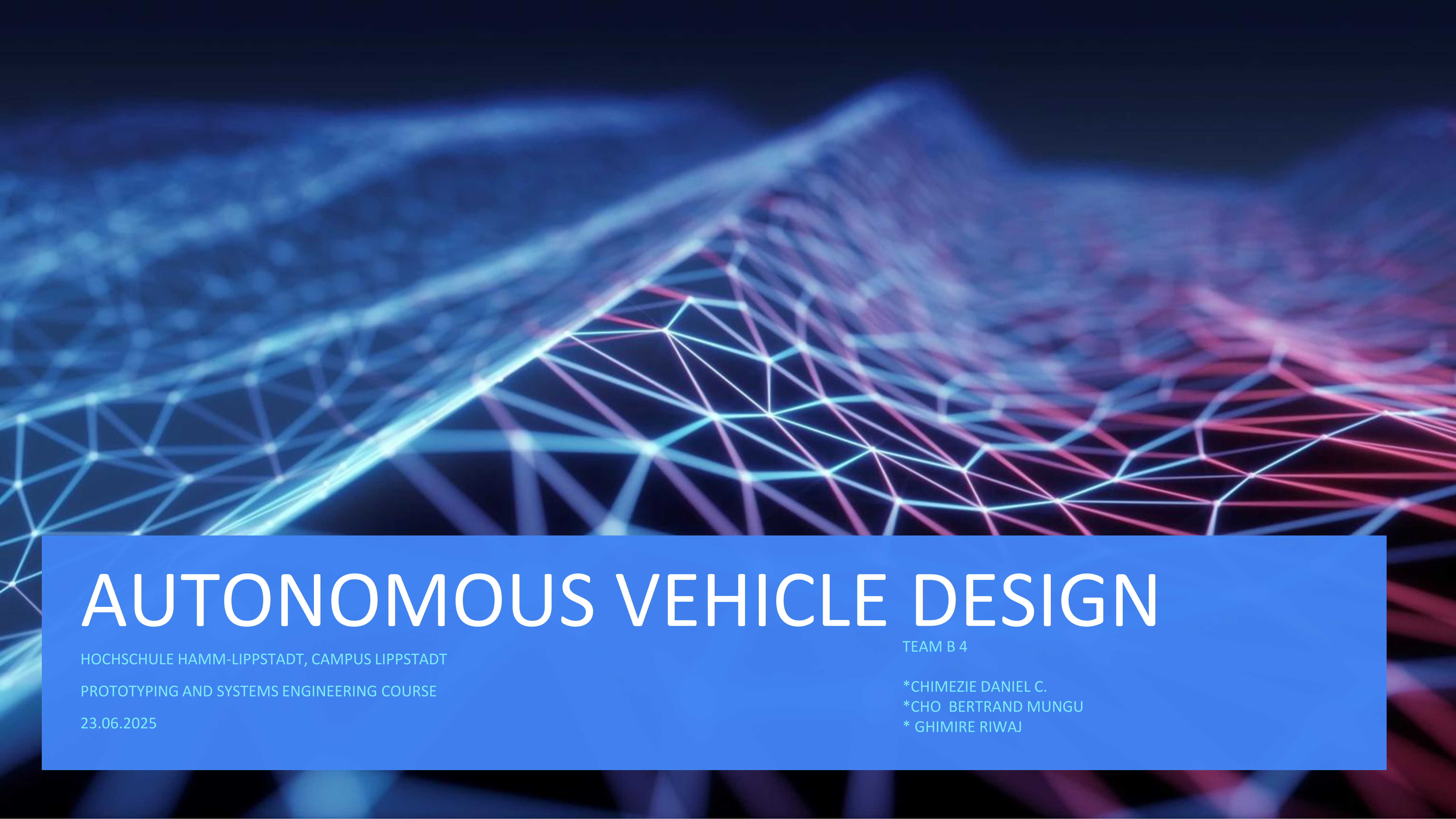


AUTONOMOUS VEHICLE DESIGN

HOCHSCHULE HAMM-LIPPSTADT, CAMPUS LIPPSTADT

PROTOTYPING AND SYSTEMS ENGINEERING COURSE

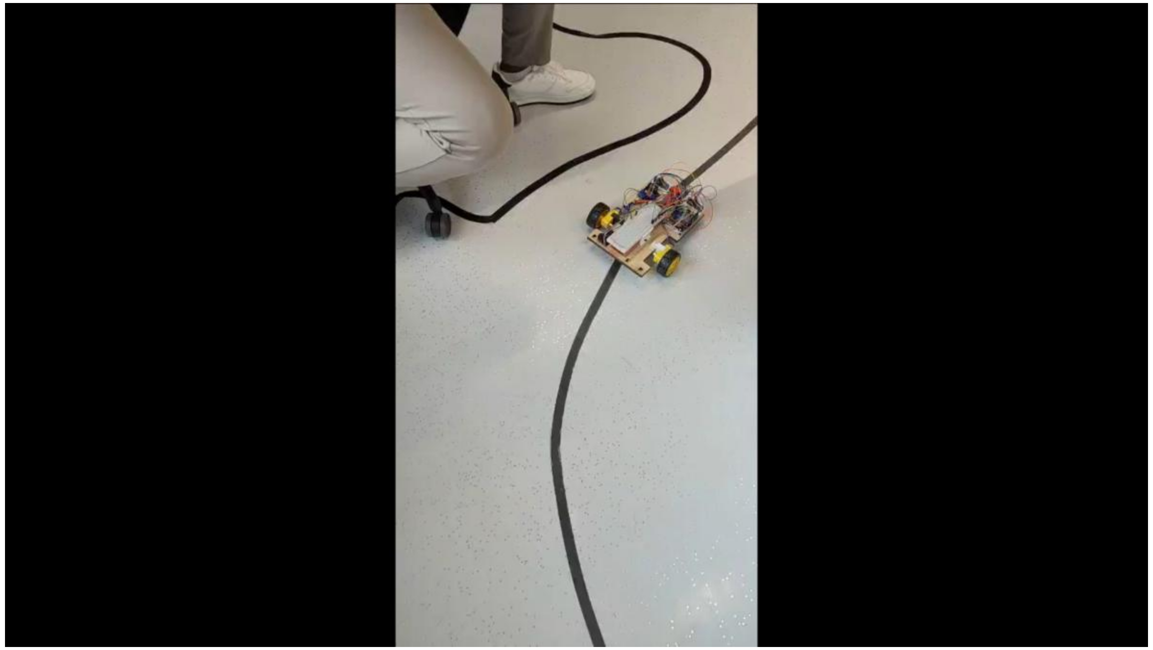
23.06.2025

TEAM B 4

*CHIMEZIE DANIEL C.
*CHO BERTRAND MUNGU
* GHIMIRE RIWAJ

WHAT IS AN AUTONOMOUS VEHICLE ?

According to IEEE standards association: "An autonomous vehicle is a system capable of sensing its environment and operating without human involvement. Such systems are designed to navigate and perform driving functions without direct human input, using onboard sensors, processors, and actuators" [1].

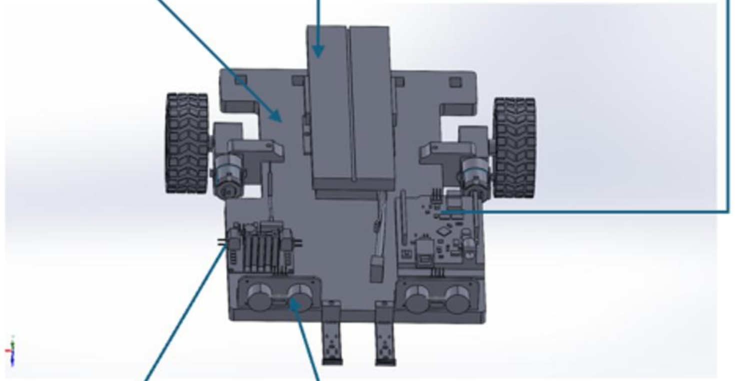


Play the short video above by:

1. Place your mouse above the black image / select the video pan. Then, press alt+p on a German keyboard.

2. Alternatively, highlight the video pan above, you will see a nub ▷ **above** to play the video manually.

SYSTEM OVERVIEW



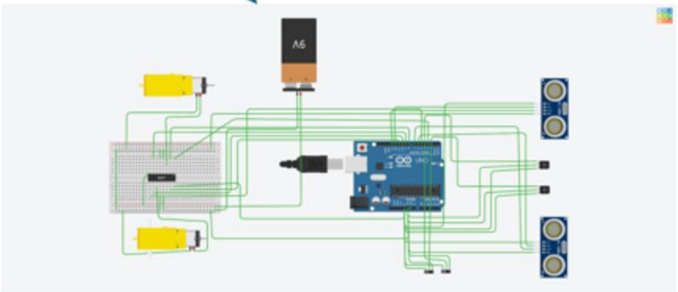
Chassis

Battery and Breadboard Holder

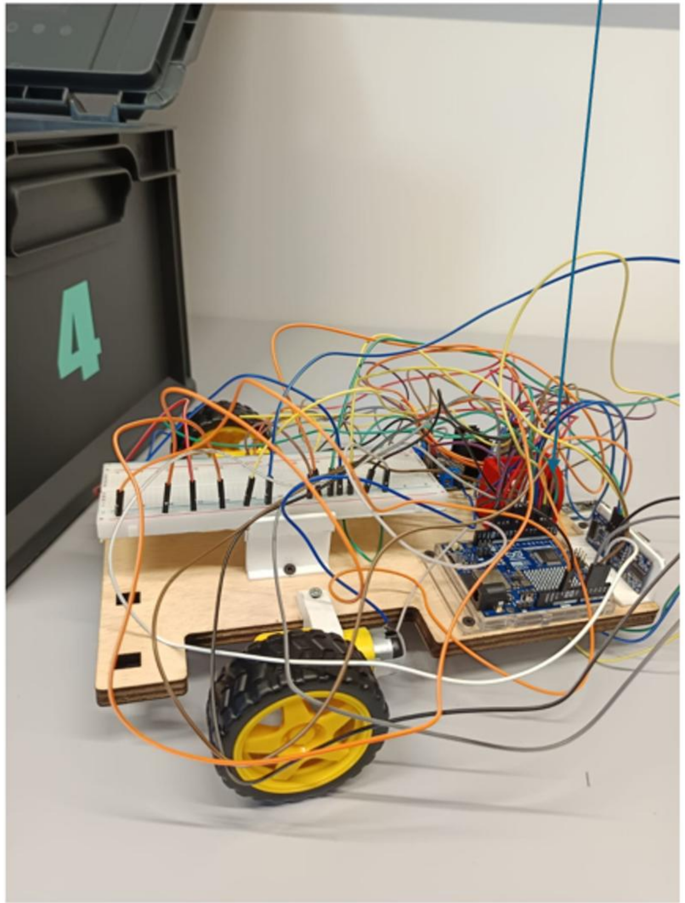
Arduino Uno R4- WiFi Unit

Motor Controller(1 units)

Ultrasonic Sensor (2 units)

