# A Study in Formation Control of Swarm Robotics

Submitted by

**Bikash Kumar Yadav**

Class Roll No.- 002010502031
Reg. No.- 154155 of 2020-21
Examination Roll No.- M4CSE22031

Thesis submitted in partial fulfillment of the requirement for the Degree of Master of Engineering

In the Department of Computer Science and Engineering,
Faculty of Engineering & Technology

Under supervision of

**Dr. Chintan Kumar Mandal**
Assistant Professor,
Dept. of Computer Science & Engineering
Faculty of Engineering and Technology

Jadavpur University
kolkata – 700032
2022

**Department of Computer Science & Engineering**
**Faculty of Engineering & Technology**
**Jadavpur University**

**To Whom It May Concern,**

This is to certify that BIKASH KUMAR YADAV, Registration Number: 154155 of 2020-21, Class Roll Number: 002010502031, Examination Roll Number: M4CSE22031, a student of MCSE, from the Department of Computer Science & Engineering, under the Faculty of Engineering and Technology, Jadavpur University has done a thesis report under my supervision, entitled as “A Study in Formation Control of Swarm Robotics”. The thesis is approved for submission towards the fulfillment of the requirement for the degree of Master of Computer Sciene & Engineering, from the Department of Computer Science & Engineering, Jadavpur University for the session 2020-22.

**Dr. Chintan Kumar Mandal**
(Supervisor)
Assistant Professor
Department of Computer Science and Engineering
Jadavpur University

**Prof. Nandini Mukhopadhyay**
(Head of the Department)
Professor
Department of Computer Science and Engineering
Jadavpur University

**Prof. Chandan Mazumdar**
(Dean)
Professor
Faculty of Engineering and Technology
Jadavpur University

**Department of Computer Science & Engineering**
**Faculty of Engineering & Technology**
**Jadavpur University**

## <u>Certificate of Approval</u>

***(only in case the thesis report is approved)***

The forgoing thesis is hereby approved as a creditable study of an engineering subject carried out and presented in a manner satisfactory to warrant its acceptance as a prerequisite to the degree for which it has been submitted. It is understood that by this approval the undersigned do not necessarily endorse or approve any statement made, opinion expressed or conclusion drawn therein, but approve this thesis only for the purpose for which it is submitted.

| | |
|---|---|
| ____________________________ | ____________________________ |
| Signature of the Examiner | Signature of the Examiner |
| Date: ______________________ | Date: ____________________ |

# Declaration of Originality & Compliance of Academics Ethics

I hereby declare that this thesis contains literature survey and original research work done by me, as part of my MCSE studies. All information in this documents have been obtained and presented in accordance with academic rules and ethical conduct.

Name: Bikash Kumar Yadav
Registration No: 154155
Class Roll No: 002010502031
Examination Roll No: M4CSE22031
Thesis Report Title: A Study in Formation Control of Swarm Robotics

**Bikash Kumar Yadav**

# Acknowledgement

I would like to express my deepest gratitude to my advisor and guide **Dr. Chintan Kumar Mandal**, for his excellent guidance, caring, patience and providing me with an excellent atmosphere for doing research. I strongly believe that I got a lot of encouragement and inspiration from him throughout the project. With his invaluable guidance, this work is a successful one. I am equally grateful to **Prof. Nandini Mukhopadhyay,** Head Of The Department, Computer Science & Engineering, Jadavpur University, for his support towards our department. Last but not least, I would like to thank my parents and all respected teachers for their valuable suggestions and helpful discussions.

Regards,

Bikash Kumar Yadav
Department of Computer Science & Engineering
MCSE
Jadavpur University

# Contents

# Abstract

This thesis presents an algorithm for generating a formation for a swarm of primitive robots. The presented algorithm provide a solutions for both collective decision making and one-to-one role assignment in a connected network of swarm robots. The algorithms can be further be "trained" for a wide range of target applications because the decisions/roles are abstract and lack specific contents. In order to obtain a desired collective shape among simple robots with a small sensor and communication range, formation control techniques are used. This algorithm show how a robot swarm can autonomously choose a loop shape and form the shape in a distributed manner, without central control. Consensus processes, decision making and role assignment, are performed consecutively with a fixed but arbitrary network. Firstly robots will start dispersed in random positions, the swarm aggregate together arbitrary to form a connected network. Consensus decision making is performed with this network fixed, making a collective decision of which loop the swarm should form. Then with the same network, role assignment is performed, assigning the target positions to each robot. After these two consensus processes are done, the swarm disperses and aggregates again, this time aiming to form a loop with robots at their designated positions. The climbing method is used to permutate the robots on the loop. When the robots gets their target positions, they dynamically adjust the local shape so the loop deforms to the target one.

## Introduction

Swarm robotics systems have received a lot of attention recently in the field of robotics as a result of the growing depth of study on robotics technology and applications. A  swarm robotics system has its unique advantages compared to a single, highly integrated robot in certain application which require flexibility and adaptability. Swarm robotics system has a wide application in theoretical research and engineering application because of its scalability, flexibility, robustness, reliability and preciseness. The following are some of a swarm robotic system's primary characteristics:

- Decentralized control: Centralized command and world information are not available to individual robots..
- Autonomy: Robots are autonomous. The autonomy of every robot in the swarm is distinct from that of the individual robots.
- Localized sensing and communication: Local sensing and communication capabilities are available to each robot.
- Cooperative action: Robots work together to complete a task as intended.

Robotic swarms are uniquely qualified by these characteristics to carry out particular types of tasks efficiently, such as (a) covering a large area quickly; (b) carrying out tasks in dynamic, uncertain, and unstructured environments; (c) scaling up or down within the task at hand; and (d) carrying out tasks requiring redundancy in information.

Despite these advantages, there are still a number of engineering issues that need to be resolved before swarm robotic systems may be used to solve practical issues. Creating swarm behavior without centralized control is one of the main challenges. The majority of these earlier research have concentrated on creating a particular global swarm behavior based on straightforward rules that are carried out by the individual robots. Target global behaviors included swarm aggregation [8-10], shape/pattern formation [11-15], cooperation on construction works [16,17], collective transportation [18], structural damage detection [19,20], and navigation [21-23]. A robotic swarm must be able to carry out multiple of these global behaviors sequentially in order to solve real-world challenges. For instance, the swarm may need to aggregate, form a line, navigate, and cooperate in order to identify and retrieve an object that is hidden inside a small tunnel.

The control design of swarm robot system is quite complex because a lot of issues should be considered, such as communication, formation and coordination between robots. Therefore, a formation control algorithm is extremely required to supervise swarm robot to avoid obstacles and maintain formation while achieving a given task.

Numerous instances of shape formation in biological systems have been documented. Migratory birds fly in V formation for better efficiency. Fish swim in schools to protect themselves from predators, improve their foraging and swim more efficiently. Similarly for swarm robots, shape formation can help them move collectively; the transitions of different shapes facilitate them to navigate through irregular environment. Shape formation of complex geometric shapes or shapes resembling real life objects can be good practice to coordinate multiple robots, and they have already found applications in drone light show [24].

Robots can be organized and distributed in space in two kinds of formations: simple pattern formations, and complex shape formations. Robots can also move objects to create patterns and structures, however we limit our discussion to formations demonstrated by the robots themselves.

Simple pattern formations include aggregation, dispersion, formations of simple geometric shapes such as lines or circles, etc. There is usually no difference among the roles of the robots in these formations, so the solutions for this category are oriented to find a set of simple local behaviors that lead to the desired formations. Simple physics rules can lead to complex group behaviors. Inspired by that, artificial physics was proposed to reproduce similar complexity [25].

Complex shape formations include formations of complex geometric shapes such as star or polygon, shapes that resembles real life objects, and others with equivalent or higher complexity. In this category, role assignment is usually required to inform each robot the target position. Based on the Iterative Closest Point (ICP) algorithm, a decentralized algorithm was proposed for each agent in a network to identify the pose of a formation and to correctly assign itself to a unique position in the formation [26]. Complex shape formation control has also been implemented in unmanned aerial vehicles (UAV) with external localization system [27]. However, in many researches demonstrated above, central control for the swarm or access of global information for individual robots is still required. A distributed formation control algorithm was presented for complex two dimensional shapes with a thousand-robot swarm [28]. It requires only local sensing and communication, and even without role assignment during the formation forming process. These are done by a suite of techniques that achieves global localization through local interactions.

To summarize, most formation control researches assume that their swarm robots have the sensing/communication range covering the entire environment, or at least most of it. Their proposed algorithms perform as expected, but in experiments they often rely on a central controller to broadcast the desired information. Physical swarm robots are often very primitive and have limited sensing and communication capability, so there lacks formation control methods that align better with primitive swarm robots.

## 1.1 Related Works

Formation control, which is defined as the task of regulating a swarm of robots to avoid collisions and manage the desired formation pattering, has received a lot of attention in swarm robot research. However, in most cases the collision avoidance at the intersection points of the paths is done by controlling the velocity of the robots. Here some of the approaches which are given by researcher.

Behavior based approach is defined as one or more robots in a group can be controlled using a decentralized control method. Decentralized control method means there is no planning to generate the responses [1]. Means there is no central part manage the system, so its implemented with less communication, behavior based approach make the robots drive the controls for multiple computing objects at the same time [2]. In [3] the formation in multi robot system are implemented with behavior based method which integrated with other navigational behavior to enable a team of robots to access the goals and obstacles avoidance. The main disadvantages of this method is that the mathematical analysis of this approach is complex and therefore the concurrence of the robotics formation to the desired structure cannot be approved. But this method have an advantages design reaction is essentially unified by pairing the weights of the actions that depends on the reciprocal counterparts of the adjoining robots.

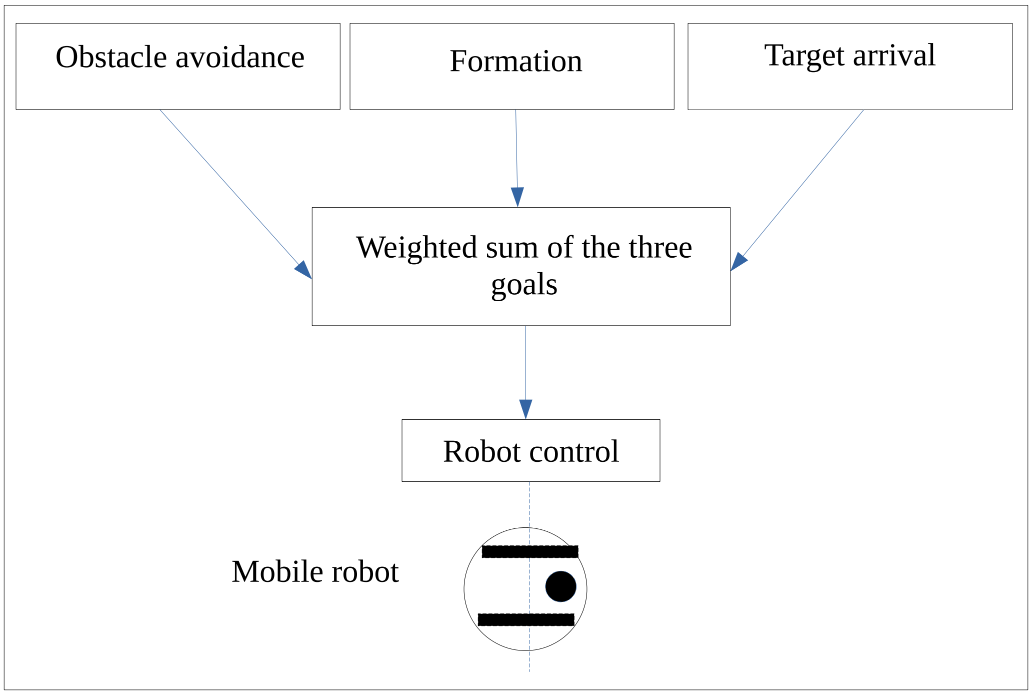


**Fig 1.1: The Behavior based approach.**

In leader follower formation method the robots in formation designed as a leader move along a predefined trajectory while the other robots are to maintain a desired formation. This strategy has the benefit of being simple to understand and put into practice, but it also has some drawbacks in that it requires centralized control (followers use the position of the leader as input control), which makes it less suited for large numbers of robots. Also there is no explicit feedback from the follower to the leader.

In leader [4] follower formation are used to perform static and moving obstacles. This approach performs well in terms of convergence stability. The multi-robot system is divided into a number of leader-follower pairs, where the leader is in charge of following a predetermined trajectory and the followers keep tabs on the leader while maintaining the desired distance and relative angle. The follower robots must also be able to follow any path that the leader robot specifies while keeping a constant distance from it. Therefor, the design of the control strategy should be based on the follower’s control signals.

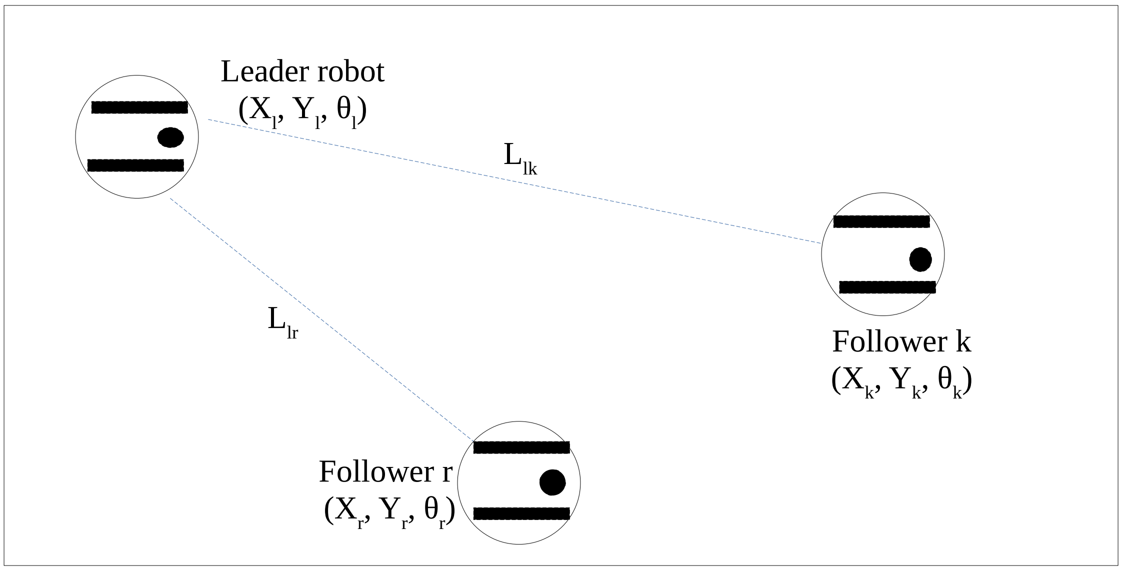


**Fig 1.2: The leader follower approach.**

In virtual structure approach each robot is given a set of controls to follow the planned trajectory of the formation as a rigid body, which treats the entire formation as a single structure. The main advantages of this approach is that a single mathematical rule translate the entire sensory input space into the actuator output space without the need of multiple rules or behavior. Utilizing vector procedures, the obtained behaviors can also be mixed. Some properties such as robustness and stability can be proved using the theoretical tools from physics, control theory and graph theory [5]. Additionally, the multi-robot formation-required collective behaviors are frequently designed using the virtual physics-based approach.

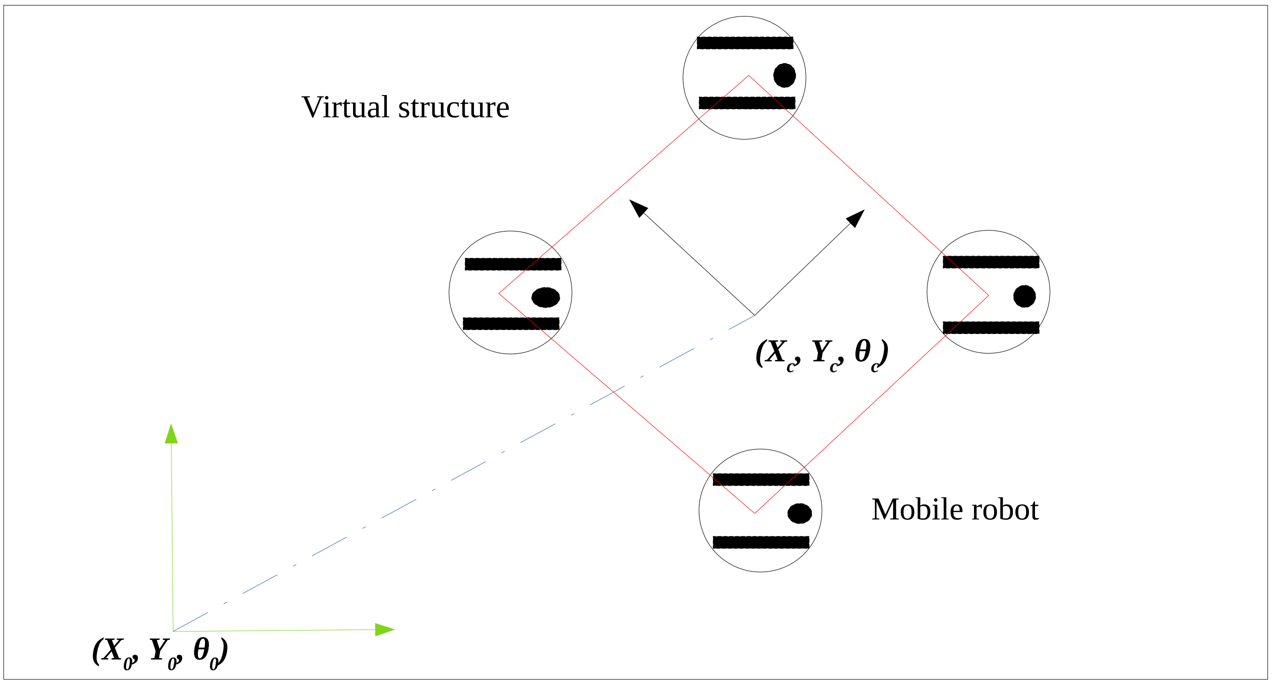


**Fig 1.3: The virtual structure.**

Another approach is displacement-based approach where robots actually control displacement of their neighboring robots to obtain the desired formation. The formation is specified by the desired displacements with respect to global coordinate system under the assumption that each robot is able to sense relative positions of its neighboring robots with respect to the global coordinate system. The intends that the robots need to know the orientation of the global coordinate system and their position with respect to the coordinate system.

