Arduino Car Code

```
#include <Wire.h>
#include <MPU6050_light.h>
#define buadRate 9600
// String constants used for parsing serial commands
String sequenceCommand = "executeSequence"  , pathCommand = "executePath";
String path_angleCommand = "angle"          , path_angleDirCommand = "dir";
String path_distanceCommand = "distance"    , path_speedCommand = "speed";
String path_substringCommand = "&"          , path_equalCommand = "=";

int pathDetails[3]; // Array to store path params (angle, distance, speed)
MPU6050 mpu(Wire); // MPU6050 sensor object for reading IMU data

// Motor control pins
#define motorL1   7
#define motorL2   6
#define motorR1   9
#define motorR2   8
#define motorEN1  5
#define motorEN2  10
#define sensor    2

// Speed settings
#define speedTurn           80
#define speed_error_factor  11
#define speed_adjust_factor 10


// Shape dimensions and speeds
#define squareLength        100
#define squareSpeed         150

#define triangleLength      100
#define triangleSpeed       150

#define rectangleLength     100
#define rectangleWidth      150
#define rectangleSpeed      150

int steps = 0;  // Tracks the number of steps (encoder counts)
int distance = 0;  // Distance traveled based on encoder steps
float yaw;  // Variable to hold the yaw angle from the MPU6050
```

```
void setup() {
  Serial.begin(buadRate); // Init serial communication at a baud rate of 9600
  Wire.begin(); // Initialize I2C communication
  mpu.begin(); // Initialize the MPU6050 sensor
  mpu.calcOffsets(); // Calibrate MPU6050 sensor
  mpu.update(); // Update sensor readings
  yaw = mpu.getAngleZ(); // Initialize yaw angle
  pinMode(sensor, INPUT); // Set sensor pin as input
  pinMode(motorR1, OUTPUT); // Set motor control pins as outputs
  pinMode(motorR2, OUTPUT);
  pinMode(motorL1, OUTPUT);
  pinMode(motorL2, OUTPUT);
  pinMode(motorEN1, OUTPUT);
  pinMode(motorEN2, OUTPUT);
  moveForward(); // Start by moving the car forward
}

void loop() {
  mpu.update(); // Continuously update MPU6050 sensor readings
  yaw = mpu.getAngleZ(); // Get the current yaw angle
  sendData("false", yaw, distance, 0); // Send current data over serial
  if (Serial.available()) {
    // Check if data is available on the serial port
    checkCommand(Serial.readStringUntil('\r'));//Read command and process it
  }
}
```

Arduino Helper Functions

```
/**
 * Function to set the car's path and control its movement.
 * This involves turning the car to a specific angle and then moving forward
 * for a specified distance while adjusting the speed based on the yaw angle.
 * @param initial_angle - The target yaw angle to turn the car to.
 * @param path_distance - The distance the car should travel after turning.
 * @param speed - The speed at which the car should move.
 */
void SetCarPath(signed int initial_angle, int path_distance, char speed) {
  mpu.update(); // Update sensor readings
  turnCar(initial_angle, speedTurn); // Turn the car to the specified angle
  moveForward(); // Start moving forward
  distance = 0; // Reset distance traveled
  steps = 0; // Reset step count

  // Continue moving until the car reaches the specified distance
  while (distance < path_distance) {
    // Adjust motor speeds based on the current yaw angle
    analogWrite(motorEN1,speed+(yaw-initial_angle)*speed_adjust_factor+speed_error_factor);
    analogWrite(motorEN2,speed-(yaw-initial_angle)*speed_adjust_factor-speed_error_factor);

    if (digitalRead(sensor)) { // Check if the sensor detects a step
      steps += 1;
      distance = (steps / 90.0) * 100; // Convert steps to distance
      while (digitalRead(sensor)); // Wait for the sensor to clear
    }
    mpu.update(); // Update sensor readings
    yaw = mpu.getAngleZ(); // Update yaw angle
  }
  sendData("true", yaw, distance, speed); // Send final status
  instantStop(); // Stop the car
}
```

```
/**
 * Function to send the current status of the car via serial communication.
