13. APPENDICES

All supporting documents including technical specifications, tables, charts, and detailed data

Appendix A: Technical Specifications

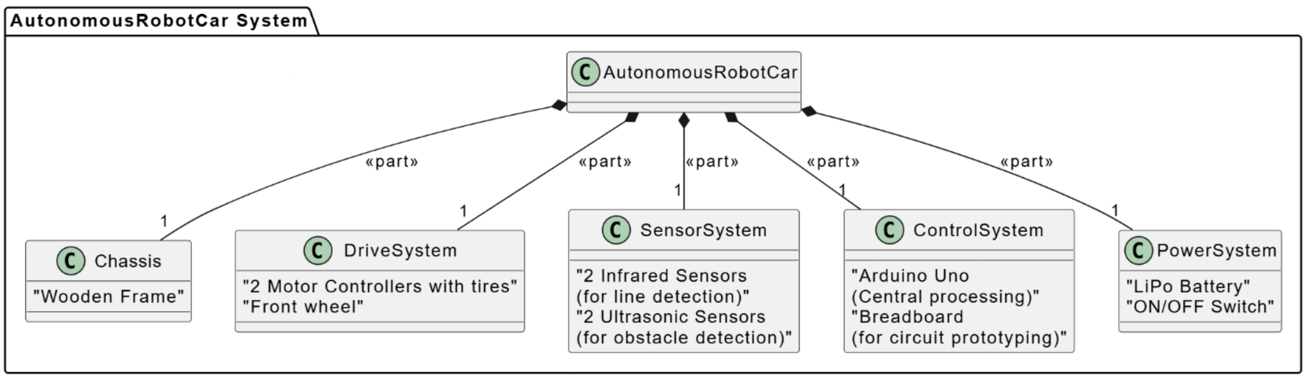
Component	Specification
IR Sensors	Digital reflectance sensors, 3–5V, 20–30 cm range
Ultrasonic Sensor	HC-SR04, 2–400 cm range, 5V operating voltage
Color Sensor	TCS3200, RGB color detection, 3–5V input
Motor Driver	SBC Motor Driver 2, dual channel, 5–30V DC
Power Source	12V LiPo Battery, 2200 mAh
Microcontroller	Arduino Uno R4 WiFI (ATmega328P)
Chassis Material	Acrylic frame with 3D-printed sensor holders
Wheels and Motors	DC geared motors, 100 RPM, plastic wheels

[2], [4].

Appendix B: PID Control Parameters

Parameter	**Value**
P (Proportional)	10
I (Integral)	0
D (Derivative)	10

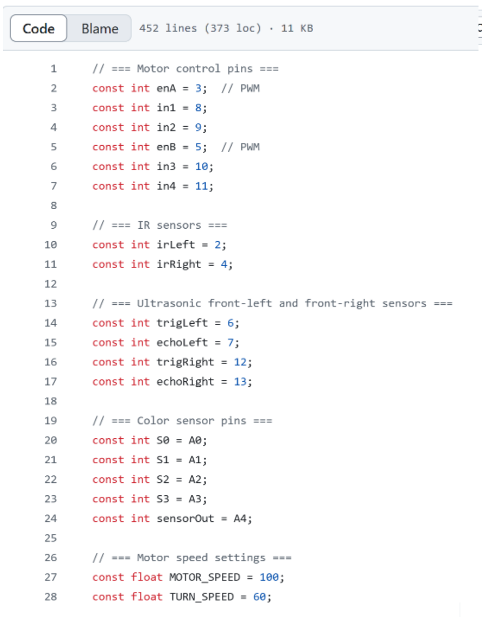
Appendix C: System Block Diagram



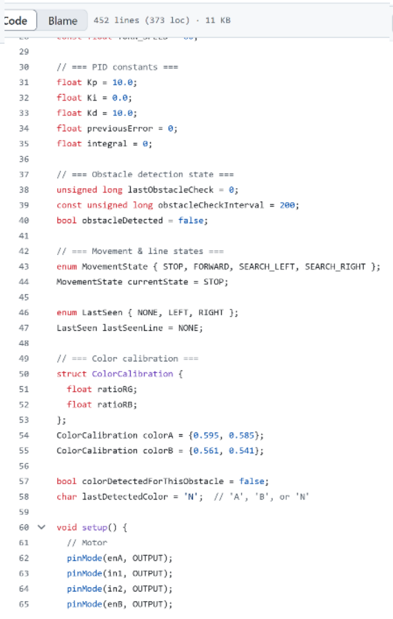
Appendix D: Test Results Summary

Test Scenario	Outcome	Notes
Lane following on white background	Successful	Minor tuning required for sharp turns
Obstacle avoidance at 25 cm	Successful	Smooth detour and return to lane
Colour detection under indoor light	Reliable (Colour A & B detected)	Requires recalibration under sunlight
Battery endurance test	2.5 hours of operation	Under typical use with full charge

