Symswarm: Symmetrical Pattern Formation in a Swarm of Mobile Robots

Symswarm: Symmetrical Pattern Formation in a Swarm of Mobile Robots

Course Name: Control System I Laboratory

Course No: EEE 318

Submitted to

Sadman Sakib Ahbab

Lecturer, Department of EEE

Bangladesh University of Engineering and Technology

Mohammad Muntasir Hassan

Lecturer, Department of EEE

Bangladesh University of Engineering and Technology

Prepared by

Md. Saheed Ullah (1606152)

Zulqarnain Bin Ashraf (1606153)

Md. Abrar Istiak (1606155)

Mortuza Minhaj Chowdhury (1606157)

Sudipta Chandra Sarker (1606162)

Level 3, Term 2

Abstract

The primary aim of this project is to build up an algorithm for symmetrical shape formation of multiple robots using the concept of Swarm Robotics. Also the algorithm is visualized using the robotic environment VREP. The whole work is implemented in environments both with and without obstacles. The main challenge includes choosing suitable sensors to detect other robots, choosing perfect type of bots, avoiding obstacles in the path and successful information transfer among the robots in the environment. The project meets the criteria of swarm robots and works well for different types of conditions. This work can be applied for real life robots in future.

Keywords:

Swarm Robotics, VREP, MATLAB, Lidar, Sonar.

1. Objectives

- To get familiarized with the concept of Swarm Robotics.
- To develop a novel algorithm for symmetric shape formation of multiple robots.
- Implementation of the algorithm through the application of MATLAB.
- Using VREP to visualize the algorithm for different types of robots and robotic accessories.
- Application in real life purposes.

2. Introduction

2.1 Swarm Robotics

Swarm robotics is the study of how to design groups of robots that operate without relying on any external infrastructure or on any form of centralized control. In a robot swarm, the collective behavior of the robots results from local interactions between the robots and between the robots and the environment in which they act. The design of robot swarms is guided by swarm intelligence principles. These principles promote the realization of systems that are fault tolerant, scalable and flexible. Swarm robotics appears to be a promising approach when different activities must be performed concurrently, when high redundancy and the lack of a single point of failure are desired, and when it is technically infeasible to set up the infrastructure required to control the robots in a centralized way. Examples of tasks that could be profitably tackled using swarm robotics are demining, search and rescue, planetary or underwater exploration, and surveillance.

Figure 1: Swarm Robots

A robot swarm is a self-organizing multi-robot system characterized by high redundancy. Robots' sensing and communication capabilities are local and robots do not have access to global information. The collective behavior of the robot swarm emerges from the interactions of each individual robot with its peers and with the environment. Typically, a robot swarm is composed of homogeneous robots, although some examples of heterogeneous robot swarms do exist.

2.2 VREP

V-REP - the Virtual Robot Experimentation Platform - is a 3D robot simulation software, with integrated development environment, that allows you to model, edit, program and simulate any robot or robotic system (e.g. sensors, mechanisms, etc.).

It offers a multitude of functionality that can be easily integrated and combined through an exhaustive API and script functionality. V-REP supports 2 different physics engines (Bullet and

ODE), handling of the inverse/forward kinematics of any type of mechanism, proximity sensor simulation, camera sensor simulation with many built-in image processing filters, path planning, minimum mesh-mesh distance calculation, surface cutting simulation, graphing, etc.

V-REP is used for remote monitoring, hardware control, fast prototyping and verification, fast algorithm development/parameter adjustment, safety double-checking, robotics-related education or factory automation simulations.

2.3 MATLAB

MATLAB is a high-performance language for technical computing. It integrates computation, visualization, and programming in an easy-to-use environment where problems and solutions are expressed in familiar mathematical notation. Typical uses include:

- Math and computation
- Algorithm development
- Modelling, simulation, and prototyping
- Data analysis, exploration, and visualization
- Scientific and engineering graphics
- Application development, including Graphical User Interface building

MATLAB is an interactive system whose basic data element is an array that does not require dimensioning. This allows you to solve many technical computing problems, especially those with matrix and vector formulations, in a fraction of the time it would take to write a program in a scalar non-interactive language such as C or FORTRAN.

MATLAB has evolved over a period of years with input from many users. In university environments, it is the standard instructional tool for introductory and advanced courses in mathematics, engineering, and science. In industry, MATLAB is the tool of choice for high-productivity research, development, and analysis.

MATLAB features a family of application-specific solutions called toolboxes. Very important to most users of MATLAB, toolboxes allow you to learn and apply specialized technology. Toolboxes are comprehensive collections of MATLAB functions (M-files) that extend the MATLAB environment to solve particular classes of problems. Areas in which toolboxes are available include signal processing, control systems, neural networks, fuzzy logic, wavelets, simulation, and many others.

3. Algorithm Explanation

To make a symmetrical pattern the whole task is divided into two phases. It is divided to maintain a harmony between the robots. The two phases can be described as:

Phase I: In the first phase the main task of the robots is to select a leader robot. In this phase the task robots don't follow the principle of swarm robotics. Here all robots work independently. In this phase, all the robots have to enter the desired radius. User has to input a desired centre and a desired radius. Thus robots will form a symmetry around a point within that radius. Taking the input from the user, all the robots start to move towards the desired centre point in the shortest distance. It is to be mentioned that obstacle avoidance is enabled throughout the whole task. If any single robot reaches within the radius then it stops and starts to act as leader robot. Afterward the first phase the leader robot dominates in decision over the swarm system.

