SPECIAL TOPIC PRESENTATION REPORT: SWARM ROBOTICS

PROJECT GUIDE: Dr. Venkatarangan MJ
SUBJECT CODE: UE17EE356A
Sumanth V Udupa: PES1201700525
Ajay Victor: PES1201701170
Pramod Kehsav: PES1201700582

ABSTRACT

Swarm robotics is a new approach to the coordination of multi-robot systems which consist of large numbers of relatively simple robots which takes its inspiration from social insects. The most remarkable characteristic of swarm robots are the ability to work cooperatively to achieve a common goal.
The initial section introduces the concept of swarm intelligence and the special features of swarm as well as the comparison between a single robot and a system consisting of multi-robots is brought out.
The main section describes the hardware part consisting of components used, building the bots and testing the code and debugging, and the software part which comprises the algorithms developed to bring about swarm nature,obstacle avoidance by robots, master-slave communication and simulations run in softwares such as ROS, gazebo.

Keywords: swarm; hardware; obstacle avoidance;swarm algorithms; ROS; gazebo;

INTRODUCTION

The term “Swarm Intelligence” refers to sophisticated collective behavior that can emerge from the combination of many simple individuals, each operating autonomously.
The essential characteristics of swarm intelligence consist of a biologically inspired emphasis on decentralized local control and local communication, and on the emergence of global behavior as the result of self organization.The application of swarm intelligence principles to collective robotics can be termed “**Swarm Robotics**”.

Swarm robotic systems have become a major research area since the 1980's , as new solution approaches are being developed and validated, it is often possible to realize the advantages of swarm robotic systems .

Characteristics of Swarm Robotics:

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 heterogeneous swarm robots do exist.

Criteria for Swarm:

(i) The robots of the swarm must be autonomous robots, able to sense and actuate in a real environment.
(ii) The number of robots in the swarm must be large or at least as required by the control rules.
(iii) Robots must be homogeneous. There can exist different types of robots in the swarm, but these groups must not be too many.
(iv) The robots must be individually incapable or inefficient with respect to the main task they have to solve i.e they need to collaborate in order to succeed or to improve the performance.
(v) Robots have only local communication and sensing capabilities. It ensures the coordination is distributed, so scalability becomes one of the properties of the system.

Potential Applications of Swarm robotics

- ❖ Tackling Dangerous Tasks
  - ➢ Search and Rescue
  - ➢ Cleanup of Toxic spills
  - ➢ Demining
- ❖ Unstructured Tasks/Environments
  - ➢ Extraterrestrial or underwater excursions
  - ➢ Surveillance
- ❖ Dynamic Environments
  - ➢ Patrolling
  - ➢ Disaster recovery tasks

Real-World Applications of Swarm robotics

- DOD Micro-drones for military use.
- RoboBees
- Cost-Effective Modular Robots

OBJECTIVE

The main objectives of this project are:

- ❑ Building a platform for the use of multi-robot systems
- ❑ Establishing communication between the different bots and bringing about the Master-Slave characteristic
- ❑ Implementing an accurate form of localisation using relative positioning/ dead reckoning
- ❑ Developing a control law which will allow the robots to form formations and maintain those formations
- ❑ Constructing algorithms and to build simulations which run successfully depicting real-world environments
- ❑ Building a robust hardware model to demonstrate and verify the work done by simulations

EVALUATION OF METHODS

1. Localisation methods

Basically, there are two methods of localisation present to develop localisation. One is **global/absolute positioning** and the other is **local/relative localisation**. Our objective was to use relative positioning to determine its position using the sensors present onboard the bot and rely less on the information from its surroundings.

Global positioning:

It helps to determine the position of the robot w.r.t coordinates that already have a reference i.e a global coordinate. GPS is most commonly used for this and is accurate.

But for indoor activities, it is not suitable for our project and can be used only outdoors because GPS receivers need an unobstructed view of the sky. We are constraining our bots to work only in indoor environments for the present.

Relative Positioning:

It determines the position of the robot w.r.t its movement from its initial position. It does not take into account any data from its environment. It evaluates the position using various on-board sensors like encoders, gyroscopes, accelerometers,etc... In our case we use wheel encoders and the IMU sensor present to obtain the position and orientation respectively.

Dead reckoning:

This is a method of relative positioning. This is a popular local technique which employs simple geometric equations on odometric data to compute the robot position relative to its initial position.It also uses an inertial navigation system[INS] which consists of a gyroscope and accelerometer to take readings of yaw angle to determine its orientation.

