Intern Report

|  |  |  |
| --- | --- | --- |
| **S.no** | **Topic** | **Page no.** |
| **1** | **Path planning using potential field algorithm** | **3-5** |
| **2** | **Path planning using threshold algorithm** | **6** |
| **3** | **Swarm Robotics** | **7-18** |
| **3.a** | **Pattern Formation Using Swarm Of P3DX** | **7-11** |
| **3.b** | **The Coverage Problem Using Swarm Robotics** | **12-13** |
| **3.c** | **The Convergence of Swarm Robots to a point** | **13** |
| **3.d** | **Platoon Formation** | **14-15** |

|  |  |  |
| --- | --- | --- |
| **3.e** | **Swarm Robotics Go To Goal And obstacle avoidance** | **16-18** |

**I. Path Planning For P3DX Using Potential Field Method**

**I. INTRODUCTION**

Theoretically finding the shortest safest path between two points in an environment with obstacles can be well defined and solved but practically it is tough to do so, and many methods are introduced to solve this problem. Potential field method is closely related to the electrical charge concept. If we consider the robot as an electrically-charged particle, then obstacles should consist of same electrical charge so that the robot repel from the obstacles.

The basic concept of this method is to fill the robot's workspace with an artificial potential field in which the robot is attracted towards the target and is repulsed away from the obstacles. This method is easy and attractive because of its elegant mathematical analysis and simplicity. Previously, potential field method was used to deal with mobile robot path planning in stationary environments, where targets and obstacles both are stationary. However, in many real-life implementations, the environment is dynamic and not only the obstacles are moving, but the target is also moving.

**II.PRINCIPLE**

***A. Potential Field***

*1)* In an environment, there are two main objects: target and obstacles.

*2)* The target should attract a robot to its position and obstacles should repulse it away from them.

*3)* The robot attracts to the target with attractive force FG and it repulses away from the obstacles with repulsive force FObs

4) Consider two-dimensional space X and Y, then the total force F in a certain point will be given as a vector sum:

𝑭⃗ (𝒙,𝒚)=𝑭𝑮⃗(𝒙,𝒚)+Σ 𝑭⃗ 𝒐𝒃𝒔𝒊(𝒙,𝒚)

𝑭⃗ (𝒙,𝒚) =− 𝛁⃗ 𝑽

*5)* At any point (x, y) in the plane, summation of forces at point (x, y) results in net force produced by all the targets and obstacles. The net force is a function of position vector force field and it represents the resultant force obtained by summing the effects of the attraction force of the target and the repulsion forces of the obstacles.

6*)* Vector field F is created from the collection of all strengths for all combinations of (x, y).

*7)* Force gradient is:

**∇⃗⃗ 𝑉=[𝜕𝑉 /𝜕𝑥 𝜕𝑉/𝜕𝑦]𝑇**

**∇⃗⃗ 𝑉=[𝐹𝑥 𝐹𝑦]𝑇**

Here, x and y components are as follows:

**𝐹𝑥(𝑥,𝑦)=−𝜕𝑉(𝑥,𝑦)/𝜕𝑥**

**𝐹𝑦(𝑥,𝑦)=−𝜕𝑉(𝑥,𝑦)/𝜕𝑦**

Suppose target is located at (x1, y1) and the currently robot is at position (xt, yt) .

Linear attractive potential is given by V(r) = k\*r, where the distance between robot and target is,

**𝑟=[(𝑥−𝑥1)2+(𝑦−𝑦1)2]1/2**

By the chain rule, the gradient force generated is

**𝐹𝑥 =−𝜕𝑉/𝜕𝑥=−𝑘 \*1/2\*[(𝑥−𝑥1)2+(𝑦−𝑦1)2]−1/2 \* 2(𝑥−𝑥1)**

**𝐹𝑥 =𝑘 \*(𝑥1−𝑥)/𝑟**

**𝐹𝑦=−𝜕𝑉/𝜕𝑦=−𝑘 \*1/2\*[(𝑥−𝑥1)2+(𝑦−𝑦1)2]−1/2 \* 2(𝑦−𝑦1)**

**𝐹𝑦=𝑘 \*(𝑦1−𝑦)/𝑟**

Here, we have two components of unit vector:

**𝑒𝑥 = (𝑥1−𝑥)/𝑟**

**𝑒𝑦 =(𝑦1−𝑦)/𝑟**

Due to these, attractive force is independent of the distance from the target. So, attractive effects of target have an unbounded range of influence.

