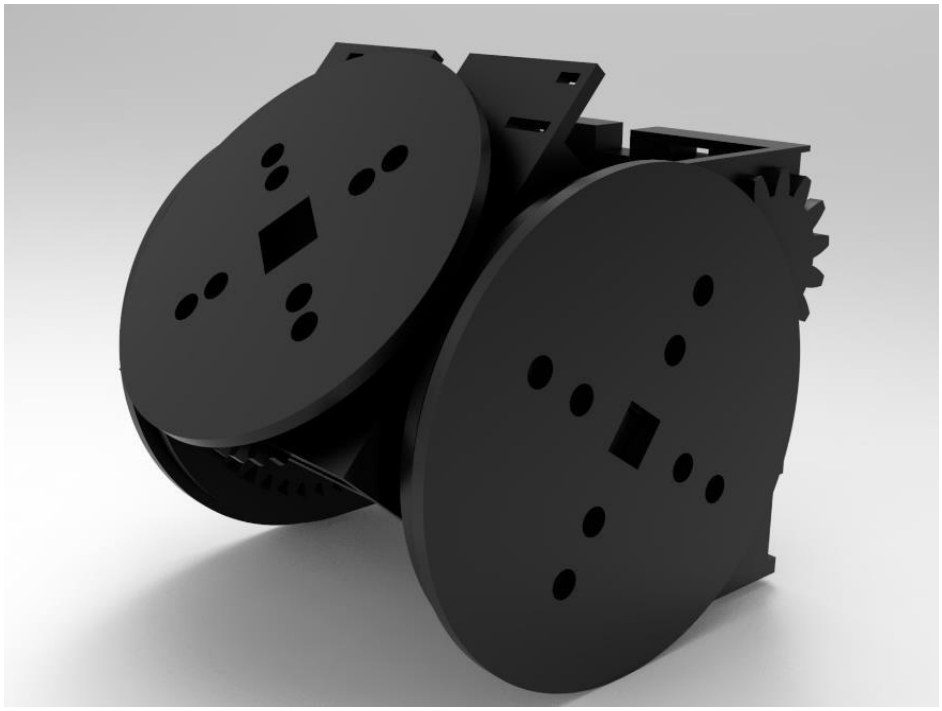
MODULAR SELF-RECONFIGURABLE ROBOT

Submitted By:

Amr Mohamed Ahmed
Ahmed Khalid Mohamed Ali Hassan
Laila Mohamed Farid
Mennallah Muhammed
Michael Ehab
Mahmoud Mohamed Abdel
Mohamed Tarek Abdellatif



Mechatronics and Automation Engineering Graduation Project Report

Supervisors

Dr. Shady Ahmed

Dr. Mohamed Ibrahim

Wednesday, January 29, 2020

1 Elsarayat St., Abbaseya, 11517 Cairo, Egypt

Fax: (+20 2) 26850617

www.eng.asu.edu.eg

QR

DECLARATION

We/I hereby certify that this Project submitted as part of our/my partial fulfilment of BSc in Mechatronics and Automation Engineering is entirely our/my own work, that we/I have exercised reasonable care to ensure its originality, and does not to the best of our/my knowledge breach any copyrighted materials, and have not been taken from the work of others and to the extent that such work has been cited and acknowledged within the text of our/my work.

Signed: by all students

Amr Mohamed Ahmed	
Ahmed Khalid Mohamed Ali Hassan	
Michael Ehab	
Mennallah Muhammed	
Laila Mohammed	
Mahmoud Mohamed Abdel	
Mohammed Tarek Abdellatif	

Date: Wednesday, 29 January 2020.

ABSTRACT

Due to the drawbacks of traditional mobile robots, a new type of mobile robots rose known as reconfigurable mobile robots. they are at heart mobile robots, yet they possess the ability to change their overall shape by changing the way they are assembled together.

In our project, we seek to implement the concept of modular mobile reconfigurable robots to gain the best of both worlds overcoming the limitations of size so that the robots can enter through a small space and then assemble to do their required task mainly search and rescue missions for example in the ruins of a building. We are going to be designing a group of six independent differential mobile robots that are able to self-assemble and move in the environment using different configurations under the guidance of humans.

ACKNOWLEDGMENT

The completion of this report could not have been possible without the participation and assistance of so many people. We would like to thank:

Dr. Shady Ahmed, and Dr. Mohamed Ibrahim for their endless support, insight and understanding throughout the semester.

Professor Jay Davey, for his supportive data and guidance throughout our progress

To all relatives, friends, and others who in one way or another shared their support, either morally, financially, or physically.

We thank you.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER ONE: INTRODUCTION

When disasters such as earthquakes, landslides or avalanches occur, and people end up trapped rescue teams face a lot of problems reaching them due to the accessibility to small and dangerous places.

Exploring unknown areas and has been a problem for humankind, countless lives were lost due to people risking their lives in such tasks, furthermore, building structures in the outer space has been also a dilemma for humans.

In a nutshell, the problems that our projects target are:

-Search and rescue operations

-Exploration missions in tight places

-Building Structures in outer space

-Disaster management

Figure 1.1.1 Current Problems Face by Humans

Hence people turned to robotics, robotics is a wide field where its main objective is to assists us and perform complex functions that were beyond human capabilities. From mars rovers to quadrupeds all of these where robotics that where introduced in the recent era to solve current problems but Most of the traditional robotic solutions are operated in a controlled environment and any changes in environments often make these traditional solutions inflexible due to lack of their adaptive nature. The repair and maintenance of such conventional designs generally require separate trained personnel for each model and hence increasing the average resource consumption in industries.

1.1 Motivation

The growing demand for reusable, space-constrained, and multipurpose solutions for real- world applications is a great motivator for research in the field of MSRR.

There are three key motivations for designing modular self-reconfigurable robotic systems.

Versatility: Self-reconfigurable robotic systems are potentially more adaptive than conventional systems. The ability to reconfigure allows a robot or a group of robots to disassemble and reassemble machines to form new morphologies that are better suited for new tasks, such as changing from a legged robot to a snake robot and then to a rolling robot.

Robustness: Since robot parts are interchangeable (within a robot and between different robots), machines can also replace faulty parts autonomously, leading to self-repair.

Adaptability. While the self-reconfigurable robot performs its task it can change its physical shape to adapt to changes in the environment

Low Cost: Self-reconfigurable robotic systems can potentially lower overall robot cost by making many copies of one (or relatively few) type(s) of modules so economies of scale and mass production come into play. Also, a range of complex machines can be made from one set of modules, saving costs through reuse and generality of the system.

The researchers in the domain of MSRR provided numerous solutions via various prototype designs, communication algorithms, coordination, and dispersion techniques using selected test

scenarios. The development of novel prototypes for MSRR is a analytical process that often has deep roots in intuition and derives better fruits from experience on basic locomotion and laws of physics. The different approaches adopted by researchers to validate the designs and prototypes make a relative comparison between the robotic modules a fairly difficult process and present challenges in quantifying and understanding the merits/demerits of various designs.

1.2 Aims

Our aim is to implement a prototype design for a modular reconfigurable robot that will deal with the current world applications needs and based on this we set basic parameters in which the robot should do this included:

- Modular system

Modular robots are composed of various units or modules, hence the name. Each module involves actuators, sensors, computational, and communicational capabilities. Usually, these systems are homogeneous where all the modules are identical; however, there could be heterogeneous systems that contain different modules to maximize versatility

- Self-Assembly

One of the main benefits of modularity is the capability of self-assembly, which is the natural construction of complex multi-unit system using simple units governed by a set of rules. The self-assembly process is ubiquitous in nature as it generates much of the living cell functionality. However, it is uncommon in the technical field because it is considered as a new concept relatively in that arena, although it could help in lowering costs and improving versatility and robustness; which are the three promises of modular robotics. The ability to form a larger stronger robot using smaller modules allows self- assembled robots to perform tasks in remote and hazardous environments.

- Self-Reconfiguration

Recently, modular robotics has gotten attention from researchers in the robotics field because of their ability to self-reconfigure [ref 27 fel list]. Modular self-reconfigurable robots involve various modules that can combine themselves autonomously into meta- modules that are capable of performing various tasks under different circumstances [5]. The ability to self-reconfigure allows these robots of metamorphosis,

which in turn makes them capable of performing different sorts of kinematics. For instance, a robot may reconfigure into a manipulator, a crawler, or a legged one.

- Self-Repair

Self-repair is a special type of self-reconfiguration that allows a robot to replace damaged modules with functional ones in order to continue with the task at hand [27]. A self-repair system must have two qualities: the ability to self-modify, and the availability of new parts or resources to fix broken ones. Therefore, modular self-repair robots usually consist of redundant modules. Self-repair involves two phases: detecting the failure module, and then ejecting the deficient module and replacing it with an efficient extra module.

1.3 METHODOLOGY

Systems engineering is the use of specific processes to design and develop systems. Systems engineering methods and processes are used to derive an assembly of potential solutions from user needs and requirements, focus solutions on satisfying the users' needs, and consider internal and external influences on the system both short term and long term. In this regard, systems engineering is a life-cycle approach to designing and building products and services.

Thus, we started following the V-Model diagram shown below.

- Requirements.
- System Design.
- Verification
- Validation.
- Performance measures
- Assurance of Properties.

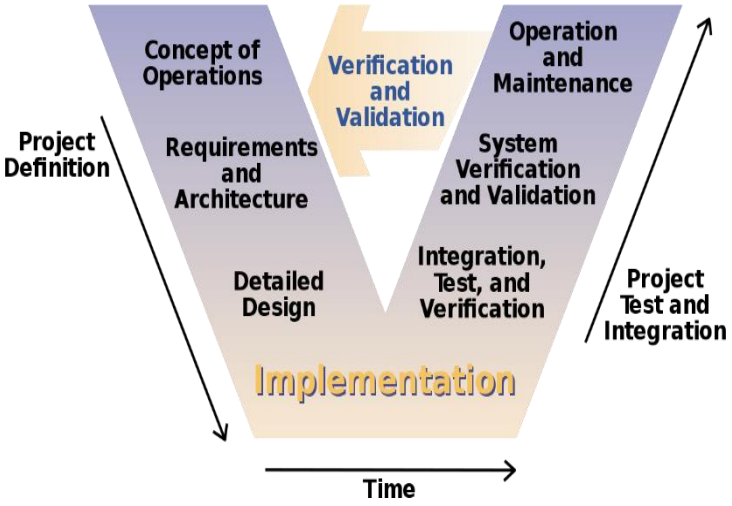


Figure 1.3.1 V-Model

1.3.1 Requirements

Designing a Modular Self Reconfigurable robots, Highlighted Important Needs:

- Moving Smooth without noise.
- Compact size
- HMI.
- Safe and Stable System.
- Ease of access to Electrical components for Debugging.
- Easy control.
- A battery that withstands a full day of work.

1.3.2 System design

1.3.2.1 Preliminary Design

We drew a freehand sketch design that met our functions we need from the data we collected, and we added some extra features to expand the functionality of our robot as well as make it more “high tech”.

The design of our robot was broken down into 3 main parts:

- Base

  This will contain 4 DC motors driving our robot using a coronal differential gear drive

- Docking and Undocking

  The robot will have 3 active connectors and 1 passive connector, where each connector will contain 4 magnets

- Reciprocating key

  A key is a component added in the project that will aid in the support of the robots as well as help in the undocking of the robots.

1.3.3 Verification

Verification is an assessment of a system's limitations to determine whether the requirements have been satisfactorily met. This assessment is performed throughout the system engineering and development process to identify and correct issues early in design. To maximize benefit, verification is not only performed at the end of the design and integration cycle, but throughout the design, development, and integration phases of a system to correct any deficiencies and to provide evidence that the system is adequate to meet the defined requirements.

1.3.4 Validation

Validation is like verification in that it is a check of the design process to ensure the solution and result of the systems engineering process meets the user's needs. Final validation of the system involves testing of an operational, production-representative system in the intended operational, or suitably simulated, environment. Additionally, end- user and other external stakeholder involvement in validation activities can ensure independent confirmation that the correct system has been developed and functions as needed.

1.3.5 Performance Measures

Measures of performance are a subset of measures of effectiveness that are used to specifically gauge the abilities of a system. The system-particular performance parameters are speed, payload, range, time-on-station, frequency, or other distinctly quantifiable features. They are used to quantify the ability of the system to achieve specific, individual attributes and are consequently tied to verification of the system and are selected and iterated through requirement allocation to features of the system.

1.4 BIG PICTURE

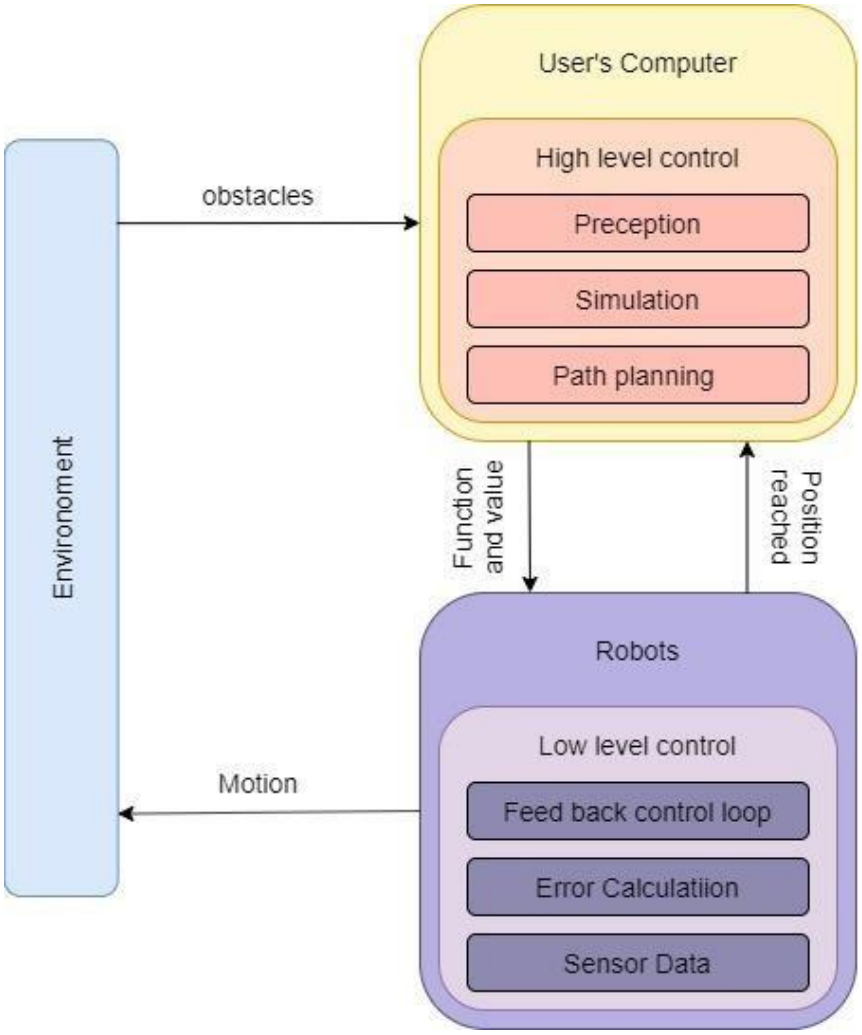


Figure 1.4.1 Big Picture Diagram

1.5 PLAN

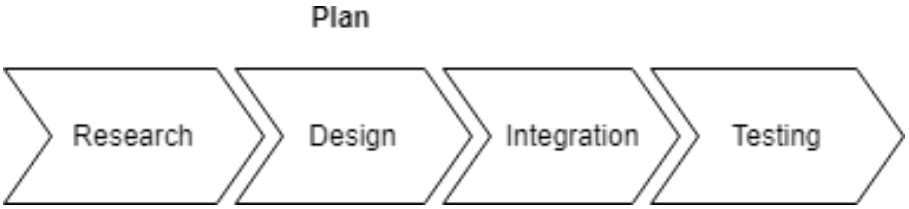


Figure 1.5.1 Plan

- Research

In the research part we searched for different types of reconfigurable robots and different types of control algorithms that was implemented till now. There were also different design to

explore, such as Cubic structures, Hexagonal structures and so on , there was also different architectures that could have been implemented .

- Design phase

The design phase is where the main design of the robot took place, it started with the mechanical design, followed through with the actuator sizing, simulation then electrical design to actuate the modules. We also wanted to explore different configurations that we are going to apply in our project through simulation software's.

- Integration

In the integration phase we will integrate the mechanical parts together to see the overall shape and also place the required sensors in order to achieve the required function.

- Testing

In the testing phase we will test all the sensors , the circuit and the movement of the module , we will carry out different phases of testing where we will test each module independently and then test full functional configurations that we selected in the previous phase .

CHAPTER TWO: LITERATURE REVIEW

Research into reconfigurable robots is vital to the current era, The roots of the concept of modular self-reconfigurable robots can be traced back to the “quick change” end effector and automatic tool changers in computer numerical controlled machining centres in the 1970s. Starting from 1998 till our present-day the reconfigurable field has been implemented with various designs, methods of control and even connector mechanisms, in here we are going to explore the robots that inspired our final design.

2.1 TYPES OF RECONFIGURABLE ROBOTS

Before going to deep on different types of developed reconfigurable robots, let us speak about architecture groups of the robots:

MRR systems can be classified into several architectural groups based on the geometrical arrangement of the units:

- Lattice reconfiguration architectures: Lattice reconfiguration architectures have units that are arranged in a regular, three-dimensional pattern, such as a cubic crystal lattice or cannonball packing. These systems exploit this regularity to ease the computational aspects of reconfiguration.

- Chain architectures: Chain architectures are characterized by units that form serial chains. These chains are often connected to formatter or closed-chain loops. Through articulation, chain architectures can potentially reach any point or orientation in space, and they are therefore more versatile. Generally, however, they are more demanding to represent and analyse computationally and more difficult to control.

- Mobile architectures: Mobile architectures have units that use the environment to manoeuvre around and can hook up to form complex chains, lattices, or a number of secondary robots that can perform swarm-like behaviours. A fourth MRR class has started to emerge: reconfigurable trusses. These systems do not fit into the lattice, mobile, or chain style (serial chains are not an inherent part of the system). Instead, prismatic members form parallel truss structures that can reconfigure, changing the topology of the network.

In addition, hybrid systems can often exploit the best properties of multiple different architectures.

There are a variety of aspects to consider when it comes to designing MRR systems. The hardware and software design issues in MRR systems are highly intertwined, while in conventional robotic systems they are often examined separately. Interestingly, many of the more difficult implementation issues occur on the software side.

2.1.1 POLYBOT

PolyBot is a modular self-reconfigurable robot that was implemented to explore how realistic it is to make robots using several homogeneous hardware modules. Three generations of PolyBot modules were prototyped, such that each generation addresses a number of shortcomings discovered in the previous one. The first generation (G1) is constructed using two modular types: node and segment. The segment modules are nominally rectangular prisms with 1 rotational DOF separating two connection ports. The design of the polybot was very simple that a single module cannot do that much without connecting with the other modules.

The node modules are fixed passive cubes with 6 connection ports. Unlike its G1 predecessor, the second generation (G2) connection ports have electromechanical latches under software control. These latch onto the pins protruding from the opposite face. The third generation (G3) modules are smaller and lack the DC motor extending past the side of each module. The new module has instead a DC pancake motor with a harmonic gear that is completely internal. The connectors are larger and have higher contact force for higher current loads to enhance performance. The first two generations of PolyBot prove versatility by executing locomotion over different terrains. However, as the number of modules increases, cost increases, and robustness decreases because of software's capability and hardware dependency problems. Currently, the maximum number of modules utilized in one connected PolyBot system is 32 with each module having 1 DOF. The third generation deals with 200 modules to show a variety of capabilities, including moving like a snake, lizard, or centipede as well as humanoid walking and rolling in a loop [33–36].

PolyBot is capable of self-reconfiguration by changing its geometry and locomotion mode depending on the terrain type – rolling over flat terrain, earthworm to move around obstacles, and a spider to step over hilly terrain. Planning the self-collision-free motions can be challenging because the size of this space is exponential in the number of modules, but proportional to the number of DOF. For many applications, a fixed set of configurations is sufficient. In this case, reconfigurations can be pre-planned off-line and stored in a table for ease of reconfiguration.

Figure 2.1.1 PolyBot Single Module

Figure 2.1.2 PolyBot Assembled Module

2.1.2 M-TRAN

M-TRAN (Modular Transformer) is a distributed lattice-based self-reconfigurable robotic system that can metamorphose into various configurations, such as a legged machine generating walking motion. The actual system was built using 10 modules and successfully demonstrated the basic operations of self-reconfiguration and motion generation.

The design of M-TRAN has the advantages of both two types of modular robots, lattice type and chain (linear) type. This hybrid design, 3-D shape of the block parts, and parallel joint axes are all keys to realize a flexible self-reconfigurable robotic system. Possible applications of the M-TRAN are autonomous exploration under the unknown environment such as planetary explorations, or search and rescue operation in disaster areas.

An M-TRAN module is made of two semi-cylindrical parts connected by a link. Each semi-cylindrical part can rotate about its axis by 180 degrees with a servomotor embedded in the link. Each semi-cylindrical part has three connecting surfaces with four permanent magnets and can connect to other modules' surfaces that have the opposite polarity of the magnets. The two semi-cylindrical parts have surfaced of different polarity.

The link part includes two sets of geared motors and servo circuits. The maximum torque of each servo is enough to lift up one module. On all the connection surfaces, there are electrodes for power supply and serial communication. There are two parts, called active and passive part. Inside the active part, there is a connection mechanism that is made of non-linear springs, SMA (Shape Memory Alloy) coils and four permanent magnets, which are fixed on the connecting plate. This mechanism is based on the technique of IBMU (Internally Balanced Magnet Unit)

The connecting plate moves automatically to connect to the other modules' surface by the attractive force of magnets. It is detached by heating SMA coils by electric current. When detaching, the springs help SMA coils. Inside the passive part, there are control circuits including an onboard micro-computer. This controls motor rotation and heating of SMA coils. Control commands corresponding to these operations are issued by the host computer through the serial communication.

Each M-TRAN module has its own controller and intelligence, and all the controllers work cooperatively forming a Distributed Autonomous System as a whole.

In order to drive M-TRAN hardware, a series of software programs have been developed including a kinematics simulator, a user interface for designing configurations and motion sequences, and an automatic motion planner. M-TRAN II is the second prototype where many improvements took place to allow versatile whole-body motions and complicated reconfigurations.

The third prototype, M-TRAN III, has been developed with an improved connection mechanism. Various control modes including single-master, globally synchronous control and parallel asynchronous control are made possible by using distributed control. Self-reconfiguration experiments using up to 24 units were performed by centralized and decentralized control. System scalability and homogeneity were maintained in all experiments.

Figure 2.1.3 M-Tran Formation A

Figure 2.1.4 M-Tran Formation B

2.1.3 ATRON

ATRON is a lattice-based system, in which modules are arranged in a subset of a surface cantered cubic lattice. In this lattice, modules are placed so that their rotation axis is parallel to the x, y or z axis. Modules are placed so that two connected modules have perpendicular rotation axes. The basic motion primitive for ATRONs is a 90◦ rotation around the equator, while one hemisphere is rigidly attached to one or two other modules and the other hemisphere is rigidly attached to the main part of the structure. This will cause the attached module(s) to be rotated around the rotation axis of the active module. Actuation is realized as rotation around an axis diagonally through the sphere, where each module can rotate 360◦ around the equator. This design allows for a very stable construction around the actuated joint since a relatively large area is available for mechanics. However, the spherical basic module design makes it hard to have large flat surfaces connecting to each other. With spherical modules, connectors need to establish essential point-to-point contacts between modules, which are not desirable because of the high collision probability.

The limited mobility of ATRON along with other motion restrictions leads to the use of ATRON meta module to reduce motion constraints. The meta-module is composed of 3 modules: a body in the centre that is connected to two legs. Modular ATRON control comprises three Artificial Neural Networks ;one to decide when to emerge, the second to decide when to stop, and the third to calculate the fitness value of every state in the self- reconfiguration and self-repair processes.

Genetic Algorithm is used to optimize the weights of the ANNs. Even though ATRON modules are minimalistic because they have only one actuated DOF, a group of modules was capable of self-reconfiguring in 3D simulation. Similarly, ATRON modules demonstrated self-repair successfully in simulation.

Figure 2.1.5 ATRON in Motion

Figure 2.1.6 ATRON Mechanical Concept

2.1.4 SUPERBOT (2006)

The SuperBot, seen in Figure X, has been developed by Shen et al. at the University of Southern California as a deployable self-reconfigurable robot for real-world applications outside laboratories. Its modules have a hybrid chain and lattice architecture [11]. The modules have three DOF (pitch, yaw, and roll) and can connect to each other through one of the six identical dock connectors. They can communicate and share power through their dock connectors. For high-level communication and control, the modules use a real- time operating system and the hormone-inspired control developed for CONRO [12] as a distributed, scalable protocol that does not require the modules to have unique IDs.

The basic design philosophy of Superbot modules is to develop flexible, powerful and sturdy modules that can efficiently perform tasks in an uncontrolled environment without requiring close attention. In order to accomplish this goal six criterions were considered in the design and construction ofSuperBot. First, as SuperBot is intended to operate in a harsh and rough environment, the design needed to allow for roughed and sealable modules. Modules and connectors needed to cover their internal electronic and mechanical components and protect them from dust, moisture, and physical impact. The building materials needed to be resistant to abrasion and other deleterious effects.A communication interface has four infra-red receiver LEDs and a transmitter LED. Any combinations of the receiver channels can be selected which results the sum of the received signals on each receiver LED to be delivered to a buffer stage. The output ofthe buffer is connected to an A/D channel of the corresponding controller

the SuperBot system to be:

Distributed: to support decentralized control and avoid single point failures (i.e., a single module failure would not paralyze the entire system). A module must select its actions based not on its

absolute address or unique identifier but based on its topological location in the current configuration.

Collaborative: to allow modules to negotiate the best actions for a global task. For example, if a snake's head module wants to move forward while the tail wants to move backward, then they must negotiate to select the best action for the entire system.

Dynamic: to be able to adapt to the topological changes in the module network and support all possible configurations.

Asynchronous: to synchronize modules actions without a global clock. * Scalable: to work for any configuration regardless of the shape and size.

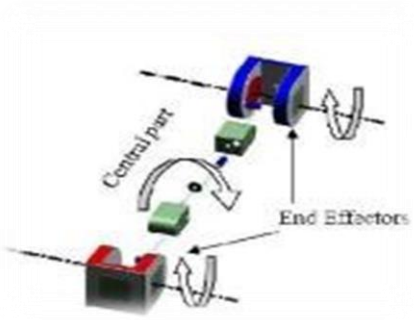


Figure 2.1.7 Superbot Assembly Concept

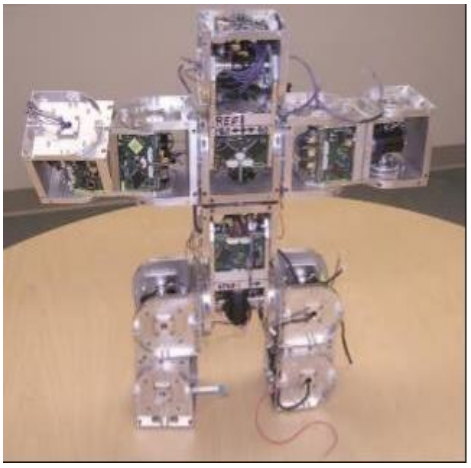


Figure 2.1.8 Superbot Robot Assembly

2.1.5 CONRO

chain-typed self-reconfigurable robot, Fig shows the CONRO robot system made of small-sized modules that can autonomously and physically connect to each other to form different configurations such as chains, trees, (e.g., legged-bodies), or loops. The top left picture shows a single autonomous CONRO module; the top right picture shows a CONRO chain (snake) configuration with eight modules, the bottom left has two CONRO insects (tree configuration) each of which has six modules for legs and three modules for the spine, and the bottom right is a CONRO loop configuration with eight modules. Each configuration can perform its locomotion, and the robot can autonomously change configurations in limited situations.

the physical connectors in CONRO must be joined and disjoined physically to change shape. Such changes in the network topology make a CONRO robot a dynamic network.

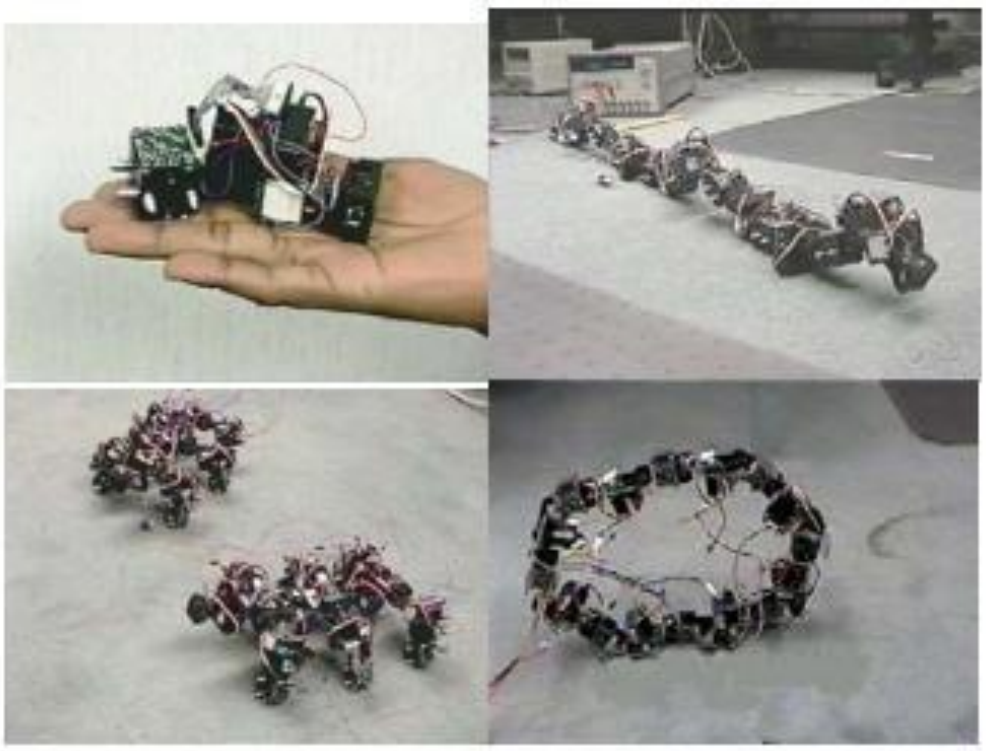


Figure 2.1.9 CONRO Formation

2.1.6 UBOTS

The UBot MSRR system consists of cubic structured cells capable of rotating in discrete steps along longest diagonal similar to Mole cubes. The internal faces are chamfered for facilitating rotation. The UBot robotic cells are categorized into active and passive modules with active modules providing four active connection interfaces and passive modules providing four passive connection interfaces. The active and passive modules have the same outer structures and rotation mechanisms. The hooks present on active connection interfaces enable firm docking with passive connectors. The active and passive modules are latched using hook and sliding mechanism guided by position sensors for forming lattice and chain structures in 3D making UBot a hybrid category robot.

2.1.7 MOLE CUBES

Mole cubes system is an open hardware and software plat form for modular robotics that was developed to remove entry barriers to the field and to accelerate progress. The system is composed of modules with 1 rotational DOF. Different types of active modules, such as gripper, actuated joint, controller, camera, and wheel along with a few passive modules were presented. Each module is a cube shaped with round corners that comprises approximately two triangular pyramidal halves connected with their bases so that their main axes are coincident. Each of the 6 faces of the module is equipped with an electromechanical connector that can be used to join two modules together. Symmetric connector design allows four possible relative orientations of two connected module interfaces, each resulting in different robot kinematics [52]. Genetic Algorithm is used to evolve the modular neural network control of the robots in simulation to generate a certain behaviour or motion [53]. To achieve self-replication, path planning is done with a gene pool that has been built using evolutionary algorithm.