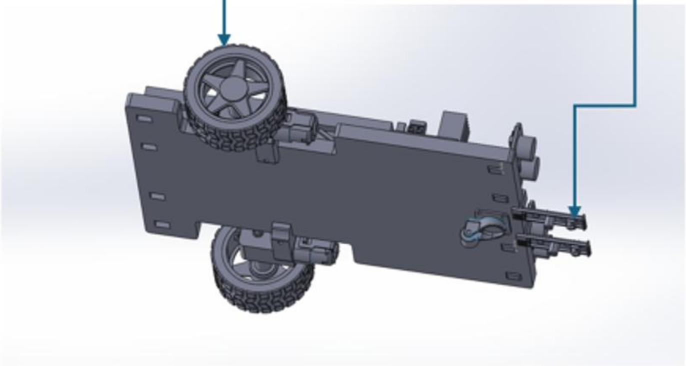


Parts connection layout



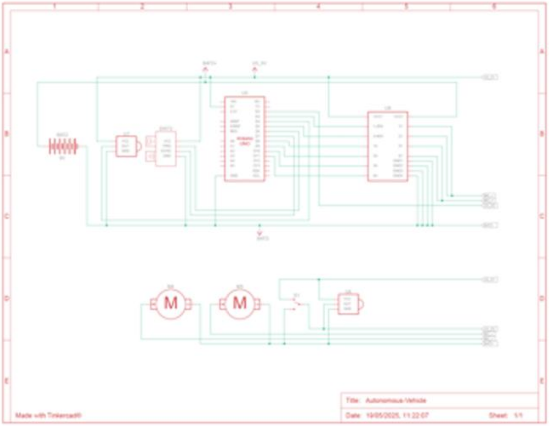
Colour Sensor in red (1 unit)

Physically assembled parts for testing



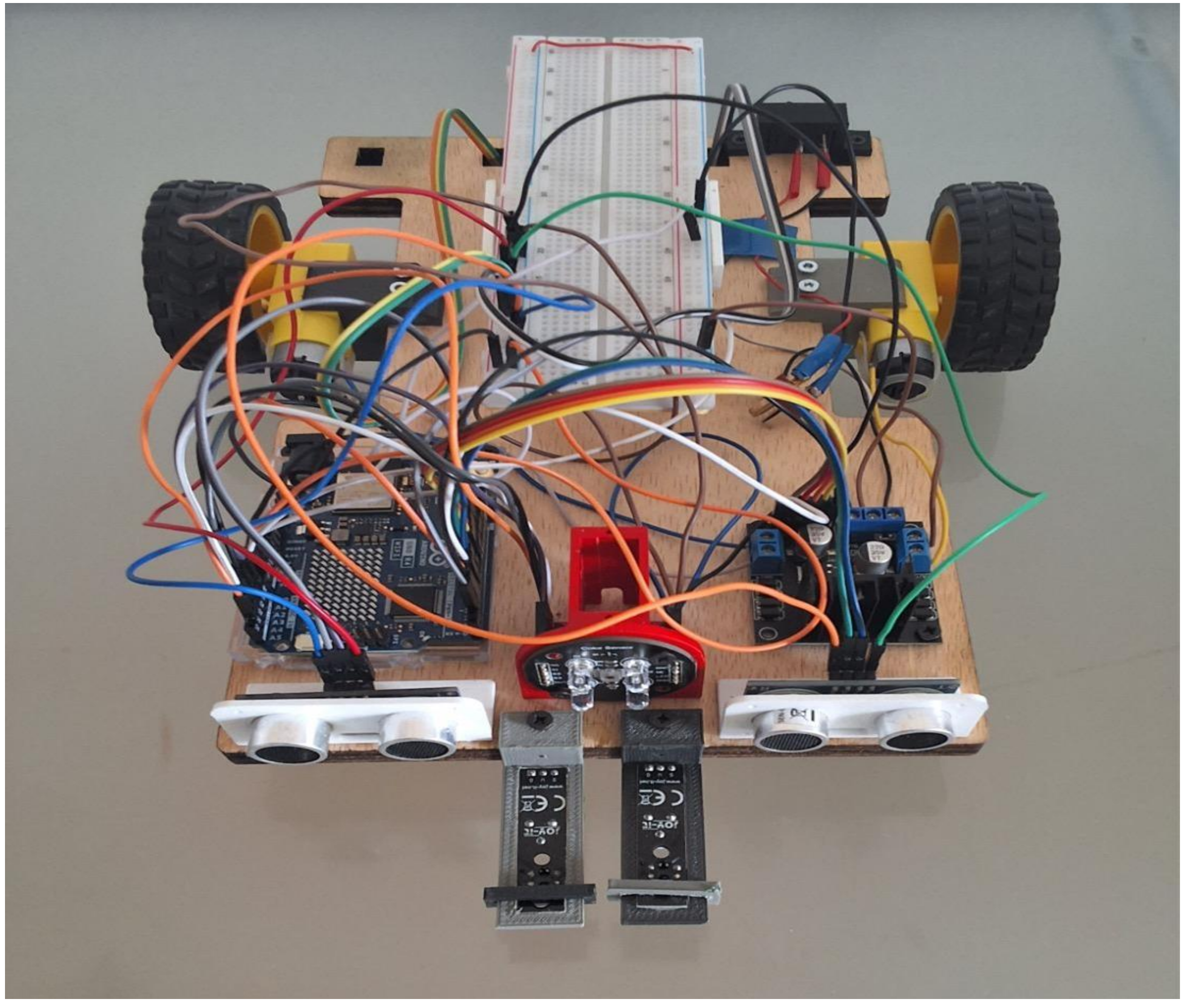
Motor Tyres (2 Units)

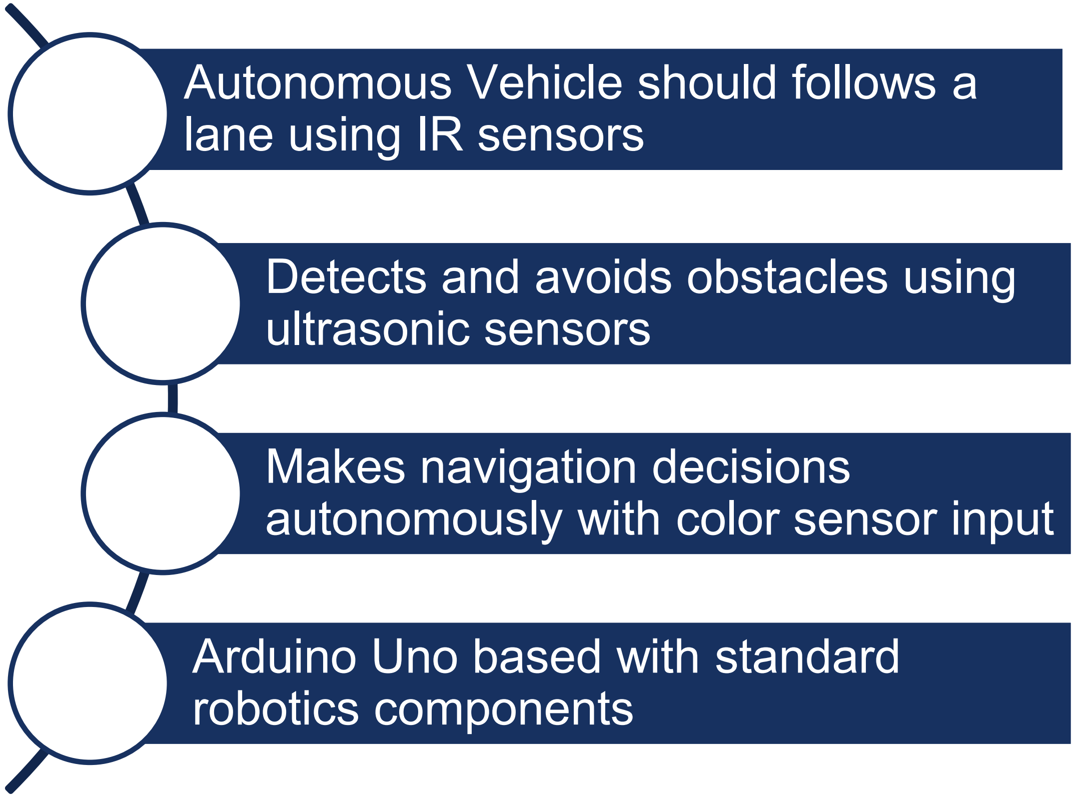
IR Sensor (2 units)



Wiring Plan

PROJECT OBJECTIVE





Autonomous Vehicle should follows a lane using IR sensors

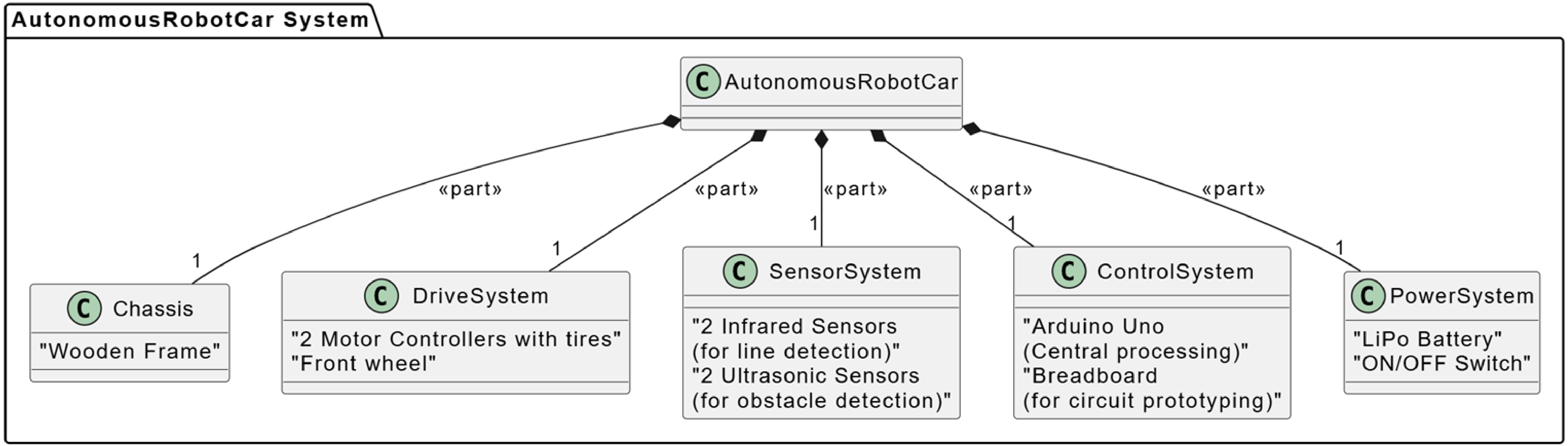
Detects and avoids obstacles using ultrasonic sensors