In distance based approach inner robots distances are actually controlled to obtain the desired formation, which is given by the desired inner robot distances. The robots are assumed to be sense relative positions of their neighboring robots with respect to their own local coordinate system. These robots don't have to be in line with one another.

## 1.2 Objective

The Overall research goal is to advance the swarm robotics one step further into practice application by developing control algorithms and methods under consideration of the physical capabilities and limitations of the swarm robots.

Formation control aims to achieve collective shape formation among primitive robots with limited sensing and communication range. Shape formation is an important groundwork for achieving higher level collaborative tasks, and thus an important area of investigation.

It is worth noting that our design principle behind the presented work is to design in the perspective of swarm robots, instead of social biological systems. Nature has truly been a source of inspiration for swarm robotics, but the hardware capabilities of robots significantly differ from those found in nature, in both content and degree. Robots now can communicate, move, and detect more information than biological systems after only a few decades of development, but we have only seen the most recent traits of them. This is despite the fact that biological systems have evolved over billions of years. Although robots are not as good as animals at perception, with artificial intelligence and robotics advancing rapidly, the gap between the artificial and the biological systems will become smaller and narrower. We do not assume that the swarm robots are equipped with every state-of-the-art technology. Instead, we assume these robots are highly limited, so that the algorithms developed for such robots can be easily applicable for the robots with higher functionality.

## 1.3 Thesis layout

The followings chapter are as follows : Chapter 2 discusses the problem statement, assumptions and details of the proposed algorithm, Chapter 3 discusses the results of the proposed algorithm and Chapter 4 concludes the thesis with future works and challenges arising from this work.

# Concepts and Details of the Algorithm

In this chapter, we discuss the distributed formation control methods for primitive swarm robots.

And as a results we have shown the simulation of the distributed formation control methods. The simulation is simulated in Pygame2.

## 2.1 Problem Statement

We consider a bounded environment in which multiple homogeneous mobile robots are randomly dispersed, and after aggregation of those robots, formation of the shapes will be performed.

In this chapter presents distributed formation control methods, with the primary goal of aggregating the robots and forming simple or complex shapes.

## 2.2 Assumptions

The followings are assumed for the formation control methods:

- All robots are considered as dot robot.
- The robots move in a bounded and empty environment without obstacles.
- Individual robots are primitive with limited sensing and communication range, compared to the size of the environment.
- The robots work as finite state machines and without temporal memory (i.e., a time log of earlier history).
- Communication in the swarm is local; each robot can communicate only with their nearby robots within the communication range.

## 2.3 Details of the algorithm

In this section, we discuss the working principle of the proposed algorithm for formation control of swarm robotics.

The entire simulation consist of five stage to form a complex shape with swarm robots. The first stage aggregate all swarm robots in one network, second stage is a decision making stage, third stage is role assignment, fourth stage is loop formation where swarm robots will formed an ordered loop and fifth stage is loop reshaping where loop will reshape to the target shape.

## 2.3.1 Stage I

The goal of this algorithm is to aggregate all robots to form a random network. The robots used only one connection to stay stable in the group. As a result, the final network architecture always resembles a tree branch. Most robots are only serial connected. This topology is often rated as low connectivity, and takes longer time for the consensus processes. Although the robots are more easily caught in the tree network, which is an unintended advantage. The swarm robots will have more uniform coverage of the area in this way. To put this into practice, whenever a robot is part of a group and has two or more connections, it moves to the destination determined the old way. If it has only one connection, it will rotate around counter-clockwise around that neighbor robot, until it finds another neighbor. Once more, there is an exception in that there are only two members in the group. The proposed algorithm is mentioned below:

**Algorithm 1:** Network Generating Algorithm

```
input :The network size m.
output :A set of node positions N for the network.

$N_a$ ={(0,0)}
$N_w$ = neighbors((0,0))
while size($N_a$)<m do
    randomly choose a node k from $N_w$
    remove k from $N_w$
    add k to $N_a$
    for i $\in$ neighbors(k) do
        if i $\notin$ $N_a$ and i $\notin$ $N_w$ then
            add i to $N_w$
        end
    end
end
return $N_a$
```

This algorithm generates a connected network in a 2D equilateral triangle coordinate system, as shown in Fig 2.1 With three sets of parallel lines intersecting at π/3 angle, they divide the space into adjacent equilateral triangles. One of the horizontal lines is chosen as x axis, the y axis is π/3 counter-clockwise to it. The position of a node is described as two integers for the distances along the two axes. For example, the origin is the black dot at (0,0), its six neighbors are the white dots around it (1,0), (-1,0), (0,1), (0,-1), (1,-1) and (-1,1).

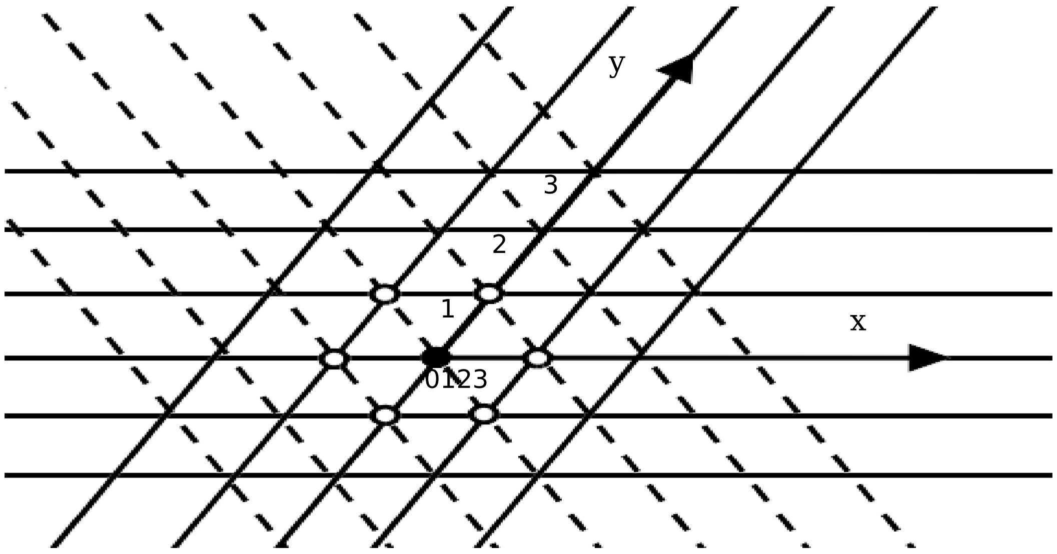


**Fig 2.1: The 2D equilateral triangle grid coordinate system.**

In Algorithm 1, $N_a$ is the set of approved node positions, $N_w$ is the set of waiting node positions. Function neighbor() returns the six neighbor of the input node. This algorithm is developed such that the new nodes grow around the boundary of the existing network.

A general state machine that runs on individual robots is designed to solve the aggregation task, and can be applied to formation of other shapes, such as a line or a loop.

### 2.3.1.1 Finite State Machine Design

The three distinct states of every individual robot are shown in Fig 2.2. In the "dormant" state, the robot is temporarily inactive; in the "active" state, the robot actively seeks for forming local connections; in the "in-group" state, the robot is in a connected group of at least two or more robots in the group. Both dormant and active robots move in straight lines and are bounced when running into the boundaries of the environment while in-group robots maintain the existing connections and adjust their relative positions for the desired formation within the group.

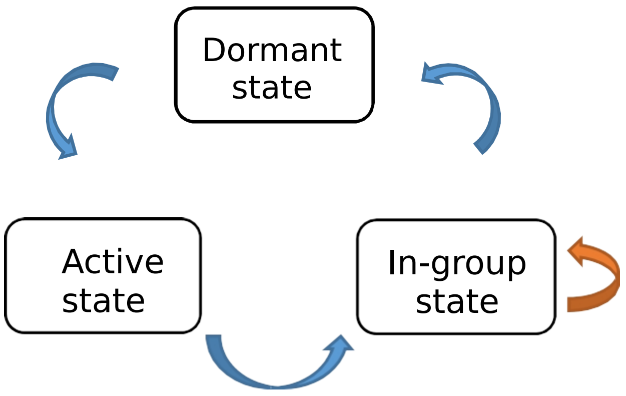


**Fig 2.2: The general finite state machine design for shape formation.**

The time limit has been applied to some of these states to trigger transition. When a robot enters the dormant state, it selects a random direction and moves in a straight line. A countdown timer for the dormant robot automatically starts with a random duration, which is randomly selected from the range of [$t_{min}$,$t_{max}$]. When the countdown timer ends, the robot enters the active state. In this state, if an in-group robot is detected within the communication range, the robot joins the group and becomes an in-group robot; otherwise if another active robot is detected, the two establishes a new group and both become in-group robots.

## 2.3.2 Stage II

After generation of the successful network, role assignment is the second stage. In this stage decision will be made that in which shape robot will be form. The decision making consists of three parts: generation of initial preference distributions, updating individual preference distributions, and lastly uncertainty reduction. On each robot in the swarm, the first part runs only once to generate the initial probability distributions. The later two pats will run iteratively until consensus is achieved. Fig 2.3 shows an overview of the algorithm.

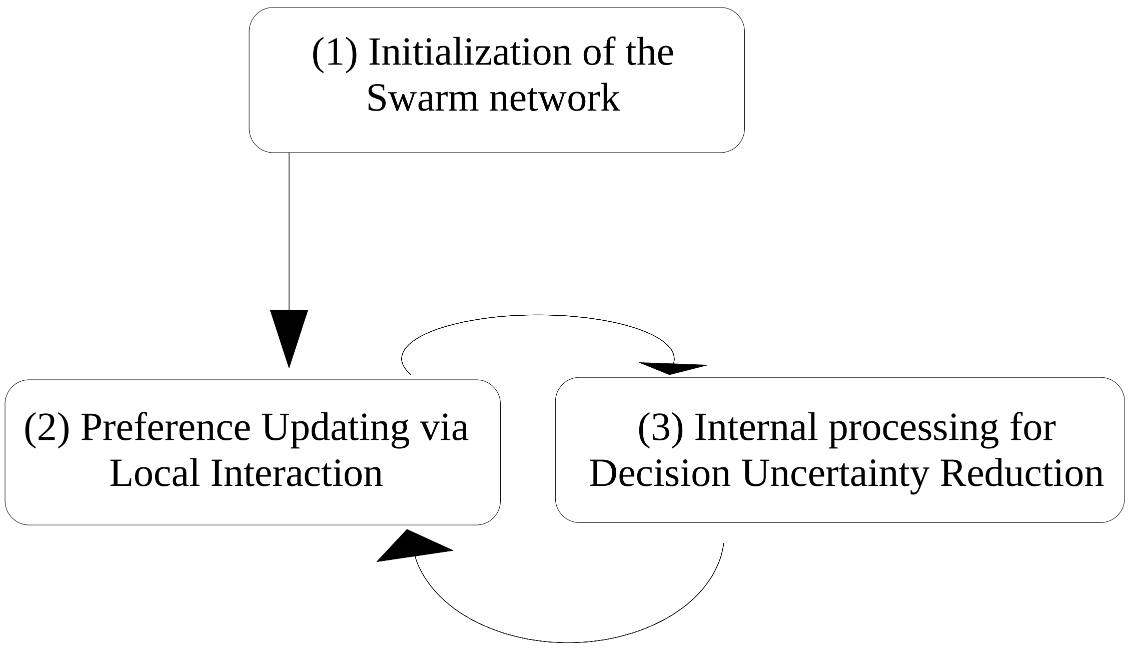

(1) Initialization of the Swarm network
(2) Preference Updating via Local Interaction
(3) Internal processing for Decision Uncertainty Reduction

**Fig 2.3: Flow chart of the presented collective decision making algorithm.**

## 2.3.3 Stage III

Once the decision is made now it time for role assignment. Here role to each robot will be assigned. Role assignment aims to achieve the opposite consensus of collective decision making. In many cases, each robot needs to execute more than one role at any given time, roles require more than one robots to be successfully executed[6].

A general algorithm of role assignment is presented, which applies to any topology of triangle grid networks, and uses message relay to communicate only minimum information. No history data will be record, only the latest states of other robots will be retained. The general role assignment algorithm consists of three parts: initial setup, gradient update and message relay, and resolving conflicts. On each robot in the swarm, the first part prepares the initial setup for the roll assignment. The latter two parts will run iteratively until all conflicts have been eliminated. Fig 2.4 shows an overview of the algorithm.

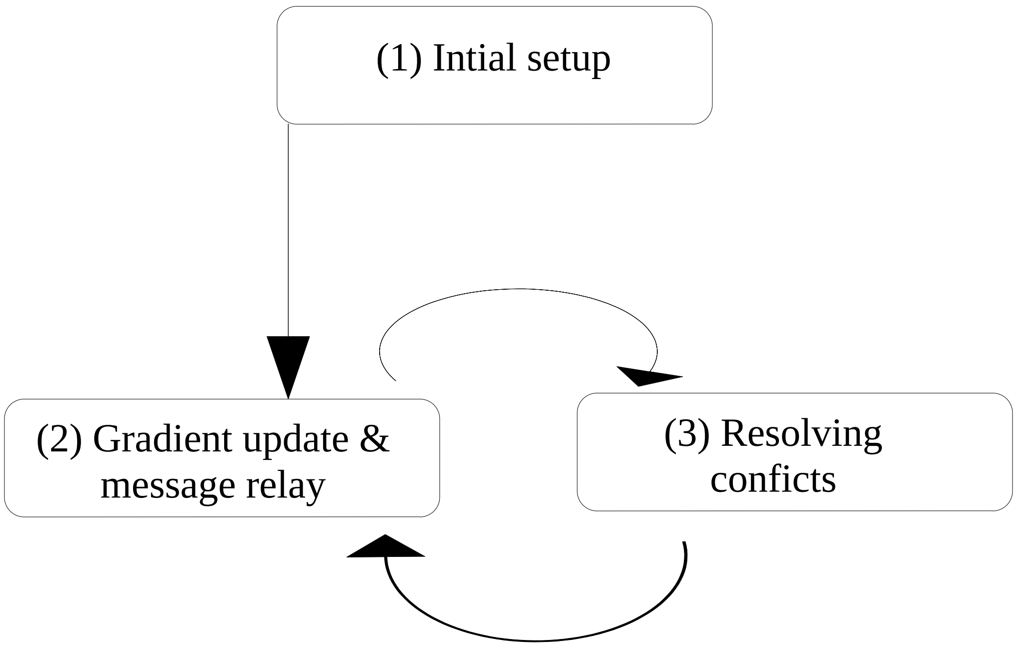


**Fig 2.4: Flow chart of the general role assignment algorithm.**

A gradient algorithm [7] has been used here to coordinate the message relay, making sure when a message is spreading out from a source to the rest network, it does not flow backward. Fig 2.5 shows the gradient values of the robots when the robot with an extra circle is the message source. When a message originated from that source is received at any other robot, it only relays the message to robots that has gradient value larger than itself. In a group of n robots, there will be n such gradient maps (with every robot being the message source for that map). Luckily in the presented algorithm, for each message source, individual robots only needs to find out which robots in its neighbors have larger gradient value than itself.

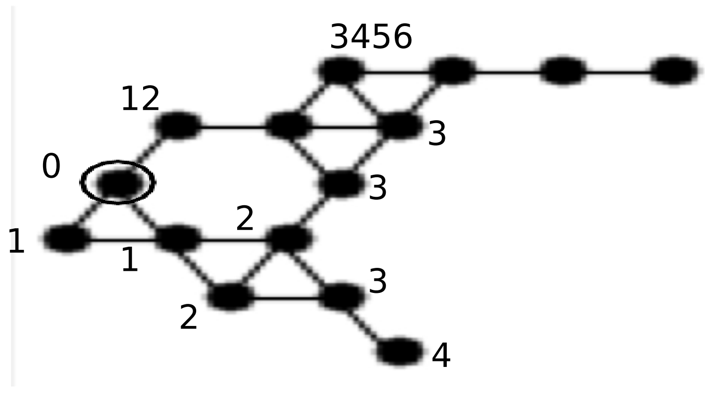


**Fig 2.5: Gradient map for the message source marked by an extra circle.**

## 2.3.4 Stage IV

This section shows how the algorithm is adapted to one-to-one role assignment on the loop. The robots start in a loop topology, with robots randomly placed on the loop, for example {k 0 , k 4 , k 3 , k 7 , k 2 , · · · }. The roles to be assigned are also in a loop topology, but ordered counter-clockwise on the loop, {r 0 , r 1 , r 3 , r 4 , r 5 , · · · }. The goal is to create a one-to-one mapping between the robots and roles, while maintaining the connections in both robot loop and target role loop. With these conditions, there are only n possible assignment schemes for a n-robot group.

The collective decision making algorithm can be applied in this special role assignment problem as follows: initially, each robot has its own preference distribution toward n target roles; by communicating with two neighbors, each robot updates its preference distribution; if certain conditions are satisfied, the robot reduces uncertainty of its distribution to help accelerate the convergence. There is only one adjustment for this algorithm. When updating the preference distributions, the neighbors' distributions must be shifted to match the corresponding roles as shown in fig 2.6.

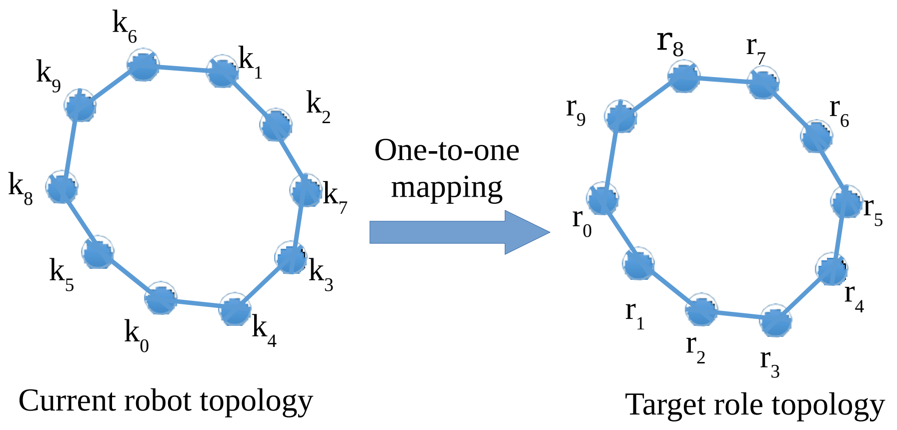


**Fig 2.6: One-to-one role assignment problem in the loop network.**

# Discussion of Results

In this chapter we discuss the results of the simulation of the proposed algorithm.