Suppose, it is desired to avoid an obstacle which is located at (x1, y1) and the robot’s current position is (xt, yt).

Repulsive potential is given by

**V(r) = k/r**

where the distance from the obstacle is,

**𝑟=[(𝑥−𝑥1)2+(𝑦−𝑦1)2]1/2**

By the chain rule, the gradient force generated is

**𝐹𝑥=−𝜕𝑉/𝜕𝑥= −𝑘(𝜕𝑟/𝜕𝑥)-1**

**𝐹𝑥=−𝑘 (−𝑟−2 )\*1/2\*[(𝑥−𝑥1)2+(𝑦−𝑦1)2]−1/2 2(𝑥−𝑥1)**

**𝐹𝑥=𝑘 (𝑥1−𝑥)/𝑟3**

**𝐹𝑦=−𝜕𝑉/𝜕𝑦= −𝑘(𝜕𝑟/𝜕𝑦)-1**

**𝐹𝑦=−𝑘 (−𝑟−2 )\*1/2\*[(𝑥−𝑥1)2+(𝑦−𝑦1)2]−1/2 \*2(𝑦−𝑦1)**

**𝐹𝑦=𝑘 (𝑦1−𝑦)/𝑟3**

Here, we have two components of unit vector:

**𝑒𝑥 = (𝑥1−𝑥)/𝑟**

**𝑒𝑦 =(𝑦1−𝑦)/𝑟**

Force follows repulsive square law, which is same as the electrostatic force or gravitational force. Obstacle’s repulsive effects have a limited influence range.

This method involves construction of artificial potentials which can be attractive or repulsive depending on whether it is a target or an obstacle in the workspace. At each instant, the robot heads in the direction of the negative of the gradient of the potential which is almost like an artificial force applied to the robot.

Attractive potential function and repulsive potential function are additive and can be developed independently. The total potential function is the sum of these potentials and is used to control the robot to move in a desired trajectory, to reach the target and avoid obstacles at the same time. Thus, the trajectory of the mobile robot is obtained by following the total force generated by summing attractive and potential forces. This total force is calculated as the negative gradient of the total potential function

**B. Path Planning Algorithm**

The general Flowchart of the path planning:



The algorithm of path planning works as follows :

1) Identify the work space of robot.

2) Start to move from start point towards the end-point.

3) If the target is reached then the robot stops moving & display the path generated and if the target is not reached, then see step-4

4) If there is an obstacle, then search for the shortest path. Move left or right to the obstacle.

5) If the path found is near to the target then follow it, else follow the left path.

6) Repeat steps 4 and 5 until the target is not reached.

7) The robot stops moving once it reaches the target by covering the optimum path.

**Drawbacks:**

When there is a local minima the bot has a chance of getting trapped in the local minima . So to check if the bot reached desired goal and not the minima, we need to make the bot move randomly to make sure it is not stuck in the local minima.So that it escapes local minima and reaches global minima(goal).

**II. Implementing Path Planning For P3DX Using Threshold Method**

**I. Principle**

The bot position p(x,y) and goal g(x,y) is known to us. The position and orientation of goal is calculated with respect to the P3DX .Then the bot is made to move along this path. When the bot encounters an obstacle using its ultrasonic sensors within a threshold limit of distance,the go to goal algorithm is disabled and avoid obstacle is enabled . When it does not detect any obstacle within the threshold limit , avoid obstacle algorithm is disabled and go to goal algorithm is enabled.

The path the robot takes is the best path but not the shortest path.

**II. Algorithm**

1) Identify the work space of robot.

2) Start to move from start point towards the end-point.

3) If the target is reached then the robot stops moving & display the path generated and if the target is not reached, then see step-4

4) If there is an obstacle, then bot moves away from the obstacle . Move left or right to the obstacle.

5) Once there is no obstacle around it ,it starts moving towards the goal.

