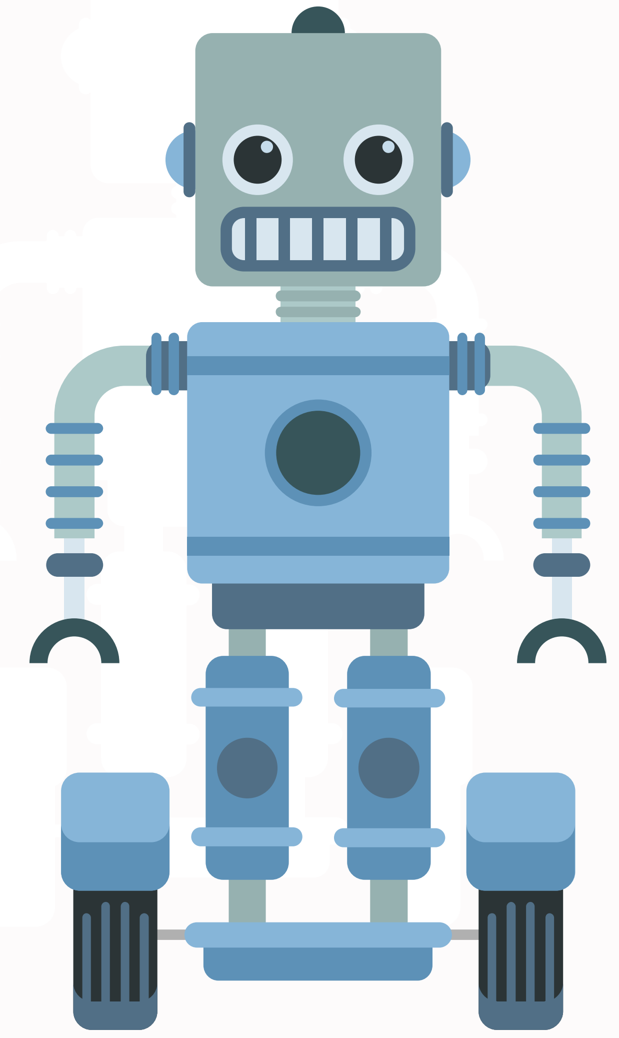
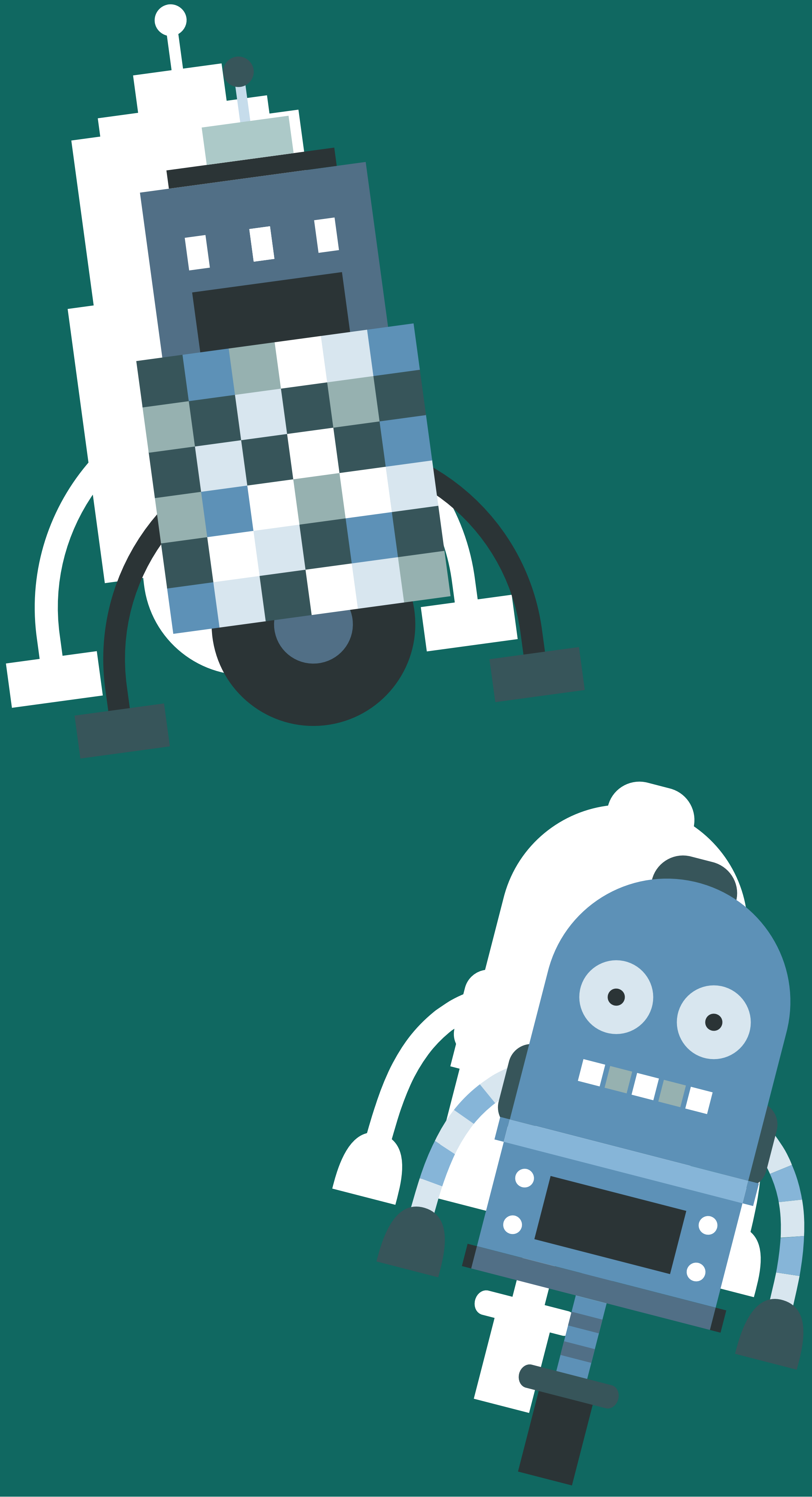
TEAM B1

