonomous pathing control using DTMF tones<br>- Integrate with the motor control system so that the robot can be stopped by certain DTMF tones (allowing the ESP to overtake the PixHawk)<br>- Map certain tones to motor functionality and gps information<br>- Begin testing outside for further ranges |




Completed PCB Layout

PTT being enabled by IO -> shuts off when morse is finished so that the radio can listen for another tone.



Dwight Look College of ENGINEERING TEXAS A&M UNIVERSITY

# Validation Plan

| Paragraph | Test Name | Success Criteria | Methodology | Status | Responsible Engineer(s) |
|---|---|---|---|---|---|
| 3.2.1.1 | Transmisison Range | Radio on foxbot is able to pick up signals from user radio within the specified foxhunt region | Radio on the foxbot is able to recognize a sent signal from the user radio by outputting the tone from its speaker. | TESTED | Sophia |
| 3.2.1.2 | Operation Time | The foxbot will operate for at least 1 hour | The batteries are able to run the system for 1 hour when left alone. | UNTESTED | Brady |
| 3.2.1.3 | DTMF Decoding Accur | A binary value, output of the decoder, will correspond to the keypad number sent by the | LEDs will show the bits that are high or low, indicating the value in binary. | TESTED | Sophia |
| 3.2.1.4 | Intutiveness of System | The system shall be straightforward and the user shall be able to troubleshoot if there is an issue | It should take the user not more than 10 minutes to load the path and start up the foxbot. | UNTESTED | Full Team |
| 3.2.1.5 | Pathing Accuracy | The foxbot will stay on the user decicded path and will avoid obstacles including trees, ditches, and moving objects. | Place the foxbot in a location and it should be able to determine its path while staying in the programmed range. It will be able to reroute if any | UNTESTED | Miguel |
| 3.2.1.6 | Morse Decoding Accur | The user radio will be able to pick up the transmitted signal from the foxbot radio and decode it using a morse decoding app. | App on phone will be able to decode the sent signal corresponding to a voltage reading or some other message. | UNTESTED | Sophia |
| 3.2.2.1 | Mass | The foxbot will not exceed 7lbs | Weigh the system once all of the componets are mounted. | UNTESTED | Full Team |
| 3.2.2.2 | Mounting | The sensors shall be able to be mounted to the corners of the chassis and the radio will be | Attatch the sensors to the corners of the chassis and test for accuracy. The | UNTESTED | Full Team |
| 3.2.2.3 | System Packaging | The radio, PCBs, and MCU will be held in custom protective cases | Cases should hold the components and protect them from enviornmental factors like heat, humidity, and water. | UNTESTED | Full Team |
| 3.2.3.1 | Input Voltage (Radio) | The Baofeng will receive an input voltage fo 7.4 V at a current of 1780mA | Use a multimeter to validate input voltage levels. | TESTED | Brady |
| 3.2.3.2 | Input Voltage (ESP) | The ESP will receive an input voltage of 3.3V at a current of 160mA | Use a multimeter to validate input voltage levels. | TESTED | Brady |
| 3.2.3.3 | Voltage Monitoring | The voltage of the batteries will be read by a GPIO pin of the ESP and will detatch the battery from the system if the voltage becomes too low. | used to show the battery level as a percentage of a fully charged rating. A trasistor will be activated to break the circuit when the battery levels drop below a usable voltage. | UNTESTED | Brady and Miguel |
| 3.2.4.1 | Enviornmental Resistar | The foxbot shall be able to traverse flat terrain and withstand temperatues in the range of 5°C to 40°C | The system will be placed in various enviornments ensuring the monitors can traverse the terrain. | UNTESTED | Full Team |
| | Full System Demo | The foxbot will self automate a loaded path from the pixhawk while communicating with the user through DTMF signals and morse. It will hide from the houndbot and will be able to run for 1 hour while avoiding obstacles | Foxbot is placed at a starting location, follows automated path with sending and receiving signals, and avoid getting caught by the houndbot. | UNTESTED | Full Team |

A&M | Dwight Look College of ENGINEERING TEXAS A&M UNIVERSITY

# Gantt Chart