UNIVERSITY OF COLORADO - BOULDER

ECEN 2270

ELECTRONICS LAB | SPRING 2024

ECEN 2270 Electronics Lab: Lab 5

Team Papa:
Gabriel AGOSTINE
Sam WALKER
Julian WERDER
Jonah YUNES

Lab Instructor:
Steven DUNBAR

Lab: Section 12

Sunday, May 5, 2024

College of Engineering & Applied Science
UNIVERSITY OF COLORADO **BOULDER**

I. Introduction

For Lab 5, our objective is to create a remote control system to independently manage all our cars. The remote is designed with a joystick to regulate the car's direction and speed. The Arduino then processes This joystick's input, which transmits the data via an RF antenna to a receiving antenna on the robot. Our remote includes a battery pack for power and buttons to select which robot to transmit to, each button corresponding to a specific frequency for antenna transmission. The remote is assembled on a breadboard, including the Arduino Uno, a joystick, three buttons, an RF antenna, and a battery pack. Additionally, we are adapting this remote for our previous robot, albeit with a modified controller. Instead of a single joystick controlling both sides of the robot, we incorporate two joysticks, each responsible for controlling one side independently.

II. Materials

Purchased Materials

Item	Quantity	Price ($)
NRF24L01+PA	6	4.00
Breakout Adapter	3	2.00

Previously Owned Materials

Item	Quantity
Arduino Nano Every	1
Arduino Uno	1
Joystick Module	3
Buttons	3

III. Theory and Application

We approached this project by testing each of the individual components and elements by themselves and then compiling them all together.

A. Joystick Control Implementation

1. Simple Approach

We first tackled building the joystick control by testing the simple capabilities of the joystick and using simple left vs right, up vs down control to determine which function from lab 4 we use to control the motor directions.

```
// Pin assignments
const int pinON = 6;
const int pinLeftForward = 11;
const int pinLeftBackward = 12;
const int pinLeftPWM = 10;
const int pinRightForward = 7;
const int pinRightBackward = 8;
const int pinRightPWM = 9;

const int pinRightEncoder = 2;
const int pinLeftEncoder = 4;

const int joystickX = A0; // X-axis of joystick
const int joystickY = A1; // Y-axis of joystick

const int deadzone = 25; // Deadzone threshold
volatile int PWM = 0;

void setup() {
  Serial.begin(9600);
  pinMode(pinON, INPUT_PULLUP);

  pinMode(pinLeftForward, OUTPUT);
  pinMode(pinLeftBackward, OUTPUT);
  pinMode(pinLeftPWM, OUTPUT);
  pinMode(pinRightForward, OUTPUT);
  pinMode(pinRightBackward, OUTPUT);
  pinMode(pinRightPWM, OUTPUT);
}

void loop() {
    int yValue = analogRead(joystickX) - 512; // Center the joystick
    int xValue = analogRead(joystickY) - 512; // Center the joystick

    // Apply deadzone
    xValue = (abs(xValue) < deadzone) ? 0 : xValue;
    yValue = (abs(yValue) < deadzone) ? 0 : yValue;

    // Map joystick values to motor speeds
    int normalizedX = map(xValue, -512, 512, -255, 255);
    int normalizedY = map(yValue, -512, 512, -255, 255);

    Serial.print(normalizedX);
    Serial.print(",");
    Serial.println(normalizedY);

    if(normalizedX == 0 && normalizedY == 0)
      stopMotors();
    else{
      if(normalizedX<0)
        turnLeft();
      if(normalizedX>0)
        turnRight();
      if(normalizedY<0)
        goBackward();
      if(normalizedY>0)
        goForward();
      PWM = (abs(normalizedY)+abs(normalizedX))/2;
    }
    analogWrite(pinLeftPWM, PWM);
    analogWrite(pinRightPWM, PWM);
}

void goForward() {
  digitalWrite(pinLeftBackward, LOW);
  digitalWrite(pinRightBackward, LOW);
  digitalWrite(pinRightForward, HIGH);
  digitalWrite(pinLeftForward, HIGH);
}

void goBackward() {
  digitalWrite(pinLeftBackward, HIGH);
  digitalWrite(pinRightBackward, HIGH);
  digitalWrite(pinRightForward, LOW);
  digitalWrite(pinLeftForward, LOW);
}

void turnRight() {
  digitalWrite(pinRightForward, LOW);
  digitalWrite(pinLeftBackward, LOW);
  digitalWrite(pinLeftForward, HIGH);
  digitalWrite(pinRightBackward, HIGH);
}

void turnLeft() {
  digitalWrite(pinRightForward, HIGH);
  digitalWrite(pinLeftBackward, HIGH);
  digitalWrite(pinLeftForward, LOW);
  digitalWrite(pinRightBackward, LOW);
}

void stopMotors() {
  digitalWrite(pinLeftForward, LOW);
  digitalWrite(pinRightForward, LOW);
  digitalWrite(pinLeftBackward, LOW);
  digitalWrite(pinRightBackward, LOW);
  analogWrite(pinLeftPWM, 0); // Set PWM to 0
  analogWrite(pinRightPWM, 0); // Set PWM to 0
}
```

This was a simple yet effective approach, but we wanted effective modular turning radius and more responsive motor control. To do this it is necessary to analyze the joystick position based on angle and magnitude and compare it to quadrant values

2. Complex Approach

- We calculate the magnitude of the joystick's position vector, which is the hypotenuse of a right-angled triangle formed by the X and Y values as the other two sides. This magnitude indicates how far from the center the joystick is moved, which we can translate to the speed of the motors.
- We then calculate the angle of the joystick's position vector, which tells us the direction in which the joystick is pointed. This angle helps us determine how to control the motors for steering.

To implement more nuanced control, we divide the joystick's plane into quadrants and shifted quadrants. The traditional quadrants are simply the four quarters of the Cartesian plane, but we introduce shifted quadrants that divide the plane into eight sections. This allows for a more precise mapping of joystick positions to motor control commands.

