Development of Automated Cleaning and Mopping Robot

Taslima Yeasmin Emu
ID: 2016-1-60-038

Sayma Obaida
ID: 2016-1-60-033

A thesis submitted in partial fulfillment of the requirements for the degree of Bachelor of Science in Computer Science and Engineering

Department of Computer Science and Engineering
East West University
Dhaka-1212, Bangladesh

October, 2020

Declaration

We, Taslima Yeasmin Emu and Sayma Obaida hereby, declare that the work presented in this thesis titled 'Development of Automated Cleaning and Mopping Robot' is outcome of the investigation performed by me under the supervision of Dr. Mohammad Salah Uddin, Assistant Professor, Department of Computer Science and Engineering, East West University. I also declare that no part of this thesis has been or is being submitted elsewhere for the award of any degree or diploma.

………………………………...	……………………………….
Dr. Mohammad Salah Uddin	Taslima Yeasmin Emu
Supervisor	ID: 2016-1-60-038
	………………………………
	Sayma Obaida
	ID: 2016-1-60-033

Letter of Acceptance

I hereby declare that, this thesis is the students own work and best effort of mine. All other sources of information used have been acknowledged. This thesis has been submitted with my approval.

....

Dr. Mohammad Salah Uddin
Assistant Professor
Department of Computer Science and Engineering
East West University
Aftabnagar, Dhaka-1212, Bangladesh

Supervisor

....

Dr. Taskeed Jabid
Associate Professor
Department of Computer Science and Engineering
East West University
Aftabnagar, Dhaka-1212, Bangladesh

Chairperson

Abstract

Autonomous Vacuum Cleaning Robot is designed to assist humans in accomplishing their cleaning task. Today’s busy lifestyle has led to an increased demand for automated vacuum cleaning robots. An autonomous vacuum cleaner has intelligent programming and cleaning system to clean a specific area. So, the vacuum cleaner robot has to follow a method to solve the cleaning task. The main focus of this project is to develop an AI based vacuum cleaning robot which is capable of detecting obstacle, avoiding obstacle and can complete its task efficiently. That’s why, we build a coverage path planning algorithm with less complexity that gives efficient performance to complete the cleaning task. For robot building and simulation, simulation tool is used. The robot is rectangle shaped with four sensor, two wheels and two castor wheels. Snake algorithm is improved and implemented to build the robot. The result shows that proposed algorithm gives better performance with lower run time.

Acknowledgments

First of all, We would like to express my deepest gratitude to the almighty Allah for His blessings on us. Next, our special thanks go to our supervisor, “Dr. Mohammad Salah Uddin”, who gave us this opportunity, initiated us into the field of “Artificial Intelligence”, and without whom this work would not have been possible. His encouragements, visionaries and thoughtful comments and suggestions, unforgettable support at every stage of our BS.c study were simply appreciating and essential. His ability to muddle us enough to finally answer our own question correctly is something valuable what We have learned and We would try to emulate, if ever We get the opportunity.

There are numerous other people too who have shown me their constant support and friendship in various ways, directly or indirectly related to our academic life. We will remember them in our heart and hope to find a more appropriate place to acknowledge them in the future.

Taslima Yeasmin Emu
October, 2020

Sayma Obaida
October, 2020

Table of Contents

List of Figures

List of Tables

Chapter 1
Introduction

The invention of robots brought a massive change in our living standards, our way of thinking and our activities. [1] With the advancement of technology, robotic cleaners have taken major attention in robotics research to make life of mankind comfortable.

1.1 Background

Nowadays autonomous floor cleaning robot applications has significant development and majorly used in industrial manufacturing, office spaces for cleaning purposes. An autonomous robotic vacuum cleaner has intelligent programming to clean a specific area through a vacuum cleaning assembly.[4] Robotic cleaners has the features like floor mopping, dry vacuum cleaning etc. Autonomous floor cleaning robot can clean inside those places easily which are dangerous for human to cleaning like fuel, gas tanks area. There are various autonomous cleaning robot such as iRobot, Robo Vac, Botvac, Trilobite are the tested robot in the market.The first robotic vacuum cleaner appeared in 1996 called “Trilobite”. It was made by the Swedish corporation Electrolux. With the help of ultrasonic sensors, it can map rooms and avoid obstacles.[5]
A widely successful domestic robot for vacuuming floors is a first-generation “Roomba”. In 2002, iRobot launched the Roomba, created by MIT roboticists Colin Angle, Helen Greiner and Rodney Brooks. This battery-operated Roomba rolls on wheels. It reacts to its environment with the help of computer processing and sensors. When any dirt hits the sensors then the robot turns towards the dirty area and clean the area.[6]
In 2006, iRobot (KITECH, Sony) launched “Braava”cleaning robot that has the capability to floor mopping for hard surfaces or dry clean.

In 2010, Neato-Robots XV series (California) launched “Neato XV-11”cleaning robot. It can complete the vacuum cleaning task with the help of laser range finder technology.[7]

Autonomous vacuum cleaning robot makes cleaning an easy and time efficient task and also gives comfort to the human by saving energy. Now in Covid-19 pandemic, autonomous floor cleaning robot is a much needed for cleaning and disinfecting. In 2015, 28 new viruses (1495AC and 12,985BC) were discovered in a melting glacier and many new viruses can be released due to global warming in future which can be the cause of future pandemic.[8] It’s not impossible to avoid the spreading of the virus with the help of automation cleaning. Automated multi-robot deployment can play a vital role to decrease the spreading of virus as multi-robot is more efficient rather than single robot to task completion and to facilitate human activities. [9] From this learning we can imagine the importance of automation cleaner. In this report, we are proposing an autonomous vacuum cleaning robot which can do the floor cleaning task completely. Robotic Operating System (ROS) platform is used to build this robot and simulated it in 3D environment by using Gazebo.[10]

1.2 Motivation

Cleaning the floor from dust is one of the activities which is an iterative process and required on daily basis consuming both time and energy. People can place the autonomous robot at any area to complete the cleaning activity. Now in COVID-19 pandemic, when human interaction can create big threat so in places like hospitals, offices, daily shops are much needed to clean and disinfect all the time. To clean and disinfect without human interaction vacuum cleaning robot is the most needed. "Demand for hospital robots and floor cleaning robots are likely to grow exponentially (3000-4000 per cent) over the next four-five years," said by Rajeev Karwal, founder Chairman of Milagrow Robots.[11] That’s why we develop a autonomous floor cleaning robot. So, we developed a virtual autonomous floor cleaning robot model using simulation tool which can do the cleaning task completely in an effective way and proposed an elementary algorithm which gives efficient cleaning.

1.3 Problem Statement

Several industries came out with different ideas of cleaning robots and got popularity in developed countries. [12] Automated cleaning robots has various applications in-floor cleaning, pool cleaning

and so on. In (Prassler, Ritter et al. 2000) there is a taxonomy of cleaning robots presented and this taxonomy is based on the function of the robot namely robotic vacuum cleaners, sweepers an so on. The main requirement for a robotic vacuum cleaner is to fulfill its task according to its function.[13] The intelligent floor cleaning robot should know about its working region. That's why a model of the environment is needed for it.[14] The principle of our robotic vacuum cleaner is to clean an area by navigating in an environment without colliding into any obstacles. So, the cleaning robot must have the function of map building and path planning.

There are three different strategies that the vacuum cleaning robot can employ for its path planning stated by Lee & Banerjee [15] and these are given below:

1. A map can be used to divide a room into cells and then the algorithm can generate efficient path.
2. A template based strategy can be used to define path patterns like- wall-to-wall, zigzag, etc.
3. AI based methods can be used to generate adaptive paths.

The focus of our study will be combining on these 3 strategy. The vacuum cleaner robot has to follow a mechanism to solve the task of cleaning the entire environment areas with obstacle avoidance and the number of turns it needs to take.[16] The selection of path coverage algorithm plays a vital role in providing complete coverage of the area which is covered by the robot from a start position to the end position with avoiding obstacles. Mapping is an important task for the cleaning robot.[17] The Neato robot uses SLAM algorithm to map the room for navigation [18] .

Developed algorithms are also discussed in this report. In our project, we propose Snake Algorithm with improved version. A map is used to divide a room into cell according to the room and robot size. This path planning algorithm is implemented to achieve maximum cleaning efficiency. By following this algorithm, the robot can clean an area with obstacle avoidance.

1.4 Aim & Objective

The aim of this project is to design and develop a virtual robot that can clean the floor. The objective of this project is given below:

1) To design and develop a vacuum cleaning robot using path coverage approach which is capable of obstacle detection and obstacle avoidance.

With the project aim, the proposed algorithm and techniques are applied to design the desired robot.

1.5 Features of the proposed robot

Obstacle detection: Hokuyo Laser Scanner is used to detect Obstacles. Information about any objects position can be obtained from the mapping of the environment using Hokuyo Laser Scanner.

Obstacle avoidance: By following our proposed path coverage algorithm, the robot can avoid obstacle succesfully.

Vacuum cleaning task: The robot can managed its movement and clean the dirt successfully by following our proposed algorithm.

1.6 Organization of the book

The layout of rest of this report has been organized as takes after.

- The Chapter 2 presents the literature review on the topic of design and development of automatic cleaning and mopping Robot where all the previous path coverage algorithm related to this project are described.
- The Chapter 3 presents the building procedure of cleaning robot.
- The Chapter 4 explains the path planning procedure of cleaning robot
- The Chapter 5 analyze and discuss the experimental results found in our thesis.

- The Chapter 6 outlines conclusion of this work precisely and describes the scope of future work for possible extended application of the project.

Chapter 2

Literature Review

There are lots of mechanisms and algorithms have been researched for autonomous cleaning robot such as Grid-mapping, SLAM , zigzag , PPCR etc.

2.1 Related Work

Grid mapping (Gmapping) method is used in [6] the paper to create the map by using ROS framework. Gmapping has the function of simultaneous localization and mapping(SLAM). In Gmapping algorithm, the workstation connect to the base station. Roscore and Gmapping node start on base station. Keyboard, Teleoperation, Ros visualization run on workstation. Tele-operation is used to control the TurtleBot2 movements through RVIZ. The map was generated after optimizing the best GMapping parameters. The frontier Exploration is used to build the map but the generated map is not accurate because of movement of the object. To obtain the best map, almost 11 attempts and 20 minutes time is needed.[8]

A simulation example of 2D and 3D map generation in an environment is presented in the paper. Hector mapping package which realizes the SLAM (Simultaneous Localization and Mapping) algorithm is used to provide robot pose estimation. Octomap package is used to create a 3D map. For navigation, Adaptive Monte Carlo Localization (AMCL) package is used into the generated map. The map building based on SLAM method work well for indoor environments but using this method it is almost impossible to execute a realistic 3D map.[19]

Zigzag algorithm is used for the movement of the autonomous floor cleaner robot [8]. The robot has the capability to wipe, water Spray and sweep. By using this algorithm, the robot movement was testing in a 1.5 m^2 area. To complete the cleaning task, the robot took 55 seconds and the speed of the robot was 0.21 m/s.[20]

Improved Q-learning (IQL), Classical Q-learning (CQL) and Extended Q-learning(EQL) algorithms are presented in this paper for path planning. By comparing these algorithms, it is found that time taken by the robot in IQL is less than CQL and EQL. 90^0 turnings required by the robot in IQL is less than CQL and EQL. [21]

There are four different algorithms (Random walk, Spiral, ‘S’ shape pathway, Wall follow) are tested for path planning . Theoretically, random walk or ‘S’ shape or spiral algorithm can do the task to complete coverage of the area. But after plotting the acquired data in MATLAB, it is found that both algorithms are involved partial or complete randomness. When the robot executing these four algorithm continuously one after another (combined mode), the robot covers the entire area faster.[22]

Chapter 3

Building Procedure

The first phase of this project is to develop a virtual robot. And for running the robot, we need the environment. We used ros , gazebo and rviz for robot environment creating. Firstly ros, gazebo and rviz need to be installed for the environment, and then we need to create a catkin workspace for robot.