The proposed Algorithm is simulated in Pygame and Matplotlib using Python. Because the combination of Python, Pygame and Matplotlib help to verify the swarm algorithm quickly in a simpler setup, and the swarm robots is model as dots to skip the collision problems.

## 3.1 Simulations of Shape Formation

Three shape formation simulations are demonstrated below: (Simulation I) arbitrary triangle grid network formation, (simulation II) line formation, and (Simulation III) loop formation. The finite state machine is implemented with different additional behavior added to the in-group robots. For the simulations, the environment is defined by a 8 * 8 square for simulation I and II and 12 * 12 for line formation in simulation III. Robots have sensing and communication range of 0.65, and are trying to maintain distance of 0.5 in their formations. Robots move around at 0.05/s when in dormant or active states. Under these specifications, we found quick and stable formations can be achieved when $t_{min} = 2s$, $t_{max} = 6s$, $\delta t = 5s$.

**Simulations for Random Triangle Grid Network Formation:** In this simulation, the robots aggregate to form a connected network following a triangle grid pattern. There is no desired topology for the final formation, each robot will join the network at its convenience position and thus forming a random network. Addition behaviors are added to the in-group robots in order to form the triangle grid network, when an active robot finds a in-group robot and joins its group, it rotates counter-clockwise around that in-group robot until it finds another robot in the group, as illustrated in Fig 3.1

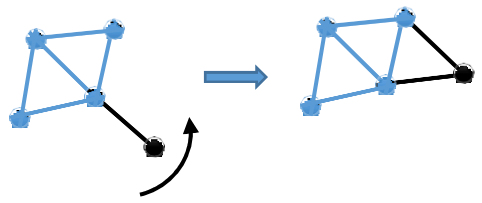


**Fig 3.1: Robot rotates around a in-group robot when first joining the group.**

At the beginning, all robots are in the dormant state (shown as empty dots) which are randomly dispersed in the environment. After the countdown timers end, dormant robots switch into the active state one after another, shown as solid dots that are not in a group. In-group robots are shown by solid dots in the groups. When one of them contains more than half of the swarm robots, it becomes the dominant group and its countdown timer is disabled. Black dots (i.e. robots) with block edges (i.e. connections), are used to indicate the dominant group (i.e. connections). Once the countdown timer ends in the other group, the robots disperse and gradually joins the dominant group. Finally, all robots form one connected network in the form of a triangle grid as shown in Fig 3.2.

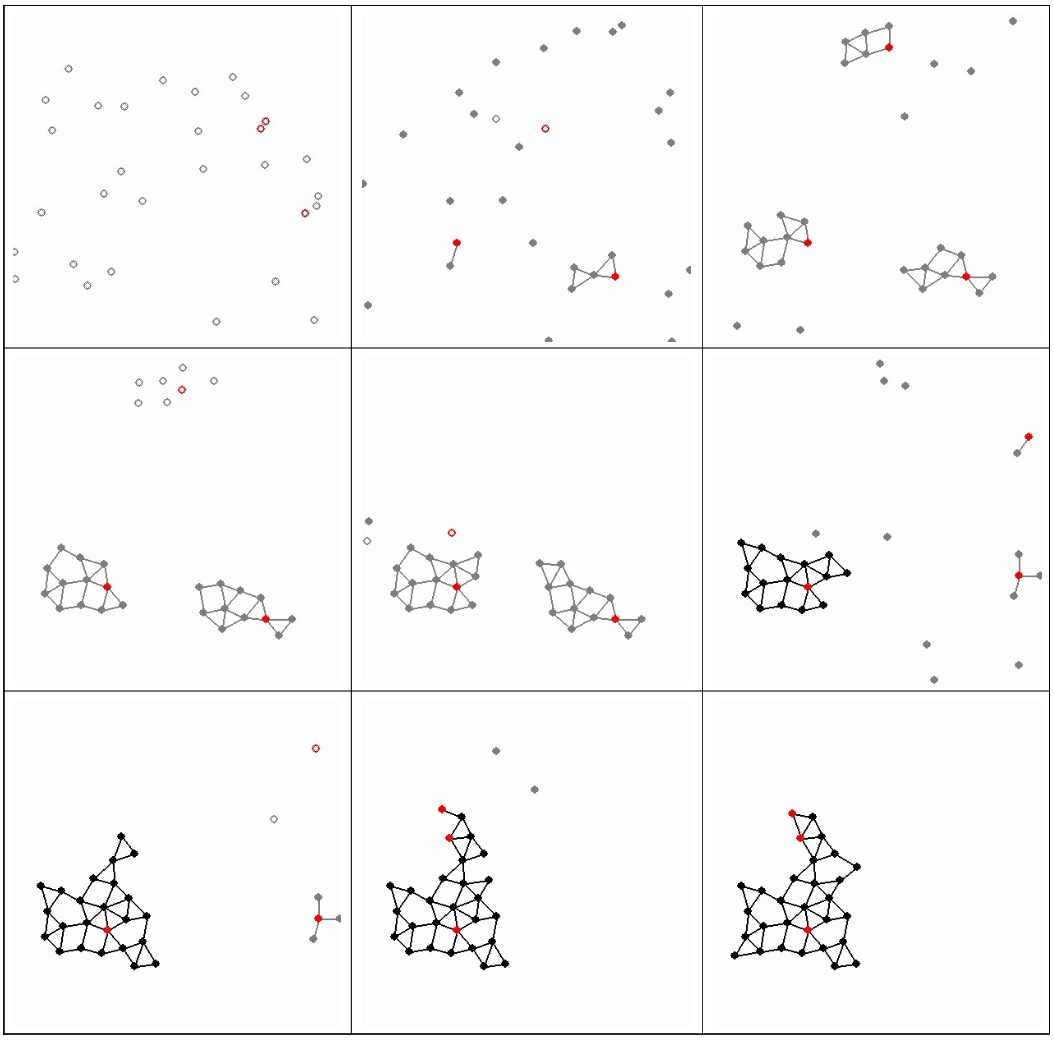


**Fig 3.2: Shape formation of a random triangle grid network with seed robot mechanism. Seed robots are shown in red.**

For a comparison another simulation is shown without seed robot mechanism in Fig 3.3. When in active state, the robots form new groups whenever they can if they don’t detect a group first. This results in many small groups being created around the environment, and decrease the chance of the emerging of the dominant group. Although simulation without seed robot mechanism eventually converges, it usually takes double or triple the time than with the seed robot mechanism.

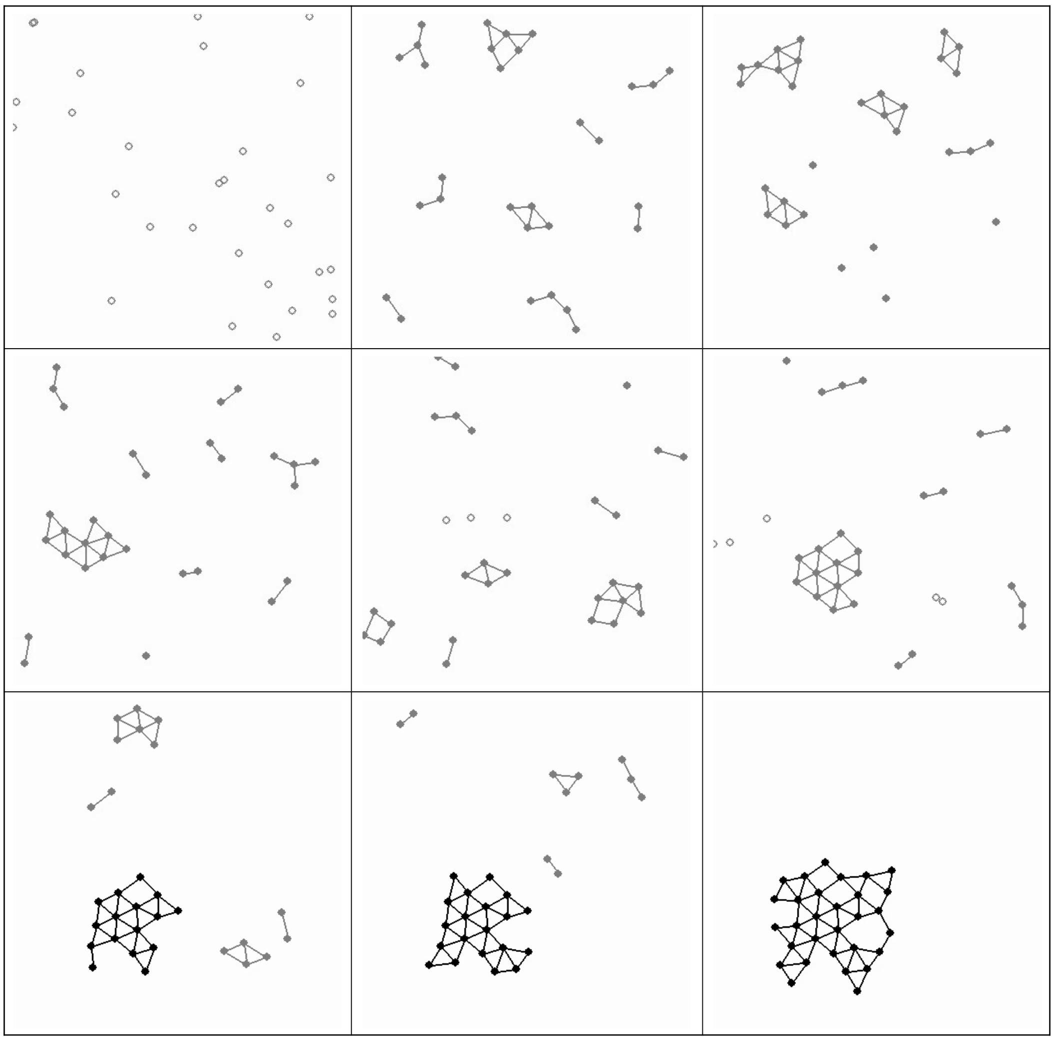


**Fig 3.3: Shape formation of a random triangle grid network without seed robot mechanism.**

**Simulations for Line Formation:** For line formation, the following behaviors are added to the in-group robots. When an active robot finds a in-group robot and joins the group, it rotates around the in-group robot until it finds another robot on the line the direction of rotation depends on which side is closer – and merges into the line between the two in-group robots. If the active robot merges to one end of the line, itself becomes the new end. The robots already on the line will adjust the spaces between each other, and straighten the line locally as shown in Fig 3.4.

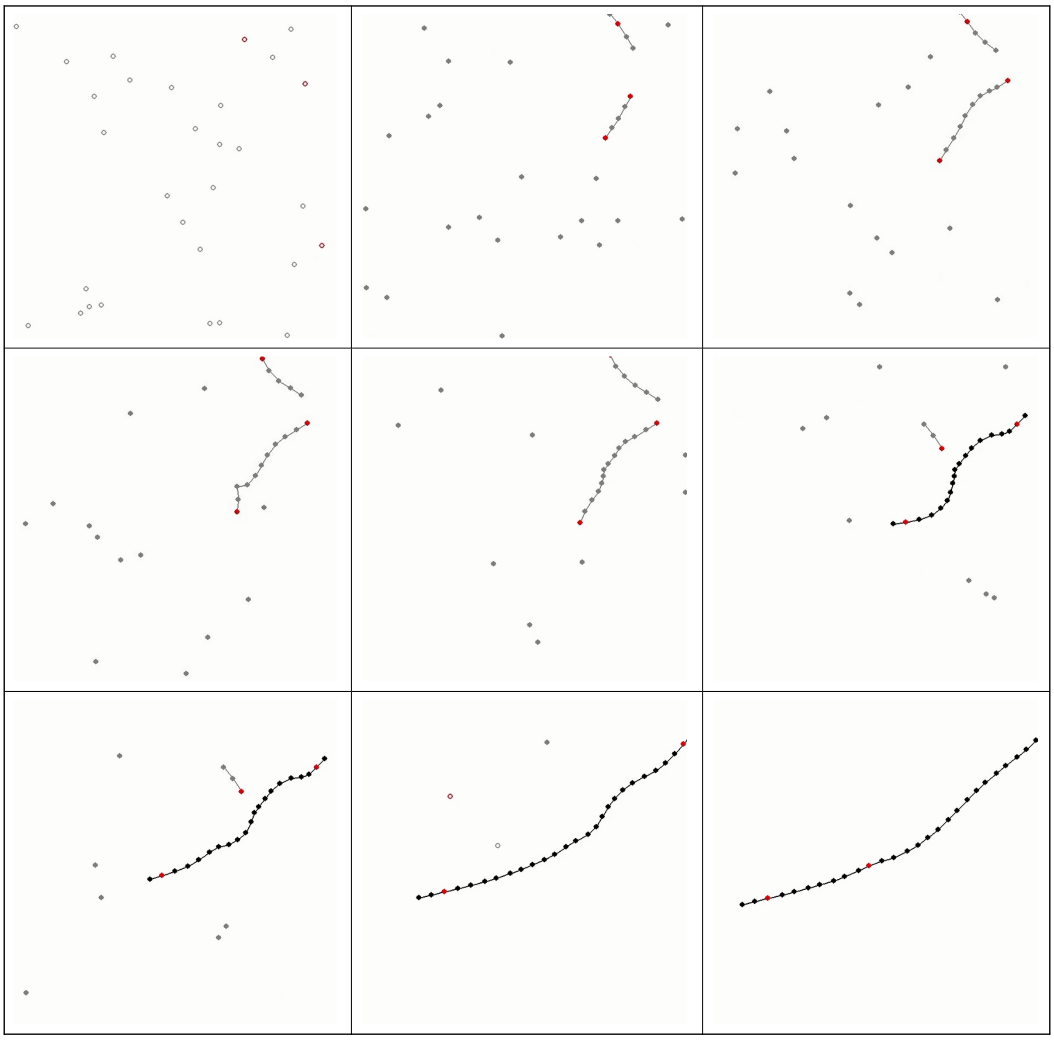


**Fig 3.4: Line formation with seed robot mechanism. Seed robots are shown in red.**

**Example of Loop Formation:** In loop formation, the behaviors added to in-group robots are similar to the line formation. When an active robot finds a in-group robot and joins the group, it rotates around the in-group robot until it finds another robot on the loop – the direction of rotation depends on which side is closer – and merges into the loop between the two in-group robots. If the group is initially established as a pair by two robots, the third member will try to form a triangle with the two, and the group will grow as a loop based on the triangle as shown in Fig 3.5.

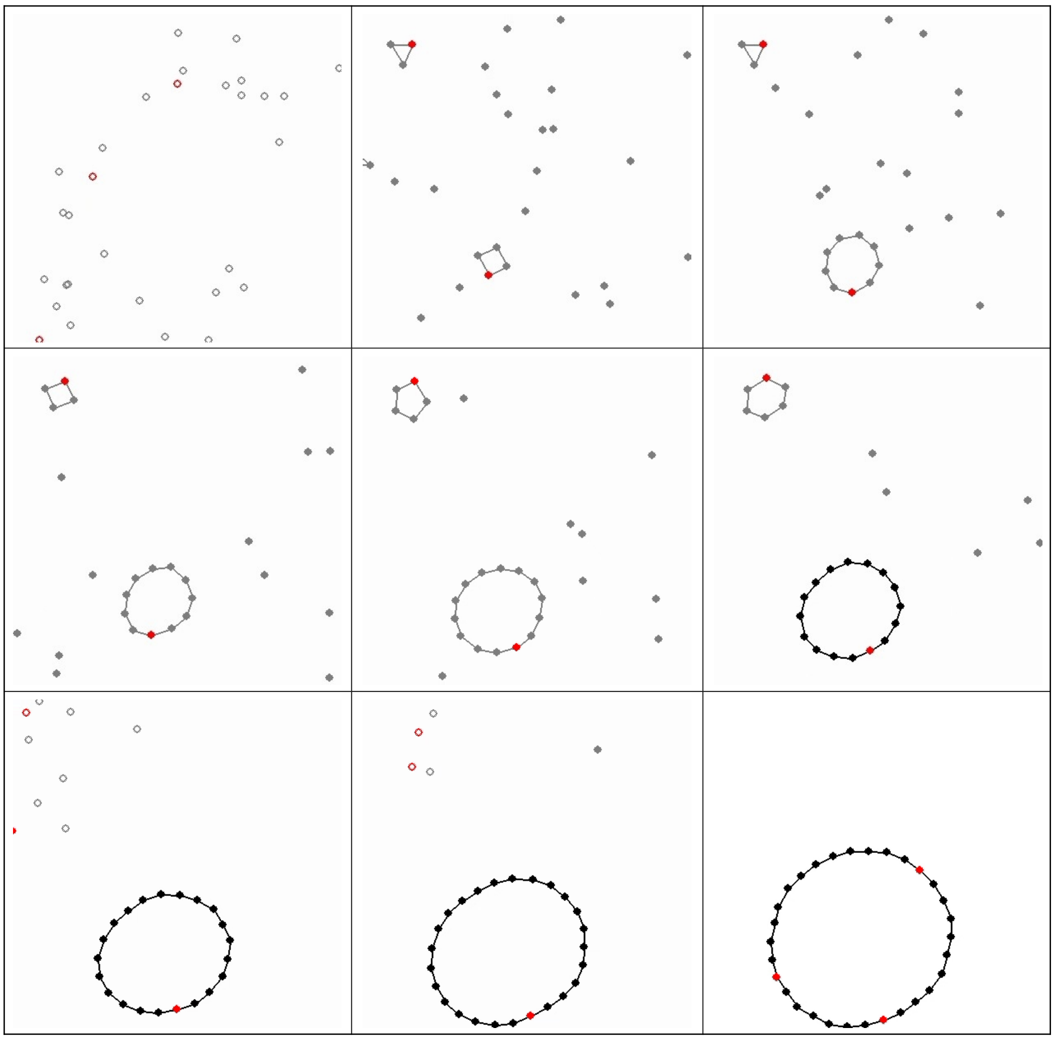


**Fig 3.5: Loop formation with seed robot mechanism. Seed robots are shown in red.**

# Chapter 5

## Conclusion:

This thesis presented an algorithm for collective shape formation in a swarm of primitive robots. It is assumes that each individual robot has limited sensing, communication, and computing power. According to this assumption, while a single robot may make any of n decisions, the swarm can only act in a coordinated manner if the majority of its members, if not all, agree on a single endpoint.

