Robotics Navigation and Obstacle Avoidance

Mohammad Aljazzazi

FEBRUARY 1, 2024

Mobile Robotics

Introduction

Context

The field of robotics has seen significant advancements in recent years, with applications ranging from manufacturing to healthcare. Autonomous navigation plays a critical role in robotics, enabling robots to move efficiently and safely in various environments. This report focuses on the development of a robust control system for a robotic vehicle, emphasizing navigation towards a target while effectively avoiding obstacles.

Objective

The primary objective of this project is to create a control system that allows a robotic vehicle to navigate towards a predefined target while actively avoiding obstacles in its path. This system aims to address the complexities involved in autonomous navigation, such as dynamic environment interaction, precision in movement, and real-time decision-making.

Problem Statement

Challenges in Robotics

Autonomous navigation in robotics presents several challenges. Robots must interact with dynamic environments, make precise movements, and make real-time decisions to reach their goals while avoiding collisions with obstacles. These complexities demand adaptive control strategies.

Specific Problem

The specific problem addressed in this project is the need for an adaptive control system that can manage varying distances to the target and integrate obstacle detection and avoidance mechanisms seamlessly. The goal is to create a control system that is robust, responsive, and capable of adapting to changing scenarios.

Solution Overview

Proposed Solution

The proposed solution combines Potential Field path planning and adaptive Proportional-Integral-Derivative (PID) control with an obstacle avoidance strategy. This combination allows the robotic vehicle to navigate towards the target while actively avoiding obstacles in its path, providing a versatile and effective control mechanism.

System Requirements

To implement this solution, certain hardware and software requirements are necessary. These include sensors for obstacle detection and computational resources for real-time decision-making. The sensors used in this project include proximity sensors and a vision sensor, and the computational resources depend on the robotic platform's capabilities.

Code Breakdown and Explanation

Initialization

The code begins by initializing the connection to the robot simulation environment (V-REP) and obtaining handles for various components, including motors, sensors, and the robot itself. This setup is crucial for communication and control.

Main Control Loop

The main control loop continuously reads sensor data, computes control actions, and adjusts the robot's movement. It encompasses the entire control process.

PID Control Implementation

The code implements PID controllers for both angular and positional control. These controllers help the robot maintain the desired angle and distance from the target.

Obstacle Avoidance Logic

The code incorporates obstacle avoidance logic using proximity sensor data. It calculates a movement direction based on the sensor readings and employs strategies to avoid obstacles effectively.

Adaptive Gains Control

Rationale for Adaptive Gains

Adaptive gains are essential for efficient control at different distances from the target. The code dynamically adjusts PID gains to optimize the robot's performance in various scenarios.

Implementation Details

The code dynamically modifies PID gains based on the robot's current distance from the target, ensuring optimal control throughout the navigation process.

Integration of PID Control and Obstacle Avoidance

Balancing Navigation and Avoidance

The code seamlessly integrates navigation towards the target with obstacle avoidance. It ensures that the robot can switch between following the PID control outputs and performing avoidance maneuvers as needed.

Decision-Making Process

The code employs a decision-making process that considers regional probabilities and the robot's angle relative to the target to determine whether to prioritize navigation or obstacle avoidance.

Methodology

The methodology section of the report integrates the provided code and offers explanations for each section. The code is written in MATLAB and is structured to demonstrate the process of robotic navigation and obstacle avoidance.

Code Breakdown and Explanation

Initialization

```
vrep=remApi('remoteApi');
vrep.simxFinish(-1);

clientID=vrep.simxStart('127.0.0.1',19999,true,true,5000,5);

if (clientID>-1)
    disp('Connected')
    % Handle
    [returnCode,left_Motor]=vrep.simxGetObjectHandle(clientID,'Pioneer_p3dx_l
eftMotor',vrep.simx_opmode_blocking);
    [returnCode,right_Motor]=vrep.simxGetObjectHandle(clientID,'Pioneer_p3dx_
rightMotor',vrep.simx_opmode_blocking); % Code to Access Motors

    % ... (handle retrieval for sensors, robot, end position, and camera)

    sensors = [sensor1, sensor2, sensor3, sensor4, sensor5, sensor6, sensor7,
sensor8];

    KpAng = 0.1; % Proportional gain
    KiAng = 0.000; % Integral gain
    KdAng = 0; % Derivative gain

    % Initialize variables
    currentErrorAng = 0;
    integralErrorAng = 0;
    angleThreshold = 0.01; % Threshold to determine when the target is reache
d

    KpPos = 1.1; % Proportional gain
    KiPos = 0; % Integral gain
    KdPos = 0; % Derivative gain

    % Initialize variables
    currentErrorPos = 0;
```

```
    integralErrorPos = 0;

    % ... (rest of the initialization)
```

Explanation: In the initialization section, we establish a connection to the robot simulation environment (V-REP) and obtain handles for various components, including motors, sensors, the robot, and the camera. These handles are crucial for subsequent communication and control operations. Additionally, the code initializes PID control parameters and variables for both angular and positional control.