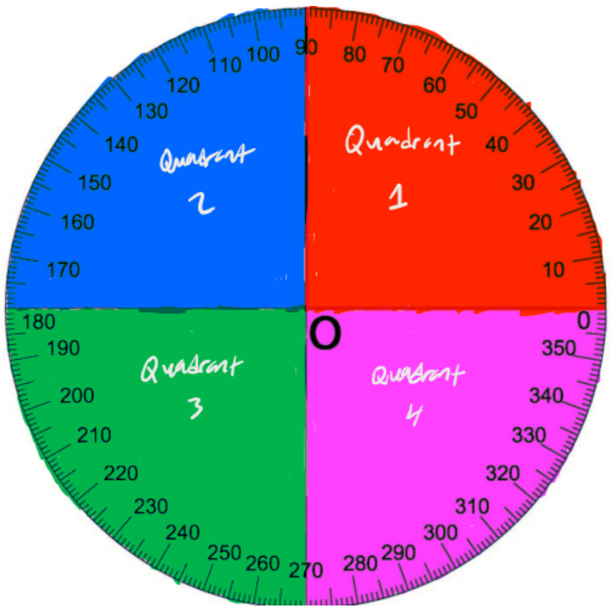


Fig. 1 Traditional Quadrants

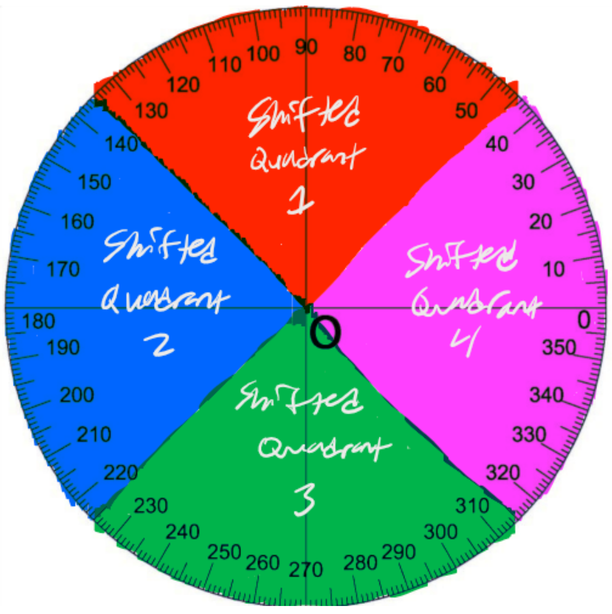


Fig. 2 Shifted Quadrants

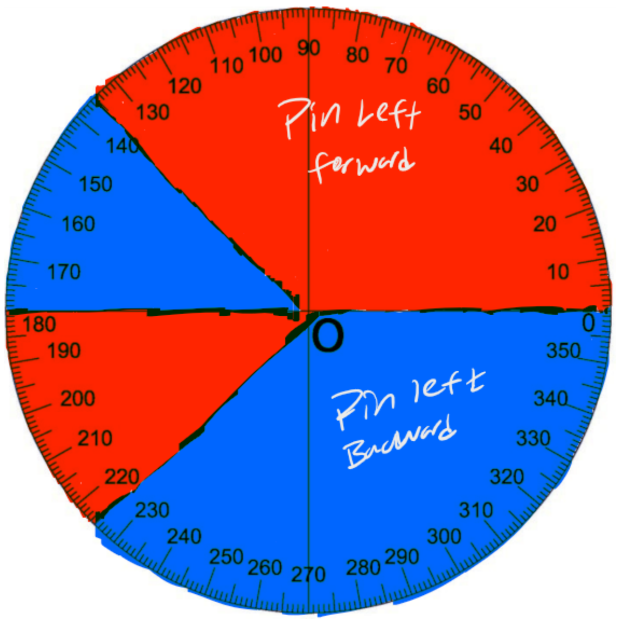


Fig. 3 Left Motor Direction Control

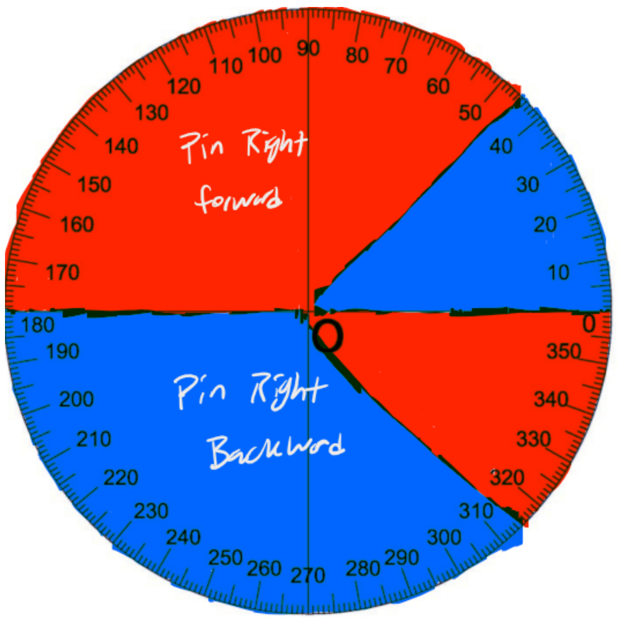


Fig. 4 Right Motor Direction Control

- Forward/Backward Movement: When the joystick is pushed directly up or down, both motors are driven at the same speed but in different directions corresponding to forward or backward motion.
- Turning: As the joystick is moved away from the direct up or down positions towards the left or right, our control system adjusts the speed of the motors differently. The closer the joystick is to the left or right edge, the larger the difference in speed between the two motors, which causes the robot to turn. The robot has the full range from turning in place to going fully straight.

The speed of each motor is controlled by PWM signals. The magnitude of the joystick vector is used to scale the PWM signal. The closer the joystick is to the edge, the higher the PWM signal, and consequently, the faster the motor speed. The angle determines the difference in PWM signals sent to the left and right motors, thus controlling the turn radius.

3. The Code

1) We calculate the joystick's position vector magnitude and angle.
2) We use these to set flags for quadrants and shifted quadrants.
3) Based on the octet the joystick is in, we determine the direction of each motor (forward or backward).
4) We adjust the PWM signals for each motor based on how close the joystick is to the horizontal or vertical axes, which corresponds to how sharply we want the robot to turn.

```
#include <math.h>

// Pin assignments
const int pinLeftForward = 15;
const int pinLeftBackward = 14;
const int pinLeftPWM = 10;
const int pinRightForward = 11;
const int pinRightBackward = 12;
const int pinRightPWM = 9;

const int joystickX = A6; // X-axis of joystick
const int joystickY = A7; // Y-axis of joystick

const int deadzone = 20; // Deadzone threshold
volatile int leftPWM = 0;
volatile int rightPWM = 0;

void setup() {
  Serial.begin(9600);

  pinMode(pinLeftForward, OUTPUT);
  pinMode(pinLeftBackward, OUTPUT);
  pinMode(pinLeftPWM, OUTPUT);
  pinMode(pinRightForward, OUTPUT);
  pinMode(pinRightBackward, OUTPUT);
  pinMode(pinRightPWM, OUTPUT);
}

void loop() {
    int yValue = analogRead(joystickX) - 512; // Center the joystick
    int xValue = analogRead(joystickY) - 512; // Center the joystick

    // Apply deadzone
    xValue = (abs(xValue) < deadzone) ? 0 : xValue;
    yValue = (abs(yValue) < deadzone) ? 0 : yValue;

    // Map joystick values to motor speeds
    int normalizedX = map(xValue, -512, 512, -255, 255);
    int normalizedY = map(yValue, -512, 512, -255, 255);

    int magnitude = constrain(sqrt(pow(normalizedX,2) + pow(normalizedY,2)),0,255);
    int angle = atan2(normalizedY, normalizedX) * 180 / PI;

    if(magnitude == 0)
      stopMotors();
    else{
      leftPWM = 255;
      rightPWM = 255;

      volatile bool quadrant1 = angle<=90&&angle>=0;
      volatile bool quadrant2 = angle<=180&&angle>90;
      volatile bool quadrant3 = angle<=-90&&angle>-180;
      volatile bool quadrant4 = angle<0&&angle>-90;

      volatile bool shiftedQuadrant1 = angle<135&&angle>45;
      volatile bool shiftedQuadrant2 = abs(angle)>135;
      volatile bool shiftedQuadrant3 = angle<-45&&angle>-135;
      volatile bool shiftedQuadrant4 = abs(angle)<45;

      // Set motor directions based on the flags
      digitalWrite(pinLeftForward, quadrant1 || shiftedQuadrant1 || (quadrant3&&shiftedQuadrant2) ? HIGH : LOW);
      digitalWrite(pinRightForward, quadrant2 || shiftedQuadrant1 || (quadrant4&&shiftedQuadrant4) ? HIGH :
           LOW);
      digitalWrite(pinLeftBackward, quadrant4 || shiftedQuadrant3 || (quadrant2&&shiftedQuadrant2) ? HIGH :
           LOW);
      digitalWrite(pinRightBackward, quadrant3 || shiftedQuadrant3 || (quadrant1&&shiftedQuadrant4) ? HIGH :
           LOW);

      //PWM
      if(quadrant1||quadrant4){
        leftPWM = magnitude;
        rightPWM = abs(map(abs(angle),0,90,-255,255));
      }
      else{
        rightPWM = magnitude;
        leftPWM = abs(map(abs(angle),90,180,-255,255));
      }
    }

    //Print the results to the Serial Monitor
    Serial.println(angle);
//    Serial.print(",");
//    Serial.println(rightPWM);

    analogWrite(pinLeftPWM, leftPWM);
    analogWrite(pinRightPWM, rightPWM);
```

```
}

void stopMotors() {
  digitalWrite(pinLeftForward, LOW);
  digitalWrite(pinRightForward, LOW);
  digitalWrite(pinLeftBackward, LOW);
  digitalWrite(pinRightBackward, LOW);
  leftPWM = 0;
  rightPWM = 0;
}
```

Building upon the sophisticated framework of our joystick vector interpretation and control system, we have engineered our motor direction control to fully utilize the potential of the joystick's range of motion, allowing for unparalleled precision in managing the robot's turning radius. This feature is critical because it translates the subtle movements of the joystick directly into dynamic steering responses. Our system doesn't merely switch between forward and backward movements or left and right turns. Instead, it interprets the exact angle and magnitude of the joystick's position to modulate the speed and direction of each motor with high fidelity. This modulation enables the robot to execute turns with a wide variety of radii—from tight, on-the-spot pivots to gentle, wide arcs. Such versatility is crucial for navigating complex environments where precise movements can make the difference between successful maneuvering and collisions.

B. Horn Implemention

1. Overview

The provided Arduino script is designed to control a horn based on the input from a joystick. The primary components involved are a joystick input and a horn output. Here's how the code functions:

2. Setup Configuration

- The **joystick input pin** is configured with an internal pull-up resistor. This means it will read HIGH (inactive) by default, unless the joystick is pressed, making it connect to ground and read LOW (active).
- The **horn output pin** is set as an OUTPUT to send signals to a horn.

3. Operational Details

The main operational loop of the code performs the following actions continuously:

1) It checks the state of the joystick input:
   - If the joystick is **pressed** (input reads LOW), the program activates the horn by generating a continuous tone at 2500 Hz on the horn's output pin.
   - If the joystick is **not pressed** (input reads HIGH), the horn is deactivated by stopping any tone generation on the output pin.