Aggregating the robots together to build simple or complicated structures is the main goal. The main focus is on collective shape formation control challenges. In the first shape formation method, the finite state machine guarantees to aggregate all robots into one connected network, and the process was accelerated with the help of the seed robot mechanism. With additional behavior added to the in-group robots, simple shapes, such as a line or a loop were achieved. The presented algorithm achieves guaranteed consensus in a connected network regardless of the number of decisions, network sizes, and topology, while the speed of convergence can be affected by these factors.

## 5.1 Challenges

This work can be also be simulated in gazebo. But in gazebo simulation time for 30 robots will be very more. So for verifying the algorithm in gazebo will be very time consuming. To do the simulation fast and smoothly in gazebo we need powerful processor and high graphics card in our system because gazebo has a capabilities to simulate any simulation in 3D also. That why we chosen pygame which help to verify the swarm algorithm quickly in a simpler setup, and model the swarm robots as dots to skip the collision problem.

## 5.2 Future Work

In this proposed thesis, we have done the entire simulation in non obstacles environment. Mean if there will be an obstacles all robots may hit to that obstacles. So it can be modified to work for an obstacle environment and to form more complex shapes with more number of robots. And also this whole work is simulated in Pygame, it can also be simulated in gazebo for better viewing experience because gazebo is a 3D simulator.

## Installation guide for Gazebo11, Ros2, Pygame2 :

**Prerequisite :**

1) Gazebo and ROS is now available for all platforms. But it is best used in Linux Ubuntu (for me its Ubuntu 20.04 LTS)
2) Check your OS version before installing any Gazebo and ROS distro. And also check your Gazebo version before installing ROS distro. (For me its Gazebo 11 and ROS2 Foxy).
3) Dedicated GPU for graphics display.
4) CPU (at least I5).
5) It is recommended to have at least 500 MB of free disk space.

**Installation of Gazebo11 :-**

**https://classic.gazebosim.org/tutorials?tut=install_ubuntu**

**Installation of ROS2 :-**

**https://docs.ros.org/en/foxy/Installation/Ubuntu-Install-Debians.html**

**Installation Troubleshooting for ROS2 :-**

**https://docs.ros.org/en/foxy/How-To-Guides/Installation-Troubleshooting.html**

**Installation of gazebo_ros_pkgs (ROS 2) :-**

**https://classic.gazebosim.org/tutorials?tut=ros2_installing&cat=connect_ros**

**Installation of Pygame :-**

**https://installati.one/ubuntu/20.04/python-pygame/**

# Appendix B

**How to create a wheel robot in gazebo:**

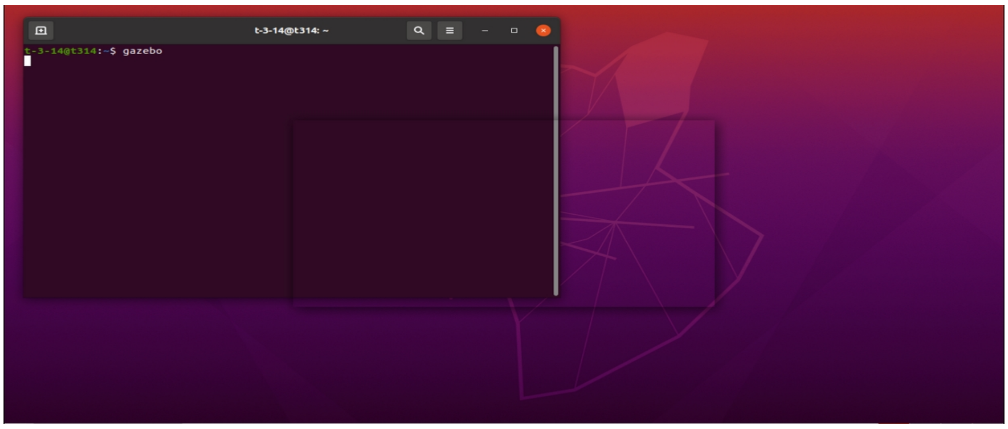
Open gazebo from terminal. But before opening it we have source to library of gazebo by running the following command:-

```
source /usr/share/gazebo/setup.sh
```

Now simply open gazebo by running the following command:-

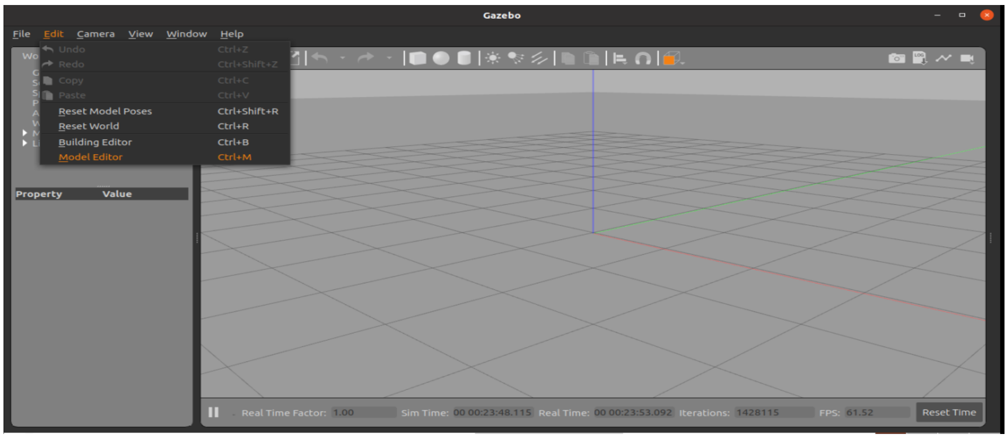
```
gazebo
```

If everything will be okay, then you will see a square box will open at middle of the screen as shown below. It may take some times to open the application. Or its depends on your system configuration.

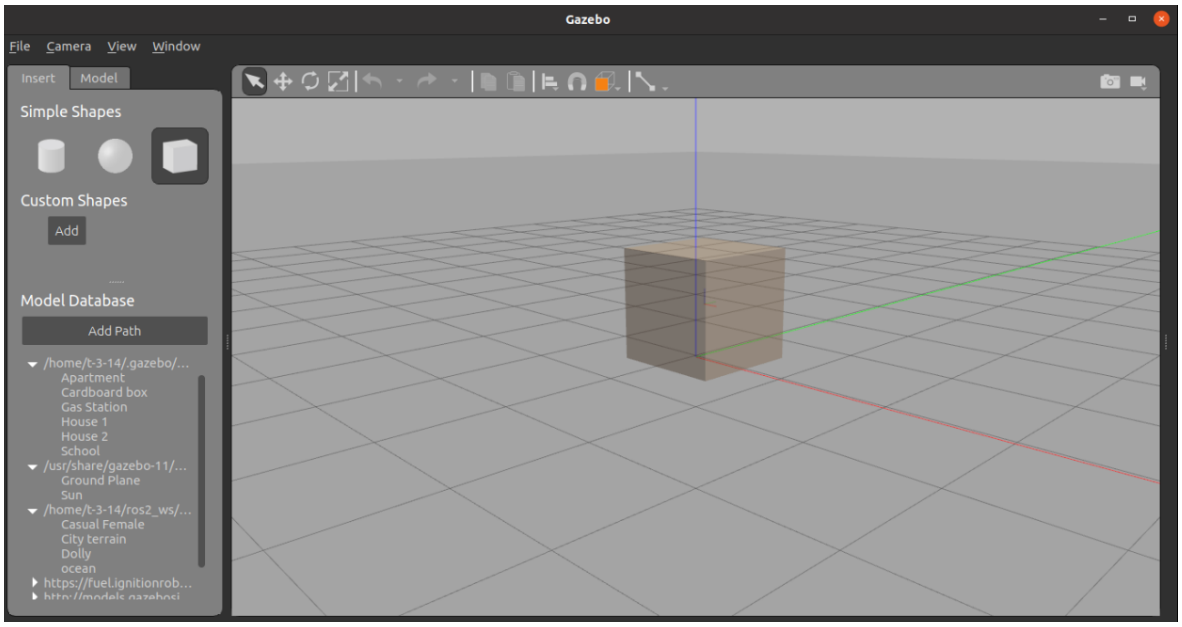


```
t-3-14@t314:~$ gazebo
```

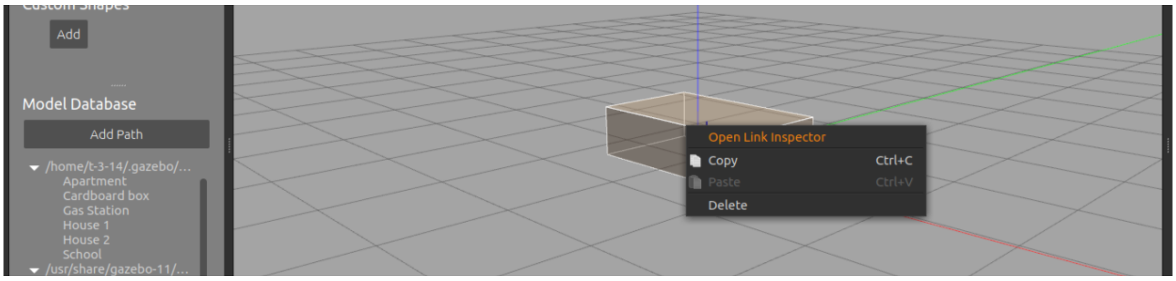
Now go to top left corner and click on "**Edit**" menu and then click on "**Model Editor**".



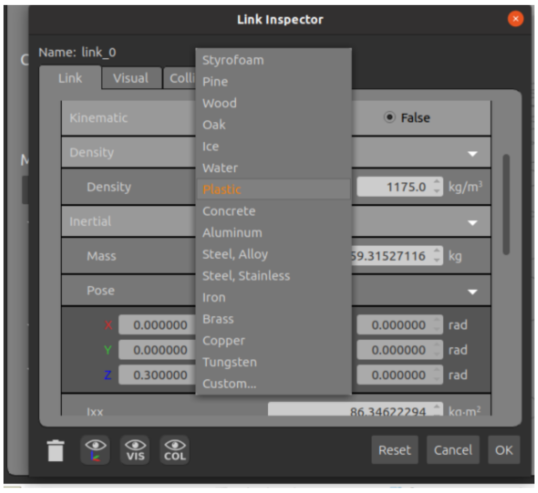
Now click on box shape from simple shape and place it on center of the map.

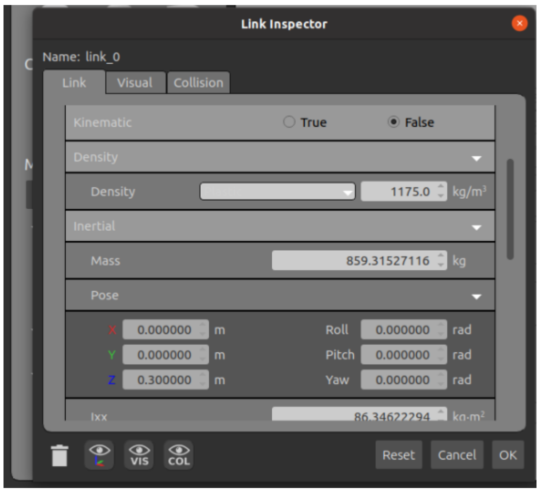


Now we will give a shape to our robot main body. For that right click on the box and click on "**Open Link inspector**".



Now in link tab, change the density to "**plastic**" and change the z value to "**0.3**" of pose.





Name: link_0

Link | Visual | Collision

Kinematic: True [✓ False]

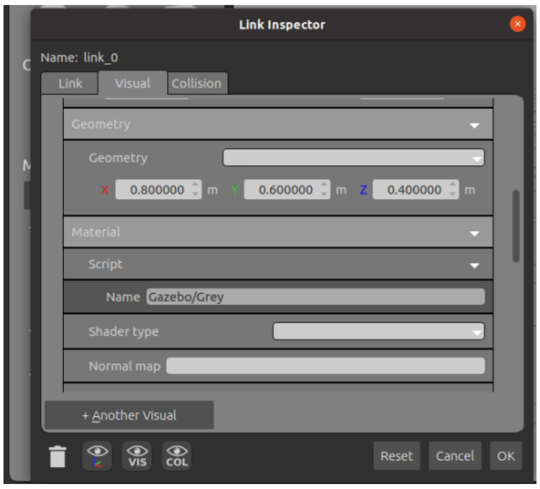
Density: 1175.0 kg/m³

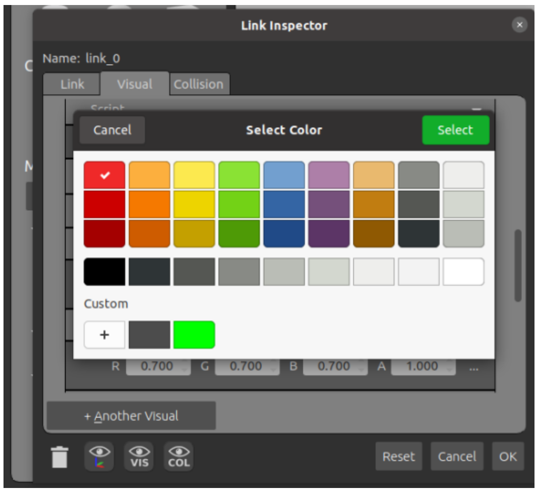
Mass: 859.31527116 kg

Pose
x 0.000000 m Roll 0.000000 rad
y 0.000000 m Pitch 0.000000 rad
z 0.300000 m Yaw 0.000000 rad

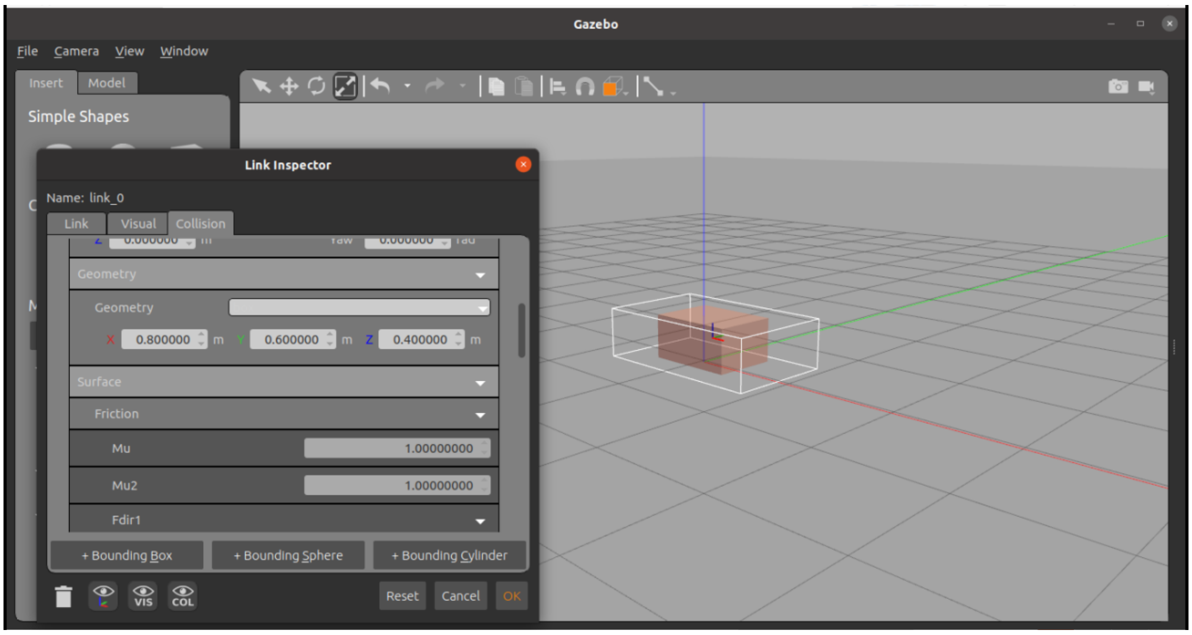
Ixx 86.34622294 kg·m²

Now click on visual tab, and go to geometry section and change the **x**, **y**, **z** value of it and also changed the “**ambient**” to give color to the robot.



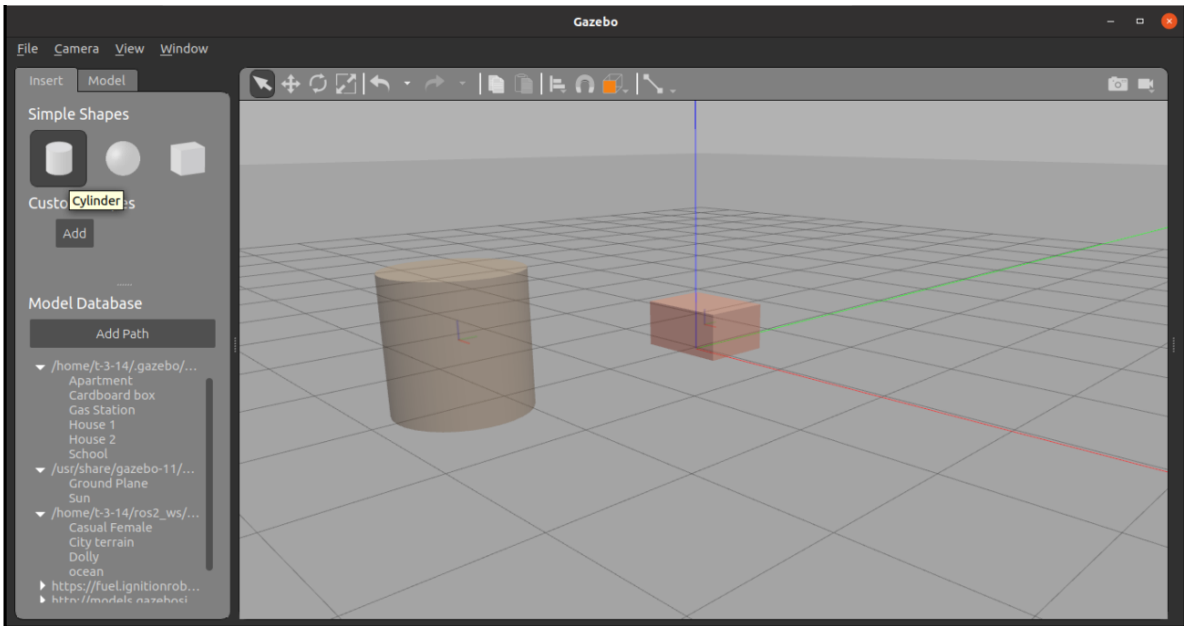


Now lastly click on collision tab, and again go to geometry section and change the **x**,**y**,**z** value of it and click on “**OK**” button in last right corner of it.