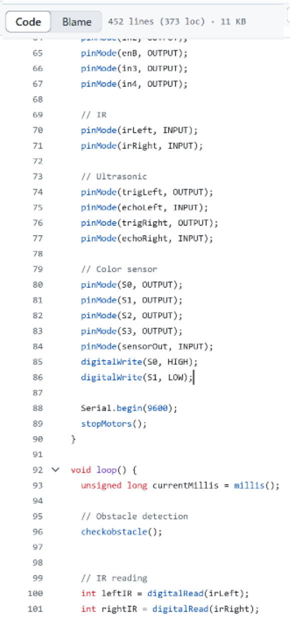
Appendix E: Source Code Snapshots



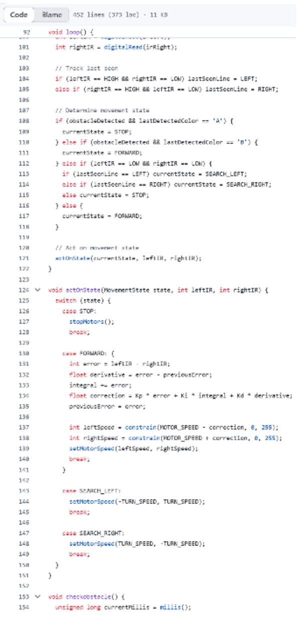
```
// === Motor control pins ===
const int enA = 3;  // PWM
const int in1 = 8;
const int in2 = 9;
const int enB = 5;  // PWM
const int in3 = 10;
const int in4 = 11;

// === IR sensors ===
const int irLeft = 2;
const int irRight = 4;

// === Ultrasonic front-left and front-right sensors ===
const int trigLeft = 6;
const int echoLeft = 7;
const int trigRight = 12;
const int echoRight = 13;

// === Color sensor pins ===
const int S0 = A0;
const int S1 = A1;
const int S2 = A2;
const int S3 = A3;
const int sensorOut = A4;

// === Motor speed settings ===
const float MOTOR_SPEED = 100;
const float TURN_SPEED = 60;
```



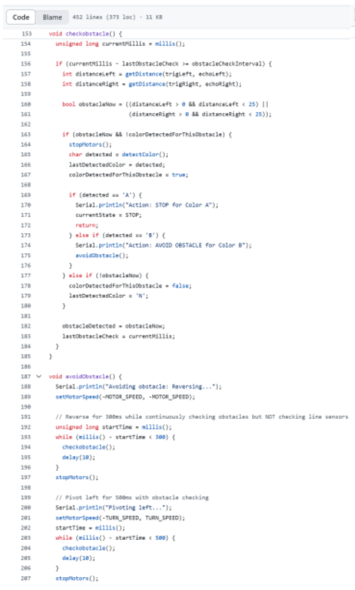
```

// === PID constants ===
float Kp = 10.0;
float Ki = 0.0;
float Kd = 10.0;
float previousError = 0;
float integral = 0;

// === Obstacle detection state ===
unsigned long lastObstacleCheck = 0;
const unsigned long obstacleCheckInterval = 200;
bool obstacleDetected = false;

// === Movement & line states ===
enum MovementState { STOP, FORWARD, SEARCH_LEFT, SEARCH_RIGHT };
MovementState currentState = STOP;

enum LastSeen { NONE, LEFT, RIGHT };
LastSeen lastSeenLine = NONE;

// === Color calibration ===
struct ColorCalibration {
  float ratioRG;
  float ratioRB;
};
ColorCalibration colorA = {0.595, 0.585};
ColorCalibration colorB = {0.561, 0.541};

bool colorDetectedForThisObstacle = false;
char lastDetectedColor = 'N';  // 'A', 'B', or 'N'

void setup() {
  // Motor
  pinMode(enA, OUTPUT);
  pinMode(in1, OUTPUT);
  pinMode(in2, OUTPUT);
  pinMode(enB, OUTPUT);
```



```
  pinMode(enB, OUTPUT);
  pinMode(in3, OUTPUT);
  pinMode(in4, OUTPUT);

  // IR
  pinMode(irLeft, INPUT);
  pinMode(irRight, INPUT);

  // Ultrasonic
  pinMode(trigLeft, OUTPUT);
  pinMode(echoLeft, INPUT);
  pinMode(trigRight, OUTPUT);
  pinMode(echoRight, INPUT);

  // Color sensor
  pinMode(S0, OUTPUT);
  pinMode(S1, OUTPUT);
  pinMode(S2, OUTPUT);
  pinMode(S3, OUTPUT);
  pinMode(sensorOut, INPUT);
  digitalWrite(S0, HIGH);
  digitalWrite(S1, LOW);

  Serial.begin(9600);
  stopMotors();
}

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

  // Obstacle detection
  checkobstacle();


  // IR reading
  int leftIR = digitalRead(irLeft);
  int rightIR = digitalRead(irRight);
```