Makes navigation decisions autonomously with color sensor input

Arduino Uno based with standard robotics components

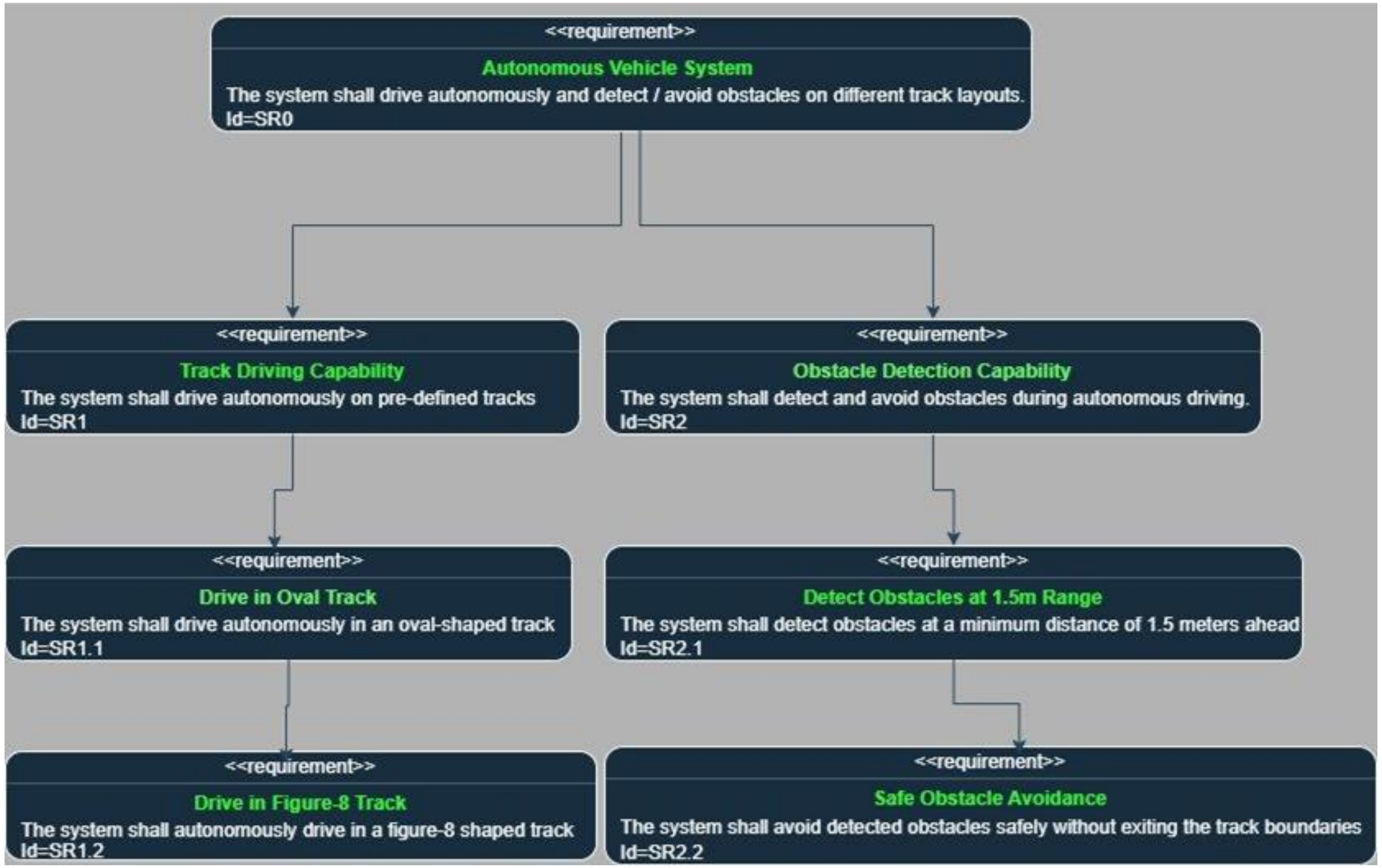
SYSTEMS ENGINEERING DIAGRAMS

1. System Architecture (SysML BDD)



SYSTEMS ENGINEERING DIAGRAMS

2. Requirement diagram



<<requirement>>
Autonomous Vehicle System
The system shall drive autonomously and detect / avoid obstacles on different track layouts.
Id=SR0

<<requirement>>
Track Driving Capability
The system shall drive autonomously on pre-defined tracks
Id=SR1

<<requirement>>
Obstacle Detection Capability
The system shall detect and avoid obstacles during autonomous driving.
Id=SR2

<<requirement>>
Drive in Oval Track
The system shall drive autonomously in an oval-shaped track
Id=SR1.1

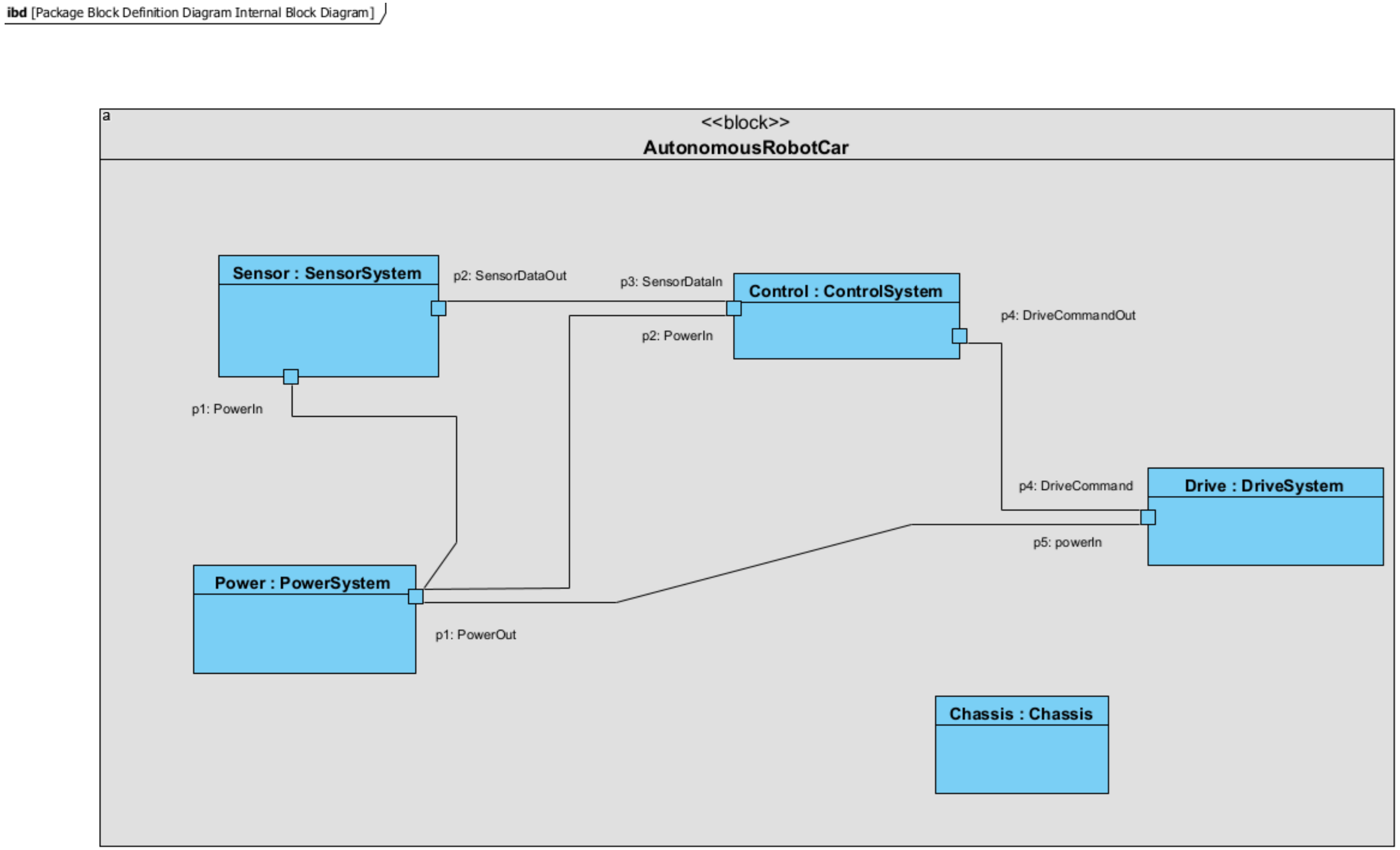
<<requirement>>
Detect Obstacles at 1.5m Range
The system shall detect obstacles at a minimum distance of 1.5 meters ahead
Id=SR2.1

<<requirement>>
Drive in Figure-8 Track
The system shall autonomously drive in a figure-8 shaped track
Id=SR1.2

<<requirement>>
Safe Obstacle Avoidance
The system shall avoid detected obstacles safely without exiting the track boundaries
Id=SR2.2