3.1 Environment Building

Catkin workspace is where we install, modify, and build catkin packages.
For setting up catkin workspace we need to source the path.
source /opt/ros/melodic/setup.bash
then create a folder called catkin_ws. In the folder create src folder where all the source code will be. The in terminal run catkin_make. Catkin_make build and run all the packages. Either use 'catkin_make' command or it can also be done manually. Then source your setup.bash file. The working environment is created.

We use robot description folder, gazebo folder and navigation folder(fig-3.1). Description folder include all the needed file folder and packages for building robot like xacro file for robot(fig-3.2). Gazebo folder include all the file for running in gazebo like world file(fig-3.3) and navigation folder include the files for navigating the robot like gmapping(fig-3.4).

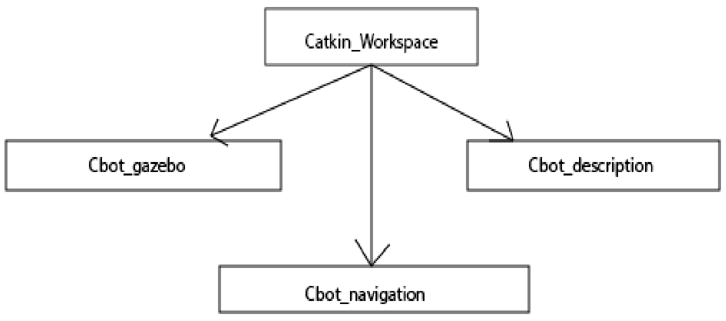


Figure 3.1 : formation of files (a)

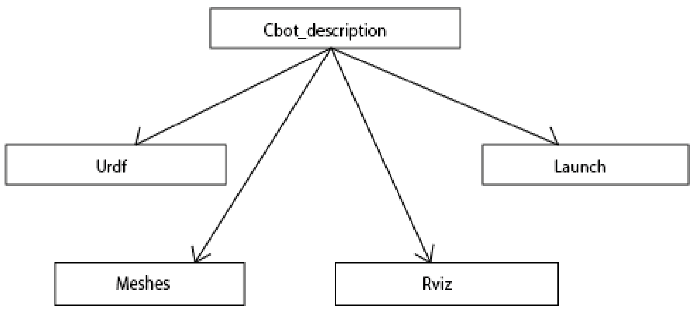


Figure 3.2 : formation of files (b)

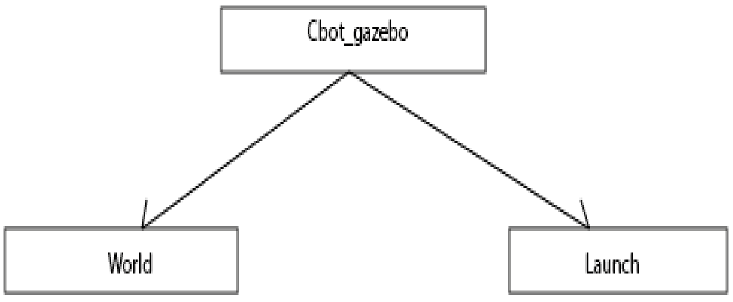


Figure 3.3: formation of files (c)

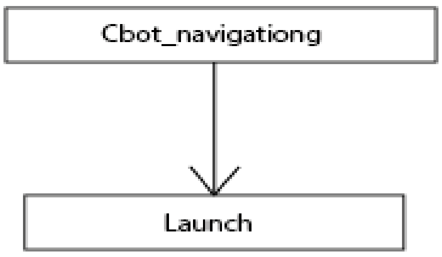


Figure 3.4 : formation of files (d)

3.2 Robot Building

Now create a folder in your src folder preference of your robot name, and create a package with folder. We used rviz, gazebo for showing the robot, tf for knowing the coordinate xacro, urdf for building the robot and c++ for algorithm and std_msgs for Standard ROS Messages and geometry_msgs for geometric primatives such as points, vectors, and poses. Run “catkin_create_pkg cbot_description tf rviz gazebo xacro urdf rospy roscpp std_msgs geometry_msgs” to create description folder with the package included.

Now create folder for robot, urdf, launch, meshes in description file. Urdf include all the xacro file. As we know xacro is a Xml language, the code should include xml version.

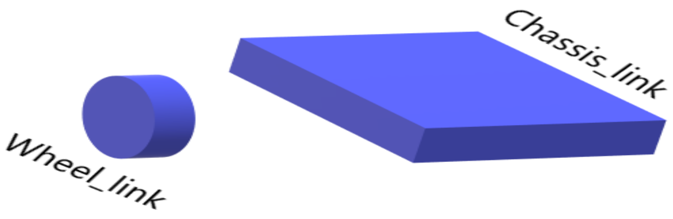


Figure 3.5: wheel & chassis

3.3 Wheel & Chassis Link:

We need 2 wheels, in rectengle box for chassis for robot base(fig-2.5). The robot has to be created link by link through joints. First xml version then robot name declaration then start of the robot model.

```
<?xml version="1.0"?>
<robotname="myfirst">
<linkname="base_link">
        //All the link code
</link>
<joint>
//joint description
```

```
</joint>
</robot>
```

The base of the robot is the rectangle shaped chassis where all the other properties will be connected. The basic writing style of all the link is first link name, then declaring visual name ,the geometry, geometry should include the size shape and geometrical description (if need any). The sample code for our chassis.

```
<linkname="base_link">
<visual>
<geometry>
        //boxsize
</geometry>
</visual>
</link>
```

But we also need inertial and collision to properly describe our chassis.

```
<linkname="base_link">
<inertial>
  ………
</inertial>
<visual>
  ………
</visual>
<collision>
  ……
</collision>
</link>
```

Inertial describe center of the gravity, visual describe the visual properties of the link and collision describe the collision property.

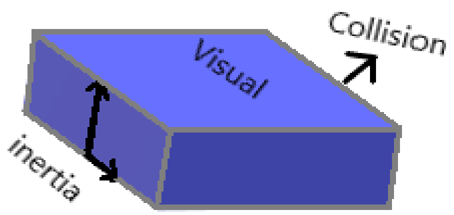


Figure 3.6: chassis building

Every inertial, visual and collision need origin. The origin is described by first xyz point and secondly rpy which means roll , pitch and yaw. Roll is applied first about the X-axis of the previous joint. The frame you're modifying will rotate X radians about the previous X-axis. Pitch is applied next about the Y-axis of the previous joint. Yaw is applied last about the Z-axis of the previous joint.

The inertial need inertia description. Inertia is standard dimension, Inertia is described by 6point in term of x,y,z axis. The points are xx,xy,xz,yy,yz,zz. To identify its inertia point its written by ixx, ixy,ixz, iyy,iyz, izz. Inertial also should include mass value.Visual implement the points of visual representation. The origin, geometrical description and materials like texture and colors. Collision also included the origin, geometrical description. The difference of collision and visual is that visual gives visual representation and collision point represent where the collision will happen, like collide with other joints or object. After the chassis building wheel building also will be in the same process.

```
<linkname="left_back_wheel">
<inertial>
   ………
</inertial>
```

```
<visual>
   ………
</visual>
<collision>
   ……
</collision>
</link>
```

This is the basis of building a xml file format to design a robot car model.

In our project we used two wheel left and right and to castor wheel with the same process. Castor wheel collision is a little more descriptive.

```
<collision name="caster_collision">
```

```
<origin xyz="-" rpy=""/>

<geometry>

<sphere radius= ""/>

</geometry>

<surface>

<friction>

<ode>

<mu>””</mu>

<mu2>””</mu2>
```

```
<slip1>””</slip1>

<slip2>””</slip2>

</ode>

</friction>

</surface>

</collision>
```

When two object collide, such as a ball rolling on a plane, a friction term is generated. In ODE this is composed of two parts,

mu - the Coulomb friction coefficient for the first friction direction.

mu2 - the friction coefficient for the second friction direction.

When robot start and stop is get slipped, to control it slip value is included in castor collision, so that runs smoothly. Otherwise the robot will slip with its own calculated way which create position difference.

3.4 Wheel & Chassis Joint

After the building of 2 wheel and 2 castor wheel, Chassis and wheels joint need to be implement.

```
<jointname=""type="">
<parentlink="base_link "/>
<childlink="wheel1"/>
<originxyz=""/>
</joint>
```

Implementing joint is a simple procedure. First joint name, then it’s type. The joint can be fixed, like car base and seats. Or It can be continuous like wheels with car. Then the parent link and child link. Parent link is the base link or where it would be connected and child link is itself. And after describing origin of the connection the joint declaring is complete. It have to be done for 4 wheels and also every joint like came, sensors.

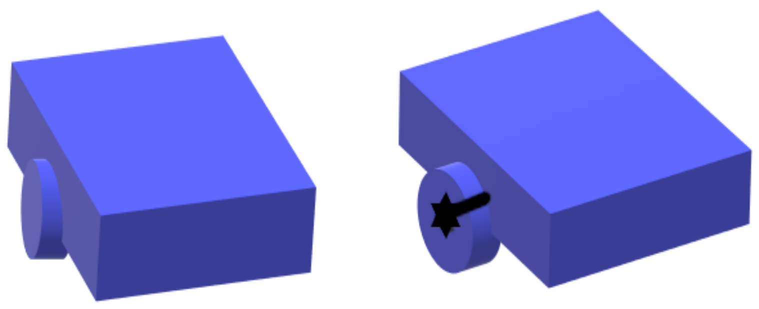


Figure 3.7 : joint wheel & chassis

The first figure is before implementing joint and the second figure is after implementing joint. Though we can’t see it in the visual representation, but its works like nut and bolts.

The chassis size of our robot is 0.4, 0.2 and 0.1, The mass of the chassis is 15. The collision size is same as box size. Each wheel mass is 5 and the radius is .1 and length is 0.05 as it is a cylinder. And collision rpy is 0, 1.57 and 1.57.