Dead reckoning cannot be used for long distances because it suffers from various drawbacks. The kinematic model always has some inaccuracies,encoders have limited precision and there are external sources affecting the motion that are observable by the sensors. The localisation errors grow with time. Hence, usually a Kalman filter is applied which results in substantial improvement. But dead reckoning is employed in this project because it has the advantage of being low cost, simple and can estimate time faster in real time compared to absolute positioning.

2. Control approach

The design of multi-robots is usually based on the following structures:

1. **Centralised:** The organization of a system having a robotic agent (a master) that is in charge of organizing the work of the other robots. The master is involved in the decisional process for the whole team,while the other members(slaves) act according to the direction of the leader.

2. **Distributed:** The organization of a system composed by robotic agents which are completely autonomous in the decisional process with respect to each other; in this class of systems a leader/master does not exist.
   A centralised control structure is not feasible,because there are a large number of individual robots with limited sensing capabilities.
   Distributed control is required for flexibility and reliability.This is a way of distributing control to certain regions of swarm so that any effective control that should be taken is limited to the affected region/neighborhood.
   However, both distributed and centralized control approaches have contributed individually to the study of swarm robotics and have generated interesting experimental results.

3. ROS

ROS or Robot Operating System is middleware that helps in robot software development. Although ROS is not an operating system by itself it provides various services such as hardware abstraction in the form of nodes, low-level device control of commonly used and implemented functionality such as message passing between various nodes as well as package management.
ROS represents its nodes graphically in the form of **rqt-graphs**. These graphs are user friendly and give a more interactive representation of the various nodes/topics running in a given program and their background details.

4. GAZEBO

Gazebo is a 3-dimensional simulation software that can be used to simulate a virtual environment and any autonomous robot of our choice and test its functionality. It offers a physics engine with high accuracy and also provides a suite of sensors available for any robot as per its specification which makes simulation more accurate.

Gazebo can be used in correlation with ROS to develop and test various algorithms and designs developed for a specific robot and also observe its working in an environment as per the requirement.

In our project we have decided to use ROS along with Gazebo simulator to demonstrate

the working of our swarm robots in a virtual environment. To do so we have used an available robot model as provided by Gazebo known as the **Pioneer 3-DX** which is described in the next section.

5. Pioneer-3DX

The Pioneer 3-DX is an all-purpose differential-drive mobile robot platform, used for research and applications involving mapping, tele-operation, localization, monitoring, reconnaissance and other behaviors. This comes with several options including various sensors, onboard computer, autonomous navigation software and more.

Due to the physical hardware similarities like the 2 wheels supported by a castor wheel, the Pioneer 3-DX shares with the Firebird V and its available sensors we have used it as a viable replacement to simulate the working of our swarm.

6. Publisher/Subscriber

A node in ROS is a term for an executable in ROS which represents a process which is connected to the ROS network. ROS nodes share information and data between each other in the form of messages and these messages are transmitted on a topic.

Each topic has its own unique name and message type by which the data and information on it can be identified. A node can receive messages on a topic by subscribing to the topic with the help of a ***Subscriber*** and similarly a node may send messages on a topic with the help of a ***Publisher*.**

This concept of Subscriber and Publisher has been used to establish communication between the 4 robots used in our simulation.

7. Hardware Design

7.1 Bot structure

We used the 2 rectangular bots constructed by our seniors and the (+1)one constructed by us,as slave bots and a FIREBIRD V (e-Yantra) bot as the master bot which communicates with the other 3 bots.

7.2 Sensors used

For localisation of the robot using dead reckoning, mainly two parameters are required: the amount by which the bot has moved and the direction in which it has moved in, after knowing the initial position. For this, we used the in-built quadrature encoders of the motors for the slave bots and the inbuilt position encoders for the FIREBIRD V and to obtain the orientation values, we used the IMU units present in all the bots. The inertial measurement unit used was the MPU6050 which is a 6-axis IMU unit, consisting of a 3-axes gyroscope and a 3-axis accelerometer. The gyroscope gives the angular motion of the robot. This data has a lot of drift associated with it, which accumulates every time, and the error of the data is very high. Hence, the Jeff Rowberg library can be implemented to minimise the

drift to a certain extent and achieve a level of accuracy.

