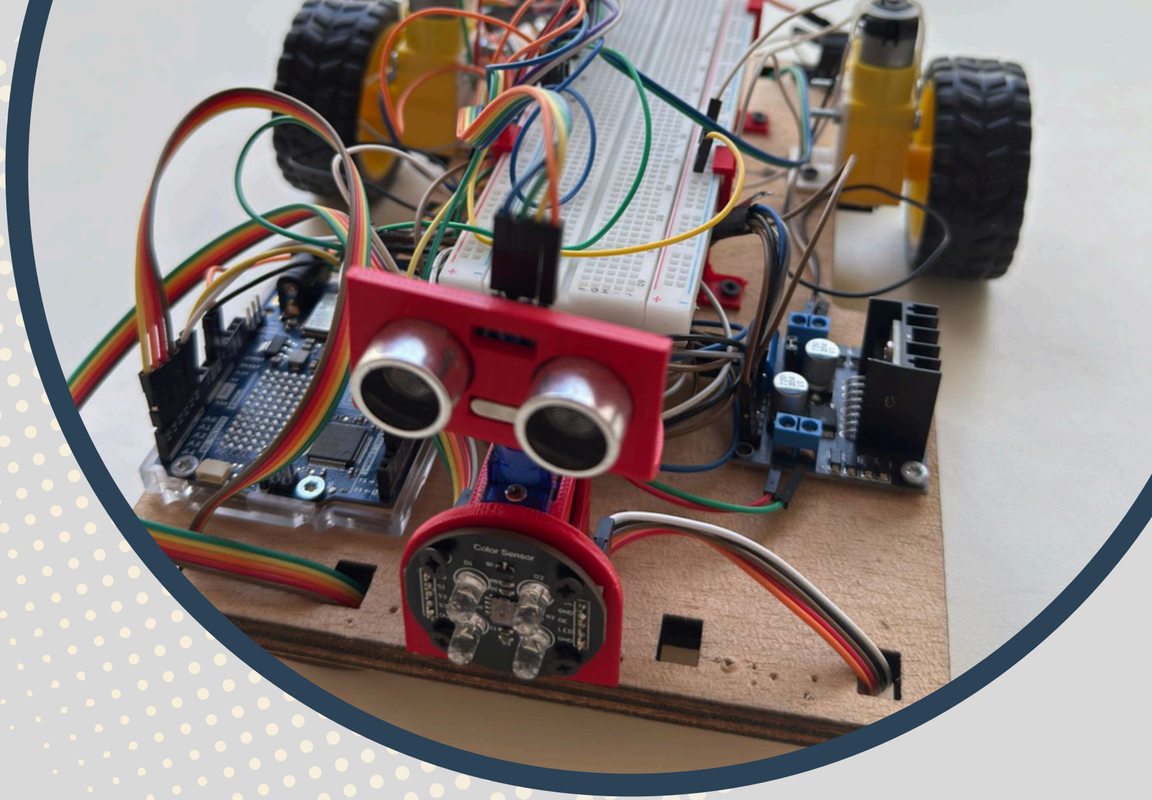
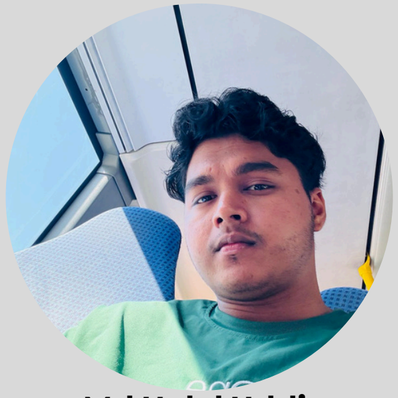
AUTONOMOUS ROBOT CAR



PRESENTED BY: TEAM D4

Md Zulkar Nain Sayed



Md Helal Uddin

Md Arman Zaid Efty

SYSTEM OVERVIEW

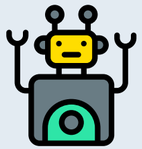
1. Sensor Suite: IR (KY-033), ultrasonic (HC-SR04), and color (TCS3200) sensors enable real-time environment awareness.

2. Adaptive Navigation: Follows line path and dynamically avoids obstacles using ultrasonic Seosor.

3. Control Core: Arduino Uno R4 WiFi handles sensor data and executes movement logic.

4. Motor Interface: SBC-MotoDriver2 allows precise, bidirectional control of motors.

5. Color-Based Action: Detects object color — avoids red, pushes blue, then resumes line following



USAGE OF GITHUB PROFESSIONALLY :

Updated activity diagram
Md-helaluddin committed 11 hours ago

Updated Sequence Diagram
Md-helaluddin committed 11 hours ago

Photo of activity Diagram
Md-helaluddin committed 11 hours ago

Block dd photo
Md-helaluddin committed 11 hours ago

Photo ibd
Md-helaluddin committed 11 hours ago

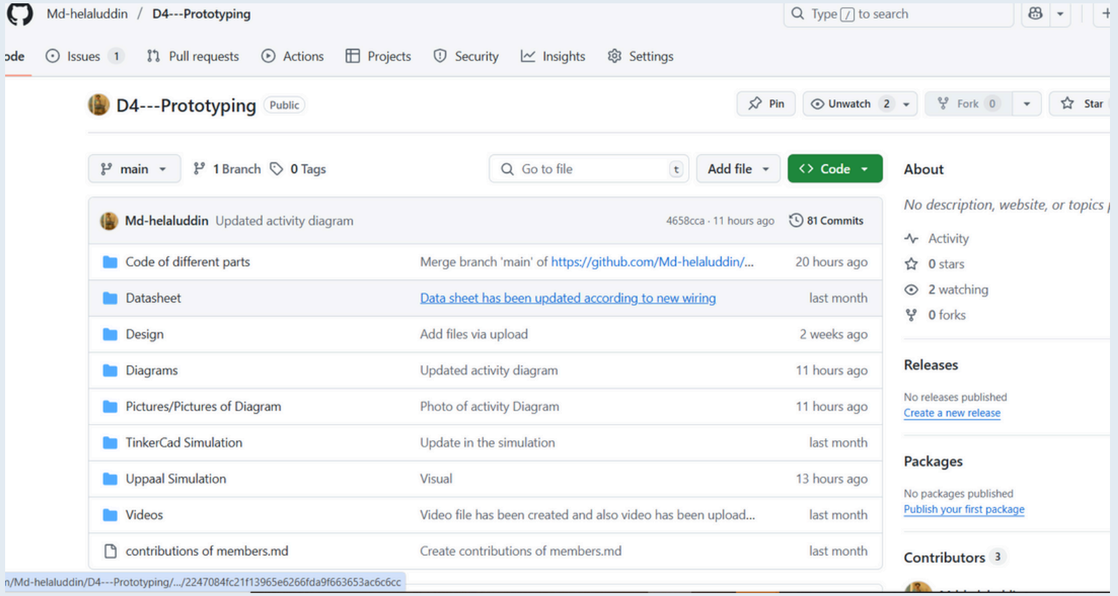
Photo of Use case diagram
Md-helaluddin committed 11 hours ago

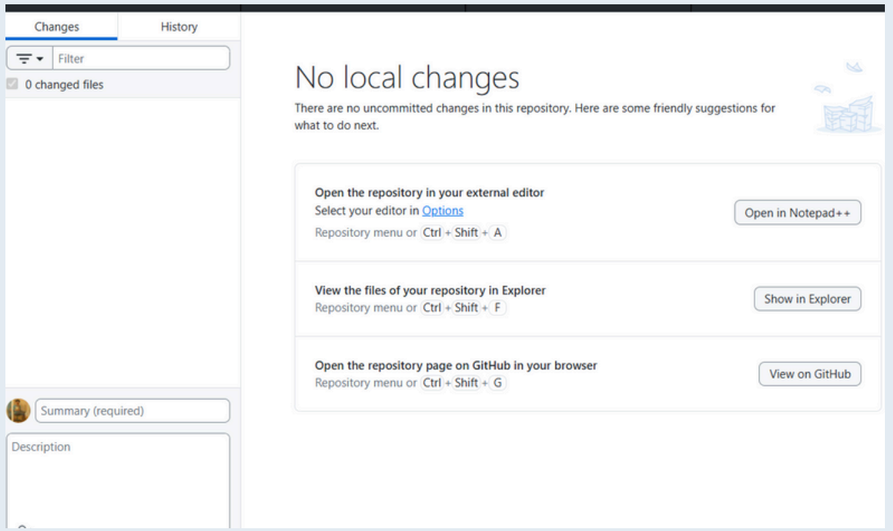
Picture requirement diagram
Md-helaluddin committed 11 hours ago

Picture of non functional requirement Diagram
Md-helaluddin committed 11 hours ago