Automated Vacum Cleaner Robot



Contents -



Introduction

Efficient indoor navigation remains a significant challenge for autonomous robots due to dynamic obstacles and spatial constraints. This project explores a vision-based approach, utilizing a single camera to achieve reliable navigation. Our goal is to develop a system that ensures simplicity and efficiency while maintaining accurate obstacle detection and path planning.

Problem Statement

The challenge of indoor robot navigation lies in accurately detecting obstacles and planning efficient paths using minimal hardware. Traditional systems rely on expensive sensors, but a vision-based approach using a single camera offers a cost-effective alternative. This project aims to develop an autonomous navigation system that utilizes camera-based obstacle detection and path planning for efficient and safe movement in indoor environments.

Our Robot vs. Existing Models

✅ Cost-Effective Solution

- Many existing models rely on expensive sensors like LiDAR, whereas our approach uses a single camera, significantly reducing hardware costs.

✅ Simplified Processing

- Instead of complex SLAM algorithms or deep learning models requiring high computational power, our system uses efficient mathematical techniques for navigation.
-

✅ Improved Obstacle Detection

- Unlike traditional infrared or ultrasonic sensors, our vision-based approach provides a more detailed and adaptable understanding of the environment.

✅ Accurate Self-Localization

- Our angle-based matching technique enhances positioning accuracy without requiring GPS or external beacons.

✅ Optimized Path Planning

- Existing models often rely on pre-mapped environments; our robot dynamically updates its path using real-time obstacle detection and grid mapping.

✅ Versatile and Scalable

- Our method can be implemented in various indoor environments, from homes to warehouses, with minimal recalibration.

Key Features of our model include:

Obstacle Detection: The robot identifies obstacles by analyzing floor images, ensuring safe movement.

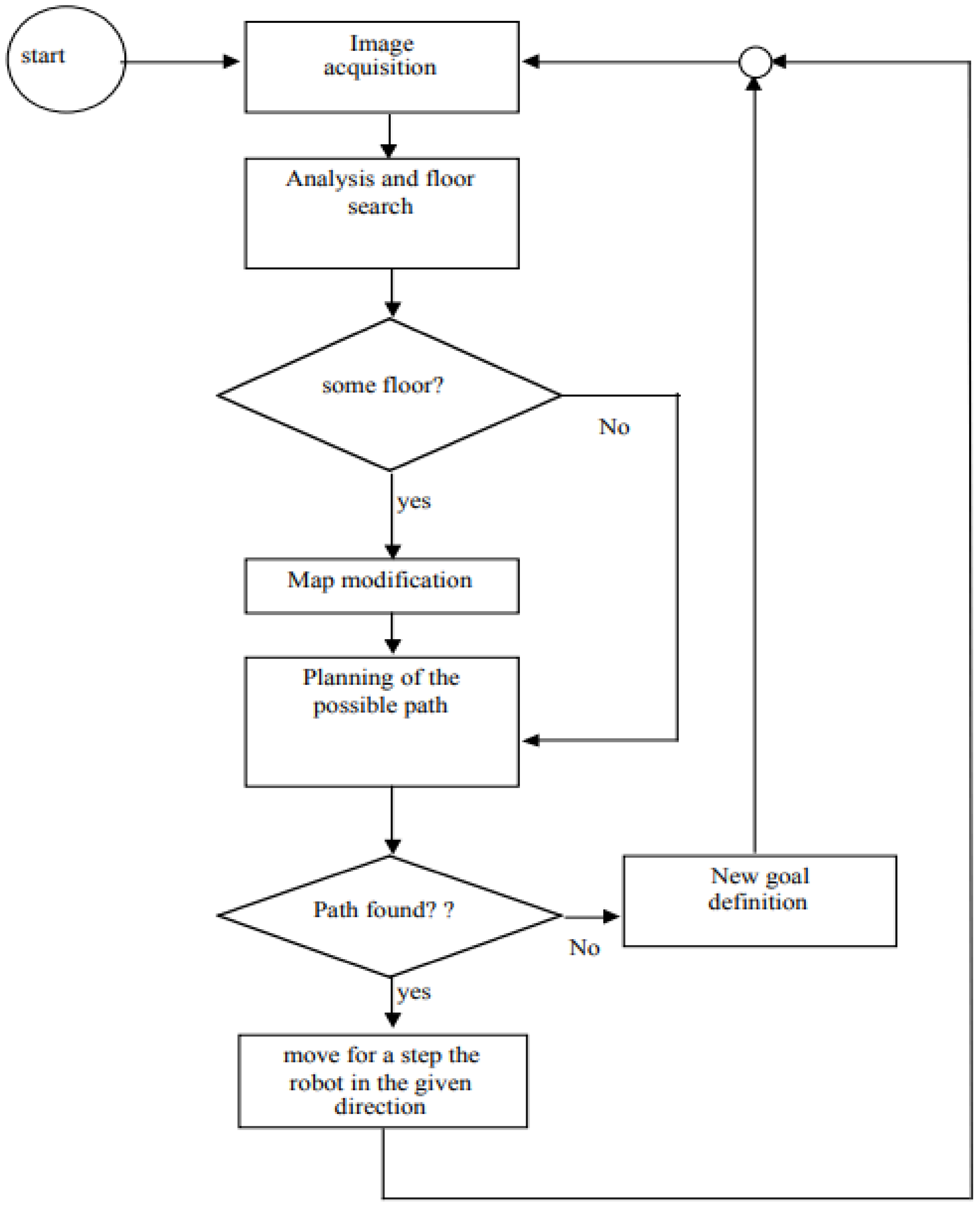
Grid Mapping: After an initial calibration, the system generates a grid-based map of the environment for structured navigation.