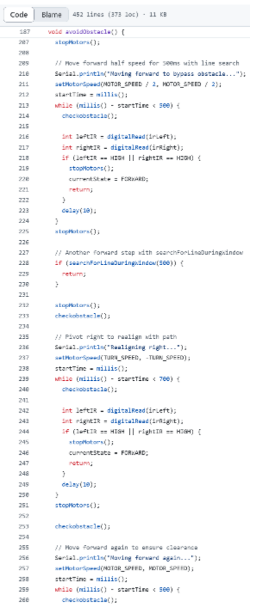


```
void loop() {
  int rightIR = digitalRead(irRight);

  // Track last seen
  if (leftIR == HIGH && rightIR == LOW) lastSeenLine = LEFT;
  else if (rightIR == HIGH && leftIR == LOW) lastSeenLine = RIGHT;

  // Determine movement state
  if (obstacleDetected && lastDetectedColor == 'A') {
    currentState = STOP;
  } else if (obstacleDetected && lastDetectedColor == 'B') {
    currentState = FORWARD;
  } else if (leftIR == LOW && rightIR == LOW) {
    if (lastSeenLine == LEFT) currentState = SEARCH_LEFT;
    else if (lastSeenLine == RIGHT) currentState = SEARCH_RIGHT;
    else currentState = STOP;
  } else {
    currentState = FORWARD;
  }

  // Act on movement state
  actOnState(currentState, leftIR, rightIR);
}

void actOnState(MovementState state, int leftIR, int rightIR) {
  switch (state) {
    case STOP:
      stopMotors();
      break;

    case FORWARD: {
      int error = leftIR - rightIR;
      float derivative = error - previousError;
      integral += error;
      float correction = Kp * error + Ki * integral + Kd * derivative;
      previousError = error;

      int leftSpeed = constrain(MOTOR_SPEED - correction, 0, 255);
      int rightSpeed = constrain(MOTOR_SPEED + correction, 0, 255);
      setMotorSpeed(leftSpeed, rightSpeed);
      break;
    }

    case SEARCH_LEFT:
      setMotorSpeed(-TURN_SPEED, TURN_SPEED);
      break;

    case SEARCH_RIGHT:
      setMotorSpeed(TURN_SPEED, -TURN_SPEED);
      break;
  }
}

void checkobstacle() {
  unsigned long currentMillis = millis();
```



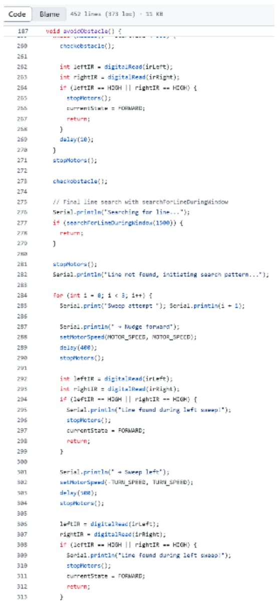
Code Blame 452 lines (373 loc) · 11 KB

```
void checkobstacle() {
  unsigned long currentMillis = millis();

  if (currentMillis - lastObstacleCheck >= obstacleCheckInterval) {
    int distanceLeft = getDistance(trigLeft, echoLeft);
    int distanceRight = getDistance(trigRight, echoRight);

    bool obstacleNow = ((distanceLeft > 0 && distanceLeft < 25) ||
                        (distanceRight > 0 && distanceRight < 25));

    if (obstacleNow && !colorDetectedForThisObstacle) {
      stopMotors();
      char detected = detectColor();
      lastDetectedColor = detected;
      colorDetectedForThisObstacle = true;

      if (detected == 'A') {
        Serial.println("Action: STOP for Color A");
        currentState = STOP;
        return;
      } else if (detected == 'B') {
        Serial.println("Action: AVOID OBSTACLE for Color B");
        avoidObstacle();
      }
    } else if (!obstacleNow) {
      colorDetectedForThisObstacle = false;
      lastDetectedColor = 'N';
    }

    obstacleDetected = obstacleNow;
    lastObstacleCheck = currentMillis;
  }
}

void avoidObstacle() {
  Serial.println("Avoiding obstacle: Reversing...");
  setMotorSpeed(-MOTOR_SPEED, -MOTOR_SPEED);

  // Reverse for 300ms while continuously checking obstacles but NOT checking line sensors
  unsigned long startTime = millis();
  while (millis() - startTime < 300) {
    checkobstacle();
    delay(10);
  }
  stopMotors();

  // Pivot left for 500ms with obstacle checking
  Serial.println("Pivoting left...");
  setMotorSpeed(-TURN_SPEED, TURN_SPEED);
  startTime = millis();
  while (millis() - startTime < 500) {
    checkobstacle();
    delay(10);
  }
  stopMotors();
```