3.5 Scanner Connection

For scanner setting we used Hokuyo laser. The link building for laser is the same, but for visual meshing .dae file is used in mesh as hokuyo.dae.

```
<link name="hokuyo_link">

<visual>
```

```
<geometry>

<mesh filename="package://$PATH"/>

</geometry>

</visual
```

Also laser need to be join through the same processed described

earlier.

```
<joint name="hokuyo_joint" type="fixed">

<axis xyz="0 1 0" />

<origin xyz="0.15 0 0.1" rpy="0 0 0"/>

<parent link="chassis"/>

<child link="hokuyo_link"/>

</joint>
```

For the visual file is already universally accessible through gazebo model. Its only need to be copied and connect it to visual link.

Gazebo: To get access the robot with camera, lasers in gazebo which transfers data we need to add gazebo plugins. The plug_in code is also universally accessible. Overall the plugin include <gazebo>

<pluginname=""filename="">

... plugin parameters ...

</plugin>

</gazebo>

For connecting we just need to add the link name In referece.

<gazeboreference="hokuyo_link">

In laser only in the topic name it should include own the path. Our robot name is cbot.

<topicName>/cbot/laser/scan</topicName>

We also used drive controller pluging to control the robot’s drive. Defining dirve plugin is a tricky one. After declaring the plugin name

<pluginname="differential_drive_controller"filename = "libgazebo_ros_diff_drive.so">

Legacy mode need to be declared , without declaring it the setting assumes, the right and left wheel are changed. That’s why legacy mode need to declared false.

<legacyMode>false</legacyMode>

To access the driver always the alwaysOn should set as true.

<alwaysOn>true</alwaysOn>

The update rate depends on how fast the plugin need to be updated. We used 10hz for update.

```
<updateRate>10</updateRate>
```

The wheel joint need to be connected.

```
<leftJoint> left_wheel_hinge </leftJoint>

<rightJoint> right_wheel_hinge</rightJoint>
```

Wheel separation indicate the distant from one wheel to other.

```
<wheelSeparation>0.170</wheelSeparation>

<wheelDiameter>0.140</wheelDiameter>
```

Maximum tourque wheel can produce.

```
<torque>36.9</torque>
```

The reference for chassis and wheel also is needed.

```
<gazebo reference = "chassis">

</gazebo>

<gazebo reference = "left_wheel">

</gazebo>
```

```
<gazebo reference = "right_wheel">

</gazebo>
```

Color can also be added with creating a different material.xacro file and define the color there

```
<gazebo reference = "chassis">

<metarial>Gazebo/Blue</metarial>
```

```
</gazebo>
```

To define a color in materials file color need to be declared at rgb format.

```
<?xml version="1.0"?>

<robot>

<material name="Blue">

<color rgba="70 229 221 1.0"/>

</material>

</robot>
```

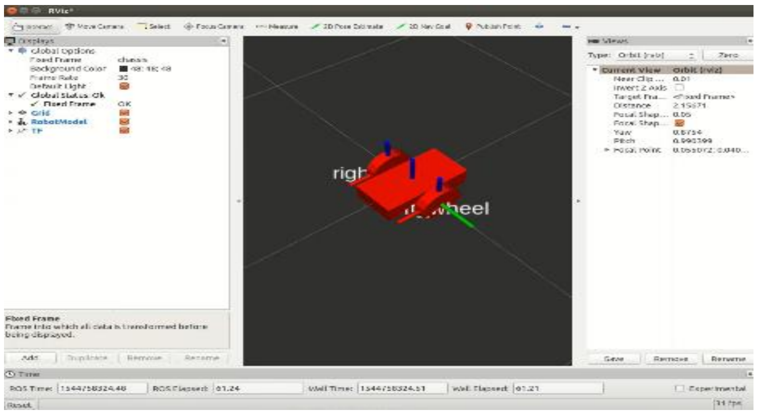
Every launch file should start by defining the xml version. And also launch statement.

```
<?xml version="1.0"?>

<launch>

...........................

</launch>
```

RVIZ Launch: For launching the robot in rviz we need a ros launch file. For that we will create a launch folder and create launch file in that folder. Launch file also used Xml language.

```
<launch>

<param name="" command=""/>

<node pkg="" type="" name="">

.....
```

```
</node>

<node pkg="" type="" name="">

</node>

<node pkg="" type="" name="">

</node>

</launch>
```



```
Global Options
  Fixed Frame         chassis
  Background Color    48; 48; 48
  Frame Rate          30
  Default Light
Global Status: Ok
  Fixed Frame         OK
Grid
RobotModel
TF

Fixed Frame
Frame into which all data is transformed before being displayed.

Type: Orbit (rviz)        Zero
Current View        Orbit (rviz)
  Near Clip ...     0.01
  Invert Z Axis
  Target Fra...     <Fixed Frame>
  Distance          2.15671
  Focal Shap...     0.05
  Focal Shap...
  Yaw               0.6754
  Pitch             0.995399
  Focal Point       0.055072; 0.040...

ROS Time: 1544758324.48   ROS Elapsed: 61.24   Wall Time: 1544758324.51   Wall Elapsed: 61.21
```

Figure 3.8 : launching the robot in rviz

Here param name is robot description name, command contain the command and path , for launch the command is to find xacro file and following the given path. Node contain the needed state to publish and also the package name where to publish, like joint_state_publisher, robot_state_publisher. This launch file is to launch rviz that why one package will be rviz. The type and name also contain the the package type and name.

3.6 Gazebo Launch

The robot need to be launched in a world at gazebo. The default world is empty_world. There is many world given default. Custom world also can be created. To launch in empty world, the world need to included with path default way.

<include file="$(find gazebo_ros)/launch/empty_world.launch">

After launching the gazebo file in empty World

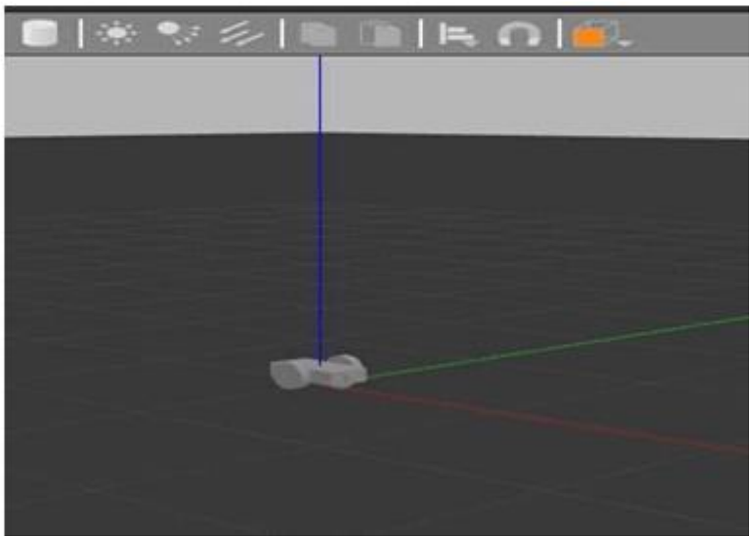


Figure 3.9: Gazebo Visualization

After custom world creating custom world can be called with world name and path,

<arg name="world_name" value="$(find spcbot_gazebo)/worlds/room.world"/>

Paused argument is used to start Gazebo in a paused state. Where default value is false.

<arg name="paused" value="false"/>

Ask node to get Gazebo-published simulation time, published over the ROS topic .

<arg name="use_sim_time" value="true"/>

Also if recording is needed recording argument value can be turned true.

<arg name="recording" value="false"/>

To convert an xacro and put on parameter server

<param name="robot_description" command="$(find xacro)/xacro.py $(find name)/PATH" />

Spawn a robot into Gazebo

<node name="spawn_urdf" pkg="gazebo_ros" type="spawn_model" output="screen"

args="-param robot_description -urdf -model ROBOT_NAME" />

Chapter 4 Methodology

The Vacuum cleaner robot should have a path planning mechanism to sweeping every accessible area with avoiding obstacles in the entire room environment. There are lots of algorithms have been researched for autonomous cleaning robot-

Chapter 4.1 Algorithm Description

Genetic Algorithm: Genetic algorithm has mainly three steps: fitness evaluation, selection and reproduction.[23] This algorithm follows a population of candidate solutions. Each candidate solution is called a chromosome. This candidate solution is usually coded as binary string. A set of chromosomes forms a population. The initial population is generated at random. A population is evaluated and ranked by fitness evaluation function. In this algorithm, three operators are used: reproduction, crossover and mutation. Reproduction is used to the population. The string makes a number of copies proportional to their own fitness and results in an intermediate population. Then select parents from the current population so that better chromosome are selected. This is achieved by the fitness value. After that, Reproduce "children" which is new string from selected parents using crossover. The algorithm is terminated after founding the acceptable solution.[24]

Spiral algorithm:By following this algorithm,the robot can create an increased circle. At first, the robot checks if there is enough place to start moving spirally. If yes then the robot turn in left hand side direction and increasing radius from center point till an obstacle is sensed. The algorithm is terminated when the obstacle is sensed.[25]

'S' shape algorithm: This algorithm is following a route map like the letter 'S'. By using this algorithm, the robot can cover the entire room area in fastest way. The robot turning his direction when it is found any obstacle. After founding every collision, the robot has a series of movements - Back ,90^0 turn Right or Left ,Go,90^0 turn Left or Right.[26]

4.2 Proposed Algorithm

In this paper, we propose a approach based on snake algorithm which consist of several steps.

To get efficiency in path planning, we used snake algorithm with dynamic approach. This algorithm is based on calculating and comparing neighbor values with snake algorithm to find best possible path.

All the node data will be initially null. The obstacle will be detected by laser scan. Visiting same node will be detected as collision and will be treated as obstacle.[27] After starting the process, robot will go forward in facing direction, detect neighbor and assign neighbor node data as 0.The node that the robot has visited will be increased by one with the previous value of node. Let, visited node as V. V(data) = V(data)+1

Smaller block: If robot facing is on x-axis, it will divide the area into +y- axis and -y-axis and calculate both neighbor's descendant in that area. And select neighbor with smaller descendent; If robot facing is on y-axis, it will divide the area into +x-axis and -x-axis and calculate both neighbor's descendant in that area. And select neighbor with smaller descendant. There are three cases in the algorithm :

Turn 90^0 left or right: If robot is facing x or y direction and found obstacle then it will turn 90^0 left or right. Here, assuming the robot is facing y direction and found obstacle. So, it will turn 90^0 left or right in x directon.