4. Functionality

This setup allows the horn to be controlled directly by the joystick's position. The horn will sound only while the joystick is actively pressed, providing precise control over the horn's activation.

5. The Code

```
const int joyIPT = 3;
const int hornPin = 5;
void setup() {
  pinMode(joyIPT, INPUT_PULLUP);
  pinMode(hornPin, OUTPUT);
  Serial.begin(9600);
}
void loop (){
  if(!digitalRead(joyIPT)){
    tone(hornPin, 2500);
  }else if(digitalRead(joyIPT)){
    noTone(hornPin);
  }
}
```

C. RF Implementation

1. Introduction

This section describes the necessary steps to optimize radio frequency communication for the nRF24L01+ module used in Arduino projects. It highlights the importance of selecting the appropriate communication channel to reduce signal loss and improve transmission reliability.

2. The Code

Transmitter Code

```
#include "SPI.h"
#include "RF24.h"
#include "nRF24L01.h"
#define CE_PIN 10
#define CSN_PIN A1
#define INTERVAL_MS_TRANSMISSION 250
RF24 radio(CE_PIN, CSN_PIN);
const byte address[6] = "00001";
//NRF24L01 buffer limit is 32 bytes (max struct size)
struct payload {
   byte data1;
   char data2;
};
payload payload;
void setup()
{
   Serial.begin(115200);
   radio.begin();
   //Append ACK packet from the receiving radio back to the transmitting radio
   radio.setAutoAck(false); //(true|false)
   //Set the transmission datarate
   radio.setDataRate(RF24_250KBPS); //(RF24_250KBPS|RF24_1MBPS|RF24_2MBPS)
   //Greater level = more consumption = longer distance
   radio.setPALevel(RF24_PA_MAX); //(RF24_PA_MIN|RF24_PA_LOW|RF24_PA_HIGH|RF24_PA_MAX)
   //Default value is the maximum 32 bytes
   radio.setPayloadSize(sizeof(payload));
   //Act as transmitter
   radio.openWritingPipe(address);
   radio.stopListening();
}
void loop()
{
   payload.data1 = 123;
   payload.data2 = 'x';
   radio.write(&payload, sizeof(payload));
   Serial.print("Data1:");
   Serial.println(payload.data1);
   Serial.print("Data2:");
   Serial.println(payload.data2);
   Serial.println("Sent");
   delay(INTERVAL_MS_TRANSMISSION);
}
```

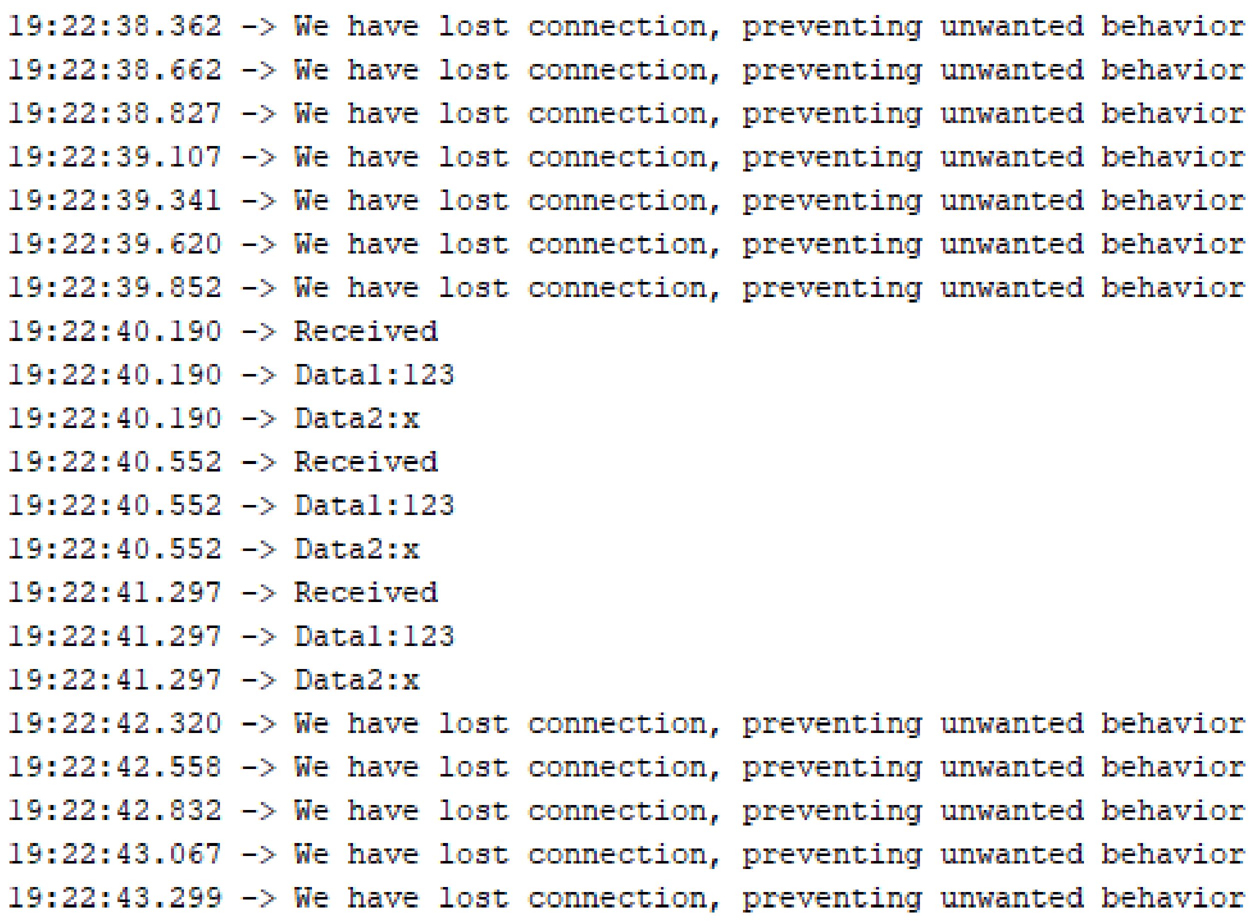
Receiver Code

```
    #include "SPI.h"
#include "RF24.h"
```

```
#include "nRF24L01.h"
#define CE_PIN 10
#define CSN_PIN A1
#define INTERVAL_MS_SIGNAL_LOST 1000
#define INTERVAL_MS_SIGNAL_RETRY 250
RF24 radio(CE_PIN, CSN_PIN);
const byte address[6] = "00001";
//NRF24L01 buffer limit is 32 bytes (max struct size)
struct payload {
   byte data1;
   char data2;
};
payload payload;
unsigned long lastSignalMillis = 0;
void setup()
{
   Serial.begin(115200);
   radio.begin();
   //Append ACK packet from the receiving radio back to the transmitting radio
   radio.setAutoAck(false); //(true|false)
   //Set the transmission datarate
   radio.setDataRate(RF24_250KBPS); //(RF24_250KBPS|RF24_1MBPS|RF24_2MBPS)
   //Greater level = more consumption = longer distance
   radio.setPALevel(RF24_PA_MAX); //(RF24_PA_MIN|RF24_PA_LOW|RF24_PA_HIGH|RF24_PA_MAX)
   //Default value is the maximum 32 bytes1
   radio.setPayloadSize(sizeof(payload));
   //Act as receiver
   radio.openReadingPipe(0, address);
   radio.startListening();
}
void loop()
{
   unsigned long currentMillis = millis();
   if (radio.available() > 0) {
     radio.read(&payload, sizeof(payload));
     Serial.println("Received");
     Serial.print("Data1:");
     Serial.println(payload.data1);
     Serial.print("Data2:");
     Serial.println(payload.data2);
     lastSignalMillis = currentMillis;
   }
   if (currentMillis - lastSignalMillis > INTERVAL_MS_SIGNAL_LOST) {
     lostConnection();
   }
}
void lostConnection()
{
   Serial.println("We have lost connection, preventing unwanted behavior");
   delay(INTERVAL_MS_SIGNAL_RETRY);
}
```

3. *Problem Identification*

During initial testing phases of the code above, it was observed, as shown below, that the communication between the transmitter and receiver was frequently interrupted, as indicated by repeated logs of lost connections. These interruptions can lead to degraded performance and unintended behaviors in critical applications.



```
19:22:38.362 -> We have lost connection, preventing unwanted behavior
19:22:38.662 -> We have lost connection, preventing unwanted behavior
19:22:38.827 -> We have lost connection, preventing unwanted behavior
19:22:39.107 -> We have lost connection, preventing unwanted behavior
19:22:39.341 -> We have lost connection, preventing unwanted behavior
19:22:39.620 -> We have lost connection, preventing unwanted behavior
19:22:39.852 -> We have lost connection, preventing unwanted behavior
19:22:40.190 -> Received
19:22:40.190 -> Data1:123
19:22:40.190 -> Data2:x
19:22:40.552 -> Received
19:22:40.552 -> Data1:123
19:22:40.552 -> Data2:x
19:22:41.297 -> Received
19:22:41.297 -> Data1:123
19:22:41.297 -> Data2:x
19:22:42.320 -> We have lost connection, preventing unwanted behavior
19:22:42.558 -> We have lost connection, preventing unwanted behavior
19:22:42.832 -> We have lost connection, preventing unwanted behavior
19:22:43.067 -> We have lost connection, preventing unwanted behavior
19:22:43.299 -> We have lost connection, preventing unwanted behavior
```

Fig. 5 Dropping Signals

4. Channel Interference

The nRF24L01+ operates in the congested 2.4 GHz band, which is shared by various devices including Wi-Fi routers, Bluetooth devices, and other RF equipment. This can cause significant interference, resulting in the loss of signal between the communicating modules.