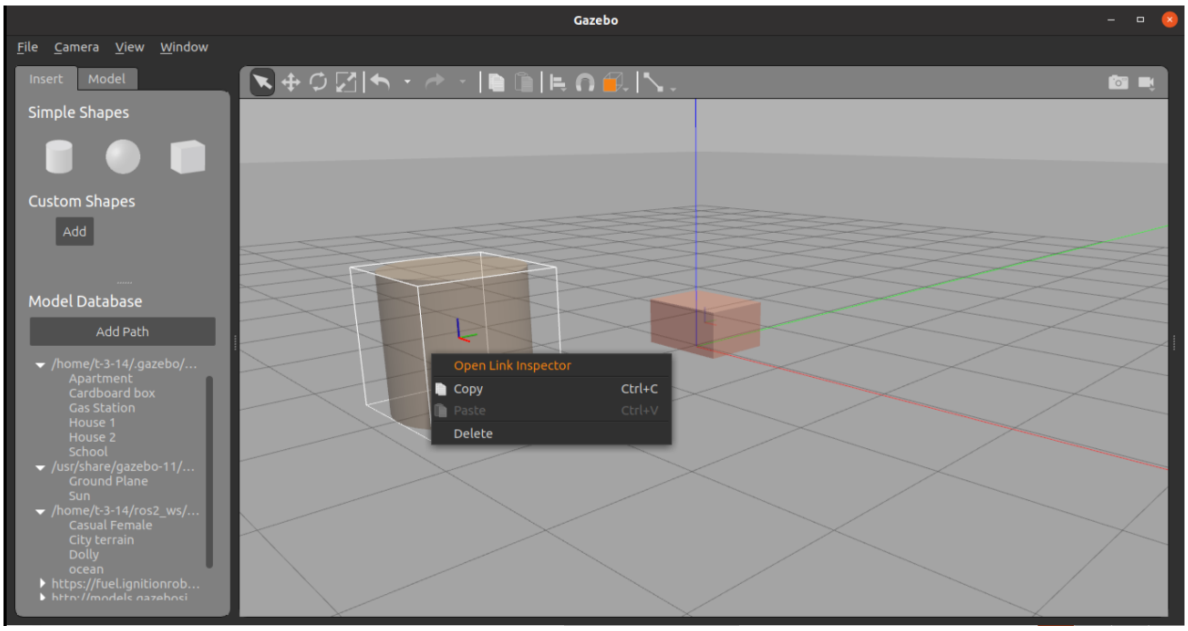


As you can also see, while changing the values of the above mentioned sections box shape is also changing. Through this you can know which shape you want to give to your robot. This is not a definite values which i have mentioned above. You can give any shape to your robots by changing its values.

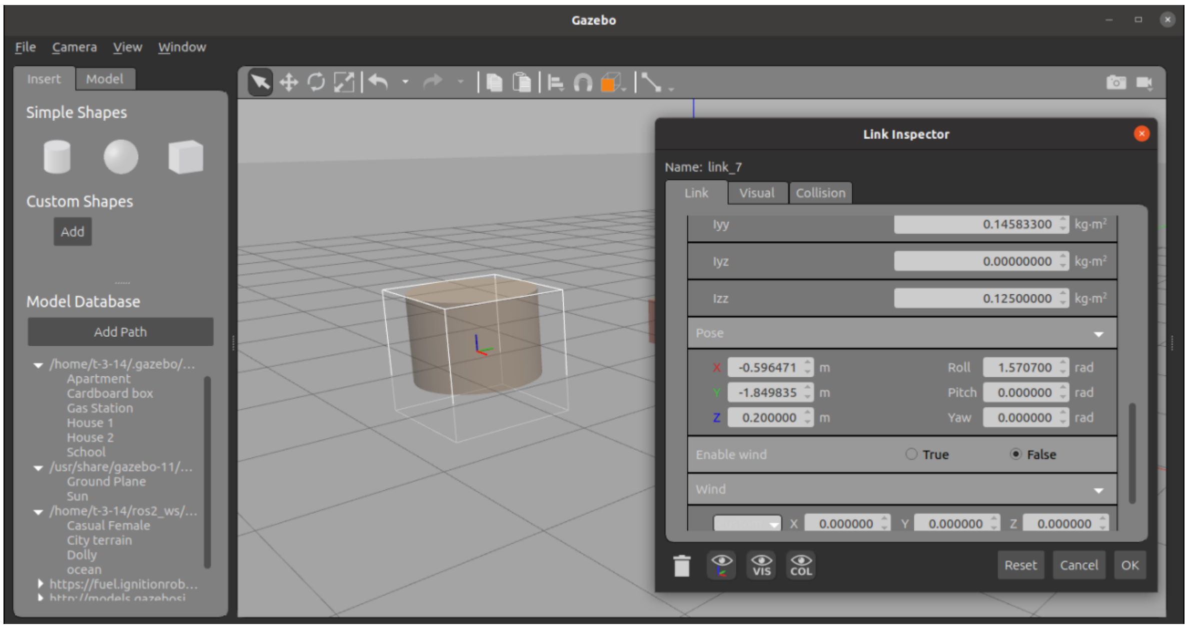
Now we have to construct wheel of our robots. For that click on cylinder figure in simple shape and place it on one side of the robot.



Now we have to give a wheel shape to this cylinder. So right click on cylinder and click on "**Open Link Inspector**".



Now in link tab, go to Pose section and change the value of "**Roll**" and "**z**" in it.



Name: link_7

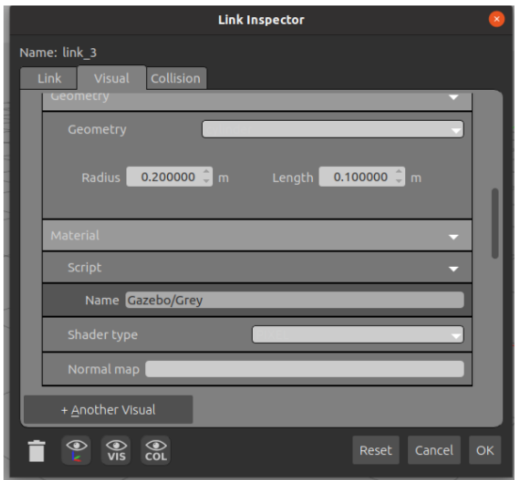
Iyy 0.14583300 kg·m²
Iyz 0.00000000 kg·m²
Izz 0.12500000 kg·m²

Pose
x -0.596471 m Roll 1.570700 rad
y -1.849835 m Pitch 0.000000 rad
z 0.200000 m Yaw 0.000000 rad

Enable wind True [✓ False]

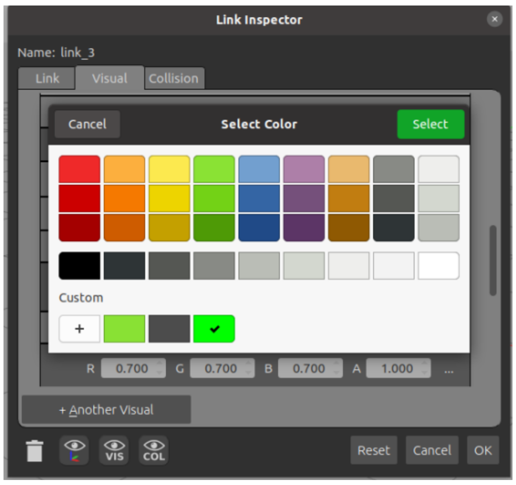
Wind
X 0.000000 Y 0.000000 Z 0.000000

Now click on visual tab, and go to geometry section and change the value of "**Radius**" and "**Length**" and also changed the "**ambient**" to give color to the wheel.



Name: link_3

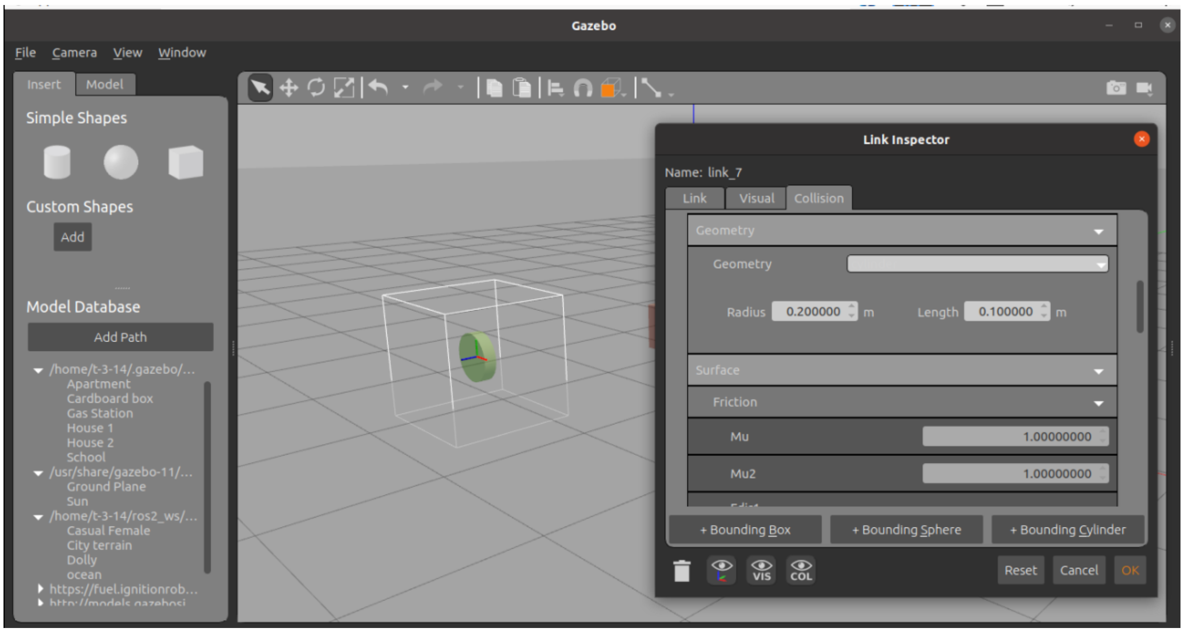
Geometry
Radius 0.200000 m Length 0.100000 m
Material
Script
Name Gazebo/Grey
Shader type
Normal map



Name: link_3

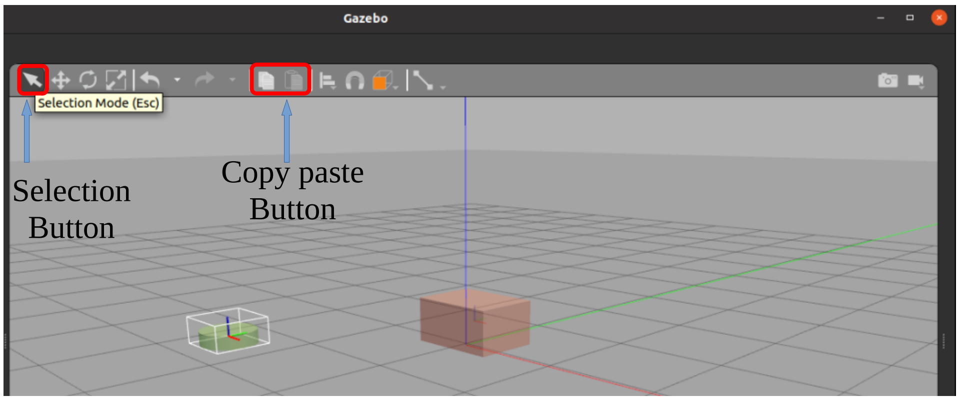
R 0.700 G 0.700 B 0.700 A 1.000

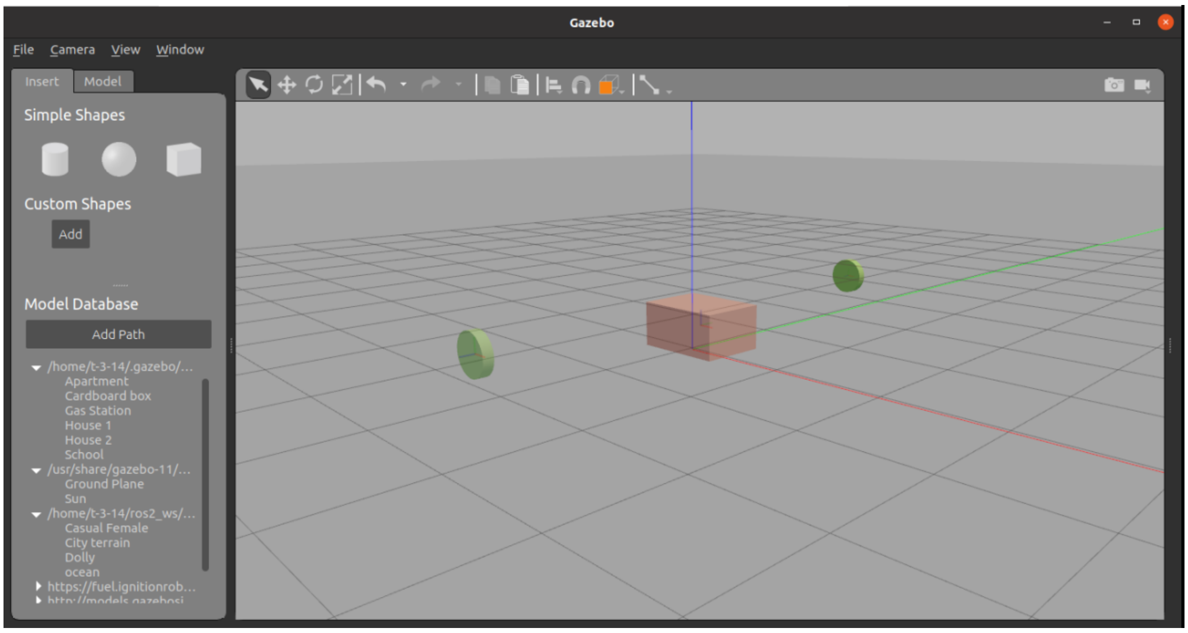
Now lastly click on collision tab, and again go to geometry section and change the value of "**Radius**" and "**Length**" and click on "**OK**" button in last right corner of it.



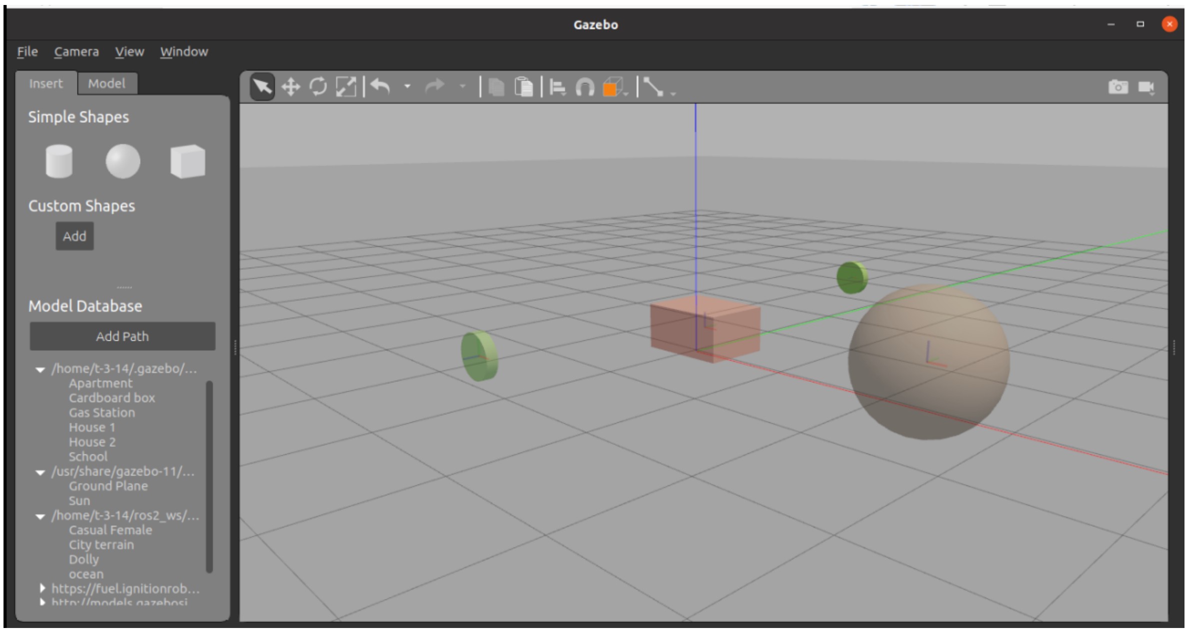
As you can see, while changing the values of the above mentioned sections cylinder shape is also changing. Through this you can know which shape you want to give to your wheels. This is not a definite values which i have mentioned above. You can give any shape to your wheels by changing its values.

After giving the shape to our first wheel now just make a copy of that wheel and paste it on other side of the robot. To do that, gazebo have an option of copy paste. Simply select the object which you want to copy and click on copy button in above menu bar. Once the object is copied your paste button will automatically be enable now click on paste button and placed where ever you want.