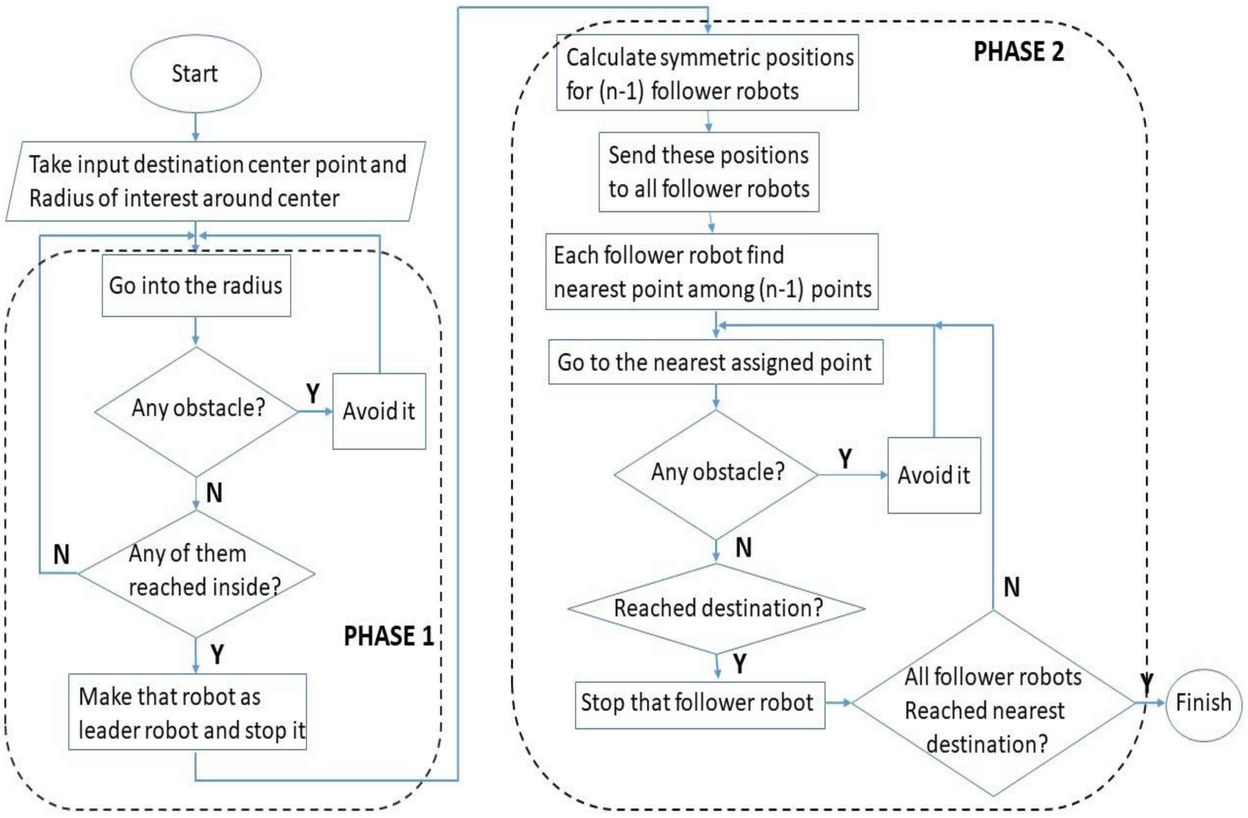

Start
Take input destination center point and Radius of interest around center
Go into the radius
Any obstacle?
Y
Avoid it
N
Any of them reached inside?
N
Y
Make that robot as leader robot and stop it
PHASE 1
PHASE 2
Calculate symmetric positions for (n-1) follower robots
Send these positions to all follower robots
Each follower robot find nearest point among (n-1) points
Go to the nearest assigned point
Any obstacle?
Y
Avoid it
N
Reached destination?
Y
Stop that follower robot
N
All follower robots Reached nearest destination?
Y
Finish

Figure 2: Complete flowchart of the whole task

Phase II: In this second phase the main task of the swarm system is to send all the follower robots to a symmetric position around the desired center point within the radius. Initial stage of this stage is the calculation for the symmetric position for follower robots. As one robot reaches within the radius so it stops and does all leading instruction to the rest of the follower robots. It is

to be mentioned that there are (n-1) number of follower robots where n is the total number of robots. Figure 2 represents the flowchart of the whole algorithm. Phase II starts by finding symmetric position and then sends (n-1) calculated symmetric position to all of the follower robots. Each follower robot has to find the nearest position among them and then start to go to that point. Gradually all of the follower robots reach their booked nearest point and eventually construct a symmetric pattern.
This is the main summarization. Later the sub-blocks of the complete algorithm will be described.

4. Mathematical Explanation of Symmetrical Position Calculation

4.1 Symmetry Figure Angle Specification

A. Geometrical Approach

We have 3 inputs specified here.

I. Number of robots (N)
II. Initial Bot's coordinate (x_r ,y_r)
III. Symmetry center coordinate (x_c ,y_c)

From geometric perspective, 'n' robots will form 'n' sided symmetrical polygon, with equal consecutive angular distance. The calculation could be a little complex with polygon due to complexity in point determination, so we approached with idea of a circle having 'N' points on its circumference equally distant from adjacent points.

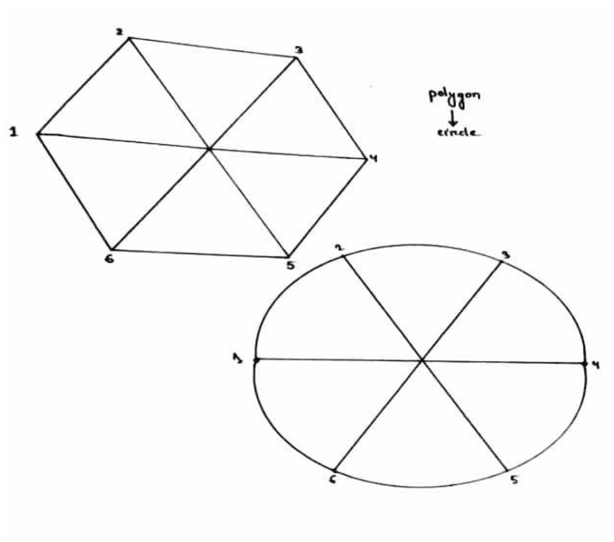


Figure 3: Polygon and circle analogy

B. Distance and Angle Specification between Consecutive Points

From geometrical derivation, it’s evident that angular distance between two consecutive points is (360/N) degree and the linear distance is 2rsin(180/N) whereas r=symmetry radius. In figure 3, r_s is the symmetry radius and the distance between p_n and p_{n-1} is $2r_s$sin(180/N).

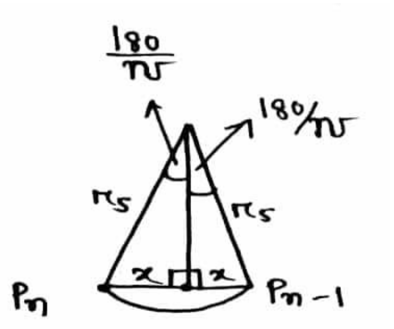


Figure 4: Angle and Distance Calculation

4.2 Determination of Other Symmetrical Points: Slope Order Selection

Now that our distance and angle are known, we are to find geometrical solution of symmetrical points. We calculate our primary slope:

$$m_i=(y_c-y_r)/(x_c-x_r) \text{ -----------(1)}$$

Initial Bot's coordinate= (x_r,y_r)

Symmetry center coordinate= (x_c,y_c)

Now we denote approximate symmetry points on the circle starting from initial bot and join all of them with center by drawing lines. So we have got 'N' number of lines. We have already determined initial bot's slope. We now intend to determine (N-1) slopes for rest (N-1) points. From starting slope we can go either clockwise or anticlockwise.