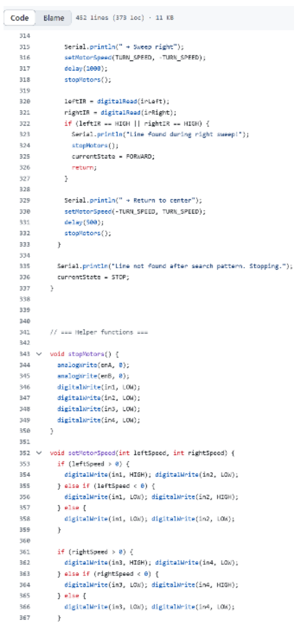
Code Blame 452 lines (373 loc) · 11 KB

```
void avoidObstacle() {
  stopMotors();

  // Move forward half speed for 500ms with line search
  Serial.println("Moving forward to bypass obstacle...");
  setMotorSpeed(MOTOR_SPEED / 2, MOTOR_SPEED / 2);
  startTime = millis();
  while (millis() - startTime < 500) {
    checkobstacle();

    int leftIR = digitalRead(irLeft);
    int rightIR = digitalRead(irRight);
    if (leftIR == HIGH || rightIR == HIGH) {
      stopMotors();
      currentState = FORWARD;
      return;
    }
    delay(10);
  }
  stopMotors();

  // Another forward step with searchForLineDuringWindow
  if (searchForLineDuringWindow(500)) {
    return;
  }

  stopMotors();
  checkobstacle();

  // Pivot right to realign with path
  Serial.println("Realigning right...");
  setMotorSpeed(TURN_SPEED, -TURN_SPEED);
  startTime = millis();
  while (millis() - startTime < 700) {
    checkobstacle();

    int leftIR = digitalRead(irLeft);
    int rightIR = digitalRead(irRight);
    if (leftIR == HIGH || rightIR == HIGH) {
      stopMotors();
      currentState = FORWARD;
      return;
    }
    delay(10);
  }
  stopMotors();

  checkobstacle();

  // Move forward again to ensure clearance
  Serial.println("Moving forward again...");
  setMotorSpeed(MOTOR_SPEED, MOTOR_SPEED);
  startTime = millis();
  while (millis() - startTime < 500) {
    checkobstacle();
```