Picture of Sequence Diagram
Md-helaluddin committed 11 hours ago

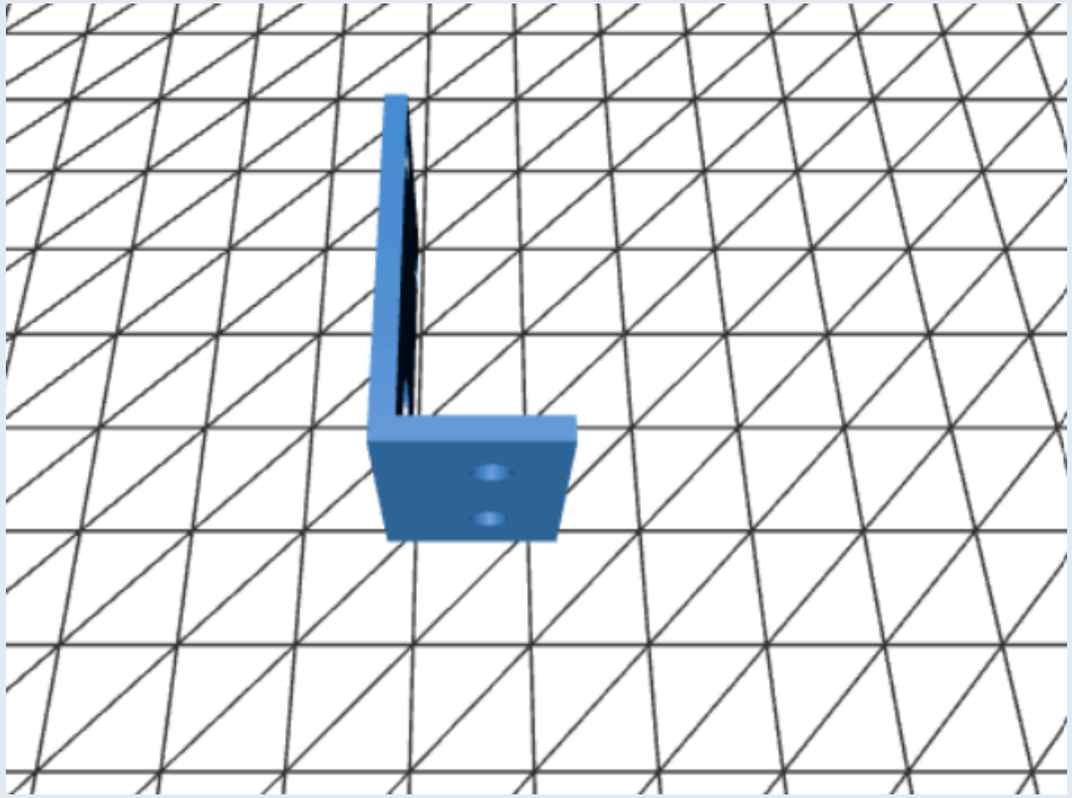
Merge branch 'main' of https://github.com/Md-helaluc
Md-helaluddin committed 11 hours ago



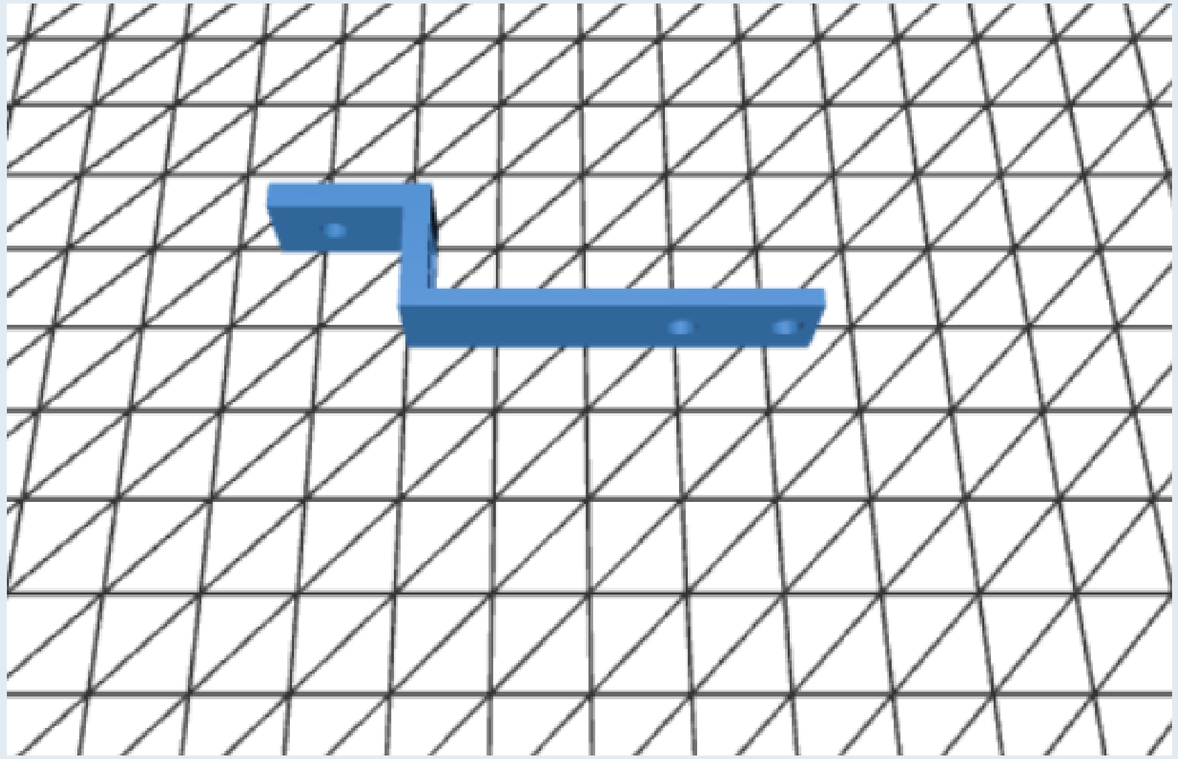


3D DESIGN OF IR AND MOTOR HOLDER:

- Designed custom 3D models for the IR sensor mount and motor holder using SolidWorks
- Ensured precise fitting and stability by aligning dimensions with actual hardware components



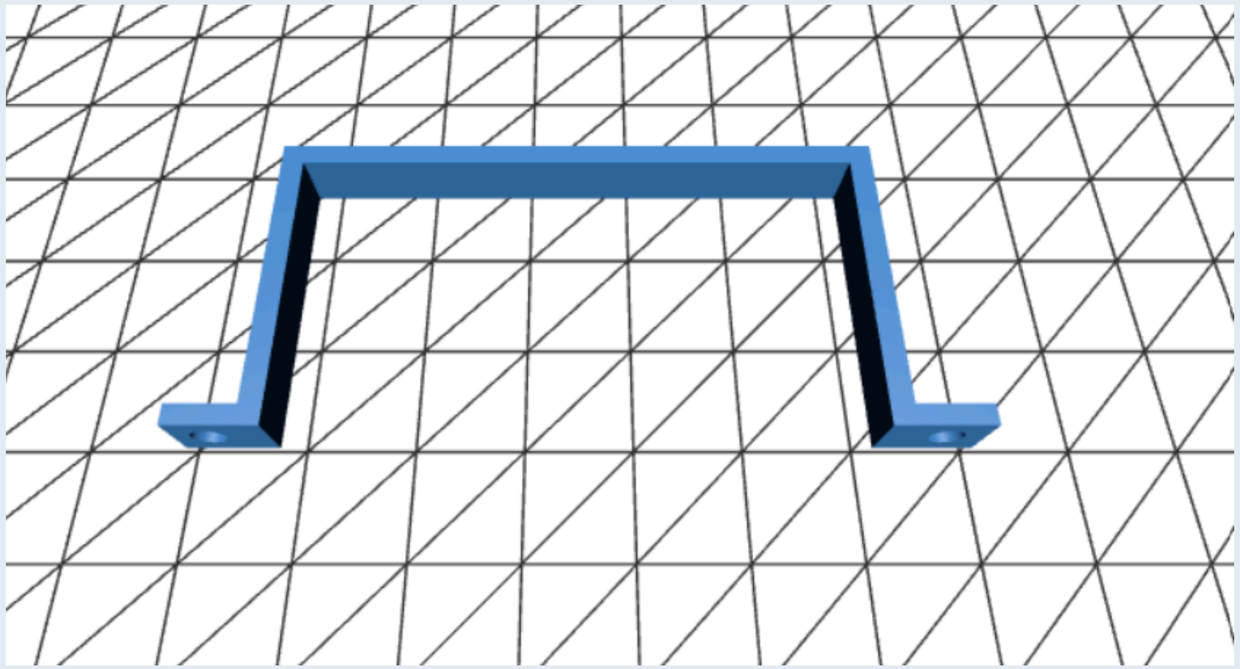
Motor Holder



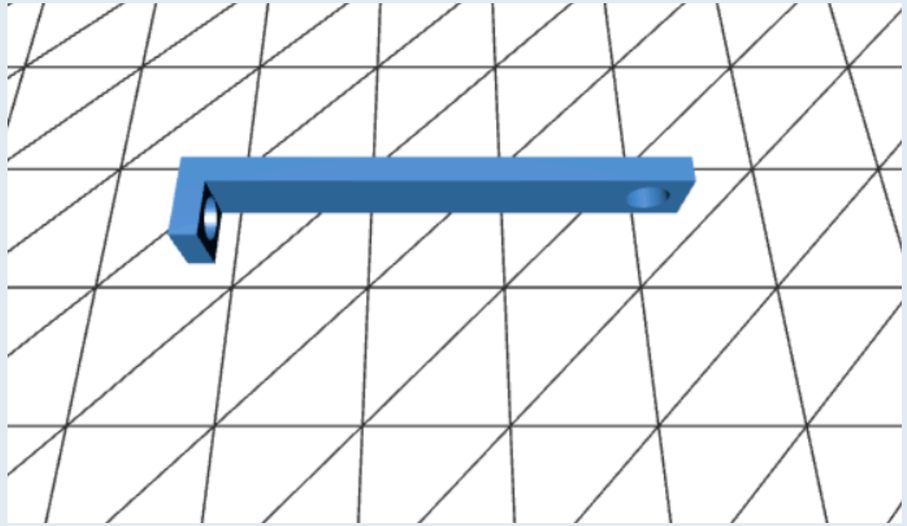
IR Holder

3D DESIGN OF ADDITIONAL COMPONENT AND BREADBOARD HOLDER:

- Created 3D models for a breadboard-battery holder and an additional support bracket for firm motor mounting
- Improved overall hardware stability and wiring management through custom SolidWorks designs