7.3 Microcontrollers used and establishment of communication

FIREBIRD V

The FIREBIRD V robot has a custom board made by e-YANTRA

1. Atmel ATMEGA2560 as Master microcontroller (AVR architecture based Microcontroller)
2. Atmel ATMEGA8 as Slave microcontroller (AVR architecture based Microcontroller)

The slave robots had 2 microcontrollers present on-board.

1. Arduino Mega 2560: The reason for choosing Mega was that it had sufficient pins for interrupts i.e it had 6 interrupts whereas Arduino Uno has only 2. Since the slave robots are 4 wheeled bots, the wheel encoders need to be provided to 4 interrupt pins on the microcontroller.Hence, Arduino Mega2560 was utilised.
2. Raspberry Pi: The reason behind choosing Pi was that it has a powerful processor necessary and had inbuilt libraries like ***numpy*** to do matrix operations and had high computational power to process data provided from various sensors. The added advantage was that there was an inbuilt wifi module present. The other reason behind employing a Pi for this project is to establish communication between the robots using ROS.

Basically, we utilised the Arduino MEGA to accumulate data from various sensors like wheel encoders, IMU,etc... and established **serial communication** between MEGA and Pi, such that Pi would use this data to achieve localisation, and this data was sent to other bots for developing formations.

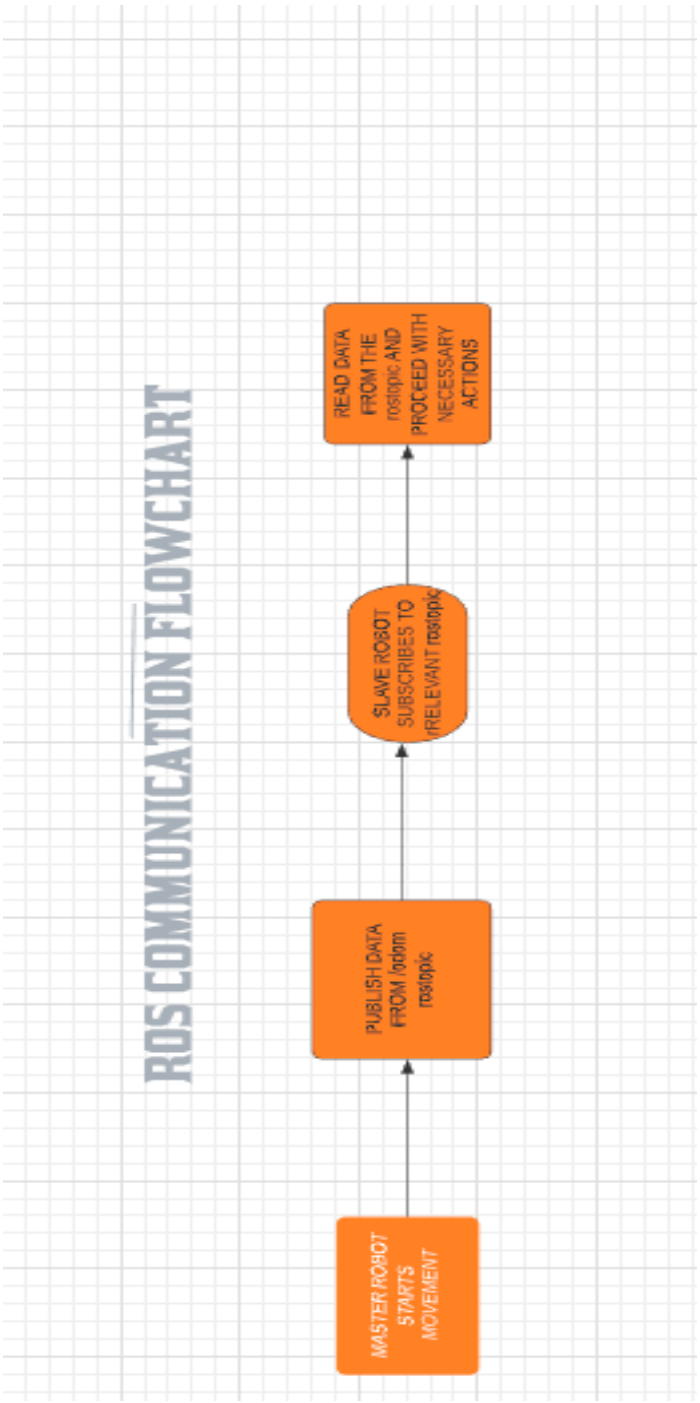
8. Communication

As presented in the previous section, Arduino acts as a sensor hub,taking in all the sensor data and sending them to the Pi, which in-turn calculates the current position of the robot w.r.t its initial position and then communicates with the other robots, regarding its current position and orientation.