Figure 2.1.10 Mole Cube in Motion

2.1.8 MICHE (2006)

The Miche system, shown in Figure 6, has been developed by Rus et al. at MIT. It is a modular lattice system capable of arbitrary shape formation. This system achieves self- assembly by disassembly and has demonstrated robust operation over hundreds of experiments. Each module is an autonomous robot cube capable of connecting to and communicating with its immediate neighbours. The connection mechanism is provided by switchable magnets. The modules use face-to-face communication implemented with an infrared system to detect the presence of neighbours. When assembled into a structure, the modules form a system that can be virtually sculpted using a computer interface and a distributed process. The group of modules collectively decides who is and is not on the final shape using algorithms that minimize the information transmission and storage. Finally, the modules not in the structure let go and fall off under the control of an external force, in this case, gravity. All the algorithms controlling these processes are distributed and are very efficient in their space and communication consumption.

Figure 2.1.11 Miche Assembly

2.1.9 SAM BOTS

Self-assembly Modular Robot for Swarm Robot, The Sambot is a self-assembly robot combining advantages of the mobile and chain-based reconfigurable robot. That is, each Sambot module is a fully autonomous mobile robot that is similar to individual robot in swarm robot, while multiple Sambot can construct a robotic structure such as a snake-like robot or multi-legged crawler robot through self-assembly. The robotic structure has the ability to locomotion and reconfiguration similar to a chain-based reconfigurable robot. In order to achieve the above functions, Sambot must meet the following design requirements: 1)

Autonomy: Each module is an autonomous mobile robot including the power supply, microprocessor, drives, sensors and communications unit. 2) Self- assembly: in order to realize the self-assembly, each Sambot should have active docking mechanism, so that it can realize autonomous connection and disconnection of two or more modules. 3) Motion Ability: the robotic structures assembled with multiple modules should have the locomotion same to that of chain-type reconfiguration robots. 4) Self- reconfiguration or self-metamorphic: the robotic structures assembled with multiple modules should have the ability of self-reconfiguration or self-metamorphic,

This unit uses STM32 microprocessor of the ARM series as the main processor. The STM32 completes the robot's navigation localization and the decision-making task \

This unit can collect the information on gyroscope and accelerometer and receive the encoder information through the I2C interface.

The autonomous docking between Sambots is divided into four phases, namely finding, guiding (navigating), docking and locking. At the beginning of docking, one Sambot (called the active Sambot), through the detecting infrared sensors detects the existence of another Sambot (called the passive Sambot). When it detects the passive Sambot, the docking infrared sensor at the active docking surface can receive the signal sent by the approaching infrared sensor at the passive docking surface of the passive Sambot; and then, guided by two pairs of docking infrared sensors, the active Sambot approaches the passive Sambot.

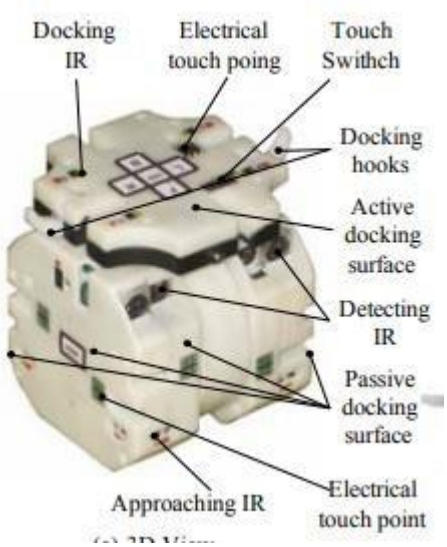


Figure 2.1.12 Sam Bot Module

Figure 2.1.13 Sam Bots Formations

2.2 Comparing Characteristics of Reconfigurable Robots

Robot	DOF	Interface	Type	No of Actuators
M-TRAN III	2	Permeant Magnets/Mechanical latch	Hybrid	5
Polybot	1	Permeant Magnets	Lattice	2
Superbot	3	Permeant Magnets	Mobile/Hybird	9
ATRON	1	Mechanical	Lattice	6
Conro	3	Permeant Magnets	Hybrid	6
U-bot	2	Mechanical latch	Chain	2
Molecubes	2	Permeant Magnets	Chain	8
SMORES	4	Permeant Magnet/ Mechanical Latch	Mobiles / Hybrid	4

Table 2.2.1 Comparison between different types of robots

CHAPTER THREE: DESIGN OF MSRR

The design described is inspired by the SMORE reconfigurable design it also integrates the features from both M-TRAN and CONRO to accomplish multimode locomotion. The design could be broken down into 3 main goals

System design goals

The be polymorphic, assuming many different shapes and configurations; metamorphic, changing between reconfigurations without physical human assistance and; inexpensive, not overly redundant so that the system becomes prohibitively expensive.

Module design goals

the modules should be able to reconfigure using lattice, chain and mobile style reconfiguration. The arrangement and number of DoF, number and type of docking ports, and geometric shape should enable the largest range of useful motions and configurations with the minimum number of actuators. Furthermore, the design should be as compact as possible that's why we need to limit the number of the power transmission system as much as possible

Docking system goals

The docking system should enable modules to connect in many useful arrangements. A system that can be manually placed in any configuration during experiments without needing power to the robot to connect/disconnect modules saves time during experiments. For this reason, we want modules that can be easily manually docked/undocked. Another useful property we want for the connectors is to be impact resistant [reference 20 in Jay Davey paper]. If a robot falls over or is subject to sudden impact such as an explosion, it is better if the modules disconnect rather than have the connectors physically breaking. For robustness, connectors should be able to disconnect from each other even if one of them is non-responsive. Finally, the docking connector should be fast, power efficient.

The optimal connector would have the following features:

- Small size
- Fast

- Strong
- Robust to wear and tear
- High tolerance to alignment errors
- Energy use only in the transition phases
- Transferal of electrical and/or communication signals between modules
- Genderless Allows connection with different orientations
- Disconnects from both sides
- Dirt-resistant

3.1 MECHANICAL DESIGN

3.1.1 Module movement

The module movement is divided into 4 DoF movements as illustrated in Fig.x, where DoF 1 and 2 are the wheel movement Dof #3, is the Tilt and Dof #4 is the pan movement. Where Dof 3 is perpendicular to the other 3. Dof#1, # 2 and #3 produce the twist motion while DOF 4 produces a bending joint.

These motions are achieved by a gear drive train. The concept of the gear train is that there are 4 main gears 2 inner gears and 2 outer gears. the 2 outer gears have a reduction ratio of .. which is responsible for transmitting the torque. the 2 inner gears have a gear crowned on them which helps to create 2 types of motion according to the gears. This fifth gear spins about the DoF #3 axis (pans) when the two inners" Pan and Tilt" spur gears are rotated in the opposite direction. When the two inner spur gears rotate in the same direction, the gear rotates (tilts) about DoF #4. Note that this last action allows the torque of two motors to be combined to increase the torque for this DoF.

Mobile movement of individual modules has been achieved on other self-reconfigurable modular robots using wheels [1] [6] [11], treads [8], the module's DoF and geometric shape [13] [7], and external vibrations. On the JHU system, Kutzer et al. [11] include a docking connector on a wheel. This implementation requires trajectory planning to correctly align the connectors during self-assembly [16]

MSRR has three connection plate disks on DoF #1, #2 and #3. The two opposing disks on DoF #1 and #2 are used as driving wheels and are slightly larger than the one on DoF #3. This allows differential wheeled drive locomotion of individual modules. Its advantages include control simplicity, efficient forward motion, and a small turning circle. The Driving wheels are fitted with rubber timing belts to provide grip (to avoid slip). The third point of contact occurs on the edge of the square face of the module as a low friction skid, seen in Fig. 1. The centre of mass is close to the geometric centre of the module (over the wheels) so the skid sees nominally light contact.

MSRR has demonstrated mobile movement on smooth flat surfaces and is capable of driving upside down. Some previous mobile module designs have not opted for this ability. Tilting the module (bending DoF #4) causes the skid and disk to contact flat terrain allowing MSRR to raise or lower the driving wheels. Raising the wheels allows SMORES to decouple connector orientation on the driving wheels without moving on the ground plane.

3.1.2 Docking and Undocking Actuation

The complexity of a module generally increases as the number of parts, and moving parts, increases. When designing a new module, the more connectors and DoF it has, it generally follows that the module will have more moving parts and hence, could be more complex.

Our robots have 3 active docking parts and 1 passive docking parts that can be represented in the gears diagram. Active docking ports control the attachment process and passive ports only provide a physical space for a neighbouring module to attach. The number of connectors on the robot can affect its versatility For instance, M-TRAN [12], Polybot [18] and CKbot [20] modules can connect two mating connectors in four different orientations. ATRON [9] can connect neighbouring modules in two orientations. In the case of the chain reconfiguration strategy, orientation can be limiting if connectors are not able to dock at arbitrary angles, even when the connectors are coincidently aligned. With SMORES[1] and the JHU [11] design, the

docking ports can rotate to any angle along with the docking, that's why we concentrated on the SMORES design in the first place.

The force provided to connect the two modules are done by 8 permeant magnets 4 per connector, this allows for a connector that is easily manually reconfigurable, self-reconfigurable and impact resistant. Docking two ports together require that the north and south permanent magnets are aligned. Docking Connector Strength: the main goal for docking connectors is maximizing the strength of the connector when attached and minimizing the force required to disconnect. Our system uses magnetic connectors which can be made to repel when disconnecting. The holding force in the tension between two docked modules is provided only by the attractive force of the permanent magnets. One pair of our neodymium-based magnets 6mm diameter x 6mm long, exertapullforceof14.5N. Two connected SMORES faces have four pairs resulting in a theoretical connector holding force of 58 N in tension. However, the magnets have almost half of this value in the direction of shear, almost 30 Newtons that is why we used key mechanism.

3.1.3 Parts

3.1.3.1 Gear Fixation

This was the tricky part we needed to make the gear fixation as compact as possible as well as to rotate with no issues thus we used the mechanism that was used by SMORES [1]

3.1.3.2 Hubs

The hub is the key to our design its used to support the gears as well as connect the outer case with the inner case. the hub is 3D printed PLA structure, the hexagonal openings are for the nuts to settle in. The outer diameter is where the external gear will rotate on and the internal diameter for the connector to rotate on it.

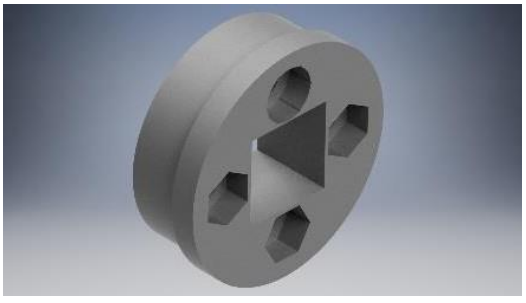


Figure 3.1.1 Hubs

The 2nd determinant factor in our design is the cylindrical bush. The dilemma is we had

two gear trains each supporting different movements, this means that the gears should rotate independently from one another that is why we used the connector an acrylic 3 mm thickness with also nuts spacing, the internal gear rotates on the connector.

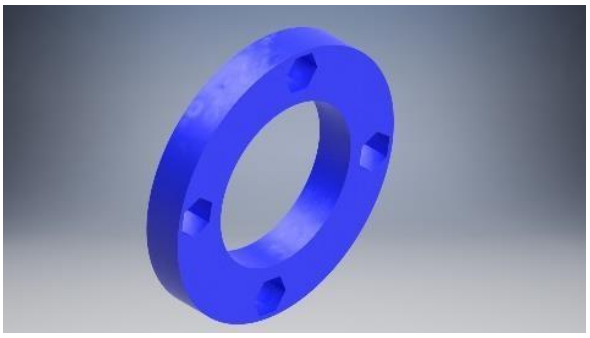


Figure 3.1.2 Bush

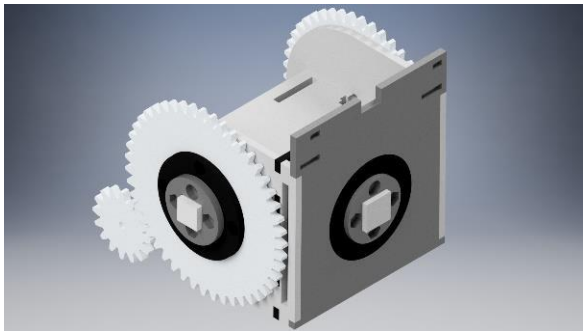


Figure 3.1.3 Inner Case Assembly

3.1.3.3 3 Wheels (3 active connectors)

The wheels were designed to fulfil two main functions:

- The motion of the robot, the wheels are responsible for the mobility of the robot in all direction in the 2D plane.
- The docking, each wheel has 4 permeant magnets that are used to attach to the other modules.

The wheels were designed as 3 mm thickness with an opening for the magnet fixations

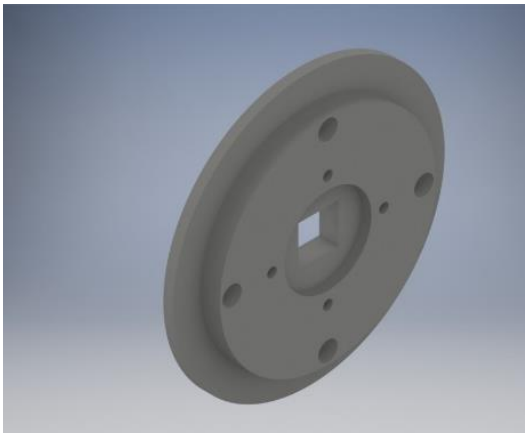


Figure 3.1.4 Front Wheel

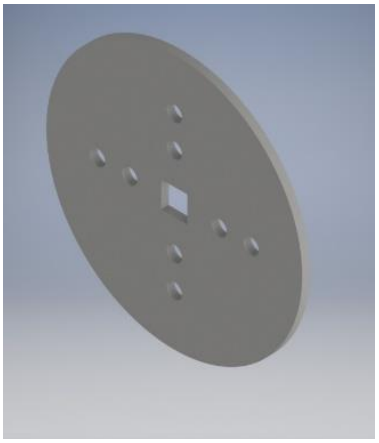


Figure 3.1.5 Side Wheels

3.1.3.4 U-Shaped motor

The 4 motors in the back were placed in the back and they were fixed by the following brackets.

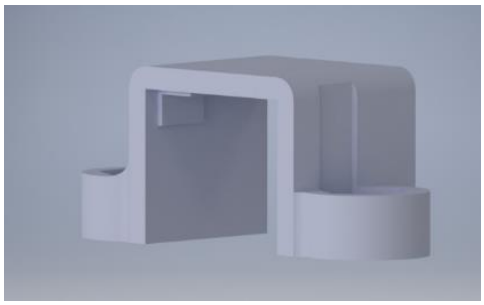


Figure 3.1.6 Motor Bracket

3.1.4 Sub-assemblies

3.1.4.1 Outer Case

The outer case has two primary function

To support the motors for the actuation, the 4 motors that are responsible for moving the robot are fixed on the back of the outer case of the robot. 4 U shaped 3D printed brackets were used to fix the motors with M3 bolts and nuts

To support the inner case, the inner case is integrated within the outer case The back of the outer case represents the passive connector

The design of the outer case was simple 2 faces connected to the Back using T slots connections

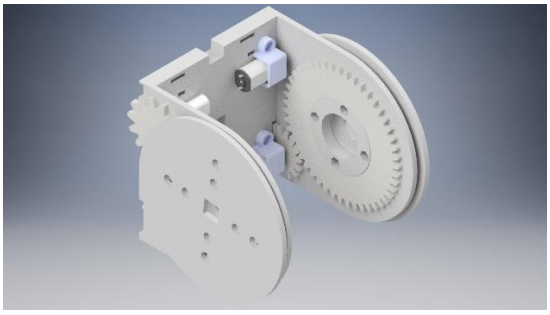


Figure 3.1.7 Outer Case Assembled Front

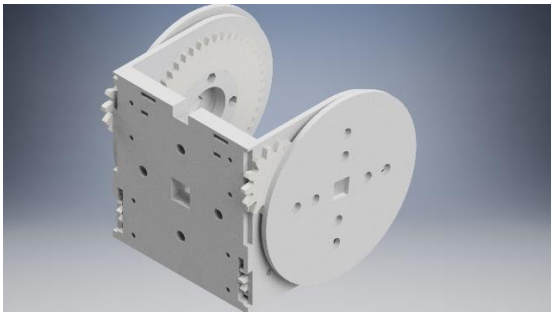


Figure 3.1.8 Outer Case Assembled Backside

The back of the outer case is thicker than the rest of the parts because we took into consideration the magnet and its interference with an encoder attached to the motor. The slot in the middle is for the IR sensors and the slots in the left and right respectively are for the TOF sensors

3.1.4.2 Inner case

Ergonomics of the module:

Since one of our main targets that the module should be as compact as possible we had to provide a place for the PCB and the wiring , this required to do a shelf like structure in the middle of the robots that provided adequate spacing for the electronics, where the battery was placed on one of the sides while the other side was used for the PCB attachment .

Figure 3.1.9 Inner Case Back View

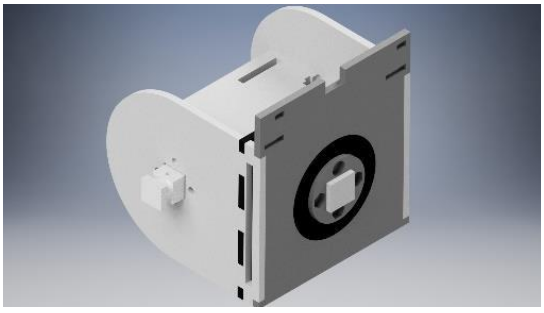


Figure 3.1.10 Inner Case Front View

The design of the inner case walls depended on T-slots connection, and the shelves was placed between the walls. The 5th motor was placed in the middle where it was attached using the bolts in the face of the motor. The Crown gear is placed on the front between two faces two constrain its movement. The above slot is for the IR sensor and the 2 lots on the right and left respectively are for the ToF sensors.

3.1.4.3 Key mechanism Assembly

The key mechanism is designed based on a reciprocating design, where two keys are placed perpendicular to each other , the key design is derived using gears with ratio 2:1 with the same motors used in the drive to the wheels .

The key mechanism is made of 3D printed part:

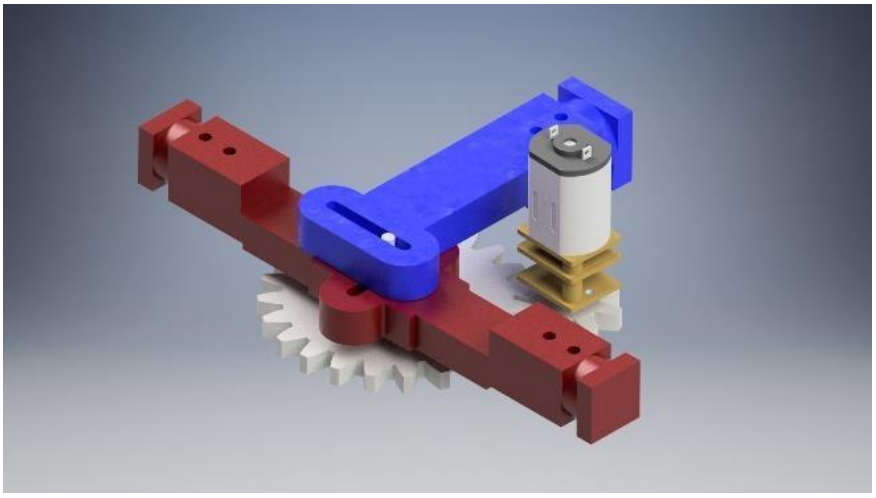


Figure 3.1.11 Latching Mechanism

3.2 Final Mechanical Assembly

This represents the final assembly of the robot as the disassembled parts of the project, showing how all the parts and the subassemblies integrated together , all the bolted connections used was M3 bolts.

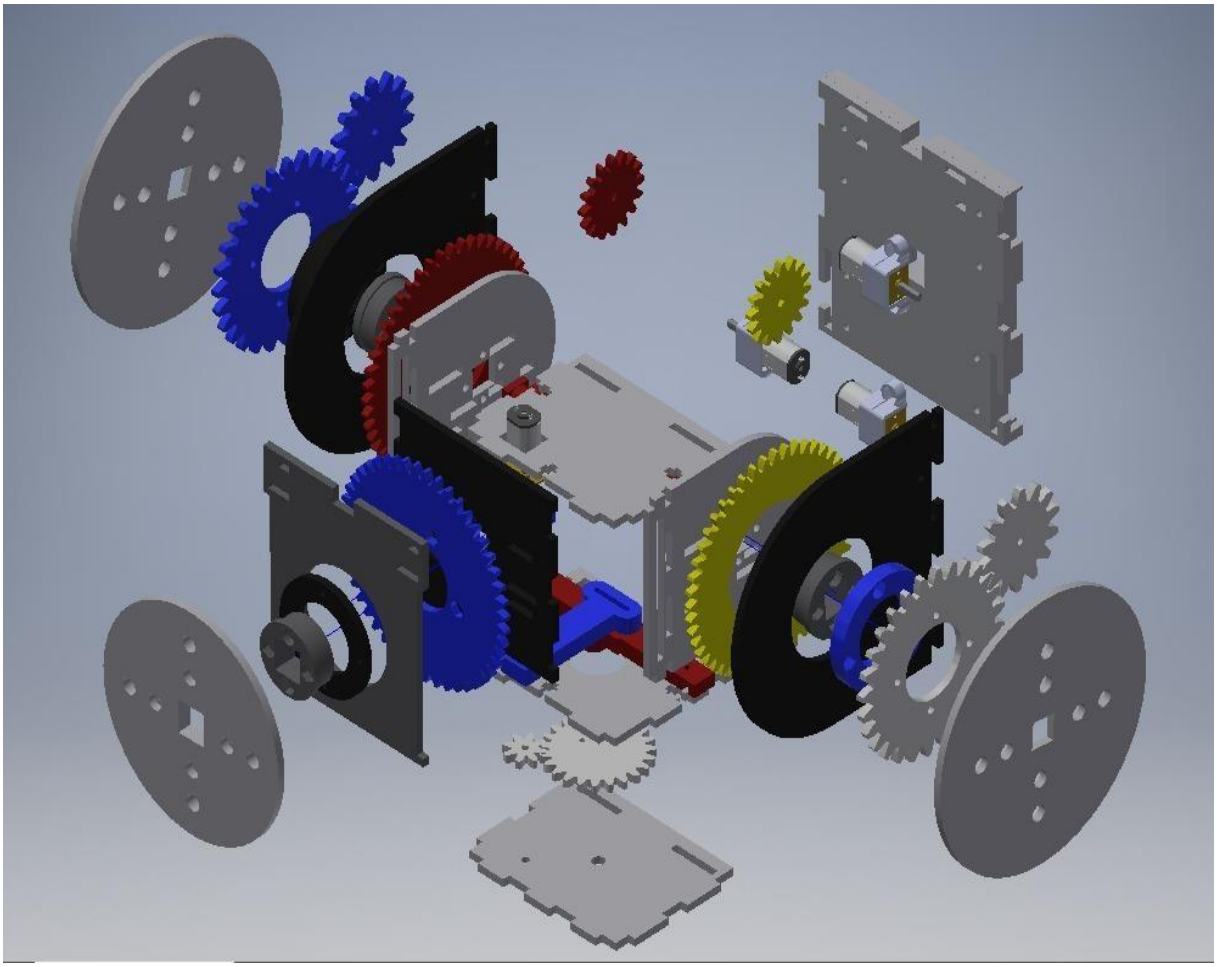


Figure 3.2.1 Assembly Exploded View

3.3 MSRR module data

SPECIFICATION	VALUE
WHEEL SPEED AT NO LOAD	30 RPM
PAN SPEED	20 RPM
TILT SPEED	20 RPM
WHEEL TORQUE	14.5 KG.CM
TILT TORQUE	32 KG.COM
MODULE WEIGHT	~700 GRAMS
MODULE OVERALL DIMENSIONS	10 CM * 10 CM * 10 CM

Table 3.3.1MSRR DATA

3.4 Actuator Sizing

In here we determined that the most appropriate motor will be using is the N20 Micro Dc gear box motor. This section is to explain what size of motor and what gear box we will use since we also determined the Nominal Voltage is 6 V.

The software we used in our actuator sizing is Solid works a specific tool actually known as motion study.

Motion studies are graphical simulations of motion for assembly models. You can incorporate visual properties such as lighting and camera perspective into a motion study.

Motion studies do not change an assembly model or its properties. They simulate and animate the motion you prescribe for a model. You can use SOLIDWORKS mates to restrict the motion of components in an assembly when you model motion.

The robot was placed on a base to create a ground contact between the wheels and the robot , we took assumptions into consideration :

- The wheel is rubber to increase the grip
- The ground is wood
- The mass of the robot with components is 1 kilogram

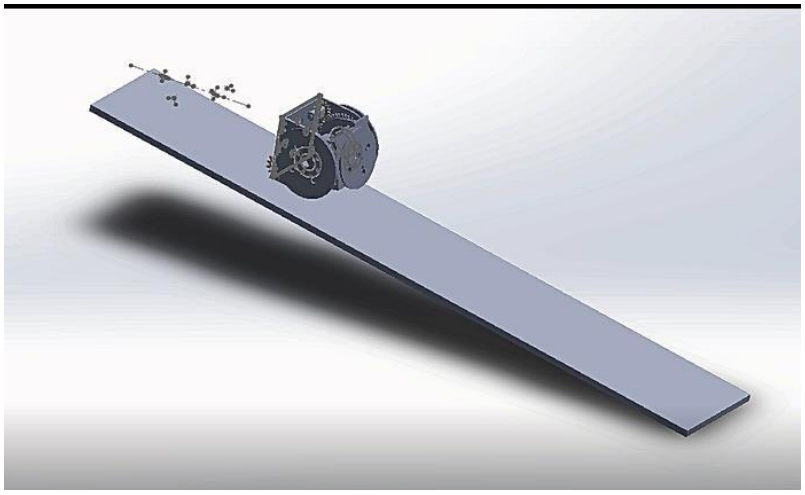


Figure 3.4.1 Simulation in Solidworks

3.4.1 Results from the motion study

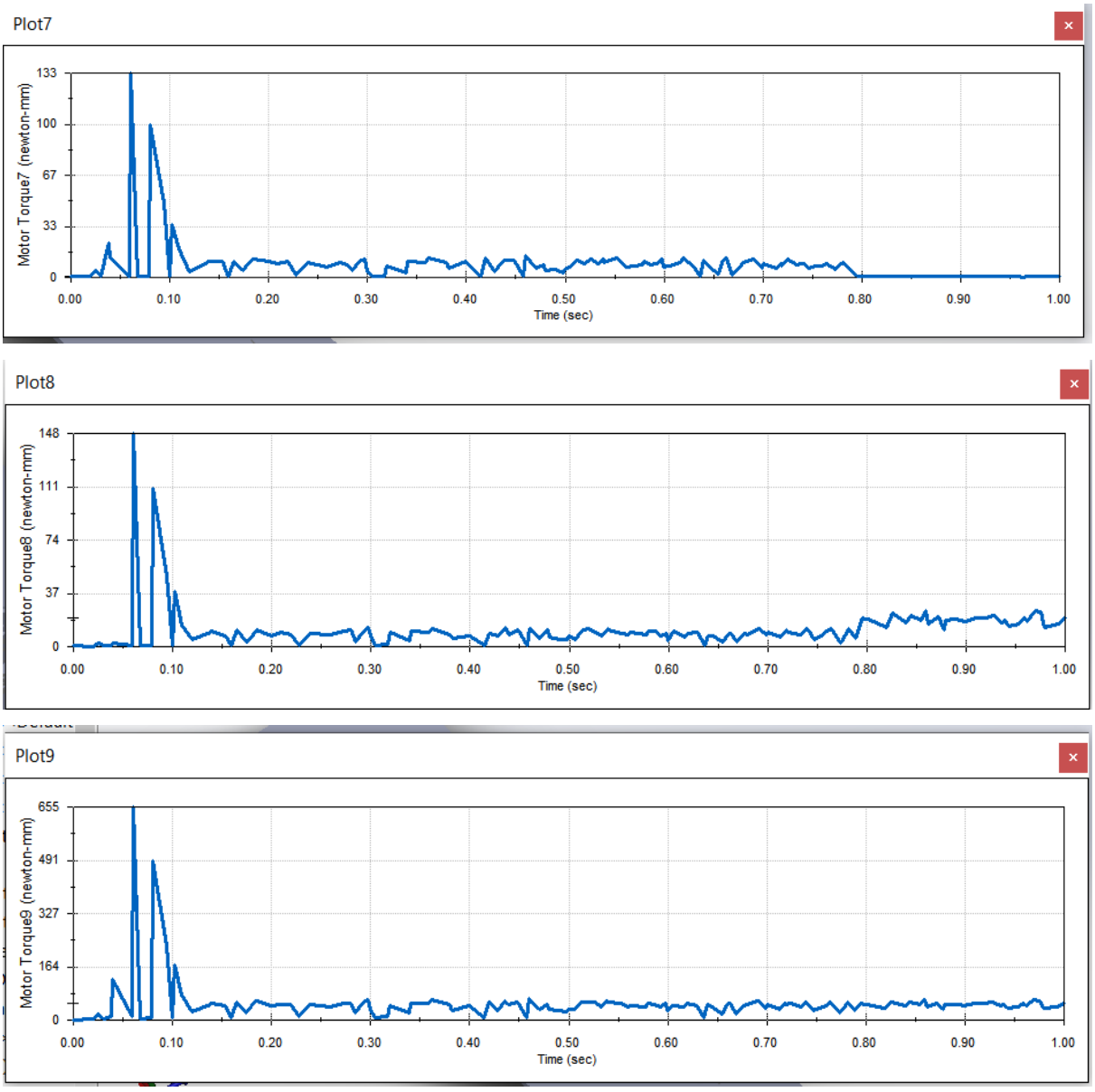


Figure 3.4.2 Front Wheel Actuator Sizing

The nominal torques calculated following:

- Right and left wheel 13 kg.cm Torque
- Tilt Torque from each operation 6.5 kg.cm

3.5 Stress Analysis

The mechanical considerations that plays a major factor in our design is the stress analysis. The stress analysis Inventor Autodesk.

A stress analysis can help you find the best design alternatives for a part or assembly. Early in design development, you can ensure that a design performs satisfactorily under expected use without breaking or deforming.

The parts that we simulated in the stress analysis is the critical parts that thought will fail.