Let the angle between two lines L_1 and L_2 be 'theta'.

$$\text{'theta'}=\pm\tan(\text{inverse})\{(m_1-m_2)/(1+m_1m_2)\}.\text{-----(2)}$$

$$m_1=L_1\text{'s slope.}$$

$$m_2=L_2\text{'s slope.}$$

Ambiguity arises with (±) sign, whether we will take + or -.Here as we know our primary slope 'mi'. Let 1st bot's line slope=m_1

Angle between them (primary line and 1st Bot line) is (360/N).

$$(360/N)=\pm\tan(\text{inverse})\{(m_i-m_1)/(1+m_1m_i)\}\text{-----(3)}$$

Let's proceed with taking + sign. Converting the eqn(3):

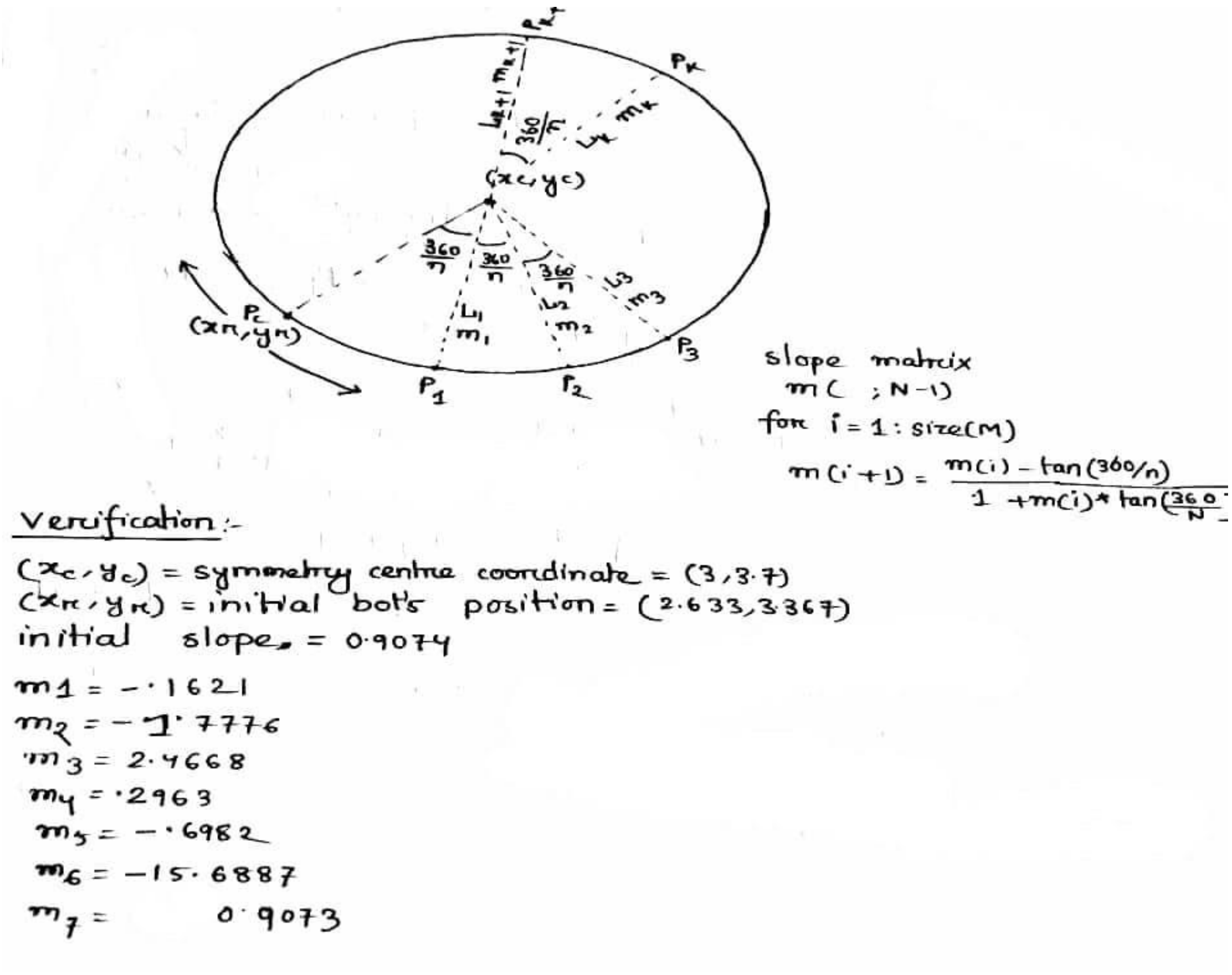
$$m_1=\{m_i-\tan(360/N)\}/\{1+m_i*\tan(360/N)\}.\text{------(4)}$$

$$m_2= \{m_1-\tan(360/N)\}/\{1+m_1*\tan(360/N)\}\text{-------(5)}$$

In vector form if we declare a slope matrix of (N-1) elements, the relation could be like:

$$m_{i+1}=\{m_i-\tan(360/N)\}/\{1+m_i*\tan(360/N)\}\text{-----(6)}$$

The logic behind this is that thus we should get all slope values serially in either clockwise or anticlockwise order. Our claim is verified by manual calculation and MATLAB code output in figure 5 and 6 respectively.



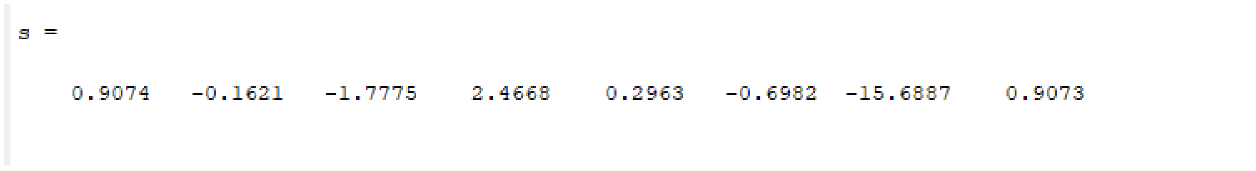
slope matrix
```
m( ;N-1)
for i = 1:size(M)
    $m(i+1) = \frac{m(i) - \tan(360/n)}{1 + m(i) * \tan(\frac{360}{N}}$
```

<u>Verification:-</u>

(x_c, y_c) = symmetry centre coordinate = (3, 3.7)
(x_r, y_r) = initial bot's position = (2.633, 3.367)
initial slope = 0.9074

$m_1 = -.1621$
$m_2 = -1.7776$
$m_3 = 2.4668$
$m_4 = .2963$
$m_5 = -.6982$
$m_6 = -15.6887$
$m_7 = 0.9073$

Figure 5: Manual Calculation for Slopes



```
s =

    0.9074   -0.1621   -1.7775    2.4668    0.2963   -0.6982  -15.6887    0.9073
```

Figure 6: MATLAB Code Output for Slopes

N.B. We calculated for 7 bots with one bot's position pre-defined and our symmetry center was (3, 3.7) and initial bot's coordinate was (2.633, 3.367).

Decision: As we have seen from output slope matrix, the initial value 0.9074 is retained in its last element, it's quite evident that our procedure of finding symmetry slope is perfect.