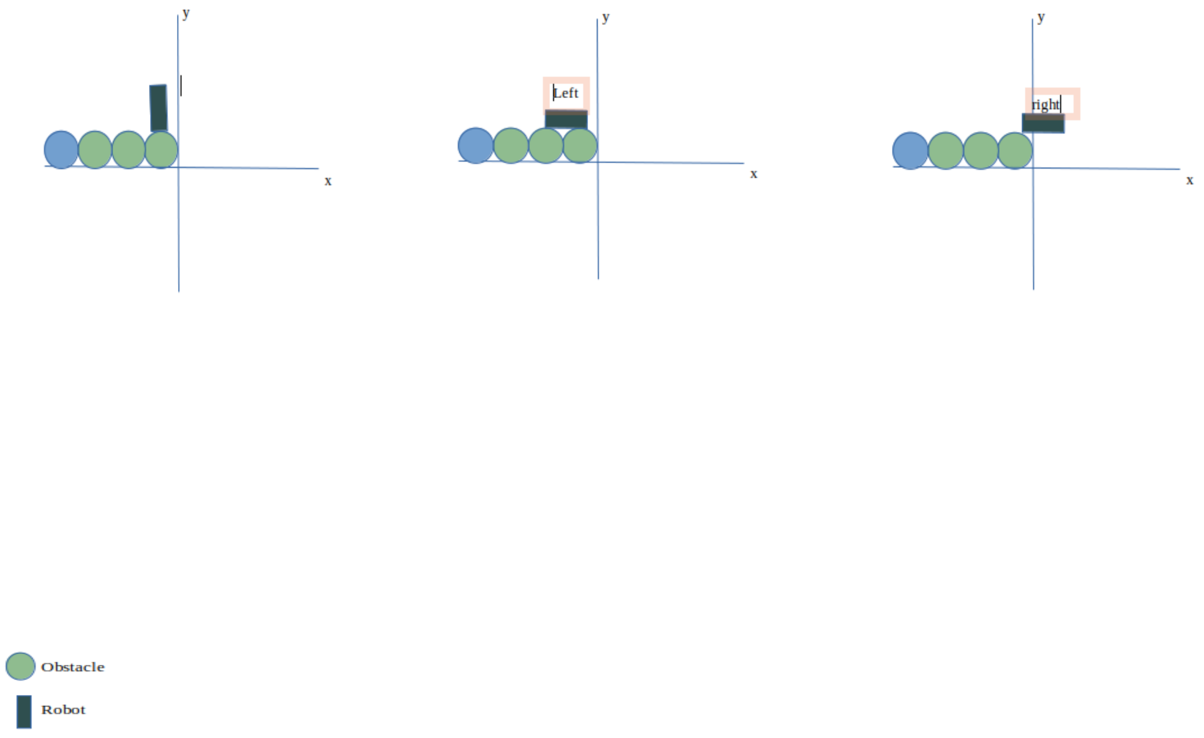


Figure 4.1 : 90^0 left or right turning

Select minimum valued neighbor: After founding obstacle, robot will turn 90^0 left or right in x directon and select the neighbor which has minimum value. Here, after turning 90^0 the minimum valued neighbor 0 is found in right direction. So the robot turn to right x direction.

If the neighbor has the same value then it will select smaller distant block.

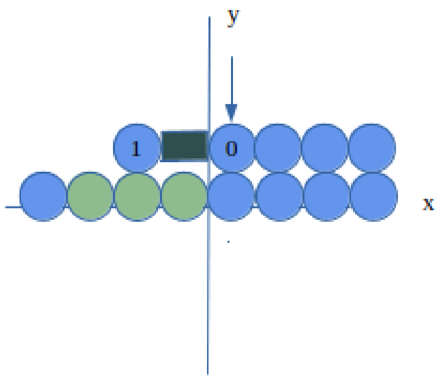


Figure 4.2 : select minimum valued neighbor

Go forward or backward: If the robot can not find one minimum valued neighbor or same valued neighbor then it will search minimum valued neighbor forward. And if minimum valued neighbor forward is not found then it will go back to the previous neighbor.

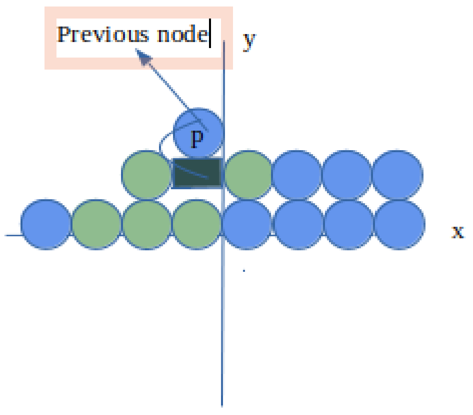


Figure 4.3 : Select previous node

4.3 Coverage Path Planning

The flowchart of the algorithm is given below:

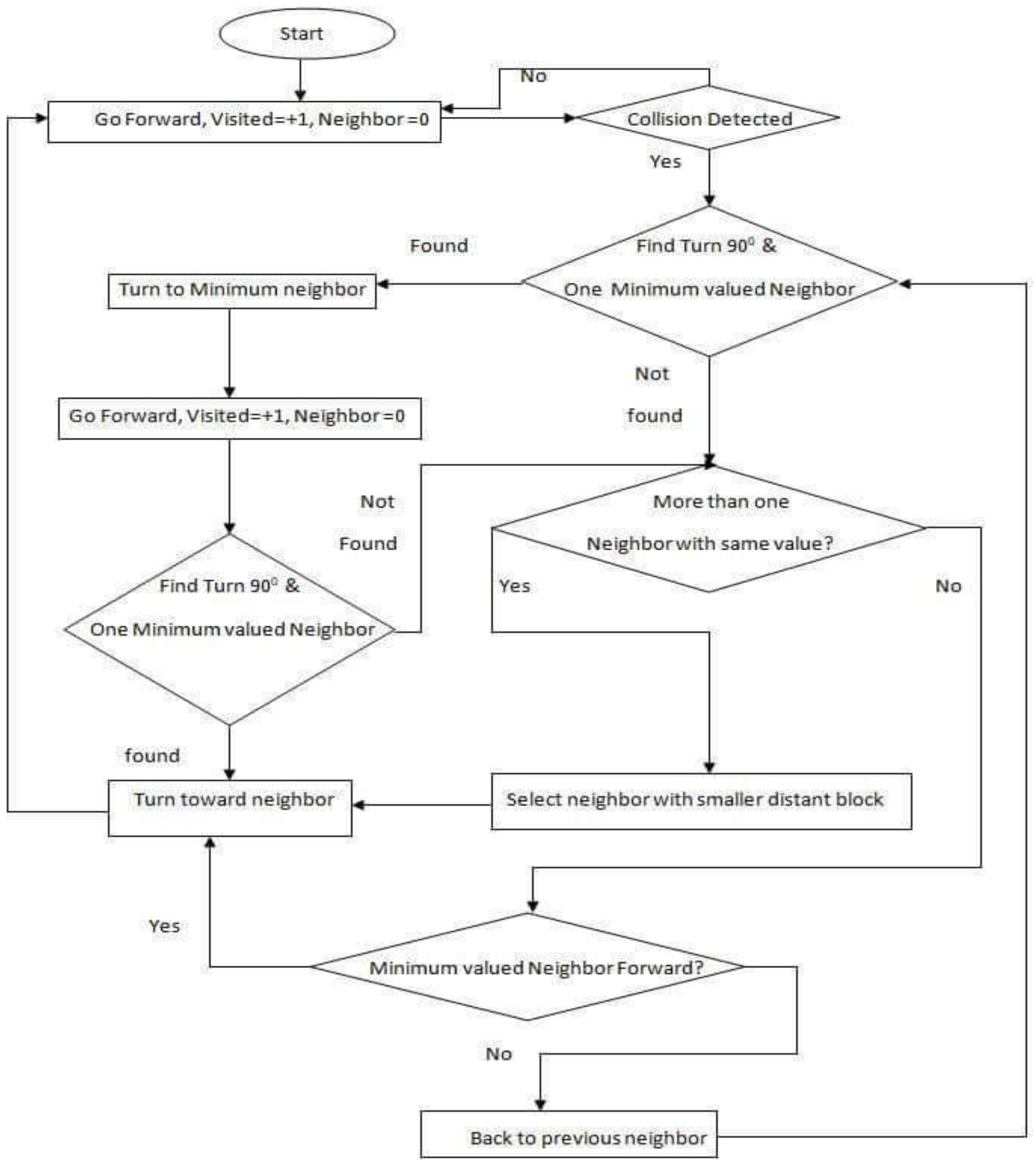

Start
Go Forward, Visited=+1, Neighbor=0
No
Collision Detected
Yes
Find Turn 90⁰ &
One Minimum valued Neighbor
Found
Turn to Minimum neighbor
Not found
Go Forward, Visited=+1, Neighbor=0
Not Found
More than one
Neighbor with same value?
Yes
No
Find Turn 90⁰ &
One Minimum valued Neighbor
found
Turn toward neighbor
Select neighbor with smaller distant block
Yes
Minimum valued Neighbor Forward?
No
Back to previous neighbor

Figure 4.4 : flowchart of proposed algorithm

The steps of the algorithm is given below:

step 1: start

step 2: Robot will go forward toward its facing, each node’s neighbor will become 0.

Step 3: If collision is detected then the robot will search for minimum valued one neighbor at 90° rotation.

Step 4: If one minimum valued neighbor at 90° rotation is found then the robot will turn to the neighbor and go forward one step. Then the robot will search for one minimum valued neighbor at

90° rotation, turn toward that neighbor and start following step 2 again. If one minimum valued neighbor is not found then the robot will follow step 5.

step 5:If more than one same minimum valued neighbor is found then the robot will select neighbor with smaller smaller block. And then the robot will turn to the neighbor and start from step 1 again. If more than one same minimum valued neighbor is not found then the robot will follow step 4.

step 6: If more than one same minimum valued neighbor is not found then the robot will find out minimum valued neighbor forward. If minimum valued neighbor forward is found then the robot will turn to the neighbor and go forward. After that the robot will start following from step 1 again. If minimum valued neighbor forward is not found then the robot will follow step 7.

step 7: If minimum valued neighbor forward is not found then the robot will turn backward and visit the previous node and again start following from step 2.

Table 1 : The working principles of Behavioral approaches with the proposed algorithm.

Function	Use of Function
moving()	It is the behavior for the moving of the robot from one node to another. Robot move forward towards its facing direction(left,right,up,down).
obstacle avoidance()	It is the behavior to avoid obstacles such as walls, objects, etc. It uses sensors to detect obstacles and calculate the neighbor value to avoid the obstacle.
calculate neighbor()	90^0 turn not cleaned neighbor(0 value) found- turn towards the neighbor. 90^0 turn not cleaned neighbor(0 value) not found or same valued neighbor found- calculate smallest descendant and select smallest descendant not found- find minimum forward value minimum forward value not found- turn backward.

Chapter 5

Experimental Results

In this part, the results of the proposed path coverage algorithm is illustrated and compared with spiral and genetic algorithm.

Chapter 5.1 Obtained Result

In our proposed algorithm, assuming that the robot is facing y direction towards down. All the node data initially null.The Robot go forward toward its facing direction and it’s neighbor value 0.