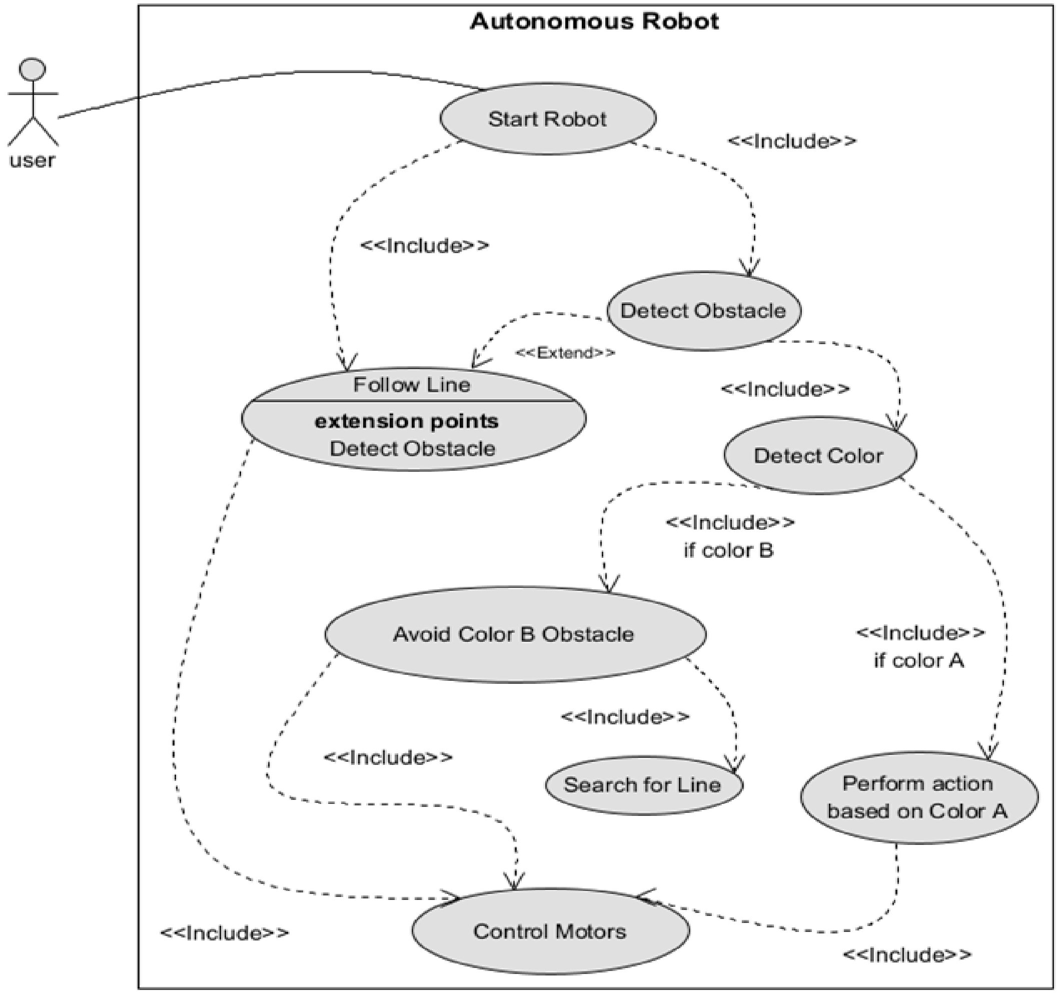
SYSTEMS ENGINEERING DIAGRAMS

3. Internal Block Diagram



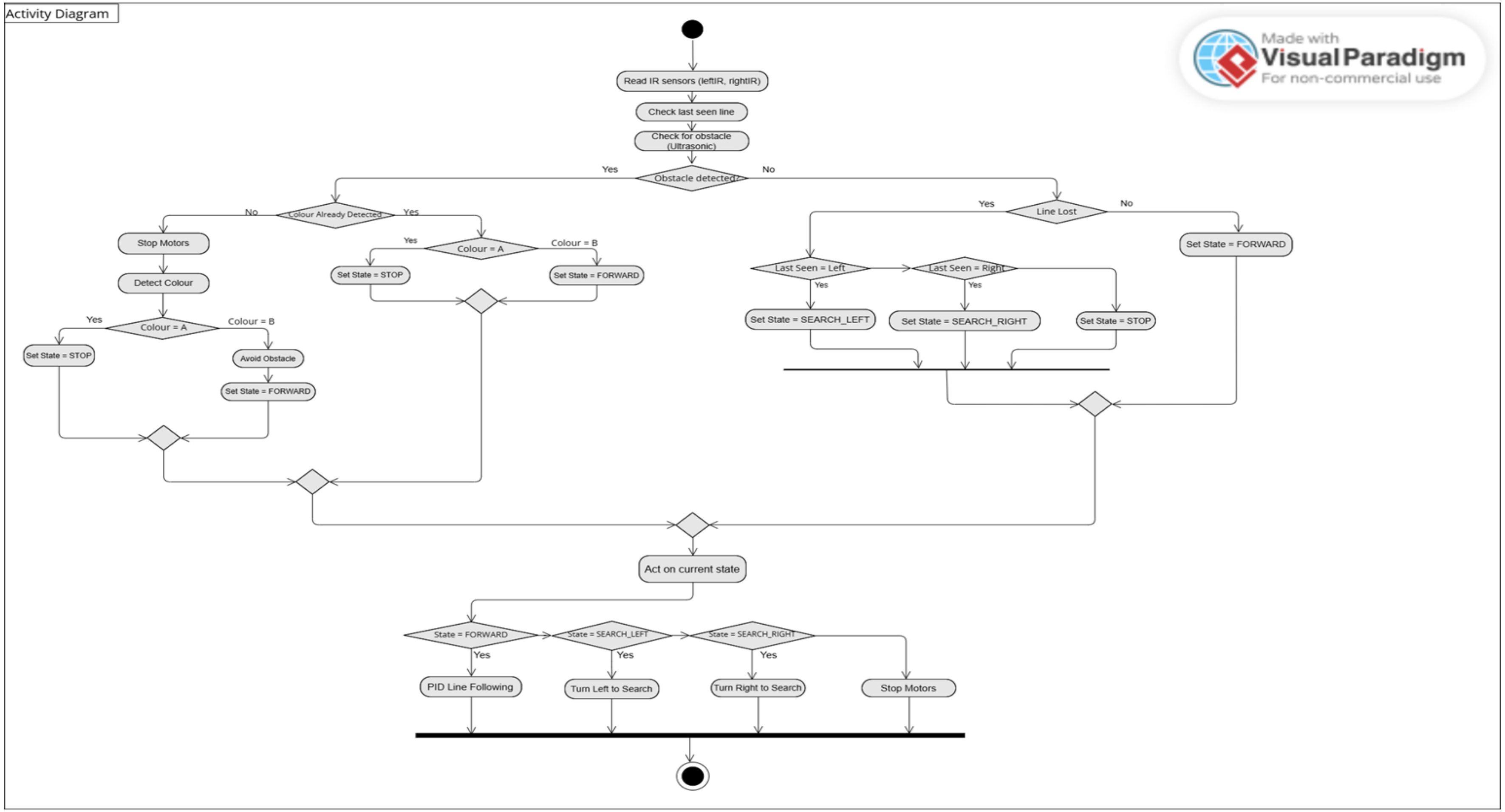
SYSTEMS ENGINEERING DIAGRAMS

4. Use Case Diagram



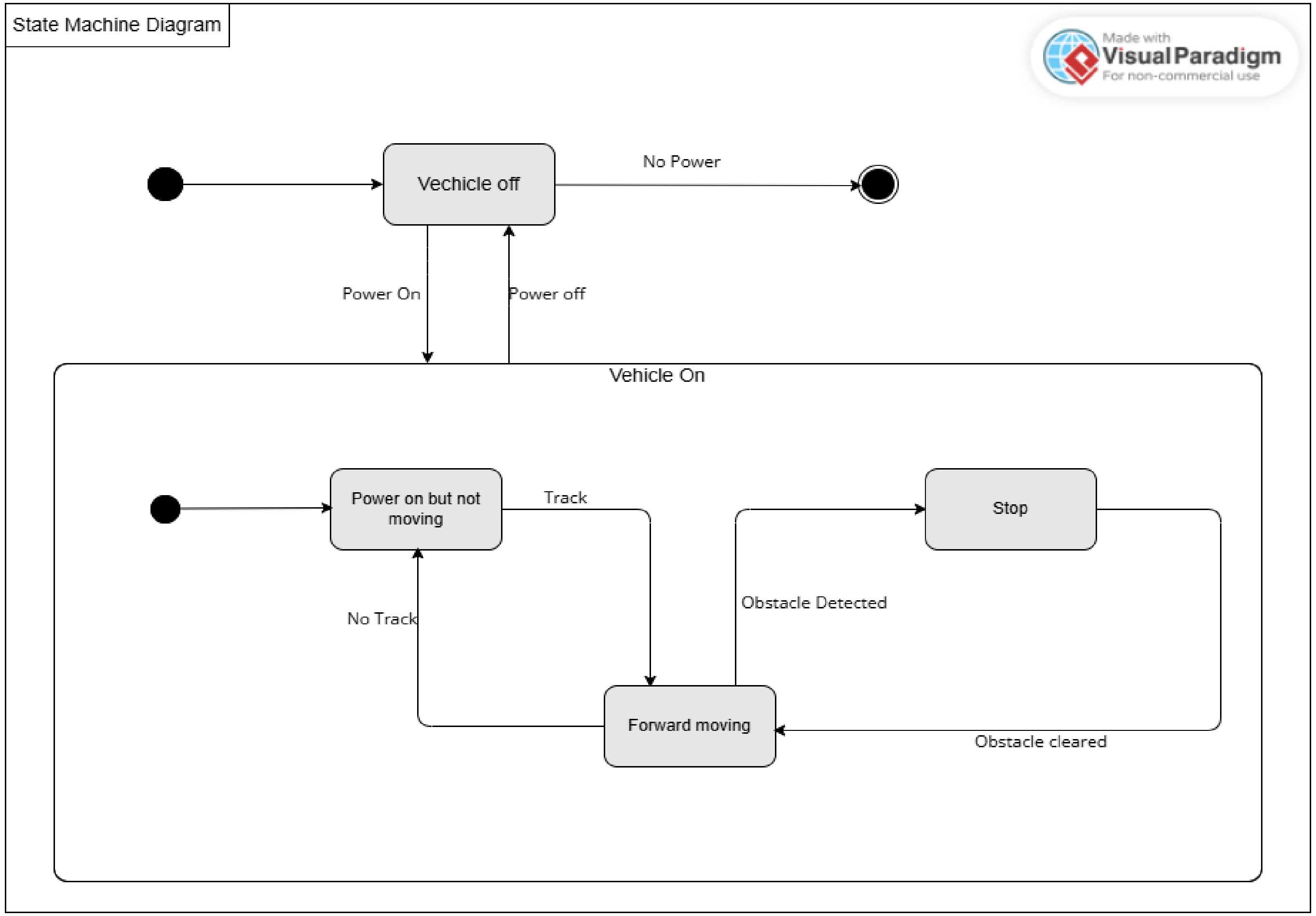
SYSTEMS ENGINEERING DIAGRAMS

5.0 Activity Diagram



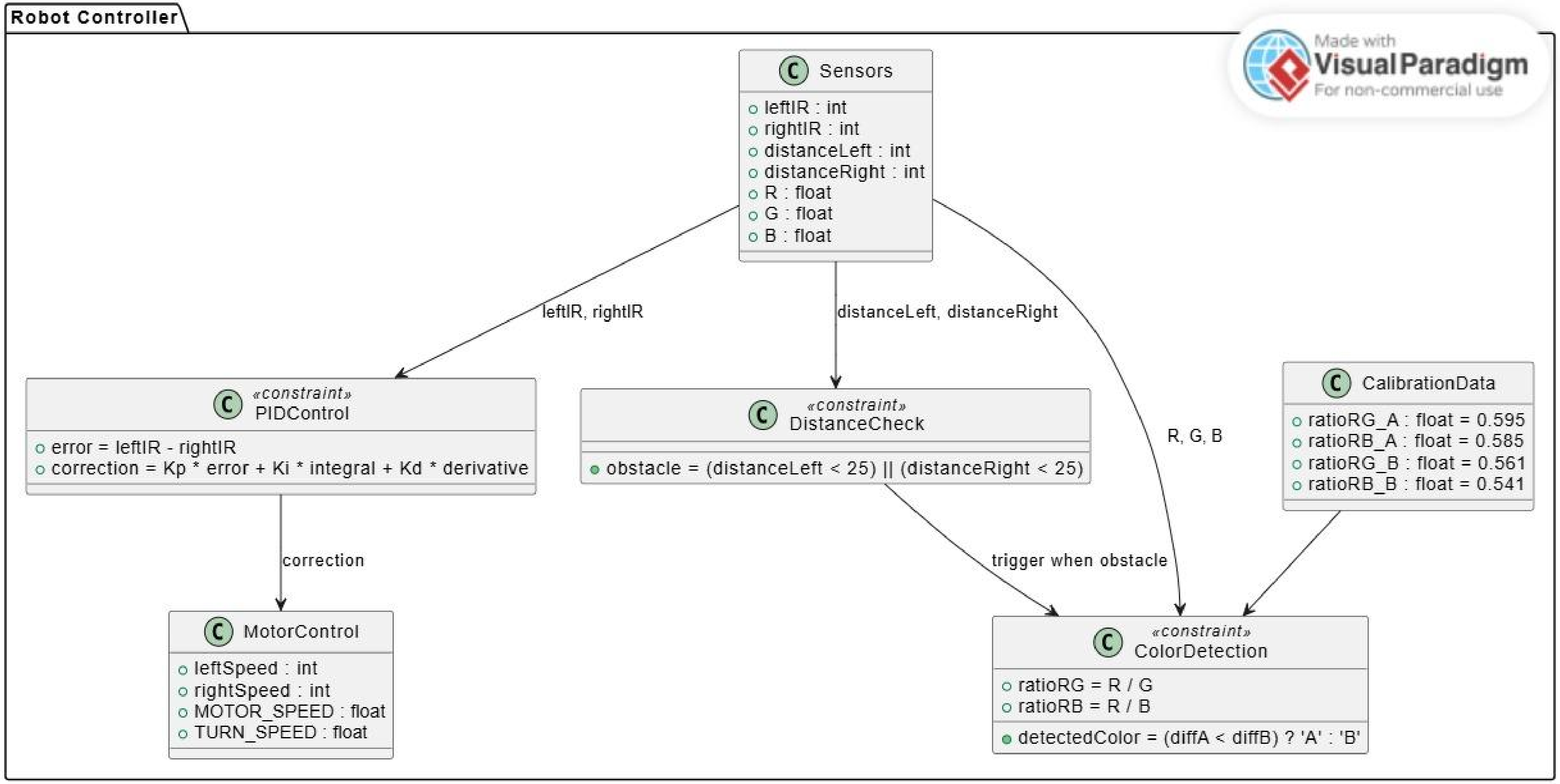
SYSTEMS ENGINEERING DIAGRAMS

6.0 State Machine Diagram



SYSTEMS ENGINEERING DIAGRAMS

7.0 Parametric Diagram



Robot Controller

C Sensors
- ○ leftIR : int
- ○ rightIR : int
- ○ distanceLeft : int
- ○ distanceRight : int
- ○ R : float
- ○ G : float
- ○ B : float

leftIR, rightIR

distanceLeft, distanceRight

R, G, B

C «constraint» PIDControl
- ○ error = leftIR - rightIR
- ○ correction = Kp * error + Ki * integral + Kd * derivative

C «constraint» DistanceCheck
- ● obstacle = (distanceLeft < 25) || (distanceRight < 25)