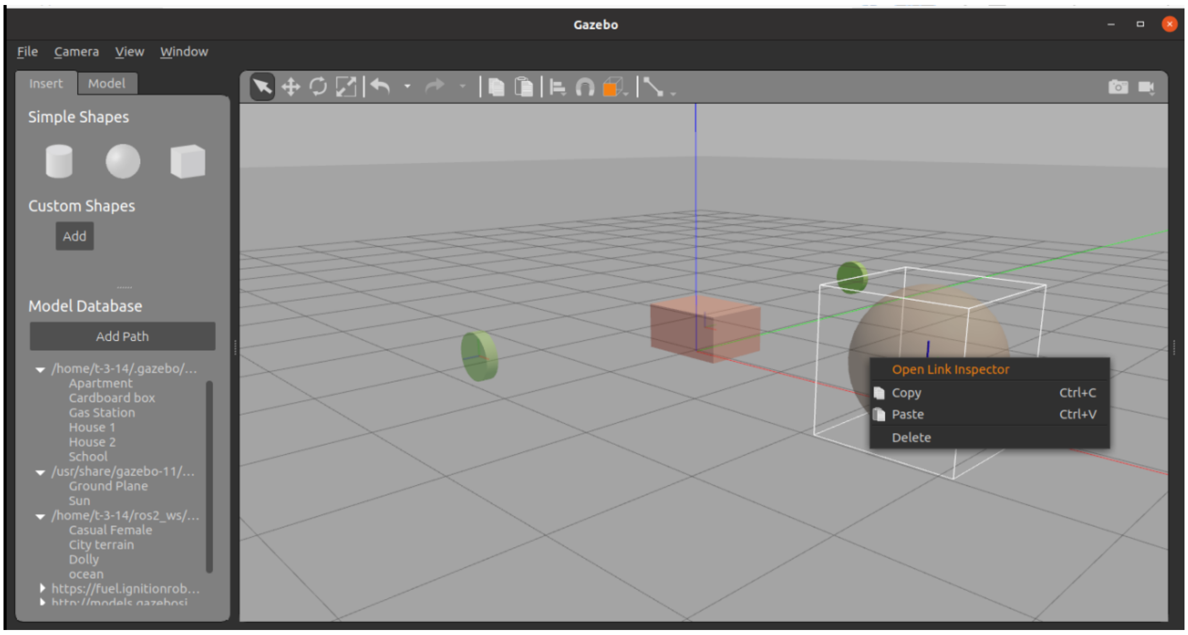




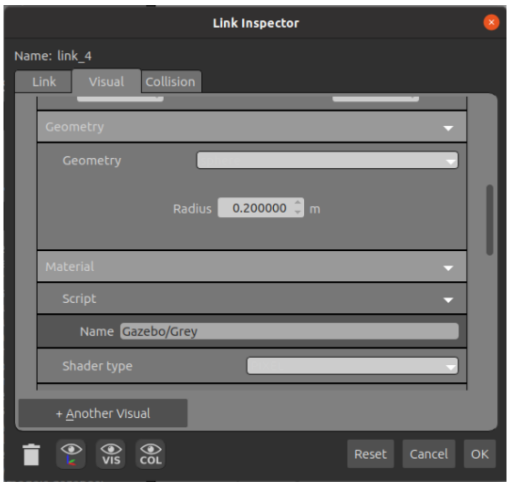
Now we will construct the front wheel for our robot. For front wheel, you can also add the two wheel which we have made before. But I am using only one sphere wheel for front part. For that click on sphere shape and place it on front of the robot.

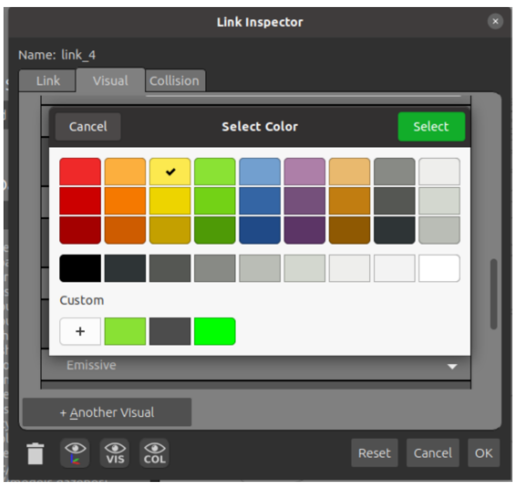


Now we have to give a wheel shape to this sphere. So right click on sphere and click on “**Open Link Inspector**”

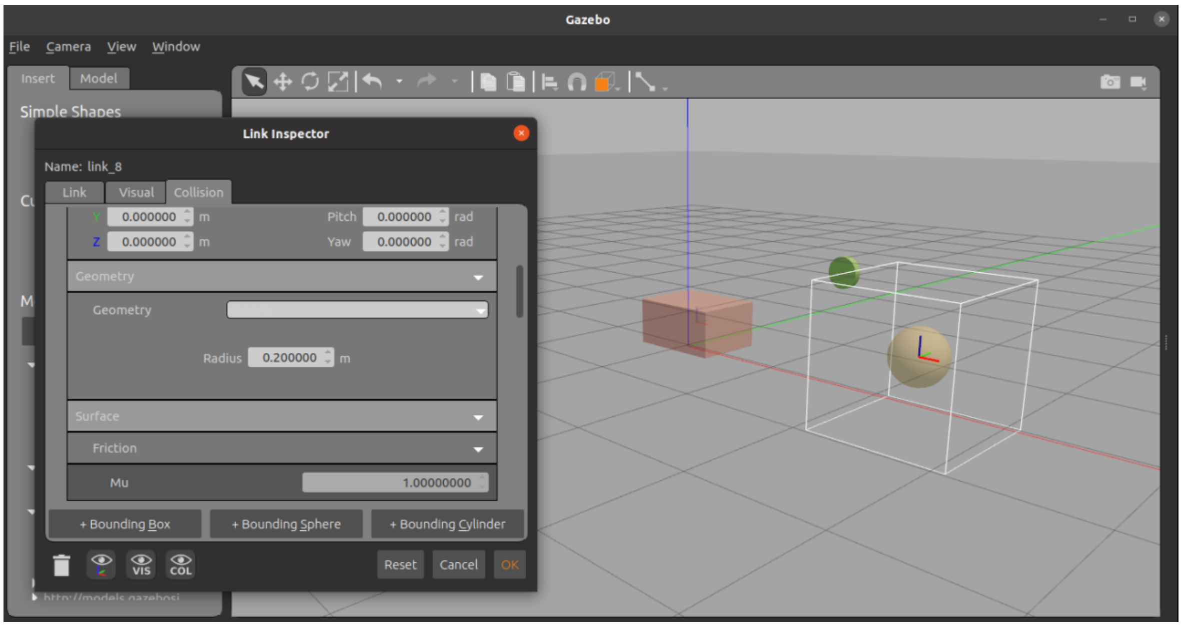


Now click on visual tab, and go to geometry section and change the value of "**Radius**" and also changed the "**ambient**" to give color to the wheel.



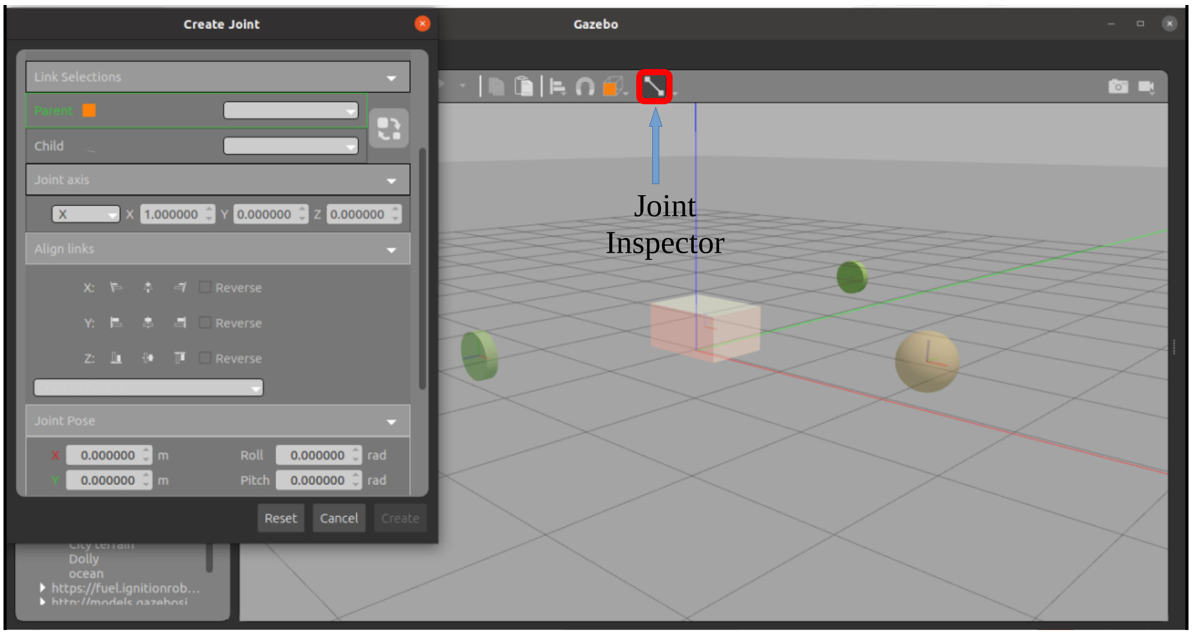


Now lastly click on collision tab, and again go to geometry section and change the value of "**Radius**" and click on "**OK**" button in last right corner of it.

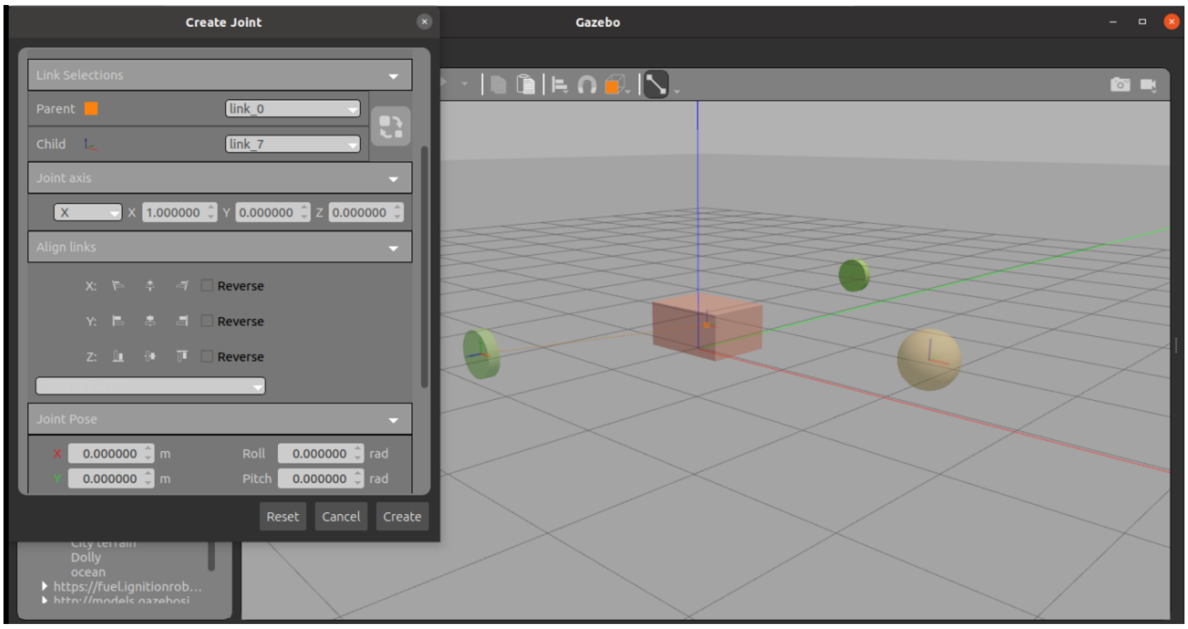


As you can see, while changing the values of the above mentioned sections sphere shape is also changing. Through this you can know which shape you want to give to your wheel. This is not a definite values which i have mentioned above. You can give any shape to your wheel by changing its values.

Now its time to attach those wheels to the main body of the robot. For that we have to use joint inspector.

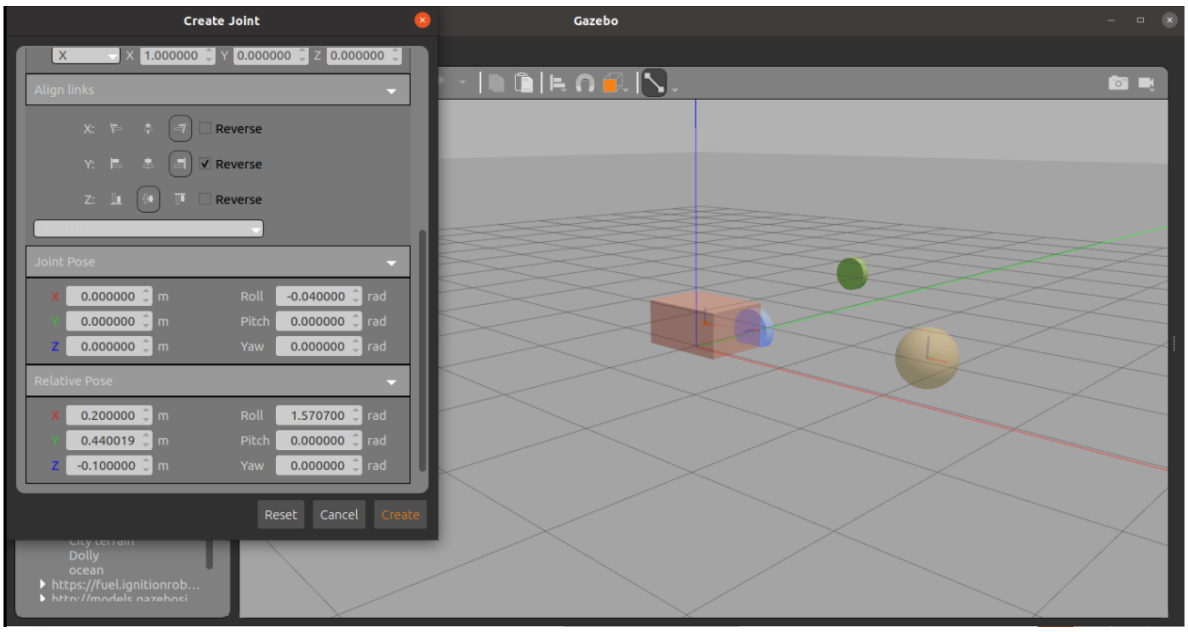


After clicking to joint inspector we have to select the parent and child for that joint. As you can see in the above picture. Every time we have to select main body of the robot as a parent. And child will be the respective three wheels.



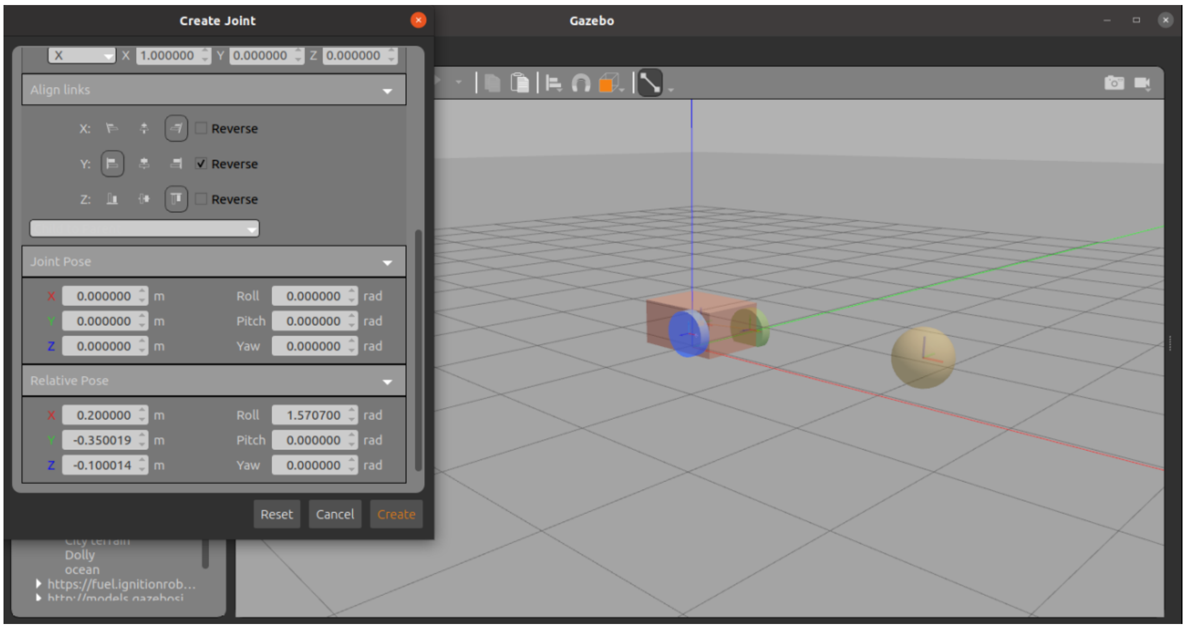
Parent link_0
Child link_7
Joint axis: X; X 1.000000 Y 0.000000 Z 0.000000
Align links: X: Reverse; Y: Reverse; Z: Reverse
Joint Pose: X 0.000000 m, Roll 0.000000 rad; Y 0.000000 m, Pitch 0.000000 rad

Now align the link between box and wheel. And change the value of “**z**” in Relative Pose section.



Joint axis: X; X 1.000000 Y 0.000000 Z 0.000000
Align links: X: Reverse; Y: ✓ Reverse; Z: Reverse
Joint Pose: X 0.000000 m, Roll -0.040000 rad; Y 0.000000 m, Pitch 0.000000 rad; Z 0.000000 m, Yaw 0.000000 rad
Relative Pose: X 0.200000 m, Roll 1.570700 rad; Y 0.440019 m, Pitch 0.000000 rad; Z -0.100000 m, Yaw 0.000000 rad

Now do the same for left wheel.


Create Joint

X X 1.000000 Y 0.000000 Z 0.000000

Align links

X: Reverse

Y: [✓ Reverse]

Z: Reverse

Joint Pose

X 0.000000 m Roll 0.000000 rad

Y 0.000000 m Pitch 0.000000 rad

Z 0.000000 m Yaw 0.000000 rad

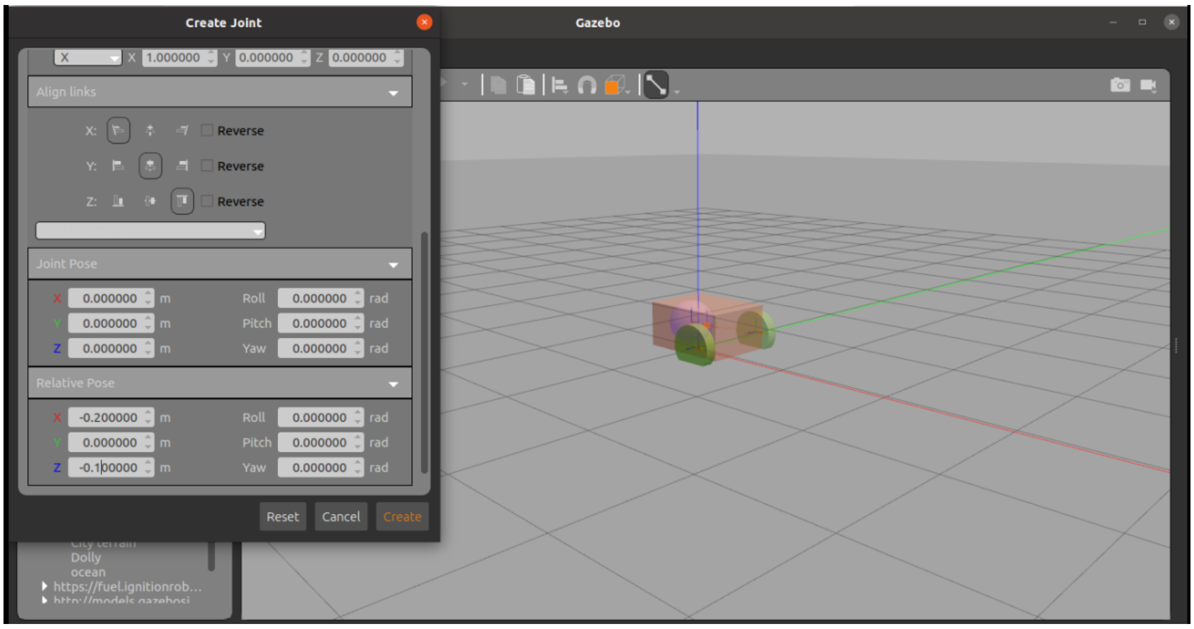
Relative Pose

X 0.200000 m Roll 1.570700 rad

Y -0.350019 m Pitch 0.000000 rad

Z -0.100014 m Yaw 0.000000 rad

Now also do the same for front wheel (i.e. ball wheel).