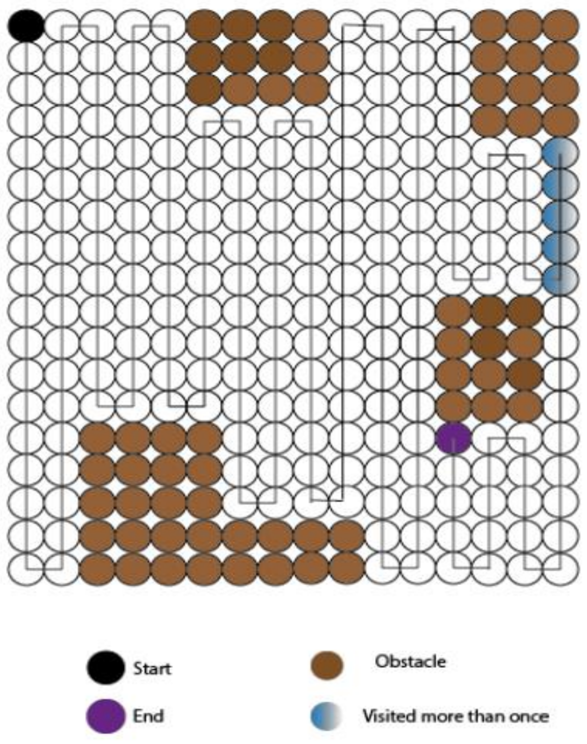


Figure 5.1 : visual representation of proposed algorithm

By following the proposed algorithm, the robot takes 37 turns to complete the cleaning task. All the nodes is considered as 20 cm length individually.

The robot has traveled 45.2m to complete the cleaning task and the number of revisited cell is 5.

5.2 Comparison of algorithm

By following the spiral algorithm, the robot takes 100 turns to complete the cleaning task. The robot has traveled 56.8mto complete the cleaning task And the number of revisited cell is 51. Visual representation of proposed algorithm is given below:

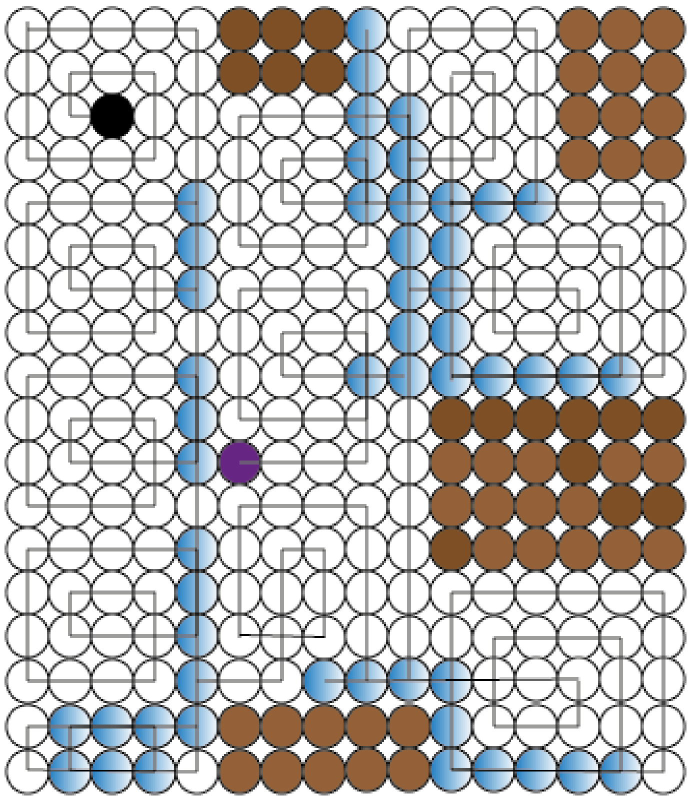


Figure 5.2 : visual representation of spiral algorithm

By following the genetic algorithm, the robot takes 81 turns to complete the cleaning task. And the number of revisited cell is 0. The robot has traveled 48mto complete the cleaning task.Visual representation of genetic algorithm is given below:

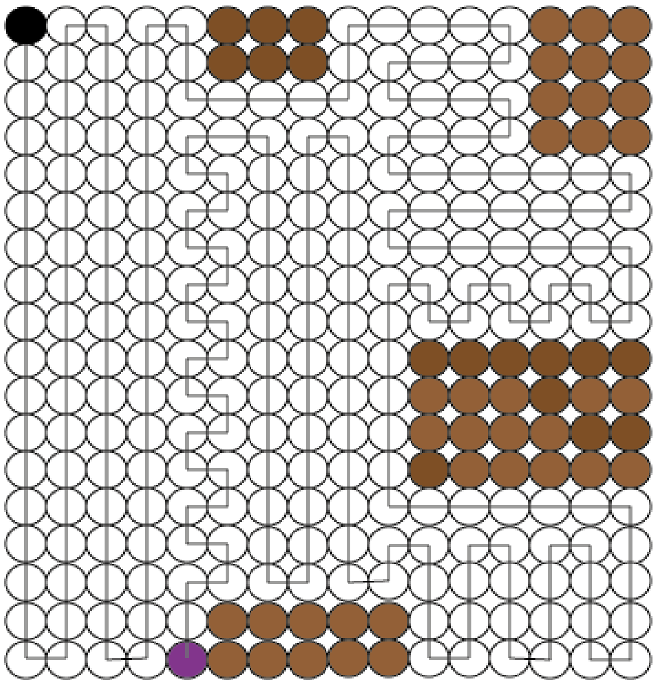


Figure 5.3 : visual representation of genetic algorithm

The Comparison table is given below to show the efficiency of our proposed algorithm.

Table 2 : Comparison between the spiral, the genetic and the proposed algorithm

Algorithm	Turns	Revisited node	Length(meter)
Genetic	81	0	45.2
Spiral	100	51	56.8
Proposed	37	5	48

The graphs are given below to show the efficiency of our proposed algorithm by comparing with other algorithms.

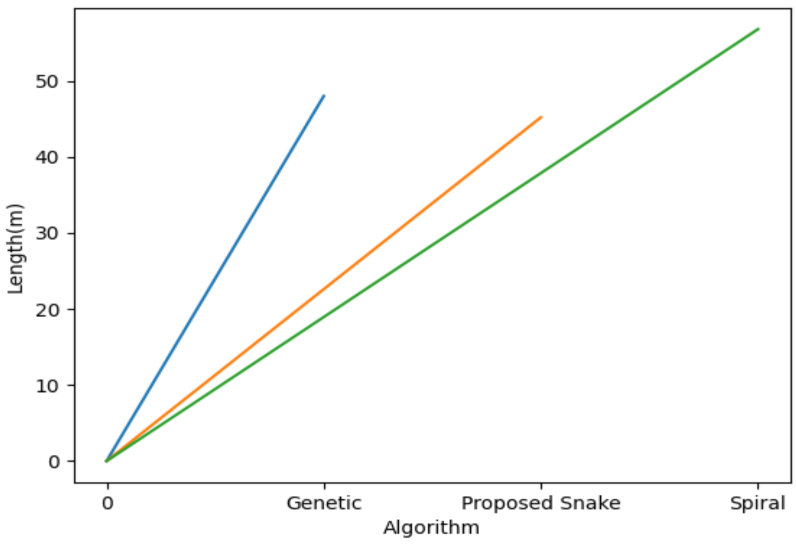


Figure 5.4 : Comparison of algorithm in case of length

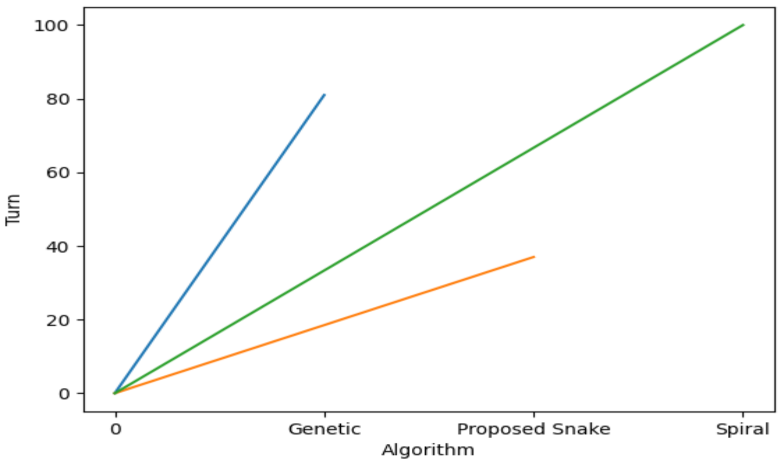


Figure 5.5 : Comparison of algorithm in case of turns

By following genetic algorithm, the number of revisited cell is 0. And the robot takes 81 turns to complete the cleaning task. The robot has traveled 48 mto complete the cleaning task.

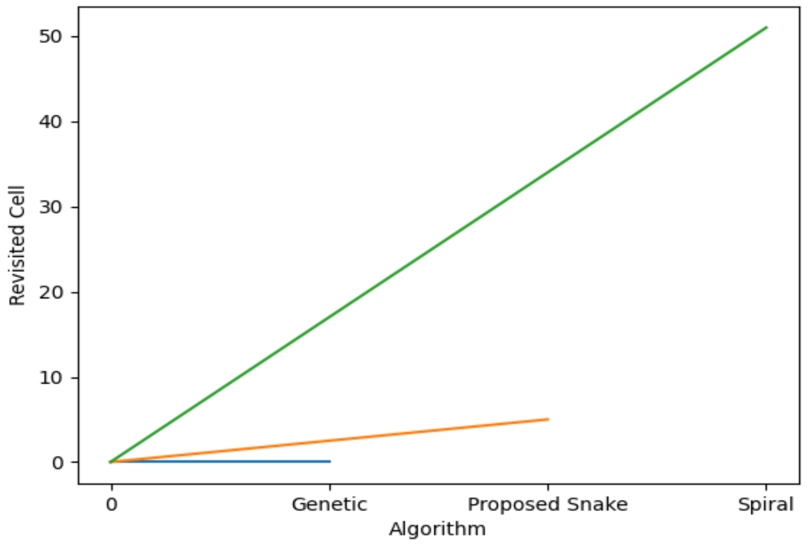


Figure 5.6 : Comparison of algorithm in case of revisited cell

In the case of genetic algorithm , the number of revisited cell is 0 but the robot takes more turns to complete the cleaning task than proposed algorithm following robot. And also the length of the path is higher than proposed algorithm following robot. By following proposed algorithm, the robot cleans5 cells twice but it takes less turns than genetic algorithm.

In the case of spiral algorithm , the number of revisited cell is higher than proposed algorithm. The robot takes more turns to complete the cleaning task than proposed algorithm following robot. And also the length of the path is higher than proposed algorithm following robot.