Main Control Loop

```
% ... (Main Control Loop section, up to the loop start)

for i=1:1000
    [returnCode,robotposition]=vrep.simxGetObjectPosition(clientID,robot, -1,
vrep.simx_opmode_buffer);
    [returnCode,endposition]=vrep.simxGetObjectPosition(clientID,endpos, -1,
vrep.simx_opmode_buffer);
    [returnCode,robotAngles]=vrep.simxGetObjectOrientation(clientID, robot, -
1, vrep.simx_opmode_buffer);

    % ... (rest of the main control loop)
```

Explanation: The main control loop is where the core control logic resides. It iterates for a specified number of steps (here, 1000) and continuously updates robot position, end position, robot angles, and vision sensor data. This information is crucial for calculating errors and making control decisions.

PID Control Implementation

```
% ... (PID Control Implementation section, up to the PID angular control)

%%%%%PID controller Angular
previousErrorAng = currentErrorAng;

currentErrorAng = angle;

integralErrorAng = integralErrorAng + currentErrorAng;

outputAng = calculatePID(currentErrorAng, previousErrorAng, integralErrorAng,
KpAng, KiAng, KdAng);

% ... (rest of the PID angular control)
```

Explanation: This part of the code implements the PID controller for angular control. It calculates the current error in the robot's orientation (angle) relative to the target and applies the PID control formula to determine the control output for angular adjustment.

Obstacle Avoidance Logic

```
% ... (Obstacle Avoidance Logic section, up to sensor readings)

%%%%%%%%%%%%%%%
% Define sensor angles (example angles)
```

```
sensorAngles = linspace(0, 180, 8); % degrees

% Initialize net force vector
netForceX = 0;
netForceY = 0;

influenceRange = 1;
K_rep = 1;
K_att = 1;

for n=1:8
    [returnCode,detectionState,detectedPoint,~,~]=vrep.simxReadProximitySenso
r(clientID,sensors(n),vrep.simx_opmode_buffer);

    % ... (rest of the obstacle avoidance logic)
```

Explanation: In the obstacle avoidance logic, the code reads data from proximity sensors to detect obstacles. It calculates regional probabilities based on sensor readings and determines the direction in which the robot should move to avoid obstacles effectively.

% ... (Continuation of Obstacle Avoidance Logic section, up to the decision-making process)

```
% Determine direction based on regional probabilities
if (angle > 30)
    % ... (obstacle avoidance strategy for right turns)
elseif (angle < -30)
    % ... (obstacle avoidance strategy for left turns)
elseif (angle <= 30 && angle >= -30)
    % ... (obstacle avoidance strategy for straight paths)
end

% ... (rest of the code)
```

Explanation: This part of the obstacle avoidance logic handles the decision-making process based on regional probabilities and the robot's angle relative to the target. It selects the appropriate obstacle avoidance strategy or navigation mode for the robot.

Conclusion

Summary of Achievements This project has successfully developed a control system that enables a robotic vehicle to navigate towards a target while effectively avoiding obstacles. The integration of adaptive PID control and obstacle avoidance logic allows the robot to handle dynamic scenarios and achieve its objectives.

Additional Code Explanation

The provided code defines a function called computeDistanceAndAngle that calculates the distance and angle between a robotic vehicle and a target. Let's identify and explain various aspects of this function:

```
function [distance, angle, angleNorm] = computeDistanceAndAngle(robotPosition
, targetPosition, robotOrientation)
```

Function Signature: This line defines the function computeDistanceAndAngle with three input parameters (robotPosition, targetPosition, and robotOrientation) and three output values (distance, angle, and angleNorm).

```
    % Calculate the distance between the robot and the target
    distance = sqrt((targetPosition(1) - robotPosition(1))^2 + (targetPositio
n(2) - robotPosition(2))^2);
```

Distance Calculation: This part of the code calculates the Euclidean distance between the robot's position (robotPosition) and the target's position (targetPosition) in a 2D space.

```
    % Calculate the angle from the robot to the target
    % The angle to the target is the angle of the line connecting the robot a
nd the target, relative to the global frame
    angleToTargetGlobalFrame = atan2(targetPosition(2) - robotPosition(2), ta
rgetPosition(1) - robotPosition(1));
```