3.5.1 Motor Fixation

The motor was tested with load of **10 N** placed on the face of the bracket The displacement was **0.00012 mm.**

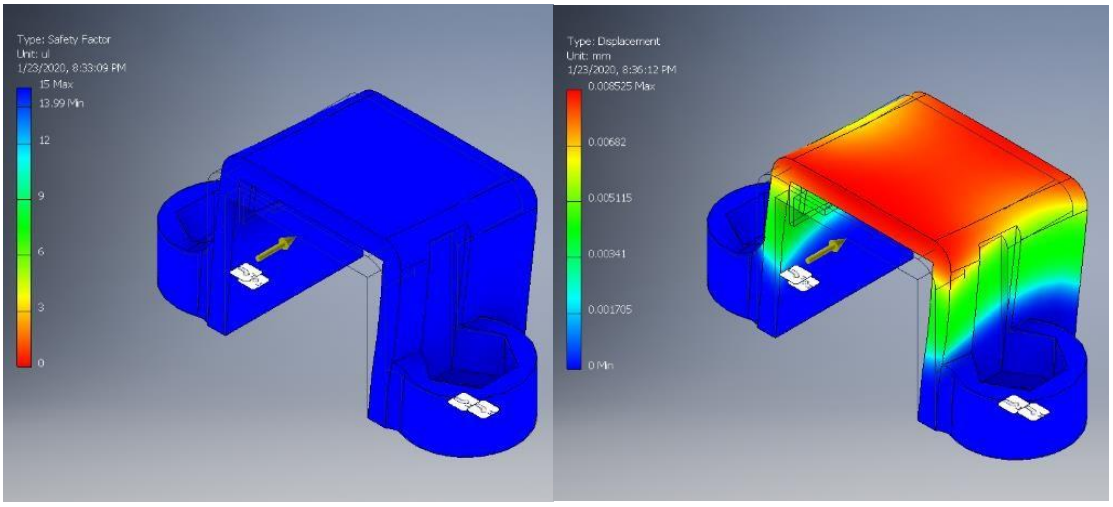


Figure 3.5.1 Motor Bracket Stress Analysis

The safety factor is **13**

3.5.2 Hubs

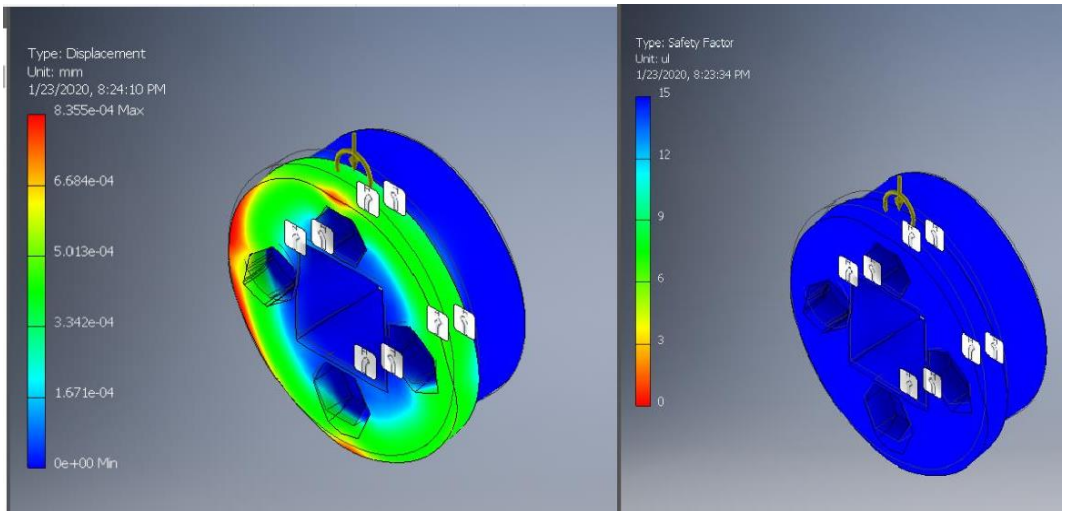


Figure 3.5.2 Hub Stress Analysis

The hub load placed was 10 N and torque placed and the Hub **10 N.mm**

The displacement was **0.000006 mmm.**

3.5.3 Material Selection

After the design was done we had a selection of materials that can be used to manufacture our robot , thus we creating a decision Matrix to select from different materials .

Each property for each material had a score of 5.

Material\ Property	Ductility	Stiffness	Density	Wear Resistance	Water resistant	Total Score
Acrylic	4	4	5	4	5	22
Aluminum	5	4	5	3	4	21
Wood	3	3	4	2	3	17
Artylon	3	4	3	4	4	19

Table 3.5.1Material selection

Based on this decision matrix we selected Acrylic will be the most suitable to manufacture our prototype model. There were other alternatives, but we compared the material according to the available materials in the Egyptian market.

CHAPTER 4: SIMULATION

The aim of the software team was to develop digital model of our module. We built a simplified model of our module. The approach we took was based on two main things, which are to make the module available for the whole team wherever they are so we don't have to have the physical ones at all times and can just test the algorithms and virtually have as many modules as we want, secondly, simulation gave us more grip on what we can and can't do with our model and made the concepts and constraints more visible.

4.1 Motivation

We needed to test the modules before building them also because most of our electronics are from abroad and take too long to be shipped. This way we can have something to try with, and if we needed to change something, we can change it quickly and easily, which makes testing with different sizes and different sensors easy. In simulation, we can have as many modules as we need without having to pay for components or wait for shipping.

Another main reason we needed a simulation was that two members of our team were going to travel abroad so in order for them to keep working on the project and not lose contact with the team completely for when they come back can catch-up easily.

4.2 Approaches and limitations

To tackle the simulation problem sub-team simulation had to test different environments to work with, we tested with different engines like Unreal(game engine), Vrep(physics engine), and Gazebo(physics engine), but we dropped Unreal engine early on as it did not meet the criteria.

We wanted to simulate our module. As can be seen under from the table is a simple comparison between Vrep and Gazebo as we tested them. A big problem that opposed us was magnetism as no physics engine has that implemented.

Points of comparison\Engines	**Vrep**	**Gazebo**
Interface	Simple GUI and easy to use	**Not very simple and harder to create and assemble models**
Core Engines	Bullet,OpenDynamics Engine(ODE),Vortex,Newton Dynamics	**Bullet,ODE,Dynamic Animation and Robotics Toolkit (DART),Simbody**
ROS support	Has plugins to support with ROS(Robot Operating System)	**The main physics engine developed and maintained by same company that maintains ROS**
Environment	The environment looks nicer and more alive	**The environment looks little dim**
Documentation and Support	**Very well documented but for it's own use not with ROS also has small community so less help when needed**	**Very well documented as standalone and interaction between it and ROS also has big community online**

Table 4.2.1 Comparison between Vrep and Gazebo

Physics engines are not perfect in fact they are far from that some bizarre things happen during simulation that can't be understood maybe some gravity errors, or a spawned model would just shake in its place without giving it orders while it should be static instead. Below we represent a simple example of one of the many behaviours that we can't describe as why or how this happens, in this picture a model has pushed this box and the box started levitating and then we retreated and left the box but that didn't even matter the box stayed floating from the ground like there is no gravity.

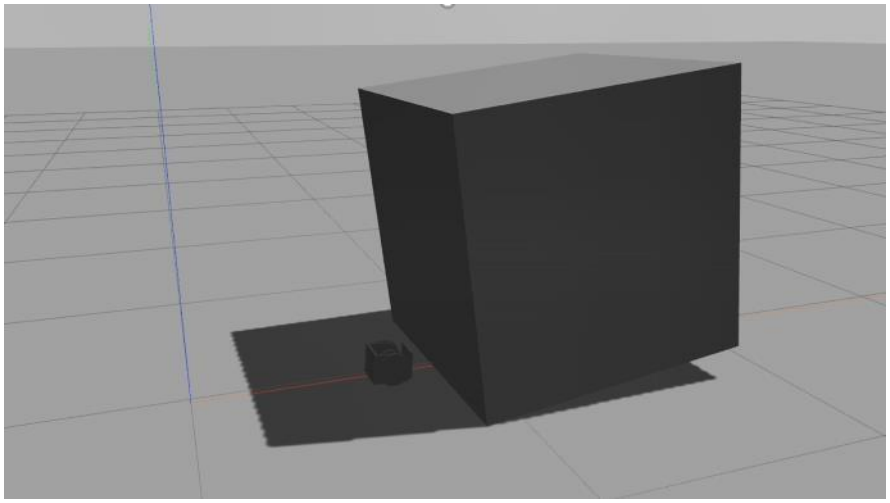


Figure 4.2.1 Unit Box Levitating

One of the limitations also was the type of file to use for our model, Gazebo accepts two types URDF(unified robot description format) and SDF(simulation description format) but even the URDF file is converted internally during the spawn of the model in Gazebo but after comparing the two files and for flexibility reasons we settled upon SDF file format is that it was more friendly when working with Gazebo and also because for some reasons when adding our plugins to the URDF file sometimes they won't work and other times they would just fine a comparison has been made between both with pros and cons of each and we might revert to URDF again in the future although SDF has more potential and more power than URDF but the later has more support in ROS, also SDF has no support in Rviz which is another engine just for visualization making visualizing the sensor data better and easier ,another downside of the URDF is that it can be used for specifying only the kinematic and dynamic properties of a single robot and can not specify the pose of the robot itself within a world. It is also not a "universal" description format since it cannot specify joint loops, and it lacks friction and other properties, in the next table we present a comparison between both types.

Points of comparison\File types	URDF	SDF
Simplicity	Very simple	Complex
Rviz support	Yes	No
ROS support	Yes	Yes, but not completely
Plugins support	Yes, but sometimes have unexpected results	Yes
Description of the robot and the environment	Can describe only the robot and is very simple does not contain many definitions	Contains descriptions to most of the aspects of both the robot and environment

Table 4.2.2 comparison between URDF and SDF

4.3 Our work

We tested with a simplified model of our module which consisted mainly of differential drive wheels and a revolute joint that lifted the front face up and down to help the modules form the 3d lattice we needed and giving it more degrees of freedom and less constrains. A huge problem that faces us was magnetism as the mechanism to join the modules was simple magnets and de-latching mechanism that consisted of linear arm that pushes the modules from each other to disconnect them, magnetism is not implemented in any of the physics engines so we had to cheat the engine with a hand-written script that joins the modules' faces together when they come near each other.

As of this moment we can control each module individually we can control it's movement and orientation also we can control the lifting mechanism, we still need to improve the magnetism script also we are working on a GUI (graphical user interface) to improve the UI.

We tested with a model generated from SolidWorks, for a more realistic model, but we had many difficulties in working with it, as generating the model is not a feature from SolidWorks, but a plugin, and that plugin was not totally accurate, so some of the model's features were lost,

and that made the simulation worse so we had to ditch it, and use the simplified one instead, but we are still testing with the plugin.

4.4 Simplified model of our module

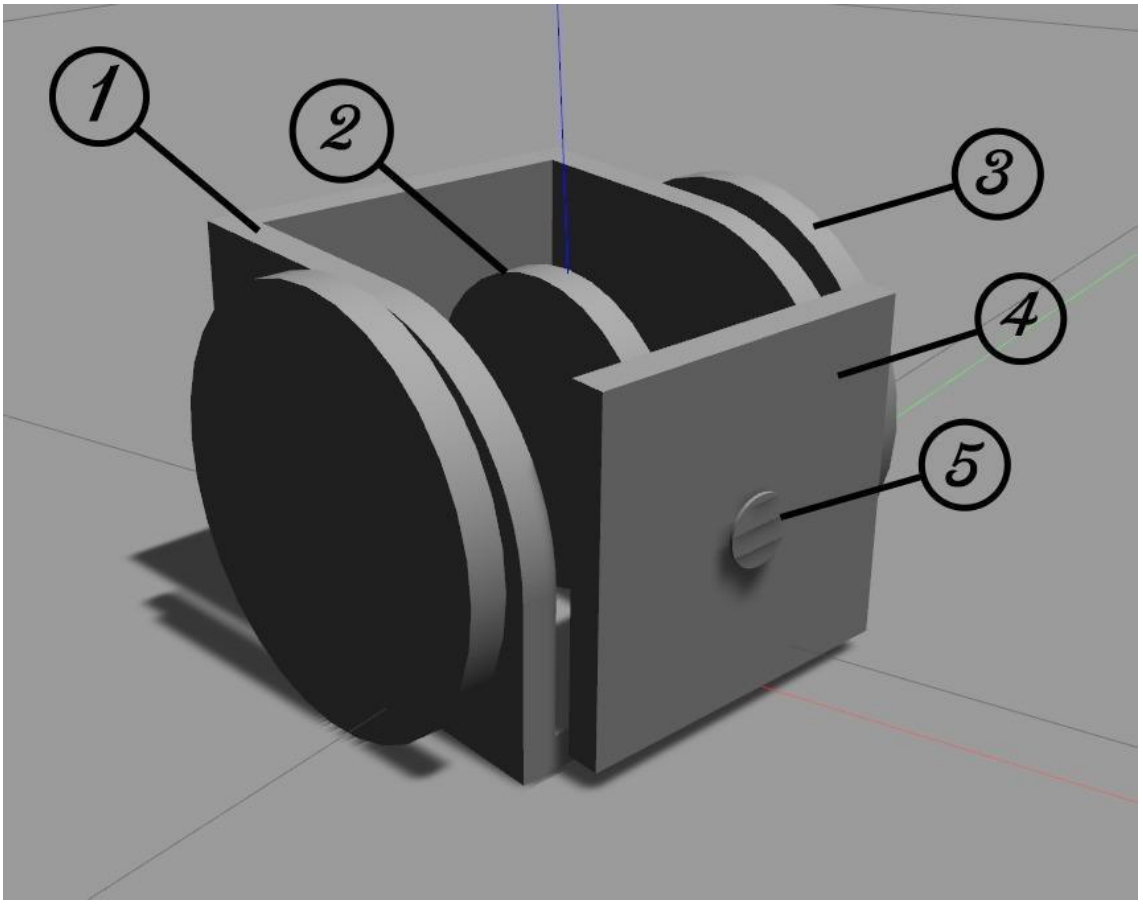


Figure 4.4.1 Simplified Version of our Module

1. The main body of the robot
2. The tilting mechanism
3. The wheels
4. Front face
5. Front face magnet

There is another magnet at the back of the body other magnets to be added to the side wheels for full 3d lattice achievement, also it is too simplified masses and inertias to be adjusted and pan mechanism to be added in order to achieve more realistic motion. Material is not specified as we are constantly changing it until we find the optimum one and changing its friction and the ground friction to match the terrain it is navigating through.

4.4.1 Generated model

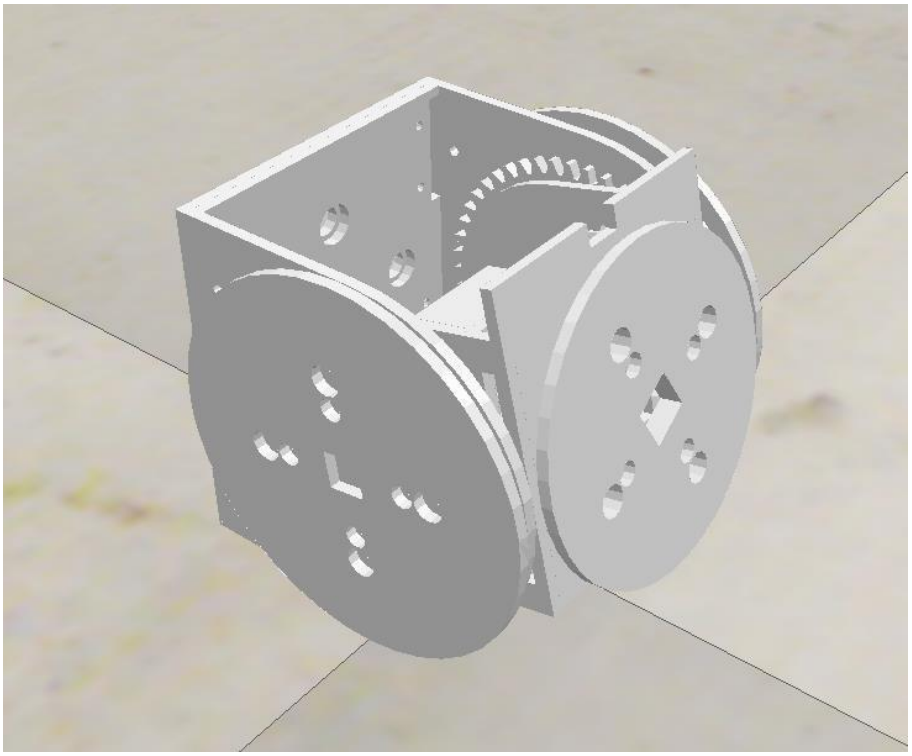


Figure 4.4.2 Generated Version of our Module

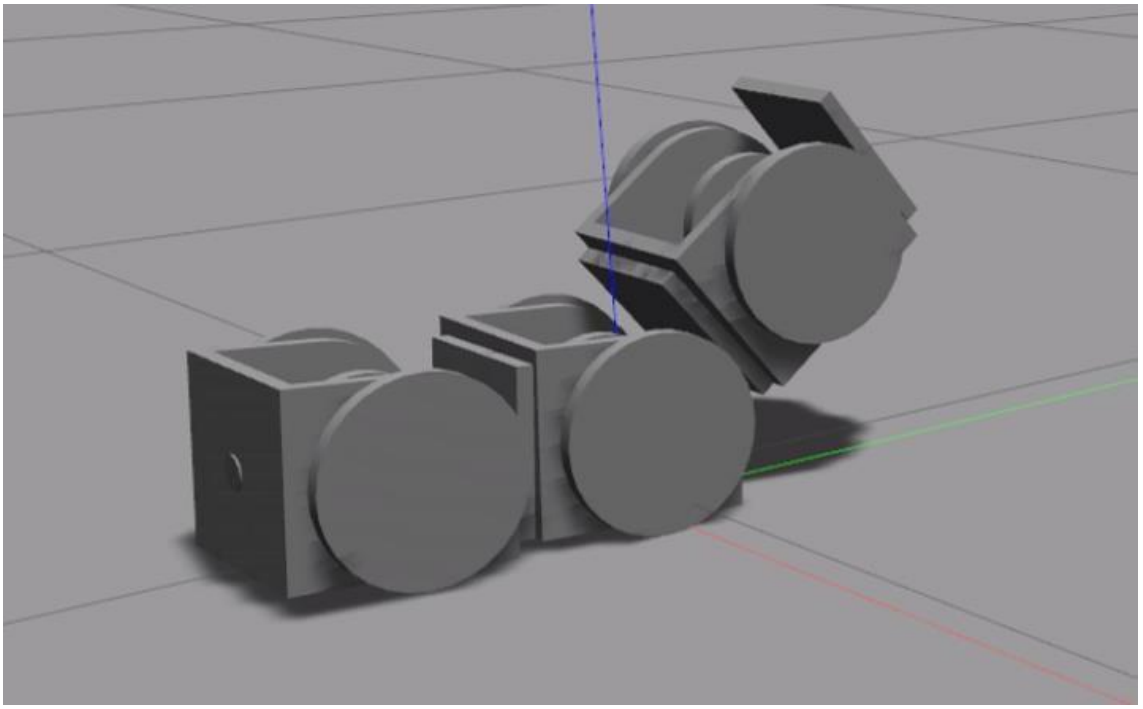


Figure 4.4.3 Human Controlled Modules with Magnetism On

4.4.2 What you need to know to get the simulation working

The codes are divided in two main packages gp_abstract_sim and simple_keys. Terminal Commands:

1. roslaunch gp_abstract_sim start_testworld.launch
2. roslaunch gp_abstract_sim spawn_sdf.launch robot_name:="your robot name"
3. rosrun simple_keys readwrite_keys
4. rosrun simple_keys readStringAndCut
5. **roslaunch gp_abstract_sim start_testworld.launch**. This command is for starting an empty Gazebo world. This node takes no arguments.
6. **roslaunch gp_abstract_sim spawn_sdf.launch robot_name**:="your robot name" This launch file is for spawning the modules into the world that you started from command no.1 this launch file takes the following arguments:
    - robot_name: for naming the robot in the simulation (no default you must specify)
    - x: Position x of the robot center by reference to world (default 0)
    - y: Position y of the robot center by reference to world (default 0)
    - z: Position z of the robot center by reference to world (default 0)
    - roll: Rotation angle x in reference to world frame (default 0)
    - pitch: Rotation angle y in reference to world frame (default 0)
    - yaw: Rotation angle z in reference to world frame (default 0)
    - sdf_robot_file: The path to where the SDF model is stored on your computer (default is set to the path on our test computer)

rosrun simple_keys readwrite_keys. This node runs in the terminal and controls the modules by the keyboard but you must not leave the terminal (the terminal running this node must not lose focus) or the keyboard keys won't control the module. The following keys are the only keys that work:

- W: move forward
- S: stop the module

- X: move backward
- A: rotate left
- D: rotate right
- Q: forward left
- E: forward right
- Z: backward left
- C: backward right
- U: rotates the tilting mechanism up
- J: rotates the tilting mechanism down
- M: stops the tilting mechanism

1. **rosrun simple_keys readStringAndCut.** This node takes the contact sensors readings and sends signals to the modules to connect them together through topics. This node takes no arguments.

The Plugins used in the model are all written from the ground up to match our model's special specifications as this model has some very special features such as magnetism which is not implemented in any of the physics engines used by Gazebo and these plugins are called internally and loaded with the spawning of the model in the world.

The plugins are:

1. my_speed_controller_plugin
2. ContactPlugin

my_speed_controller_plugin. This is the main plugin in the module which contains subscription nodes that does the following:

- controls the motion
- controls which face attaches to which other module's face
- controls the tilting mechanism and this happens through four subscription nodes which are:

    1. module_name/right_wheel_speed as the name implies controls the speed of right wheel (arguments are in float)
    2. module_name/left_wheel_speed as the name implies controls the speed of the left wheel (arguments are in float)
    3. module_name/front_face_speed this topic controls the front face tilting movement (arguments are in float)
    4. module_name/attach takes the name of the two sides to be linked (arguments in String)

Contact Plugin. This plugin is connected to the magnets on the model it also starts with the summoning of the model this plugin is connected to a contact sensor on the faces of the model when this sensor collides with anything in the world it checks what that object is if that object is other module's collision sensor and this sensor is a small enough in order to make accurate docking. This plugin creates the sensor_data node which sends a string with the names of the collided links which is received by the readStringAndCut node to process it and send to the my_speed_controller_plugin node to connect them if accepted as attachable nodes.

4.4.3 Software and Programs used:

OS→ Ubuntu 18.04 LTS

Languages→ C++, ROS distro Melodic IDE→ Visual studio code

plugins: 1- C++ version 0.26.3 by Microsoft 2- ROS version 0.6.2 by Microsoft

Physics Engine→ Gazebo version 9.10

→Vrep version 3.6

GUI Environment and Libraries → QT creator Code packages→ gp_abstract_sim, simple_keys

4.5 Robot's Design Modification

4.5.1 Previous Model

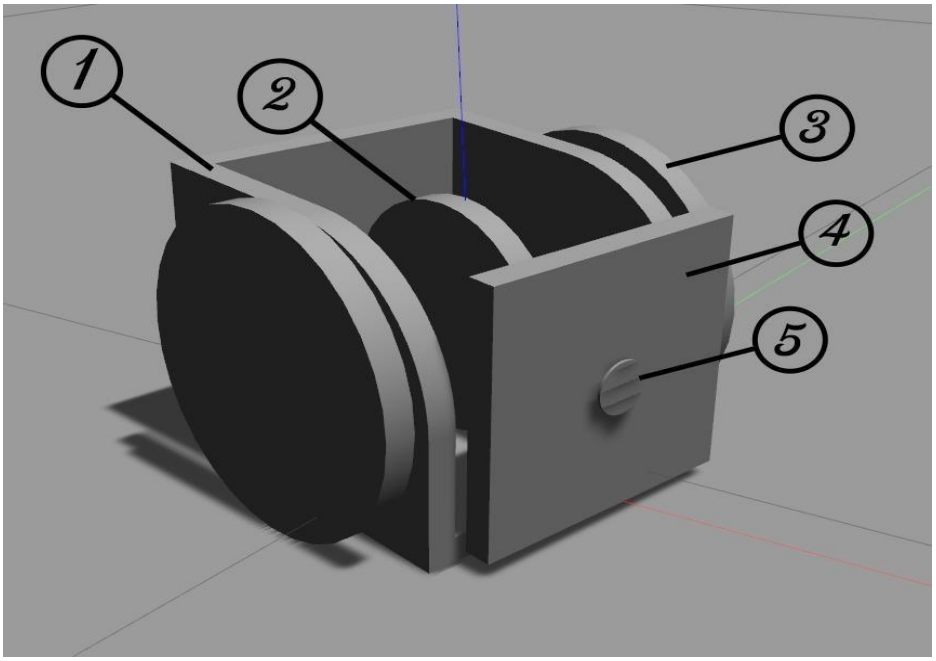


Figure 4.5.1 Previous Model

4.5.1.1 Robot's Components:

1) The main body of the robot
2) The tilting mechanism
3) The wheels
4) Front face
5) Front face magnet

4.5.2 Present Model

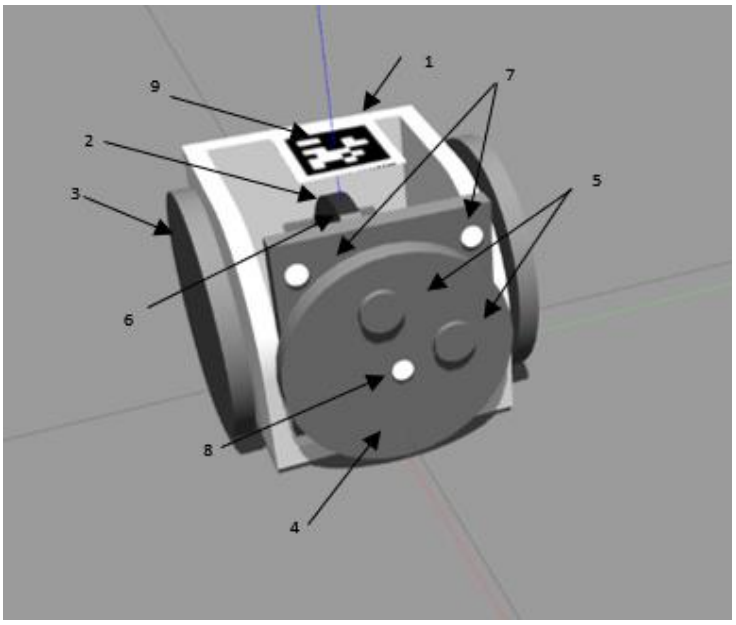


Figure 4.5.2 Present Model

4.5.2.1 Robot's Components:

1) The main body of the robot
2) The tilting mechanism
3) The wheels
4) Front face
5) Front face magnets (South – North)
6) The rotating mechanism
7) ToF sensors
8) IR sensors
9) Apriltag (ID: 0)

As shown, there were some modifications to enhance the robot performance.

4.5.2.2 Modifications

4.5.2.2.1 Front face:

The front face became round instead of square, so when the face rotates it does not hit the floor. Instead of one magnet in the middle of each face (Front face – Back), 2 magnets were added in each face to ensure the docking process.

4.5.2.2.2 Adding sensors

In each face 2 ToF sensors (Time of Flight) have been added. In each face an IR sensor (Infrared) has been added.

4.5.2.2.3 Adding caster wheels

Two caster wheels have been added to the robot, to enhance the stability of the robot.

4.5.2.2.4 Apriltag

They are placed on top of the robot. Apriltags are detected by the camera, to determine the position and orientation of each robot.

4.5.2.3 Other features

The robot front face tilt from 0 degree to 90 degree, and rotate CW and CCW.

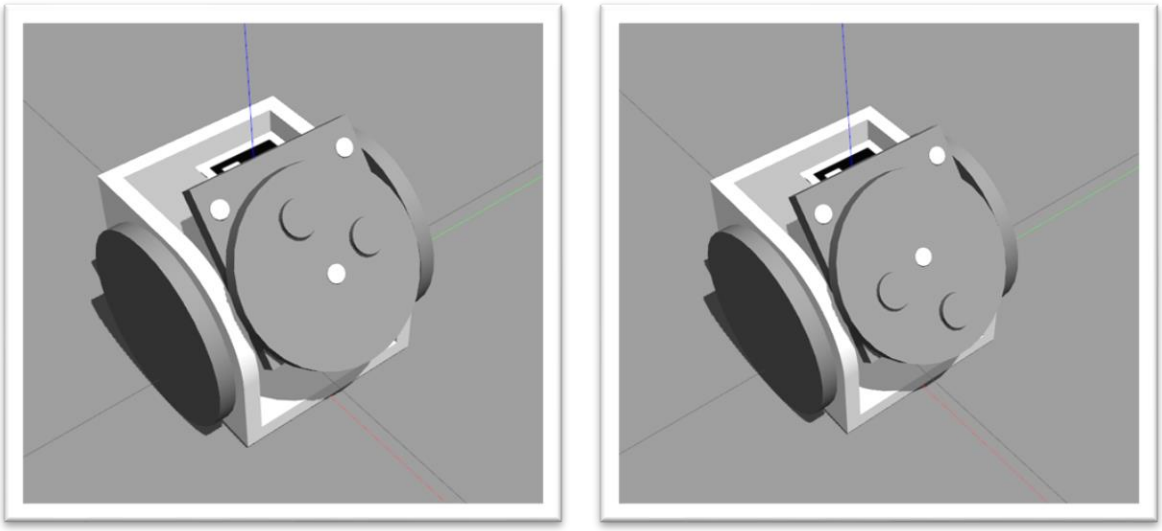


Figure 4.5.3 Robot Face Tilt

4.6 Robots' Localization

For a robot to navigate its environment safely, first, it must know its position in this environment. Moreover, in our case, it is not single robot that we are trying to manipulate; therefore, we need to know the exact position and orientation of each robot in the field.

4.6.1 Problem Breakdown

There are some requirements and constraints imposed on the system. We will mention both below.

Requirements:

- It is vital to the system that the pose of each robot is measured with high accuracy since the robots are going to be joined to create the desired formation.
- All the robots' poses must always be readily available with considerable updating frequency.
- We also need to be able to identify each module. Every module should have a unique ID.

Constraints:

- The space inside each robot is quite small. The sensor must not take big space.

4.6.2 Various Techniques for Localization

We will discuss various methods here that we have thought of and found which should help us achieve our target. All the methods mentioned below utilize a camera, since it would be better if we did not rely on an internally mounted sensor inside the robot.

4.6.2.1 Color Detection and Tracking

This method focuses on tracking certain color, as the title suggests. For example, consider that we want to track a ball, a red ball. The algorithm will go as follows:

- Acquire the image, or video, from the desired source.
- In a similar sized, but empty, matrix, give any pixel that is not red, a black color.
- And give any red pixel, a white color.
- After a full scan, the outcome will be a binary image, with only the red ball visible.
- Appling some calculations, the center of this white ball can be determined.

We have noticed some things that will make the utilization of this technique hard. First of all, it is highly dependent on the lighting and colors. In a controlled environment, it would not be hard to control the lights, but elsewhere will be a problem. Secondly, the robot's exterior is made from acrylic, which is hard to paint and reflects light. This would result in some parts of robot to be disregarded, making the center of the robot shift a lot. Also, all the robots will be detected and tracked, but there will be no way to ID each one of them uniquely, or to determine their direction (orientation.)

4.6.2.2 Contour Detection

This is the simplest method that came to our minds. We capture the feed from the camera, then we apply a contour detection algorithm to identify the robots. This idea was dismissed immediately because of its inconsistency in the detections, and there is no way to identify the robots uniquely, or their orientation.

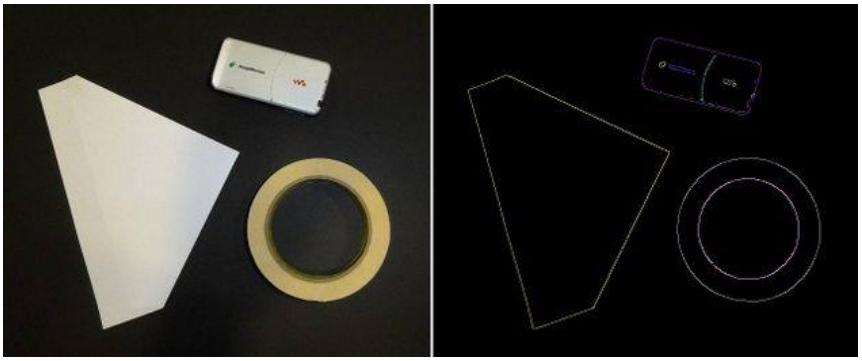


Figure 4.6.1 Contour Detection

4.6.2.3 Pose Estimation, Using 3D CAD Model

We found this method to be quite interesting. It allows you to extract the 6-DOF pose of known 3D CAD objects from a single 2D image.