C CalibrationData
- ○ ratioRG_A : float = 0.595
- ○ ratioRB_A : float = 0.585
- ○ ratioRG_B : float = 0.561
- ○ ratioRB_B : float = 0.541

correction

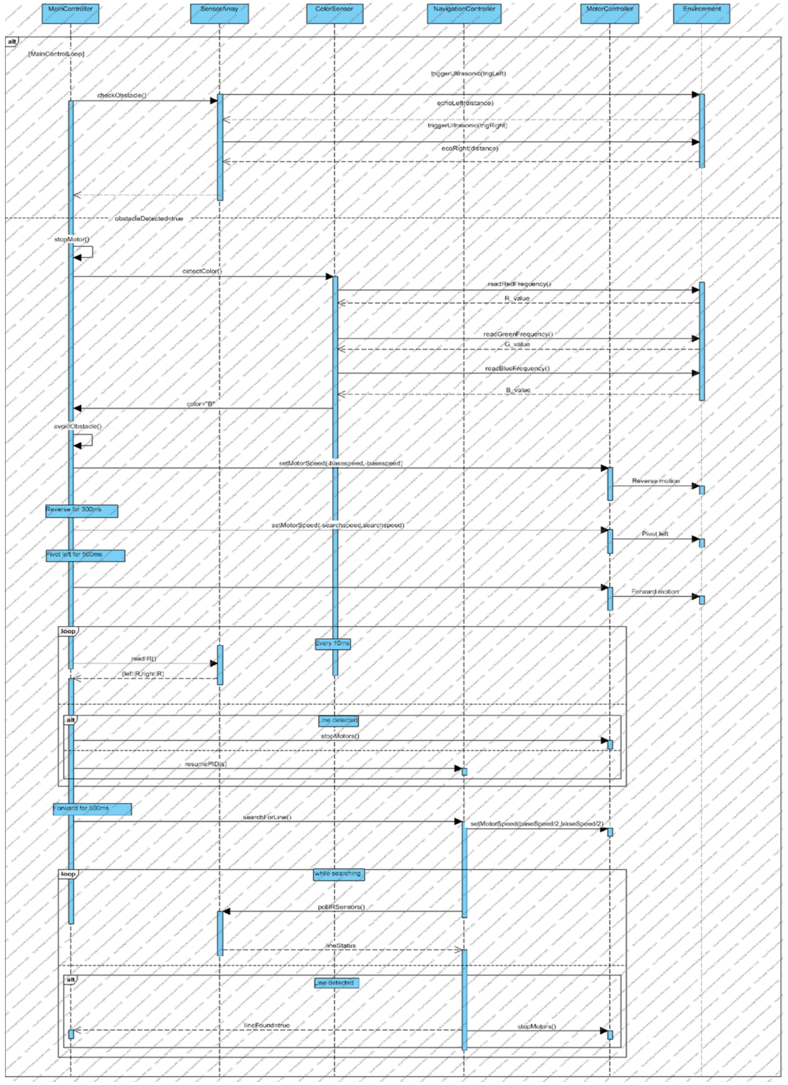
trigger when obstacle

C MotorControl
- ○ leftSpeed : int
- ○ rightSpeed : int
- ○ MOTOR_SPEED : float
- ○ TURN_SPEED : float

C «constraint» ColorDetection
- ○ ratioRG = R / G
- ○ ratioRB = R / B
- ● detectedColor = (diffA < diffB) ? 'A' : 'B'