Path Planning: Using the generated grid, the robot computes efficient, obstacle-free paths to reach its destination.

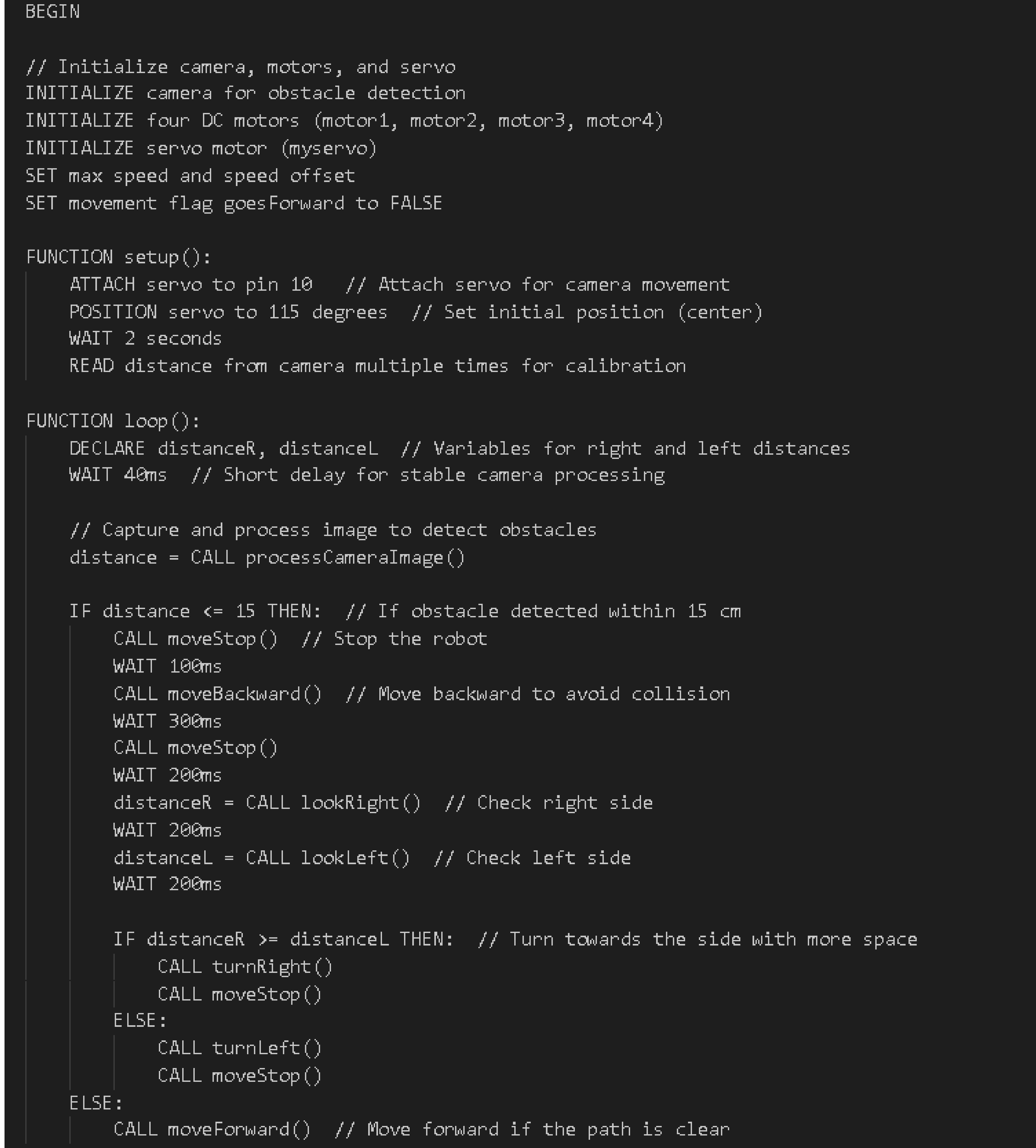
Self-Localization: By leveraging angle-based matching techniques, the robot accurately determines its position without relying on complex sensors.

Efficiency: Our approach optimizes navigation using simple mathematical techniques, reducing computational overhead while maintaining accuracy.