4.3 Point Determination by Distance Verification

After gaining slope values for all points, corresponding line equation can easily be found:

$$(y-y_c)=m_i*(x-x_c);\ [i=1:N-1]\text{-----}(7)$$

Symmetry radius has to be determined from initial condition.

$$r=\sqrt{}\{(x_r-x_c)+(y_r-y_c)^2\}\text{-----}(8)$$

So, our symmetry circle's equation will be:

$$(x-x_c)^2+(y-y_c)^2=r^2\text{-------}(9)$$

For each line, m_i, solving line equation and circle's equation gives us two different solution pairs.

$$x_i=x_c\pm r/\sqrt{}\{1+m_i^2\}\text{-------}(10)$$

$$y_i=y_c+m_i*\{x_i-x_c\}\text{-------}(11)$$

So we are getting two solutions for each line. We necessarily can't take two values because we are supposed to take one exact point after another. Now we will use our distance case, we know that two consecutive symmetrical points have distance fixed.

$$D_s=2r*\sin(180/N)\text{-------}(12)$$

Now we have checked continuous distance between points. Solving for bot 1,$\{x_1,y_1\}$ has two solutions, we have to check which one is at 2r*sin(180/n) unit distance from initial bot position (x_r,y_r),that's our desired position. In short, our distance condition is:

$$\sqrt{}[\{x_i-x_{i+1}\}^2+\{y_i-y_{i+1}\}^2]\text{------}(13)$$

Our final output for 7 bots and previously declared point pairs are shown in figure 7.

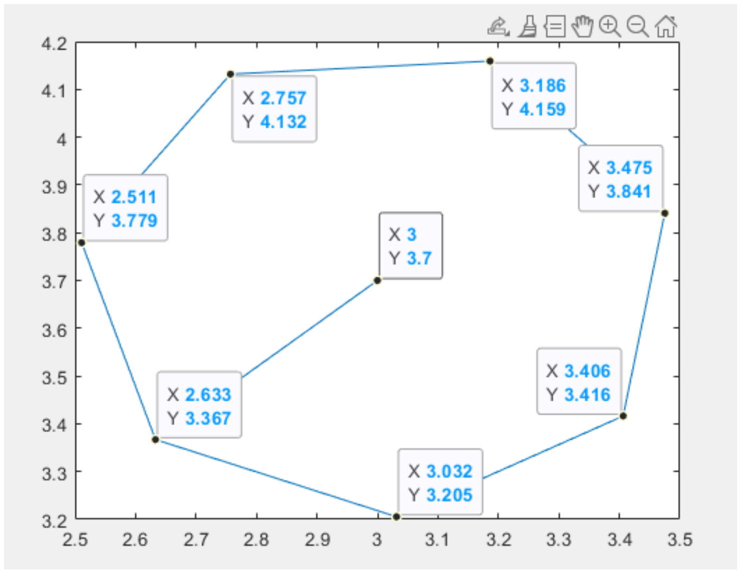


Figure 7: Final Symmetry

5. Implementation

- **VREP Environment:** VREP software is used for simulation and demonstration purposes. It is executable by MATLAB and very easy to customize. Some static objects are added in this VREP environment to make it more realistic. Each object contains a child sheet with necessary code to make it responsive by sonar sensor.
- **Sensor for Path Planning:** When it comes to the sensor selection for obstacle avoidance process, it is pretty hard to decide which sensor should we use and which should not. Lidar is a good option to choose when it comes to the question of accuracy as this sensor provides good accuracy in obstacle detecting. But the price of Lidar is pretty high and multiple sensors are needed for multiple robots. So it would be quite impractical if we used a Lidar in our project in purpose of obstacle avoidance. Thus for obstacle avoidance of individual swarms, sonar sensors are a better fit. It is pretty cheap and much more available than a Lidar. But Sonar sensors are less sensitive towards accuracy. So for maximum optimization, multiple units of Sonar sensors for each individual swarm to improve the accuracy of obstacle avoidance process.
- **Bots:** Pioneer bots are used here as individual swarms in the project. It is a mobile bot found readymade in VREP. It is made of two wheels which can easily be controlled by MATLAB program. It can move 360 degrees as it contains only two wheels. Each pioneer bot contains three sonar sensors for obstacle avoidance purposes. Each pioneer bot contains a child sheet which we have disabled to make them responsive and executable by MATLAB program.

6. Path Planning

6.1 Path Follower Algorithm

At first the robot will see sonar’s reading. If the sonar’s reading is good enough to ensure that there is nothing around the robot in its way, it will calculate the desired angle it needs to achieve to ensure that no obstacle is found on the way to go straight to destination. The robot will rotate according to desired angle & after turning the angle it will go straight for a specific time. But if it finds sonar reading indicating that something is here according to 7 conditions given below it will follow obstacles avoidance algorithm. Else after turning it will go straight towards the destination. Repeatedly the robots check their present position, destination angle, destination point and sonar values. After finding obstacles they will go to obstacle avoidance algorithm & after avoiding it will again check all the values & follow path follower algorithm.

6.2 Obstacle Avoidance

For detecting obstacles & avoiding it, we have done our work with sonar sensors. At first we thought of working the process in Lidar. But it is very expensive. As we are working with multiple robots, not single robots we need to do our work in a cost efficient way. So, we have gone with a sonar sensor. For each Pioneer robot, we have used 3 sonars in 3 directions. As sonar is established in pioneer robot readymade found in **VREP**, we need not to add sonar just to use it. In the forward direction there is a sensor for detecting forward obstacles. There is also a sensor between forward and right direction and another one between forward & left direction for detecting obstacles. We have defined 7 conditions:

Condition 1: If reading of left sided sonar is less than 0.6 meter & others sonar reading is greater than 0.6meter, it means that something is in the left side & nothing is in the right or front side. So, we will turn the robot slightly rightwards to prevent the collision.

Condition 2: If reading of right sided sonar is less than 0.6 meter & others sonar reading is greater than 0.6meter, it means that something is in the right side & nothing is in the left or front side. So, we will turn the robot slightly leftwards to prevent the collision.

Condition 3: If reading of front sided sonar is less than 0.6 meter & others sonar reading is greater than 0.6meter, it means that something is in the front side & nothing is in the left or right side. It is a dangerous case, the robot is on the way to collision. So, we will turn the robot backwards for some moments to prevent the collision. Then turn the robot in the leftwards or rightwards which is up-to us. And here we need to rotate a big angle to change the robot's direction.

Condition 4: : If reading of right & left sided sonar is less than 0.6 meter & front sonar reading is greater than 0.6meter, it means that something is in the right & left side but nothing is in the front side. Therefore, the robot can go in the front direction but as in the right & left there are obstacles the robot needs to go safely at lower speed. We decrease the robot velocity here.

Condition 5: If reading of left & front sided sonar is less than 0.6 meter & right sonar reading is greater than 0.6meter, it means that something bigger is on the left side also front side and hopefully nothing is in the right side. So, we will turn the robot rightwards to prevent the collision.

Condition 6: If reading of right & front sided sonar is less than 0.6 meter & left sonar reading is greater than 0.6meter, it means that something bigger is in the front side also right side and hopefully nothing is in the left side. So, we will turn the robot leftwards to prevent the collision.

Condition 7: If reading of all sonars are less than 0.6 meter it means that obstacles are surrounding the robot .It is a dangerous case, the robot is on the way to collision. So, we will turn the robot backwards for some moments to prevent it from the trap. Then turn the robot in the leftwards or rightwards which is up-to us. And here we need to rotate a big angle to change the robot's direction.