5. Channel Optimization Strategy

To mitigate interference, the RF channel can be manually set to a less congested frequency within the range. The nRF24L01+ supports 125 channels, allowing for flexibility in selecting the optimal channel based on environmental conditions.

```
#define CHANNEL 76 // Example of setting a specific channel

void setup() {
  Serial.begin(115200);
  radio.begin();
  radio.setAutoAck(false);
  radio.setDataRate(RF24_250KBPS);
  radio.setPALevel(RF24_PA_MAX);
  radio.setPayloadSize(sizeof(payload));
  radio.setChannel(CHANNEL); // Set the radio to use a specific channel
}
```

6. Benefits

Adjusting the channel minimizes the likelihood of interference from other devices, thereby enhancing the stability and reliability of the RF communication. This adjustment is crucial for applications where consistent data transmission is essential.

7. Conclusion

Identifying and setting an optimal channel for the nRF24L01+ module is a critical step in ensuring robust and reliable communication in environments with potential RF interference. This process involves both testing different channels and monitoring the system's performance to select the most suitable frequency. Our testing revealed that at home the optimal channel was 15, whereas at lab the optimal channel was 115. There is more information regarding the channel testing in the Challenges and Lessons Learned section.

IV. Sam, Jonah, and Julian's Remote

A. Introduction

This section outlines the implementation of Sam, Jonah and Julian's transmitter and receiver scripts designed for joystick-controlled robots via RF communication. The system includes additional functionalities such as robot-specific control and debugging features.

B. System Overview

The system consists of two main scripts:

- **Transmitter Script:** Handles the reading of joystick inputs, button states, and transmits these commands after some computation to the receiver via RF.
- **Receiver Script:** Receives the transmitted commands and controls the robot's motors and other functionalities accordingly.

C. Transmitter Script

The transmitter script uses a joystick to control motor speeds and directions, and three buttons to select which robot is currently active. It includes debouncing functionality to prevent rapid toggling of robot control.

1. Key Features

- Joystick control for dynamic speed and direction adjustments.
- Button inputs to switch control between robots, allowing multiple robots to be controlled separately or simultaneously.

2. Transmitter Code

```
#include "SPI.h"
#include "RF24.h"
#include "nRF24L01.h"

//RF INFO
#define CE_PIN 10
#define CSN_PIN A1
#define INTERVAL_MS_TRANSMISSION 1
RF24 radio(CE_PIN, CSN_PIN);
const byte address[6] = "11111";
struct payload {
  int leftPWM = 0;
  int rightPWM = 0;
  bool leftForward = false;
  bool rightForward = false;
```

```
    bool leftBackward = false;
    bool rightBackward = false;
    bool hornToggle = false;
    bool robotSam = false;
    bool robotJulian = false;
    bool robotJonah = false;
    bool stopDaMotors = true;
};
payload payload;

//JOYSTICK INFO
#define JOYSTICK_X_PIN A6
#define JOYSTICK_Y_PIN A7
#define JOYSTICK_BUTTON 2
const int deadzone = 20; // Deadzone threshold

//BUTTON INFO
#define SAM_BUTTON_PIN 3
#define JULIAN_BUTTON_PIN 4
#define JONAH_BUTTON_PIN 5
unsigned long buttonDelayMillis = 500;
unsigned long lastSamButtonMillis = 0;
unsigned long lastJulianButtonMillis = 0;
unsigned long lastJonahButtonMillis = 0;

void setup()
{
  Serial.begin(9600);

  //RF
  radio.begin();
  radio.setAutoAck(false);
  radio.setDataRate(RF24_250KBPS);
  radio.setPALevel(RF24_PA_MAX);
  radio.setPayloadSize(sizeof(payload));
  radio.setChannel(115);
  radio.openWritingPipe(address);
  radio.stopListening();

  //BUTTON CONTROL
  pinMode(SAM_BUTTON_PIN, INPUT_PULLUP);
  pinMode(JULIAN_BUTTON_PIN, INPUT_PULLUP);
  pinMode(JONAH_BUTTON_PIN, INPUT_PULLUP);
  pinMode(JOYSTICK_BUTTON, INPUT);
}

void loop()
{
  unsigned long currentMillis = millis();

  int yValue = analogRead(JOYSTICK_X_PIN) - 512; // Center the joystick
  int xValue = analogRead(JOYSTICK_Y_PIN) - 512; // Center the joystick

  // Apply deadzone
  xValue = (abs(xValue) < deadzone) ? 0 : xValue;
  yValue = (abs(yValue) < deadzone) ? 0 : yValue;

  // Map joystick values to motor speeds
  int normalizedX = map(xValue, -512, 512, -255, 255);
  int normalizedY = map(yValue, -512, 512, -255, 255); //Flip

  int magnitude = constrain(sqrt(pow(normalizedX,2) + pow(normalizedY,2)),0,255);
  int angle = atan2(normalizedY, normalizedX) * 180 / PI;
  if(digitalRead(JOYSTICK_BUTTON))
    payload.hornToggle = true;
  else
    payload.hornToggle = false;
  if(digitalRead(SAM_BUTTON_PIN) == LOW && currentMillis - lastSamButtonMillis > buttonDelayMillis){
    payload.robotSam = !payload.robotSam;
    lastSamButtonMillis = currentMillis;
  }
  if(digitalRead(JULIAN_BUTTON_PIN) == LOW && currentMillis - lastJulianButtonMillis > buttonDelayMillis){
    payload.robotJulian = !payload.robotJulian;
    lastJulianButtonMillis = currentMillis;
  }
  if(digitalRead(JONAH_BUTTON_PIN) == LOW && currentMillis - lastJonahButtonMillis > buttonDelayMillis){
    payload.robotJonah = !payload.robotJonah;
    lastJonahButtonMillis = currentMillis;
  }
  if(magnitude == 0)
      payload.stopDaMotors = true;
  else{
      payload.stopDaMotors = false;
      payload.leftPWM = 255;
      payload.rightPWM = 255;

      volatile bool quadrant1 = angle<=90&&angle>=0;
      volatile bool quadrant2 = angle<=180&&angle>90;
      volatile bool quadrant3 = angle<=-90&&angle>-180;
      volatile bool quadrant4 = angle<0&&angle>-90;

      volatile bool shiftedQuadrant1 = angle<135&&angle>45;
      volatile bool shiftedQuadrant2 = abs(angle)>135;
      volatile bool shiftedQuadrant3 = angle<-45&&angle>-135;
      volatile bool shiftedQuadrant4 = abs(angle)<45;

      // Set motor directions based on the flags
      payload.leftForward = quadrant3 || shiftedQuadrant3 || (quadrant1&&shiftedQuadrant4) ? true : false;
      payload.rightForward = quadrant4 || shiftedQuadrant3 || (quadrant2&&shiftedQuadrant2) ? true : false;
      payload.leftBackward = quadrant2 || shiftedQuadrant1 || (quadrant4&&shiftedQuadrant4) ? true : false;
      payload.rightBackward = quadrant1 || shiftedQuadrant1 || (quadrant3&&shiftedQuadrant2) ? true : false;

      //PWM
      if(quadrant1||quadrant4){
        payload.leftPWM = magnitude;
        payload.rightPWM = abs(map(abs(angle),0,90,-255,255));
      }
      else{
        payload.rightPWM = magnitude;
        payload.leftPWM = abs(map(abs(angle),90,180,-255,255));
      }
    }

  radio.write(&payload, sizeof(payload));
  delay(INTERVAL_MS_TRANSMISSION);
}
```

D. Receiver Script

The receiver script implements the commands received from the transmitter. It controls motor directions and speeds based on joystick input, and handles specific robot control logic based on the active robot selected by the transmitter.