Chapter 6

Conclusion and Future Work

The aim of this work is to design and develop an autonomous vacuum cleaner robot. It is natural that we need to clean rooms in regular basis and now in this pandemic cleaning and disinfecting rooms has become a necessity. So, we developed a system which is fast and give full coverage while clean and disinfect the floor. An autonomous floor cleaning robot has to follow path coverage algorithm.[28] This paper describes robot building procedure and simulation using simulation tool . Also proposed a coverage path planning algorithm for better performance with lower run time. The less complex algorithm like snake algorithm, spiral algorithm has higher number of revisited cell than our proposed algorithm. And the complex algorithm like genetic algorithm use multiplication which cause more run time and slow the robot velocity. Our algorithm visit cleaned area lesser and have lower run time. The robot has successfully managed its movement and clean the dirt by following proposed algorithm. The performance of the developed robot is demonstrated by comparing the experimental results with lower and higher complex algorithm. And we hope that the results presented here will be useful for robotics community. The autonomous floor cleaner robot can only clean dirt and detect obstacles on flat surface. This research will enhance some future works like clean the stairs. Also it is tough to clean some inconvenience places like, very small corner where robot can not enter. So our future attempt would be to add function to it so that it can clean the stairs and clean small obstacle gap.

Bibliography

[1] A. Konar, I. G. Chakraborty, S. J. Singh, L. C. Jain, and A. K. Nagar, "A deterministic improved q-learning for path planning of a mobile robot," *IEEE Trans. Syst. Man, Cybern. Part ASystems Humans*, vol. 43, no. 5, pp. 1141–1153, 2013, doi: 10.1109/TSMCA.2012.2227719.

[2] M. R. Khan, N. M. L. Huq, M. M. Billah, and S. M. Ahmmad, "Design and development of mopping robot-'HotBot'," *IOP Conf. Ser. Mater. Sci. Eng.*, vol. 53, no. 1, pp. 1–7, 2013, doi: 10.1088/1757-899X/53/1/012008.

[3] A. Amin, A. Madbhavi, S. Somesh, P. Values, and S. Sigma, "A review report on robotic system for survilliance," no. October, 2017.

[4] S. Yatmono, M. Khairudin, H. S. Pramono, and A. Asmara, "Development of Intelligent Floor Cleaning Robot," *J. Phys. Conf. Ser.*, vol. 1413, no. 1, 2019, doi: 10.1088/1742-6596/1413/1/012014.

[5] M. A. Yakoubi and M. T. Laskri, "The path planning of cleaner robot for coverage region using Genetic Algorithms," *J. Innov. Digit. Ecosyst.*, vol. 3, no. 1, pp. 37–43, 2016, doi: 10.1016/j.jides.2016.05.004.

[6] W. Xiaoyu, L. Caihong, S. Li, Z. Ning, and F. U. Hao, "On adaptive monte carlo localization algorithm for the mobile robot based on ROS," *Chinese Control Conf. CCC*, vol. 2018-July, no. Mcl, pp. 5207–5212, 2018, doi: 10.23919/ChiCC.2018.8482698.

[7] W. H. Wang, Y. H. Chien, H. H. Chiang, W. Y. Wang, and C. C. Hsu, "Autonomous Cross-Floor Navigation System for a ROS-Based Modular Service Robot," *Proc. - Int. Conf. Mach. Learn. Cybern.*, vol. 2019-July, pp. 1–6, 2019, doi: 10.1109/ICMLC48188.2019.8949176.

[8] A. Swidan, H. B. Abdelghany, R. Saifan, and Z. Zilic, "Mobility and Direction Aware Ad-hoc on Demand Distance Vector Routing Protocol," *Procedia Comput. Sci.*, vol. 94, no. MobiSPC, pp. 49–56, 2016, doi: 10.1016/j.procs.2016.08.011.

[9] M. Gianni and M. S. Uddin, “Role and task allocation framework for Multirobot collaboration with latent knowledge estimation,” *Eng. Reports*, no. June, 2020, doi: 10.1002/eng2.12225.

[10] P. Vyavahare, S. Jayaprakash, and K. Bharatia, “Construction of URDF model based on open source robot dog using Gazebo and ROS,” *2019 Adv. Sci. Eng. Technol. Int. Conf. ASET 2019*, pp. 1–5, 2019, doi: 10.1109/ICASET.2019.8714265.

[11] K. M. Hasan, Abdullah-Al-Nahid, and K. J. Reza, “Path planning algorithm development for autonomous vacuum cleaner robots,” *2014 Int. Conf. Informatics, Electron. Vision, ICIEV 2014*, no. May, 2014, doi: 10.1109/ICIEV.2014.6850799.

[12] C. Gurel, R. Sathyam, and A. Guha, “ROS-based Path Planning for Turtlebot Robot using Rapidly Exploring Random Trees (RRT*),” *Researchgate.Net*, no. May, 2018, [Online]. Available:https://www.researchgate.net/profile/Canberk_Suat_Gurel/publication 325473185_ROS-based_Path_Planning_for_Turtlebot_Robot_using_Rapidly_Exploring_Random_Trees_RRT/links/5b100c850f7e9b4981ff0c41/ROS-based-Path-Planning-for-Turtlebot-Robot-using-Rapidly-Exp.

[13] O. Tekdas, Wei Yang, and V. Isler, “Robotic routers: Algorithms and implementation,” *Int. J. Rob. Res.*, vol. 29, no. 1, pp. 110–126, 2010, doi: 10.1177/0278364909105053.

[14] E. Gambao and M. Hernando, “Control system for a semi-automatic façade cleaning robot,” *2006 Proc. 23rd Int. Symp. Robot. Autom. Constr. ISARC 2006*, no. November 2019, pp. 406–411, 2006, doi: 10.22260/isarc2006/0078.

[15] A. Gylling, “Improving robotic vacuum cleaners,” *Examensarbete Inom Technol. Grundnivå*, 2018.

[16] N. L. Kushal, H. Chaudhuri, and H. R. Nikithesh, “Autonomous Floor Cleaning Bot,” pp. 1667–1671, 2018.

[17] T. H. Kong, W. S. Kim, J. S. O, M. H. Choi, H. J. Im, and Y. G. Jung, “로봇청소기Robotic vacuum cleaner,” vol. 7, no. 5, pp. 180–182, 2019.

[18] J. Lee, A. S. Ab Ghafar, N. Mohd Nordin, F. A. Saparudin, and N. Katiran, “Autonomous multi-function floor cleaning robot with zig zag algorithm,” *Indones. J. Electr. Eng. Comput. Sci.*, vol. 15, no. 3, pp. 1653–1663, 2019, doi: 10.11591/ijeecs.v15.i3.pp1653-1661.

[19] A. Singh, Anshul, C. Gong, and H. Choset, “Modelling and Path Planning of Snake Robot in cluttered environment,” *2018 Int. Conf. Reconfigurable Mech. Robot. ReMAR 2018 - Proc.*, vol. 2018-Janua, 2018, doi: 10.1109/REMAR.2018.8449833.

[20] H. A. Shakhawat Hossen Prayash, M. Ragib Shaharear, M. F. Islam, S. Islam, N. Hossain, and S. Datta, “Designing and Optimization of An Autonomous Vacuum Floor Cleaning Robot,” *2019 IEEE Int. Conf. Robot. Autom. Artif. Internet-of-Things, RAAICON 2019*, no. June 2020, pp. 25–30, 2019, doi: 10.1109/RAAICON48939.2019.11.

[21] A. Selamat, M. Zolfpour-Arokhlo, S. Z. Hashim, and M. H. Selamat, “A fast path planning algorithm for route guidance system,” *Conf. Proc. - IEEE Int. Conf. Syst. Man Cybern.*, no. October, pp. 2773–2778, 2011, doi: 10.1109/ICSMC.2011.6084092.

[22] S. Savage, “Lecture 8 : Routing I Distance-vector Algorithms.”

[23] W. A. S. Norzam, H. F. Hawari, and K. Kamarudin, “Analysis of Mobile Robot Indoor Mapping using GMapping Based SLAM with Different Parameter,” *IOP Conf. Ser. Mater. Sci. Eng.*, vol. 705, no. 1, 2019, doi: 10.1088/1757-899X/705/1/012037.

[24] Z. B. Rivera, M. C. De Simone, and D. Guida, “Unmanned ground vehicle modelling in Gazebo/ROS-based environments,” *Machines*, vol. 7, no. 2, pp. 1–21, 2019, doi: 10.3390/machines7020042.

[25] W. Qian *et al.*, “Manipulation task simulation using ROS and Gazebo,” *2014 IEEE Int. Conf. Robot. Biomimetics, IEEE ROBIO 2014*, no. December, pp. 2594–2598, 2014, doi: 10.1109/ROBIO.2014.7090732.

[26] V. Prabakaran, M. R. Elara, T. Pathmakumar, and S. Nansai, “Floor cleaning robot with reconfigurable mechanism,” *Autom. Constr.*, vol. 91, no. July 2017, pp. 155–165, 2018, doi: 10.1016/j.autcon.2018.03.015.

[27] B. Li, H. Shen, and D. Tse, “An adaptive successive cancellation list decoder for polar codes with cyclic redundancy check,” *IEEE Commun. Lett.*, vol. 16, no. 12, pp. 2044–2047, 2012, doi: 10.1109/LCOMM.2012.111612.121898.

[28] J. Gamarra, D. Molina, J. Palushi, R. Perez, J. Seborowski, and J. Chung, “Robovac Autonomous Robotic Vacuum Cleaner,” pp. 1–52, 2006, [Online]. Available: https://web.stevens.edu/ses/me/fileadmin/me/senior_design/2007/group01/DesignFinal.pdf.

Appendix A

ROS: To write robot software Robot Operating System (ROS) is a malleable framework. It has a large verity of tools, libraries, and conventions that simplify the task of creating complex and robust robot.

RVIZ: RVIZ is a powerful 3D visualization tool for ROS that allows to visualize a lot of information like simulated robot model, log sensor information from the robot's sensors, information published by camera etc.

Gazebo:Gazebo is a real world physics simulator with the ability to accurately and efficiently simulate robots in complex indoor and outdoor environments. Gazebo offers physics simulation at a much higher degree of constancy, a suite of sensors, and interface between user and program.