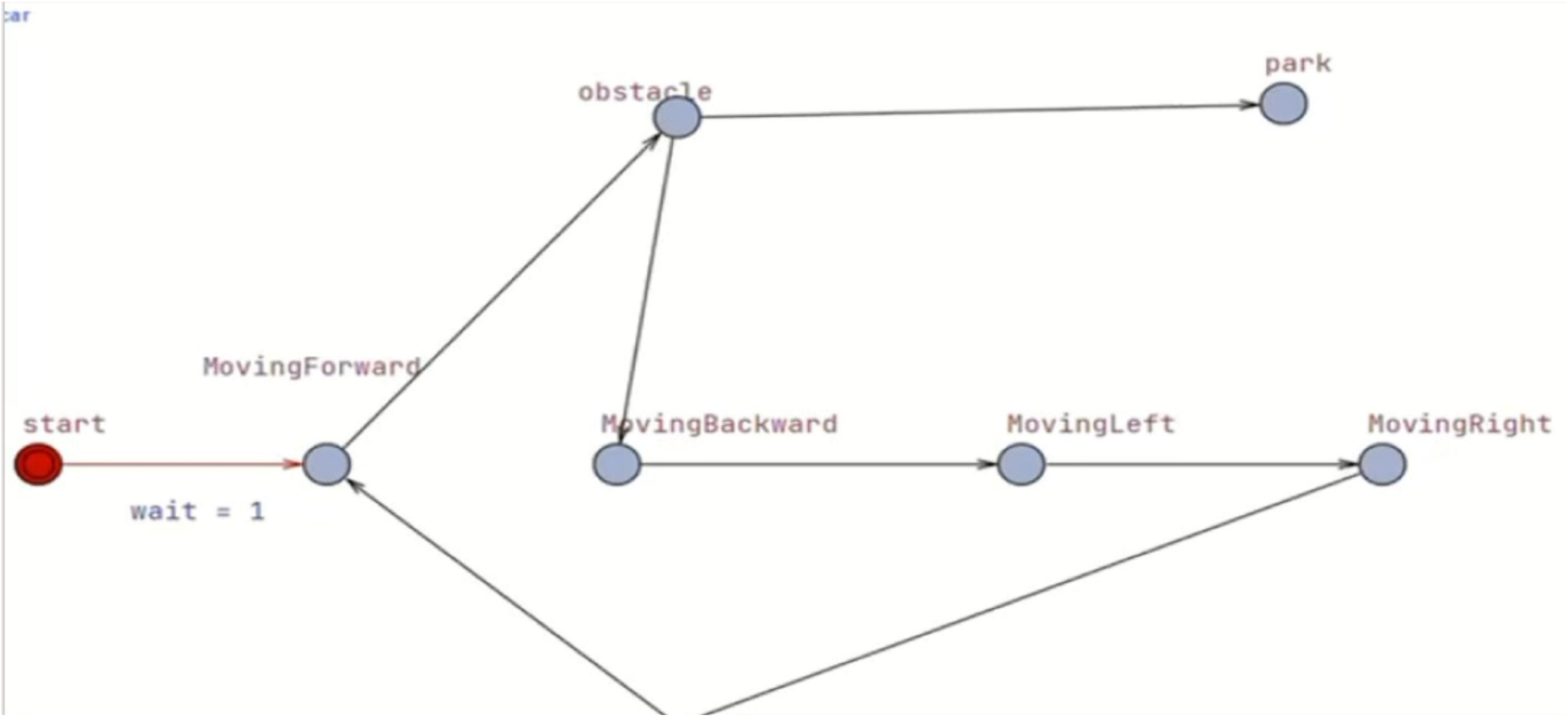
SYSTEMS ENGINEERING DIAGRAMS

8.0 Sequence Diagram

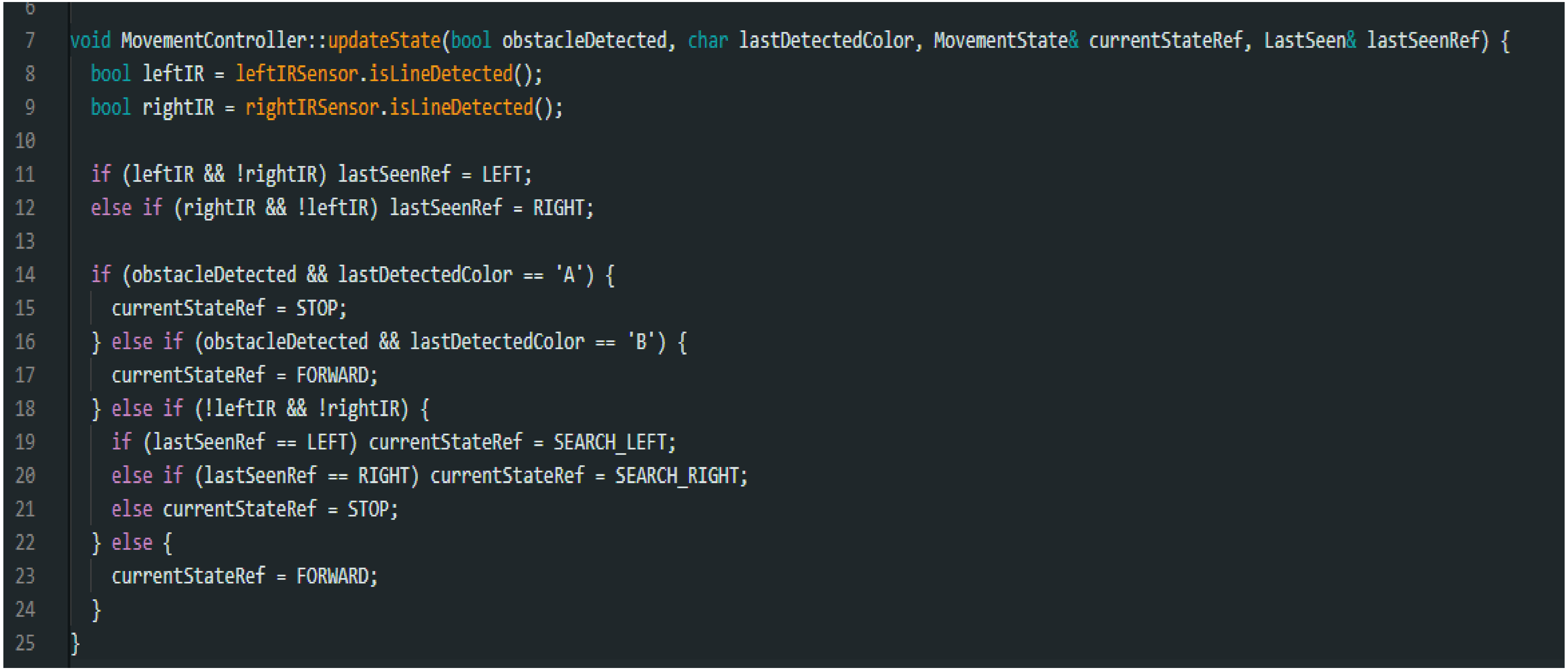


CODING, TESTING AND IMPLEMENTATION

UPAAL ANALYSIS

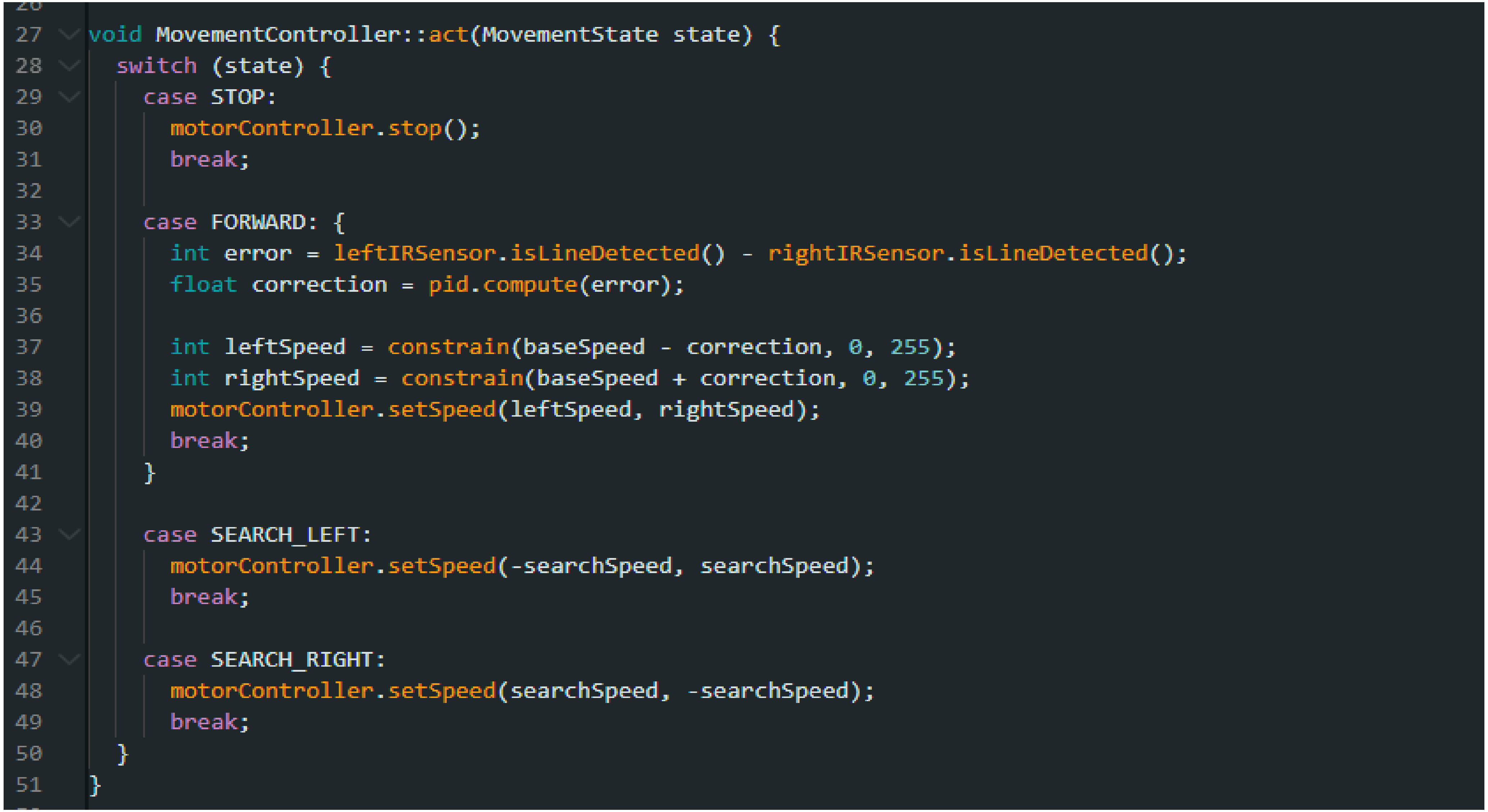


Implementation Snapshot-Movement Logic



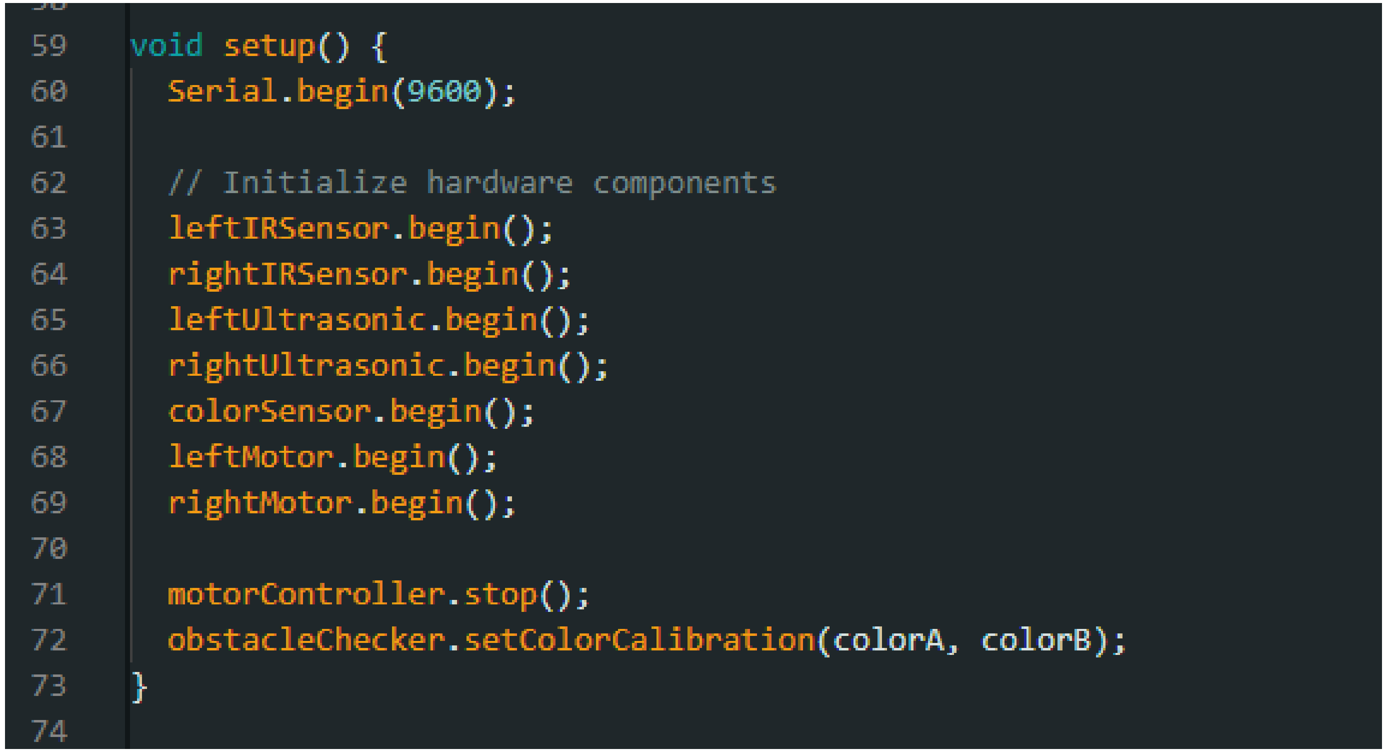
```cpp
void MovementController::updateState(bool obstacleDetected, char lastDetectedColor, MovementState& currentStateRef, LastSeen& lastSeenRef) {
  bool leftIR = leftIRSensor.isLineDetected();
  bool rightIR = rightIRSensor.isLineDetected();

  if (leftIR && !rightIR) lastSeenRef = LEFT;
  else if (rightIR && !leftIR) lastSeenRef = RIGHT;

  if (obstacleDetected && lastDetectedColor == 'A') {
    currentStateRef = STOP;
  } else if (obstacleDetected && lastDetectedColor == 'B') {
    currentStateRef = FORWARD;
  } else if (!leftIR && !rightIR) {
    if (lastSeenRef == LEFT) currentStateRef = SEARCH_LEFT;
    else if (lastSeenRef == RIGHT) currentStateRef = SEARCH_RIGHT;
    else currentStateRef = STOP;
  } else {
    currentStateRef = FORWARD;
  }
}
```

Implementation Snapshot- Act on Movement State

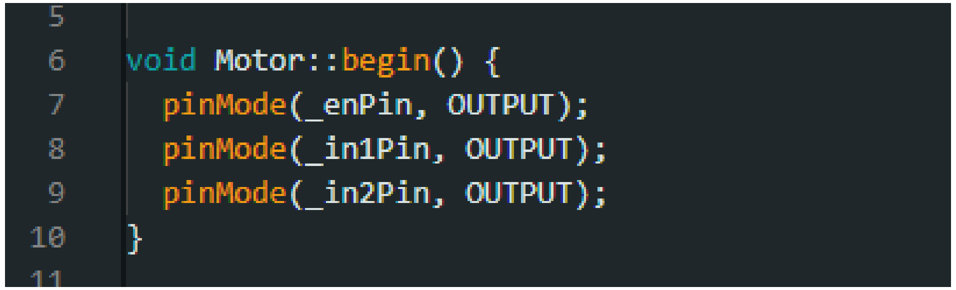


```cpp
void MovementController::act(MovementState state) {
  switch (state) {
    case STOP:
      motorController.stop();
      break;

    case FORWARD: {
      int error = leftIRSensor.isLineDetected() - rightIRSensor.isLineDetected();
      float correction = pid.compute(error);

      int leftSpeed = constrain(baseSpeed - correction, 0, 255);
      int rightSpeed = constrain(baseSpeed + correction, 0, 255);
      motorController.setSpeed(leftSpeed, rightSpeed);
      break;
    }

    case SEARCH_LEFT:
      motorController.setSpeed(-searchSpeed, searchSpeed);
      break;

    case SEARCH_RIGHT:
      motorController.setSpeed(searchSpeed, -searchSpeed);
      break;
  }
}
```

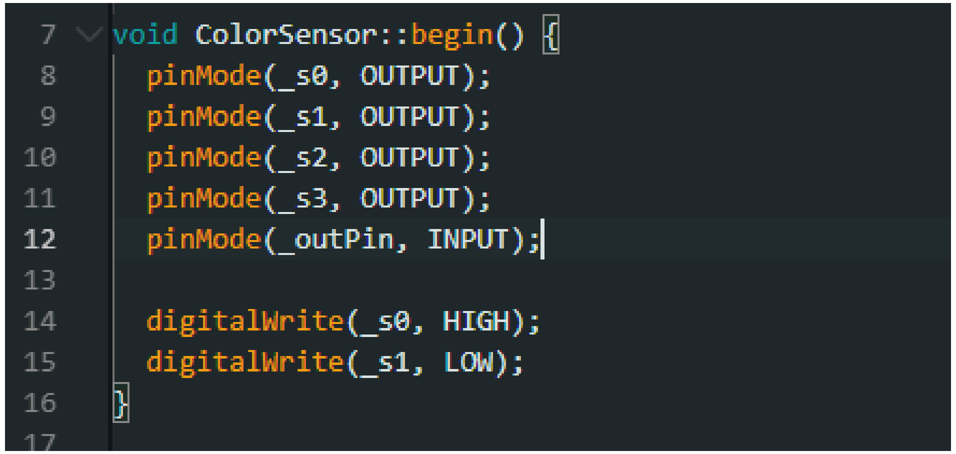
Implementation Snapshot- Setup code with setup function example