Create Joint

X X 1.000000 Y 0.000000 Z 0.000000

Align links

X: Reverse

Y: Reverse

Z: Reverse

Joint Pose

X 0.000000 m Roll 0.000000 rad

Y 0.000000 m Pitch 0.000000 rad

Z 0.000000 m Yaw 0.000000 rad

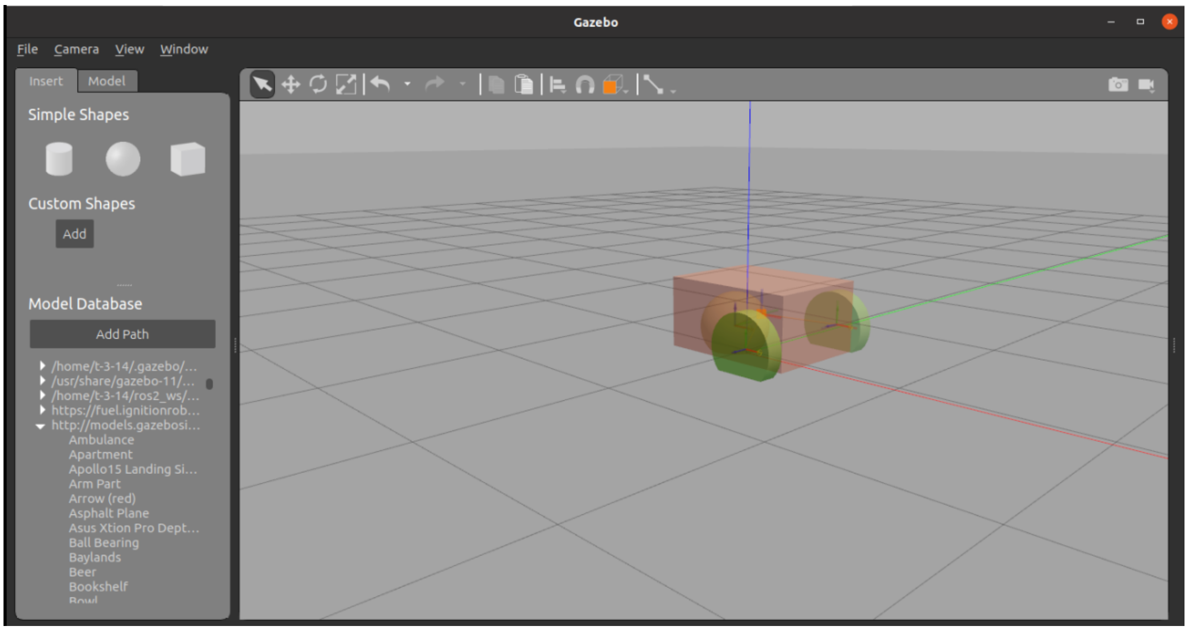
Relative Pose

X -0.200000 m Roll 0.000000 rad

Y 0.000000 m Pitch 0.000000 rad

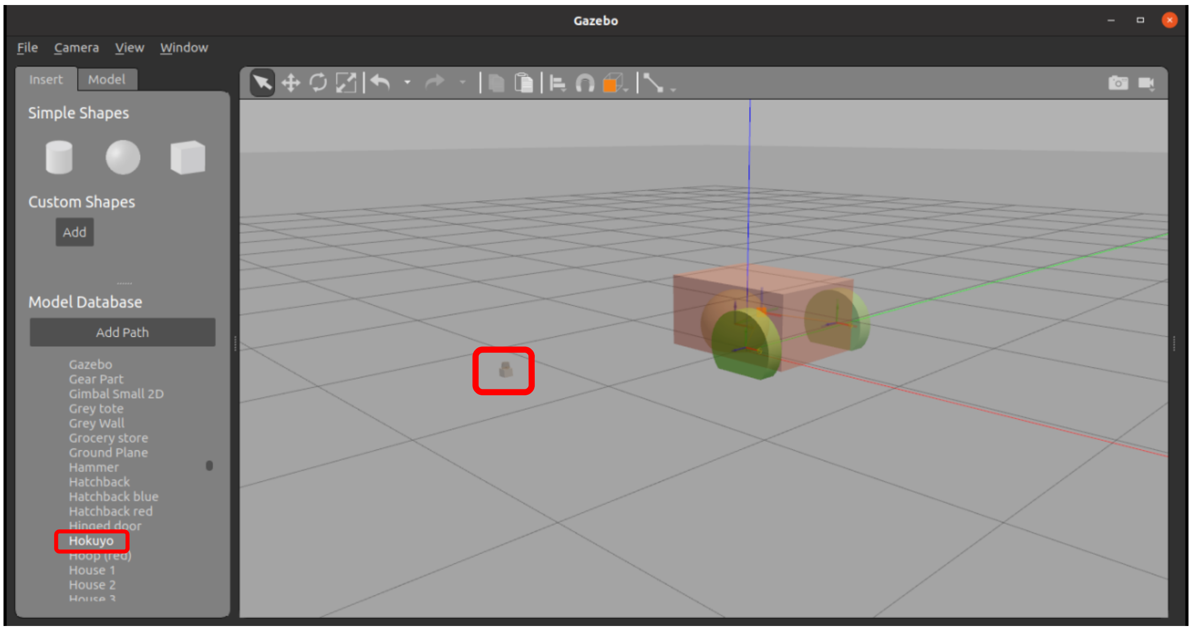
Z -0.100000 m Yaw 0.000000 rad

Now we have completely build our wheeled robot. The final looks of our model look like:-

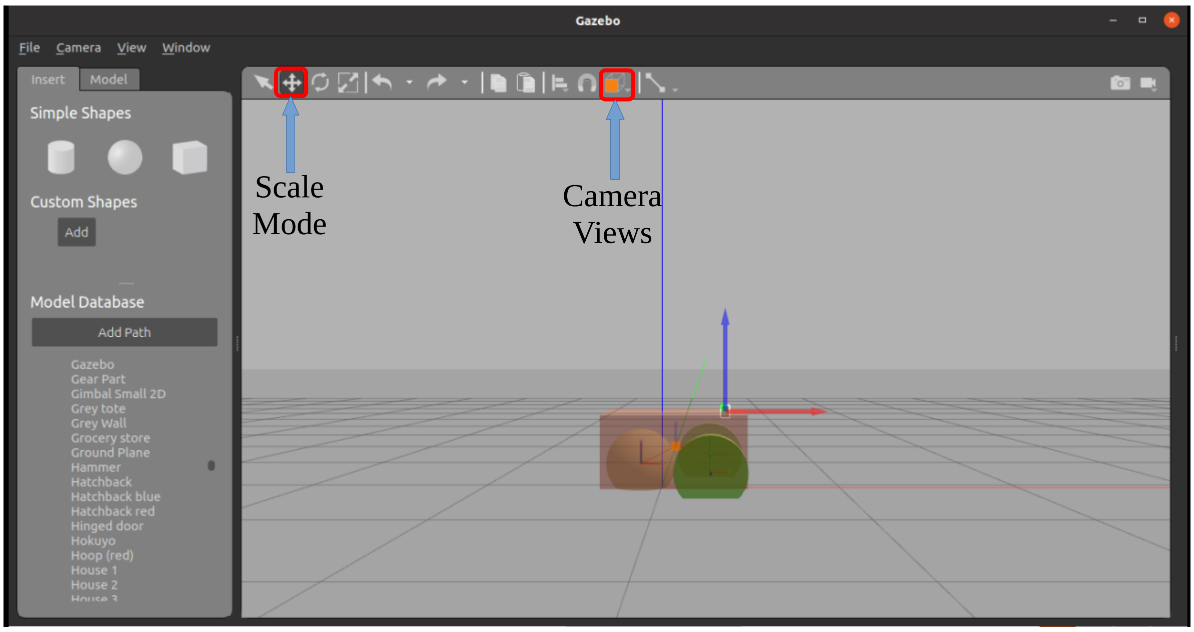


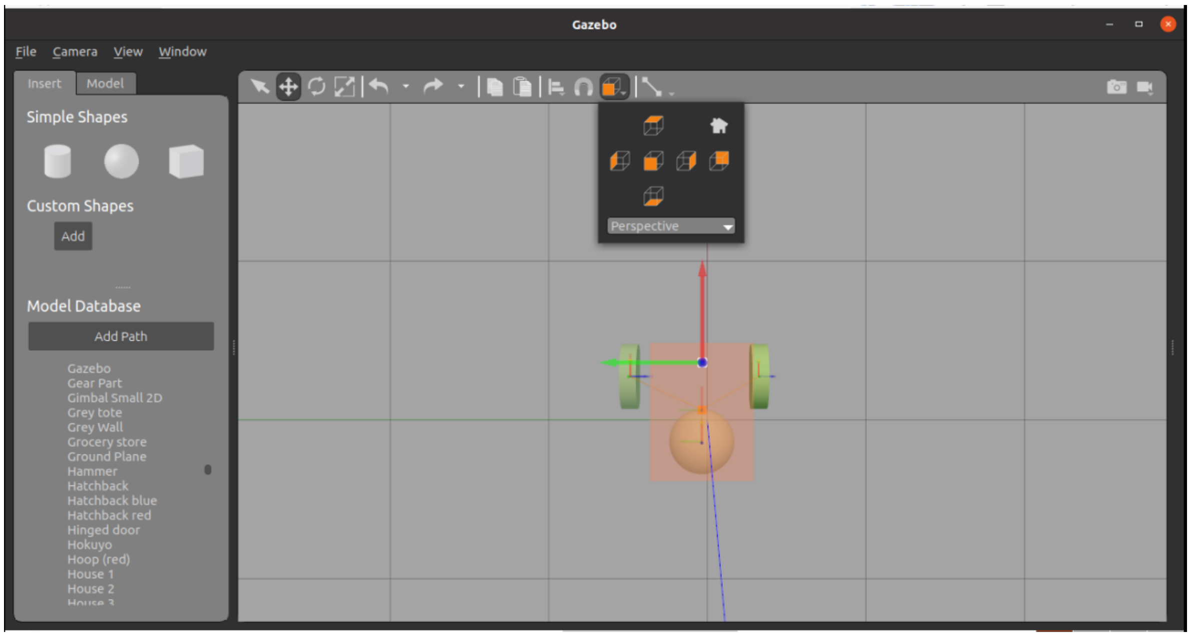
Now lets add one sensor to this model. For that you need to know that Gazebo relies on a database to store and maintain models available for use within simulation. The model database is a community-supported resource, so People upload and maintain models that you create and use. Gazebo is able to dynamically load models into simulation either programmatically or through the GUI. Models exist on your computer, after they have been downloaded or created by you. Models in Gazebo define a physical entity with dynamic, kinematic, and visual properties. In addition, a model may have one or more plugins, which affect the model's behavior. A model can represent anything from a simple shape to a complex robot; even the ground is a model.

There are many inbuilt sensor available in gazebo database. For example i am using Hokuyo sensor in this robot. Hokuyo is a Laser Range Finder (LRF) sensor. To add it first click on the models database of gazebo. One model chart will open. Now find Hokuyo sensor in that database, then click on Hokuyo and place it on the map.

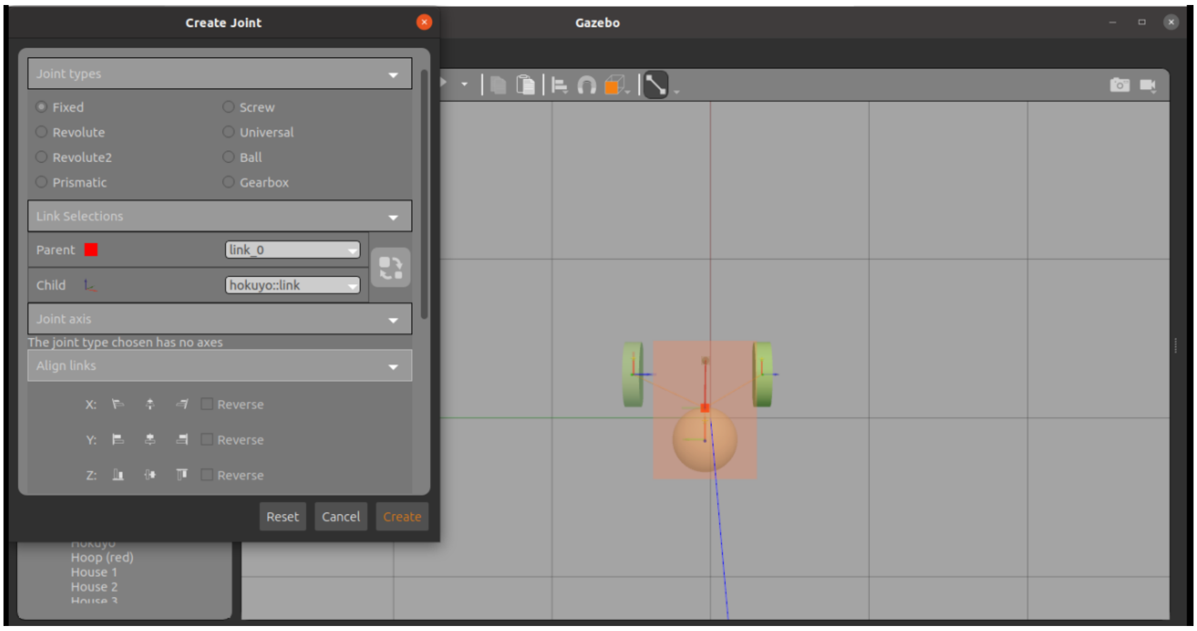


Now move the Hokuyo sensor through Scale mode and placed it on top of the robot. Check the alignment through camera for exact position.





After placing the sensor exactly to the top of the models now next step is to joint the sensor to the robot. For that open joint inspector and select robot body as parent and sensor as child and make joint type to fixed and click on create button.


Create Joint

Joint types

[✓ Fixed] Screw
Revolute Universal
Revolute2 Ball
Prismatic Gearbox

Link Selections

Parent: link_0
Child: hokuyo::link

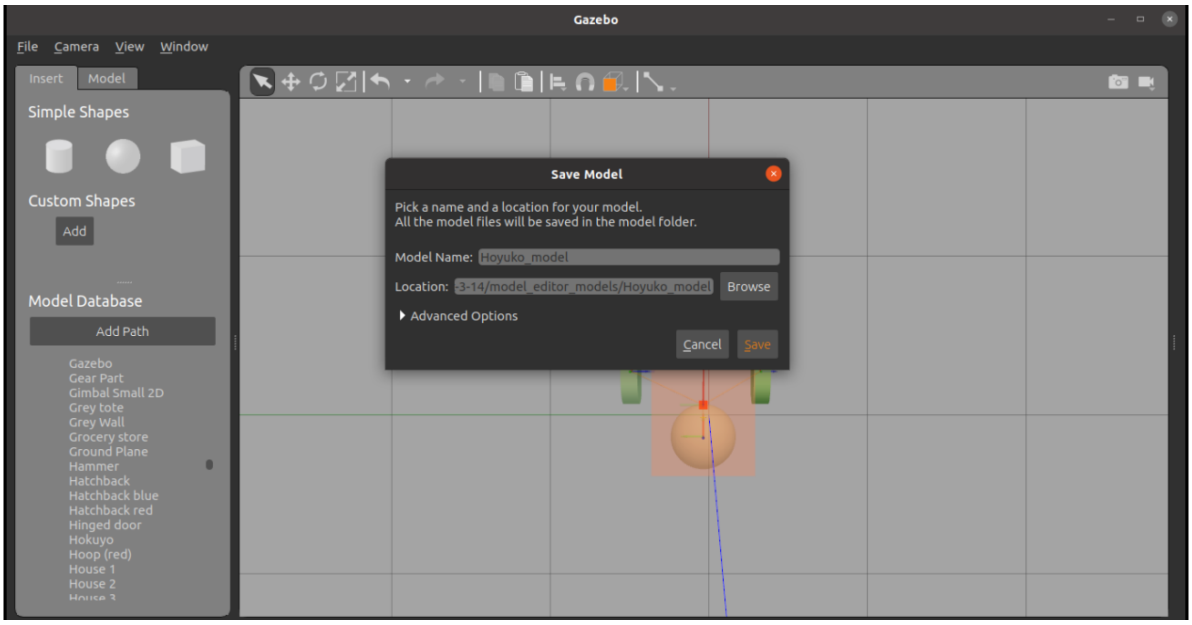
Joint axis

The joint type chosen has no axes

Align links

X: Reverse
Y: Reverse
Z: Reverse

Now our model is fully build with one sensor and ready to move in the environment. So simply save the model and exit model editor.



Save Model

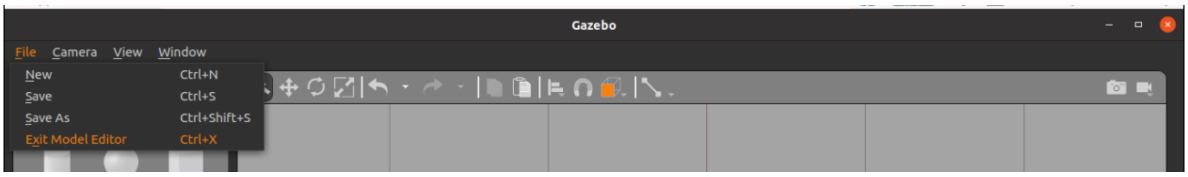
Pick a name and a location for your model.
All the model files will be saved in the model folder.