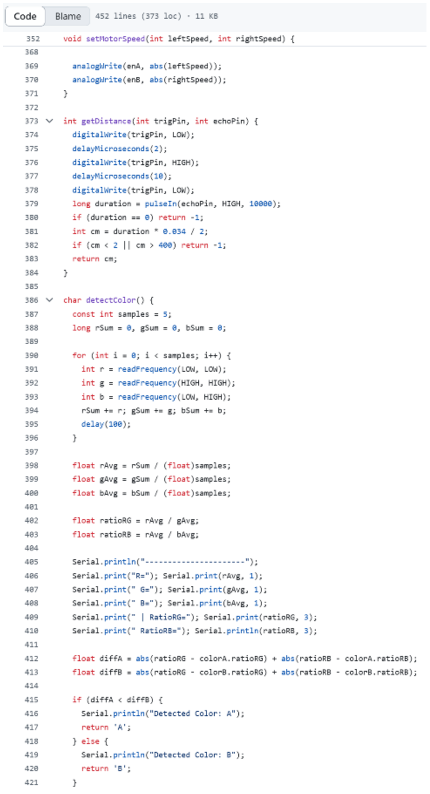
Code Blame 452 lines (373 loc) · 11 KB

```
void avoidObstacle() {
    checkobstacle();

    int leftIR = digitalRead(irLeft);
    int rightIR = digitalRead(irRight);
    if (leftIR == HIGH || rightIR == HIGH) {
      stopMotors();
      currentState = FORWARD;
      return;
    }
    delay(10);
  }
  stopMotors();

  checkobstacle();

  // Final line search with searchForLineDuringWindow
  Serial.println("Searching for line...");
  if (searchForLineDuringWindow(1500)) {
    return;
  }

  stopMotors();
  Serial.println("Line not found, initiating search pattern...");

  for (int i = 0; i < 3; i++) {
    Serial.print("Sweep attempt "); Serial.println(i + 1);

    Serial.println(" → Nudge forward");
    setMotorSpeed(MOTOR_SPEED, MOTOR_SPEED);
    delay(400);
    stopMotors();

    int leftIR = digitalRead(irLeft);
    int rightIR = digitalRead(irRight);
    if (leftIR == HIGH || rightIR == HIGH) {
      Serial.println("Line found during left sweep!");
      stopMotors();
      currentState = FORWARD;
      return;
    }

    Serial.println(" → Sweep left");
    setMotorSpeed(-TURN_SPEED, TURN_SPEED);
    delay(500);
    stopMotors();

    leftIR = digitalRead(irLeft);
    rightIR = digitalRead(irRight);
    if (leftIR == HIGH || rightIR == HIGH) {
      Serial.println("Line found during left sweep!");
      stopMotors();
      currentState = FORWARD;
      return;
    }
```



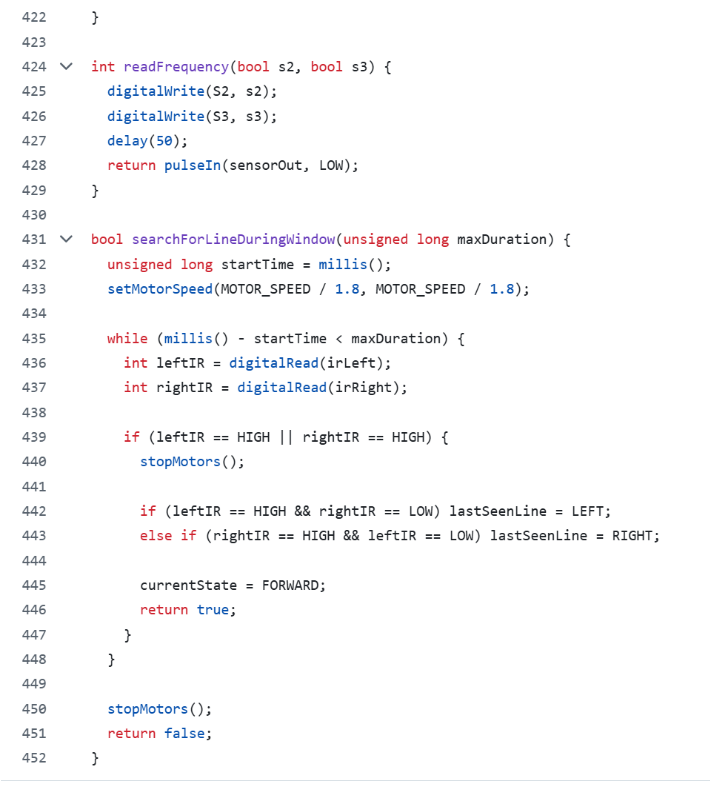
Code Blame 452 lines (373 loc) · 11 KB

```

    Serial.println(" → Sweep right");
    setMotorSpeed(TURN_SPEED, -TURN_SPEED);
    delay(1000);
    stopMotors();

    leftIR = digitalRead(irLeft);
    rightIR = digitalRead(irRight);
    if (leftIR == HIGH || rightIR == HIGH) {
      Serial.println("Line found during right sweep!");
      stopMotors();
      currentState = FORWARD;
      return;
    }

    Serial.println(" → Return to center");
    setMotorSpeed(-TURN_SPEED, TURN_SPEED);
    delay(500);
    stopMotors();
  }

  Serial.println("Line not found after search pattern. Stopping.");
  currentState = STOP;
}



// === Helper functions ===

void stopMotors() {
  analogWrite(enA, 0);
  analogWrite(enB, 0);
  digitalWrite(in1, LOW);
  digitalWrite(in2, LOW);
  digitalWrite(in3, LOW);
  digitalWrite(in4, LOW);
}

void setMotorSpeed(int leftSpeed, int rightSpeed) {
  if (leftSpeed > 0) {
    digitalWrite(in1, HIGH); digitalWrite(in2, LOW);
  } else if (leftSpeed < 0) {
    digitalWrite(in1, LOW); digitalWrite(in2, HIGH);
  } else {
    digitalWrite(in1, LOW); digitalWrite(in2, LOW);
  }

  if (rightSpeed > 0) {
    digitalWrite(in3, HIGH); digitalWrite(in4, LOW);
  } else if (rightSpeed < 0) {
    digitalWrite(in3, LOW); digitalWrite(in4, HIGH);
  } else {
    digitalWrite(in3, LOW); digitalWrite(in4, LOW);
  }
```