6) Repeat steps 4 and 5 until the target is not reached.

7) The robot stops moving once it reaches the target by covering the best path.

**III. Swarm Robotics**

**A. Pattern Formation Using Swarm Of P3DX**

**I. Introduction**

This is a replica of work done by GRITS laboratory in Virginia tech. My work greatly resembles this work.

[https://www.youtube.com/watch?v=TqgHGUH61Hg]. In this link the bots spell G R I T S. It is also possible to do this with my work.

Multi-robot or swarm robotic systems consist of a large number of small and simple autonomous robots, each having limited communication/sensing capability and computational resources. Developing self-organising collective multi-robot systems has become an active research area in recent decades due to their attractive properties such as robustness to faults and damages, adaptability to unknown environments and cost efficiency . Recent advances in robotics make it possible to build and operate a large number of inexpensive robots for various tasks beyond the scope of any single robots. 

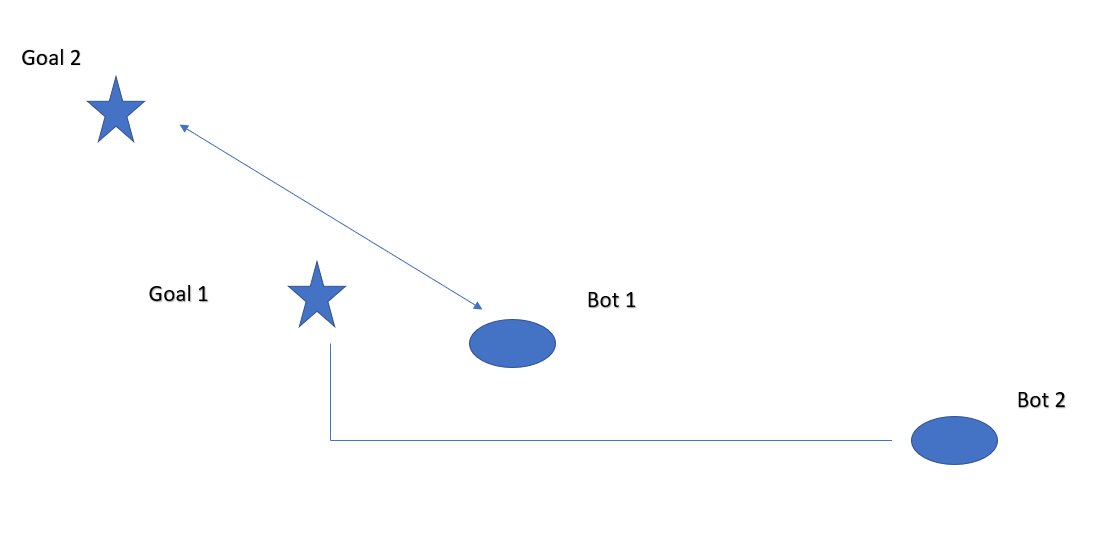
In particular, considerable interests have been paid to pattern formation allowing multi-robots to move in a loose or tight formation as a basic function for accomplishing a given mission. The pattern formation problem in multi-robot systems represents the coordination of a group of robots to generate and maintain a formation with a certain shape, in which the shape can be either a pre-defined pattern or adaptively formed in a self-organised way through local interactions with neighbouring robots and with the environment. In the former case of using pre-defined patterns, a group of autonomous robots shall follow a predefined trajectory while keeping a pre-specified spatial pattern and individual robots should maintain a specific relative orientation and distance between each other. In this pattern formation process, different control methodologies can be applied depending on the objectives.

The method used here was inspired by the work done in GRITS laboratory in Virginia Tech.

**II. Principle**

The co-ordinate of shapes are given. The bots position is known to us. The objective is to minimize time , energy and to make the bots move intelligently to minimize these factors.

This can be done by implementing Hungarian algorithm. The matrix of POSITION x DISTANCE OF BOTS FROM DESTINAION is created . By Hungarian Algorithm total distance travelled ,time and energy is minimized .



In this case to share the work among the bots , we expect bot 1 to move to goal 2 and bot 2 to move to goal 1 . By this the total work and energy spent by each bot is shared and is almost same. This is exactly what is done by Hungarian algorithm.