The algorithm takes an image and the 3D mesh of the object of interest as inputs, also the camera's intrinsic parameters. Then it extracts the ORB features and descriptors from the input image, and then uses the mesh to compute the 3D coordinates of the found features. Finally, the 3D points and the descriptors are stored in different lists.

To use this method for complicated CAD design, like ours, both the CAD model and the real-life model must be almost identical. This would be a little problematic, since our CAD model and real-life are almost identical, but our simulated version is not due to difficulties mention before. Also, the material should be well textured to be able to extract better features. Moreover, in the demonstration of this method, at 15 FPS, the estimation was relatively slow and extremely faulty, and noisy. It oscillates quickly when the object of interest is moving. This was noticed when a box was moved by hand slowly; therefore, we can deduce that if the object is constantly moving, and at higher speeds, this error would increase.

Figure 4.6.2 Pose Estimation Using 3D CAD

4.6.2.4 AprilTags

AprilTag is a visual fiducial system, useful for a wide variety of tasks including augmented reality, robotics, and camera calibration. Targets can be created from an ordinary printer, and the AprilTag detection software computes the precise 3D position, orientation, and identity of the tags relative to the camera.

AprilTags are conceptually like QR Codes, in that they are a type of two-dimensional bar code. However, they are designed to encode far smaller data payloads (between 4 and 12 bits), allowing them to be detected more robustly and from longer ranges. Further, they are designed for high localization accuracy— you can compute the precise 3D position and orientation of the AprilTag with respect to the camera.

4.6.2.5 Shape and Color Detection

This method is combination of two methods explained before, color detection and contour detection. The result from an approach like this is a unique ID for each robot consisting of two parts, the shape's name, and its color. As well as the center of these shapes in an image frame.

The process of this shape detection and analysis goes as follows:

- First, we detect the contour of the shape and find its center.
- Then, we perform a shape detection algorithm. Using the number of vertices that a shape has we can determine its type.
- Finally, we label each shape with its detected color.

However, it seemed somewhat convenient to use this method, but it also has its drawbacks. As the benefits of using colors and shapes for our project are combined, their problems are also combined.

Figure 4.6.3 Shape and Color Detection

One of the primary drawbacks to using the method presented in this post to label colors is that due to lighting conditions, along with various hues and saturations, colors rarely look like pure red, green, blue, etc. While this method works for small color sets in semi-controlled lighting conditions, it will likely not work for larger color palettes in less controlled environments.

And a major setback is that this algorithm script loops over each of the shapes individually, performs shape detection on each one, and then draws the name of the shape on the object. This means that it will have to loop multiple times for the same frame to detect all the robots in the scene. Not to mention that it also cannot measure the orientations of the robots.

4.6.3 The Selected Method: AprilTags

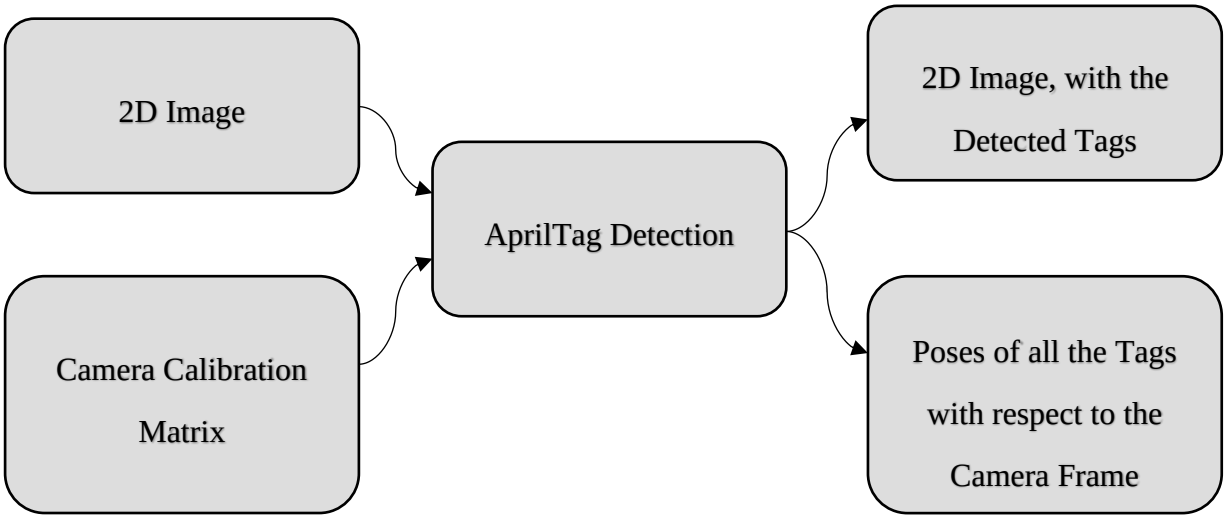
4.6.3.1 Why AprilTags?

There are many reasons why we choose to use AprilTag in localization of the robot with the camera and they are:

- Accurate localization, measuring the orientation and distance accuracy using AprilTag is high and reliable.

- High adaptability to the camera, the object carrying the tag is always localized and identified by the camera as long as the object is seen in the field of view and the camera pose is easily estimated related to the tag.
- Variation and availability, there is many categories and families from the AprilTag available everywhere and with variable sizes easy to be customized.

4.6.3.2 Inputs and Outputs



4.6.3.3 Downsides

4.6.3.4 Camera Calibration

The first step taken after deciding on the best method to be used, was to calibrate the camera itself to be able to use it. But why should we? Because of distortions.

The camera matrix does not account for lens distortion because an ideal pinhole camera does not have a lens. To accurately represent a real camera, the camera model includes the radial and tangential lens distortion.

Radial distortion occurs when light rays bend more near the edges of a lens than they do at its optical center. The smaller the lens, the greater the distortion.

Calibrating a camera in your workspace typically happens in two steps:

- Calibrate the “intrinsics”, the camera sensor & lens, using something like ROS’ camera calibration package. The intrinsic parameters are the parameters intrinsic to the camera itself, such as the focal length and lens distortion.
- Armed with the intrinsics, calibrate the “extrinsics”, or the pose of the camera in your workcell. Extrinsic parameters are those that describe the pose (i.e., position and orientation) of a sensor with respect to an external frame of reference.

Of course, we only need to calibrate the intrinsics only since we are using the camera sensor for pose estimation. The AprilTag algorithm uses only a 2D images for pose estimation.

The laptop's camera was used for initial testing. These types of cameras do not require calibration.

4.6.3.5 Utilizing AprilTag-ROS Library: Initial Testing

Now that we have calibrated the camera, time to test the algorithm for ourselves.

The apriltag_ros package is a ROS package of the AprilTag 3 visual detection algorithm. Provides full access to the core AprilTag 3 algorithm's customizations and makes the tag detection image and detected tags' poses available over ROS topics.

The package is fully defined by the two configuration files config/tags.yaml (which defines the tags and tag bundles to look for) and config/settings.yaml defines the settings of the tag, it includes:

AprilTag family used: (36h11 family type is used in our model).

AprilTag list of IDs that could be detected: (0 to 8 for 8 models in our case).

Size of each AprilTag ID: which is the real size of the tag which is the black are of the tag which it contains the code (4x4 cm tag in our case).

The border of the tag: Width of the tag outer (circumference) black border (1 border in our case).

This package contains many topics and nodes we used the most important node (apriltag_ros), a detector for a continuous image which is fit our case for continuous dynamic detection. The following figure represents the main input and outputs of the algorithm.

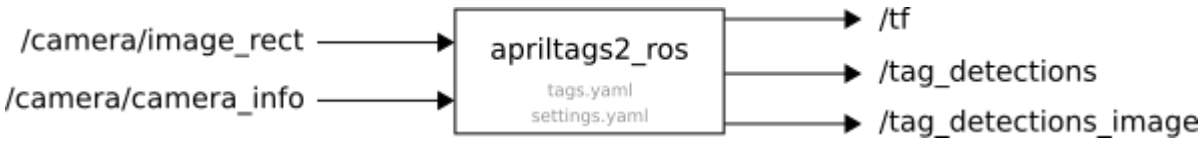


Figure 4.6.4 Apriltags Inputs and Outputs

- **/camera/image_rect:** A sensor_msgs/Image topic which contains the image (e.g. a frame of a video stream coming from a camera).
- **/camera/camera_info:** A sensor_msgs/CameraInfo topic which contains the camera calibration matrix.

- **tag_detections (apriltag_ros/AprilTagDetectionArray):** Stores the Array of tag detections' pose relative to the camera frame.
- **/tf:** relative pose between the camera frame and each detected tag's or tag bundle's frame (specified in tags.yaml) using tf.
- **/tag_detections_image**: the same image as input by /camera/image_rect but with the detected tags highlighted.

The laptop camera detects AprilTag id signs successfully, that are placed in front of the laptop, hanged in wall at a first trial.

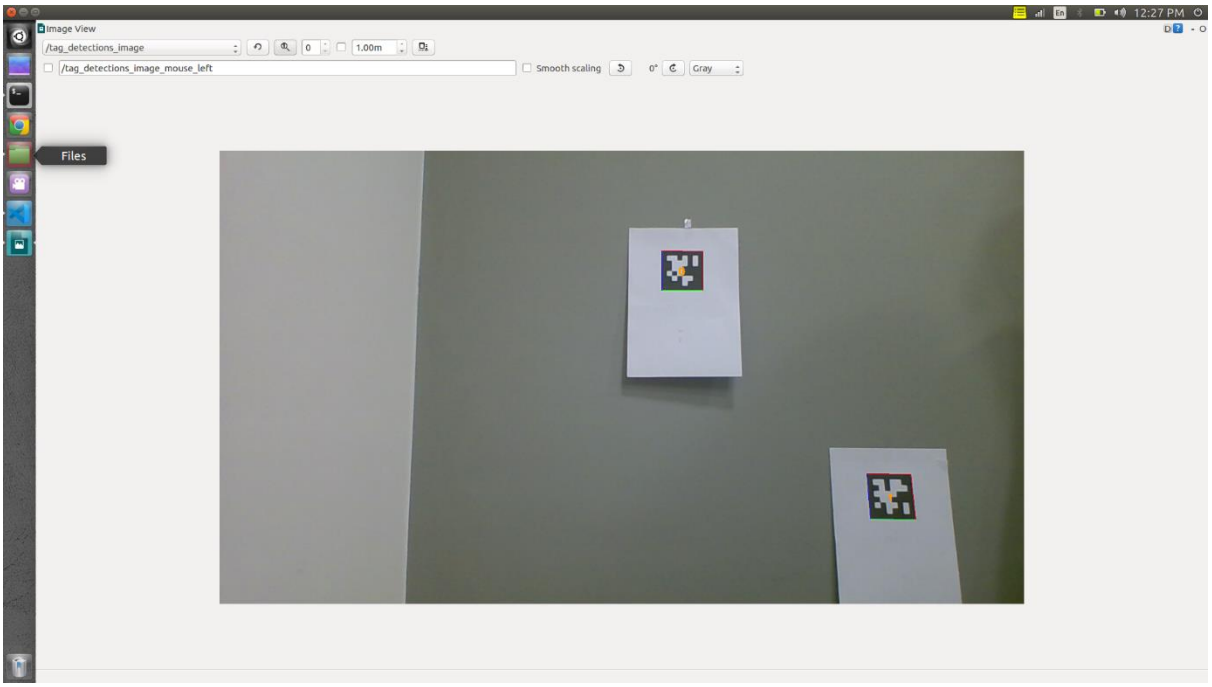


Figure 4.6.5 AprilTag Detection Using Laptop Camera

4.6.3.6 Switching Cameras

After using the laptop's camera for initial testing, we needed to switch to a camera that is moveable, unlike the laptop's camera, so we can fixate it in our environment in the future.

We needed a camera with information and datasheet that are available and reliable. The parameters that are essential are those:

- Frame rate
- Horizontal field of view
- Resolution
- Focal length in pixels
- Distortion Coefficients, this requires calibrating the Kinect camera before using it.

So, repeating the same test as before to check if all the packages and libraries are working correctly together, we plugged in the Kinect and with a couple of tags on the wall. This time,

we also visualized all the frames. We used Rviz. It shows the camera frame and all the detected tags' frames.

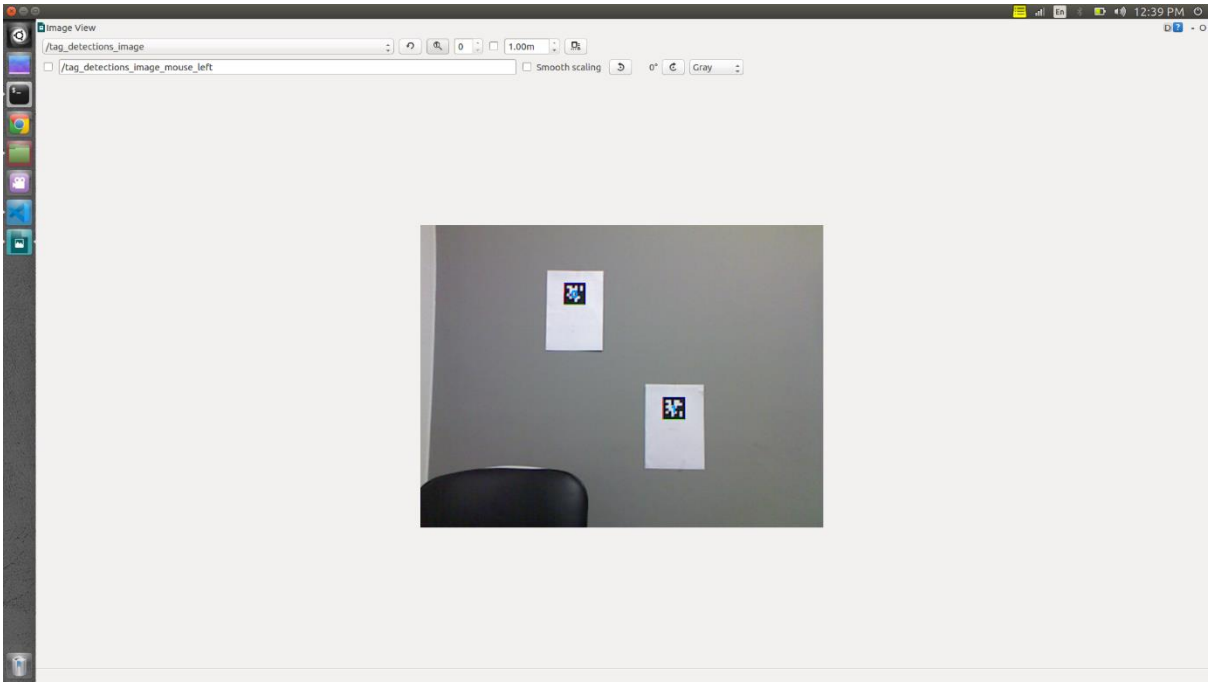


Figure 4.6.6 AprilTag Detection Using Kinect Camera

Using Rviz for visualization really shows the benefits of using AprilTags. It shows all the frames with their transformations with respect to the camera frame. It also shows the amount of noise in the readings.

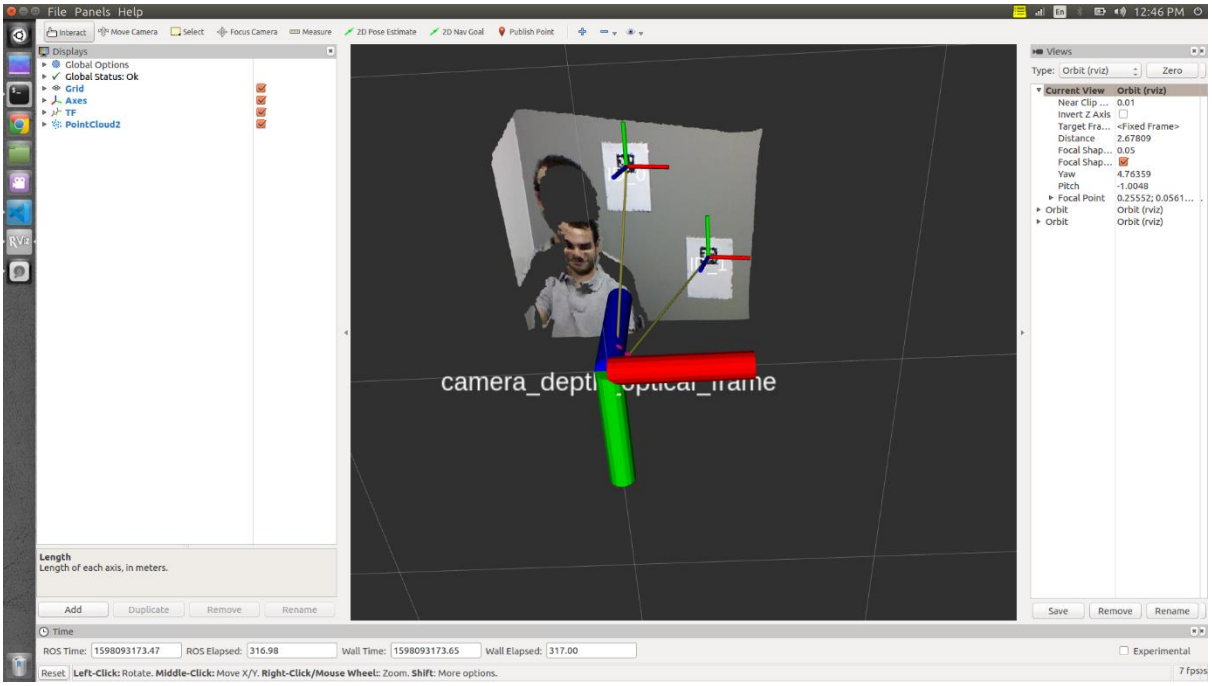


Figure 4.6.7 AprilTag Detection, RVIZ Visualization

4.6.3.7 Accuracy Testing

Now that we have tested the algorithm and works fine, but is it accurate? We are not sure if it is even close to the level of accuracy that we need; therefore, we needed a method to make sure that we are moving in the correct path. Two steps were taken to check the accuracy of our current setup:

- Accuracy Checking C++ code: It gathers a huge amount of readings from the camera on the detected tags. It then calculates the average of these readings along with the maximum and minimum reading acquired. Compared with the real distance that the tag is placed from the camera frame, we can see how accurate it really is.
- A testing rig: It is a mount for the Kinect, along with multiple tags placed in front of it in multiple positions and orientations. It was created so we can be able to calculate the true poses of the tags. It can be calculated from the CAD simulated environment.

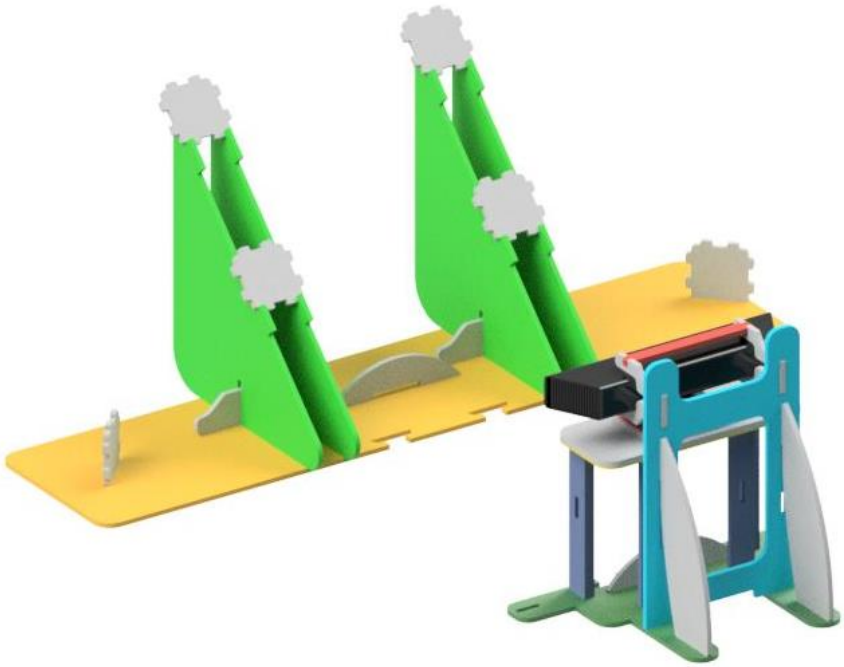


Figure 4.6.8 Kinect Test Rig

By this time, all the workshops and manufacturing facilities were closed due to the curfew caused by the Covid-19 virus spreading. This meant that we had to change our workflow from testing in real life to testing in a simulated environment. It took some time to transfer our setup into the environment. The Kinect models supplied by Gazebo model library had two problems:

- The Kinect camera frame was shifted from the actual frame of the camera.
- The physics engine was messed up. The Kinect always got a bit buried under the ground. Also, if it was placed on any other object, it pushes it down into the ground.

Also, the there are no AprilTag models that are ready online, so we had to create our own from scratch.

After all the simulated environment was ready and set up, we started testing by simply placing the camera at a ceiling mount looking directly down on the floor, with a tag that is flat on the ground. The results were incredibly inaccurate. It was easily noticeable in the Z direction, also in the X and Y directions. This also means that the orientations are probably faulty as well.

#	Tag Size (m)	Actual Height(m)	Read Height(m)	Error in Height(m)
1	0.04	**1.75**	**1.796112636**	**0.046112636**
2	0.04	**1.8**	**1.864518745**	**0.064518745**
3	0.04	**1.85**	**1.898473349**	**0.048473349**
4	0.04	**1.9**	**1.915818248**	**0.015818248**
5	0.04	**1.95**	**2.01067048**	**0.06067048**
6	0.04	**2**	**2.084829154**	**0.084829154**
7	0.04	**2.05**	**2.090199806**	**0.040199806**
8	0.04	**2.1**	**2.154255784**	**0.054255784**
9	0.04	**2.15**	**2.272816303**	**0.122816303**
10	0.04	**2.2**	**2.308974957**	**0.108974957**
11	0.04	**2.25**	**2.337361408**	**0.087361408**

As shown above, at different heights, in the Z-axis, there are huge errors.

The smallest error is: **0.015818248, while the largest is: 0.122816303.**

4.6.3.8 Accuracy Improvement

These errors are completely unacceptable, so we needed to find what is the source of these errors. We did some experiments to try and understand the cause of these issues.

4.6.3.8.1 Errors Due to Wrong Configuration Parameters

The first test was to check if the 'Tag Size', defined in the parameters file, is actually 4 cm. If not, it may be the cause of the errors.

All the units are in meters.

So, at a constant height, we will place the camera and read the tag's Z position.

Tag Size (m)	Actual Height (m)	Read Height (m)	Expected Tag size (m)	Error (m)
0.0395	**2**	**2.057751169**	0.038391425	0.057751169
0.03945	**2**	**2.064616815**	0.038215324	0.064616815
0.03955	**2**	**2.069850317**	0.038215324	0.069850317
0.04	**2**	**2.093401079**	0.038215324	0.093401079

0.0405	**2**	**2.119568592**	0.038215324	0.119568592
0.038	**2**	**1.988731025**	0.038215324	0.011268975
0.0387	**2**	**2.025365544**	0.038215324	0.025365544
0.0384	**2**	**2.009665036**	0.038215324	0.009665036
0.03845	**2**	**2.012281788**	0.038215324	0.012281788
0.03835	**2**	**2.007048285**	0.038215324	0.007048285
0.038215324	**2**	**2.000000002**	0.038215324	2.27E-09

In this last, yellow highlighted, row, we used the most trending 'Expected Tag Size' in the configuration file. As seen in the green cell, it created minimal error.

After we acquired this 'Tag Size', we will use it at different heights to check if it solved the problem.

Tag Size	Actual Height	Read Height	Expected Tag size	Error
0.038215324	**1**	**0.94200109**	0.040568237	0.05799891
0.038215324	**1.1**	**1.036067916**	0.040573456	0.063932084
0.038215324	**1.2**	**1.133932718**	0.040441896	0.066067282
0.038215324	**1.3**	**1.246855579**	0.039844166	0.053144421
0.038215324	**1.4**	**1.340970913**	0.039897549	0.059029087
0.038215324	**1.5**	**1.445456381**	0.039657361	0.054543619
0.038215324	**1.6**	**1.539963818**	0.039705165	0.060036182
0.038215324	**1.7**	**1.658641826**	0.039168222	0.041358174
0.038215324	**1.8**	**1.748227904**	0.039347034	0.051772096
0.038215324	**1.9**	**1.847812188**	0.03929464	0.052187812
0.038215324	**2**	**2.000000002**	0.038215324	2.27E-09
0.038215324	**2.1**	**2.07084672**	0.038753317	0.02915328
0.038215324	**2.2**	**2.225752152**	0.037773169	0.025752152
0.038215324	**1**	**0.94200109**	0.040568237	0.05799891

The table above showed us that there is no relation between the configuration parameters and the errors happening.

4.6.3.8.2 Source of Error

Unfortunately, we were not able to identify the source of errors after the previous experiments; therefore, we needed to dig deeper and pin the cause of these errors. After searching a lot online for any documented cases of similar problems encountered while using AprilTags.

Finally, we have found the real culprit, it was related to the **maximum distance for robust detection** while using AprilTags.

It is possible to achieve high tag detection accuracy with increasing the maximum detection distance. To achieve optimal conditions there is equation that describe the relation between the maximum detection distance and the focal length of camera.

The maximum detection distance (z) at quality High can be approximated by using the following formulae:

$$z = \frac{f\ s}{p}$$

$$s = \frac{t}{r}$$

where (f) is the **focal length** in pixels and (s) is the **size of a module in meters**. (s) can easily be calculated by the latter formula, where (t) is the **size of the tag in meters** and (r) is the **width of the code in modules** (for AprilTags without the white border). (p) denotes the number of image pixels per module required for detection. it varies with the tag's angle to the camera and illumination. Approximate value for robust detection: **p=5 pixels/module**.

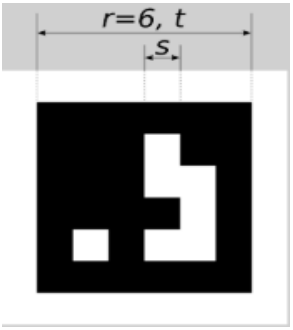


Figure 4.6.9 Visualization of module size *ss*, size of a tag in modules *r*, and size of a tag in meters *t* for AprilTags.

The following tables give sample maximum distances for different situations, assuming a focal length of 1075 pixels and the parameter quality to be set to High.

Table: Maximum detection distance examples for AprilTags with a width of t=4t=4 *cm*

AprilTag family	Tag width	Maximum distance
36h11 (recommended)	8 modules	1.1 m
16h5	6 modules	1.4 m

So, ultimately, we needed to make some adjustments. First of all, we can not change the size of the AprilTag because of our robot's size constraint. We tried changing the AprilTag family

from 36H11 to 16H5. It produced a lot of False Positives in the readings. So, we can only change the focal length of the camera, which means changing the camera itself.

4.6.3.8.3 Mitigating the Errors

After applying those equations with the Kinect parameters, we found out that the maximum distance we were able to use is 0.45 meters from the tags. With the Kinect's field of view, it left us with almost no area for the robots to move on.

Our first option was trying with laptop camera in the simulated environment. Unfortunately, it was also not good enough for our application.

As we mentioned before, not many locally available cameras with accurate datasheet are available. But, luckily, some of us used Intel Realsense D435 Camera before and had access to one. After applying the equations again on this camera, we found that we can place the camera at 1.1 meters with errors that range from 0.1 cm to 0.8 cm. That was a huge improvement over the Kinect.

So, with the camera model placed at X = 0.25m and Y = 0.25 m, the expected results are:

Camera Setup in Simulation Scene:

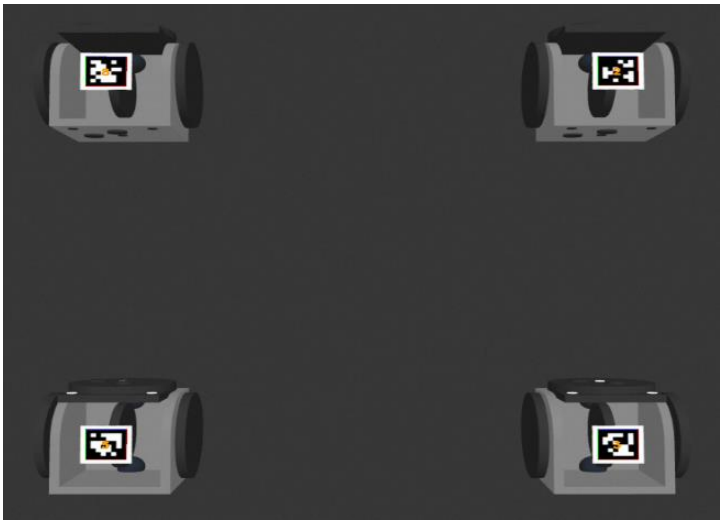
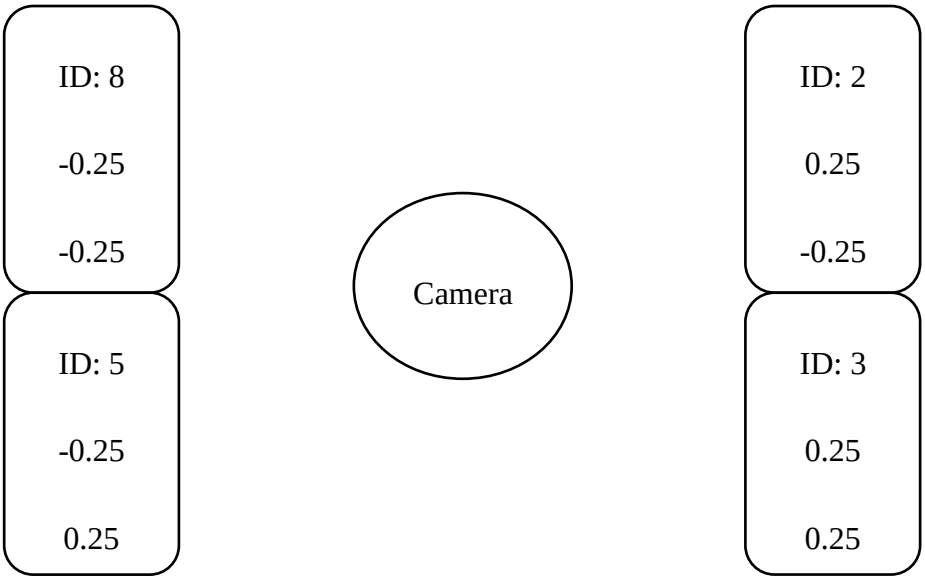


Figure 4.6.10 four Modular robots Square arrangment

Expected Readings:



Actual Tag Readings:

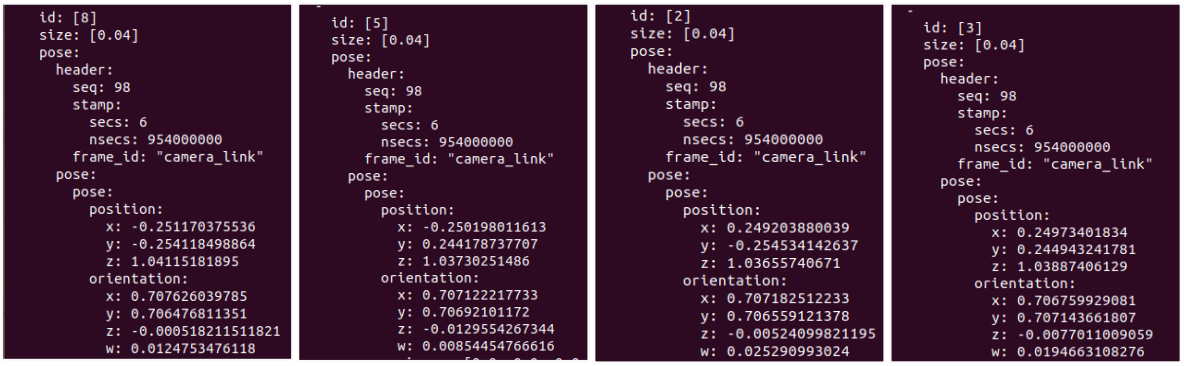
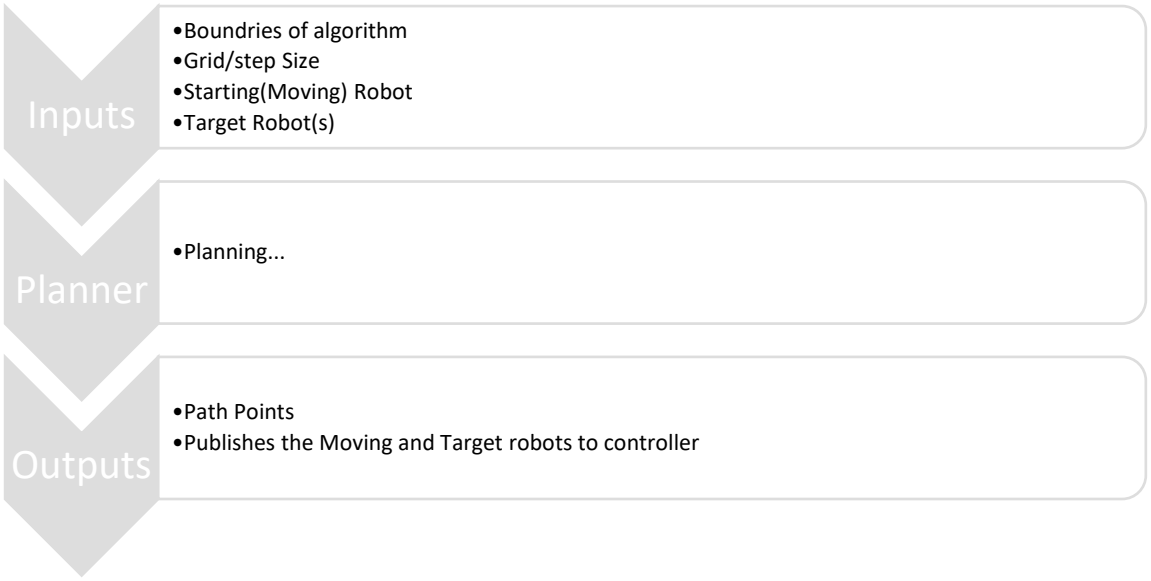


Figure 4.6.11 AprilTags Algorithm Results

4.7 Path Planning Between Robots

We managed to get the poses of all the robots as shown before. Now we need to move one robot to another so we can ultimately create some desired formation in the end. It is an A* planner. It operates as shown in the next diagram:



This planner is tailored to our needs, let me explain. Moving one robot to another robot is no problem, but at some point, those two will be connected. After that we will want to join those two (joined robots) with a third robot. The planner will be able to create this path with the correct input from the user, starting from the moving robot and ending with the robot that we want to connect to.