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

  // Initialize hardware components
  leftIRSensor.begin();
  rightIRSensor.begin();
  leftUltrasonic.begin();
  rightUltrasonic.begin();
  colorSensor.begin();
  leftMotor.begin();
  rightMotor.begin();

  motorController.stop();
  obstacleChecker.setColorCalibration(colorA, colorB);
}
```

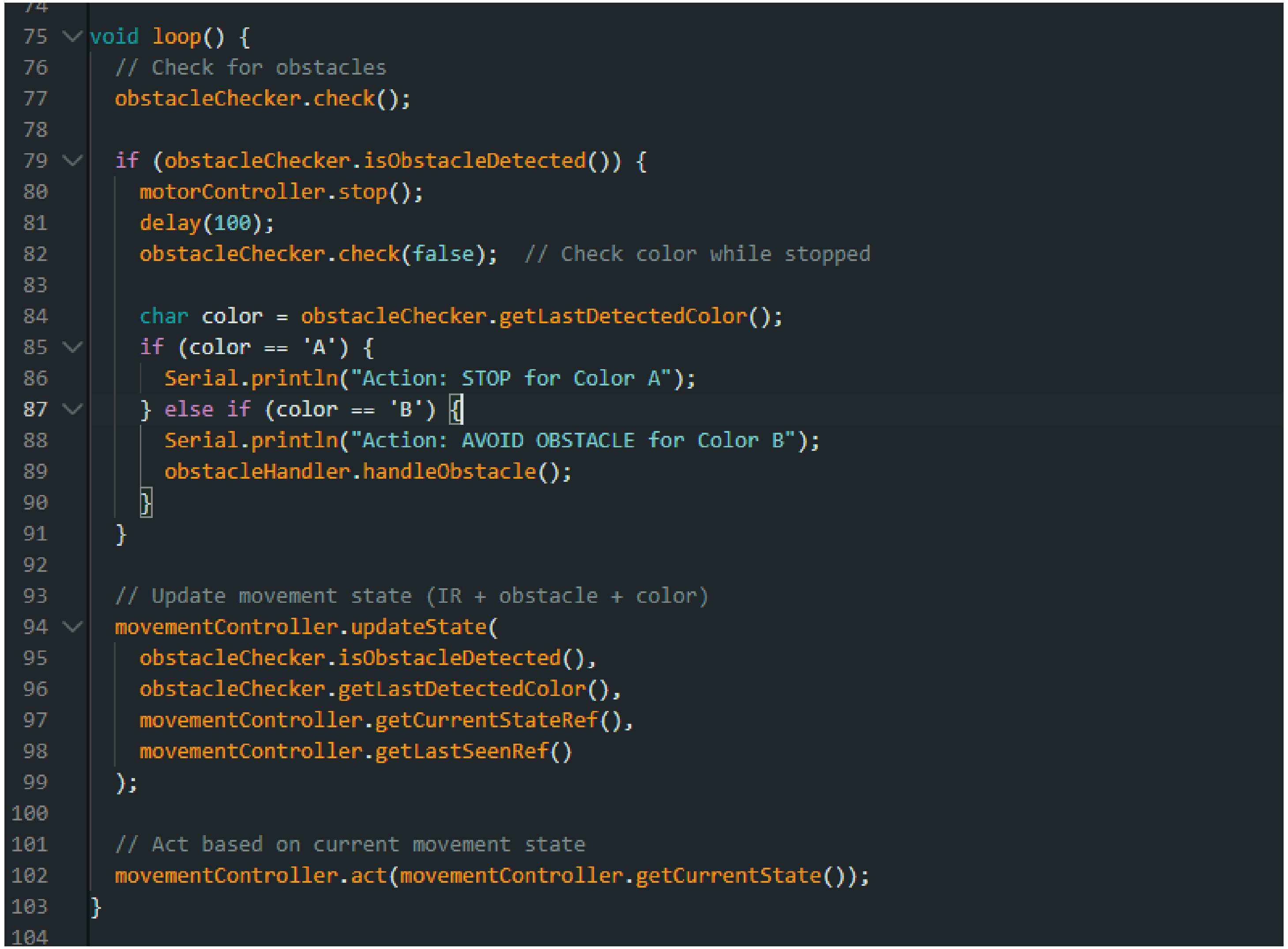


```cpp
void Motor::begin() {
  pinMode(_enPin, OUTPUT);
  pinMode(_in1Pin, OUTPUT);
  pinMode(_in2Pin, OUTPUT);
}
```



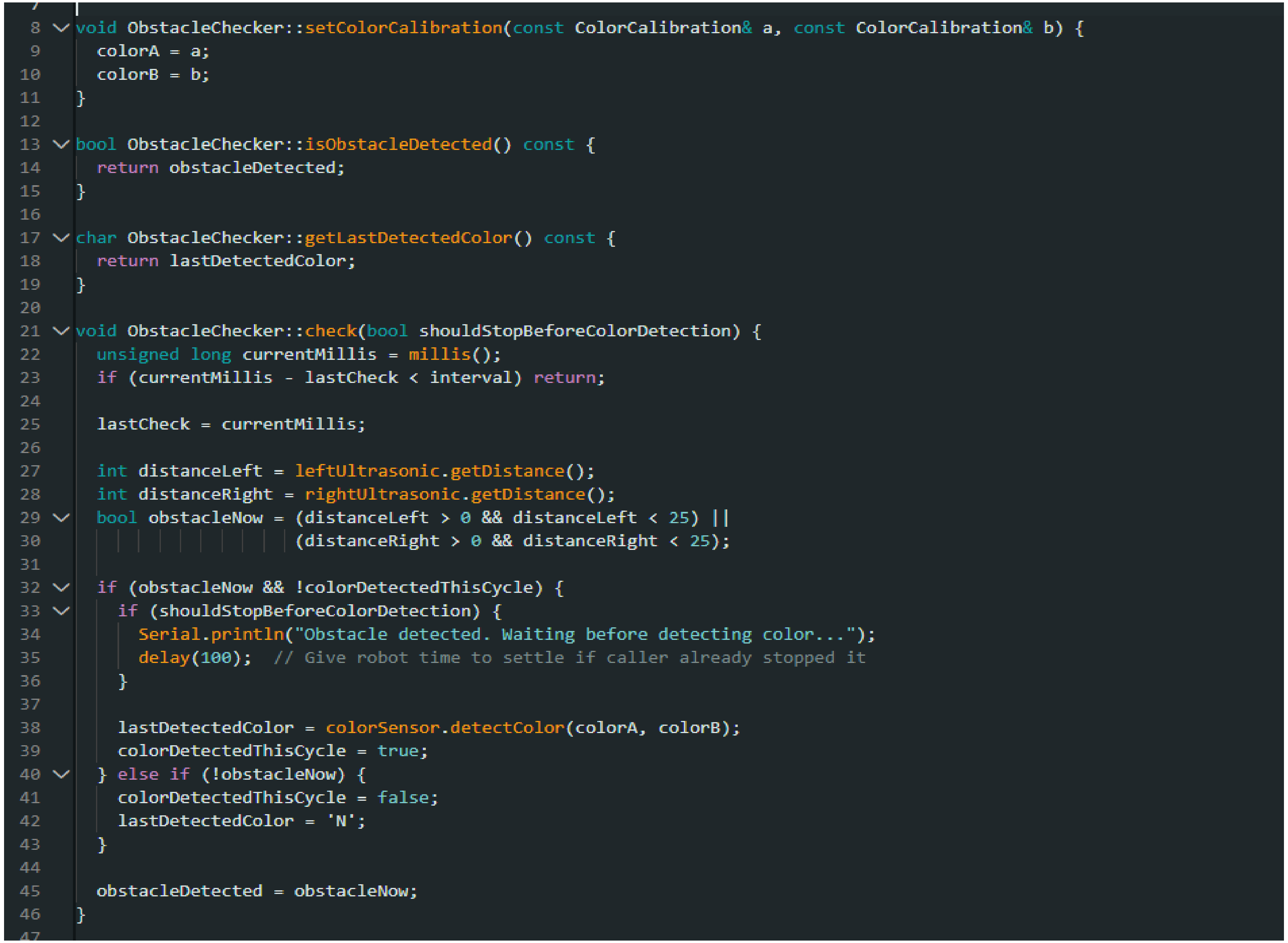
```cpp
void ColorSensor::begin() {
  pinMode(_s0, OUTPUT);
  pinMode(_s1, OUTPUT);
  pinMode(_s2, OUTPUT);
  pinMode(_s3, OUTPUT);
  pinMode(_outPin, INPUT);

  digitalWrite(_s0, HIGH);
  digitalWrite(_s1, LOW);
}
```

Implementation Snapshot- Inside Loop



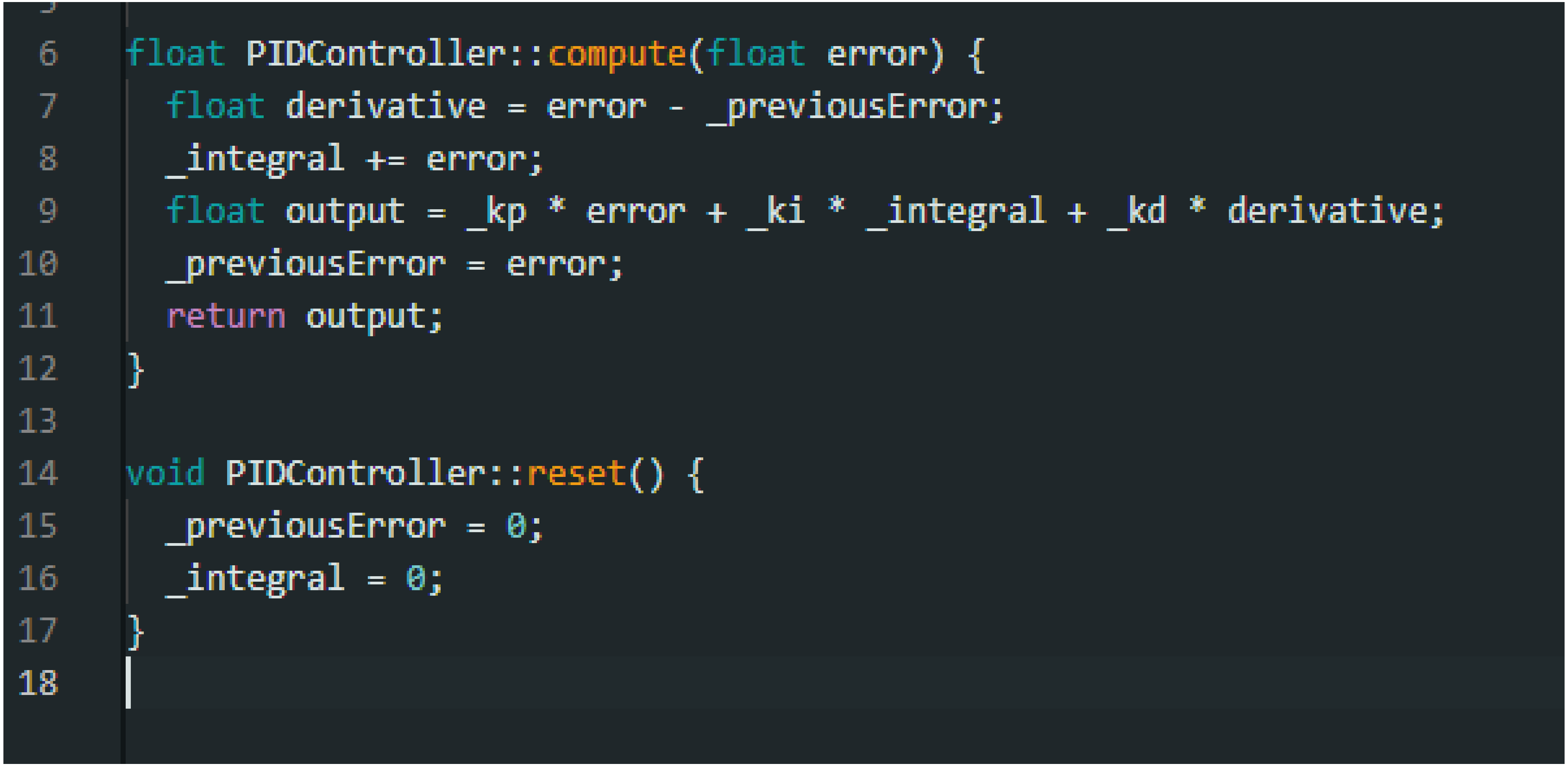
```
void loop() {
  // Check for obstacles
  obstacleChecker.check();

  if (obstacleChecker.isObstacleDetected()) {
    motorController.stop();
    delay(100);
    obstacleChecker.check(false);  // Check color while stopped

    char color = obstacleChecker.getLastDetectedColor();
    if (color == 'A') {
      Serial.println("Action: STOP for Color A");
    } else if (color == 'B') {
      Serial.println("Action: AVOID OBSTACLE for Color B");
      obstacleHandler.handleObstacle();
    }
  }

  // Update movement state (IR + obstacle + color)
  movementController.updateState(
    obstacleChecker.isObstacleDetected(),
    obstacleChecker.getLastDetectedColor(),
    movementController.getCurrentStateRef(),
    movementController.getLastSeenRef()
  );