1. Robot Specific Control

Each robot's behavior can be customized by altering the condition within the if statement that checks which robot's control is active. For instance in the code outlined below, changing 'robotJulian' to 'robotSam' or 'robotJonah' in the receiver script tailors the control to the respective robot.

```
if(payload.stopDaMotors||!payload.robotJulian){
  stopMotors();
}
```

2. Debugging and Monitoring

The system includes a ‘printPayload‘ function in the receiver script, which outputs the current state of all control variables to the serial monitor. This function is crucial for debugging and ensures that all inputs and states are monitored in real-time.

Receiver Code

```
#include "SPI.h"
#include "RF24.h"
#include "nRF24L01.h"

//RF INFO
#define CE_PIN 8
#define CSN_PIN A1
#define INTERVAL_MS_SIGNAL_LOST 0
#define INTERVAL_MS_SIGNAL_RETRY 0
RF24 radio(CE_PIN, CSN_PIN);
const byte address[6] = "11111";
struct payload {
  int leftPWM = 0;
  int rightPWM = 0;
  bool leftForward = false;
  bool rightForward = false;
  bool leftBackward = false;
  bool rightBackward = false;
  bool hornToggle = false;
  bool robotSam = false;
  bool robotJulian = false;
  bool robotJonah = false;
  bool stopDaMotors = true;
};
payload payload;
unsigned long lastSignalMillis = 0;

//ROBOT INFO
#define LEFT_FORWARD_PIN 17
#define LEFT_BACKWARD_PIN 16
#define LEFT_PWM_PIN 10
#define RIGHT_FORWARD_PIN 7
#define RIGHT_BACKWARD_PIN 6
#define RIGHT_PWM_PIN 9
#define HORN_OUTPUT 3

void setup()
{
  Serial.begin(9600);

  //RF SETUP
  radio.begin();
  radio.setAutoAck(false);
  radio.setDataRate(RF24_250KBPS);
  radio.setPALevel(RF24_PA_MAX);
  radio.setPayloadSize(sizeof(payload));
  radio.setChannel(115);
  radio.openReadingPipe(0, address);
  radio.startListening();

  //MOTOR SETUP
  pinMode(LEFT_FORWARD_PIN, OUTPUT);
  pinMode(LEFT_BACKWARD_PIN, OUTPUT);
  pinMode(LEFT_PWM_PIN, OUTPUT);
  pinMode(RIGHT_FORWARD_PIN, OUTPUT);
  pinMode(RIGHT_BACKWARD_PIN, OUTPUT);
  pinMode(RIGHT_PWM_PIN, OUTPUT);
  pinMode(HORN_OUTPUT, OUTPUT);
}
void loop()
```

```
{
  unsigned long currentMillis = millis();
  if (radio.available() > 0) {
    radio.read(&payload, sizeof(payload));
    if(payload.stopDaMotors||!payload.robotJulian){
      stopMotors();
    }
    else{
      digitalWrite(LEFT_FORWARD_PIN, payload.leftForward ? HIGH : LOW);
      digitalWrite(RIGHT_FORWARD_PIN, payload.rightForward ? HIGH : LOW);
      digitalWrite(LEFT_BACKWARD_PIN, payload.leftBackward ? HIGH : LOW);
      digitalWrite(RIGHT_BACKWARD_PIN, payload.rightBackward ? HIGH : LOW);
      analogWrite(LEFT_PWM_PIN, payload.leftPWM);
      analogWrite(RIGHT_PWM_PIN, payload.rightPWM);

      if(payload.hornToggle)
       tone(HORN_OUTPUT, 2500);
      else
       noTone(HORN_OUTPUT);
    }
    printPayload();
    lastSignalMillis = currentMillis;
  }
  else if (currentMillis - lastSignalMillis > INTERVAL_MS_SIGNAL_LOST) {
    lostConnection();
  }
}

void stopMotors() {
  digitalWrite(LEFT_FORWARD_PIN, LOW);
  digitalWrite(RIGHT_FORWARD_PIN, LOW);
  digitalWrite(LEFT_BACKWARD_PIN, LOW);
  digitalWrite(RIGHT_BACKWARD_PIN, LOW);
  payload.leftPWM = 0;
  payload.rightPWM = 0;
}

void printPayload()
{
   Serial.println("Received Payload");
   Serial.print("Left PWM:");
   Serial.println(payload.leftPWM);
   Serial.print("Right PWM:");
   Serial.println(payload.rightPWM);
   Serial.print("Left Forward:");
   Serial.println(payload.leftForward);
   Serial.print("Right Forward:");
   Serial.println(payload.rightForward);
   Serial.print("Left Backward:");
   Serial.println(payload.leftBackward);
   Serial.print("Right Backward:");
   Serial.println(payload.rightBackward);
   Serial.print("Sam Button:");
   Serial.println(payload.robotSam);
   Serial.print("Julian Button:");
   Serial.println(payload.robotJulian);
   Serial.print("Jonah Button:");
   Serial.println(payload.robotJonah);
   Serial.print("STOP:");
   Serial.println(payload.stopDaMotors);
   Serial.println("=====================");
}

void lostConnection()
{
   Serial.println("We have lost connection, preventing unwanted behavior");
   stopMotors();
   delay(INTERVAL_MS_SIGNAL_RETRY);
}
```

E. The Remote

Outlined below is a picture of the remote showcasing the three buttons, joystick and antenna all connected through the arduino nano every.

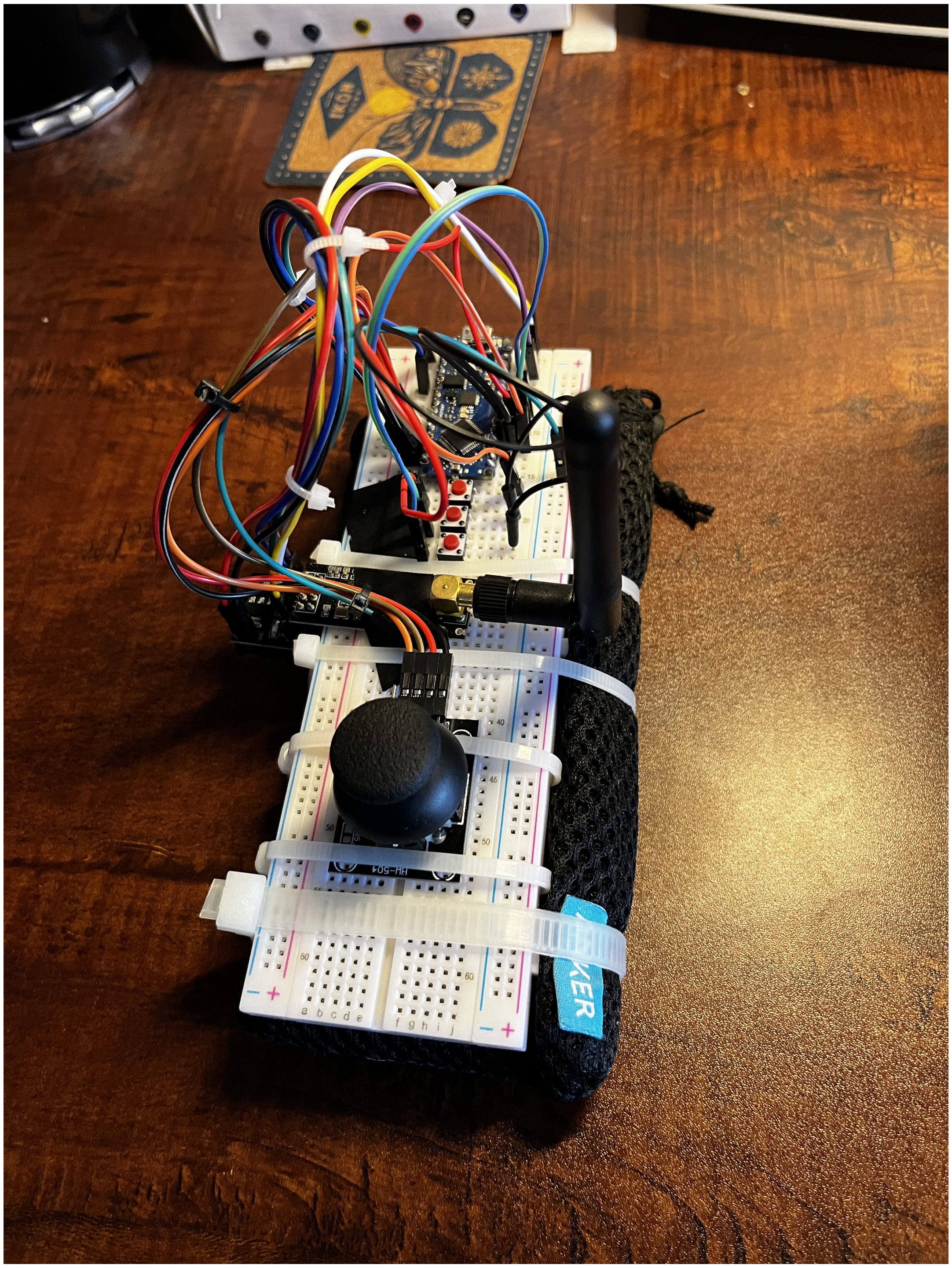


Fig. 6 Remote

F. The Robot

Here is a picture of Sam’s robot showcasing the RF antenna mounted. Both Jonah’s and Julian’s robots have the same pinout and structure.