URDF is used and systematized in ROS for description of all elements (sensors, joints, links etc.) of a robot model.URDF can define the kinematic and dynamic properties of a single robot in isolation. The addition of these properties makes the URDF file adaptable with the native SDF (Simulation Description Format) Gazebo's model description.The gazebo plugins generate a complete interface (Topic) between ROS and Gazebo and allows to see robot simulation.There are several plugins available in gazebo plugins[4], camera (ROS interface for simulating cameras), Multicamera , GPU Laser,Inertial Measurement Unit (IMU), Planar Move Plugin and many more. And by connecting the robot description to Rviz [Fig-4] it allows to see the gazebo data of the real Robot.ROS has a meta package called robot_model that contains packages. All the packages are inside the meta-package.

urdf: One of the packages inside the robot_model meta package is urdf. This package contains a C++ parser for the Unified Robot Description Format, which is an XML file to represent a robot model. In urdf robot description model,the elements between tag <link> and tag</link> are the description of one specific link.

urdf XML tags: In this section,we created a file and write the relationship between each link and joint in the robot and then save this file with the .urdf extension.

link: The link tag shows a single link of a robot. Using this tag, we could model a robot link and its properties. The modeling includes shape, size,color and import a 3D mesh to represent the robot link.

creating Ros package for robot: Before creating urdf file for the robot,a ROS package is created in the catkin workspace.The package depends on the urdf and xacro packages.

urdf file: In the code, we added a <robot> tag at the top of the definition. Here,we named the robot.
visualizing the robot 3D model in Rviz: After designing urdf, it viewed on RViz.
by Adding physical and collision properties to a urdf model: Before simulating a robot in Gazebo simulator we need to define the robot link's physical properties sand the collision properties of the link.
xacro:This package is used because of large XML documents such as robot descriptions. It is used in urdf packages.We used a command in the ROS launch file for converting xacro to UDRF and used it as a robot_description parameter. Inside this xacro we use constants to make robot descriptions shorter and we defined the length, height, and width of each of the links here.
The wheels control the speed of the robot by adjusting respective velocity. There are also two supporting wheels called caster wheels that supports the robot and rotate according to the movement of the main wheels.
creating the COLLADA file(.dae) of a robot : In this part, urdf robot models is used with OpenRave. Then convert an urdf in a collada file (.dae) format.This file is used to generate the IKFast source file. To convert urdf model into a collada file,a ROS package is used.It is called collada_urdf.The urdf file of ABB-IRB 6640 model is on abb_irb6600_support/urdf folder named irb6640.urdf. we Copied this file into our working folder and run a command for the conversion.
After designing the model of robot, the next stage is its simulation. We used the Gazebo simulator to simulate the robot. Gazebo has a interface in ROS, which exposes the whole control of gazebo in ROS.Before working with Gazebo and ROS, we installed the following packages to work with Gazebo and ROS.
gazebo-ros-pkgs: This consists tools and wrappers for interfacing ROS with Gazebo.
gazebo-msgs: This contains service data structures messages and for interfacing with Gazebo from ROS.
gazebo-plugins: This have Gazebo plugins for actuators, sensors and so on.
gazebo-ros-control: This consists standard controllers to communicate between ROS and Gazebo.
ROS MoveIt:Using the ROS MoveIt and Navigation we interfaced out of the box functionalities like robot manipulation and navigation.
Installing MoveIt

RViz using MoveIt configuration package : MoveIt presents a plugin for RViz which allows it to make new scenes where robot works, visualize the planning output and connect with the visualized

robot,add new objects,create motion plans.

RViz using MoveIt configuration package:MoveIt presents a plugin for RViz which allows it to make new scenes where robot works, visualize the planning output and connect with the visualized robot,add new objects,create motion plans.

Pluginlib**:**We need plugins for our robotics application.We have used Gazebo plugins to get the sensor and robot behavior inside the Gazebo simulator.Gazebo plugins help us to control the robot models and the way Gazebo runs. Gazebo plugins give urdf models greater functionality. It can tie in ROS messages and service calls for sensor output and motor input.

a.Sensor plugin:The sensor plugins are for modeling sensors such as camera.

b.Model plugin:The model plugin is added to a exact model in Gazebo and controls its properties. The parameters like control of the joints,joint state of the model , and on are controlled using this plugin.

c.World plugin:By Using the world plugin,the properties of a specific world in Gazebo can be controlled. ROS provides this plugin framework to load or unload plugins. Pluginlib is a set of a C++ library that helps to write plugins and load or unload.

Camera:It provides ROS interface for simulating cameras.Spcbot: In this part, a RGB camera added to base. Inside spcbot. xacro the expla-nation is showed. After saving both spcbot.xacro and spcbot.gazebo,both Rviz and Gazebo launched in separate terminals.In Rviz, add a Camera display and under Image Topic set it to /spcbot/camera1/image_raw. A camera view is seen of our Gazebo environment.

spcbot_description: This package contains robot description files for spcbot.

urdf/ contains urdf descriptions of various parts of the spcbot

meshes/contains mesh files(.stl,.dae)for visualization and collision properties.

ROS Controllers and Visualization Plugins: The spcbot-mechanism stack consist packages for writing ROS real-time controllers. After creating the controller,use the controller in spcbot simulation.

All kinds of data from sensors viewed through this tool. RViz is installed along with the ROS desktop full installation. RViz has a Graphical User Interface to allow users to modify and configure robots.