For the communication between MEG2560 of FIREBIRD V and Pi, we took the help of serial communication UART. We set the same

baud rates for both the microcontrollers. We took precautionary steps such as assigning special characters, before and after the data bytes to account for any lossage in data that might have occured. Only if the data was intact and complete, necessary actions were to be taken according to the chain of commands.

ROS Communication Flowchart



SIMULATIONS

The next section briefly describes the working of the simulation carried out and the results obtained in the respective domains. Important points that need to be considered for the SIMULATIONS part of the text :

To enable smooth and simple simulations without very complex computations, only ***static environments with even surfaces*** are considered.
Another important point regarding the obstacle avoidance in the coming section , is that , at all times, the location(position and orientation) are known to the simulator.

Working of the Simulation

As mentioned above the Pioneer 3-DX has been used to simulate the working of our swarm. Here,we have launched 4 robots in the **'Single_Pillar.world'** environment in Gazebo.
The robot **r1** at the far-right acts as the Master bot and the other 3 bots represented by **r2, r3, r4** respectively are slave bots. The idea here is to move the slave robots in relation to the master bot using the leader-follower approach. The slave robots are required to maintain a fixed distance from the master bot at any given point of time. To do this the relative position of the master bot is continuously published to each of the slave robots so that they can maintain the specified distance as described by the control law.

The relative position of any of the bots may be retrieved with the help of the **/odom** topic, by the command **rostopic echo**.The relative position of the master bot is obtained with the help of its /odom topic and is used to locate it at any given point of time, which is , subsequently published to all the slave robots with the help of a publisher which is then received at the respective nodes of the slave bots with the use of the subscriber.

This kind of formation creates an image to the end user that there is a slash formation done by the swarm robots.

```
f callback(data):
      global x
      global y
      move = Twist()
      x = data.pose.pose.position.x
      y = data.pose.pose.position.y
      if(abs(goal.x - x) > 0.1) and not rospy.is_shutdown():
            move.linear.x = 0.4
      pub.publish(move)
f robot2(data):
      move = Twist()
      x2 = data.pose.pose.position.x
      y2 = data.pose.pose.position.y
      if(abs(x - x2) >= 1) and not rospy.is_shutdown():
            move.linear.x = 0.4
      pub2.publish(move)
```

Code for Formation

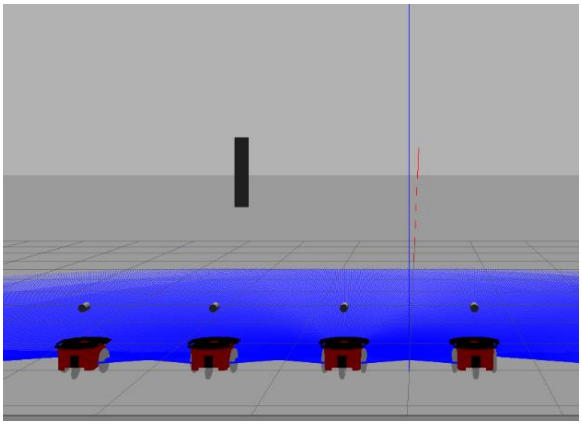
Localisation

With the help of our seniors and studying various research papers recommended by them, we chose the method of *dead reckoning* as the localisation method to be implemented as presented in the above section.

Control Law

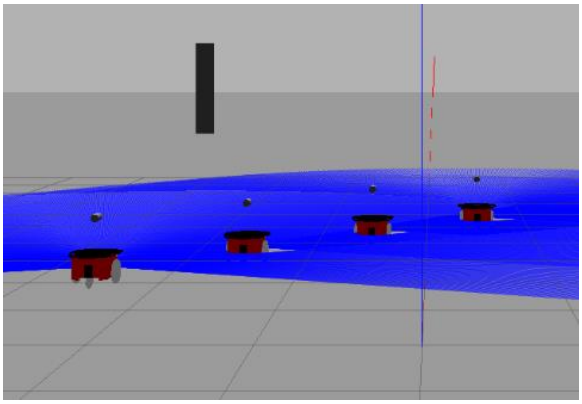
For our demonstration we have come up with a simple,rudimentary direct control law based on the relative position of the master bot at any given time.When the slave robots

receive the position of the master bot **r1**,the neighboring slave bots **r2, r3, r4** shall maintain a distance of 1m, 2m, 3m respectively, w.r.t the master robot, to ensure enough spacing and also allow the formation of a discernable pattern.