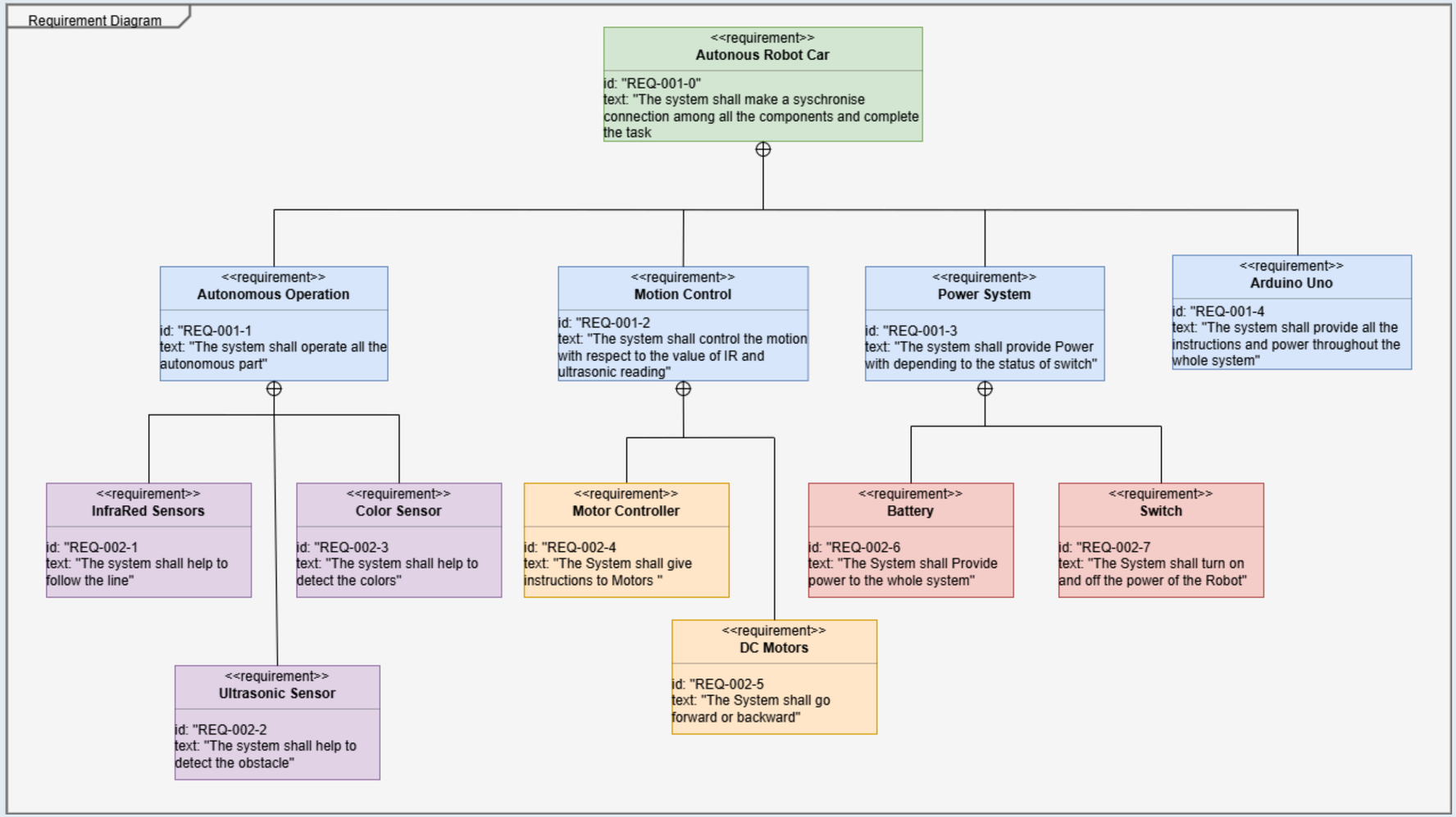
Bread Board and Battery Holder



Additional Motor Holder

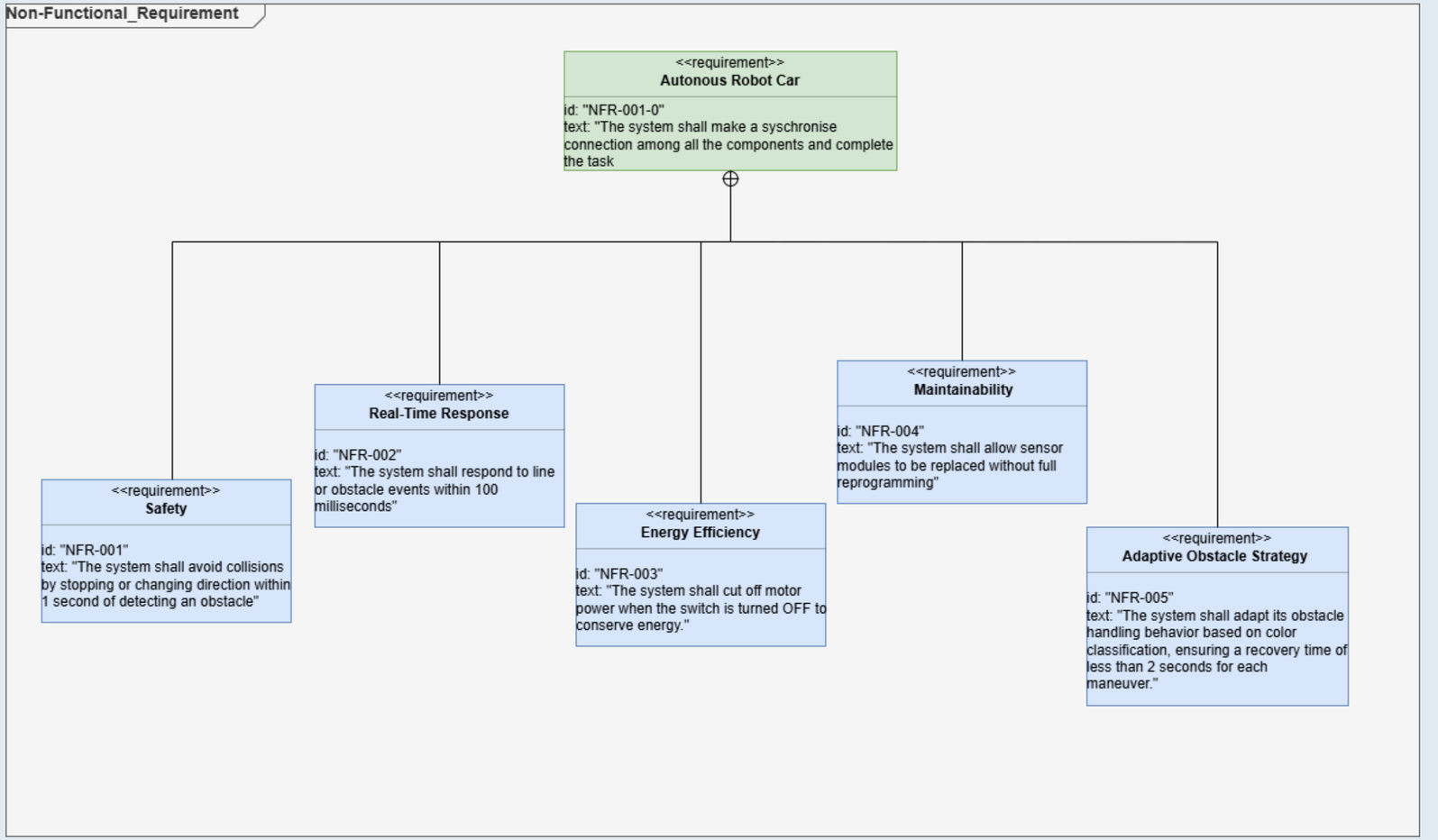
REQUIREMENT DIAGRAM(FUNCTIONAL):

• This diagram has been used to define and organize the system requirements for line following and obstacle avoidance, ensuring all functional goals are clearly specified and traceable.



