OBSTACLE AVOIDANCE

Project Report

EKLAVYA MENTORSHIP PROGRAMME

At

SOCIETY OF ROBOTICS AND AUTOMATION, VEERMATA JIJABAI TECHNOLOGICAL INSTITUTE, MUMBAI

JUNE 2020

ACKNOWLEDGMENT

We wish to express our deep gratitude and sincere thanks to our mentors, Sanath Menon, Neha Kurian and Maunil Shah for their guidance and constant supervision as well as for providing necessary information regarding the project. We would also like to thank Vedant Paranjape and Dhruva Golhe for their support in completing the project. Special thanks to Sravan Chittupalli and Omkar Bhilare for their support at our initial stages.

We are highly indebted to the whole team of Eklavya and SRA for giving us this wonderful opportunity which helped us in doing a lot of research and opened doors of many new things for us.

-Parvathy Nair

9096888430

parvati2109@gmail.com

-Sanika Gadge

9096338836

sanikagadge15@gmail.com

-

Saurabh Powar

7040730350

saurabhpowar1823@gmail.com

TABLE OF CONTENTS

Overview

The project is a very basic version of the latest self-driven/ autonomous car technology. At the basic levels, simple obstacle avoidance and goal to goal navigation is achieved. Detecting the environment around it, the car takes necessary actions. Meanwhile, it orients itself in the direction of the target position after every turn or deviation.

Traversing the robot from any position to a goal position, avoiding the spontaneous obstacles in its path has been our main objective. However, advanced functions like pre-calculation of path and generation of new path (path planning) have not been included as this project is a very basic attempt.

Introduction

Robotics is a part of today's communication. In today's world robotics is fast growing and increasing field. It is the simplest way for the latest technology modification. Obstacle avoidance is one of the most important aspect of mobile robotics. Without it, robot movement will fragile and restrictive. Obstacle avoidance is a robotic discipline with the objective of moving vehicles on the basics of the sensorial information.

In robotics, obstacle avoidance means automatic sensing of the environment and avoiding collision with objects in the environment. This concept leads to the development of autonomous robots. Being the primary and most important step, it requires the integration of many sensors according to this task. The robot gets the information from the surrounding through these sensors and accordingly the movement is made. The ultrasonic sensor is most suitable for obstacle detection as it is of low cost and has a high ranging capability.

For an autonomous mobile robot performing a navigation-based task in a vague environment, to detect and to avoid the encountered obstacles is an important issue and a key function for robot body safety as well as for the task continuity. Obstacle detection and avoidance in real world that appears so easy to humans, is a rather difficult task for autonomous mobile robots.

Software stack

- Coppeliasim (VRep)
  - ❖ Lua programming (regular API)
  - ❖ Model imported: Robotnik summit XL

OBSTACLE AVOIDANCE ALGORITHM

In robotics, obstacle avoidance is the task of satisfying some control objective subject to non-intersection non-collision position constraints. For this project we have designed a basic model of a self-driving car whose one of the main objective is obstacle avoidance.

The basic idea that we have followed for our project is that our bot should take a turn as soon as it senses any obstacle. As a first step to sense the obstacle we have used an Ultrasonic Sensor i.e. a Proximity Sensor. The basic principle of a Ultrasonic sensor is that it sends out an ultrasonic pulse and receives a pulse back. Using the time difference between the sent and receiving pulse the distance between the sensor and the object detected can be determined. For accuracy purposes and to be on more of a safer side in this bot we have used a total of 3 proximity sensors covering an environment of 180 degrees around the bot so that we can know in which direction exactly the obstacle is sensed and the robot can take an informed turn accordingly.

ALGORITHM AND CODE

For this project we have used the recommended language by the Coppeliasim software the Lua language.

Here onwards we shall be explaining the Lua functions that we have used in this code and their purpose.

Object Handle

In order to refer to the different parts of the Robot conveniently in code we create different object handles at the very beginning of the code. In lay man's language object handle is the name given to the any object present in the simulation environment in the Lua code.

The function can be used to create any object handle like for example Proximity Sensors, joints etc.

For example-

Robot = sim.getObjectHandle('Robotnik_Summit_XL')

Here "Robot" is the object handle created and the parameter in the function sim.getObjectHandle is the name of the robot in the simulation environment.