**III. Algorithm**

1] Identify the workspace of the robot.

2] Check if the formation is formed. If not go to step 3 else go to step 8]

3] Get the Co-ordinates of each robots.

4] We know the co-ordinates of shape to be formed and co-ordinates of bots. We form a matrix consisting of distance of each bot from each co-ordinates of shape. The entry of each cell corresponds to distance of a bot to a point in the shape.

5] We apply the Hungarian algorithm to the matrix . And know where each bot has to go.

6] The bot is made to go to its destination by go to goal algorithm.

7] go to step 2.

8] Finish

**IV. Hungarain Algorithm**

Given n{\displaystyle n}nnmn n workers and tasks, and an *n*×*n* matrix containing the cost of assigning each worker to a task, find the cost minimizing assignment.

First the problem is written in the form of a matrix as given below

|  |  |  |  |
| --- | --- | --- | --- |
| a1 | a2 | a3 | a4 |
| b1 | b2 | b3 | b4 |
| c1 | c2 | c3 | c4 |
| d1 | d2 | d3 | d4 |

where a, b, c and d are the workers who have to perform tasks 1, 2, 3 and 4. a1, a2, a3, a4 denote the penalties incurred when worker "a" does task 1, 2, 3, 4 respectively. The same holds true for the other symbols as well. The matrix is square, so each worker can perform only one task.

**Step 1**

Then we perform row operations on the matrix. To do this, the lowest of all *ai* (i belonging to 1-4) is taken and is subtracted from each element in that row. This will lead to at least one zero in that row (We get multiple zeros when there are two equal elements which also happen to be the lowest in that row). This procedure is repeated for all rows. We now have a matrix with at least one zero per row. Now we try to assign tasks to agents such that each agent is doing only one task and the penalty incurred in each case is zero. This is illustrated below.

|  |  |  |  |
| --- | --- | --- | --- |
| 0 | a2' | a3' | a4' |
| b1' | b2' | b3' | 0 |
| c1' | 0 | c3' | c4' |
| d1' | d2' | 0 | d4' |

The zeros that are indicated as 0' are the assigned tasks.

**Step 2**

Sometimes it may turn out that the matrix at this stage cannot be used for assigning, as is the case for the matrix below.

|  |  |  |  |
| --- | --- | --- | --- |
| 0 | a2' | a3' | a4' |
| b1' | b2' | b3' | 0 |
| 0 | c2' | c3' | c4' |
| d1' | 0 | d3' | d4' |

In the above case, no assignment can be made. Note that task 1 is done efficiently by both agent a and c. Both can't be assigned the same task. Also note that no one does task 3 efficiently. To overcome this, we repeat the above procedure for all columns (i.e. the minimum element in each column is subtracted from all the elements in that column) and then check if an assignment is possible.

In most situations this will give the result, but if it is still not possible then we need to keep going.

**Step 3**

All zeros in the matrix must be covered by marking as few rows and/or columns as possible. The following procedure is one way to accomplish this:

First, assign as many tasks as possible.

* Row 1 has one zero, so it is assigned. The 0 in row 3 is crossed out because it is in the same column.
* Row 2 has one zero, so it is assigned.
* Row 3's only zero has been crossed out, so nothing is assigned.
* Row 4 has two uncrossed zeros. Either one can be assigned (both are optimum), and the other zero would be crossed out.

Alternatively, the 0 in row 3 may be assigned, causing the 0 in row 1 to be crossed instead.

|  |  |  |  |
| --- | --- | --- | --- |
| 0' | a2' | a3' | a4' |
| b1' | b2' | b3' | 0' |
| 0 | c2' | c3' | c4' |
| d1' | 0' | 0 | d4' |

Now to the drawing part.