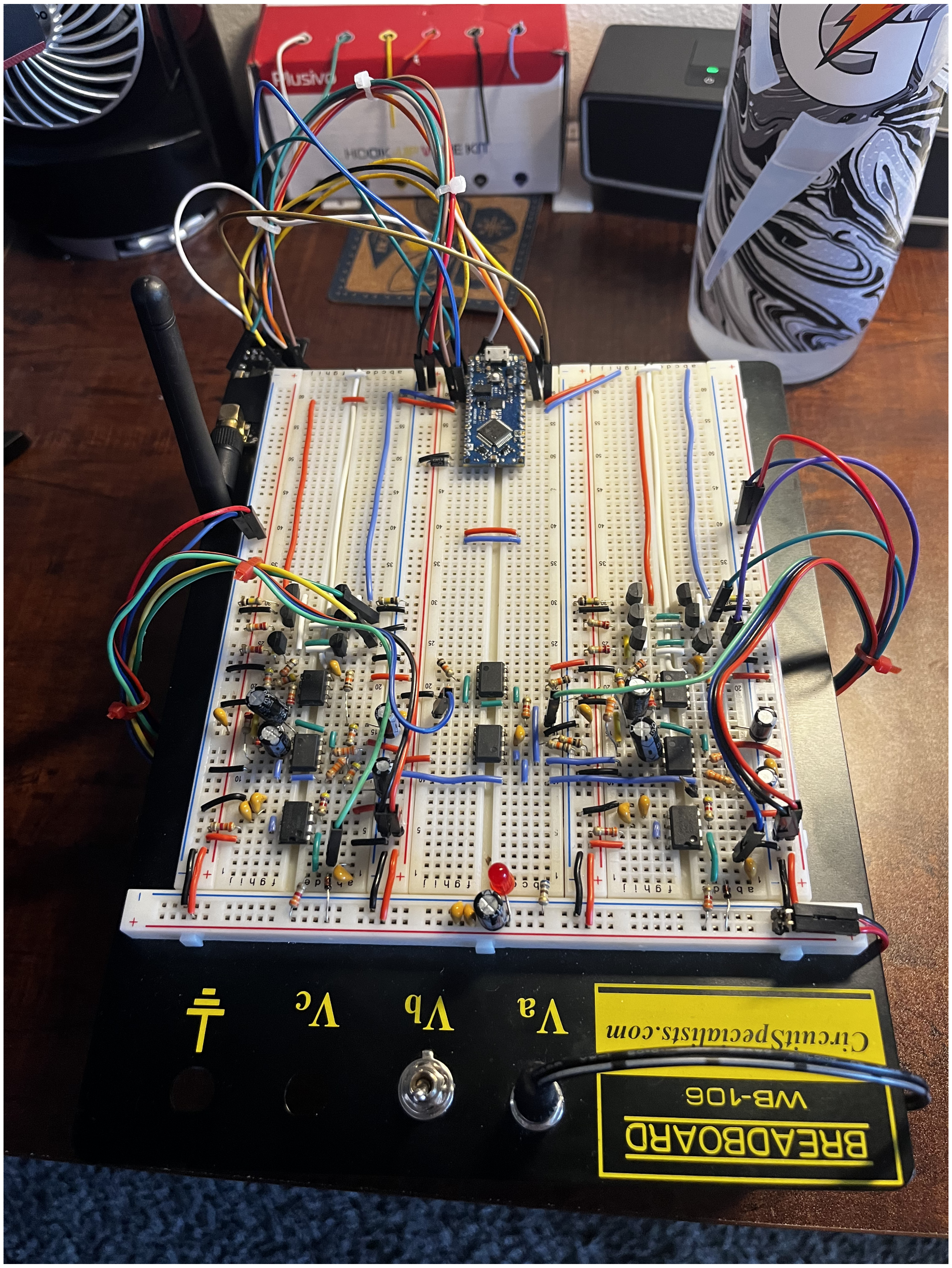


Fig. 7 Robot

V. Tank Drive Configuration

This remote system was based on a typical tank control. The remote has 2 joysticks connected to the Arduino, and only its x-direction voltage is registered by the Arduino. These two values are sent over RF via the transmitter antenna using the Transmitter Code:

```
#include <SPI.h>
#include <nRF24L01.h>
#include <RF24.h>

//create an RF24 object
RF24 radio(9, 8); // CE, CSN

//address through which two modules communicate.
const byte address[6] = "11001";

const int joystickL = A1;
const int joystickR = A0;
int xb = 0;
int yb = 0;
// Variable initialization

void setup() {

  SPI.begin();
  Serial.begin(9600);

  radio.begin();
  radio.setAutoAck(false);
  radio.setPALevel(RF24_PA_MAX);//Transmitter RF Power Setting
  //MIN=-18dBm, LOW=-12dBm, HIGH=-6dBm, MAX=0dBm.
  radio.setChannel(15); //above most WiFi frequencies. RF Channel setting 0-125
  radio.setDataRate( RF24_250KBPS );

  //set the address
  radio.openWritingPipe(address);

  //Set module as transmitter
  radio.stopListening();

}

void loop() {

  xb = analogRead(joystickL);
  yb = analogRead(joystickR);
  // Serial.print(xb);
  // Serial.print(" || ");
  // Serial.println(yb);
  // Centers position in joystick coordinate frame

  //Send message to receiver
  int data[2]; // Declare the array
  data[0] = xb; // Store xb
  data[1] = yb; // Store yb
  radio.write((uint8_t *)&data, sizeof(data));

}
```

Figure 8 shows the transmitting remote for the Tank Drive Configuration. Each joystick is responsible for the forwards and backwards control and movement of their corresponding left and right wheels.