7. Results

VREP simulation verifies our proposed algorithm. We have shown our experimentation for 3, 4, 5 robots. With and without obstacles we also see our algorithm’s success. The algorithm is supposed to work with an even higher number of robots. After this success we can prove the robustness of our algorithm. Now we see the simulation result of our algorithm.

For 3 Robots

For all of the simulation we set the desired centre is [3, 3.7] and desired radius is 2m. From figure 8 we see the initial position of 3 robots. It can be random inside the environment.

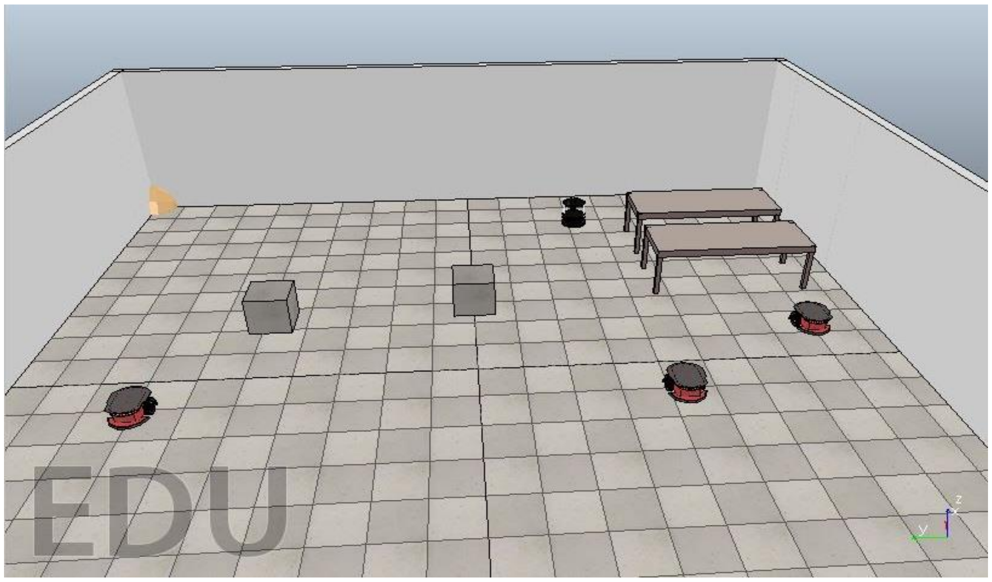
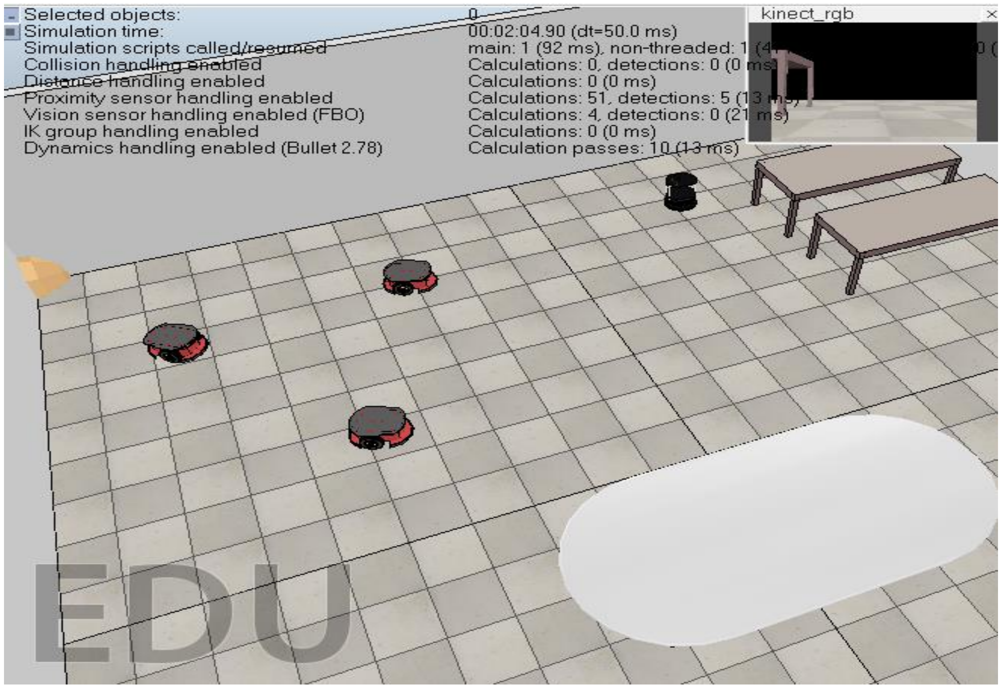


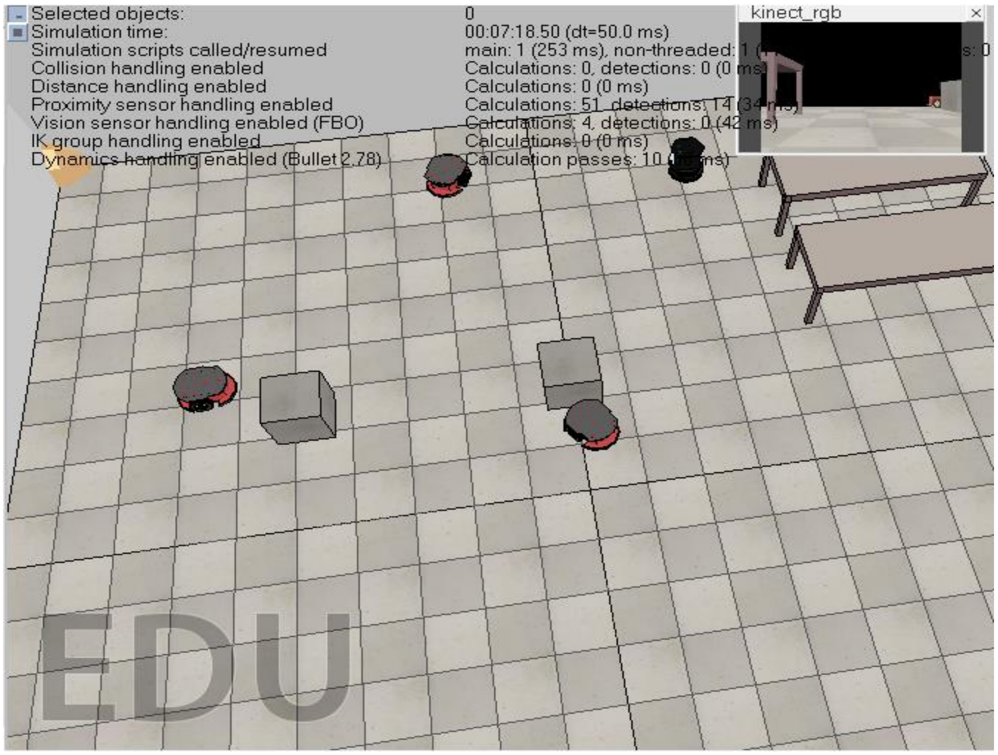
Figure 8: Initial position of 3 robots



```
Selected objects:                          0
Simulation time:                           00:02:04.90 (dt=50.0 ms)
Simulation scripts called/resumed          main: 1 (92 ms), non-threaded: 1 (4
Collision handling enabled                 Calculations: 0, detections: 0 (0 ms)
Distance handling enabled                  Calculations: 0 (0 ms)
Proximity sensor handling enabled          Calculations: 51, detections: 5 (13 ms)
Vision sensor handling enabled (FBO)       Calculations: 4, detections: 0 (21 ms)
IK group handling enabled                  Calculations: 0 (0 ms)
Dynamics handling enabled (Bullet 2.78)    Calculation passes: 10 (13 ms)
```