* Mark all rows having no assignments (row 3).
* Mark all (unmarked) columns having zeros in newly marked row(s) (column 1).
* Mark all rows having assignments in newly marked columns (row 1).
* Repeat for all non-assigned rows.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| × |  |  |  |  |
| 0' | a2' | a3' | a4' | × |
| b1' | b2' | b3' | 0' |  |
| 0 | c2' | c3' | c4' | × |
| d1' | 0' | 0 | d4' |  |

Now draw lines through all marked columns and **unmarked** rows.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| × |  |  |  |  |
| 0' | a2' | a3' | a4' | × |
| b1' | b2' | b3' | 0' |  |
| 0 | c2' | c3' | c4' | × |
| d1' | 0' | 0 | d4' |  |

The aforementioned detailed description is just one way to draw the minimum number of lines to cover all the 0s. Other methods work as well.

**Step 4**

From the elements that are left, find the lowest value. Subtract this from every unmarked element and add it to every element covered by two lines.

Repeat steps 3–4 until an assignment is possible; this is when the minimum number of lines used to cover all the 0s is equal to max(number of people, number of assignments), assuming dummy variables (usually the max cost) are used to fill in when the number of people is greater than the number of assignments.

Basically you find the second minimum cost among the remaining choices. The procedure is repeated until you are able to distinguish among the workers in terms of least cost.

**B. The Coverage Problem Using Swarm Robotics**

**I. Introduction**

One of the fundamental applications of swarm robotics is coverage. Covering is the first step in many, more complex, algorithms and practical implementations of swarm robotics such as search and rescue, area surveying, and space exploration . While many solutions have been proposed such as adapting Lloyd’s Algorithm or the Stackelberg leadership model , they are not general enough to be implemented on simple robotic swarms. Many of the proposed solutions start with certain assumptions about the swarm which are not practical for many implementations. Such assumptions include the availability of certain sensors and capabilities of each robot agent in the swarm, global knowledge of the environment or the use of “leader” robots. The Algorithm here, also has some assumptions about the robot agent swarm, but they are easily met by many commercially available robots for swarm purposes. The Algorithm, is intended for a relatively small homogeneous swarm with the capability of sensing obstacles and other robots around them. The experiments conducted were done by using simulation software. This is meant to show that the Algorithm is implementable and a viable solution to the coverage problem.

**II. Principle**

The Algorithm is inspired by the way gas molecules expand to fill a container . The gas molecules move from areas of high density to low density, until the container is at equilibrium. The idea behind the Algorithm is that each robot agent act like a gas molecule, each moving away from the more densely populated areas to ones with less robots. By doing this the swarm spreads out to cover an area evenly.

This can be achieved by the following equation:

**pinew=pi +c\*{ \*dist((pj-pi))\*(pj-pi) } [1]**

where pi and pj are the position vectors of i th and j th robots respectively.

c is a constant .

w is the weight associated with each bot . Here it is -1.

The equation finds the displacement vector in the direction opposite to the direction of the vector connecting the centroid of the robots and the i th robot.

The equation above is of great importance. On can also achieve different formations by correctly assigning the weight.

By increasing the weight corresponding to a bot , making it negative and by carefully assigning the weight we can make the other bots follow the required bot.

**III. Pseudo Code**

for each bot :

calculate the pinew using the equation [1]

for each bot :

if (obstacle in front):

avoid obstacle algorithm

else:

move forward( pinew )

**C. The Convergence of Swarm Robots to a point**

**I. Principle**

In the following equation :

**pinew=pi +c\*{ \*dist((pj-pi))\*(pj-pi) }**

by making the weight (w) = 1 , we can make all bots converge to a point in the work space.

**II. Pseudo Code**

for each bot :

calculate the pinew using the equation [1]

for each bot :

if (obstacle in front):

avoid obstacle algorithm

else:

move forward( pinew )

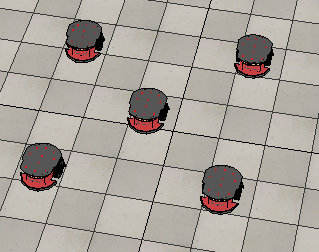
**D. Platoon Formation**

**I. INTRODUCTION**

Formation control is an important issue in coordinated control for a group of unmanned autonomous vehicles/robots. In many applications, a group of autonomous vehicles are required to follow a predefined trajectory while maintaining a desired spatial pattern. Moving in formation has many advantages over conventional systems, for example, it can reduce the system cost, increase the robustness and efficiency of the system while providing redundancy, reconfiguration ability and structure flexibility for the system .