Read Proximity Sensor

This function is just to inform whether the sensor has or has not sensed any object around the environment.

The function is as follows-

result=sim.readProximitySensor(Sensor)

The function returns 1 or 0 if it senses any obstacle or if it doesn't respectively. Therefore, result is a value either 1 or 0. The parameter in the function sim.readProximitySensor is the object handle created for the Proximity Sensor.

Target Velocity of the Joints

In the Coppeliasim software, in order to power the wheels we have got to enable motor of the joint. The type of joint that we have used in this bot is a Revolute Joint. Basically what revolute joint does is it helps enable the expected circular motion of the wheel and also has motor which powers the wheel which we can control according to our convenience especially while controlling the turns which we will discuss in detail further.

Now as now we know what a revolute joint does, we use a function provided by the Lua language to set the velocity (linear or angular depending upon the type of joint) of the joints. As revolute joint is a non-spherical joint, we use this function to set the linear velocity of the joint.

The function is as follows-

sim.setJointTargetVelocity(motorHandle,-2)

The first parameter in the function sim.setJointTargetVelocity is the object handle of the joint for which we need to set the velocity and the second parameter is the desired velocity (in this case linear).

Turns:

Monitoring sensor readings and taking turns is the major task. When obstacles are encountered, three sensors placed at the front of the robot give values 0 or 1 (0 when obstacle is out of range of the sensor and 1 when it is in the range)

Sensor readings Right Middle Left	Position of the obstacle	Movement/turn
0 0 0	No obstacle	Straight
1 1 1	Obstacles on all sides (180 degrees)	Stop/180 degrees turn
1 0 0	Obstacle on right	Left turn
1 1 0	Obstacle on right and front	Left turn
0 0 1	Obstacle on left	Right turn
0 1 1	Obstacle on left and front	Right turn
0 1 0	Obstacle in front	Default left turn
1 0 1	Obstacle on either side	Default left turn

- Powering wheels for Movement/turns:
  - Straight – When left wheels are set to positive non-zero velocities and right wheels to negative velocities, the car goes straight.
  - Left turn – By setting target velocities of front and back left wheels to 0 and velocities of both the right wheels to a non-zero value, the car takes a left turn.
  - Right turn - By setting target velocities of front and back right wheels to 0 and velocities of both the left wheels to a non-zero value, the car takes a right turn.
  - 180 degrees left turn – If the target velocities of both left wheels are set to a negative value and that of the right wheels are set to the same positive value, the car takes 180 degrees turn from the left side.

- 180 degrees right turn – If the target velocities of both right wheels are set to a negative value and that of the left wheels are set to the same positive value, the car takes 180 degrees turn from the right side.
- Reverse– When right wheels are set to positive non-zero velocities and left wheels to negative velocities, the car goes straight.
- Rotation about own axis along left side – Setting target velocities of all wheels to a negative value.
- Rotation about own axis along right side – Setting target velocities of all wheels to a positive value.

GOAL TO GOAL NAVIGATION

- GOAL TO GOAL NAVIGATION

Goal to Goal Navigation is basically moving a bot from one position to desired position that can be achieved as follows-

- ORIENTING THE BOT TOWARDS THE GOAL

First of all, we get the initial position co-ordinates of the bot as well as co-ordinates of the Goal. Then we find the angle by which robot should turn and orient itself towards the goal. It can be done by getting tangent of the increment in x and y co-ordinates. Formula used for getting angle is:
If Initial Position Co-ordinates are (x_1, y_1, z_1) and Goal Position Co-ordinates are (x_2, y_2, z_2) then Theta can be found out by formula

$$\textbf{Theta} = \tan^{-1}(y_2-y_1/x_2-x_1)$$

Then we rotate bot by angle theta as follows, we find the time required for bot to rotate around itself in 360 degrre . Then we rotate the bot through angle Theta by calculating timing required for that.

Time of Rotation = (Time required to rotate 360^0 * Theta) / 360^0

Turning the bot to the desired angle was done in the same way as explained previously.