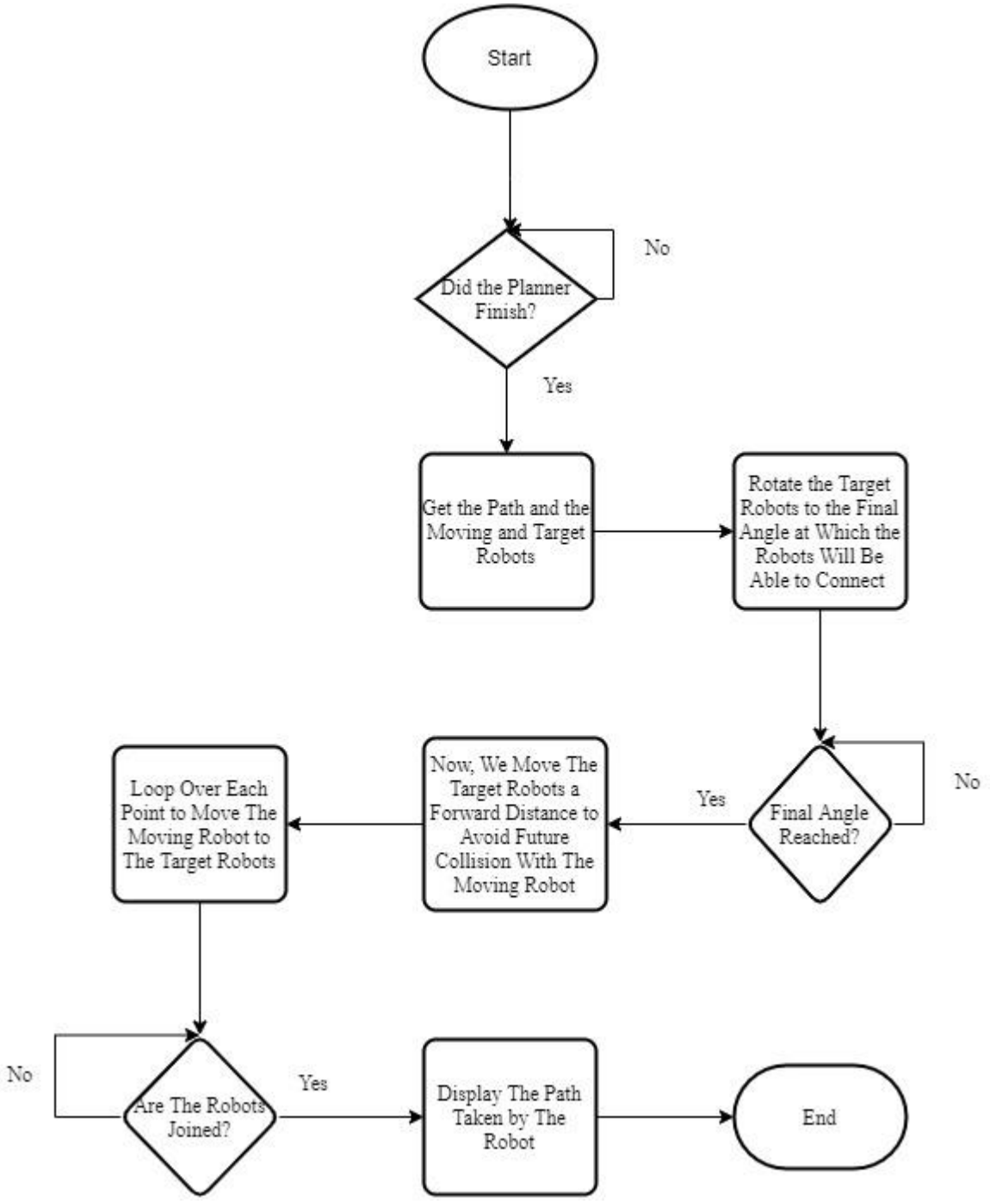
Also, the grid size used for planning is adjustable by the user. Larger grid means faster planning but with lower accuracy, and vice versa.

Finally, the code publishes a message type that is created by us for this application. It contains an array with the X point coordinates, an array with the corresponding Y point coordinates, the ID of the moving robot, and the IDs of the target robots. This message is read by the controller that handles the docking of those robots together.

4.8 The Control Algorithm

The control algorithm is completely abstract. It can move control the movement from any robot to any number of joined robots. We only experimented with 4 robots at a time due to the lack of computational power required to simulate more than 4 robots at a time. The following flowchart will explain how the controller works.

4.8.1 Control Flowchart



4.8.2 Controller Functions

In this section we will highlight how this controller is adaptive to the situations that we need. There are two main functions that mobilize the robot(s) in the environment and one additional function that just makes things easier for us. The three functions are:

1. Forward Function

2. Rotate Function
3. Forward Distance Function

General notes about this controller:

- It acquires the IDs of the moving robot and the target robots from the path planner.
- It can deal with any number of robots. Only four are used, since we do not have enough processing power to operate more.
- This controller only creates publishers for the IDs received from the planner. It is not hard coded. We utilized the python dictionary here.

4.8.2.1 Forward Function

This function moves the robot in a forward direction assuming it already has the correct heading. It takes four input arguments: X goal coordinate, Y goal coordinate, the speed of the wheels, and the desired robot ID(s) to move. The ID is an integer, but it is actually sent as a string. By finding how many elements in this string and what are they, we can publish the speed to them.

For example, if the ID that is sent is the 'Moving Robot', then the string is only one number. This means that the function will only operate on a single robot. If the ID that is sent is for the 'Target Robots', it may look like this: 234, this means that the function will move robots 2, 3, and 4 simultaneously forward.

After the function knows which robots to move, it calculates the euclidian distance between the current position and the goal position. When this goal is reached, the robots stops and the function is exited.

4.8.2.2 Rotate Function

This function also is generic and can operate on any number of robots. It can rotate a single robot, or four robots docked together at the same time. The main trick here is that, due to noises in the simulated environment, the robots tend to rotate and move without being ordered to do so.

So, we created an external text file to store all the robots angles, the real angles that they should be at without the errors that rotate the robots. This helped us keep track of the orientations that the robots should be at.

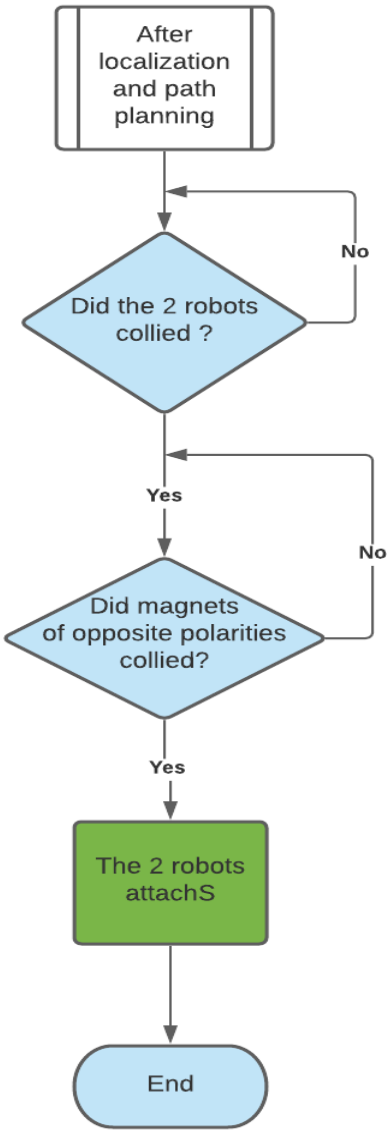
This was essential since this function takes, as an input, the desired angle to rotate. It as an incremental value to rotate, not the absolute. This means that in every time we use this function on the same robot, the errors will also accumulate.

4.8.2.3 Forward Distance Function

This is exactly like the forward function mentioned before, but the only difference is that it takes only the desired distance to move the robots.

4.9 Docking process

In this process, 2 plugins were used which were completely written from scratch to match our robot requirements and to operate efficiently.



4.9.1 Contact Plugin

Since there are no magnets in gazebo simulation, contact sensor was used to fulfill the docking task.

The primary task of ContactPlugin is to receive **magnetic face** which is the name of contact sensor.

➔ For example: The contact sensor on the north of the front face is called: **front_face_magnet_north**.

In each MSRR SDF file, the contact plugin is defined as:

```
<sensor name='my_plugin_2' type='contact'>
  <plugin name='ContactPlugin_2' filename='libcontact.so'>
    <robotNamespace>/</robotNamespace>
    <magnetic_face>body_magnet_south</magnetic_face>
  </plugin>
  <contact>
    <collision>box_collision</collision>
    <topic>__default_topic__</topic>
  </contact>
  <update_rate>5</update_rate>
</sensor>
```

Figure 4.9.1 Front Face SDF File Snip

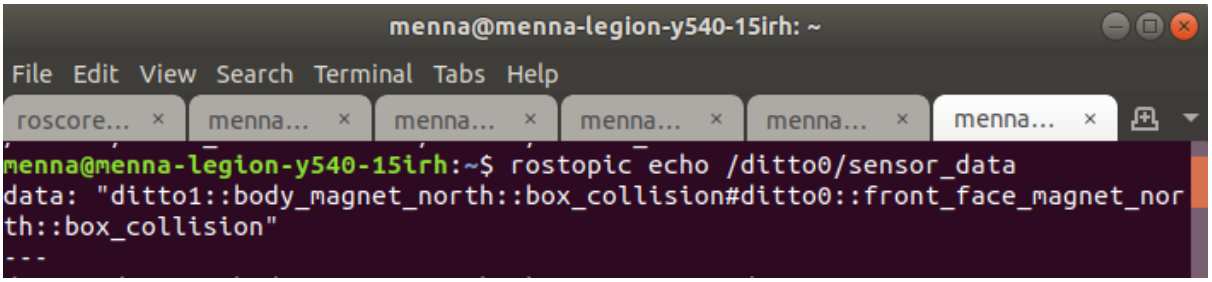
Simply, when a contact sensor collides with any object, it states which object it has collided into.

Knowing that, we can easily investigate which object it has collided into and if it was one of the contact sensors that are placed in the front face or back, and it will attach both robots (must be opposite polarity) ; otherwise nothing happens.

The command used to operate the docking between 2 robots:

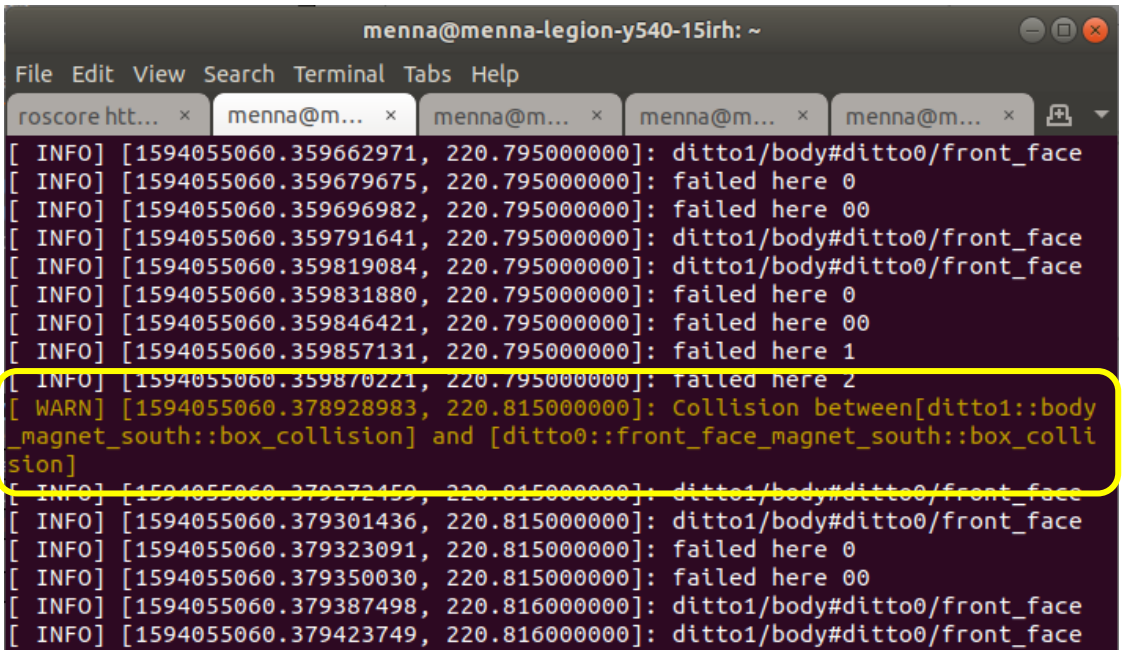
➔ **Command**: **rosrun gp_abstract_sim magnetism**

The **magnetism** task is to check whether the robot has collided to another magnet (contact sensor) or not; this happens by subscribing on topic **ditto.No\sensor_data** and slicing the message.



4.9.1.1 The Message Content

Collision between **ditto1::body_magnet_south** and **ditto0::front_face_magnet_south**



The output message after slicing is:

For Robot1 (Ditto0), Robot2 (Ditto1):

1) **ditto0/body#ditto1/front_face**: First_Collided_Robot_name/magnet_ position

   or

2) **ditto1/front-face#ditto0/body**: Second_Collided_Robot_name/magnet_position

The magnet position can be:

- front face
- body (referring to Back)

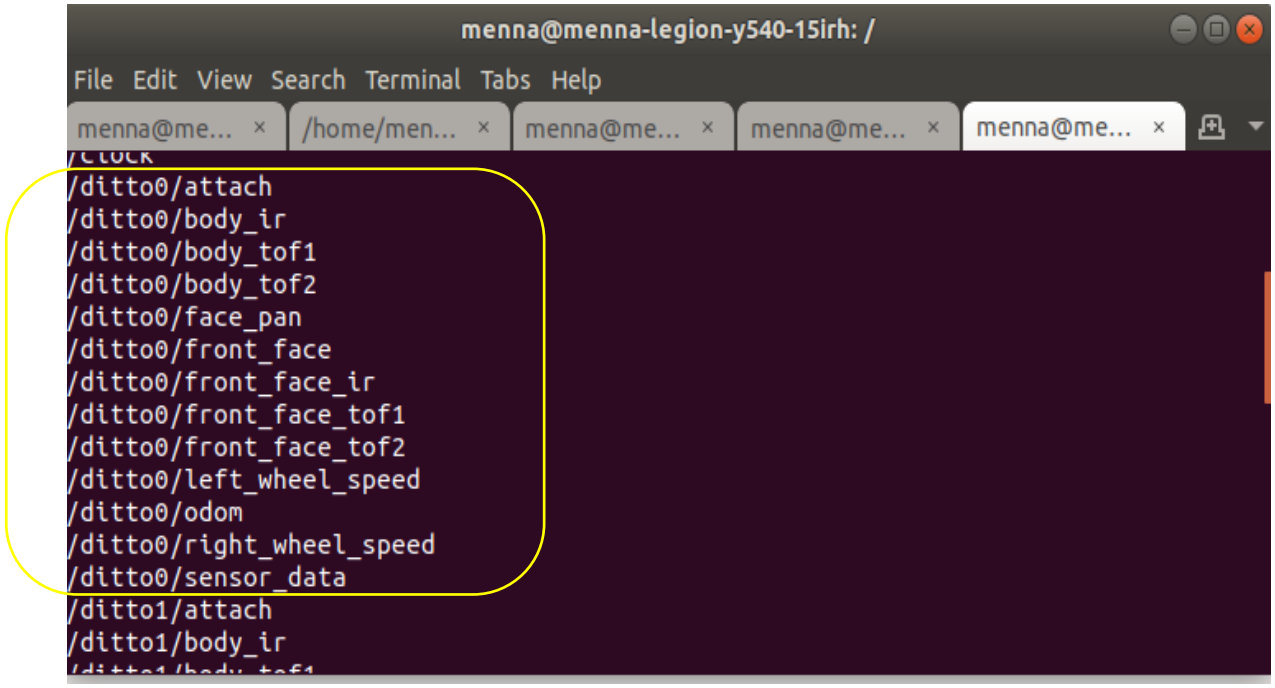
After that the message is published to **my_speed_controller_plugin** via topic **ditto.No/attach.**

```
^Cmenna@menna-legion-y540-15irh:~$ rostopic echo /ditto0/attach
data: "ditto1/body#ditto0/front_face"
---
```

4.9.2 My Speed Controller Plugin

This plugin is responsible for controlling the movement of the robot and docking of 2 robots.

It provides 12 topics for each robot in the scene:



4.9.2.1 Topics

4.9.2.1.1 /ditto0/attach

It is responsible for attaching 2 robots. When it receives a message from topic **ditto.No\attach ,** it takes action to attach or not depending on the content of the message.

The result of docking 2 robots:

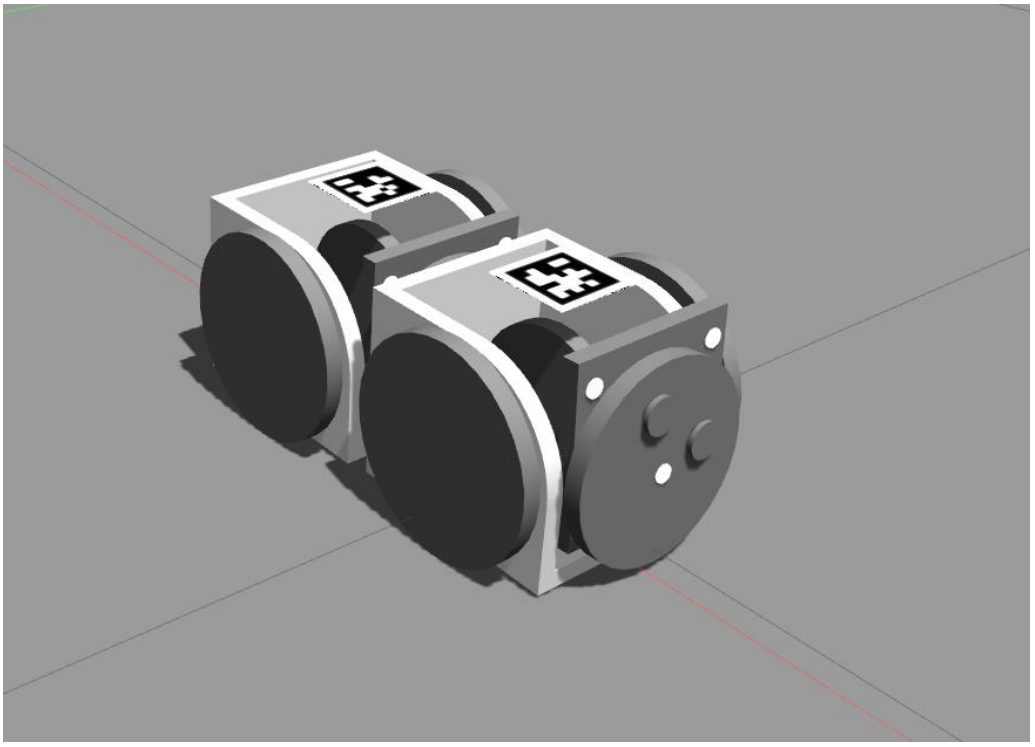


Figure 4.9.2 Docking 2 Robots

4.9.2.1.2 /ditto0/body_ir , ditto0/front_face_ir

/ditto0/body_ir is responsible for receiving the IR sensor readings on the back of the robot, while **ditto0/front_face_ir** is responsible for receiving the IR sensor readings on the front face of the robot.

4.9.2.1.3 /ditto0/body_tof1, ditto0/body_tof2, /ditto0/front_face_tof1, /ditto0/front_face_tof2

/ditto0/body_tof1 and **/ditto0/body_tof2** are responsible for receiving the reading of TOF sensor on the north west and north east of the back, while /**ditto0/front_face_tof1, /ditto0/front_face_tof2** are responsible for receiving the reading of TOF sensor on the north west and north east of the front face.

4.9.2.1.4 /ditto0/face_pan

It is responsible for publishing the speed to the front face pan of the robot, and for rotating the front face pan.

4.9.2.1.5 /ditto0/front_face

It is responsible for publishing the speed to the front face of the robot, and for tilting the front face.

4.9.2.1.6 /ditto0/right_wheel_speed

It is responsible for publishing the speed to the right wheel of the robot.

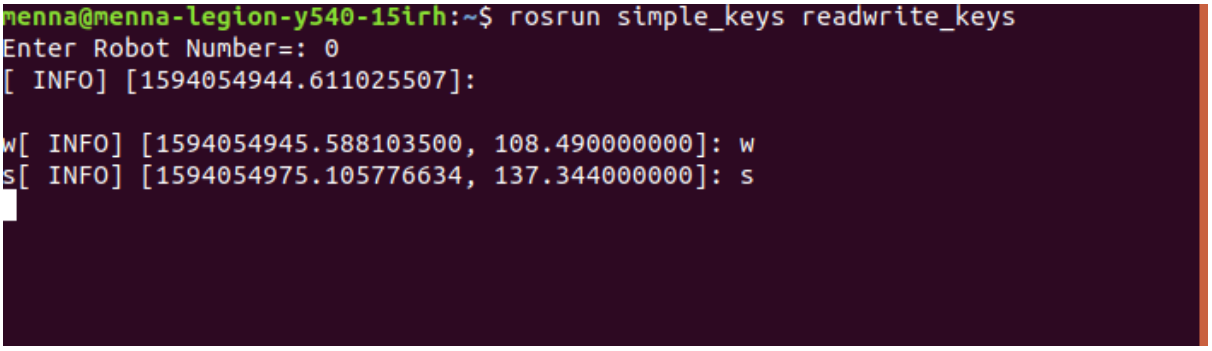
4.9.2.1.7 /ditto0/left_wheel_speed

It is responsible for publishing the speed to the left wheel of the robot.

4.9.3 Simple Keys Package

This node runs in the terminal and controls the modules by the keyboard, but you must not leave the terminal (the terminal running this node must not lose focus) or the keyboard keys won't control the module.

➔ **Command: rosrun simple_keys readwrite_keys**



4.9.3.1 Key dictionary

- W: move forward
- S: stop the module
- X: move backward
- A: rotate left
- D: rotate right
- Q: forward left
- E: forward right
- Z: backward left
- C: backward right
- U: rotates the tilting mechanism up
- J: rotates the tilting mechanism down

- M: stops the tilting mechanism
- O: front face rotates CCW
- L: front face rotates CW
- I: stops rotating mechanism

4.10 Experimentation

The following settings are set for all the next experiments:

- Elements: Four MSR Robots.
- Path Planning Method: A* algorithm.
    - Grid size: 0.5
    - Robot radius: 0.05
    - Source, goal robots must be defined each time.
- Goal: Docking Four MSR robots.

4.10.1 Experiment #1 "Docking Four MSR Robots on Straight line "

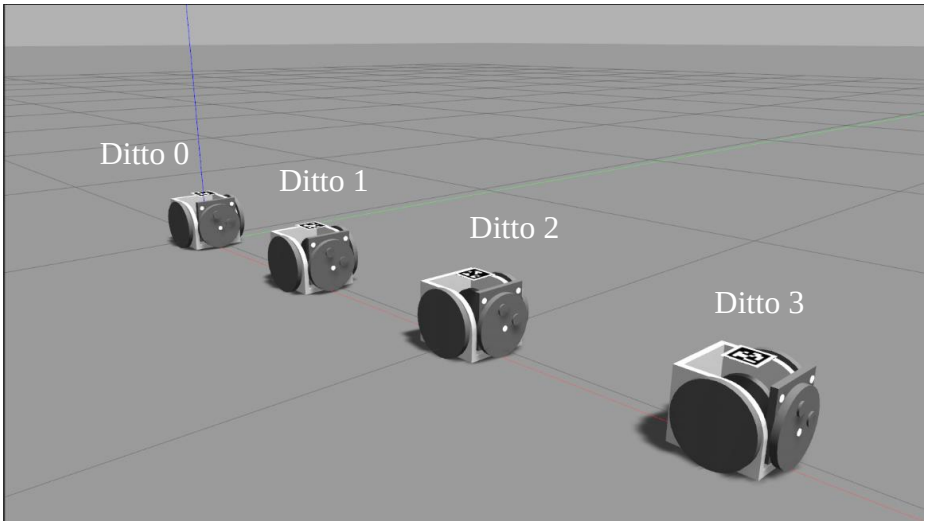
In this experiment we are going to dock **Four** robots each is apart by 0.5 m on x-axis.