Code Blame 452 lines (373 loc) · 11 KB

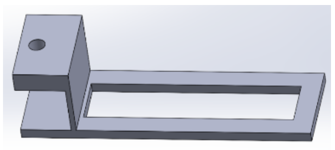
```
void setMotorSpeed(int leftSpeed, int rightSpeed) {

  analogWrite(enA, abs(leftSpeed));
  analogWrite(enB, abs(rightSpeed));
}

int getDistance(int trigPin, int echoPin) {
  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);
  long duration = pulseIn(echoPin, HIGH, 10000);
  if (duration == 0) return -1;
  int cm = duration * 0.034 / 2;
  if (cm < 2 || cm > 400) return -1;
  return cm;
}

char detectColor() {
  const int samples = 5;
  long rSum = 0, gSum = 0, bSum = 0;

  for (int i = 0; i < samples; i++) {
    int r = readFrequency(LOW, LOW);
    int g = readFrequency(HIGH, HIGH);
    int b = readFrequency(LOW, HIGH);
    rSum += r; gSum += g; bSum += b;
    delay(100);
  }

  float rAvg = rSum / (float)samples;
  float gAvg = gSum / (float)samples;
  float bAvg = bSum / (float)samples;

  float ratioRG = rAvg / gAvg;
  float ratioRB = rAvg / bAvg;

  Serial.println("---------------------");
  Serial.print("R="); Serial.print(rAvg, 1);
  Serial.print(" G="); Serial.print(gAvg, 1);
  Serial.print(" B="); Serial.print(bAvg, 1);
  Serial.print(" | RatioRG="); Serial.print(ratioRG, 3);
  Serial.print(" RatioRB="); Serial.println(ratioRB, 3);

  float diffA = abs(ratioRG - colorA.ratioRG) + abs(ratioRB - colorA.ratioRB);
  float diffB = abs(ratioRG - colorB.ratioRG) + abs(ratioRB - colorB.ratioRB);

  if (diffA < diffB) {
    Serial.println("Detected Color: A");
    return 'A';
  } else {
    Serial.println("Detected Color: B");
    return 'B';
  }
```



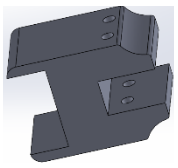
```
}

int readFrequency(bool s2, bool s3) {
  digitalWrite(S2, s2);
  digitalWrite(S3, s3);
  delay(50);
  return pulseIn(sensorOut, LOW);
}

bool searchForLineDuringWindow(unsigned long maxDuration) {
  unsigned long startTime = millis();
  setMotorSpeed(MOTOR_SPEED / 1.8, MOTOR_SPEED / 1.8);

  while (millis() - startTime < maxDuration) {
    int leftIR = digitalRead(irLeft);
    int rightIR = digitalRead(irRight);

    if (leftIR == HIGH || rightIR == HIGH) {
      stopMotors();

      if (leftIR == HIGH && rightIR == LOW) lastSeenLine = LEFT;
      else if (rightIR == HIGH && leftIR == LOW) lastSeenLine = RIGHT;

      currentState = FORWARD;
      return true;
    }
  }

  stopMotors();
  return false;
}
```

Appendix F: 3D-Printed Part Designs

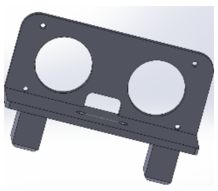
(a) IR Sensor holder



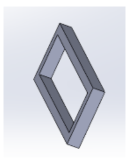
(b) Motor Holder



(c) Ultrasonic Sensor Holder [2].



(d) IR Sensor Clip



(d) Battery and Breadboard holder