FLOWCHART

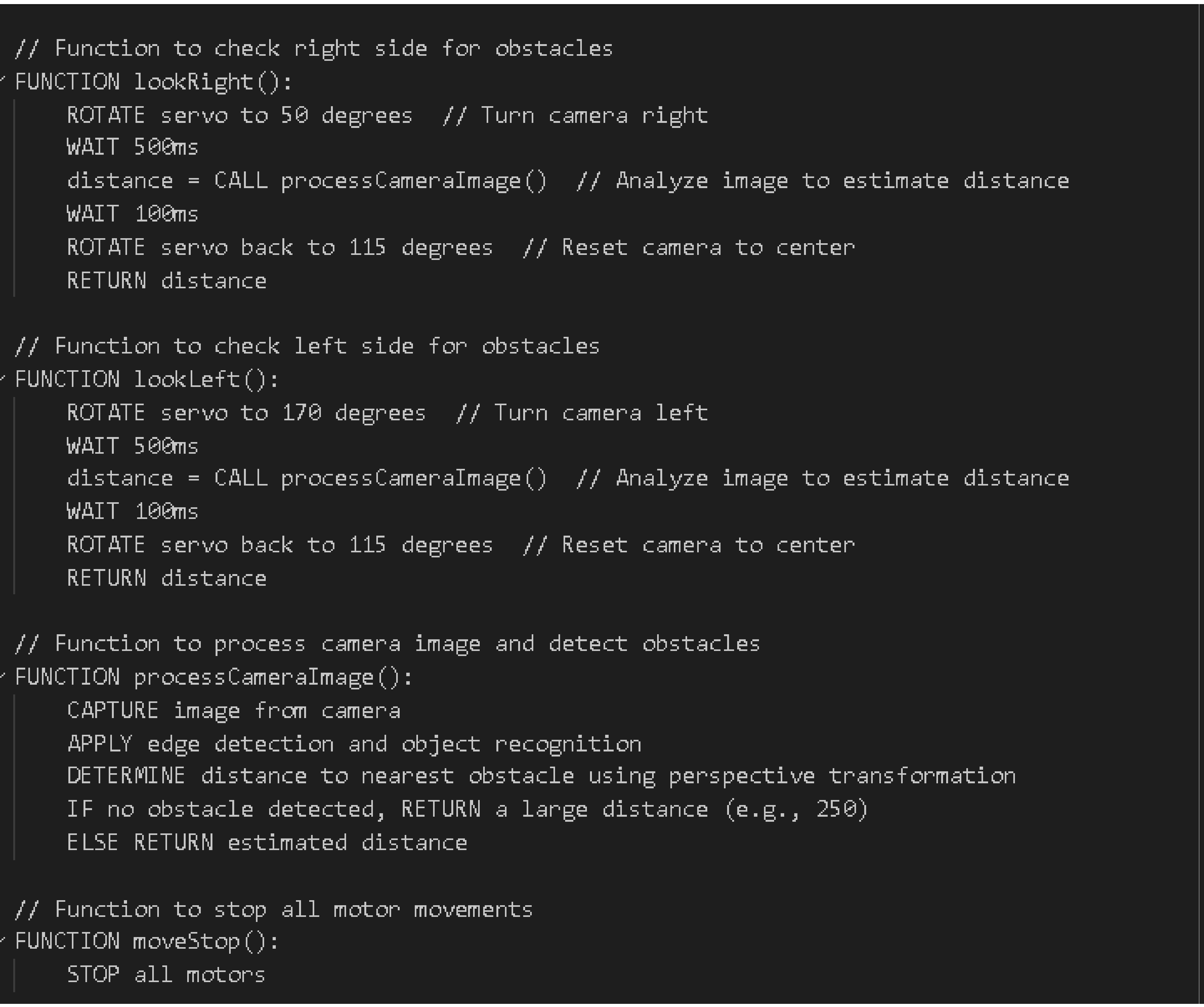

start
Image acquisition
Analysis and floor search
some floor?
No
yes
Map modification
Planning of the possible path
Path found? ?
No
New goal definition
yes
move for a step the robot in the given direction