Appendix B

Code for Robot building:

```
#include<bits/stdc++.h>
#include<iostream>
#include<stack>
using namespace std;
#define DEBUG 1
#define check(a) DEBUG==1?(cout << a << endl):null;
#define cc(a) DEBUG==1?(cout << a << endl):cout<<"";
#define msg(a,b) DEBUG==1?(cout << a <<" : "<< b << endl):cout<<"";
#define msg2(a,b,c) DEBUG==1?(cout << a <<" : "<< b <<" : "<< c << endl):cout<<"";
#define msg3(a,b,c,d) DEBUG==1?(cout << a <<" : "<< b <<" : "<< c <<" : "<< d <<
endl):cout<<"";
#define msg4(a,b,c,d,e) DEBUG==1?(cout << a <<" : "<< b <<" : "<< c <<" : "<< d <<" : "<< e <<
endl):cout<<"";
#define _INIT ios_base::sync_with_stdio(0); cin.tie(0); cout.tie(0);


#define MX 1000

int block = 999999;
int inf = -111111;
int n;
int mat[MX][MX];
stack<int> prevX;
stack<int> prevY;
int currX , currY;
int blockA, blockB; // These are used to calculate horizontal or vertical split, A will have top or left,
B will have value for bottom or right
char facing; /// it can have 'r','l', 'u','d'
```

```
map<char, int> directionWiseMoveX, directionWiseMoveY, directionWiseMoveopX,
directionWiseMoveopY;


/// This will split the area corresponding to direction, either 'v' for vertical, or 'h' for horizontal. and
put the value in
void countNumberOfZeros(char direction){
blockA = 0;
blockB = 0;
if(direction == 'h'){
for(int i=0;i<currY;i++){
for(int j=0;j<n;j++){
if(mat[i][j] == 0){
blockA++;
}
}
}
for(int i=currY+1;i<n;i++){
for(int j=0;j<n;j++){
if(mat[i][j] == 0){
blockB++;
}
}
}
}else if(direction == 'v'){
for(int i=0;i<n;i++){
for(int j=0;j<currX;j++){
if(mat[i][j] == 0 || mat[i][j] == inf){
blockA++;
}
}
}
```

```
for(int i=0;i<n;i++){
for(int j=currX+1;j<n;j++){
if(mat[i][j] == 0 || mat[i][j] == inf){
blockB++;
}
}
}
}
}


/// returns -1 to end the algo
/// returns 1 for up, 2 for down, 3 for left, 4 for right
int bothNeighbourHasSameValueWhereShouldIGo(){
if(facing == 'l' || facing == 'r'){
countNumberOfZeros('h');
if(blockA==0 && blockB == 0){ // we don't have any zeros left, so our algo ends
return -1;
}
else if(blockA == 0)
{
return 2;
}
else if(blockB == 0)
{
return 1;
}
else if(blockA == blockB){ // both has same value, we can go either direction
// we go to the direction which is closer to the boundary.
}
if(currY < n-currY){ // top side is closer to the boundary
return 1;
}else
```

```
{

return 2;
}
}else if(blockA < blockB){
return 1;
}else{
return 2;
}
}else{
countNumberOfZeros('v');
if(blockA==0 && blockB == 0){ // we don't have any zeros left, so our algo ends
return -1;
}else if(blockA == blockB){ // both has same value, we can go either direction
// we go to the direction which is closer to the boundary.
if(currX < n-currX){ // left side is closer to the boundary
return 3;
}else{
return 4;
}
}else if(blockA < blockB){
return 3;
}else{
return 4;
}
}
}
```

```
void makeAllInfNeighboursZero(){
if(currX + 1 < n){
mat[currY][currX+1] = mat[currY][currX+1] == inf ? 0 : mat[currY][currX+1];
```

```
}
if(currX - 1 >= 0){
mat[currY][currX-1] = mat[currY][currX-1] == inf ? 0 : mat[currY][currX-1];
}
if(currY + 1 < n){
mat[currY+1][currX] = mat[currY+1][currX] == inf ? 0 : mat[currY+1][currX];
}
if(currY - 1 >=0){
mat[currY-1][currX] = mat[currY-1][currX] == inf ? 0 : mat[currY-1][currX];
}
}

/// returns true if next move in the facing direction is collision to block or to boundary
bool collisionAhead(){
// checking the boundary
if( currX + directionWiseMoveX[ facing ] < 0||currX + directionWiseMoveX[ facing ] >=n || currY
+ directionWiseMoveY[ facing ] < 0||currY + directionWiseMoveY[ facing ] >=n  ){
return true;
}else{
if( mat[currY + directionWiseMoveY[facing]][currX + directionWiseMoveX[facing]] == block){
return true;
}else{
return false;
}
}
}
bool cleanedAreaAhead(){
if( mat[currY + directionWiseMoveY[facing]][currX + directionWiseMoveX[facing]] > 0){
return true;}
```

```
}
/// it returns 0 if we can't find minimum neighbour,
/// it returns 1 for up, 2 for down, 3 for left, 4 for right, 5 for go back, 6 for both direction has same
```

```
value
int findMinimumNeighbour(){
if(facing == 'u' || facing == 'd'){
if( currX + 1 < n  && currX-1 >=0 ){ // we can go both left and right
if(mat[currY][currX+1] == block && mat[currY][currX-1] == block){ /// both left and right has
block, so we need to go back.
return 5;
}else if( mat[currY][currX+1] == block ){ // we know both doesn't have blcok, so if right has block
we go left, or vice versa
return 3; // right has block so we go left
}else if(mat[currY][currX-1] == block){
return 4; // left has block so we go right
}else if(mat[currY][currX+1]==mat[currY][currX-1]){ // both is same
return 6;
}else if(mat[currY][currX+1] < mat[currY][currX-1]){ // right is less
return 4;
}else{
return 3;
}
}else if( currX + 1 < n ){ // we can go only one direction, either left, or right
if(mat[currY][currX+1] == block){
// our right has block, left has boundary, so we need to go back
return 5;
}else{
return 4;
}
}else if( currX - 1 >= 0){
if(mat[currY][currX-1] == block){
// our left has block and right has boundary
```

```
return 5;
}else{
return 3;
```

```
}
}
}else{
if( currY + 1 < n  && currY-1 >=0 ){ // we can go up and down
if(mat[currY+1][currX] == block && mat[currY-1][currX] == block){ /// both top and down has
block, so we need to go back.
return 5;
}else if(mat[currY+1][currX] >0 && mat[currY-1][currX] == block){
return 7;
}else if(mat[currY+1][currX] ==block && mat[currY-1][currX] > 0){
return 7;
}
else if( mat[currY+1][currX] == block){
return 1; // down has block so we go up
}else if(mat[currY-1][currX] == block){
return 2; // up has block so we go down
}else if(mat[currY+1][currX] == mat[currY-1][currX]){ // both are same
return 6;
}else if(mat[currY+1][currX] < mat[currY-1][currX]){ // down is less
return 2;
}else{
return 1;
}
}else if( currY + 1 < n ){ // we can go only one direction, either up, or down
if(mat[currY+1][currX] == block){
// our down has block, up has boundary, so we need to go back
return 5;
}else{
return 2;
```

```
}
}else if( currY - 1 >= 0){
if(mat[currY-1][currX] == block){
```

```
// our up has block and down has boundary
return 5;
}else{
return 1;
}
}
}
}
void makeRobotFaceToward(int intDirection){
if(intDirection == 1){
facing ='u';
}else if(intDirection == 2){
facing ='d';
}else if(intDirection == 3){
facing ='l';
}else if(intDirection == 4){
facing ='r';
}
}

bool moreThanOneSameNeighbor()
{
if(facing == 'u' || facing == 'd'){
if(mat[currY+1][currX] == mat[currY-1][currX]){ // both are same
return true;}
else
{return false;}
}
else{
```

```
if(mat[currY+1][currX] == mat[currY-1][currX]){
return true;}
else
```

```
{
return false;
}
}

}

bool minimumValueForward()
{
if(facing == 'r'&& currX+1 <n){
if(currY+1 > currX+1 && currY-1 > currX+1){
return true;
}
else{
return false;
}
}
else if(facing == 'l'&& currX-1 >0){
if(currY+1 > currX-1 && currY-1 > currX-1){
return true;}
else{
return false;}
}
else if(facing == 'u'&& currY-1 >0){
if(currX+1 > currY-1 && currX-1 > currY-1){
return true;}
else
{
return false;
}
}
else if(facing == 'd'&& currY+1 >n){
```

```
if(currX+1 > currY-1 && currX-1 > currY-1){
return true;}
else
{
return false;}
}
}

void goBack(){
currX =  prevX.top();
currY = prevY.top();
prevX.pop();
prevY.pop();
}
void printRoom(){
cout << endl;
for(int i=0;i<n;i++){
for(int j=0;j<n;j++){
if(i == currY && j == currX){
if(facing == 'u'){
printf("%6c", '^');
}else if(facing == 'd'){
printf("%6c", 'v');
}else if(facing == 'l'){
printf("%6c", '<');
}else if(facing == 'r'){
printf("%6c", '>');
}else{
printf("%6c", 'X'); // if at any moment this prints, then something has gone wrong.
}
}else{
printf("%6d", mat[i][j]);
```

```
}
}
cout << endl;
}
cout << endl;
}
void makeMove(){
int turn = 0;
mat[currY][currX]++;
makeAllInfNeighboursZero();
printRoom();
if(collisionAhead() || cleanedAreaAhead()){
// Now we can't go forward, so we have to check our left or right neighbours and make decision.
int nextDirection = findMinimumNeighbour();
if( nextDirection == 7){
if(moreThanOneSameNeighbor() == false){
if(minimumValueForward() == false)
{
// we need to go back
//mat[currY][currX]++;
makeAllInfNeighboursZero();
prevX.push(currX+ directionWiseMoveopX[facing]);
prevY.push(currY + directionWiseMoveopY[facing]);
goBack();
printRoom();
makeMove();}
else
{
//going forward
}
mat[currY][currX]++;
prevX.push(currX+ directionWiseMoveopX[facing]);
prevY.push(currY + directionWiseMoveopY[facing]);
```

```
currX = currX + directionWiseMoveX[facing];

currY = currY + directionWiseMoveY[facing];
makeAllInfNeighboursZero();
printRoom();
goto secondturn;
makeMove();
}
}
else{

int myNextMove = bothNeighbourHasSameValueWhereShouldIGo();
cout<<"myNextMove: "<<myNextMove;
if(myNextMove == -1){
cout <<"ALGO COMPLETE!! HURRAY!!"<< endl;
return;
}
else{
makeRobotFaceToward(myNextMove);
prevX.push(currX+ directionWiseMoveX[facing]);
prevY.push(currY + directionWiseMoveY[facing]);
currX = currX + directionWiseMoveX[facing];
currY = currY + directionWiseMoveY[facing];
makeMove();
}
}
}
if( nextDirection == 5){ // we need to go back
mat[currY][currX]++;
```

```
makeAllInfNeighboursZero();
prevX.push(currX+ directionWiseMoveX[facing]);
prevY.push(currY + directionWiseMoveY[facing]);
```

```
goBack();
printRoom();
makeMove();
}else if(nextDirection == 6){ // both has same direction
int myNextMove = bothNeighbourHasSameValueWhereShouldIGo();
if(myNextMove == -1){
cout <<"ALGO COMPLETE!! HURRAY!!"<< endl;
exit(0);
}else{
makeRobotFaceToward(myNextMove);
prevX.push(currX+ directionWiseMoveX[facing]);
prevY.push(currY + directionWiseMoveY[facing]);
currX = currX + directionWiseMoveX[facing];
currY = currY + directionWiseMoveY[facing];
mat[currY][currX]++;
makeAllInfNeighboursZero();
printRoom();
goto secondturn;
makeMove();
}
}else{
// we have found a minimum to rotate to.
makeRobotFaceToward(nextDirection);
// this is our first rotation
int myNextMove = bothNeighbourHasSameValueWhereShouldIGo();
if(myNextMove == -1){
cout <<"ALGO COMPLETE!! HURRAY!!"<< endl;
exit(0);
}
```

```
prevX.push(currX+ directionWiseMoveX[facing]);// we have come to our this position from
previous position
prevY.push(currY + directionWiseMoveY[facing]);
```

```
currX = currX + directionWiseMoveX[facing];
currY = currY + directionWiseMoveY[facing];
mat[currY][currX]++;
makeAllInfNeighboursZero();
printRoom();
goto secondturn;
}
}else{
// as there is no collision or boundary ahead so we move to the direction we are moving.
prevX.push(currX+ directionWiseMoveX[facing]);
prevY.push(currY + directionWiseMoveY[facing]);
currX = currX + directionWiseMoveX[facing];
currY = currY + directionWiseMoveY[facing];
makeMove();
}
secondturn:{
int nextDirection = findMinimumNeighbour();

if( nextDirection == 7){
if(moreThanOneSameNeighbor() == false){
if(minimumValueForward() == false)
{
// we need to go back
mat[currY][currX]++;
makeAllInfNeighboursZero();
prevX.push(currX+ directionWiseMoveX[facing]);
prevY.push(currY + directionWiseMoveY[facing]);
goBack();
printRoom();
```

```
makeMove();}
else
{
```

```
//going forward
mat[currY][currX]++;
prevX.push(currX+ directionWiseMoveX[facing]);
prevY.push(currY + directionWiseMoveY[facing]);
currX = currX + directionWiseMoveX[facing];
currY = currY + directionWiseMoveY[facing];
makeAllInfNeighboursZero();
printRoom();
goto secondturn;
makeMove();
}
}
else{

int myNextMove = bothNeighbourHasSameValueWhereShouldIGo();
cout<<"myNextMove: "<<myNextMove;
if(myNextMove == -1){
cout <<"ALGO COMPLETE!! HURRAY!!"<< endl;
return;
}
else{
cout<<"my next move "<<myNextMove;
makeRobotFaceToward(myNextMove);
prevX.push(currX+ directionWiseMoveX[facing]);// we have come to our this position from
previous position
prevY.push(currY + directionWiseMoveY[facing]);
currX = currX + directionWiseMoveX[facing];
currY = currY + directionWiseMoveY[facing];
makeMove();
```

```
}
}
}
```

```
if(nextDirection == 5){ // we need to go  back
mat[currY][currX]++;
makeAllInfNeighboursZero();
goBack();
printRoom();
prevX.push(currX+ directionWiseMoveX[facing]);
prevY.push(currY + directionWiseMoveY[facing]);
currX = currX + directionWiseMoveX[facing];
currY = currY + directionWiseMoveY[facing];
makeMove();
}else if(nextDirection == 6){ // both has same value
int myNextMove = bothNeighbourHasSameValueWhereShouldIGo();
if(myNextMove == -1){
cout <<"ALGO COMPLETED!!"<< endl;
return;
}else{
makeRobotFaceToward(myNextMove);
prevX.push(currX+ directionWiseMoveX[facing]);// we have come to our this position from
previous position
prevY.push(currY + directionWiseMoveY[facing]);
currX = currX + directionWiseMoveX[facing];
currY = currY + directionWiseMoveY[facing];
makeMove();
}
}else{
makeRobotFaceToward(nextDirection);
// this is our second rotation
prevX.push(currX+ directionWiseMoveX[facing]);// we have come to our this position from
previous position
```

```
prevY.push(currY + directionWiseMoveY[facing]);
currX = currX + directionWiseMoveX[facing];
currY = currY + directionWiseMoveY[facing];
```

```
makeMove();
}

}



}
int main(){

/// setup
directionWiseMoveX['u'] = 0;
        directionWiseMoveX['d'] = 0;
        directionWiseMoveX['l'] = -1;
        directionWiseMoveX['r'] = 1;

        directionWiseMoveY['u'] = -1;
        directionWiseMoveY['d'] = 1;
        directionWiseMoveY['l'] = 0;
        directionWiseMoveY['r'] = 0;

        directionWiseMoveopX['u'] = 0;
        directionWiseMoveopX['d'] = 0;
        directionWiseMoveopX['l'] = 1;
        directionWiseMoveopX['r'] = -1;

        directionWiseMoveopY['u'] = 1;
        directionWiseMoveopY['d'] = -1;
        directionWiseMoveopY['l'] = 0;
        directionWiseMoveopY['r'] = 0;
        for(int i=0;i<MX;i++){
for(int j=0;j<MX;j++){
mat[i][j] = inf;
```

```
}
}
/// For dynamic input taking
//    cin >> n;
//    for(int i=0;i<n;i++){
//        for(int j=0;j<n;j++){
//            cin >> mat[i][j];
//        }
//    }

n = 8;
int preset[n][n] = {
{  inf,   inf, inf, block, block, inf, inf, inf},
{  inf,   inf, inf, block, block, inf, inf, inf},
{  inf,   inf, inf, block, block, inf, inf, inf},
{  inf,   inf, inf,   inf,   inf, inf, inf, inf},
{  inf,   inf, inf,   inf,   inf, inf, inf, inf},
{  inf,   inf, inf,   inf,   inf, inf, inf, inf},
{block, block, inf,   inf,   inf, inf, inf, inf},
{block, block, inf,   inf,   inf, inf, inf, inf}
};
for(int i=0;i<n;i++){
for(int j=0;j<n;j++){
mat[i][j] = preset[i][j];
}
}
/// starting from top left side of the map, facing right
mat[0][0] = 0;
facing = 'r';
```

```
makeMove();
return 0;
}
```