```
#include <WiFi.h>
#include <PubSubClient.h>
#include <Adafruit_BNO08x.h>
#include <Arduino.h>
#include <ESP32Servo.h>

// WiFi Pratik
//const char* ssid = "Apt501"; // Enter your Wi-Fi name
//const char* password = "Pratik&Sandeer";  // Enter Wi-Fi password

// WiFi Smit
//const char* ssid = "Linksys01756"; // Enter your Wi-Fi name
//const char* password = "mzc1vfhxi5";  // Enter Wi-Fi password

// WiFi Hesse - NOT WORKING YET
//const char* ssid = "Berkeley-IoT"; // Enter your Wi-Fi name
//const char* password = "y&WCo0o!";  // Enter Wi-Fi password

//Wifi Berkeley Visitor
const char* ssid = "Smits iPhone 12 Pro"; // Enter your Wi-Fi name
const char* password = "smithotspot";  // Enter Wi-Fi password

// MQTT Broker
const char* mqtt_broker = "broker.emqx.io";
const char* topic = "emqx/esp32/pj/";
const char* mqtt_username = "emqx";
const char* mqtt_password = "public";
const int mqtt_port = 1883;

WiFiClient espClient;
PubSubClient client(espClient);

void callback(char *topic, byte *payload, unsigned int length) {
    // Handle incoming MQTT messages if needed
}

//UART BNO doesn't require reset
#define BNO08X_RESET -1

//Define BNO085
```

```
Adafruit_BNO08x bno08x(BNO08X_RESET);
sh2_SensorValue_t sensorValue;

//Declaring servos and pins
Servo servo1;
int servo1Pin = 2;
Servo servo2;
int servo2Pin = 8;
ESP32PWM pwm;

// Variables to store the previous linear acceleration values
float prevLinearAccX = 0.0;
float prevLinearAccY = 0.0;

// Variable to store the previous angular velocity values
float prevAngularVelocityZ = 0.0;

// Boolean operators for each type of turn
bool linTurn = false;
bool angTurn = false;

//Sensor values to report
void setReports(void) {
  if (!bno08x.enableReport(SH2_LINEAR_ACCELERATION)) {
    Serial.println("Could not enable linear acceleration");
  }
  if (!bno08x.enableReport(SH2_GYRO_INTEGRATED_RV)) {
    Serial.println("Could not enable gyro integrated remote vector");
  }
}

//Setup
void setup(void) {

//Set pin for LED
  pinMode(7, OUTPUT);
//Servo setup
  ESP32PWM::allocateTimer(0);
  ESP32PWM::allocateTimer(1);
  ESP32PWM::allocateTimer(2);
```

```
  ESP32PWM::allocateTimer(3);
  //Standard 50hz servo
  servo1.setPeriodHertz(50);
  servo1.attach(servo1Pin, 1000, 2000);
  servo2.setPeriodHertz(50);
  servo2.attach(servo2Pin, 1000, 2000);

  Serial.begin(115200);

  // Connecting to a WiFi network
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.println("Connecting to WiFi..");
  }
  Serial.println("Connected to the Wi-Fi network");

  // Connecting to an MQTT broker
  client.setServer(mqtt_broker, mqtt_port);
  client.setCallback(callback);
  while (!client.connected()) {
    String client_id = "esp32-client-";
    client_id += String(WiFi.macAddress());
    Serial.printf("The client %s connects to the public MQTT broker\n",
client_id.c_str());
    if (client.connect(client_id.c_str(), mqtt_username, mqtt_password)) {
      Serial.println("Public EMQX MQTT broker connected");
    } else {
      Serial.print("failed with state ");
      Serial.print(client.state());
      delay(2000);
    }
  }

//Initialize BNO085 over UART
  if (!bno08x.begin_UART(&Serial1)) {
    Serial.println("Failed to find BNO08x chip");
    while (1) {
      delay(10);
    }
```

```
  }
  Serial.println("BNO08x Found!");

  setReports();

  Serial.println("Reading events");
  delay(100);
}

void loop() {
  delay(10); //Report frequency

  if (bno08x.wasReset()) {
    Serial.print("sensor was reset ");
  }

  if (!bno08x.getSensorEvent(&sensorValue)) {
    return;
  }

  switch (sensorValue.sensorId) {
    case SH2_LINEAR_ACCELERATION:
      detectTurn(sensorValue, (sensorValue.un.gyroIntegratedRV.angVelZ) *
(180 / 3.14));
      break;
  }
}

void detectTurn(sh2_SensorValue_t sensorValue, float angularVelocityZ) {
  // Threshold values for detecting a turn based on acceleration changes
  float accThresholdX = 1.0;
  float accThresholdY = 0.5;

  // Calculate the current linear acceleration values
  float currentLinearAccX = abs(sensorValue.un.linearAcceleration.x);
  float currentLinearAccY = abs(sensorValue.un.linearAcceleration.y);

  // Check if linear acceleration in X is decreasing and Y is increasing
  if ((currentLinearAccX < prevLinearAccX - accThresholdX) &&
(currentLinearAccY > prevLinearAccY + accThresholdY)) {
```

```
    //Serial.println("Turn detected based on decreasing X and increasing Y
linear acceleration");
    linTurn = true;
    // Add actions to perform when a turn is detected
  } else {
    linTurn = false;
  }

  // Update the previous values for the next iteration
  prevLinearAccX = currentLinearAccX;
  prevLinearAccY = currentLinearAccY;

  // Threshold value for detecting a turn based on angular velocity around
Z-axis
  float turnThreshold = 65.0; // Adjust this value based on your needs

  // Check for a significant change in angular velocity around Z-axis
  if (abs(angularVelocityZ) > turnThreshold && abs(angularVelocityZ) >
abs(prevAngularVelocityZ)) {
    //Serial.println("Car is turning based on angular velocity around
Z-axis");
    angTurn = true;
    // Add actions to perform when a turn is detected based on angular
velocity
  } else {
    angTurn = false;
  }

  // Update the previous value for the next iteration
  prevAngularVelocityZ = angularVelocityZ;

//Loop for actions to perform when turn is detected from both types of
turn detection
  if (linTurn && angTurn) {
    String payload = String("car turn");
    //Serial.println("car turn");
    client.publish(topic, payload.c_str());
    servo1.write(70);
    servo2.write(95);
    digitalWrite(7,HIGH);
```

```
    delay(1250);
  }
  else{
    servo1.write(155);
    servo2.write(180);
    digitalWrite(7, LOW);
  }
}
```