  // Act based on current movement state
  movementController.act(movementController.getCurrentState());
}
```

Implementation Snapshot- Obstacle Checker and Detecting Color



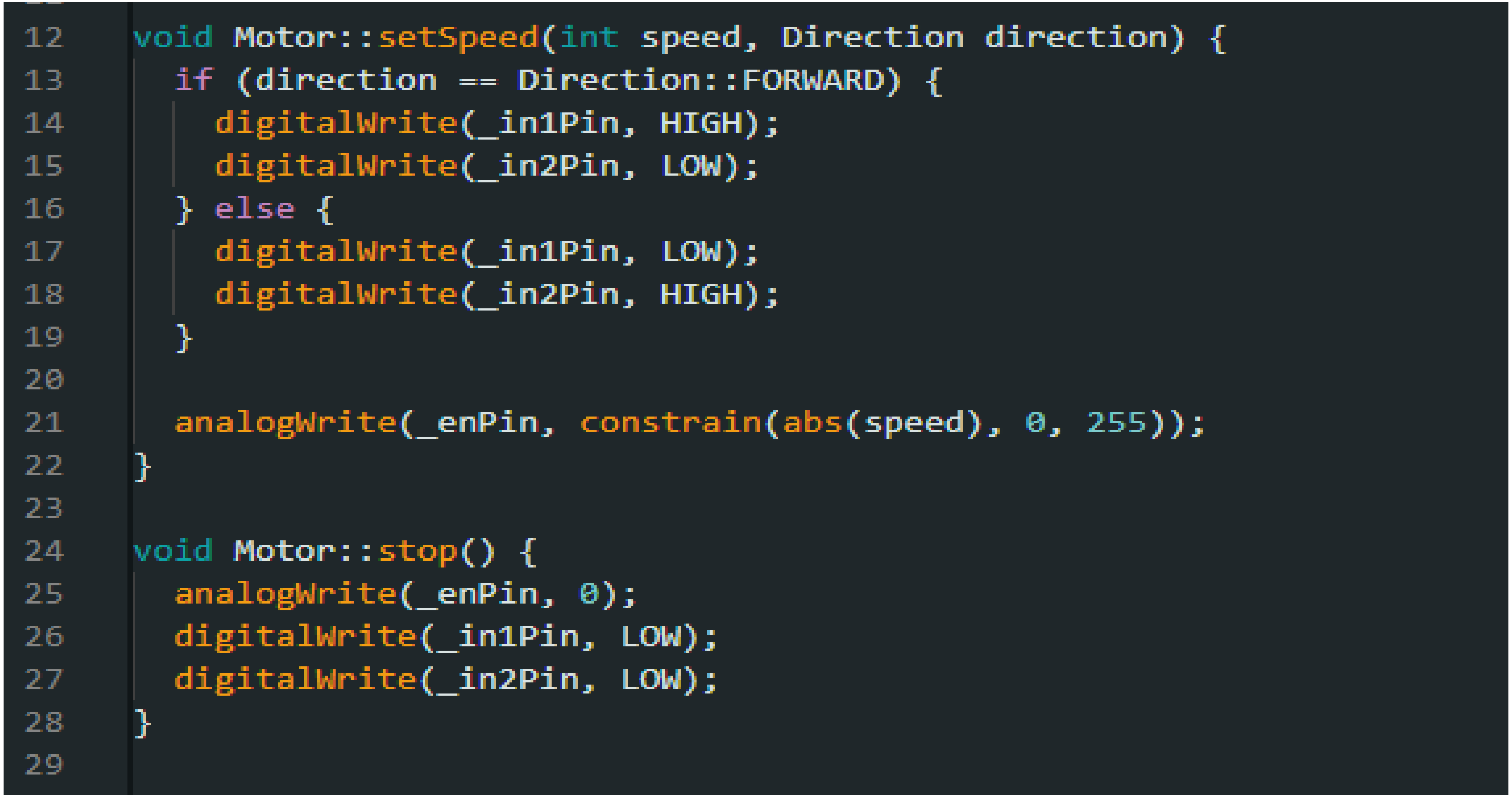
```cpp
void ObstacleChecker::setColorCalibration(const ColorCalibration& a, const ColorCalibration& b) {
  colorA = a;
  colorB = b;
}

bool ObstacleChecker::isObstacleDetected() const {
  return obstacleDetected;
}

char ObstacleChecker::getLastDetectedColor() const {
  return lastDetectedColor;
}

void ObstacleChecker::check(bool shouldStopBeforeColorDetection) {
  unsigned long currentMillis = millis();
  if (currentMillis - lastCheck < interval) return;

  lastCheck = currentMillis;

  int distanceLeft = leftUltrasonic.getDistance();
  int distanceRight = rightUltrasonic.getDistance();
  bool obstacleNow = (distanceLeft > 0 && distanceLeft < 25) ||
                     (distanceRight > 0 && distanceRight < 25);

  if (obstacleNow && !colorDetectedThisCycle) {
    if (shouldStopBeforeColorDetection) {
      Serial.println("Obstacle detected. Waiting before detecting color...");
      delay(100);  // Give robot time to settle if caller already stopped it
    }

    lastDetectedColor = colorSensor.detectColor(colorA, colorB);
    colorDetectedThisCycle = true;
  } else if (!obstacleNow) {
    colorDetectedThisCycle = false;
    lastDetectedColor = 'N';
  }

  obstacleDetected = obstacleNow;
}
```

Implementation Snapshot- PID Logic



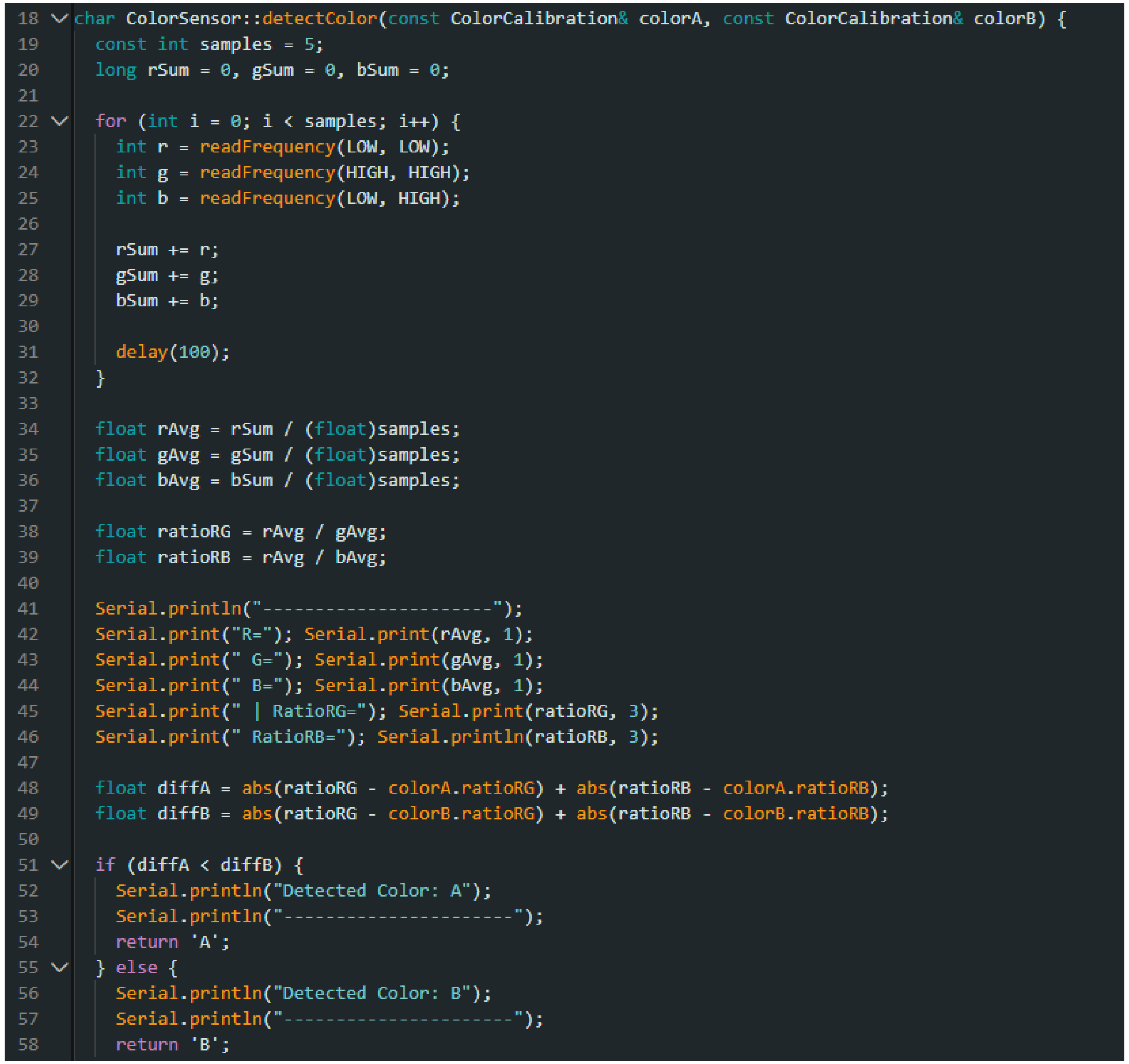
```cpp
float PIDController::compute(float error) {
  float derivative = error - _previousError;
  _integral += error;
  float output = _kp * error + _ki * _integral + _kd * derivative;
  _previousError = error;
  return output;
}

void PIDController::reset() {
  _previousError = 0;
  _integral = 0;
}
```

Implementation Snapshot- Set Motor Speed



```
void Motor::setSpeed(int speed, Direction direction) {
  if (direction == Direction::FORWARD) {
    digitalWrite(_in1Pin, HIGH);
    digitalWrite(_in2Pin, LOW);
  } else {
    digitalWrite(_in1Pin, LOW);
    digitalWrite(_in2Pin, HIGH);
  }

  analogWrite(_enPin, constrain(abs(speed), 0, 255));
}

void Motor::stop() {
  analogWrite(_enPin, 0);
  digitalWrite(_in1Pin, LOW);
  digitalWrite(_in2Pin, LOW);
}

```

Implementation Snapshot- Color sensor



```cpp
char ColorSensor::detectColor(const ColorCalibration& colorA, const ColorCalibration& colorB) {
  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");
    Serial.println("----------------------");
    return 'A';
  } else {
    Serial.println("Detected Color: B");
    Serial.println("----------------------");
    return 'B';
```

Challenges and solutions

1. IR Sensor Calibration

❑*Issue*: Unreliable black tape detection

❑*Fix*: Adjusted angle + threshold calibration

2.Ultrasonic Sensor Range

❑*Issue*: Inconsistent obstacle readings

❑*Fix*: Tuned to 2-400 cm range + noise filtering

3.Color Sensor Stability

❑*Issue*: Fluctuating RGB values

❑*Fix*: Averaged readings + ratio normalization

4. Motor Control @ 12V

❑*Issue*: Uncontrollable speed

❑*Fix*: Unmasked enable pins + PWM limits

5. Line Following PID

❑*Issue*: Jerky turns

❑*Fix*: Implemented PID (Kp=10.0, Ki=0.00, Kd=10.0)

Conclusion and Future Projections

Conclusion

- ➢Successfully built an autonomous vehicle using:
  - Infrared sensors
  - Ultrasonic sensors
  - color sensor
- ➢System operated without human intervention using Arduino-based control.
- ➢Demonstrated stable navigation, responsive obstacle handling, and consistent performance on a contrast-based track.

Future Work

- ➢Machine Learning Integration
- ➢Vision System Enhancement

Reference

[1] IEEE Standards Association, *Autonomous Systems Framework Working Group: P2040 Draft Standard for Framework for Autonomous Systems*, IEEE, 2020. [Online]. Available: https://standards.ieee.org

THANK YOU !