Figure 9: Symmetrical pattern for 3 robots without obstacle

In figure 9, we see the final result after phase 2. Robots were able to place in a symmetrical place around the desired centre within the radius.



```
Selected objects:                        0
Simulation time:                         00:07:18.50 (dt=50.0 ms)
Simulation scripts called/resumed        main: 1 (253 ms), non-threaded: 1 (
Collision handling enabled               Calculations: 0, detections: 0 (0 ms)
Distance handling enabled                Calculations: 0 (0 ms)
Proximity sensor handling enabled        Calculations: 51, detections: 14 (34 ms)
Vision sensor handling enabled (FBO)     Calculations: 4, detections: 0 (42 ms)
IK group handling enabled                Calculations: 0 (0 ms)
Dynamics handling enabled (Bullet 2.78)  Calculation passes: 10 (
```

Figure 10: Symmetrical pattern for 3 robots with Obstacle

From the figure 10, we see the obstacle avoidance capability of our algorithm. Robots were able to be symmetric around the desired center even after obstacle.

For 4 Robots

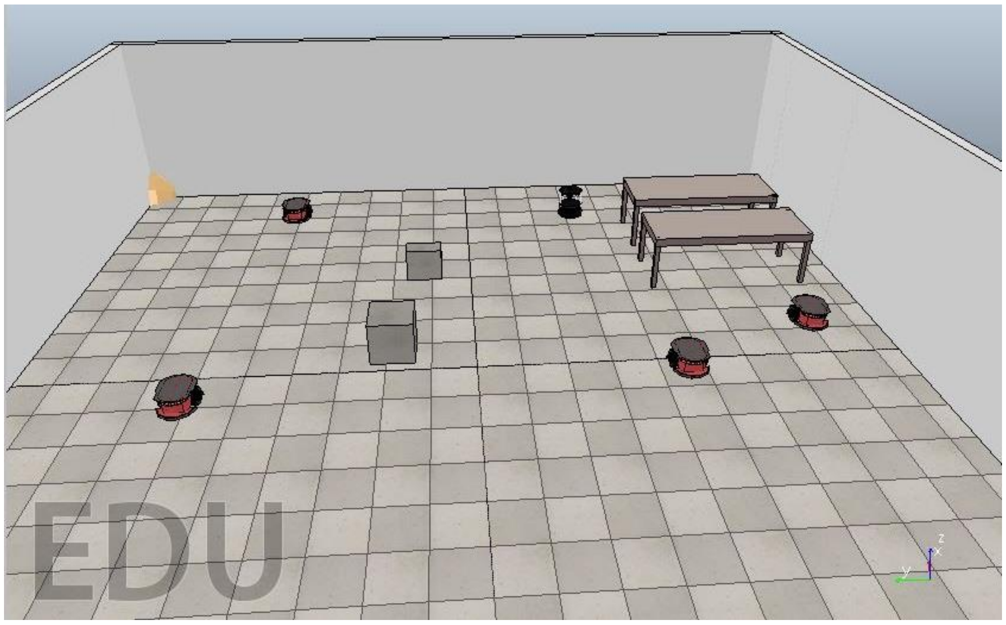
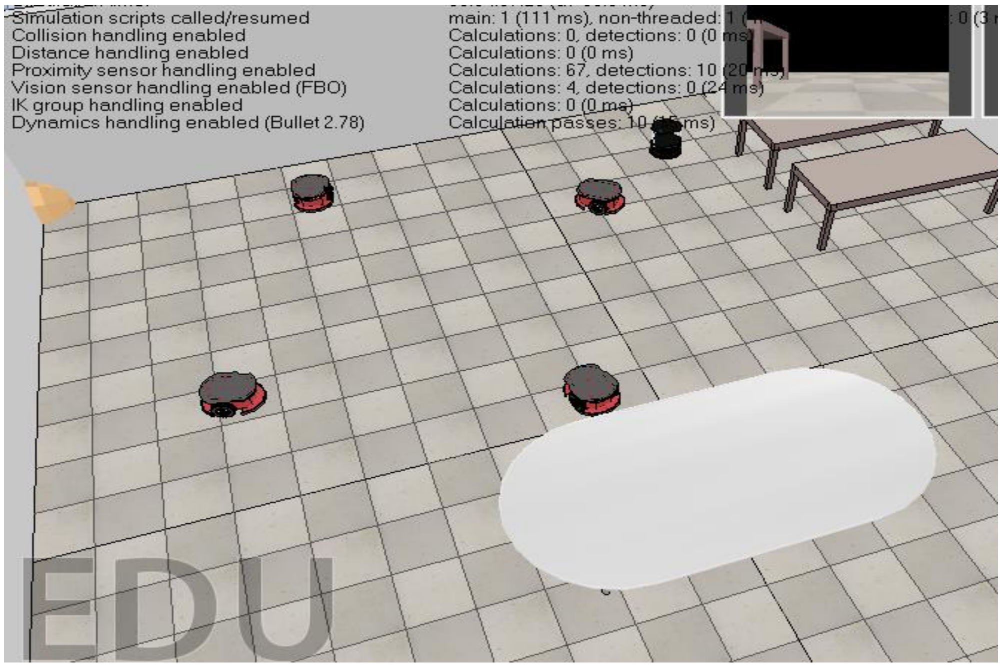
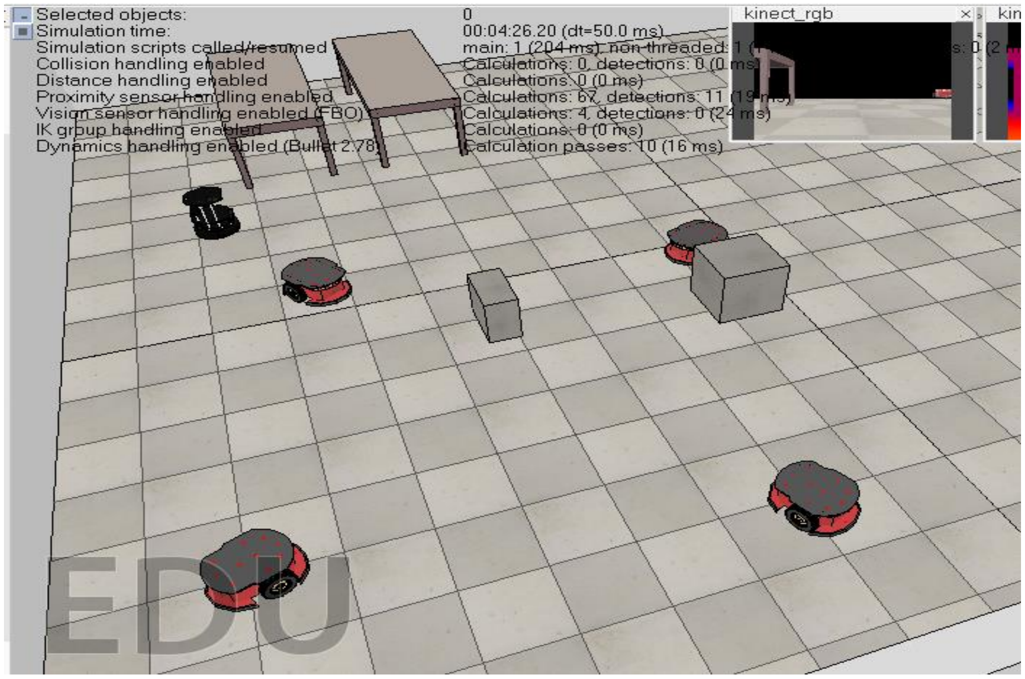