PSEUDO CODE

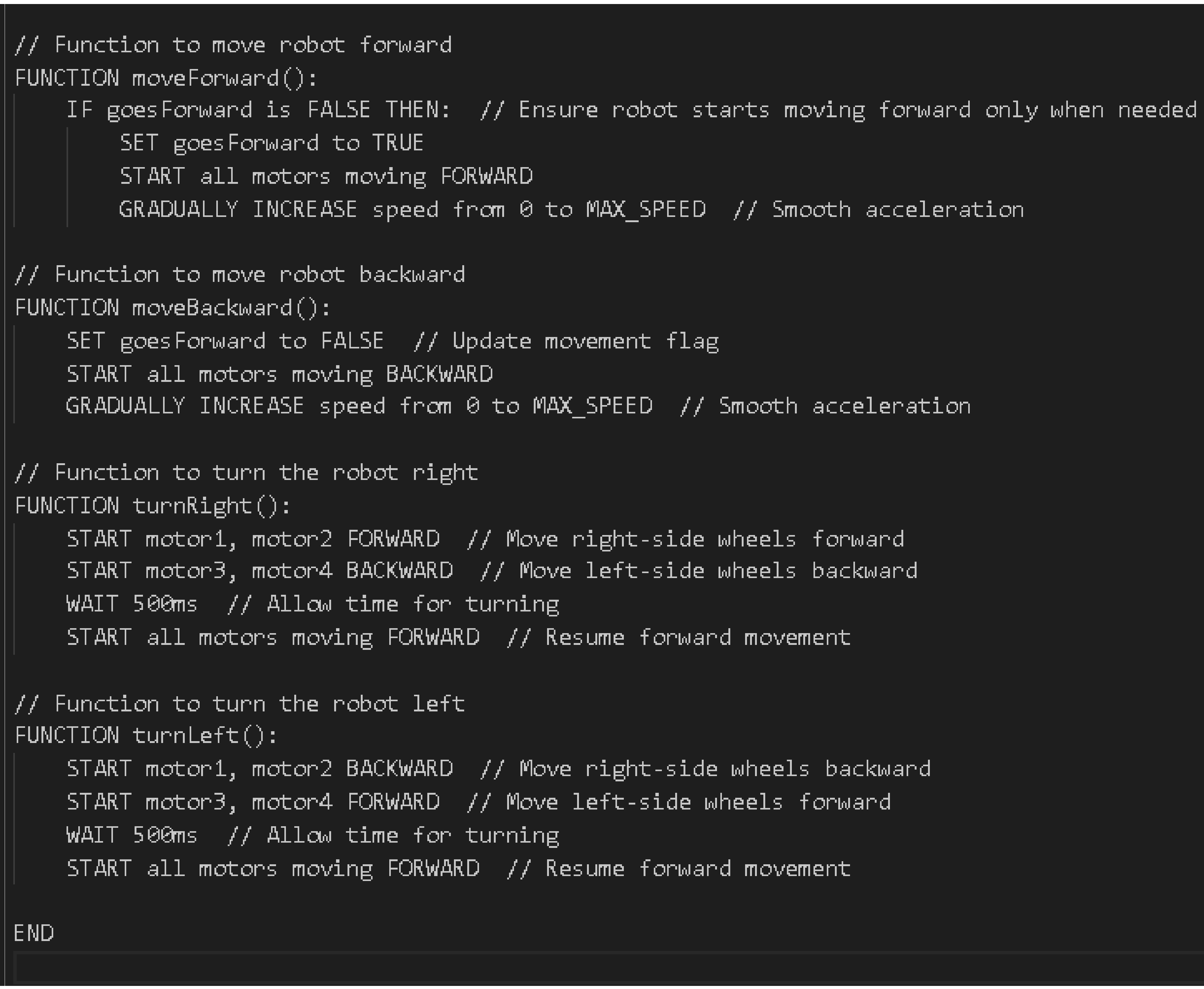


```
BEGIN

// Initialize camera, motors, and servo
INITIALIZE camera for obstacle detection
INITIALIZE four DC motors (motor1, motor2, motor3, motor4)
INITIALIZE servo motor (myservo)
SET max speed and speed offset
SET movement flag goesForward to FALSE

FUNCTION setup():
    ATTACH servo to pin 10   // Attach servo for camera movement
    POSITION servo to 115 degrees  // Set initial position (center)
    WAIT 2 seconds
    READ distance from camera multiple times for calibration

FUNCTION loop():
    DECLARE distanceR, distanceL  // Variables for right and left distances
    WAIT 40ms  // Short delay for stable camera processing

    // Capture and process image to detect obstacles
    distance = CALL processCameraImage()

    IF distance <= 15 THEN:  // If obstacle detected within 15 cm
        CALL moveStop()  // Stop the robot
        WAIT 100ms
        CALL moveBackward()  // Move backward to avoid collision
        WAIT 300ms
        CALL moveStop()
        WAIT 200ms
        distanceR = CALL lookRight()  // Check right side
        WAIT 200ms
        distanceL = CALL lookLeft()  // Check left side
        WAIT 200ms

        IF distanceR >= distanceL THEN:  // Turn towards the side with more space
            CALL turnRight()
            CALL moveStop()
        ELSE:
            CALL turnLeft()
            CALL moveStop()
    ELSE:
        CALL moveForward()  // Move forward if the path is clear
```