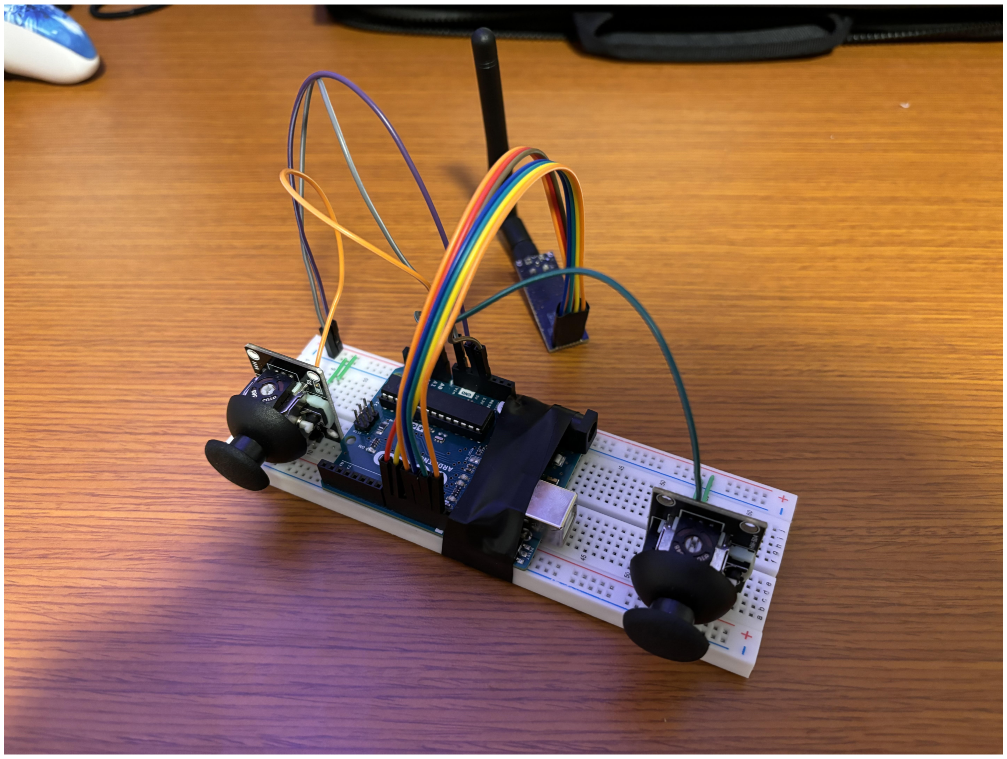


Fig. 8 Gabe's Remote

The receiver antenna meanwhile receives both values sent using the same packet and adjusts the forwards and backwards direction control depending on whether the joystick (and its subsequent value) is angled forwards or backwards. This is also done individually and simultaneously (assuming infinitely small calculation time) for each set of wheels (both right and left side). The receiver code is as follows:

```
//Include Libraries
#include <nRF24L01.h>
#include <RF24.h>
#include <SPI.h>

//create an RF24 object
RF24 radio(10, A1); // CE, CSN

//address through which two modules communicate.
const byte address[6] = "11001";

const int dead = 20;
const int pinLeftForward = 8;
const int pinLeftBackward = 9;
const int pinLeftPWM = 3;
const int pinRightForward = 4;
const int pinRightBackward = 5;
const int pinRightPWM = 6;

int xL = 0;
int xR = 0;

void setup() {

  SPI.begin();
  Serial.begin(9600);

  pinMode(pinLeftForward, OUTPUT);
  pinMode(pinLeftBackward, OUTPUT);
  pinMode(pinLeftPWM, OUTPUT);
  pinMode(pinRightForward, OUTPUT);
  pinMode(pinRightBackward, OUTPUT);
  pinMode(pinRightPWM, OUTPUT);

  digitalWrite(pinLeftForward, LOW);
  digitalWrite(pinRightForward, LOW);
  digitalWrite(pinLeftBackward, LOW);
  digitalWrite(pinRightBackward, LOW);

  analogWrite(pinLeftPWM, 0);
  analogWrite(pinRightPWM, 0);

  radio.begin();
  radio.setAutoAck(false);
  radio.setPALevel(RF24_PA_MAX); //Transmitter RF Power Setting
                                 //MIN=-18dBm, LOW=-12dBm, HIGH=-6dBm, MAX=0dBm.
  radio.setChannel(15); //above most WiFi frequencies. RF Channel setting 0-125
  radio.setDataRate( RF24_250KBPS );

  //set the address
  radio.openReadingPipe(0, address);

  //Set module as receiver
  radio.startListening();

  delay(1000);

}

void loop() {

  if (radio.available()) {

    // Serial.println("Radio established");

    int data[2]; // Declare an array to hold the received data
    radio.read((uint8_t *)&data, sizeof(data)); // Read the binary data into the array

    xL = data[0] - 511;
    xR = data[1] - 511;
    Serial.print(xL);
    Serial.print(" || ");
    Serial.println(xR);

  }

  // xL = (abs(xL) < dead) ? 0 : xL;
  // xR = (abs(xR) < dead) ? 0 : xR;

  if (xL > dead) {
    digitalWrite(pinLeftForward, HIGH);
    digitalWrite(pinLeftBackward, LOW);
    analogWrite(pinLeftPWM, 255);
  }
  else if (xL < -dead) {
    digitalWrite(pinLeftForward, LOW);
    digitalWrite(pinLeftBackward, HIGH);
    analogWrite(pinLeftPWM, 255);
  }
  else{
    digitalWrite(pinLeftForward, LOW);
    digitalWrite(pinLeftBackward, LOW);
    analogWrite(pinLeftPWM, 0);
  }

  if (xR > dead) {
    digitalWrite(pinRightForward, HIGH);
```

```
            digitalWrite(pinRightBackward, LOW);
            analogWrite(pinRightPWM, 255);
          }
          else if (xR < -dead) {
            digitalWrite(pinRightForward, LOW);
            digitalWrite(pinRightBackward, HIGH);
            analogWrite(pinRightPWM, 255);
          }
          else{
            digitalWrite(pinRightForward, LOW);
            digitalWrite(pinRightBackward, LOW);
            analogWrite(pinRightPWM, 0);
          }

        }
```

Unfortunately, this code did not work 100% of the time. On certain days in the same locations the system would work properly while on other days it would fail. The most common error that this method encountered was the receiver antenna receiving 0's despite the transmitting antenna being off. This cannot be attributed to a "lack of a signal" as the receiver antenna is looking for a specific sequence of bits before the payload before loading the data in. Yet somehow the receiver would still receive 0's for an unknown reason.

Despite this, this system did work occasionally. The other common problem was a stuttering for the robot, where it would receive 0's intermittently between correct data, leading to a staggering forwards and backwards jittering with a slight net movement in the desired direction.

We believe that if the issue of receiving 0's randomly could be identified and solved this system would work properly 100% of the time, as it does work occasionally, just not consistently enough to be deemed a success.

VI. Challenges & Lessons Learned

This section reflects on the key challenges and valuable lessons learned throughout our project with the nRF24L01+ module. Each experience has significantly contributed to our understanding of wireless communication technologies and practical problem-solving skills in electronics.

A. Testing NRF24L01 Modules

This section outlines the procedure used to evaluate the performance of antennas connected to the nRF24L01+ radio module via an Arduino platform. The process involves a spectrum analysis across multiple radio channels to assess both antenna functionality and environmental interference.

1. Setup

The Arduino is programmed to interface with the nRF24L01+ module, employing the following key configurations:

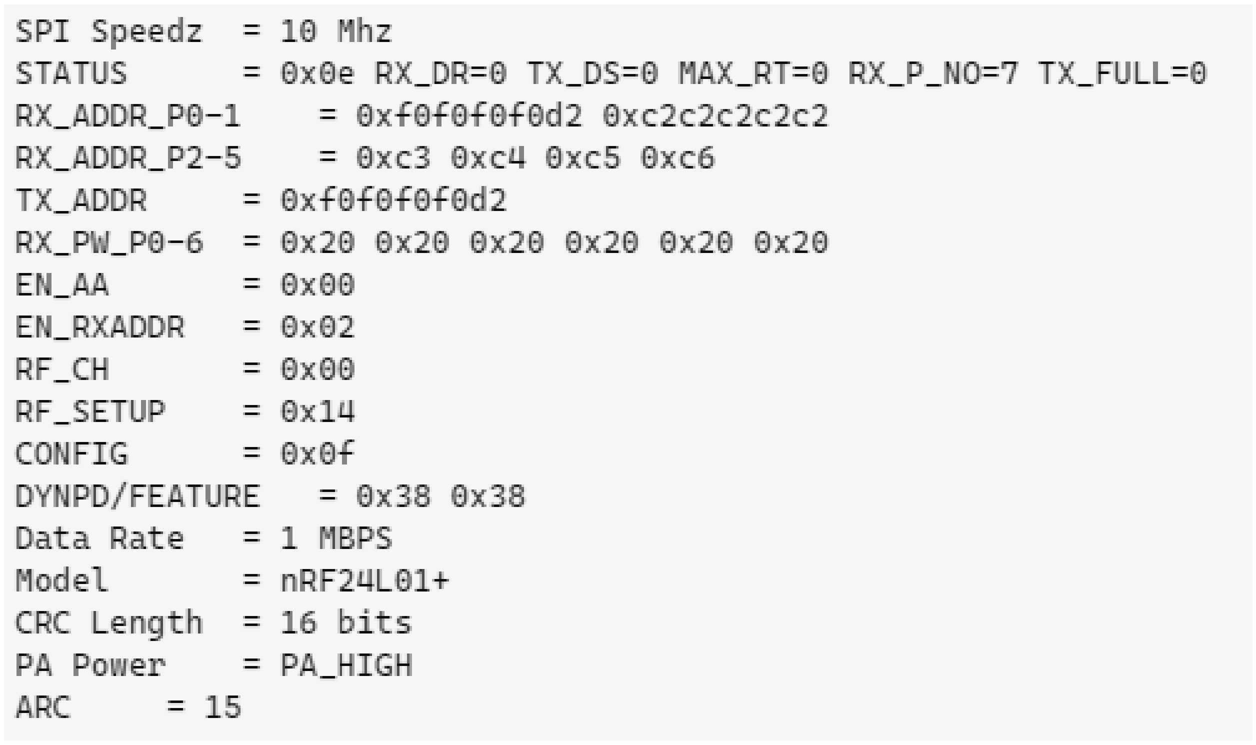
- **Serial Communication:** Initiated at 9600 baud to facilitate data output.
- **Radio Initialization:** The RF24 library functions are utilized to start and configure the radio.
- **Listening Mode:** The radio is set to listen briefly on each channel to detect any carrier signals indicating activity.