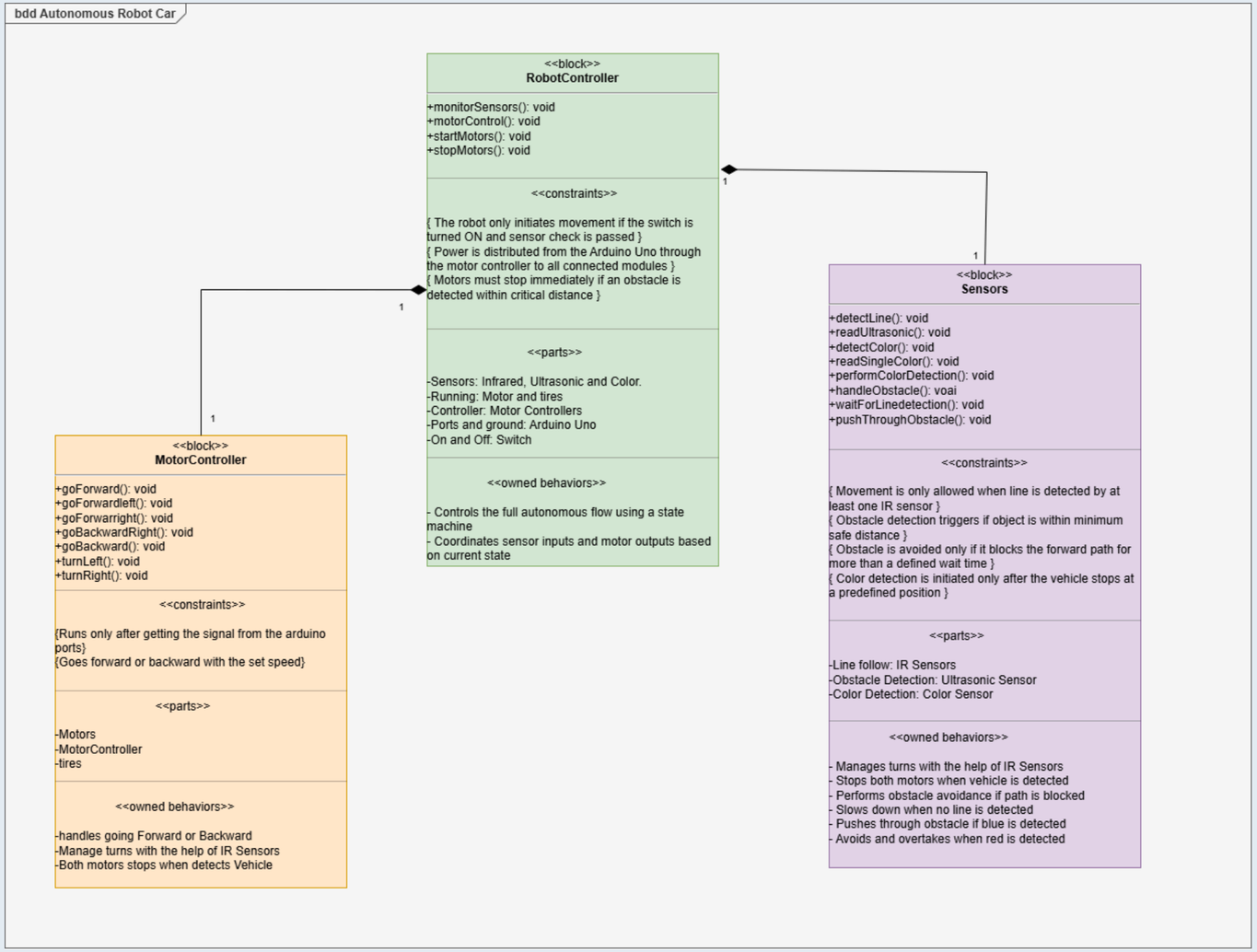
REQUIREMENT DIAGRAM(NON-FUNCTIONAL):

• This diagram has been used to capture non-functional requirements such as performance, reliability, and usability, ensuring the system meets quality expectations.



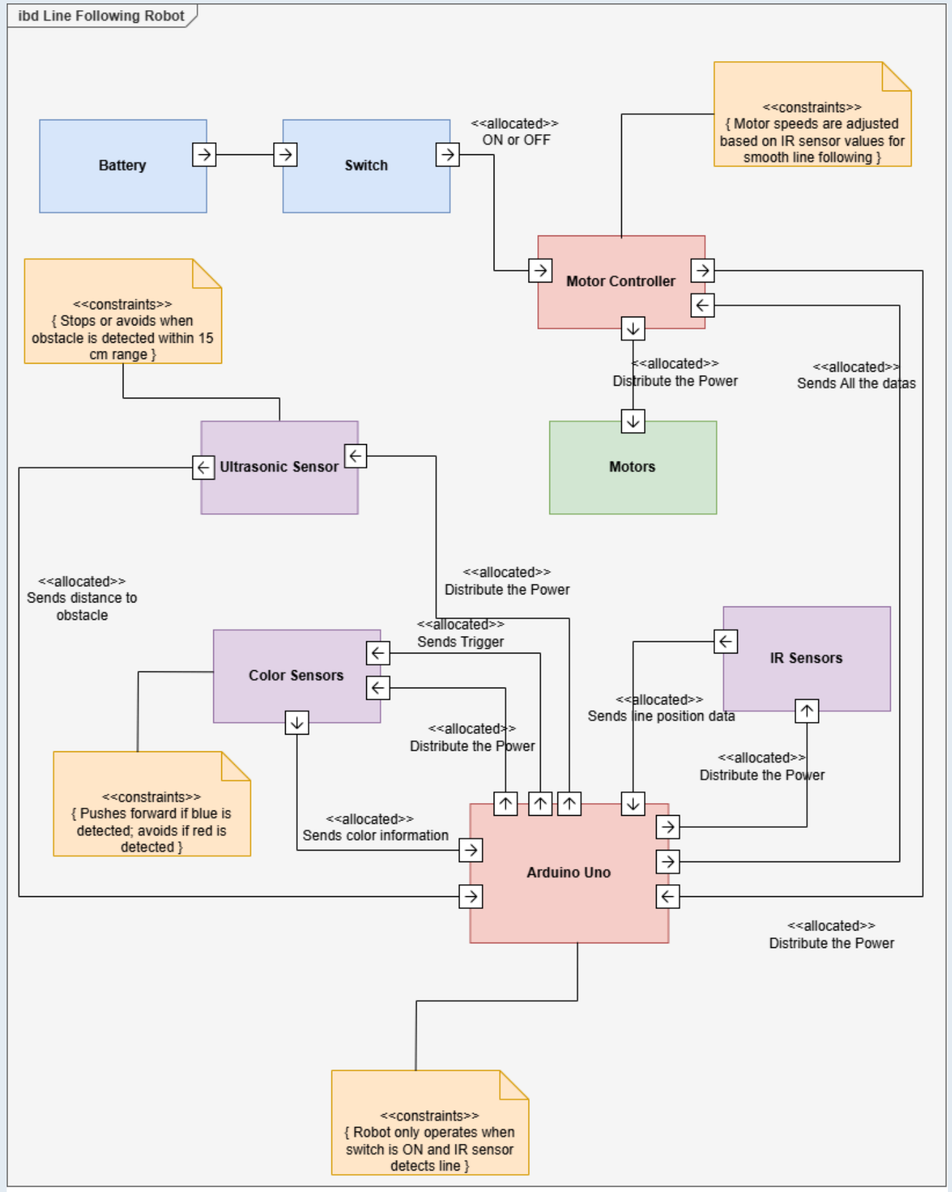
STRUCTURAL DIAGRAM: BLOCK DEFINITION

• This diagram has been used to represent the structural layout of the robot system, including sensors, controller, motor driver, and actuators, along with their relationships.



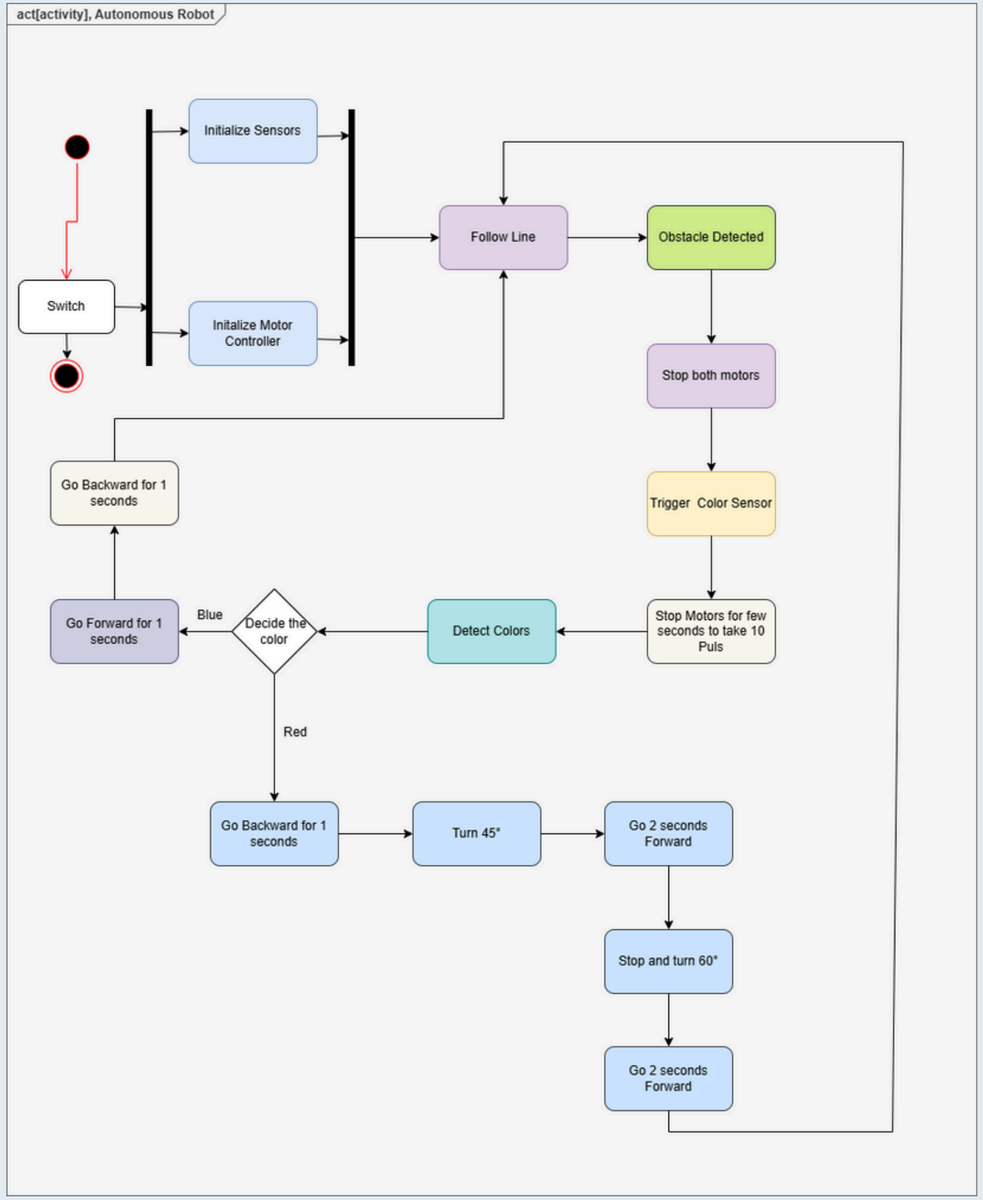
STRUCTURAL DIAGRAM: INTERNAL BLOCK

- Shows internal component interactions
- Includes arduino, sensors, controller, motor driver
- Covers line following, color detection, and obstacle avoidance