PSEUDO CODE



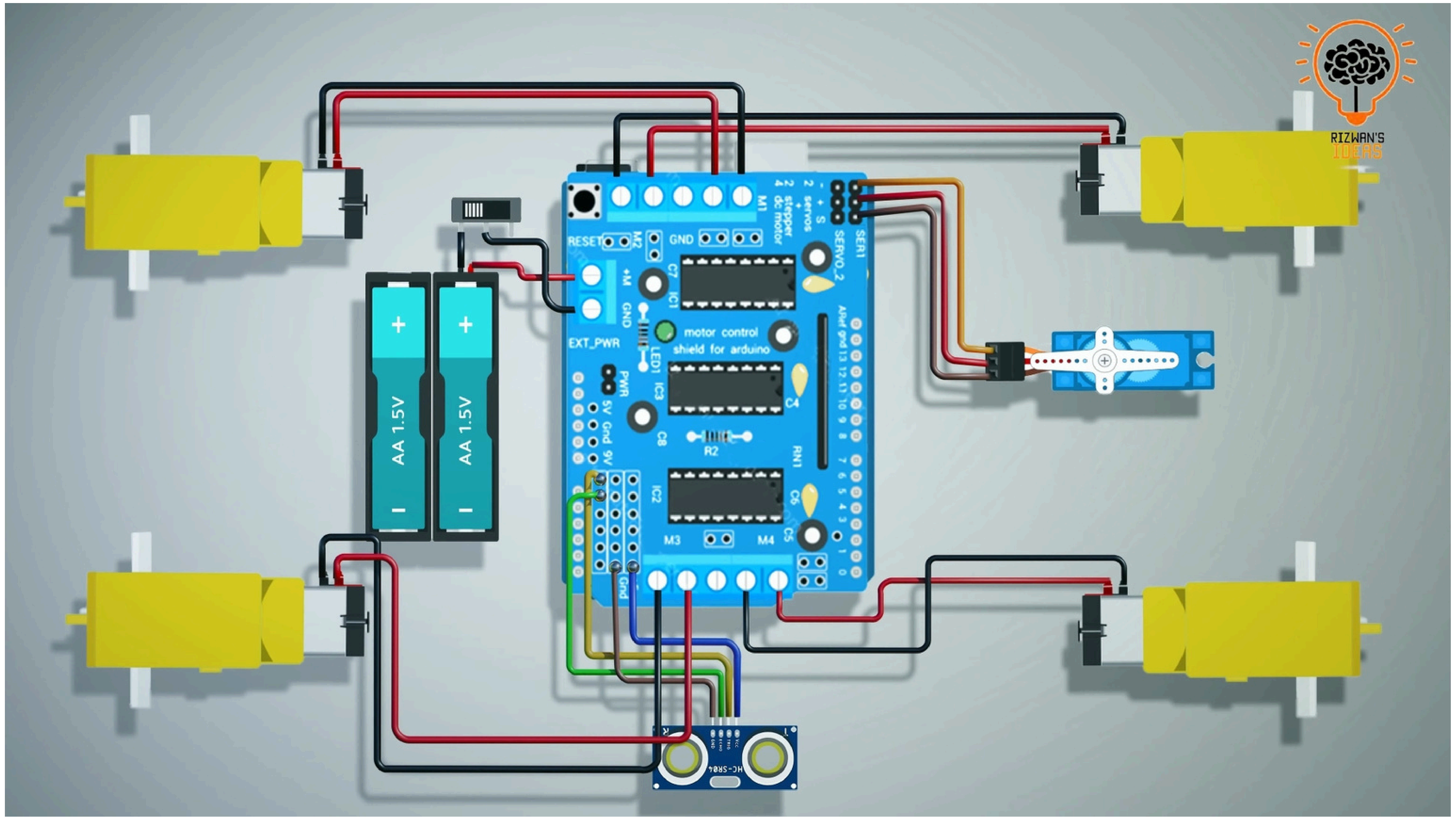
```
// Function to check right side for obstacles
FUNCTION lookRight():
    ROTATE servo to 50 degrees  // Turn camera right
    WAIT 500ms
    distance = CALL processCameraImage()  // Analyze image to estimate distance
    WAIT 100ms
    ROTATE servo back to 115 degrees  // Reset camera to center
    RETURN distance

// Function to check left side for obstacles
FUNCTION lookLeft():
    ROTATE servo to 170 degrees  // Turn camera left
    WAIT 500ms
    distance = CALL processCameraImage()  // Analyze image to estimate distance
    WAIT 100ms
    ROTATE servo back to 115 degrees  // Reset camera to center
    RETURN distance

// Function to process camera image and detect obstacles
FUNCTION processCameraImage():
    CAPTURE image from camera
    APPLY edge detection and object recognition
    DETERMINE distance to nearest obstacle using perspective transformation
    IF no obstacle detected, RETURN a large distance (e.g., 250)
    ELSE RETURN estimated distance

// Function to stop all motor movements
FUNCTION moveStop():
    STOP all motors
```



```
// Function to move robot forward
FUNCTION moveForward():
    IF goesForward is FALSE THEN:  // Ensure robot starts moving forward only when needed
        SET goesForward to TRUE
        START all motors moving FORWARD
        GRADUALLY INCREASE speed from 0 to MAX_SPEED  // Smooth acceleration

// Function to move robot backward
FUNCTION moveBackward():
    SET goesForward to FALSE  // Update movement flag
    START all motors moving BACKWARD
    GRADUALLY INCREASE speed from 0 to MAX_SPEED  // Smooth acceleration

// Function to turn the robot right
FUNCTION turnRight():
    START motor1, motor2 FORWARD  // Move right-side wheels forward
    START motor3, motor4 BACKWARD  // Move left-side wheels backward
    WAIT 500ms  // Allow time for turning
    START all motors moving FORWARD  // Resume forward movement

// Function to turn the robot left
FUNCTION turnLeft():
    START motor1, motor2 BACKWARD  // Move right-side wheels backward
    START motor3, motor4 FORWARD  // Move left-side wheels forward
    WAIT 500ms  // Allow time for turning
    START all motors moving FORWARD  // Resume forward movement

END
```

COMPONENTS

- Gear Motor X 4
- 12v AC to DC Power Supply
- Arduino Uno
- Arduino motor shield
- Servo motor
- 6v motor
- Lithium-ion battery cell
- Camera



CAMERA CALIBRATION

Developing a calibration procedure using the pin-hole camera model to map image coordinates to real-world coordinates.
H matrix transformation is used to compute world coordinates from image coordinates.

$$\begin{bmatrix} u' \\ v' \\ w' \end{bmatrix} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} * \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = H * \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Adjustments are made to account for principal point offset, aspect ratio, and non-orthogonal axes using a K matrix.

- **the origin of the geometric system is (u0 v0),**
- **the transform of u is [1, 0, 0] T**
- **the transform of v is the product a . [α, 1, 0] T .**

$$K = \begin{bmatrix} 1 & a * \alpha & u_0 \\ 0 & a & v_0 \\ 0 & 0 & 1 \end{bmatrix}$$

The complete transform from scene to image is so:

$$\begin{bmatrix} u' \\ v' \\ w' \end{bmatrix} = K * \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} * H * \begin{bmatrix} x \\ y \\ z \\ t \end{bmatrix}$$

Since we are interested only in the transform between the two reference frames, we can multiply the three matrices and obtain M:

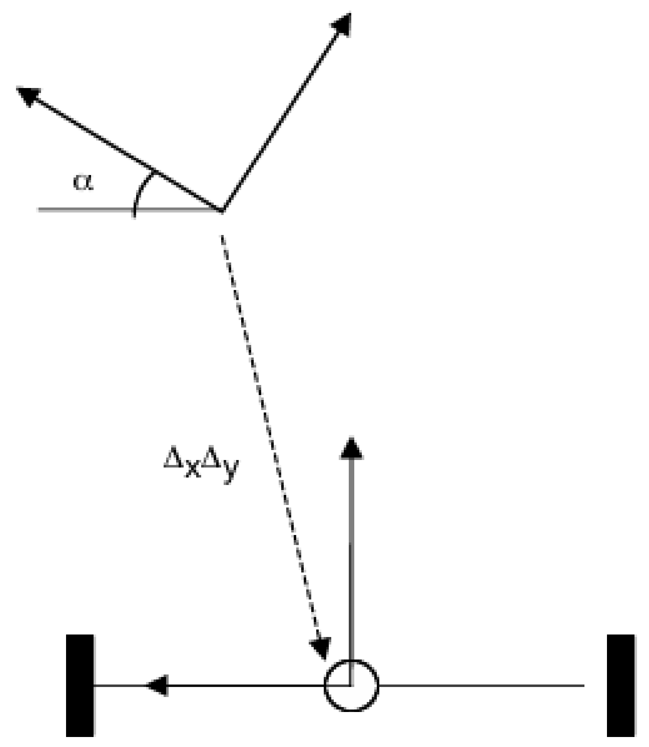
$$\begin{bmatrix} u' \\ v' \\ w' \end{bmatrix} = \begin{bmatrix} m1 & m2 & m3 & m4 \\ m5 & m6 & m7 & m8 \\ m9 & m10 & m11 & m12 \end{bmatrix} * \begin{bmatrix} x \\ y \\ z \\ t \end{bmatrix} = M * \begin{bmatrix} x \\ y \\ z \\ t \end{bmatrix}$$

two-phase calibration:
1.Image to floor transformation using a reference white square (21 cm width).
2.Floor to robot transformation by capturing images from different positions and orientations.

$$T = \begin{bmatrix} a & b & 0 & h \\ c & d & 0 & k \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \cos(\alpha) & \sin(\alpha) & 0 & \Delta x \\ -\sin(\alpha) & \cos(\alpha) & 0 & \Delta y \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

where !x , !y , and 0 are translations, ! the rotation between the two systems. To compute T we put the square on the floor, collect the coordinates of the vertices and the odometry many times moving the robot to obtain coordinates for different viewpoints. Remember that T is a roto-translation matrix, where non-linear conditions should hold:

$$a^2 + c^2 = 1$$
$$b^2 + d^2 = 1$$
$$a * b + c * d = 0$$



Final transformation matrix M allows conversion between Cartesian and pixel coordinates for accurate navigation.

$$\begin{bmatrix} u' \\ v' \\ w' \end{bmatrix} = M * T^{-1} * \begin{bmatrix} x \\ y \\ 0 \\ 1 \end{bmatrix}$$

$$u = \frac{u'}{w'}$$

$$v = \frac{v'}{w'}$$

For the reverse transformation, we invert the equation and get the formulas:

$$S = \frac{m8 - v * m12 - \dfrac{(v * m9 - m5) * (m4 - m12 * u)}{u * m9 - m1}}{\dfrac{(v * m9 - m5) * (m2 - u * m10)}{u * m9 - m1} + v * m10 - m6}$$

$$D = \frac{(m2 - u * m10) * S + m4 - m12 * u}{u * m9 - m1}$$

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = T * \begin{bmatrix} S \\ D \\ 0 \\ 1 \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} x'/z' \\ y'/z' \\ 0 \end{bmatrix} \qquad (8)$$

IMAGE ANALYSIS

Assumes the floor has a uniform texture, and obstacles disrupt this pattern.
Uses statistical analysisto extract floor features:

- Computes mean intensity and 4th order moment for each pixel in RGB format.
- Higher moment values indicate obstacles, while lower values correspond to uniform floor areas.

$$\mu\_red(x,y) = \frac{\sum_{(i,j)\in(x,y)} red(i,j)}{N}$$

and

$$m\_red(x,y) = \frac{\sum_{(i,j)\in(x,y)} (red(i,j) - \mu\_red(i,j))^4}{N}$$

(where N is the number of pixels in the region)

Optimization techniques:

- Analysis is performed only on a subset of pixels to reduce computation.
- Linear interpolation fills in missing data.

$$new\_red = \mu\_red(x,y) + (red(x,y) - \mu\_red(x,y)) * m\_red\_norm(x,y)$$

Converts the processed image to HSL format (Hue, Saturation, Luminance) for better contrast.
Uses a dynamic thresholding technique to binarize the image, distinguishing floor from obstacles.

$$fx = \frac{(3 - \sum_{red,green,blue} \frac{moment(x,y)}{moment\_max})}{3}$$

Final Output: A processed binary image where the floor is white and obstacles are black.

SELF - LOCALIZATION

Robot compares its local map (built dynamically) with the global map for localization.