4.10.1.1 The First Phase

4.10.1.1.1 Step (1)

To start the experiment type in terminal:

➔ Command: **rosrun gp_abstract_sim robotcams.launch**



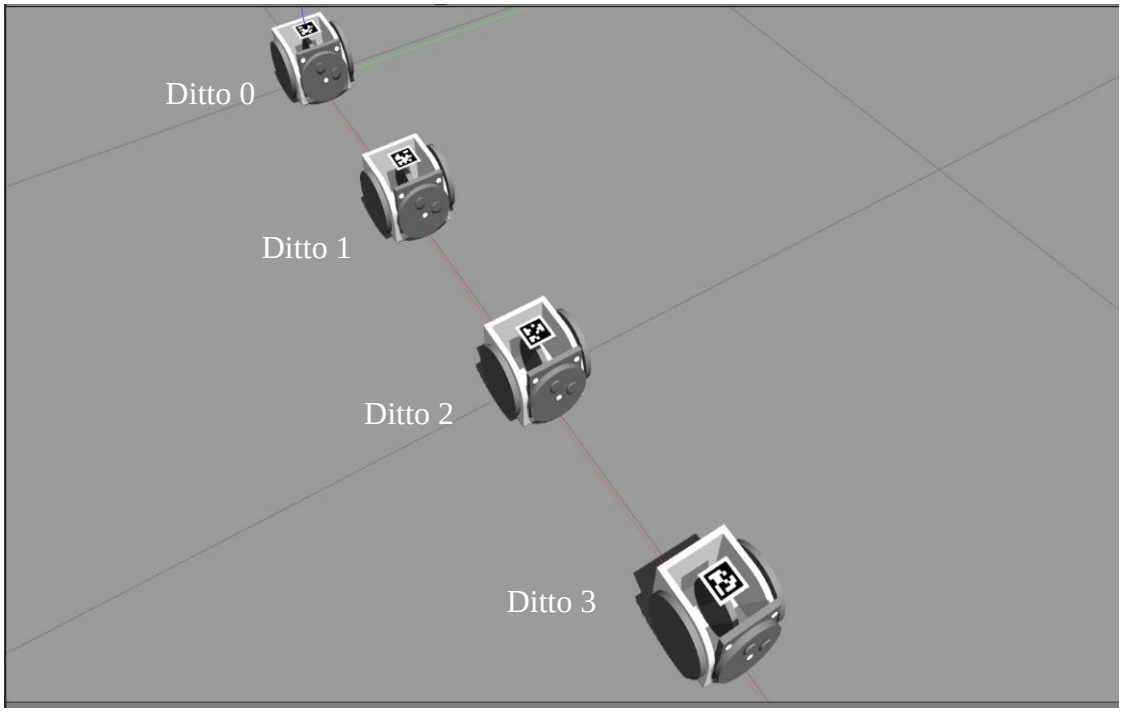


Figure 4.10.1 4 Robots in Row

4.10.1.1.2 Step (2)

To determine the position and orientation of each robot, multiple_model_states.py is used

➔ Command: **rosrun gp_abstract_sim multiple_model_states.py**

```
menna@menna-legion-y540-15irh:~$ rosrun gp_abstract_sim multiple_models_states_v
2.py
```

Since there are 4 robots in the scene the output of the command is:

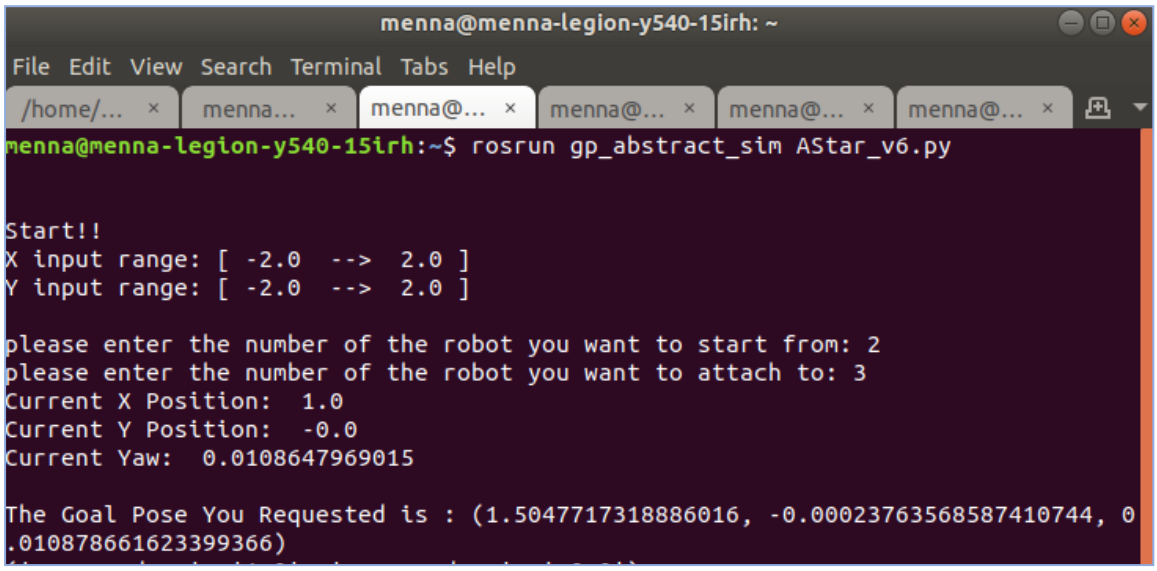
```
menna@menna-legion-y540-15irh: ~
File  Edit  View  Search  Terminal  Tabs  Help
/home/...  ×  menna...  ×  menna@...  ×  menna@...  ×  menna@...  ×  menna@...  ×

ditto1 Caught in Scene
ditto2 Caught in Scene
ditto3 Caught in Scene
ditto0 Caught in Scene
```

4.10.1.1.3 Step (3)

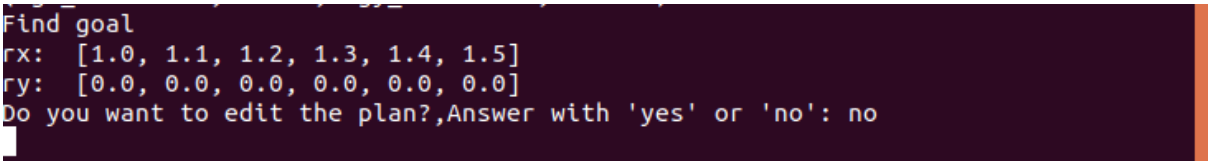
After determining the position and orientation of each robot, start path planning, A* path planner is used:

➔ Command : **rosrun gp_abstract_sim AStar_v6.py**

In this case the moving robot is “ditto2” to the goal robot “ditto3”



Depending on the goal robot the path is created



4.10.1.1.4 Step (4)

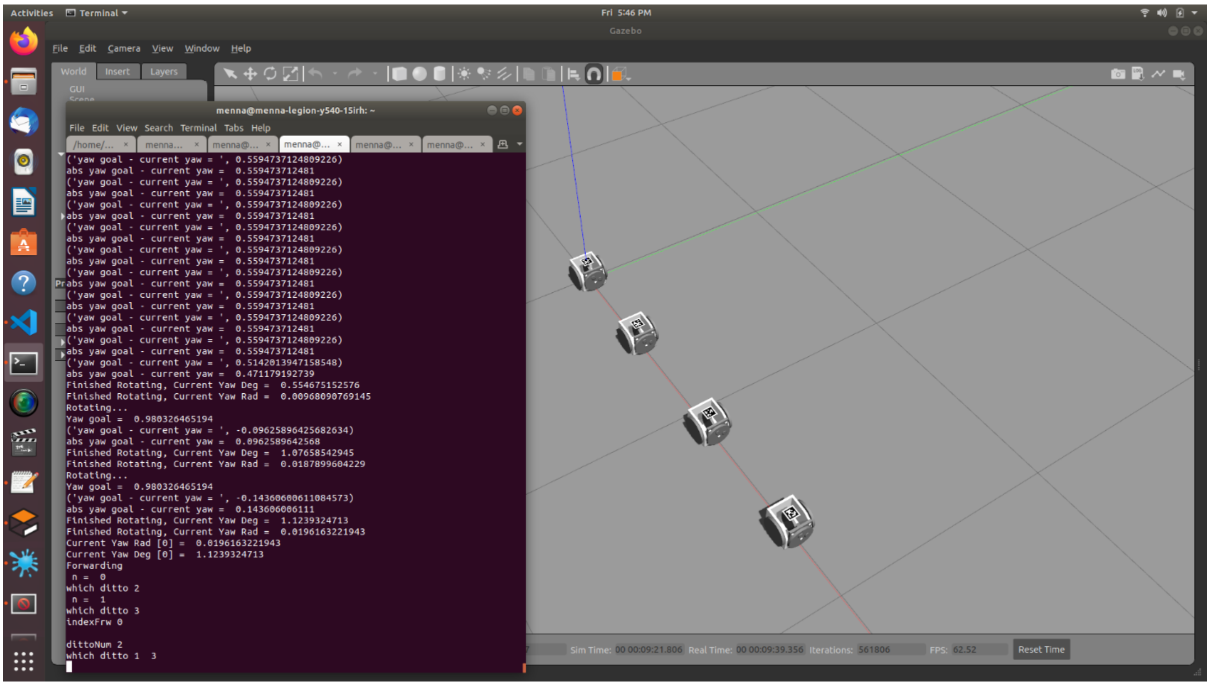
To execute the movement of the robot**, multi_trial_control_v7.py** is used :

➔ Command: **rosrun gp_abstract_sim mult_trial_control_v7.py**

First, ditto3 moves forward 0.18 m, so that when ditto2 reaches the goal it doesn’t collide with it.

Second, ditto2 adjust its position and orientation at each point, then starts moving.

In case the back of the goal robot isn’t facing the front face of the moving robot, the goal robot rotates to adjust its orientation to ensure that the back is facing the front face of the moving robots, in our case ditto3’s back is already facing ditto2’s front face.



4.10.1.1.5 Problem 1 ☹

At the last point of the path, robot rotates complete cycle to adjust its orientation instead of rotating to the opposite direction to get the desired angle, this causes wasting of time during simulation, and unnecessary extra rotation.

Robot (ditto2) has adjusted its orientation after rotating a complete cycle.

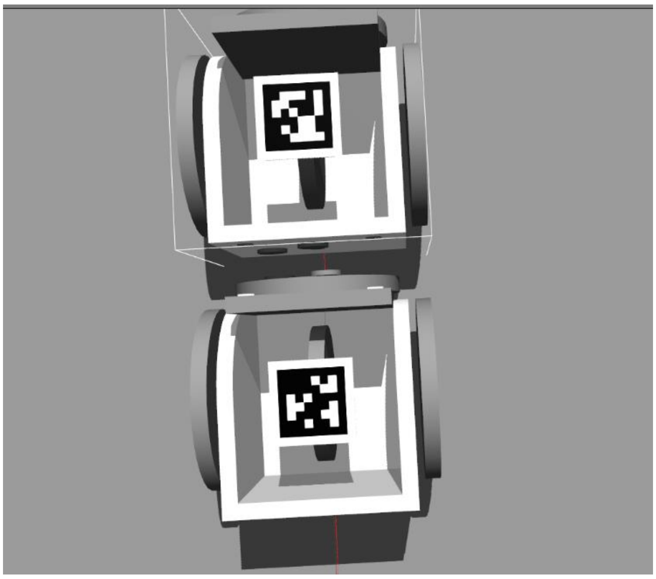


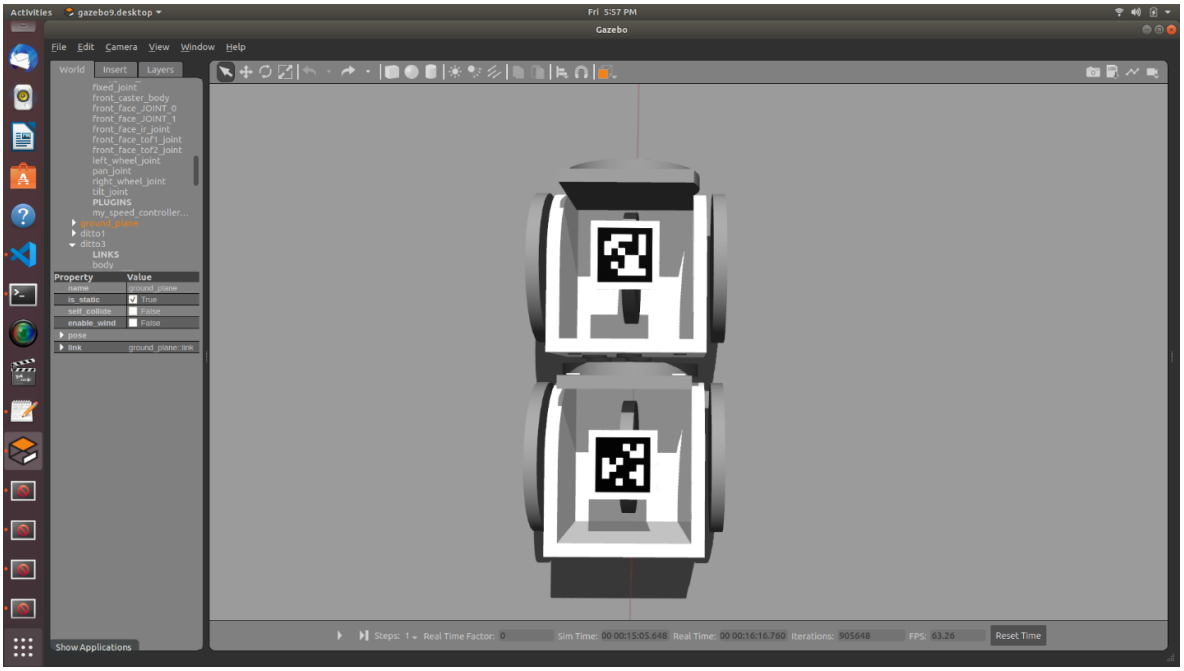
Figure 4.10.2 Problem 1 in Docking

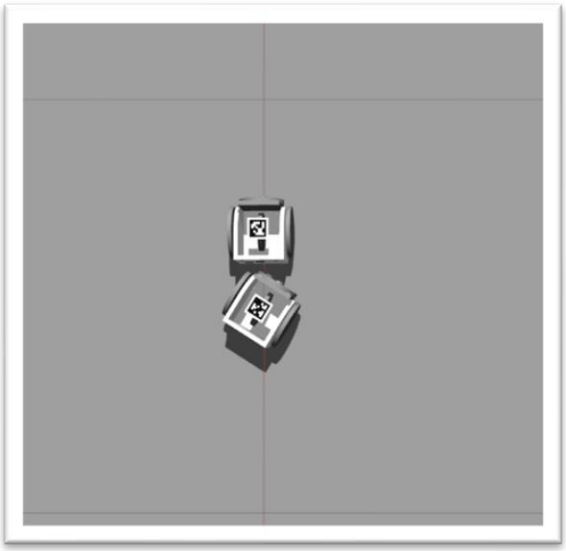
4.10.1.1.6 Step (5)

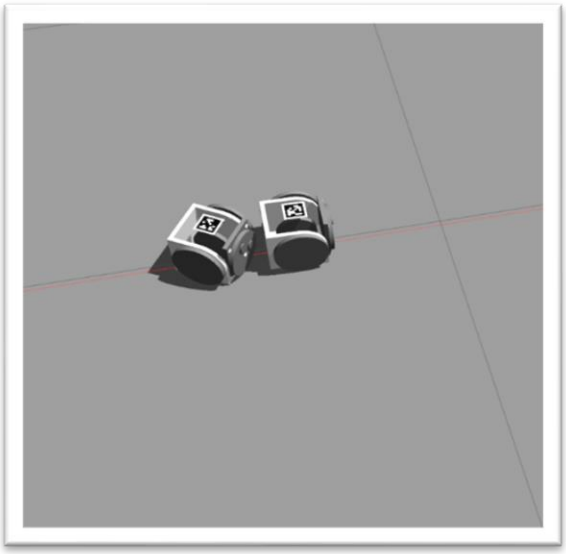
For a smooth docking after ditto2 reaches the destinantion, it moves forward with a very low speed 0.02 m/sec

Until it attaches ditto3 via **magnetism.py**

➔ Command: **rosrun gp_abstract_sim magnetism.py**







The view after ditto2 and ditto3 have attached 😊

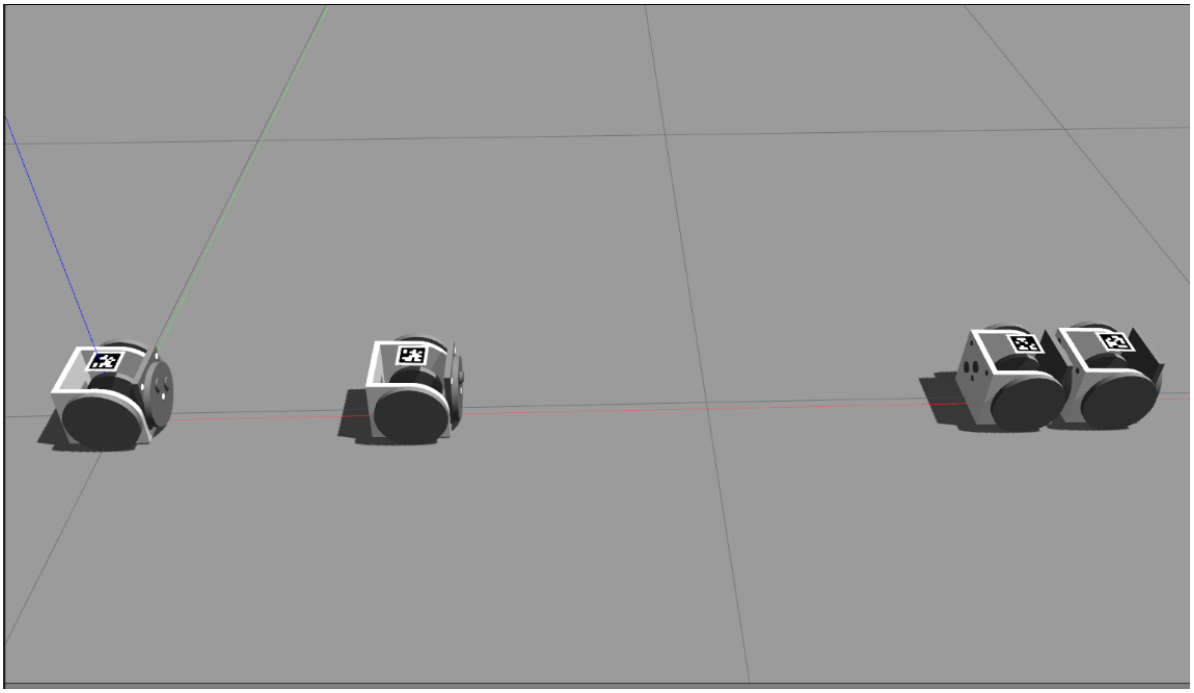


Figure 4.10.3 View After Ditto 2 and 3 are Attached

4.10.1.2 The Second Phase ditto1 attaches to ditto2, and ditto3:

4.10.1.2.1 Steps 3,4,5 are repeated

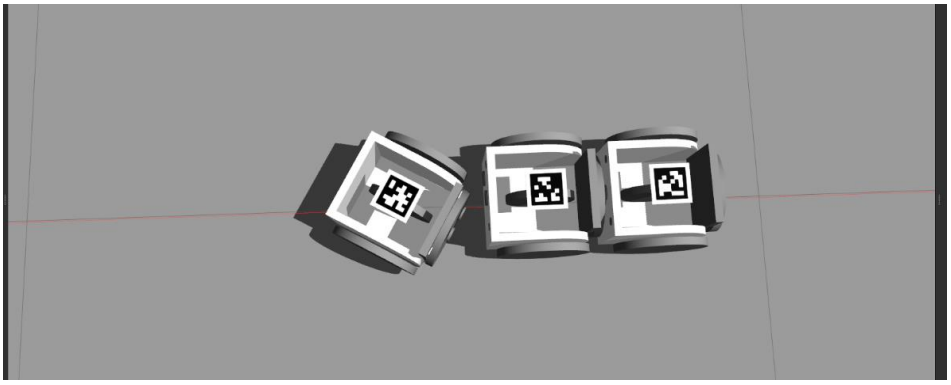
In step 3: in order to attach a robot to a pile of robots, the number of robots is written from the back to the front robot → in this case the last robot is ditto2 while the first one is ditto3.

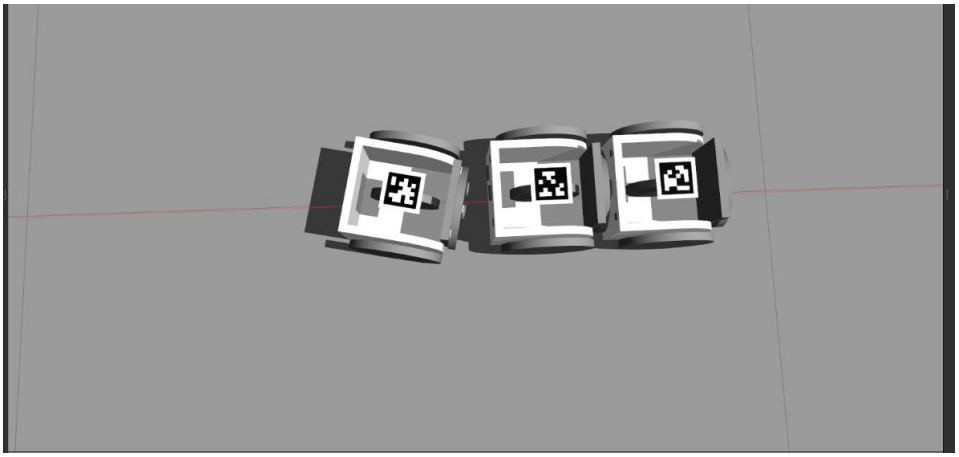
Therefore, the number of robots you want attach to is: 23

```
menna@menna-legion-y540-15irh:~$ rosrun gp_abstract_sim AStar_v6.py


Start!!
X input range: [ -2.0  -->  2.0 ]
Y input range: [ -2.0  -->  2.0 ]

please enter the number of the robot you want to start from: 1
please enter the number of the robot you want to attach to: 23
Current X Position:  0.5
Current Y Position:  -0.0
Current Yaw:  0.0195823408202

The Goal Pose You Requested is : (1.523244323442088, -0.0008111221399844085, 0.0
11385782984092731)
('sx_round:: ', '0.5', 'sy_round:: ', '-0.0')
('gx_round:: ', '1.52', 'gy_round:: ', '-0.0')
Find goal
rx:  [0.5, 0.6, 0.7, 0.8, 0.9, 1.0, 1.1, 1.2, 1.3, 1.4, 1.5]
ry:  [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
Do you want to edit the plan?,Answer with 'yes' or 'no': no
```

4.10.1.2.2 The same problem happens again ☹

In the last point the moving robot rotates a complete cycle!

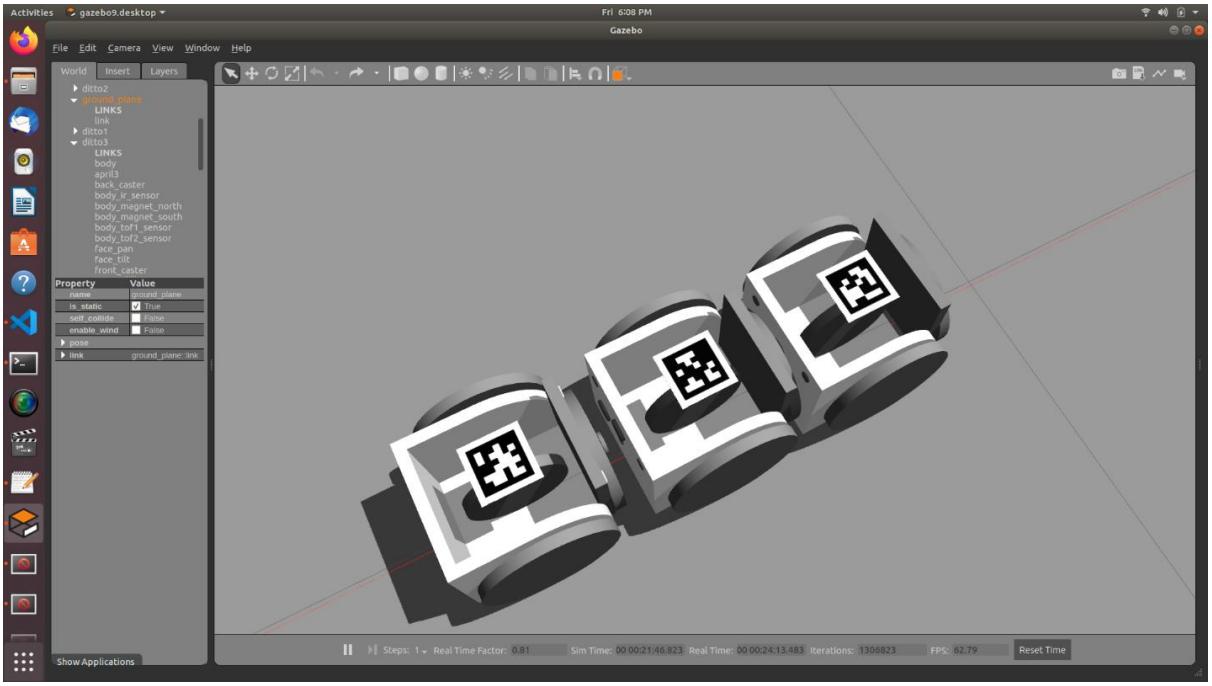




4.10.1.2.3 2nd time Step (5)

For a smooth docking after ditto1 reaches the destinantion, it moves forward with a very low speed 0.02 until it attaches ditto2 via **magnetism.py.**

➔ Command: **rosrun gp_abstract_sim magnetism.py**



The view of ditto1 is attached to ditto2, and ditto3 😊

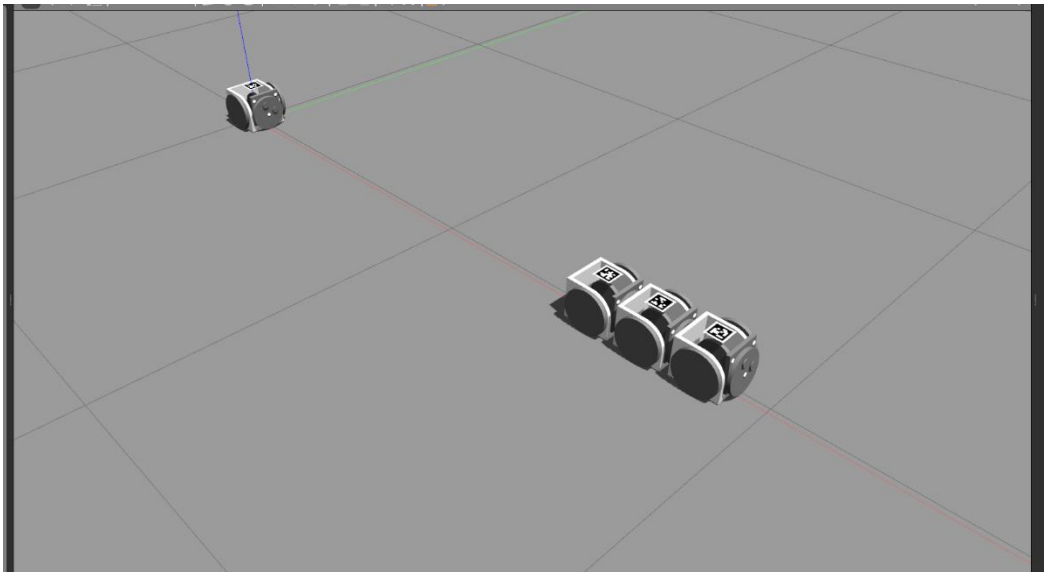


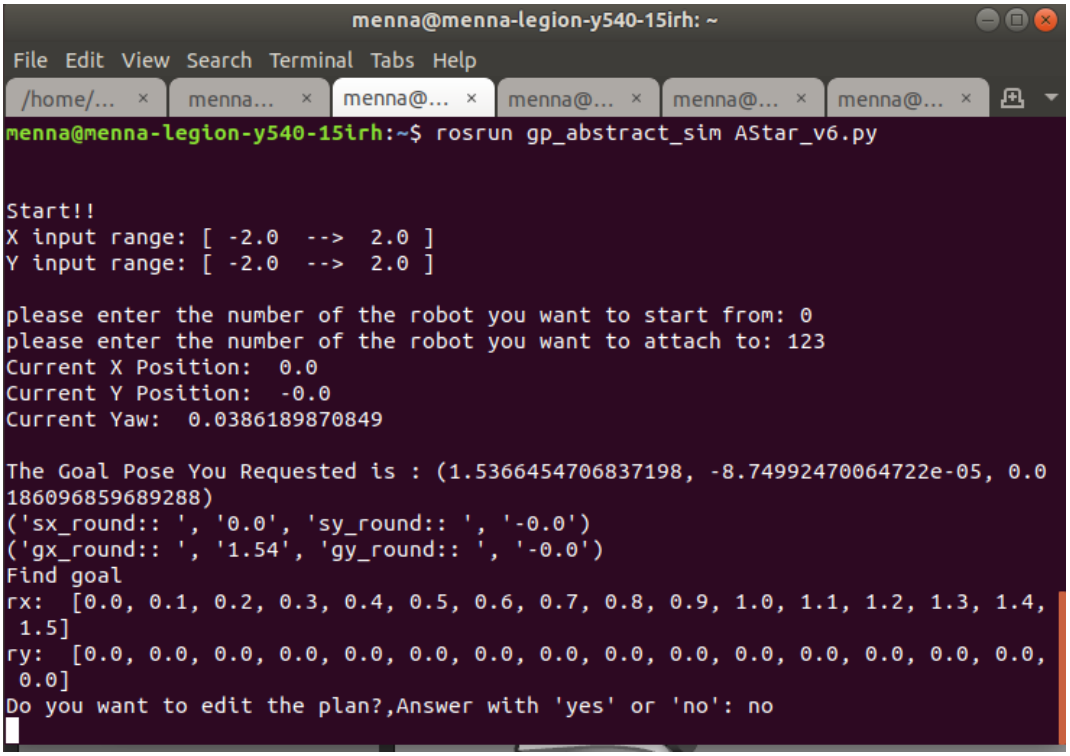
Figure 4.10.4 3 Robots Attached

4.10.1.3 The Last Phase

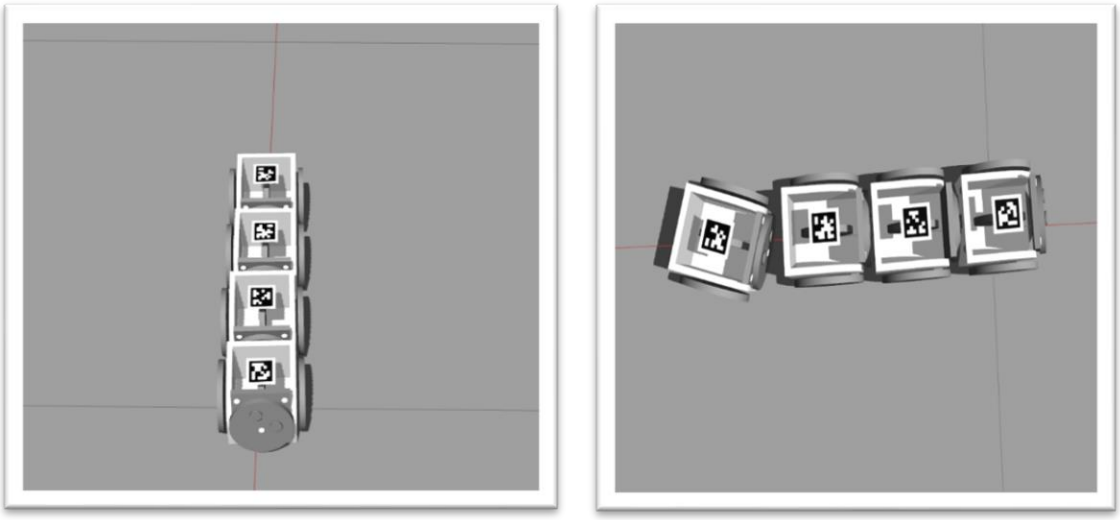
Ditto0 attaches to ditto1, ditto2, and ditto3:

4.10.1.3.1 Steps 3,4,5 are repeated

In step 3: The number of robots want to attach to is written from back to front, in this case it is 123.



4.10.1.3.2 The same problem happens for the 3rd time



4.10.1.4 The Final Result

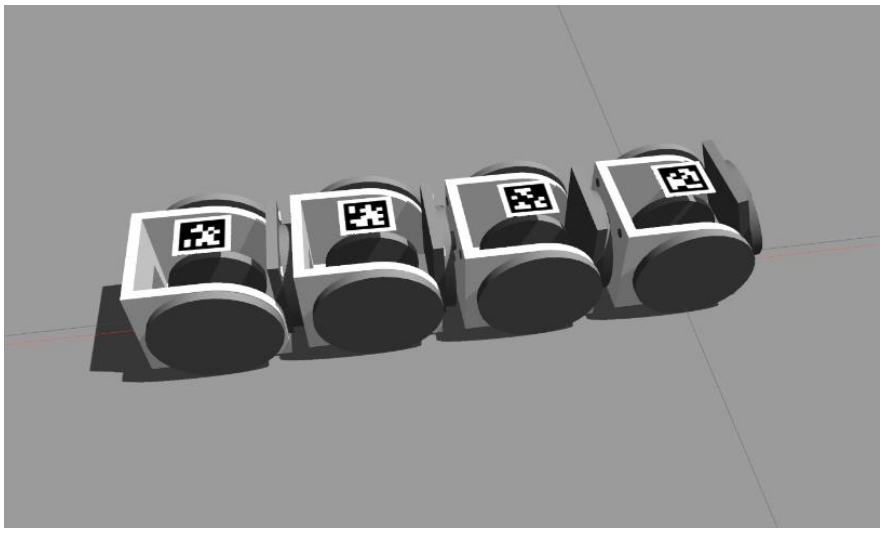


Figure 4.10.5 Four Robots Attached

4.10.1.5 Observation

There is a slight misalignment takes place when 2 robots attach, but it can be excused and neglected, because the 2 magnets still collide and once, they collide the 2 robots attach to each other smoothly.