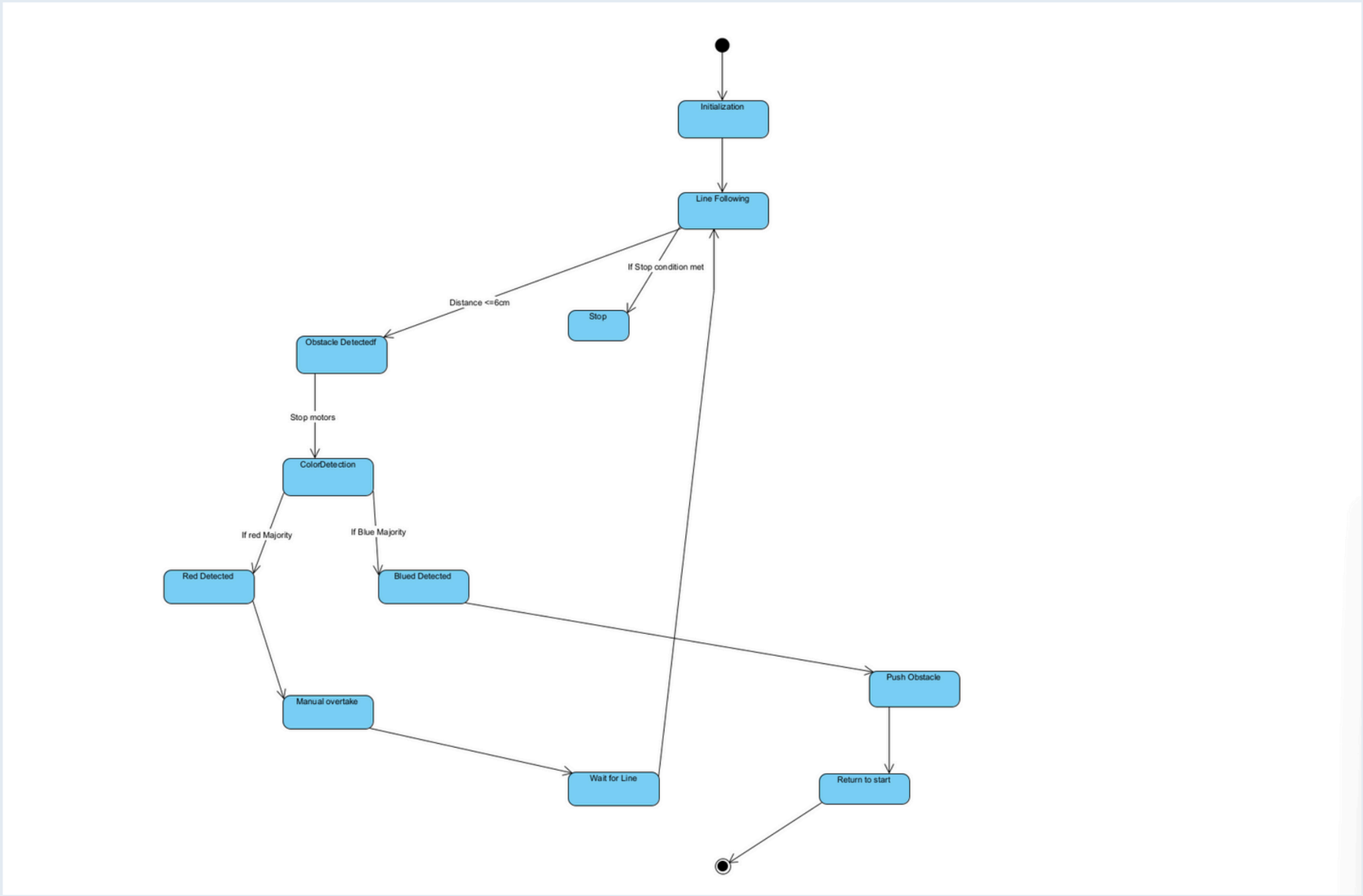
BEHAVIOURAL DIAGRAM: ACTIVITY

- Represents robot's action flow
- Includes line following, obstacle avoidance
- Adds color-based object response



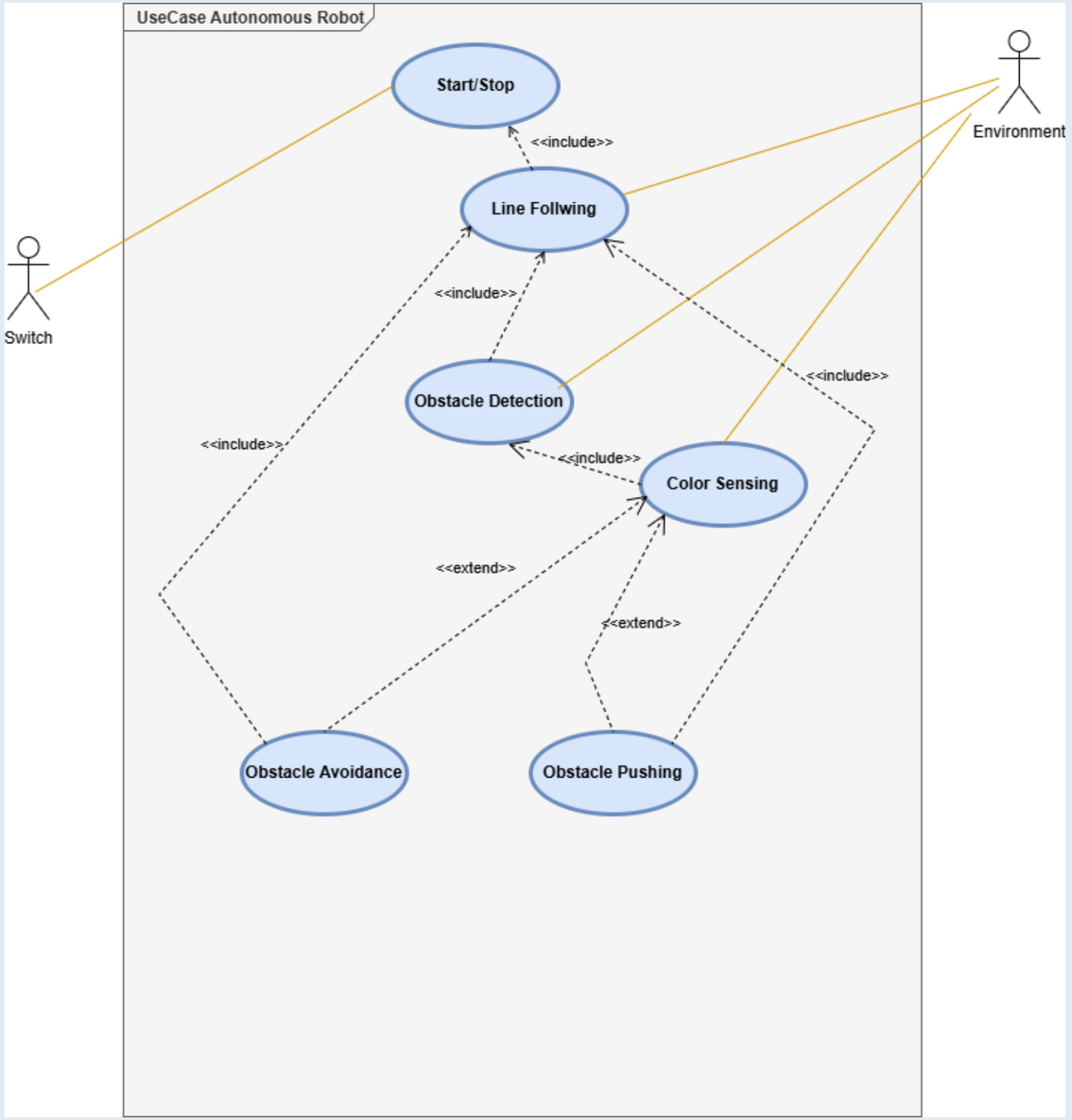
BEHAVIOURAL DIAGRAM: STATE MACHINE

• Represents different operating states of the robot
• Includes line following, obstacle handling, and color response states
• Shows how the system transitions between behaviors based on sensor input