Initial position of swarm bots

After the successful completion of the basic control structure, the simulation model was run for different test cases.



Final position of swarm bots

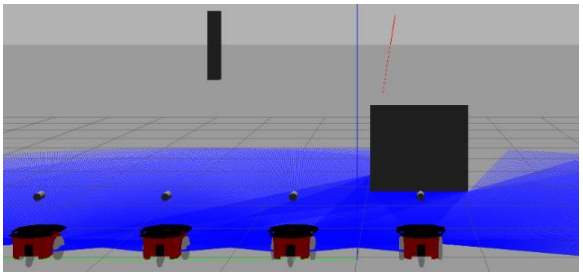
1.When the master bot is approaching any one of the slave robots randomly:
We programmed the master robot to move towards any one of the slave robots. When this happens, the slave robot, in accordance with the control law stated above, moves back, maintaining a steady distance of 1m with the master, while, simultaneously, the other 2 companions try to move towards the master to reduce the distance between them to 1m. This motion will be present with some drift error which should be reduced with the implementation of a complementary filter.

Obstacle Avoidance

To further enhance the capabilities and performance of our swarm in an actual working environment we have decided to implement obstacle avoidance. Without obstacle avoidance the working of the bots would be restricted and not versatile enough which makes the system inefficient when implemented for practical applications.

Due to the time constraint, obstacle avoidance for all the slave robots could not be implemented which is similar to the master robot.

Here we have decided to use the **/front_laser/scan** node of the pioneer bot to scan the bot's immediate environment and return the distance of the obstacles in vicinity of the bot and return said values in a python list. Return attribute of the **data.ranges** in the code section, gives the distance between the immediate obstacle and the robot in meters.



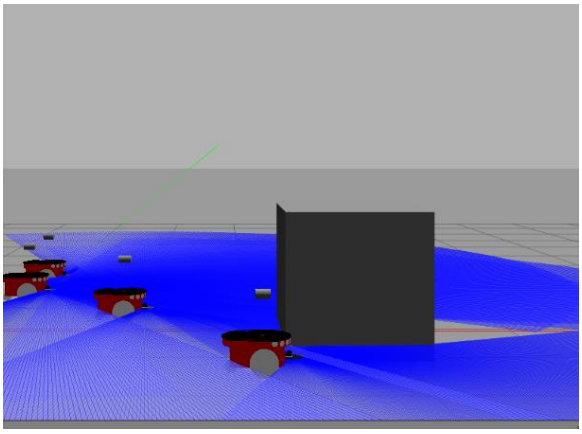
Initial position of swarm bots

We have decided to ensure that the bot only keeps a track of its immediate obstacle(within a distance of 2.5m) and one which is present exactly in front of it,for the sake of simplicity. Hence for this, only the values at the midpoint of the list are to be accounted for and processed,at any given point of time.

The bot decides on the optimum path it must take to avoid the obstacle with minimum deviation from its original path. To do this,we find the value stored at the midpoint to compare the number of sensor values on either side. Based on this,the bot then chooses the direction that has the least number of distance values so as to ensure that the deviation is as less as possible. The bot may now turn clockwise or anti-clockwise based on the above comparison until the laser scan values tend to infinity, which means the obstacle has been avoided. We should note that there will be some drift and slippage associated with the wheels of the robots,during turning,which may cause some random orientations at the position of the goal-point.

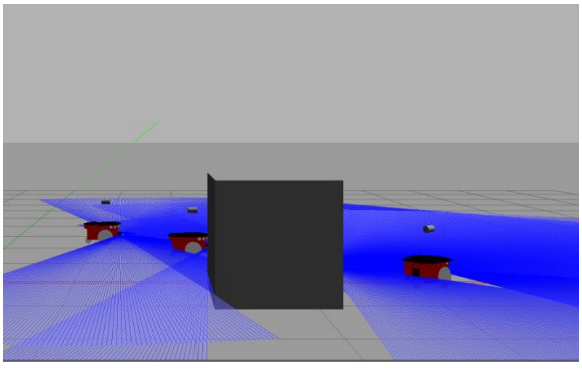
It is worthy to note that the code takes care of the drift and slippage of the wheels during the journey to the goal-point.