- MAINTAINING THE SAME DIRECTION AFTER A TURN

Whenever bot detects obstacle using Proximity Sensors it will change its direction according to the position of obstacle in front. In order to maintain initial direction, we have to recover the direction of bot by rotating it in opposite side through the same angle it rotated when it had detected obstacle. So that it will again orient towards the goal. For example, if obstacle comes at left side of bot then it will rotate itself though certain angle Alpha towards right then bot goes straight for some time then again it recovers direction by rotating itself towards left by same angle Alpha.

Ref: https://github.com/Git-Saurabh5/Team-Kerbecks-and-PS/blob/master/ObstacleAvoidance%20andNavigation.ttt

ALGORITHM AND CODE

Here we will explain the functions that we have used in the Lua code in order to achieve goal to goal navigation.

OBJECT POSITION

This function is used to get the present coordinates of the provided object handle with respect to the desired frame of reference. The function is as follows-

position = sim.getObjectPosition(Robot,-1)

Here position has stored 3 values that are returned by the sim.getObjectPosition i.e. the x, y, z coordinates of the passed object handle of the Robot i.e. the first parameter. The second parameter -1 represents that the coordinates are returned with respect to the absolute or the parent frame of reference.0

WAIT

The function preceding this function works for the specified time in the wait function. The function is as follows-

sim.wait(4)

The parameter passed in the sim.wait function is the number of seconds we want the preceding task to work before the proceeding task is executed.

TANGENT

If we have coordinates of two points. Then the angle made by the line passing through these two points with the positive direction of X-axis is determined by this function. The function is as follows-

angle = math.atan2(y2-y1,x2-x1)

EXPERIMENTAL RESEARCH AND ANALYSIS

✓ General

- For an autonomous car, sensors are most important and thus the number of sensors mounted on the car as well the pattern of sensors are factors which are to be taken into close concern.
- Various changes were made in the course of project regarding different sensors. These are as follows:
  - 3 sensors placed on the front side of the car: Uses minimum number of sensors and effectively avoids obstacles. Cost effective.
    Ref: https://github.com/Git-Saurabh5/Team-Kerbecks-and-PS/blob/master/final.ttt
  - 3 sensors placed on the front side of the car, 2 on either side: Uses more number of sensors and avoids obstacles much more effectively, protecting the sides of the vehicle. Not much cost effective as it uses more number of sensors.
    Ref: https://github.com/Git-Saurabh5/Team-Kerbecks-and-PS/blob/master/sensor5.ttt
  - 1 sensor placed on a rotating platform placed at the front of the car (this rotating platform performs angular simple harmonic motion): Very cost effective but obstacle avoidance is not up to mark. Not reliable.
    Ref: https://github.com/Git-Saurabh5/Team-Kerbecks-and-PS/blob/master/rotate.ttt

Based on all these observations we concluded that we would use 3 sensors in the front of the car.

We also tried making the car encounter a C curved obstacle.

Ref: https://github.com/Git-Saurabh5/Team-Kerbecks-and-PS/blob/master/CcurveObstacle.ttt

Check out this YouTube video to see a demo of all the above results :-
https://youtu.be/kudX5SUs7Vo

✓ Shortcoming

Main idea of our goal to goal navigation was to orient the robot in the direction of the target each time it takes a turn or undergoes a deviation. Initially, before encountering any obstacle, the robot orients itself perfectly in the direction of the target. It works satisfactorily for 1 or 2 small obstacles in its path kept at sufficient distance from each other. But with more obstacle kept in its path close to each

other, the robot does not get enough time to orient itself after encountering one obstacle. Thus, there is a certain error arising in the orientation angle and the robot ends up deviating a little from the orientation of the target.

CONCLUSION AND FUTURE WORK

Having achieved our main objective i.e. obstacle avoidance and partially the goal to goal navigation, we would like to conclude saying that our curiosity and research does not end here. Further we would like to study path planning and successfully implement it.

Going forward, we would wish to achieve objectives like pre-calculation of obstacle-free path, changing path according to static and dynamic objects and successfully reaching and stopping at the correct position with desired orientation.

Implementing the same on ROS platform with much more convenience and libraries like OMPL have to be studied.

REFERNCES

- www.coppeliarobotics.com
- https://www.coppeliarobotics.com/helpFiles/en/apiFunctions.htm
- https://forum.coppeliarobotics.com