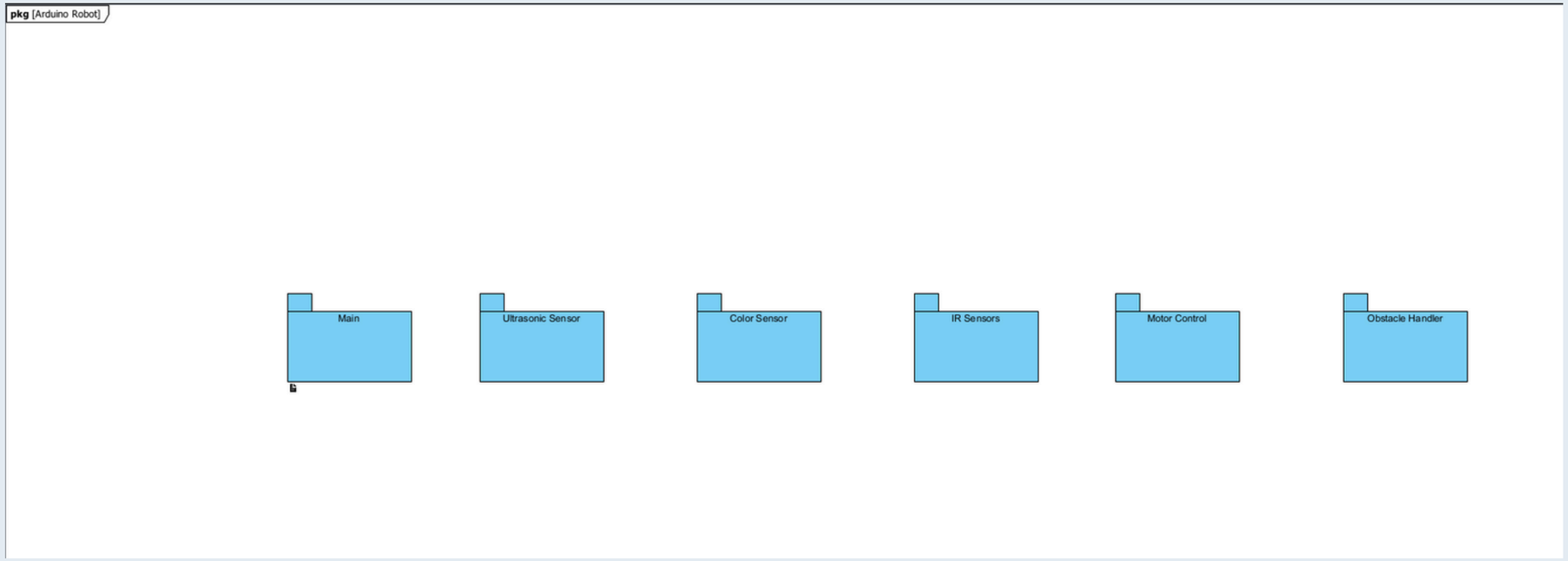
BEHAVIOURAL DIAGRAM: USE CASE

• Highlights the main interactions between the user and the robot
• Covers use cases like starting the robot, detecting obstacles, and reacting to object color
• Defines the system's functional boundaries from the user's perspective



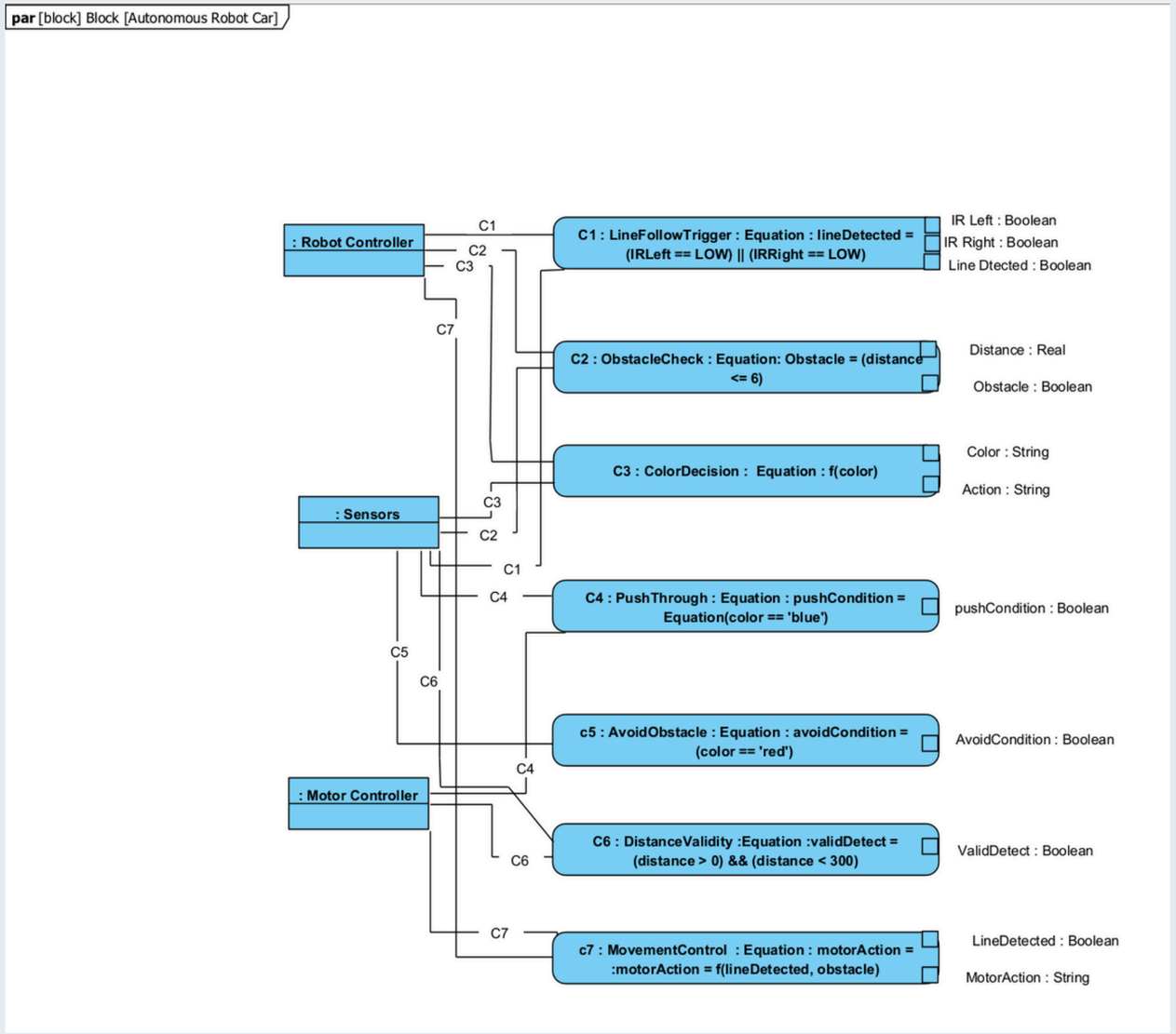
STRUCTURAL DIAGRAM: PACKAGE

• Organizes the system into logical groups or modules
• Separates functionalities like sensing, control logic, and actuation
• Helps manage complexity by structuring related elements together



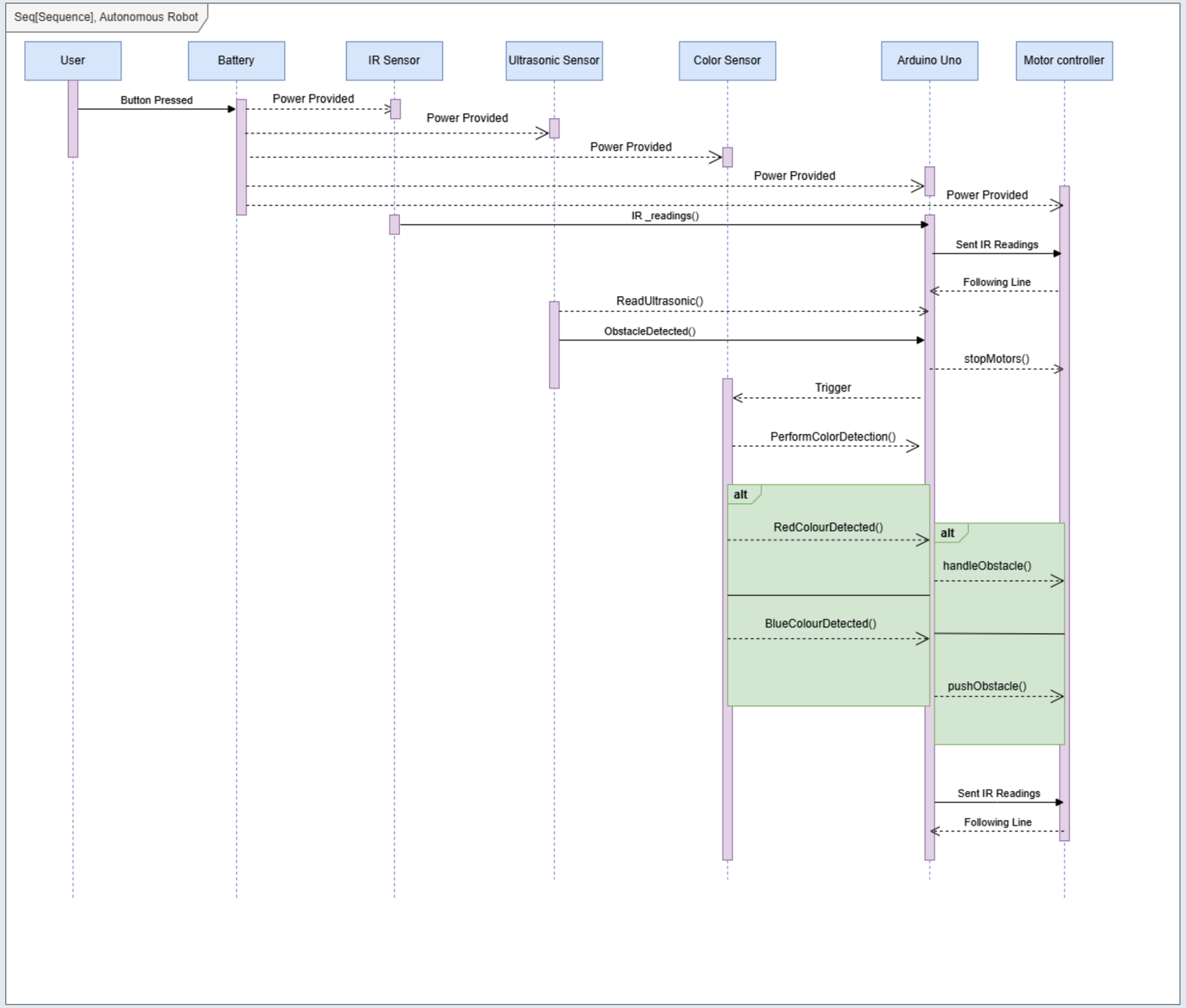
PARAMATRIC DIAGRAM:

- Defines relationships between physical parameters of the system
- Models constraints like speed, distance, and sensor thresholds
- Supports performance analysis and system tuning