This algorithm works irrespective of the position of the obstacle, whether it is facing away from the centre.



Clockwise turn to avoid the obstacle

Now the bot must return back to the original path by turning in the clockwise or anti-clockwise direction as necessary based on the initial turn made which corrects the change made for obstacle avoidance and ensures the bot reaches the desired end point.



Counter-Clockwise turn to return to original path

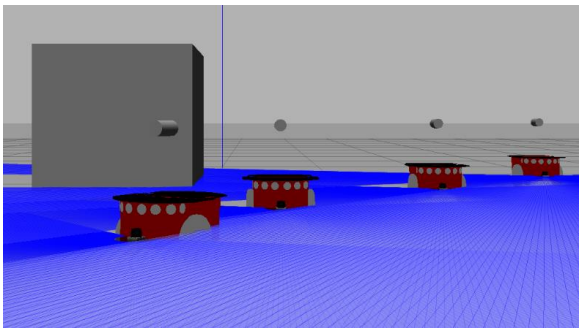
Note: values 0.4 and -0.4 are specified in the code to obtain a smooth deviation so as to avoid the obstacle, but not deviating too much off the original prescribed course. The ideal value should be selected by the implementation of a P-controller.

```
def Lasercall(data):
        obstacle_position = x + data.ranges[342]
        print("obstacle_position : ", obstacle_position)
        print("laser values : ", data.ranges[337:342])
        if(obstacle_position < goal.x):
                for i in (data.ranges[342:352]):
                        if(i < 2.5):
                                move.linear.x = 0.2
                                move.angular.z = 0.4
                                print(" has to turn right")
                for i in (data.ranges[332:342]):
                        if(i < 2.5):
                                move.linear.x = 0.2
                                move.angular.z = -0.4
                                print(" has to turn left")
                print("laser detected obstacle to tackle")
        pub.publish(move)
```

Code for Obstacle avoidance

Test cases simulated in Obstacle Avoidance:

1. **Obstacle is present farther away from the goal point or present at the goal point position:**
   In this case,the bot would continue to maintain its original path until it reaches its desired end point and comes to a position of rest.
2. **When the obstacle is present before the position of the goal point:**
   In this case, as stated above,the bot would choose the optimum deviation from its original path to avoid the object after which it will return to its original path and continue towards the desired end point.
   Note: The case where the height of the obstacle is much less than the height of the laser mounted on Pioneer-3DX is invalid for our discussion and is not considered.



Final position of the swarm bots

FUTURE WORK

1. All simulations presented need to be tested on the hardware and accordingly make adaptations for the algorithm as needed.
2. Need to formulate a more sophisticated framework and algorithm for the control structure
3. It becomes necessary to implement a more advanced filter such as the **Kalman filter** or a complementary filter for more optimal results on the control structure, as a lot of errors accumulate over time and hinder the performance of the entire system, which results in rendering our objective moot.
4. Adapt the simulations done for obstacle avoidance for dynamic environments.

REFERENCES

1. Anish Shridharan,Samarth Bhat, Vinay V.Patil - "Estimation and control of nodes in an abstract space"

2. "Qian, J., Zi, B., Wang, D., Ma, Y. and Zhang, D. (2017). The Design and Development of an Omni-Directional Mobile Robot Oriented to an Intelligent Manufacturing System. Sensors

3. "Cho, B., Moon, W., Seo, W. and Baek, K. (2011). A dead reckoning localization system for mobile robots using inertial sensors and wheel revolution encoding. Journal of Mechanical Science and Technology. **4.**"Song, Peng and Kumar, R. Vijay, "A Potential Field Based Approach to Multi-Robot Manipulation" (2002). Departmental Papers (MEAM).

5. "Introduction to Autonomous Mobile Robots", Roland Siegwart and Illah R. Nourbakhsh.

6. "An Extensive Review of Research in Swarm Robotics", Yogeshwaran Mohan, S. G. Ponnambalam,School of Engineering, Monash University, Sunway campus, 46150 Petaling Jaya, Selangor, Malaysia

7. "Swarm Robotics", Scholarpedia

8. "Robot Localization Using Relative and Absolute Position Estimates",Puneet Goel, Stergios I.Roumeliotis and Gaurav S. Sukhatme Department of Computer Science Institute for Robotics and Intelligent Systems University of Southern California Los Angeles.