Figure 11: Initial position of 4 robots



```
Simulation scripts called/resumed          main: 1 (111 ms), non-threaded: 1 (
Collision handling enabled                 Calculations: 0, detections: 0 (0 ms)
Distance handling enabled                  Calculations: 0 (0 ms)
Proximity sensor handling enabled          Calculations: 67, detections: 10 (20 ms)
Vision sensor handling enabled (FBO)       Calculations: 4, detections: 0 (24 ms)
IK group handling enabled                  Calculations: 0 (0 ms)
Dynamics handling enabled (Bullet 2.78)    Calculation passes: 10 (15 ms)
```

Figure 12: Symmetrical Pattern for 4 robots without Obstacle



```
Selected objects:                          0
Simulation time:                           00:04:26.20 (dt=50.0 ms)
Simulation scripts called/resumed          main: 1 (204 ms), non-threaded: 1 (
Collision handling enabled                 Calculations: 0, detections: 0 (0 ms)
Distance handling enabled                  Calculations: 0 (0 ms)
Proximity sensor handling enabled          Calculations: 67, detections: 11 (19 ms)
Vision sensor handling enabled (FBO)       Calculations: 4, detections: 0 (24 ms)
IK group handling enabled                  Calculations: 0 (0 ms)
Dynamics handling enabled (Bullet 2.78)    Calculation passes: 10 (16 ms)
```

Figure 13: Symmetrical pattern for 4 robots with Obstacle

From figure 11 we see the initial position of 4 robots, figure 12, 13 respectively shows the simulation result of our algorithm. It proves the robustness of our algorithm that our algorithm works for multiple numbers of robots.

For 5 Robots

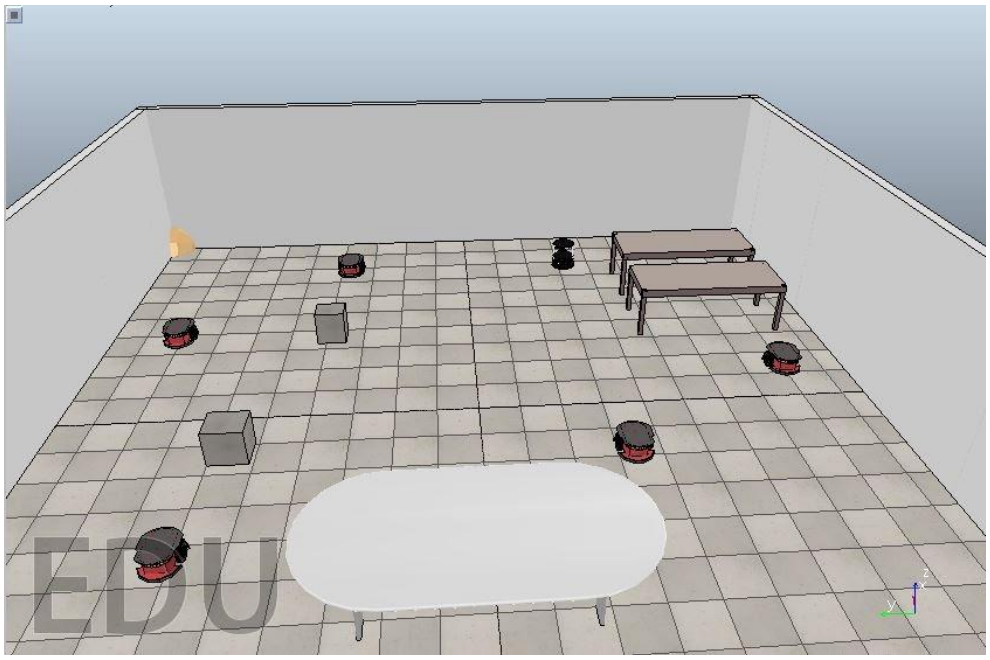
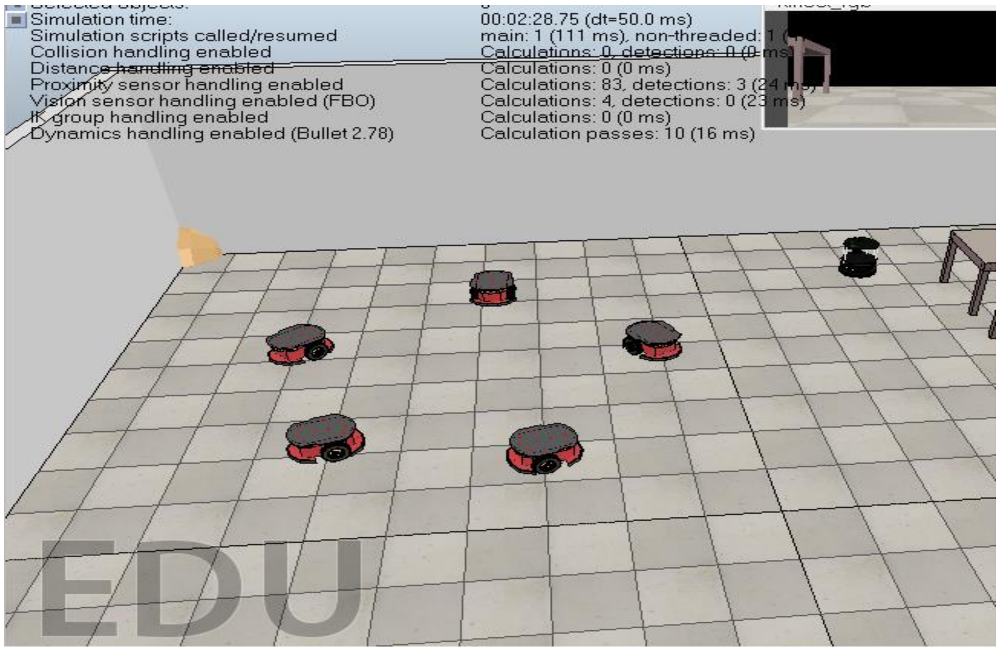
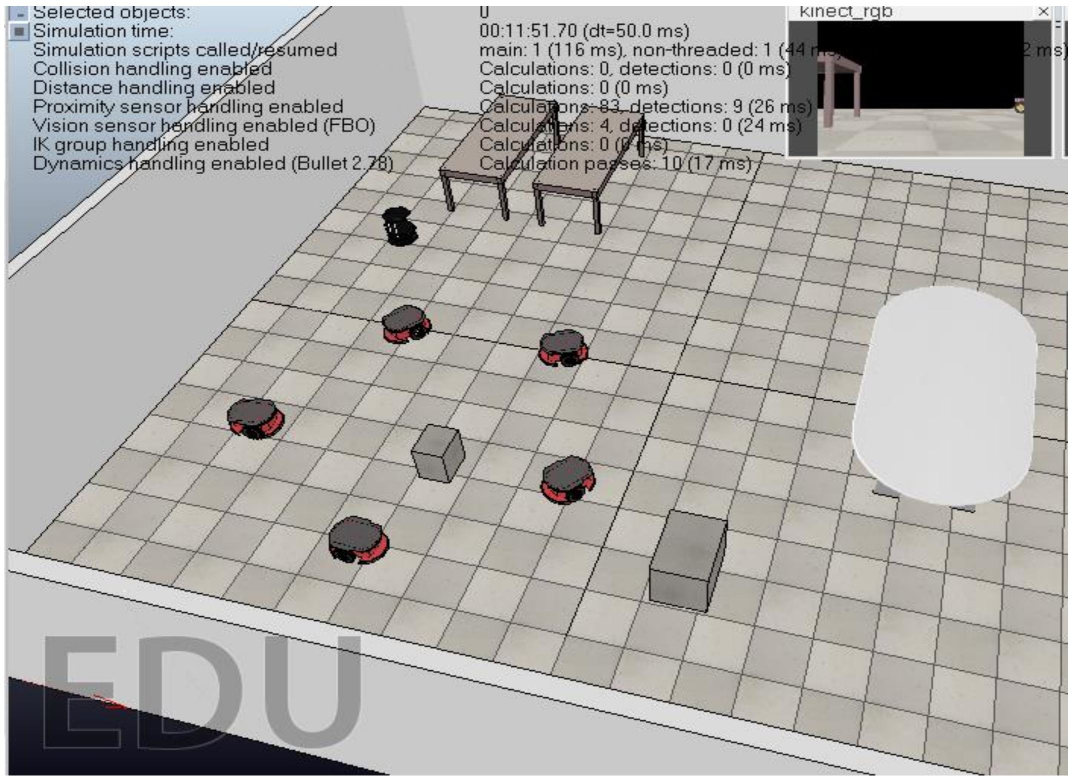