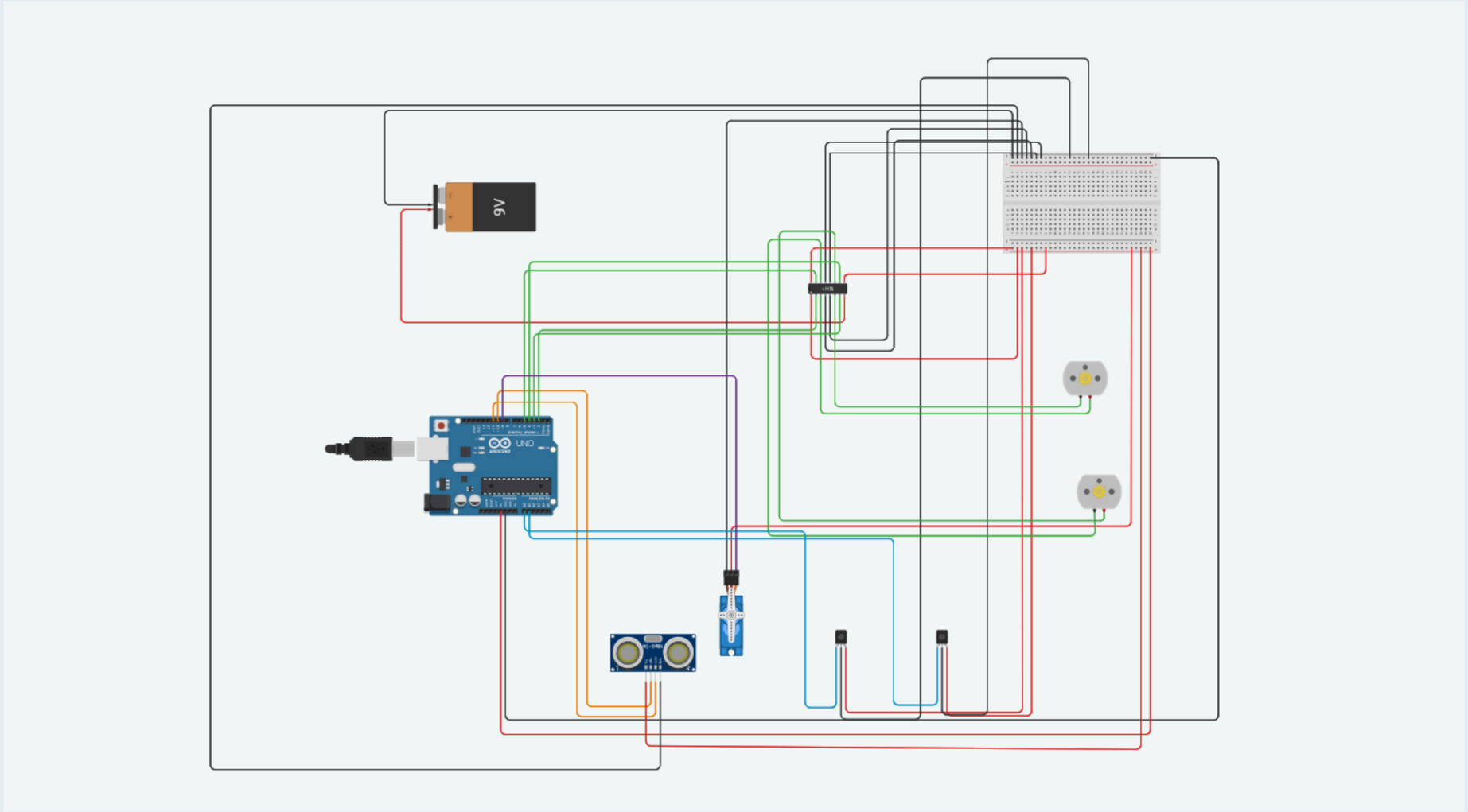


BEHAVIOURAL DIAGRAM: SEQUENCE

- Describes the interaction flow between system components over time
- Shows the sequence of events during line following, obstacle avoidance, and color detection
- Helps visualize timing and message exchange between sensors, controller, and actuators



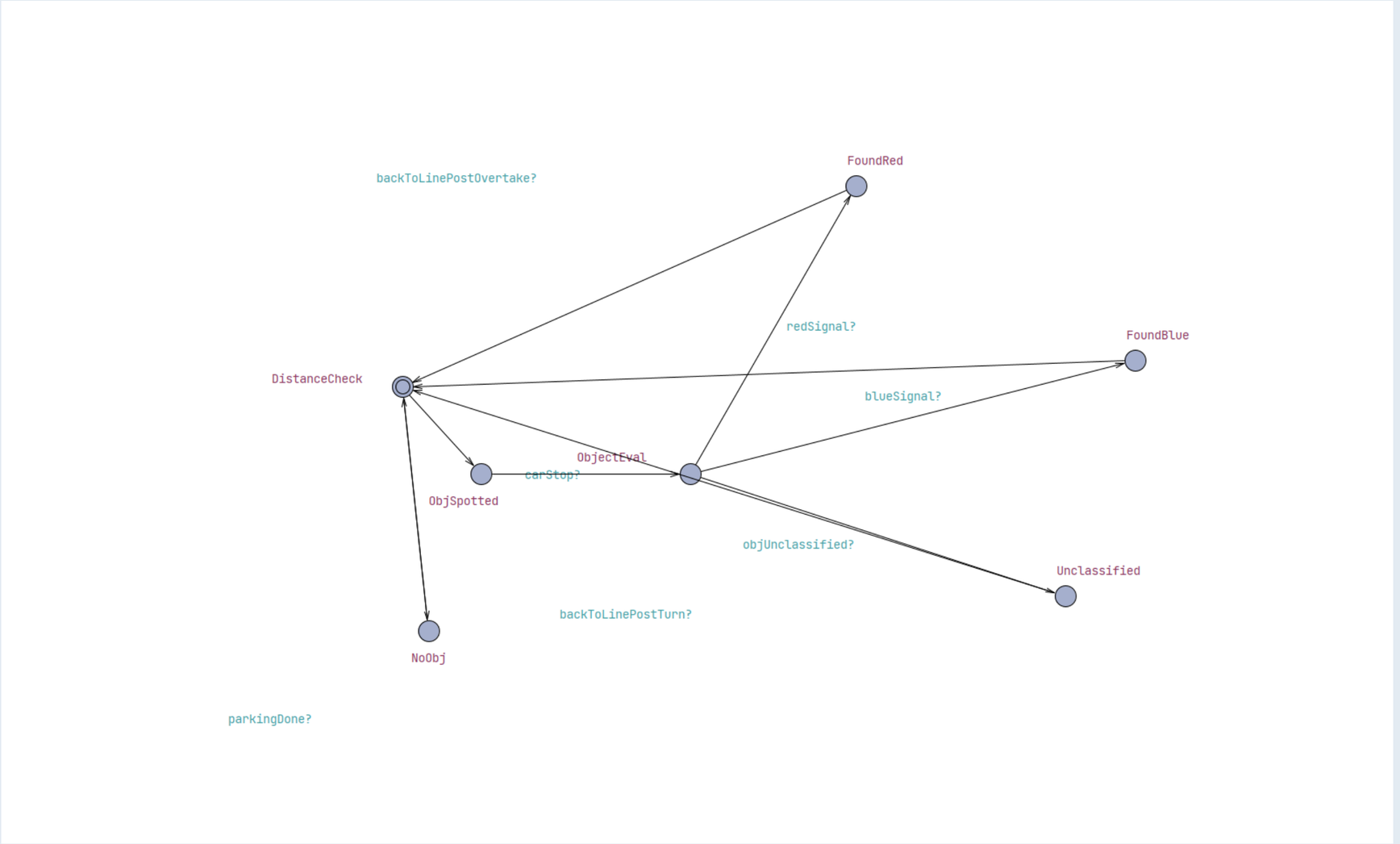
TINKERCAD SIMULATION:



UPPAAL SIMULATION:

• Used to verify the timing and logical behavior of the robot's states

• Simulates transitions between line following, avoiding, and color handling which follows the State Machine Diagram



CODE: LOOP, IR AND ULTRASONIC SENSORS

IR Code

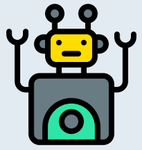
```
}

// Normal line following logic
int left = digitalRead(leftIR);   // HIGH = black line
int right = digitalRead(rightIR);

Serial.print("Left IR: "); Serial.print(left);
Serial.print(" | Right IR: "); Serial.println(right);

if (left == 0 && right == 0) {
  moveForward(speedA, speedB);
}
else if (left == 0 && right == 1) {
  leftForward(80);
  rightBackward(60);
}
else if (left == 1 && right == 0) {
  leftBackward(60);
  rightForward(80);
}
else {
  stopMotors();
}


// === Color Detection Functions ===

String performColorDetection() {
  Serial.println("Starting color detection - taking 10 readings...");
```

```
long readUltrasonic() {
  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);

  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);

  digitalWrite(trigPin, LOW);

  long duration = pulseIn(echoPin, HIGH, 20000);  // Timeout after 20 ms (max ~3.4m)

  if (duration == 0) return -1;  // No echo received
  long distance = duration * 0.034 / 2;

  if (distance > 300 || distance <= 0) return -1;  // Invalid range (>3m or <=0)

  return distance;  // Valid distance in cm
}

void handleObstacle() {
  stopMotors(); delay(300);

  moveBackward(80, 80); delay(500); stopMotors(); delay(300);
  moveWithTurn(-180, 180); delay(180); stopMotors(); delay(300);
  moveForward(65, 65); delay(1600); stopMotors(); delay(300);
  moveWithTurn(200, -200); delay(280); stopMotors(); delay(300);
  moveForward1(80); delay(300);

  waitForLineDetection();
  avoidingObstacle = false;
}
```