Model Name: Hoyuko_model

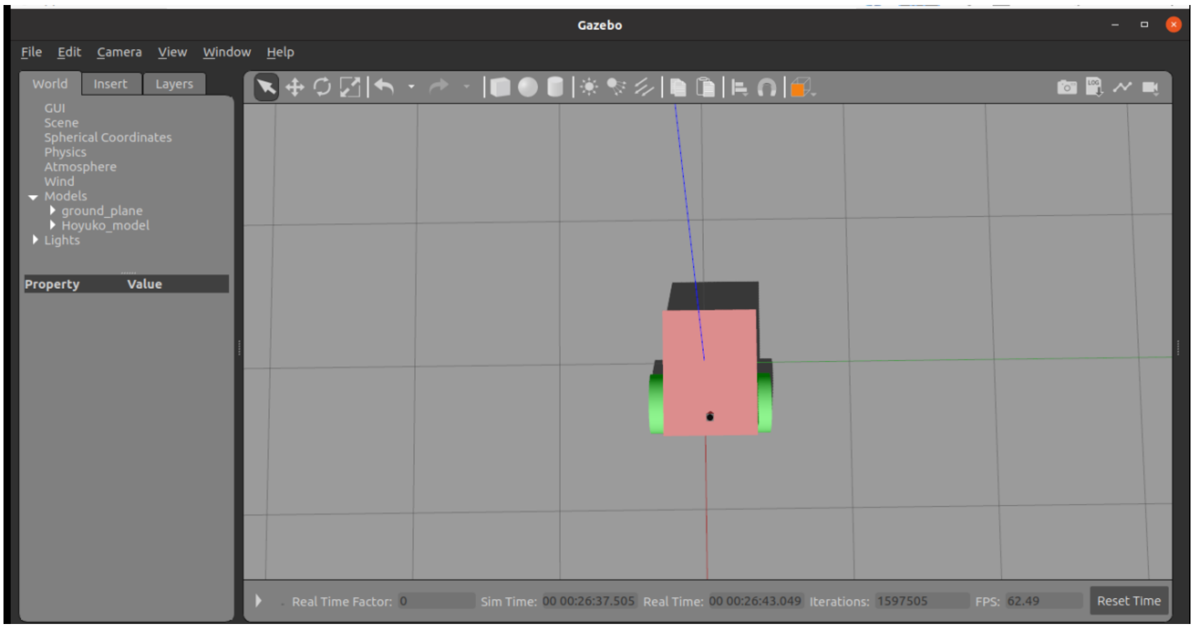
Location: -3-14/model_editor_models/Hoyuko_model Browse

Advanced Options

Cancel Save

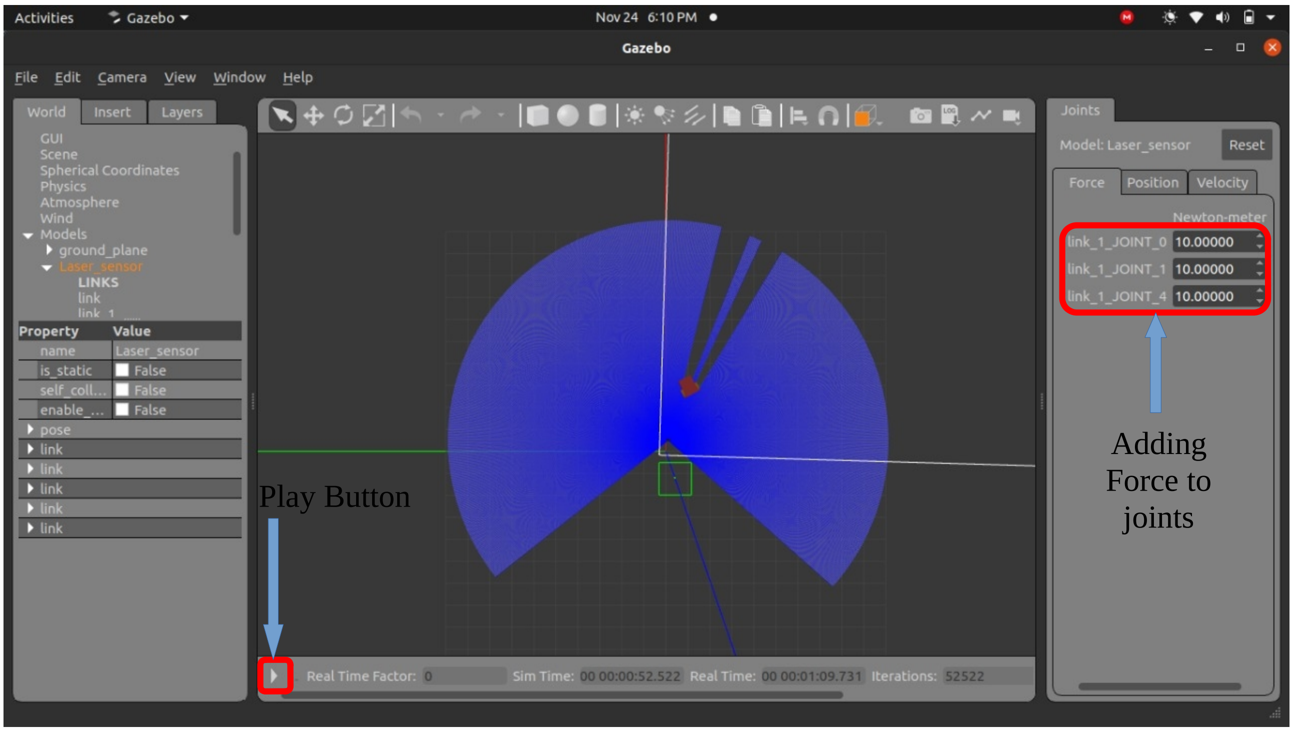


So after exiting our final model will look like this.



Real Time Factor: 0 Sim Time: 00 00:26:37.505 Real Time: 00 00:26:43.049 Iterations: 1597505 FPS: 62.49

Now just Click on play button to start the simulation. You will need to add some force to the wheels in order to move as shown in below image.



## 1. How to create a workspace :-

A workspace is a set of directories (or folders) where you store related pieces of ROS code. So first we will create our own workspace. And this workspace need to be created before you start writing code in ROS.

**Create your workspace:-**

Open up a new terminal window (I’m assuming you are using ROS on Ubuntu Linux), and type the following commands to create and build a workspace.

```
mkdir -p ~/dev_ws/src
```

Navigate to the workspace.

```
cd ~/dev_ws/src
```

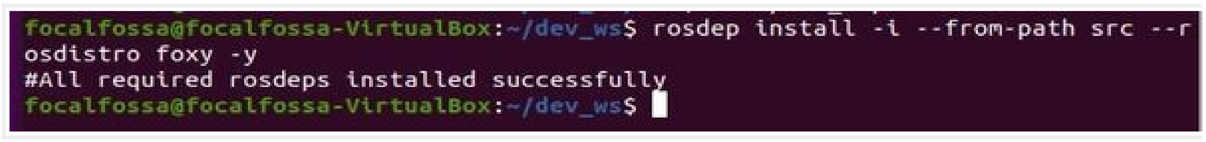
**Resolve Package Dependencies:**

Since we just added a bunch of packages inside our workspace, we need to make sure each package has the software dependencies which its needs in order to run.

```
  Cd ..
```

Now resolve the dependencies by following command:

```
  rosdep install -i --from-path src --rosdistro foxy -y
```



```
focalfossa@focalfossa-VirtualBox:~/dev_ws$ rosdep install -i --from-path src --r
osdistro foxy -y
#All required rosdeps installed successfully
focalfossa@focalfossa-VirtualBox:~/dev_ws$
```

As you can see that all dependencies which our packages required are already installed.

## 2. How to build Your Workspace:

To build the packages, first go to the root of the workspace.

```
  cd ~/dev_ws/
```

Make sure colcon is installed. If not install it by sudo command. Colcon is a tool used to build software packages of ROS.

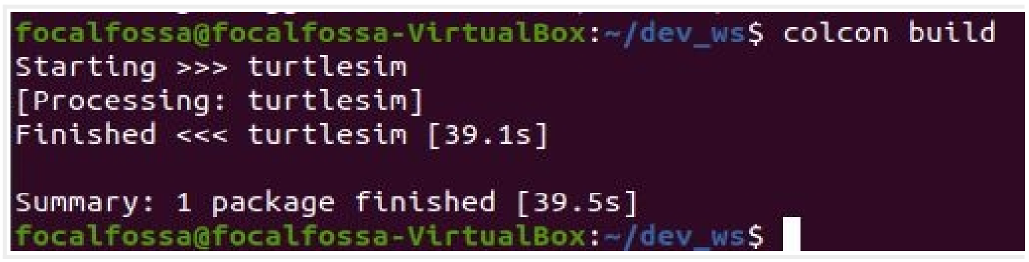
```
sudo apt update
```

```
sudo apt install python3-colcon-common-extensions
```

Press Y and Enter to complete the installation.

Build the packages.

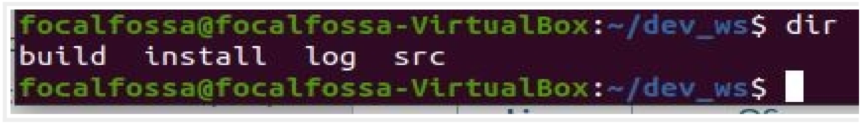
```
colcon build
```



```
focalfossa@focalfossa-VirtualBox:~/dev_ws$ colcon build
Starting >>> turtlesim
[Processing: turtlesim]
Finished <<< turtlesim [39.1s]

Summary: 1 package finished [39.5s]
focalfossa@focalfossa-VirtualBox:~/dev_ws$
```

Type the **dir** command, and you will see four folders inside of this directory: build, install, log and src.



```
focalfossa@focalfossa-VirtualBox:~/dev_ws$ dir
build  install  log  src
focalfossa@focalfossa-VirtualBox:~/dev_ws$
```

## 3. How to source the Workspace:

Now we need to source the setup.bash file. This file sets the path of the workspace so that packages and code inside the workspace can be found.

```
Source install/setup.bash
```

Make sure the workspace is properly overlayed by the setup script (which we ran above).

```
echo $ROS_PACKAGE_PATH
```

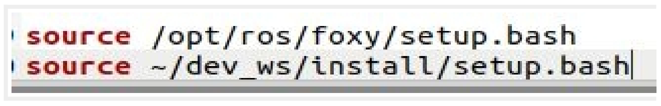
So we don't have to source the setup.bash file every time we open a new Linux terminal, let's add the ~/ros2_ws/install/setup.bash command to the .bashrc file. Open a new Linux terminal window.

Type the following command to edit the .bashrc text file:

```
gedit ~/.bashrc
```

Add this line to the end of the .bashrc file:

```
source ~/dev_ws/install/setup.bash
```



```
source /opt/ros/foxy/setup.bash
source ~/dev_ws/install/setup.bash
```

That's it! You' re all done. Just click Save and exit the text editor.

## 4. How to create our own packages:

First navigate to your workspace where you want to create these packages.

```
cd ~/dev_ws/src
```

Type the following command to build your own packages:

```
ros2 pkg create my_package --build-type ament_cmake --dependencies rclcpp
```

## 5. How to compile any package :

Navigate to the workspace.

```
cd ~/dev_ws/
```

Type the following command to compile the package. Let us assume that package name is my_package, which we have created in the above section.

```
colcon build --symlink-install --packages-select my_package
```

## 6. How to launch any launch files :

Navigate to the workspace.

```
cd ~/dev_ws/
```

Now you have to source to the bash file for both gazebo and ros.

```
source /usr/share/gazebo/setup.sh
```

```
source install/setup.bash
```

Now you are ready to launch any of the files which u have written.

```
ros2 launch my_package obstacle_avoidance.launch.py
```

## 7. Libraries Files of CPP used in this project:

```
#include "rclcpp/rclcpp.hpp"                    // ROS Core Libraries
#include "geometry_msgs/msg/twist.hpp"      // Twist
#include "sensor_msgs/msg/laser_scan.hpp" // Laser Scan
```

## 8. Libraries Files of Python used in this project :

```
import pygame                                # pygame package
import pickle                                 # for storing variables
import numpy                                 # array-processing package
import sys, os, getopt, math, random   # standard library
```

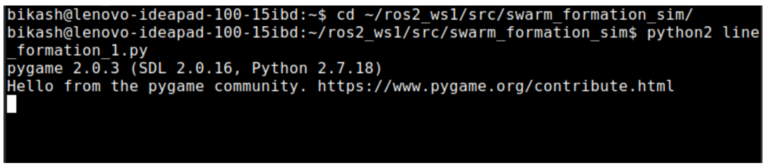
## 9. How to launch a python file in Pygame :

Navigate to the source of the files.

```
cd ~/ros2_ws1/src/swarm_formation_sim/
```

Type the following command to launch your python file in pygame:

```
python2 line_formation_1.py
```



```
bikash@lenovo-ideapad-100-15ibd:~$ cd ~/ros2_ws1/src/swarm_formation_sim/
bikash@lenovo-ideapad-100-15ibd:~/ros2_ws1/src/swarm_formation_sim$ python2 line
_formation_1.py
pygame 2.0.3 (SDL 2.0.16, Python 2.7.18)
Hello from the pygame community. https://www.pygame.org/contribute.html
```

# Bibliography

[1] T. Soleyman and F. Saghafi "Behavior-based acceleration commanded formation flight control", International Conference on Control, Automation and Systems 2010 IEEE, 17 December 2010.

[2] K. K. Oha, M.C. Park , and H.S. Ahnb,"A survey of multi agent formation control", Automatica, Vol. 53, pp. 424-440, 2015.

[3] T. Balch, R.C. Arkin "Behavior based formation control for multi robot team" IEEE Transactions on Robotics and Automation, Vol. 14, No. 6, Dec 1998.

[4] S. Nouyan, and M. Dorigo "Path formation of robotic swarm", Swarm Intelligence, Vol.2, No. 1, pp. 1-23, 2008.

[5] L. Barnes, M.David Fields, and K. P. Valavanis "Unmanned ground vehicle swarm formation control using potential field" Mediterranean Conference on Control 2007.

[6] Adam Campbell and Annie S Wu. Multi-agent role allocation: issues, approaches, and multiple perspectives. Autonomous agents and multi-agent systems, 22(2):317–355, 2011.

[7] Radhika Nagpal, Howard Shrobe, and Jonathan Bachrach. Organizing a global coor-
dinate system from local information on an ad hoc sensor network. In Information processing in sensor networks, pages 333–348. Springer, 2003.

[8] Timmis J, Ismail AR, Bjerknes JD, Winfield AF (2016) An immune-inspired swarm aggregation algorithm for self-healing swarm robotic systems. Biosystems 1(146):60–76

[9] Amjadi AS, Raoufi M, Turgut AE, Broughton G, Krajník T, Arvin F (2019) Cooperative pollution source localization and cleanup with a bio-inspired swarm robot aggregation. arXiv preprint arXiv:1907.09585

[10] Ramroop S, Arvin F, Watson S, Carrasco-Gomez J, Lennox BA (2018) Bio-inspired aggregation with robot swarm using real and simulated mobile robots. In: Annual conference towards autonomous robotic systems, vol 25. Springer, Cham, pp 317–329

[11] Derakhshandeh Z, Gmyr R, Richa AW, Scheideler C, Strothmann T (2016) Universal shape formation for programmable matter. In: Proceedings of the 28th ACM symposium on parallelism in algorithms and architectures. ACM, pp 289–299

[12] Yang J, Wang X, Bauer P (2018) Line and V-shape formation based distributed processing for robotic swarms. Sensors 18(8):2543

[13] Wang Q, Mao X, Yang S, Chen Y, Liu X (2018) Grouping-based adaptive spatial formation of swarm robots in a dynamic environment. Int J Adv Robot Syst 15(3):1729881418782359

[14] Rubenstein M, Cornejo A, Nagpal R (2014) Programmable self-assembly in a thousand-robot swarm. Science 345(6198):795–799

[15] Klavins E (2007) Programmable self-assembly. IEEE Control Syst 27(4):43–56

[16] Ardiny H, Witwicki S, Mondada F (2015) Construction automation with autonomous mobile robots: a review. In: 2015 3rd RSI international conference on robotics and mechatronics (ICROM). IEEE, pp 418–424

[17] Werfel J, Petersen K, Nagpal R (2014) Designing collective behavior in a termite-inspired robot construction team. Science 343(6172):754–758

[18] Ferrante E, Brambilla M, Birattari M, Dorigo M (2013) Socially mediated negotiation for obstacle avoidance in collective transport. In: Distributed autonomous robotic systems. Springer, Berlin, pp 571–583

[19] Shakya A, Mishra M, Maity D, Santarsiero G (2019) Structural health monitoring based on the hybrid ant colony algorithm by using Hooke-Jeeves pattern search. SN Applied Sciences 1(7):799

[20] Mishra M, Barman SK, Maity D, Maiti DK (2019) Ant lion optimisation algorithm for structural damage detection using vibration data. J Civ Struct Health Monit 9(1):117–136

[21] Ducatelle F, Di Caro GA, Pinciroli C, Mondada F, Gambardella L (2011) Communication assisted navigation in robotic swarms: self-organization and cooperation. In: 2011 IEEE/RSJ international conference on intelligent robots and systems. IEEE, pp 4981–4988

[22] Marcolino LS, dos Passos YT, de Souza ÁA, dos Santos Rodrigues A, Chaimowicz L (2017) Avoiding target congestion on the navigation of robotic swarms. Auton Robots 41(6):1297–320

[23] Inácio FR, Macharet DG, Chaimowicz L (2018) United we move: decentralized segregated robotic swarm navigation. In: Distributed autonomous robotic systems. Springer, Cham, pp 313–326

[24] Nir Kshetri and Diana Rojas-Torres. The 2018 winter olympics: A showcase of technological advancement. IT Professional, (2):19–25, 2018.

[25] William M Spears, Diana F Spears, Jerry C Hamann, and Rodney Heil. Distributed, physics-based control of swarms of vehicles. Autonomous Robots, 17(2-3):137–162, 2004.

[26] Edward A Macdonald. Multi-robot assignment and formation control. PhD thesis, Georgia Institute of Technology, 2011.

[27] Alex Kushleyev, Daniel Mellinger, Caitlin Powers, and Vijay Kumar. Towards a swarm of agile micro quadrotors. Autonomous Robots, 35(4):287–300, 2013.

[28] Michael Rubenstein, Alejandro Cornejo, and Radhika Nagpal. Programmable self-assembly in a thousand-robot swarm. Science, 345(6198):795–799, 2014.