 * @param carIsMoving - Indicates if the car is currently moving ("true" or "false").
 * @param angle - The current yaw angle of the car.
 * @param dis - The distance traveled by the car.
 * @param speed - The current speed of the car.
 */
void sendData(String carIsMoving, signed int angle, int dis, int speed) {
  Serial.print("status:carIsMoving=");
  Serial.print(carIsMoving);
  Serial.print("&dir=");

  // Determine the direction based on the angle (right or left)
  if (angle < 0) {
    Serial.print("r"); // Right
  } else {
    Serial.print("l"); // Left
  }

  Serial.print("&angle=");
  Serial.print(abs(angle)); // Send the absolute value of the angle
  Serial.print("&distance=");
  Serial.print(dis); // Send the distance traveled
  Serial.print("&speed=");
  Serial.print(speed); // Send the current speed
  Serial.print("\r"); // End of message
}
/**
 * Function to execute a sequence of movements based on the command received.
 * @param str - The command string indicating the sequence to execute (e.g., "moveSquare").
 */
void performSequence(String str) {
  Serial.print("executeSequence:success=true&status=inProgress\r");
  mpu.update(); // Update sensor readings
  // Execute specific movement based on the command
  if (str == "moveSquare") {
    moveSquare(squareLength, squareSpeed);
  }
  if (str == "moveRectangle") {
    moveRectangle(rectangleLength, rectangleWidth, rectangleSpeed);
  }
  if (str == "moveTriangle") {
    moveTriangle(triangleLength, triangleSpeed);
  }
  Serial.print("executeSequence:success=true&status=completed\r");
}
```

```
/**
 * Function to parse and execute commands received via serial communication.
 * It determines whether the command is a sequence or a path and acts accordingly.
 * @param str - The command string received via serial communication.
 */
void checkCommand(String str) {
  mpu.update(); // Update sensor readings
  // Check if the command is a sequence command
  if (str.indexOf(sequenceCommand) > -1) {
    str.remove(str.indexOf(sequenceCommand), str.indexOf(sequenceCommand)+sequenceCommand.length()+6);
    performSequence(str); // Execute the sequence command
  }
  // Check if the command is a path command
  if (str.indexOf(pathCommand) > -1) {
    String param, val;
    str.remove(0, str.indexOf(pathCommand) + pathCommand.length() + 1);
    // Parse the parameters from the command string
    while (str.length() > 0) {
      mpu.update();
      param = str.substring(0, str.indexOf(path_equalCommand)); // Get parameter name
      str.remove(0, str.indexOf(path_equalCommand) + 1);

      val = str.substring(0, str.indexOf(path_substringCommand)); // Get parameter value
      str.remove(0, str.indexOf(val) + val.length());

      // Assign values to the pathDetails array based on parameter names
      if (param == path_angleCommand) {
        pathDetails[0] = val.toInt(); // Set angle
      }
      if (param == path_angleDirCommand && val == "r") {
        pathDetails[0] = -pathDetails[0]; // Reverse angle direction if "r"
      }
      if (param == path_distanceCommand) {
        pathDetails[1] = val.toInt(); // Set distance
      }
      if (param == path_speedCommand) {
        pathDetails[2] = val.toInt(); // Set speed
      }
      str.remove(0, str.indexOf(path_substringCommand) + 1);
    }
    Serial.print("executePath:success=true&status=inProgress\r");
    SetCarPath(yaw + pathDetails[0], pathDetails[1], pathDetails[2]); // Execute the path command
    Serial.print("executePath:success=true&status=completed\r");
  }
}
```

```
/**
 * Function to move the car in a square pattern.
 * The car moves forward along the sides of a square, turning 90 degrees at each corner.
 * @param length - The length of each side of the square.
 * @param speed - The speed at which the car should move.
 */
void moveSquare(float length, char speed) {
  SetCarPath(yaw, length, speed);         // Move forward for the first side
  SetCarPath(yaw - 90, length, speed);    // Turn and move for the second side
  SetCarPath(yaw - 90, length, speed);    // Turn and move for the third side
  SetCarPath(yaw - 90, length, speed);    // Turn and move for the last side
  turnCar(yaw - 180, speedTurn);          // Final turn to complete the square
}

/**
 * Function to move the car in a triangle pattern.