Figure 14: Initial Position of 5 Robots



```
Simulation time:                          00:02:28.75 (dt=50.0 ms)
Simulation scripts called/resumed         main: 1 (111 ms), non-threaded: 1
Collision handling enabled                Calculations: 0, detections: 0 (0 ms)
Distance handling enabled                 Calculations: 0 (0 ms)
Proximity sensor handling enabled         Calculations: 83, detections: 3 (24 ms)
Vision sensor handling enabled (FBO)      Calculations: 4, detections: 0 (23 ms)
IK group handling enabled                 Calculations: 0 (0 ms)
Dynamics handling enabled (Bullet 2.78)   Calculation passes: 10 (16 ms)
```

Figure 15: Symmetrical Pattern for 5 Robots without Obstacle



```
Selected objects:                              0
Simulation time:                               00:11:51.70 (dt=50.0 ms)
Simulation scripts called/resumed              main: 1 (116 ms), non-threaded: 1 (44 ms)
Collision handling enabled                     Calculations: 0, detections: 0 (0 ms)
Distance handling enabled                      Calculations: 0 (0 ms)
Proximity sensor handling enabled              Calculations: 83, detections: 9 (26 ms)
Vision sensor handling enabled (FBO)           Calculations: 4, detections: 0 (24 ms)
IK group handling enabled                      Calculations: 0 (0 ms)
Dynamics handling enabled (Bullet 2.78)        Calculation passes: 10 (17 ms)
```

Figure 16: Symmetrical Pattern for 5 Robots with Obstacle

From figure 14 we see the initial position of 5 robots, figure 15, 16 respectively show the simulation result of our algorithm. It again strongly proves the robustness of our algorithm that our algorithm works for multiple numbers of robots. We expect that our algorithm will also work for even higher numbers of robots with or without obstacle.

8. Constraints

There are some constraints in our code due to the working environment of VREP. VREP has some constraints so we can't follow the same desired input like in real life and so results are not always as desired. It is also to be mentioned that there is a factor called error_factor in our code which is the error patience of our implementation. In code we have listed some exceptional cases:

1. We can't place obstacles at the destination point. But it can be overcome if we increase the error factor. This is not constraints of our algorithm but in simulation environment.

2. We can't place any robot inside the 1st phase radius initially before run, if we do so then the symmetry will be a congested symmetry (which may lead to an incomplete simulation).

3. Now robot is sorting shortest distance serially. So you might not see a robot to pick the absolute nearest distance.

4. We can't give that input which leads to negative destination coordinates. This is constraints in the VREP environment simulation, not in our algorithm.

5. As we haven't use native parallel property of a robotics system, we use the calculation which would have been used for parallel operation. Thus it takes longer time to finish simulation.

Beside these exceptional cases our simulation runs successfully and forms a symmetrical pattern.

9. Real life application and Usefulness

Swarm robots are mostly used for aiding purpose to people. Our algorithm can certainly do a task of aid where symmetrically work is required. Suppose an object can be handled properly if all the swarm robots are in equidistance and distribute the weight properly. Another application can be in decoration, lighting events. UAV can certainly follow this algorithm to make a symmetrical visually charming pattern in the air around a specific given point. We don’t need to supervise anything about symmetrical positions if we use it. Even firefighter robots can implement this algorithm to extinguish fire from a desired fire location working with symmetry can be a strong opposition to fire. In military purposes it can also help that kind of autonomous robot. Suppose military robot has to attack an enemy. Enemy position can be the desired center and military robots can attack simultaneously around it to make life of the enemy more difficult.

10. Conclusion

In the project, an algorithm was successfully implemented for symmetrical shape formation using multiple robots. This algorithm follows the basic concept of swarm robotics and works both for environment with and without obstacles. But in the process, there were also some challenges faced. Further research can be done to reduce the operation time, to increase the accuracy rate and also for successful implementation in different environments in real life conditions.