Angle Calculation (Global Frame): This section computes the angle from the robot to the target in the global frame. It uses the atan2 function to calculate the angle based on the differences in the y and x coordinates between the target and the robot.

```
    % The orientation of the robot is its angle relative to the global frame
    % The desired angle of motion relative to the robot's frame is the angle
to the target minus the robot's orientation
    robotOrientation3 = robotOrientation(3);
    angleDiff = mod(rad2deg(angleToTargetGlobalFrame) - rad2deg(robotOrientat
ion3) + 180, 360) - 180;
```

Orientation Handling: Here, the code handles the robot's orientation. It calculates the angle difference (angleDiff) between the angle to the target in the global frame and the robot's orientation in the global frame (robotOrientation3). This angle difference is normalized to be within the range [-180°, 180°].

```
   % angle = angleToTargetGlobalFrame - robotOrientation3;  % Assuming the ro
bot's orientation in the 3rd component is the heading angle
    %angle = rad2deg(angle)
    angle = angleDiff;
```

Angle Result: The angle is assigned the value of angleDiff, which represents the desired angle of motion relative to the robot's frame.

```
    % Normalize the angle to the range [-pi, pi]
    angleNorm = atan2(sin(angle), cos(angle));
    angleNorm = rad2deg(angleNorm);
```

Angle Normalization: Finally, the code normalizes the angle to the range [-180°, 180°] using the atan2, sin, and cos functions. The result is stored in angleNorm, and its value is converted from radians to degrees.

This function is useful for determining the distance and angle between a robot and a target, which is crucial information for navigation and control algorithms in robotics.

Integration of Potential Field-Based Navigation

As part of system improvements, the initial solution has been enhanced by incorporating a potential field-based navigation approach. This addition augments the robot's obstacle-avoidance capabilities and provides a more robust navigation strategy.

Improved Navigation using Potential Field

calculateMovementDirection Function

```
function [movementDirection] = calculateMovementDirection(robotPos, targetPos
, sensorReadings, sensorAngles, K_att, K_rep, influenceRange)
    % Calculate Attractive Force
    attractiveForce = calculateAttractiveForce(robotPos, targetPos, K_att);

    % Initialize Net Repulsive Force
    netRepulsiveForce = [0, 0];

    % Calculate and Sum Repulsive Forces
    for i = 1:length(sensorReadings)
        distanceToObstacle = sensorReadings(i);
        repulsiveForce = calculateRepulsiveForce(distanceToObstacle, K_rep, i
nfluenceRange);
        angleRad = deg2rad(sensorAngles(i));
        netRepulsiveForce = netRepulsiveForce + [repulsiveForce * cos(angleRa
d), repulsiveForce * sin(angleRad)];
    end

    % Combine Attractive and Repulsive Forces
    totalForce = attractiveForce + netRepulsiveForce;

    % Determine Movement Direction
    movementDirection = atan2(totalForce(2), totalForce(1));
    movementDirection = rad2deg(movementDirection);
end
```

Explanation: The code snippet calculateMovementDirection represents a significant improvement in the system's navigation capabilities. It implements a potential field-based navigation approach, which offers enhanced obstacle avoidance and target attraction. Here's a breakdown of its functionality:

- **attractiveForce** is calculated using calculateAttractiveForce, which represents the force pulling the robot towards the target. This attractive force ensures the robot's tendency to move towards the desired destination.
- **netRepulsiveForce** is initialized as a vector [0, 0] to accumulate repulsive forces from obstacles.
- For each proximity sensor reading, the code calculates a repulsive force using calculateRepulsiveForce, considering the distance to the obstacle, K_rep

(repulsion constant), and influenceRange (range within which repulsion is effective). The angle of the sensor reading is also taken into account.

- Repulsive forces are summed up in **netRepulsiveForce**, considering contributions from all sensors.
- The attractive and repulsive forces are combined into **totalForce**, which represents the net force acting on the robot.
- The **movementDirection** is determined by calculating the angle of **totalForce**, ensuring that the robot moves in a direction that balances attraction towards the target and repulsion from obstacles.

This potential field-based approach significantly improves the system's ability to navigate through dynamic environments while avoiding obstacles effectively. It provides a more adaptive and robust navigation strategy compared to the initial solution.

Conclusion

Summary of Achievements

This project has successfully developed a control system that enables a robotic vehicle to navigate towards a target while effectively avoiding obstacles. The integration of Potential Field, PID control and obstacle avoidance logic allows the robot to handle dynamic scenarios and achieve its objectives.

Links of the videos:

Mobile Robotics