 * The car moves forward along the sides of an equilateral triangle, turning 120 degrees at
each corner.
 * @param length - The length of each side of the triangle.
 * @param speed - The speed at which the car should move.
 */
void moveTriangle(float length, char speed) {
  SetCarPath(yaw, length, speed);         // Move forward for the first side
  SetCarPath(yaw - 120, length, speed);    // Turn and move for the second side
  SetCarPath(yaw - 120, length, speed);    // Turn and move for the third side
  turnCar(yaw - 180, speedTurn);          // Final turn to complete the triangle
}

/**
 * Function to move the car in a rectangular pattern.
 * The car moves forward along the sides of a rectangle, turning 90 degrees at each corner.
 * @param length - The length of the rectangle.
 * @param width - The width of the rectangle.
 * @param speed - The speed at which the car should move.
 */
void moveRectangle(float length, float width, char speed) {
  SetCarPath(yaw, length, speed);         // Move forward for the first side (length)
  SetCarPath(yaw - 90, width, speed);     // Turn and move for the second side (width)
  SetCarPath(yaw - 90, length, speed);    // Turn and move for the third side (length)
  SetCarPath(yaw - 90, width, speed);     // Turn and move for the fourth side (width)
  turnCar(yaw - 90, speedTurn);           // Final turn to complete the rectangle
}
```

```
/**
 * Function to turn the car to a specific yaw angle.
 * The car will adjust its direction until the desired angle is reached.
 * @param ang - The target yaw angle to turn the car to.
 * @param speed - The speed at which the car should turn.
 */
void turnCar(signed int ang, int speed) {
  while (1) {
    mpu.update(); // Update sensor readings
    yaw = mpu.getAngleZ(); // Get the current yaw angle

    if (yaw > (ang + 1)) { // If the car needs to turn right
      moveRight();         // Turn the car to the right
      applyCarSpeed(speed); // Apply the turning speed
    } else if (yaw < (ang - 1)) { // If the car needs to turn left
      moveLeft();          // Turn the car to the left
      applyCarSpeed(speed); // Apply the turning speed
    } else {
      instantStop(); // Stop the car when the desired angle is reached
      break;
    }
  }
}

/**
 * Function to apply a specified speed to both motors of the car.
 * @param speed - The speed value to apply to the motors (0-255).
 */
void applyCarSpeed(int speed) {
  analogWrite(motorEN1, speed); // Set speed for the right motor
  analogWrite(motorEN2, speed); // Set speed for the left motor
}

/**
 * Function to immediately stop the car by setting all motor pins high.
 */
void instantStop() {
  digitalWrite(motorR1, HIGH);
  digitalWrite(motorR2, HIGH);
  digitalWrite(motorL1, HIGH);
  digitalWrite(motorL2, HIGH);
}
```

```
// Movement control functions

/**
 * Function to move the car forward.
 * The right and left motors are set to rotate in the forward direction.
 */
void moveForward() {
  digitalWrite(motorR1, LOW);
  digitalWrite(motorR2, HIGH);
  digitalWrite(motorL1, LOW);
  digitalWrite(motorL2, HIGH);
}

/**
 * Function to turn the car to the right.
 * The right motor moves backward and the left motor moves forward.
 */
void moveRight() {
  digitalWrite(motorR1, HIGH);
  digitalWrite(motorR2, LOW);
  digitalWrite(motorL1, LOW);
  digitalWrite(motorL2, HIGH);
}

/**
 * Function to move the car backward.
 * The right and left motors are set to rotate in the reverse direction.
 */
void moveBackward() {
  digitalWrite(motorR1, HIGH);
  digitalWrite(motorR2, LOW);
  digitalWrite(motorL1, HIGH);
  digitalWrite(motorL2, LOW);
}

/**
 * Function to turn the car to the left.
 * The right motor moves forward and the left motor moves backward.
 */
void moveLeft() {
  digitalWrite(motorR1, LOW);
  digitalWrite(motorR2, HIGH);
  digitalWrite(motorL1, HIGH);
  digitalWrite(motorL2, LOW);
}
```