4.10.2 Experiment #2 “Docking Four MSR Robots on Corners “

In this experiment we are going to dock **Four** robots each is on corner grid of the ground plane

4.10.2.1 The First Phase

4.10.2.1.1 Step (1)

To start the experiment type in terminal:

- ➔ Command: **rosrun gp_abstract_sim robotcams.launch**

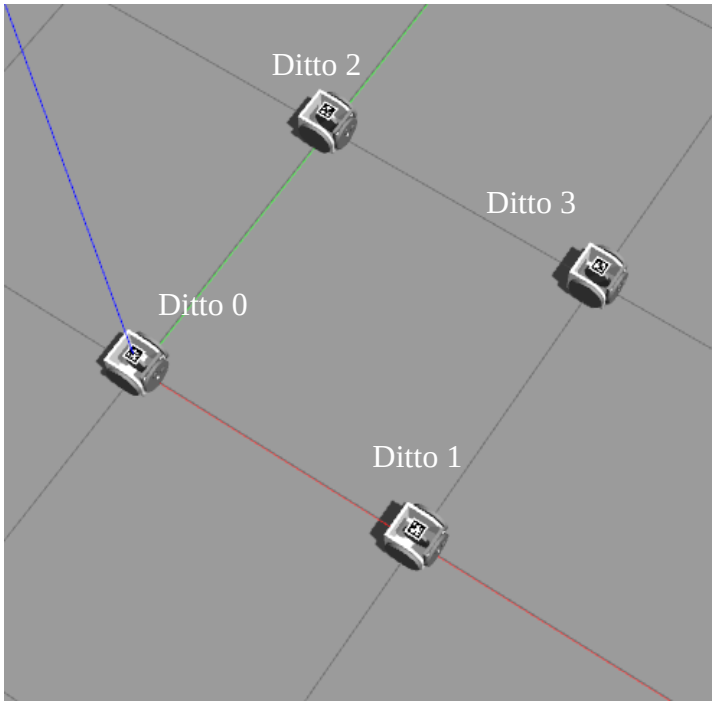


Figure 4.10.6 Experiment 2 Setup

4.10.2.1.2 Step (2)

To determine the position and orientation of each robot, multiple_model_states.py is used

➔ Command: **rosrun gp_abstract_sim multiple_model_states.py**

```
menna@menna-legion-y540-15irh:~$ rosrun gp_abstract_sim multiple_models_states_v
2.py
```

Since there are 4 robots in the scene the output of the command is:

```
menna@menna-legion-y540-15irh: ~
File  Edit  View  Search  Terminal  Tabs  Help
/home/...  ×  menna...  ×  menna@...  ×  menna@...  ×  menna@...  ×  menna@...  ×

ditto1 Caught in Scene
ditto2 Caught in Scene
ditto3 Caught in Scene
ditto0 Caught in Scene
```

4.10.2.1.3 Step (3)

After determining the position and orientation of each robot, start path planning, A* path planner is used:

➔ Command : **rosrun gp_abstract_sim AStar_v6.py**

In this case the moving robot is “ditto0” to the goal robot “ditto1”

```
menna@menna-legion-y540-15irh:~$ rosrun gp_abstract_sim AStar_v6.py


Start!!
X input range: [ -2.0  -->  2.0 ]
Y input range: [ -2.0  -->  2.0 ]

please enter the number of the robot you want to start from: 0
please enter the number of the robot you want to attach to: 1
Current X Position:  0.0
Current Y Position:  -0.0
Current Yaw:  0.000139416597392

The Goal Pose You Requested is : (1.0015887713825216, -1.032860238810217e-05, 0.
00013981062068866762)
('sx_round:: ', '0.0', 'sy_round:: ', '-0.0')
('gx_round:: ', '1.0', 'gy_round:: ', '-0.0')
Find goal
rx:  [0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0]
ry:  [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
Do you want to edit the plan?,Answer with 'yes' or 'no': no
```

Path Points

4.10.2.1.4 Step (4)

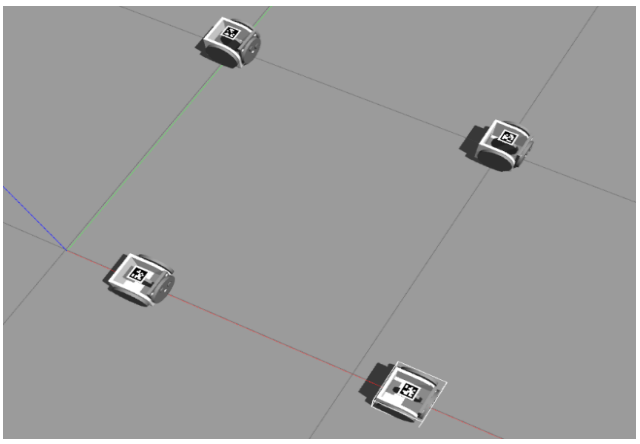
To execute the movement of the robot**, trial_control.py** is used:

➔ Command: **rosrun gp_abstract_sim trial_control.py**

First, ditto1 moves forward 0.18 m, so that when ditto0 reaches the goal it doesn’t collide with it.

Second, ditto0 adjust its position and orientation at each point, then starts moving.

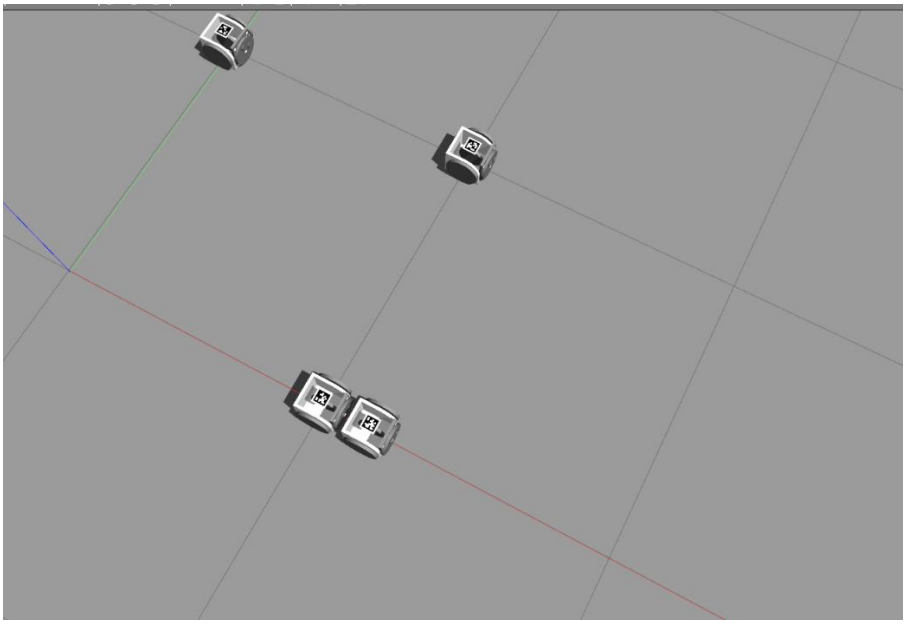
In case that the back of the goal robot isn’t facing the front face of the moving robot, the goal robot rotates to adjust its orientation to ensure that the back is facing the front face of the moving robots, in our case ditto1’s back is already facing ditto0’s front face.



4.10.2.1.5 Step (5)

For a smooth docking after ditto0 reaches the destination, it moves forward with a very low speed 0.02 m/sec, until it attaches ditto1 via **magnetism.py**

➔ Command: **rosrun gp_abstract_sim magnetism.py**

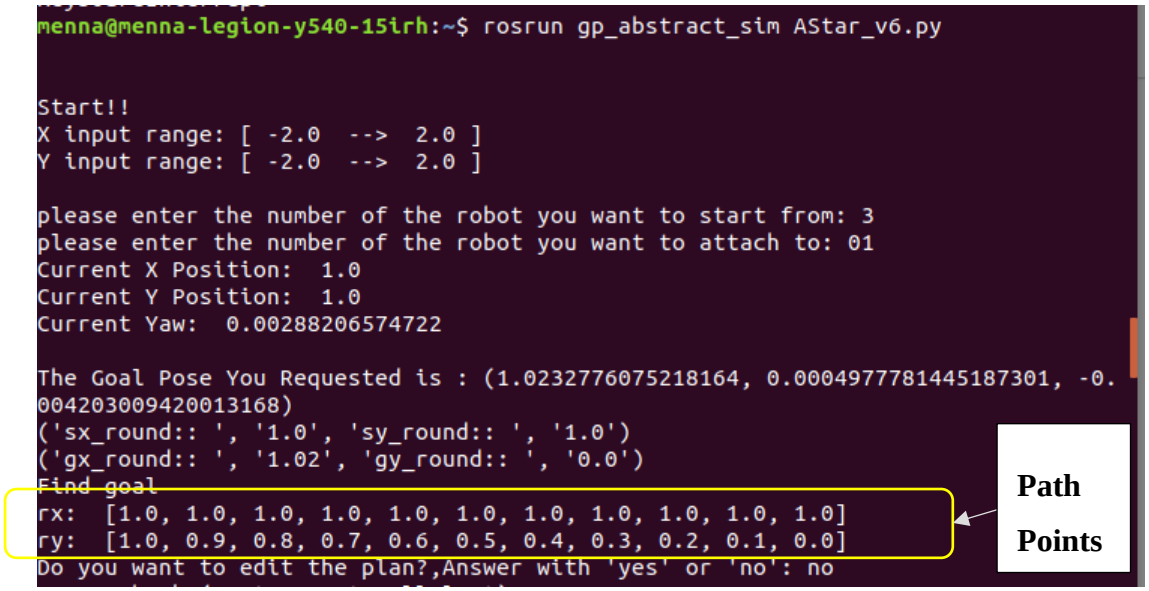


4.10.2.2 The Second Phase ditto3 attaches to ditto0, and ditto1:

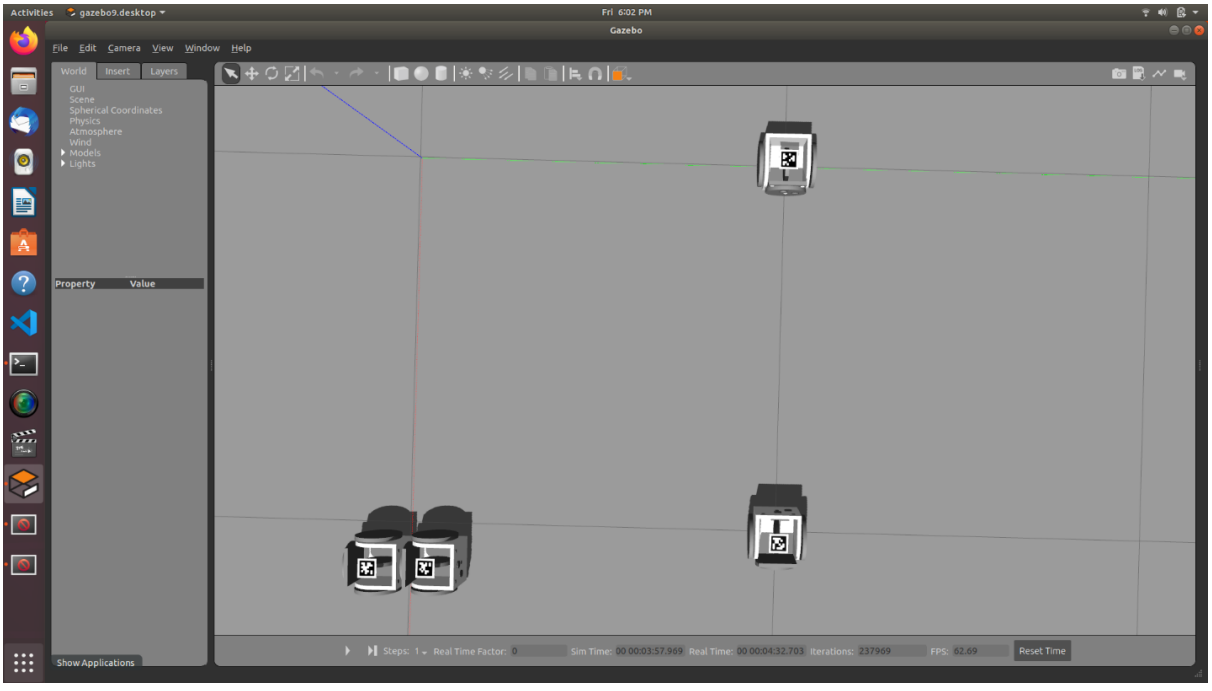
4.10.2.2.1 Steps 3,4,5 are repeated

In step 3 : in order to attach a robot to a pile of robots, the number of robots is written from the back to the front robot → in this case the last robot is ditto0 while the first one is ditto1.

Therefore, the number of robots you want attach to is: 01



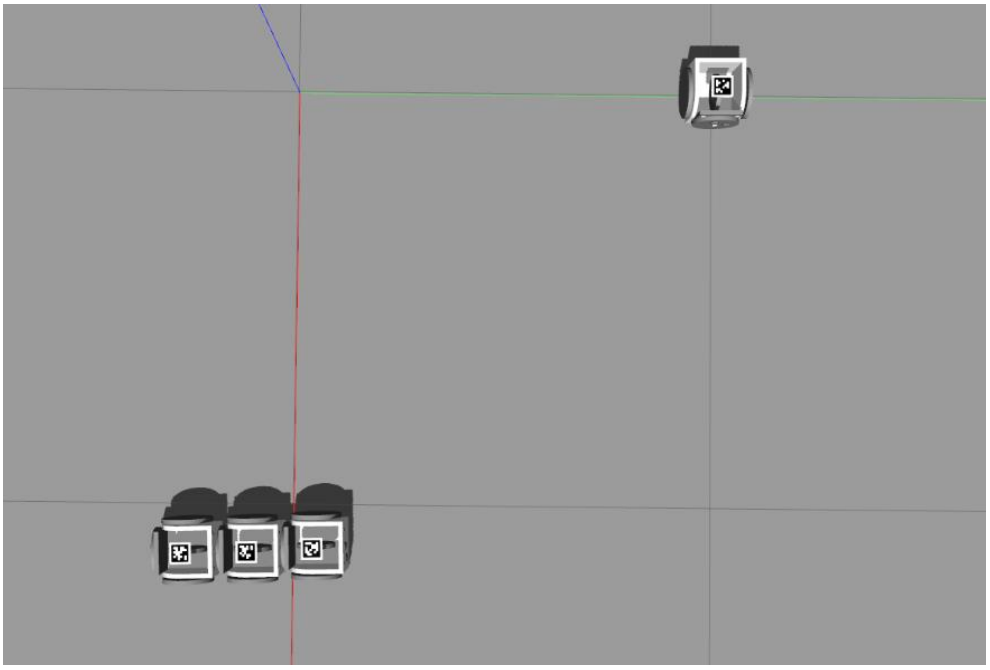
In step 4: ditto0, ditto1 rotate 90 degree, so that the back of ditto0 is facing the front face of ditto3



4.10.2.2.2 2nd time Step (5):

For a smooth docking after ditto3 reaches the destination, it moves forward with a very low speed 0.02, until it attaches to ditto0 via **magnetism.py**

➔ Command: **rosrun gp_abstract_sim magnetism.py**

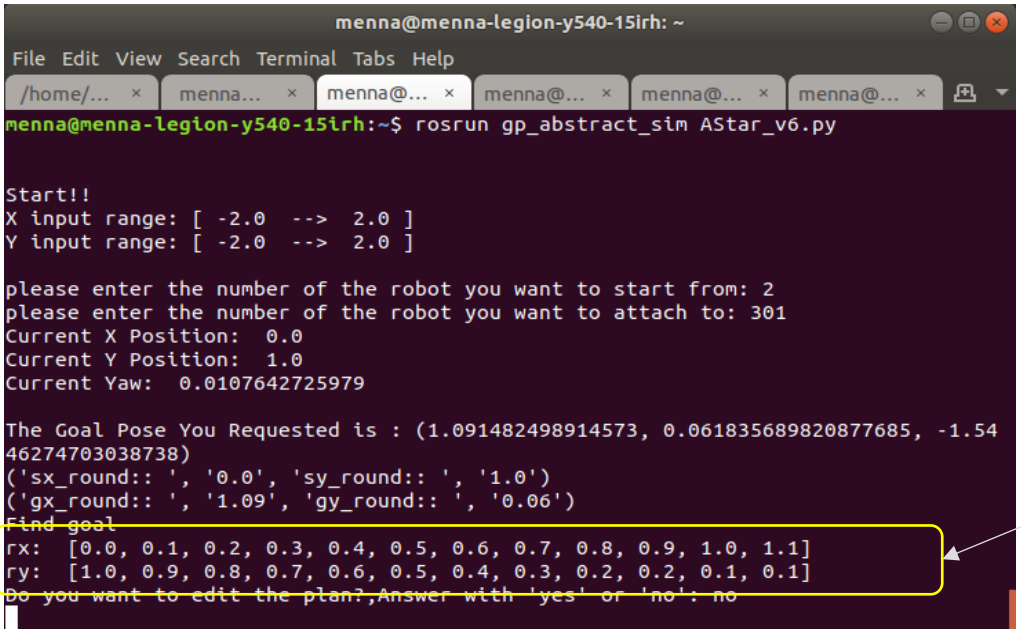


4.10.2.3 Last Phase

Ditto2 attaches to ditto3, ditto0, and ditto1:

4.10.2.3.1 Steps 3,4,5 are repeated

In step 3: The number of robots want to attach to is written from back to front, in this case it's 301



4.10.2.4 The Final Result

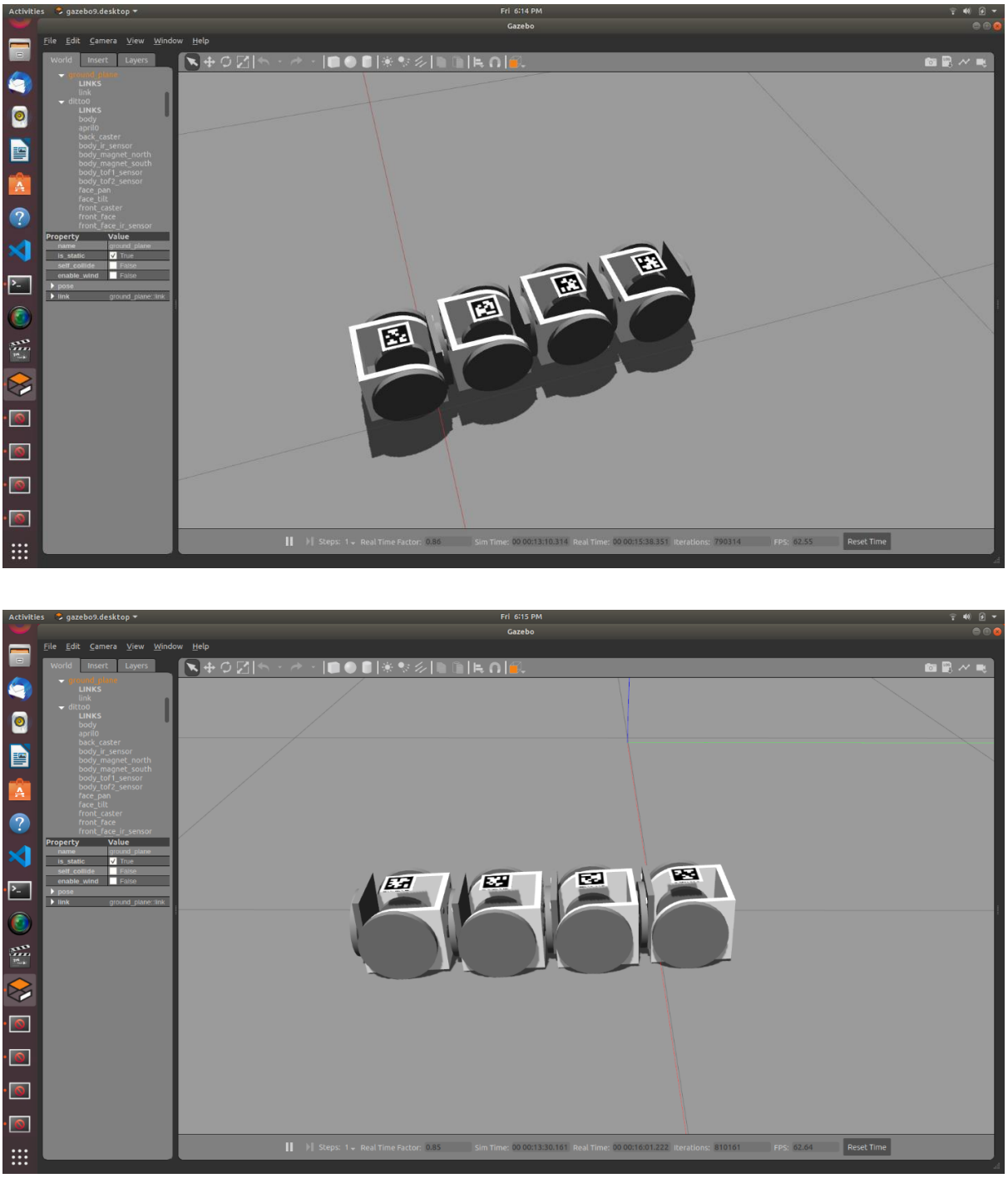


Figure 4.10.7 Experiment Two Final Result

4.10.2.5 Observation

There is a slight misalignment especially at the last attachment of ditto2 to ditto3, but since contact sensors are used instead of magnets so there is no self-alignment, which maybe the reason for this problem.

4.10.3 Experiment #3 “Docking Four MSR Robots at Random positions “

In this experiment we are going to dock **Four** robots at random positions.

4.10.3.1 The First Phase

4.10.3.1.1 Step (1)

To start the experiment type in terminal:

➔ Command: **rosrun gp_abstract_sim robotcams.launch**

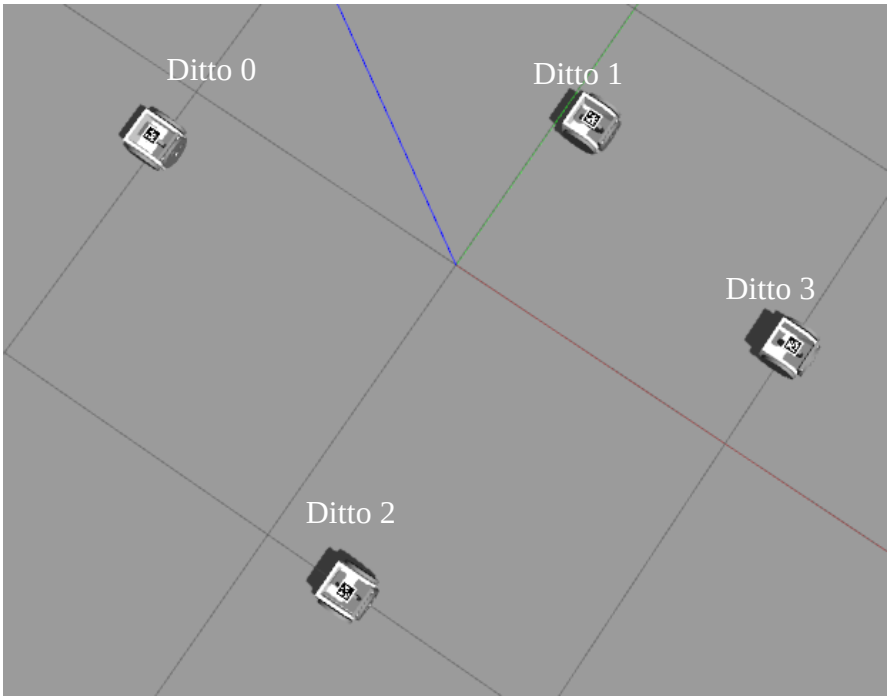


Figure 4.10.8 Experiment Three Setup

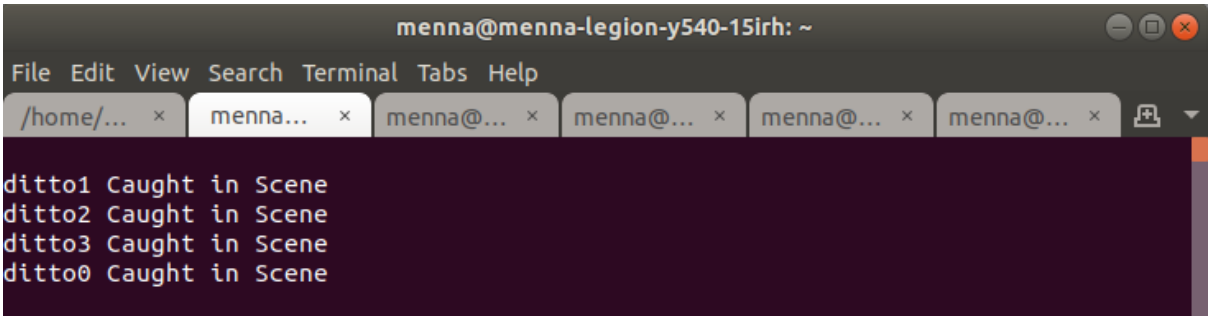
4.10.3.1.2 Step (2)

To determine the position and orientation of each robot, multiple_model_states.py is used

➔ Command: **rosrun gp_abstract_sim multiple_model_states.py**

```
menna@menna-legion-y540-15irh:~$ rosrun gp_abstract_sim multiple_models_states_v
2.py
```

Since there are 4 robots in the scene the output of the command is:

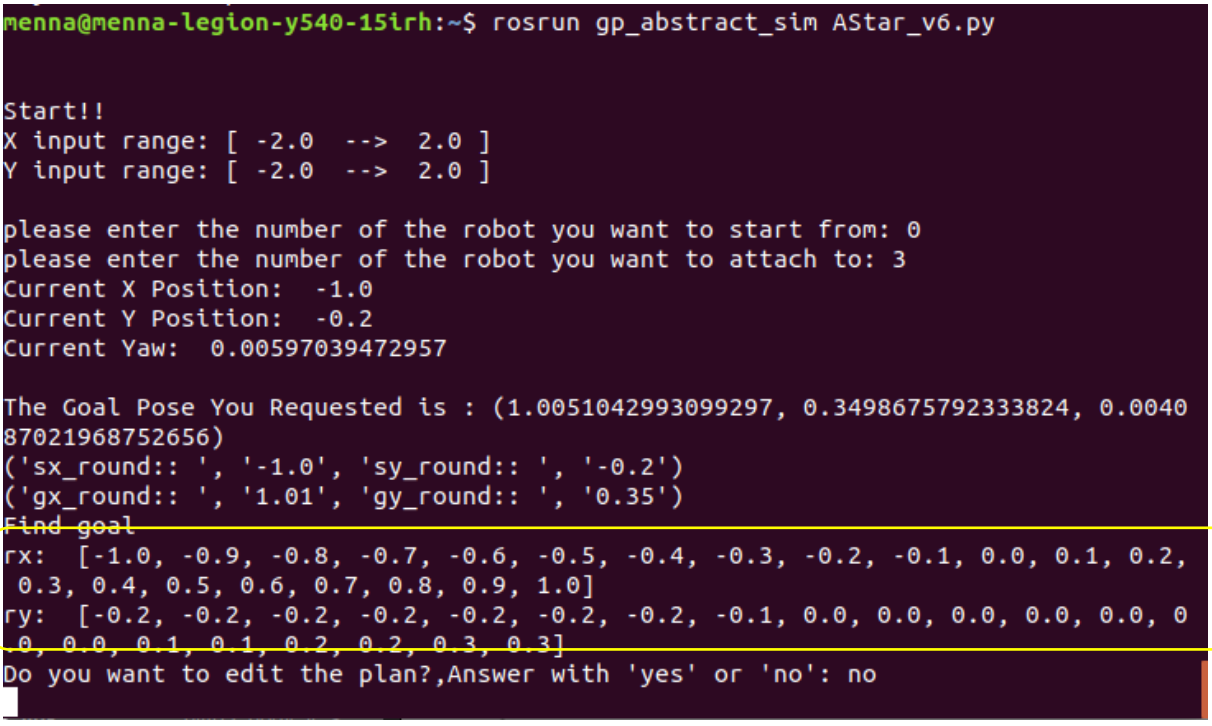


4.10.3.1.3 Step (3)

After determining the position and orientation of each robot, start path planning, A* path planner is used:

➔ Command : **rosrun gp_abstract_sim AStar_v6.py**

In this case the moving robot is “ditto0” to the goal robot “ditto3”



Path Points

4.10.3.1.4 Step (4)

To execute the movement of the robot**, trial_control.py** is used:

➔ Command: **rosrun gp_abstract_sim trial_control.py**

First, ditto3 moves forward 0.18 m, so that when ditto0 reaches the goal it doesn’t collide with it.

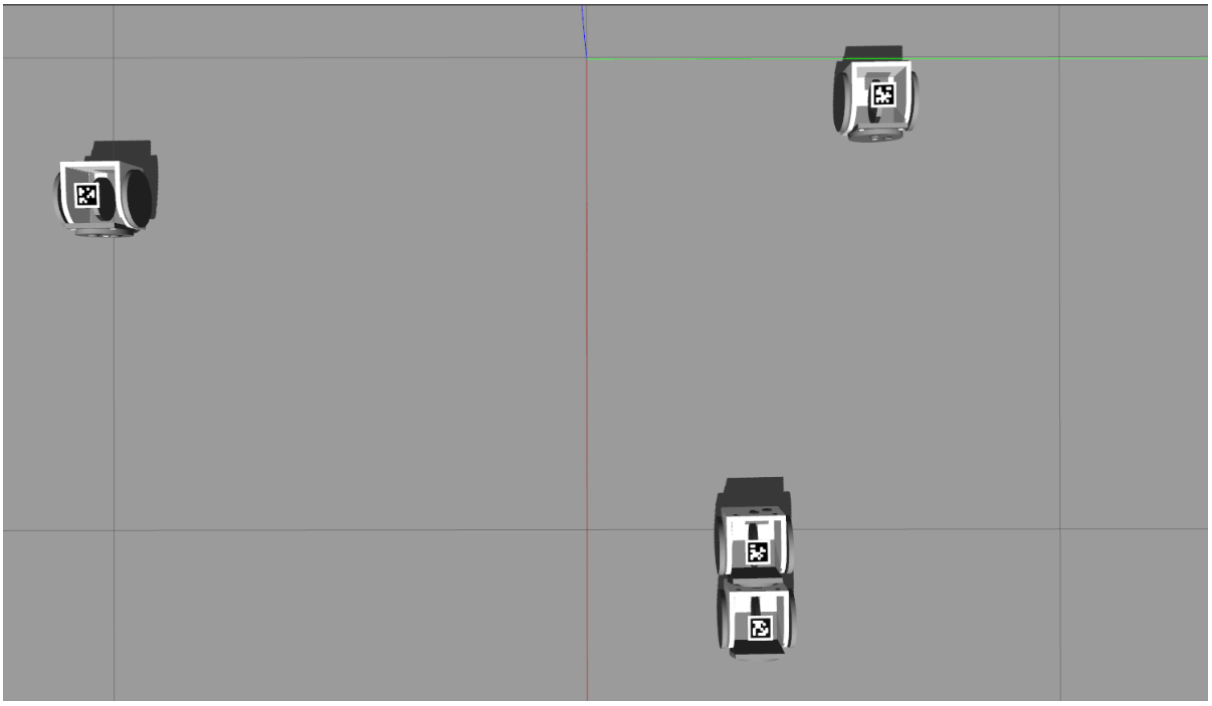
Second, ditto0 adjust its position and orientation at each point, then starts moving.

Since the back of ditto3 is facing the front face of ditto0 (moving robot), ditto3 will not rotate.