2. Testing Process

Channel testing is executed through a scripted loop that performs the following steps for each of the 128 channels:

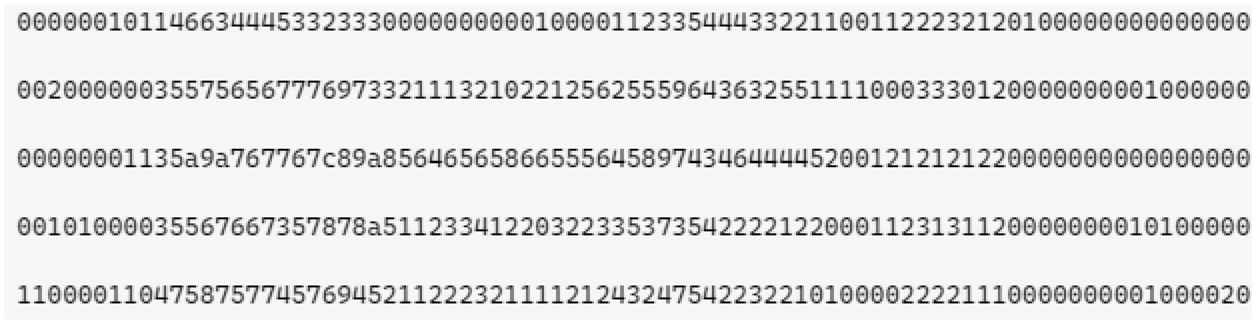
1) Set the radio to the specific channel.
2) Enable listening mode for a short duration (128 microseconds).
3) Check for the presence of carrier signals.
4) Record the results in an array, incrementing the count for channels where a signal was detected.

This process is repeated multiple times (100 repetitions) to ensure accuracy and reliability of the results. Further, this entire process is repeated until the output looks desirable. Here's an example of what desirable looks like:



```
SPI Speedz  = 10 Mhz
STATUS      = 0x0e RX_DR=0 TX_DS=0 MAX_RT=0 RX_P_NO=7 TX_FULL=0
RX_ADDR_P0-1    = 0xf0f0f0f0d2 0xc2c2c2c2c2
RX_ADDR_P2-5    = 0xc3 0xc4 0xc5 0xc6
TX_ADDR     = 0xf0f0f0f0d2
RX_PW_P0-6  = 0x20 0x20 0x20 0x20 0x20 0x20
EN_AA       = 0x00
EN_RXADDR   = 0x02
RF_CH       = 0x00
RF_SETUP    = 0x14
CONFIG      = 0x0f
DYNPD/FEATURE   = 0x38 0x38
Data Rate   = 1 MBPS
Model       = nRF24L01+
CRC Length  = 16 bits
PA Power    = PA_HIGH
ARC     = 15
```

Fig. 9   Desirable Program Setup Output



```
00000010114663444533233300000000000100001123354443322110011222321201000000000000000

0020000003557565677769733211132102212562555964363255111100033301200000000001000000

00000001135a9a767767c89a856465658665555645897434644445200121212122000000000000000000

00101000035567667357878a511233412203223353735422221220001123131120000000010100000

1100001104758757745769452112223211112124324754223221010000222211100000000001000020
```

Fig. 10   Desirable Noise Output

3. Data Output

Post-testing, the collected data are displayed via the serial monitor of the Arduino IDE, showing the detected activity for each channel in hexadecimal format. This output helps in identifying channels with significant environmental noise and verifying the effective range of the antenna.

4. Importance of the Test

The test is crucial for:
- Verifying the correct functionality and optimal placement of antennas.
- Ensuring proper connections and operational integrity of the Arduino and nRF24L01+ module.
- Assessing the environmental conditions that may affect wireless communications.

5. Conclusion

This testing procedure is an essential part of setting up and maintaining robust wireless communication systems using the nRF24L01+ module, ensuring both hardware compatibility and environmental suitability.

6. The Code

```
#include <SPI.h>
#include <nRF24L01.h>
#include <RF24.h>

RF24 radio(10,A1); // for arduino nano every
//RF24 radio(9,10); // for arduino uno

const uint8_t num_channels = 128;
uint8_t values[num_channels];
void setup(void)
{
  Serial.begin(9600);
  printf_begin();
  radio.begin();
  radio.setAutoAck(false);
  radio.startListening();

  radio.printDetails();
  delay(5000);

  radio.stopListening();
  int i = 0;
  while ( i < num_channels ) {
    printf("%x",i>>4);
    ++i;
  }
  printf("\n\r");
  i = 0;
  while ( i < num_channels ) {
    printf("%x",i&0xf);
    ++i;
  }
  printf("\n\r");
}
const int num_reps = 100;

void loop(void)
{
  memset(values,0,sizeof(values));
  int rep_counter = num_reps;
  while (rep_counter--) {
    int i = num_channels;
    while (i--) {
      radio.setChannel(i);
      radio.startListening();
      delayMicroseconds(128);
      radio.stopListening();
      if ( radio.testCarrier() )
        ++values[i];
    }
  }
  int i = 0;
  while ( i < num_channels ) {
    printf("%x",min(0xf,values[i]&0xf));
    ++i;
  }
  printf("\n\r");
}
int serial_putc( char c, FILE * ) {
  Serial.write( c );
  return c;
}

void printf_begin(void) {
  fdevopen( &serial_putc, 0 );
}
```

B. Arduino Uno vs Arduino Nano Every

1. Introduction

This document presents findings from testing the nRF24L01+ module with Arduino platforms, focusing on the necessary pin configurations for effective operation. Special attention is given to the differences in pin configuration requirements between the Arduino Nano Every and the Arduino Uno.

2. Testing Results

The testing program implemented on both Arduino Nano Every and Arduino Uno revealed crucial differences in the Chip Select Not (CSN) pin configuration. It was determined that for the Arduino Nano Every, the CSN pin needs to be connected specifically to either pin A0 or A1. This configuration differs from that typically used in the Arduino Uno, where CSN is often connected to digital pins such as pin 10.

3. Arduino Board Differences

The Arduino Uno is based on the ATmega328P microcontroller, whereas the Arduino Nano Every is equipped with the more powerful ATmega4809. This difference in microcontroller technology influences not only the performance but also the pinout and functionality across the boards.

4. Importance of Testing Program

The testing program was crucial in identifying the specific pin requirements for the nRF24L01+ on the Arduino Nano Every. By systematically testing various configurations, the program enabled pinpointing the optimal CSN pin connection, thus ensuring reliable communication with the radio module.

C. Lessons Learned

As a team, our key takeaways from this project are multifaceted and significantly enhance our understanding of both the theoretical and practical aspects of wireless communication using the nRF24L01+ module. Below, we outline our primary lessons:

- **Joystick Calibration and Implementation:** We learned to set and calibrate a joystick using the principles of the unit circle. This exercise not only improved our understanding of geometric principles in a practical setting but also allowed us to develop custom code that effectively translates joystick movements into digital commands. This skill is crucial for projects requiring precise control over inputs.
- **Debugging and Utilization of the nRF24L01+:** Throughout the project, we encountered and overcame various challenges associated with the nRF24L01+ module. These included issues with signal integrity, interference, and hardware compatibility. Our ability to debug these problems has enhanced our troubleshooting skills and our understanding of the module's operational parameters.
- **Exploration of Wireless Connection Options:** We explored various options for wireless connections, comparing the nRF24L01+ with other technologies such as Wi-Fi and Bluetooth. This comparison gave us insights into selecting the appropriate wireless technology based on factors like range, data rate, power consumption, and overall system requirements.
- **Handling RF Signals:** Dealing with radio frequency signals taught us about the complexities of RF design, including antenna placement, frequency selection, and the mitigation of environmental interference. These skills are vital for any work involving wireless communications.

These experiences have collectively broadened our technical expertise and prepared us for more advanced projects in wireless systems and embedded electronics.

VII. Conclusion

In today's technological landscape, the role of radio-frequency antennas and transmission systems cannot be overstated, facilitating seamless communication across various devices over significant distances. Our laboratory immersion provided a firsthand understanding of this technology, guiding us through its practical applications and the hurdles encountered. From troubleshooting signal interference to extending transmission ranges, we confronted these challenges head-on, fostering a culture of innovation and problem-solving.

Within the laboratory's controlled environment, we learned to operate radio-frequency systems and cultivated the skills necessary to adapt and innovate in the face of adversity. This hands-on experience went beyond theoretical knowledge, instilling confidence born from practical application. As we grappled with real-world constraints, we devised unique solutions, each obstacle serving as a stepping stone toward a deeper understanding of the technology.

Armed with this foundational knowledge and practical experience, we are poised to navigate the ever-evolving landscape of technology with confidence and competence. By immersing ourselves in radio-frequency systems early on, we have laid the groundwork for future success, reducing the barriers to entry and accelerating our trajectory toward innovation. Whether in academic research, industrial applications, or entrepreneurial ventures, our proficiency in radio- frequency communication positions us as catalysts for progress in an interconnected world.