Ultrasonic

Loop

```
void loop() {
  distance = readUltrasonic();

  if (distance > 0 && distance <= 6) {
    stopMotors();
    delay(500);

    // Perform color detection when obstacle is detected
    String detectedColor = performColorDetection();
    Serial.print("Final Color Decision: ");
    Serial.println(detectedColor);

    // Handle obstacle based on color
    if (detectedColor == "RED") {
      Serial.println("RED detected - Performing manual overtaking");
      avoidingObstacle = true;
      handleObstacle();  // Your existing manual overtaking function
    }
    /*
    else if (detectedColor == "GREEN") {
      Serial.println("GREEN detected - Pushing through obstacle");
      pushThroughObstacle();
    }
    */
    else if (detectedColor == "BLUE") {
      Serial.println("BLUE detected - Pushing through obstacle");
      pushThroughObstacle();
    }
    else {
      Serial.println("Unknown color - Default overtaking");
```



CODE: CONTROLING MOTORS, FINDING LINE AND PUSHING THE OBSTACLE

Finding line

```
void waitForLineDetection() {
  Serial.println("Searching for line...");
  while (true) {
    int l = digitalRead(leftIR);
    int r = digitalRead(rightIR);

    if (l == 1 || r == 1) {
      Serial.println("Line detected!");
      break;
    } else {
      moveForward1(50);
    }
    delay(10);
  }
}
```

Motor Control

```
// === Original Movement Functions ===

void moveForward(int speedVal1, int speedVal2) {
  analogWrite(enA, speedVal1);
  analogWrite(enB, speedVal2);

  digitalWrite(mr1, LOW);
  digitalWrite(mr2, HIGH);  // Left motor forward (reversed wiring)

  digitalWrite(ml1, HIGH);
  digitalWrite(ml2, LOW);   // Right motor forward
}

void moveBackward(int speedVal1, int speedVal2) {
  analogWrite(enA, speedVal1);
  analogWrite(enB, speedVal2);

  digitalWrite(mr1, HIGH);
  digitalWrite(mr2, LOW);  // Left motor backward (reversed wiring)

  digitalWrite(ml1, LOW);
  digitalWrite(ml2, HIGH);  // Right motor backward
}

void leftForward(int speedVal) {
  analogWrite(enA, speedVal);
  digitalWrite(mr1, LOW);
  digitalWrite(mr2, HIGH);  // Left forward (reversed wiring)
}

void leftBackward(int speedVal) {
  analogWrite(enA, speedVal);
```

Pushing Obstacle

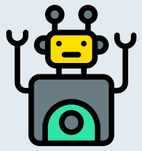
```
void pushThroughObstacle() {
  Serial.println("Pushing blue obstacle forward...");

  // Move forward with full power to push obstacle away
  moveForward(100, 100);
  delay(600);  // Push forward for 2 seconds

  stopMotors();
  delay(300);

  Serial.println("Returning to original position...");

  // Move backward to return to original position
  moveBackward(100, 100);
  delay(600);  // Move back for 2 seconds

  stopMotors();
  delay(300);

  Serial.println("Obstacle cleared! Resuming line following");
}

void stopForever() {
  Serial.println("STOP SIGNAL DETECTED - ROBOT STOPPED PERMANENTLY");
  while (true) {
    stopMotors();
    delay(1000);
    // Robot will stay here forever
  }
}

// === Original Movement Functions ===
```

CODE: PERFORMING COLOR DETECTION

```
133
134   String performColorDetection() {
135     Serial.println("Starting color detection - taking 10 readings...");
136
137     // Arrays to store color counts
138     int redCount = 0;
139     int greenCount = 0;
140     int blueCount = 0;
141     int unknownCount = 0;
142
143     // Take 10 readings with delay between each
144     for (int i = 0; i < 10; i++) {
145       String color = readSingleColor();
146
147       Serial.print("Reading ");
148       Serial.print(i + 1);
149       Serial.print(": ");
150       Serial.println(color);
151
152       // Count each color detection
153       if (color == "RED") {
154         redCount++;
155       }
156       else if (color == "GREEN") {
157         greenCount++;
158       }
159       else if (color == "BLUE") {
160         blueCount++;
161       }
162       else {
163         unknownCount++;
164       }
```

```
163         unknownCount++;
164       }
165
166       delay(200);  // Wait between readings
167     }
168
169     // Print results
170     Serial.print("Results - Red: ");
171     Serial.print(redCount);
172     Serial.print(", Green: ");
173     Serial.print(greenCount);
174     Serial.print(", Blue: ");
175     Serial.print(blueCount);
176     Serial.print(", Unknown: ");
177     Serial.println(unknownCount);
178
179     // Determine final color based on majority vote
180     if (redCount >= greenCount && redCount >= blueCount && redCount >= 3) {
181       return "RED";
182     }
183     else if (greenCount >= redCount && greenCount >= blueCount && greenCount >= 3) {
184       return "GREEN";
185     }
186     else if (blueCount >= redCount && blueCount >= greenCount && blueCount >= 3) {
187       return "BLUE";
188     }
189     else {
190       return "UNKNOWN";
191     }
192   }
193
194   String readSingleColor() {
```

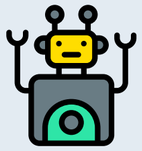
```
194   String readSingleColor() {
195     // Read Red frequency
196     digitalWrite(S2, LOW);
197     digitalWrite(S3, LOW);
198     redFrequency = pulseIn(sensorOut, LOW);
199
200     delay(50);
201
202     // Read Green frequency
203     digitalWrite(S2, HIGH);
204     digitalWrite(S3, HIGH);
205     greenFrequency = pulseIn(sensorOut, LOW);
206
207     delay(50);
208
209     // Read Blue frequency
210     digitalWrite(S2, LOW);
211     digitalWrite(S3, HIGH);
212     blueFrequency = pulseIn(sensorOut, LOW);
213
214     delay(50);
215
216     return detectColor();
217   }
218
219   String detectColor() {
220     // Check if object is actually present (minimum signal strength)
221     int totalSignal = redFrequency + greenFrequency + blueFrequency;
222
223     // If total signal is too high, no object is present (only ambient light)
224     if (totalSignal > 3000) {
225       return "NO OBJECT";
```



CODE: PERFORMING COLOR DETECTION

```
231       }
232
233       // Calculate the difference between colors for better detection
234       int redGreenDiff = abs(redFrequency - greenFrequency);
235       int redBlueDiff = abs(redFrequency - blueFrequency);
236       int greenBlueDiff = abs(greenFrequency - blueFrequency);
237
238       // Different thresholds for different colors
239       int minDifference = 15;
240       int greenMinDiff = 10;
241
242       // Check if all colors are very similar first (white/gray object)
243       if (redGreenDiff < 8 && redBlueDiff < 8 && greenBlueDiff < 8) {
244         return "WHITE/GRAY";
245       }
246
247       // Find the minimum frequency (strongest color signal)
248       int minFreq = min(redFrequency, min(greenFrequency, blueFrequency));
249
250       // Enhanced GREEN detection logic
251       if (greenFrequency == minFreq) {
252         if ((redGreenDiff > greenMinDiff || greenBlueDiff > greenMinDiff)) {
253           if (greenFrequency < (redFrequency * 0.9) || greenFrequency < (blueFrequency * 0.9)) {
254             return "GREEN";
255           }
256         }
257         if (greenFrequency < redFrequency && greenFrequency < blueFrequency) {
258           float greenAdvantage = ((float)(redFrequency + blueFrequency) / 2) / greenFrequency;
259           if (greenAdvantage > 1.05) {
260             return "GREEN";
261           }
262         }
```

```
// RED detection
if (redFrequency == minFreq && redFrequency < greenFrequency && redFrequency < blueFrequency) {
  if (redGreenDiff > minDifference && redBlueDiff > minDifference) {
    if (redFrequency < (greenFrequency * 0.8) && redFrequency < (blueFrequency * 0.8)) {
      return "RED";
    }
  }
}

// BLUE detection
if (blueFrequency == minFreq && blueFrequency < redFrequency && blueFrequency < greenFrequency) {
  if (redBlueDiff > minDifference && greenBlueDiff > minDifference) {
    if (blueFrequency < (redFrequency * 0.8) && blueFrequency < (greenFrequency * 0.8)) {
      return "BLUE";
    }
  }
}

// Only use ratio detection if there's a significant color difference
int maxDiff = max(redGreenDiff, max(redBlueDiff, greenBlueDiff));
if (maxDiff > 20) {
  float redRatio = (float)redFrequency / totalSignal;
  float greenRatio = (float)greenFrequency / totalSignal;
  float blueRatio = (float)blueFrequency / totalSignal;

  if (greenRatio < redRatio && greenRatio < blueRatio) {
    if ((redRatio - greenRatio) > 0.05 && (blueRatio - greenRatio) > 0.05) {
      return "GREEN";
    }
  }
  else if (redRatio < greenRatio && redRatio < blueRatio) {
    if ((greenRatio - redRatio) > 0.05 && (blueRatio - redRatio) > 0.05) {
```



WHAT WE FACED, FIXED, AND FIGURED OUT

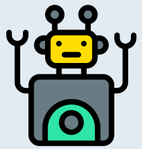
• Simulated the full circuit and logic in Tinkercad, enabling early debugging

• Verified state transitions using UPPAAL, ensuring reliable system behavior

• Faced IR sensor issues—learned sensitivity varies across surfaces

• Used AI to analyze color sensor frequency and identify working thresholds

• Overcame challenges through testing, tuning, and sensor-specific adjustments



THANK YOU VERY MUCH!

Refrences:
Github: https://github.com/Md-helaluddin/D4---Prototyping
Tinkercad: https://www.tinkercad.com/things/hqYCjAoHVhd-frantic-gaaris-curcan