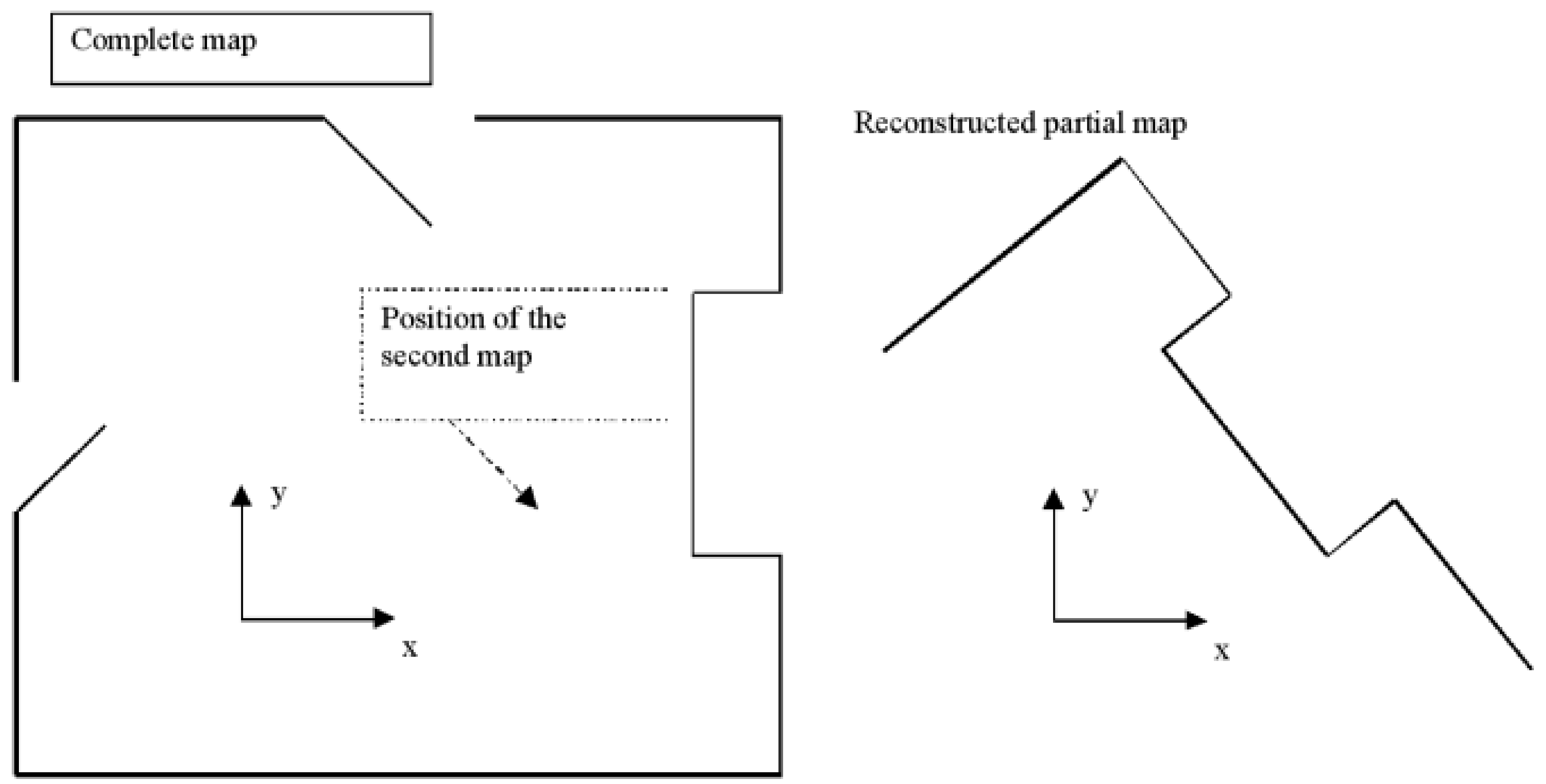
- Key concept: Uses angles between walls as features for map matching.

Voting system:

- Each detected angle in the local map is compared to angles in the global map.
- A rototranslation matrix is computed for position estimation.
- A Gaussian distribution is used to assign weights to each match.

Enhancements with segment matching:

- Segments are compared based on angular difference, centroid distance, and alignment.
- This refines the robot’s estimated position, reducing localization errors.



Real-world challenges:
Imperfections in image processing and vector quantization.
Dynamic obstacles that may temporarily interfere with localization.

PATH PLANNING (DIJKSTRA'S ALGORITHM)

Concept

The robot must compute the shortest path to clean the floor while avoiding obstacles.

Mathematical Foundation

The robot's environment is a graph where:

Each grid cell is a node

Edges represent possible movement paths

Obstacle cells have infinite cost

Dijkstra's Algorithm finds the shortest path:

$$d(v)=\min(d(u)+w(u,v))$$

where:

d(v)d(v)d(v) = shortest distance to node vvv

w(u,v)w(u, v)w(u,v) = weight (distance between nodes)

Application in the Robot

The robot assigns a cost to each grid cell.

It runs Dijkstra's Algorithm to find the shortest path.

It moves toward the target while avoiding obstacles.

LITERATURE REVIEW

SERIAL NO	AUTHOR	TITLE OF THE PAPER	CONCEPTUAL UNDERSTANDING
1	Latombe Jean-Claude	"Robot Motion Planning"	Path Planning: Dijkstra, and reinforcement learning optimize routes Real-time adaptation to obstacles
2	Burgard W., Fox D. Thrun S	"Active Mobile Robot Localization"	Self-Localization: Angle-based feature matching for precise positioning SURF and geometric transformations enhance accuracy
3	Freeman Herbert	"Machine Vision for Inspection and Measurement",	Efficiency & Optimization: Lightweight CNNs for real-time processing Mathematical models reduce computation time

SERIAL NO	AUTHOR	TITLE OF THE AUTHOR	CONCEPTUAL UNDERSTANDING
4	Horswill Ian	"Polly: A Vision-Based Artificial Agent"	Vision-Based Navigation: Uses a single camera for cost-effective navigation Deep learning and edge-detection methods improve accuracy
5	Burgard W., Fox D., Thrun S	"Active MobileRobotLocalization "	Obstacle Detection: Optical flow and depth estimation help identify static & dynamic obstacles Camera-based techniques replacing traditional sensors
6	Thrun Sebastian	"Learning Metric-Topological Maps for Indoor Mobile Robot Navigation",	Grid Mapping: Calibration-based mapping for structured navigation Probabilistic methods improve localization