Formation control has broad applications, for example, security patrols, search and rescue in hazardous environments. In military missions, a group of autonomous vehicles are required to keep in a specified formation for area coverage and reconnaissance; in small satellite clustering, formation helps to reduce the fuel consumption for propulsion and expand their sensing capabilities . In automated highway system (AHS), the throughput of the transportation network can be greatly increased if vehicles can form to platoons at a desired velocity while keeping a specified distance between vehicles . Research on formation control also helps people to better understand some biological social behaviours, such as swarm of insects and flocking of birds.

**II . Formation Chosen In This Project**

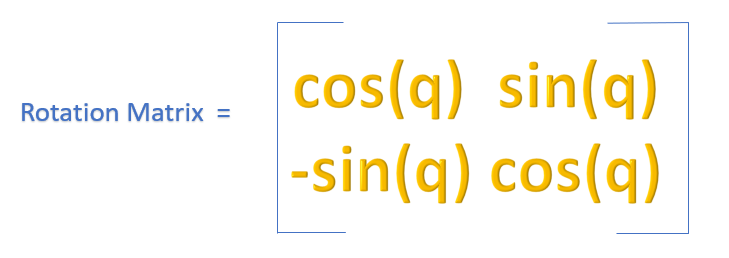


**III . Why This Specific Formation**

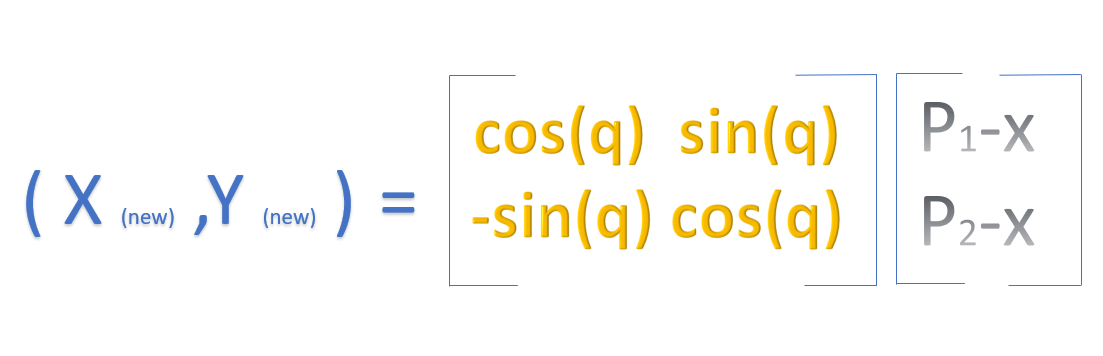
1] The use of sensor fusion can be greatly maximized in this formation. 2] As the formation is compact, the stability of the platoon increases. 3] The platoons can avoid obstacles easily in this formation. 4] The centre bot acts as the leader bot. 5] Since the leader bot is in the centre, the subordinate bots protect the leader from external attacks .

**IV . Formation of Platoon**

The formation can be accomplished by using the shape formation algorithm discussed in Swarm robotics section A. The co-ordinates of the square are given . The position of five bots are known to us. The co-ordinates we get are with respect to the world frame. We must transform the four subordinate co-ordinates in the world frame to the leader frame (the fifth bot frame). This can be done using the rotation matrix.



Where ‘q‘ is the angle by which the leader bot frame is rotated with respect to world frame.



(Xnew,Ynew) are the co-ordinate of the of the subordinate bot with respect to the leader bot frame.

(P1,P2) are the co-ordinate of the leader bot w.r.t the world frame.

(X,Y) are the co-ordinate of the subordinate bots w.r.t the world frame.

**V. Pseudo code**

While formation is maintained:

Get position of sub-ordinate and leader bot.

Transform the sub-ordinate point in world frame to leader bot frame.

Make the sub-ordinate bots go to the square co-ordinates using the shape formation

algo discussed in swarm robotics section A.

End;

**E. Swarm Robotics Go To Goal And obstacle avoidance**

**I. Introduction**

Robotics has been used in areas like agriculture, medical, automation education etc. The main aim in swarm robotics now is to avoid an obstacle based on the inputs provided by the members of platoon bots. Robotics is a branch of technology which deals with the study, design, and implementation of robots, Whereas a robot is a programmed machine which is capable of carrying out some specified task which has been assigned by the programmer or the user. Swarm robotics is the implementation of swarm intelligence into robotic for obstacle avoidance.

Swarm intelligence is the artificial intelligence based on the study of collective behaviour of decentralized and self-organized systems. The natural swarm intelligence has been implemented for the master and the slave robot, the slave when encounters any obstacles it shares the information of the obstacle with the master robot, the sensors used are the ultrasonic sensors . These senses any obstacle in the path of the robots and sends the signal to the controller, based on the information from the sensor the controller guides the robot in a clear path, the same information will be shared with the slave robot to overcome the obstacle as it will be following the controls of the master robot. This sharing of information with the slave robot helps to reduce the time and also the efforts required for slave robot to avoid the obstacle, eventually reducing cost as well as time.

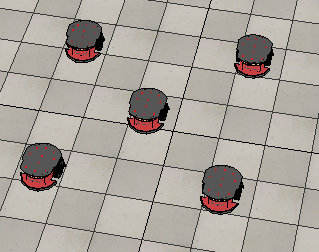
**II. Principle**

To avoid obstacles the swarm robots use ultrasonic sensors to detect obstacle. We assume that the slave bot sensor data is known to the leader bot.

The principle used here to avoid obstacle and to reach goal is the Threshold Method discussed in section II .

**III. Theory**

The swarm formation used to avoid obstacles is given below:

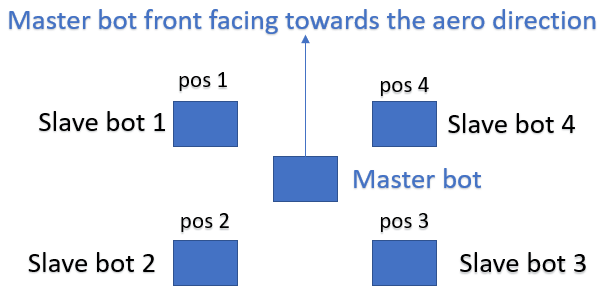


Each P3dx has 16 ultrasonic sensors around it. The slave sensor data is shared with the master.

The motion control commands given to the slaves is same as the motion control command of the master bot. Hence, all the bot will have a synchronized movement and will follow the motion pattern of the master bot.

The formation chosen above makes sure that the entire P3dx platoon stays as a single P3dx bot. That is the control given to single bot while using threshold method can be used here as the control of the master bot which in turn is given to the slave bots.

**IV. Working**



The slave bot corresponding to pos1, pos2, pos3, pos4 with respect to the orientation of the master bot are named as slave bot 1,2,3,4 respectively .

The sensor data from the slave bot are taken by the master bot as follows:

Slave bot 1: Data of Ultrasonic sensor 1,2,3 and 4.

Slave bot 2: Data of Ultrasonic sensor 16 .

Slave bot 3: Data of Ultrasonic sensor 8 and 9.

Slave bot 4: Data of Ultrasonic sensor 5,6,7 and 8.

The master bot Ultrasonic sensor data is not used.

The minimum of all these sensor data is found out. If the minimum sensor data value is less than a threshold value the master bot enables obstacle avoid algorithm. Else it enables go to goal algorithm. As master and slave bots share the same motion control ,they avoid obstacles and reach goal in a synchronized manner.

**V. Pseudo code**

While ( goal is reached):

Sensor\_val1=[ ultrasonic sensor 1,2,3 and 4 data of slave bot 1]

Sensor\_val2=[ ultrasonic sensor 16 data of slave bot 2]

Sensor\_val3=[ ultrasonic sensor 8 and 9 data of slave bot 3]

Sensor\_val4=[ ultrasonic sensor 5,6,7 and 8 data of slave bot 4]

Min\_sensor\_val= min of (Sensor\_val1, Sensor\_val2, Sensor\_val3, Sensor\_val4)

If ( Min\_sensor\_val <threshold value)

Avoid\_obstacle\_algorithm()

Else

Go\_to\_goal\_algo()

End;

The End