4.10.3.1.5 Step (5)

For a smooth docking after ditto0 reaches the destination, it moves forward with a very low speed 0.02 m/sec, until it attaches ditto3 via **magnetism.py**

➔ Command: **rosrun gp_abstract_sim magnetism.py**



4.10.3.2 The Second Phase ditto2 attaches to ditto0, and ditto3:

4.10.3.2.1 Steps 3,4,5 are repeated

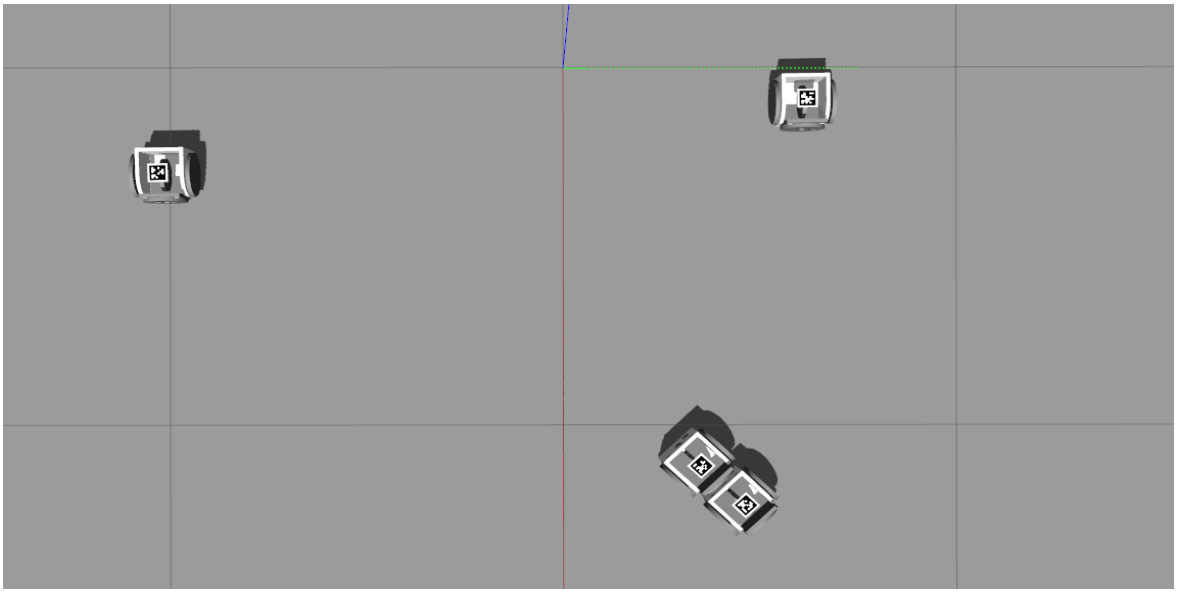
In step 3 : in order to attach a robot to a pile of robots, the number of robots is written from the back to the front robot → in this case the last robot is ditto0 while the first one is ditto3.

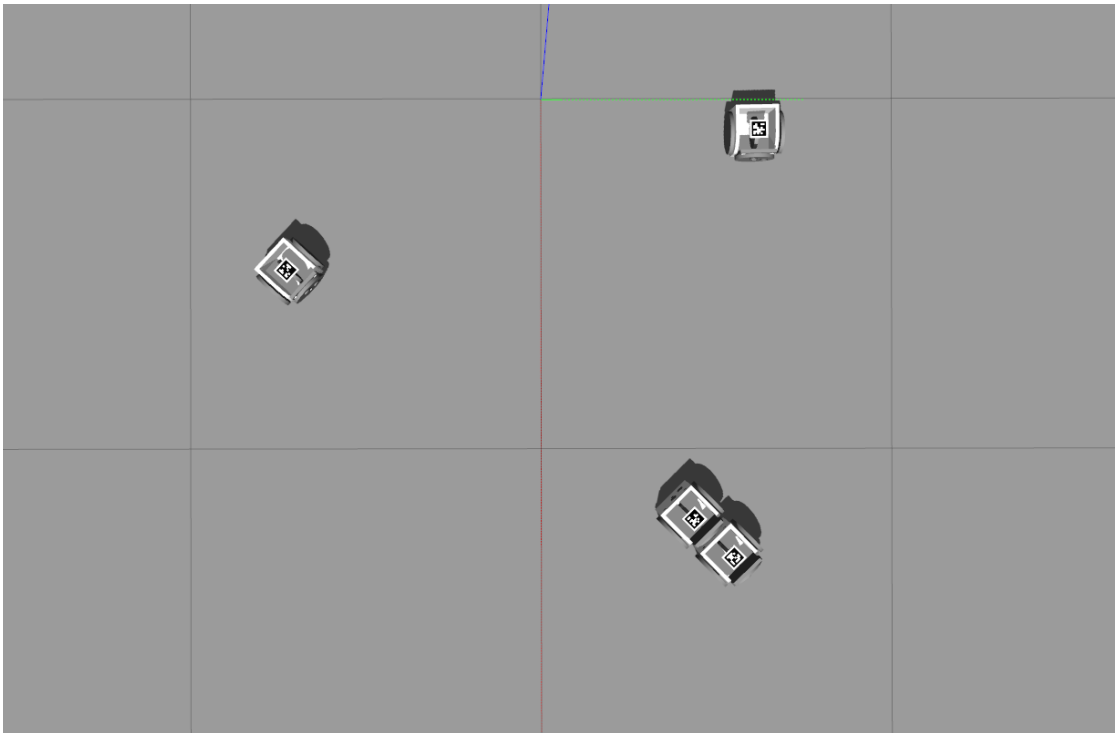
Therefore, the number of robot you want attach to is: 03

Path Point

```
menna@menna-legion-y540-15irh:~$ rosrun gp_abstract_sim AStar_v6.py


Start!!
X input range: [ -2.0  -->  2.0 ]
Y input range: [ -2.0  -->  2.0 ]

please enter the number of the robot you want to start from: 2
please enter the number of the robot you want to attach to: 03
Current X Position:  0.3
Current Y Position:  -1.0
Current Yaw:  0.0142935651601

The Goal Pose You Requested is : (1.02491369936018, 0.34558398979888605, 0.02011
8357131690946)
('sx_round:: ', '0.3', 'sy_round:: ', '-1.0')
('gx_round:: ', '1.02', 'gy_round:: ', '0.35')
Find goal
rx:  [0.3, 0.4, 0.4, 0.5, 0.6, 0.6, 0.7, 0.7, 0.8, 0.8, 0.9, 0.9, 0.9, 1.0]
ry:  [-1.0, -0.9, -0.8, -0.7, -0.6, -0.5, -0.4, -0.3, -0.2, -0.1, 0.0, 0.1, 0.2,
 0.3]
Do you want to edit the plan?,Answer with 'yes' or 'no': no
```

In step 4: ditto0, ditto3 rotate 45 degree, so that the back of ditto0 is facing the front face of ditto2

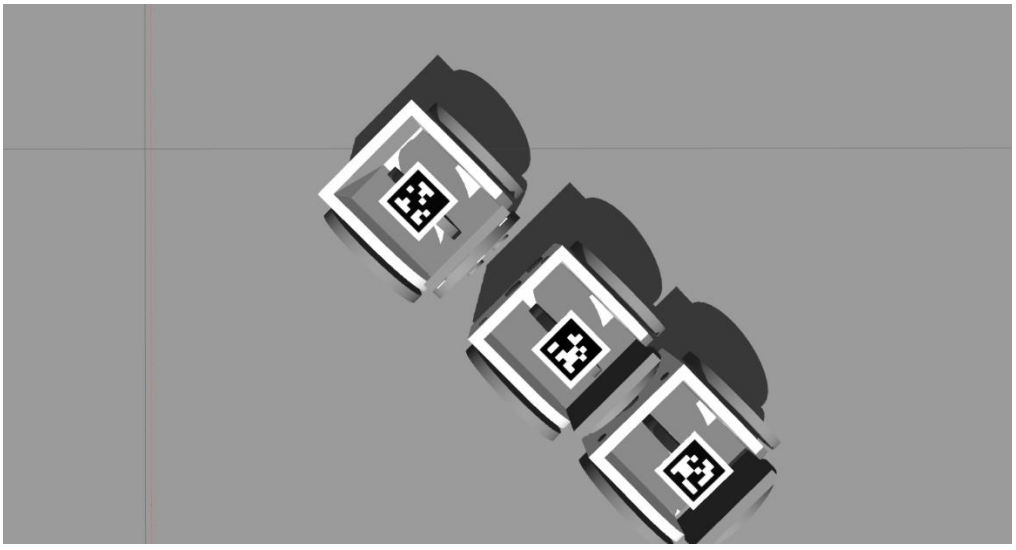


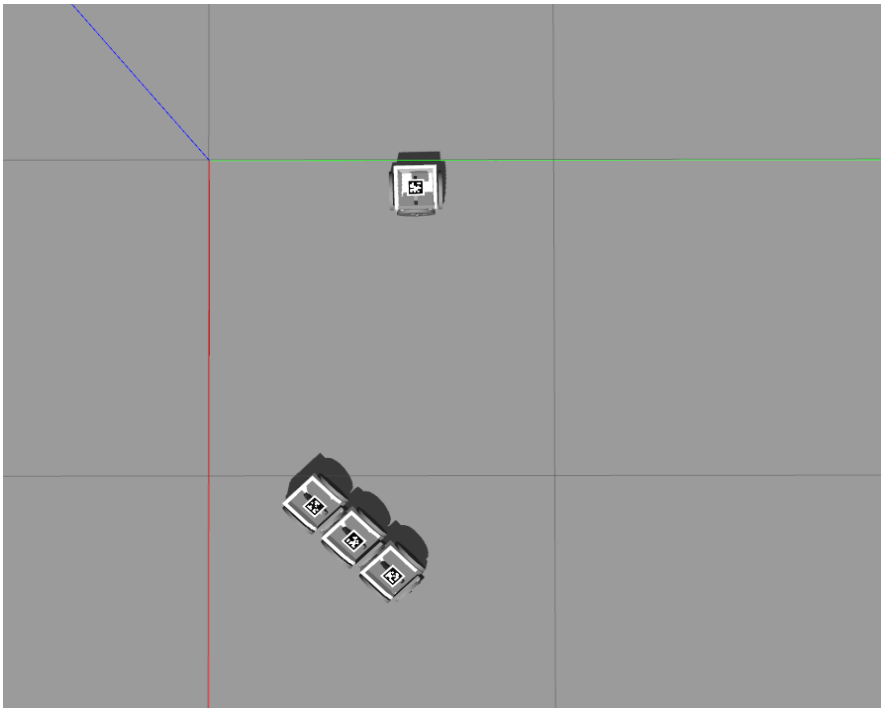


4.10.3.2.2 2nd time Step (5):

For a smooth docking after ditto2 reaches the destinantion, it moves forward with a very low speed 0.02, until it attaches to ditto0 via **magnetism.py**

➔ Command: **rosrun gp_abstract_sim magnetism.py**





4.10.3.3 The Last Phase

Ditto1 attaches to ditto2, ditto0, and ditto3:

4.10.3.3.1 Steps 3,4,5 are repeated

In step 3: The number of robots want to attach to is written from back to front, in this case it's 203.

```
menna@menna-legion-y540-15irh:~$ rosrun gp_abstract_sim AStar_v6.py


Start!!
X input range: [ -2.0  -->  2.0 ]
Y input range: [ -2.0  -->  2.0 ]

please enter the number of the robot you want to start from: 1
please enter the number of the robot you want to attach to: 203
Current X Position:  0.11
Current Y Position:  0.6
Current Yaw:  0.0253272580868

The Goal Pose You Requested is : (1.0834287326736898, 0.31348896506238294, 0.786
5036978417573)
('sx_round:: ', '0.11', 'sy_round:: ', '0.6')
('gx_round:: ', '1.08', 'gy_round:: ', '0.31')
Find goal
rx:  [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0, 1.1]
ry:  [0.6, 0.6, 0.6, 0.6, 0.5, 0.4, 0.3, 0.3, 0.3, 0.3, 0.3]
Do you want to edit the plan?,Answer with 'yes' or 'no': no
```

Path Points

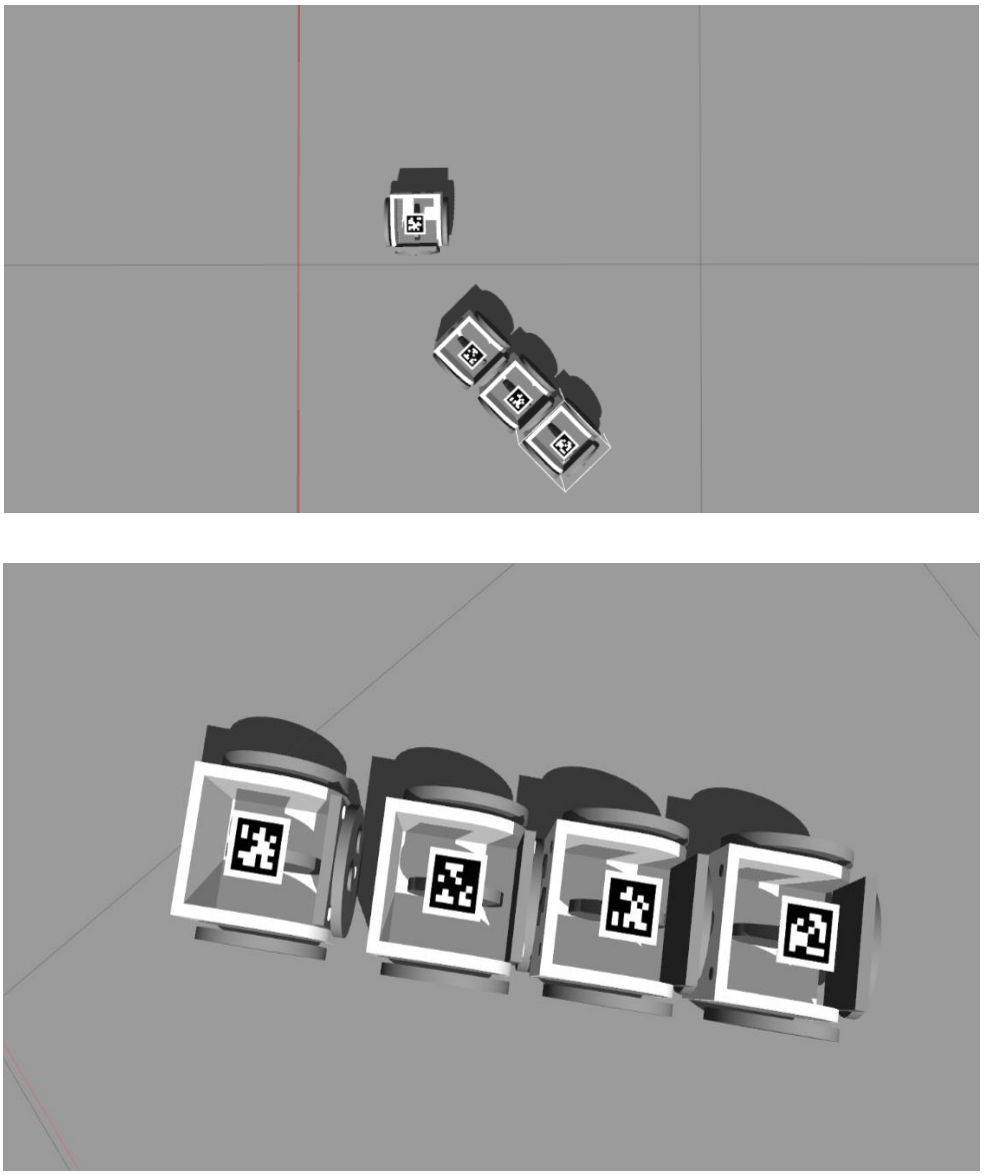


Figure 4.10.9 View Before Last Robot Attaching

4.10.3.4 The Final Result

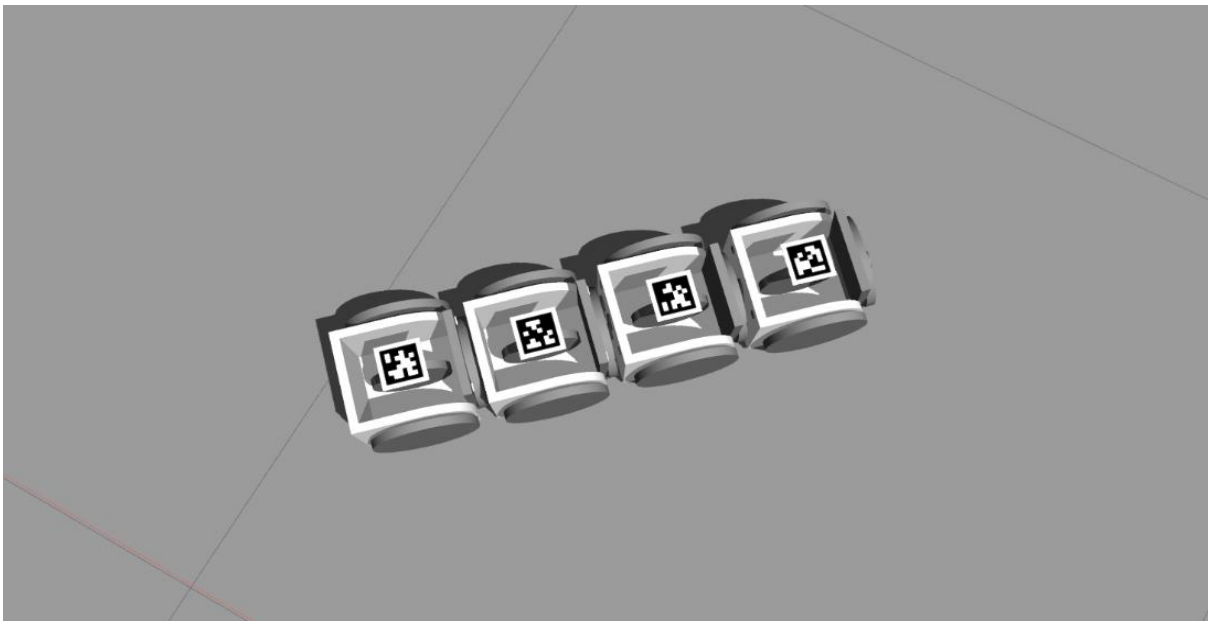


Figure 4.10.10 Experiment Final Result

4.10.3.5 Observation

The first issue was at the last docking of ditto1 to ditto2,0,3, ditto1 rotates extra cycle because when it reaches the last point its orientation was different than the orientation of ditto2.

The second issue is that ditto1 rotates in CCW instead of CW which leads that it rotates fully cycle in CCW direction instead of rotating a small angle in the CW direction.

Shockingly, the misalignment of the faces of each robot is really small and cannot be seen!

4.11 The MSSR Application

We have our own dedicated application that is made from scratch. This application helps us:

- Control
- Navigate
- Monitor all the robots and sensors in the scene

Through the application, we can control every actuator of every robot in the scene. Even the front faces of each robot, which is capable of panning and tilting, can be accurately controlled with the desired angles and velocities.

We can also monitor every robot state in the scene with a numerical and graphical representation. Multiple graphs can be viewed simultaneously.

4.11.1 Application's User Guide

Before running the application, you need to start the gazebo world and spawn some robots(physical control is not implemented yet.) You also need to spawn the camera and run the script **web_video_server** from the package **web_video_server** for handling the camera stream.

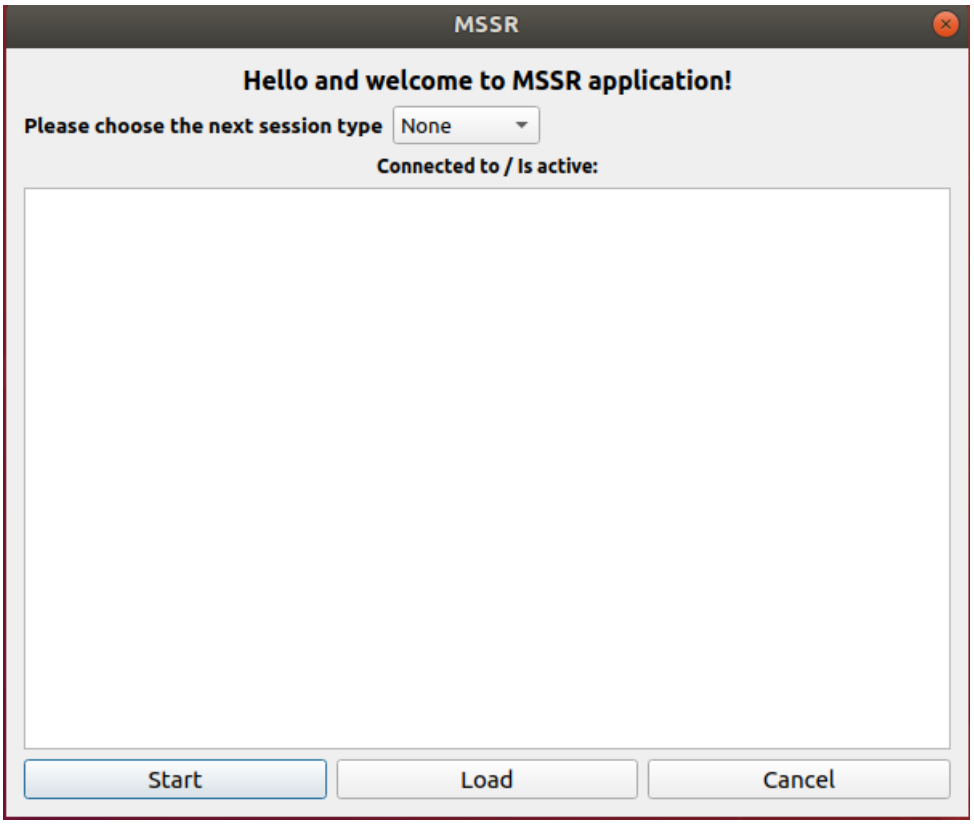
```
rosrun web_video_server web_video_server
```

Next, you need to run **multiple_models_states_class.py** from **gp_abstract_sim** package to be able to get the position and orientation of the robots to be able to control them.

```
menna@menna-legion-y540-15irh:/$ rosrun gp_abstract_sim multiple_models_states_c
lass.py
```

Now that the environment is properly set, you start by sourcing your **catkin_ws3** workspace for this app. The application itself is written in Python 3 and on a different workspace than the other packages.

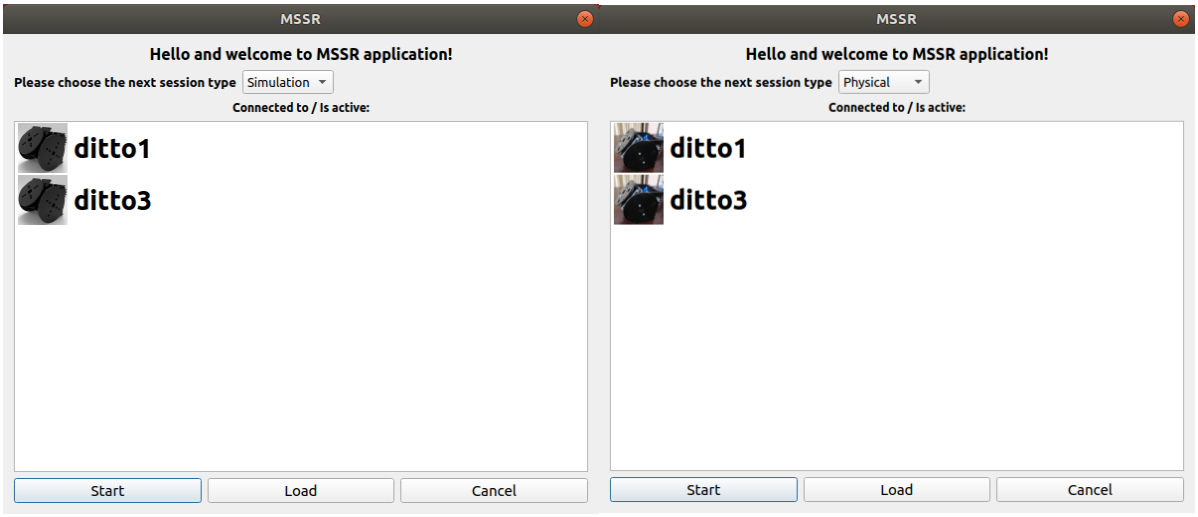
After sourcing the workspace, run the command: **rosrun qt_app dialog.py**



A dialog like this should appear if all requirements are met (you installed the required packages from the git), and you followed the above instructions.

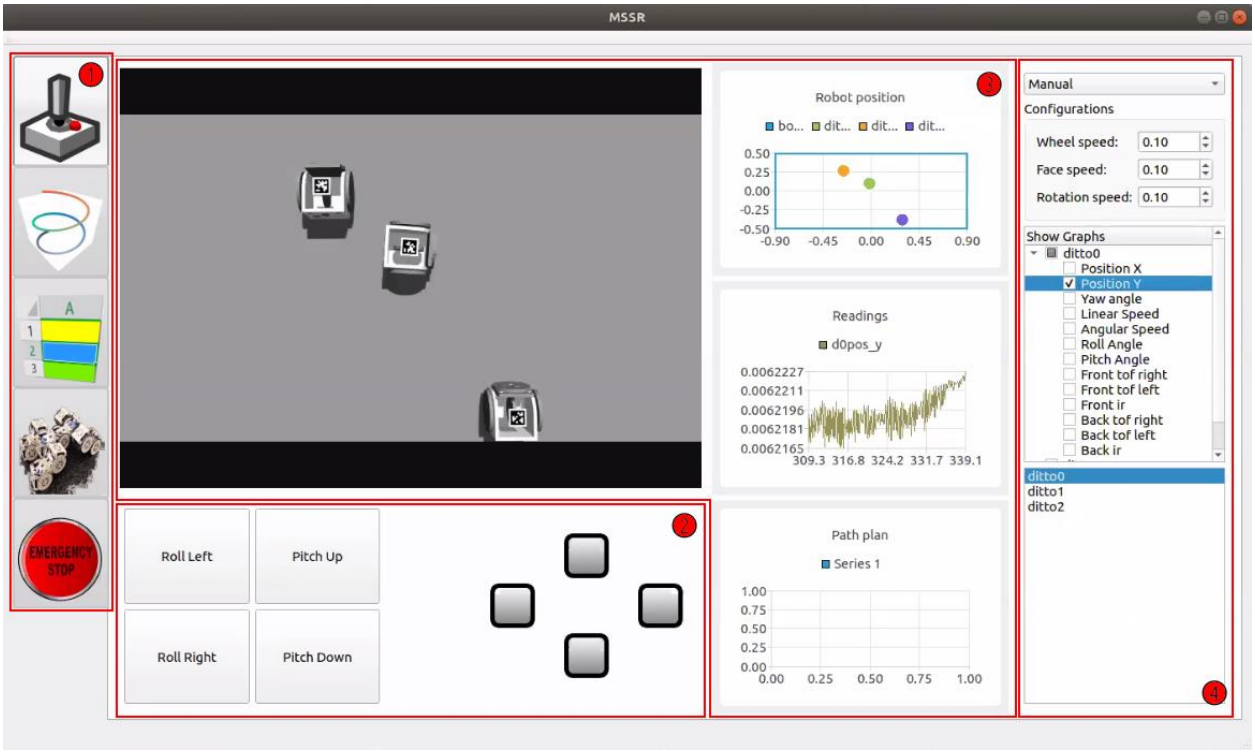
This dialog provides you with three choices from the dropdown menu you can choose to either start a ‘simulation’ session, or a ‘physical’ one, or you could choose another complete approach and choose the ‘None’ and ‘Load’ buttons to review an old session. Due to the current unfortunate events, we only implemented the simulation option.

Once you choose the ‘simulation’ option all your robots that are scattered around the scene should appear under the **connected to / is active label**. It is another way to double check if all robots are connected successfully and read by our application successfully.

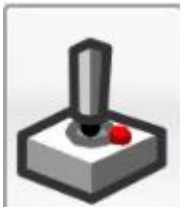


Now, press the ‘Start’ button.

After pressing the start button, the main application window should appear



4.11.1.1 The Vertical Bar on the Left

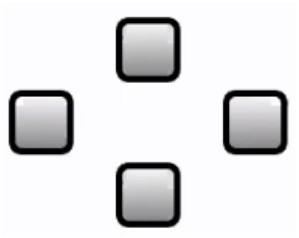
The Main Page: It contains all the main functionalities of the application	
The Monitoring Page: The page that shows the graphs for out robots' states with respect to time	
The Sensors' Log: The page with tables filled by all our sensory data	

The Formations Page: (Not implemented yet) This page contains predefined configurations to be assembled by the push of a button. Hahaha, that is why it is not implemented. We are not that good.	
The Red Button: (Not implemented yet) As it is very obvious, this emergency button stops the whole simulation process.	

4.11.1.2 The Not so Joyful Joystick

These buttons control the front face of the robot. Roll right and Roll left control the panning of the face, while Pitch up and Pitch down control the tilting movements.

You could also use the keys ‘R,F,T,G’ for the same effect as pressing the buttons with ‘R’ being assigned to roll left, ‘F’ for roll right, ‘T’ for pitch up, and ‘G’ for pitch down button.

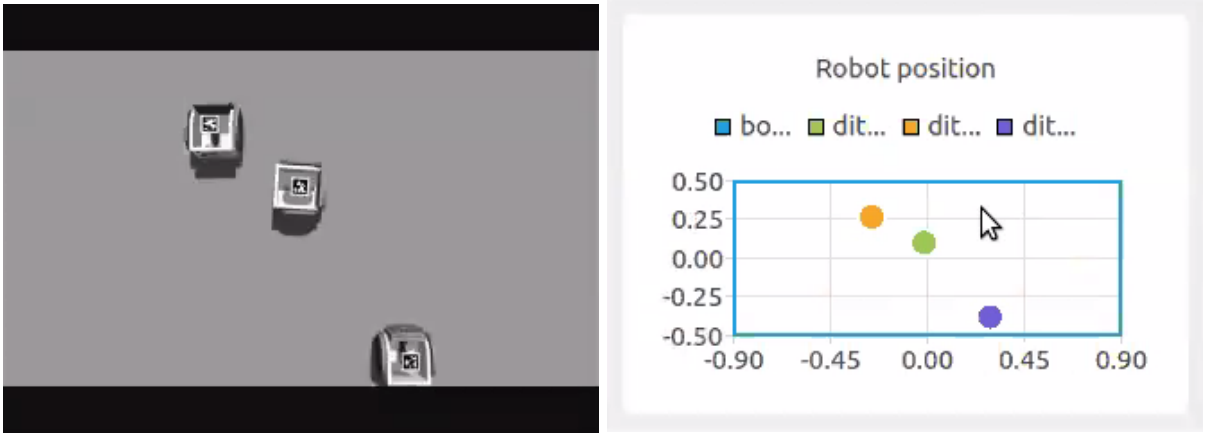


The above buttons are for moving the robot around. The upper and lower buttons control the forward and backward movements of the robot. The buttons on the sides control the robot’s rotation around its axis, left or right.

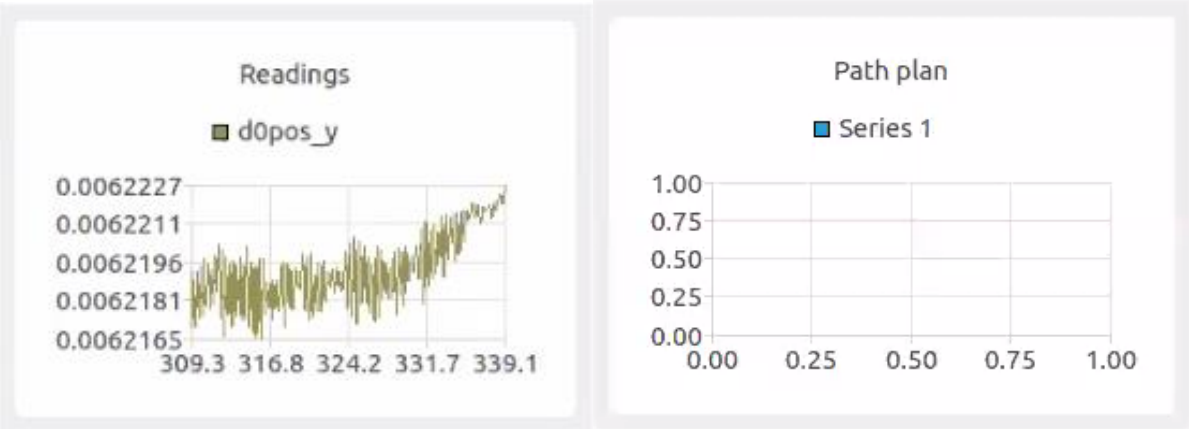
These buttons, also, have keys assigned to them ‘W’ being forward, ‘A’ for left, ‘S’ for backward, and ‘D’ for right button.

From this menu, you can easily choose the robot you wish to control. You can also choose multiple robots at the same time to control by pressing **control key** and choosing the robots you want them operating and following the exact same commands.

4.11.1.3 Graphs and Camera Stream

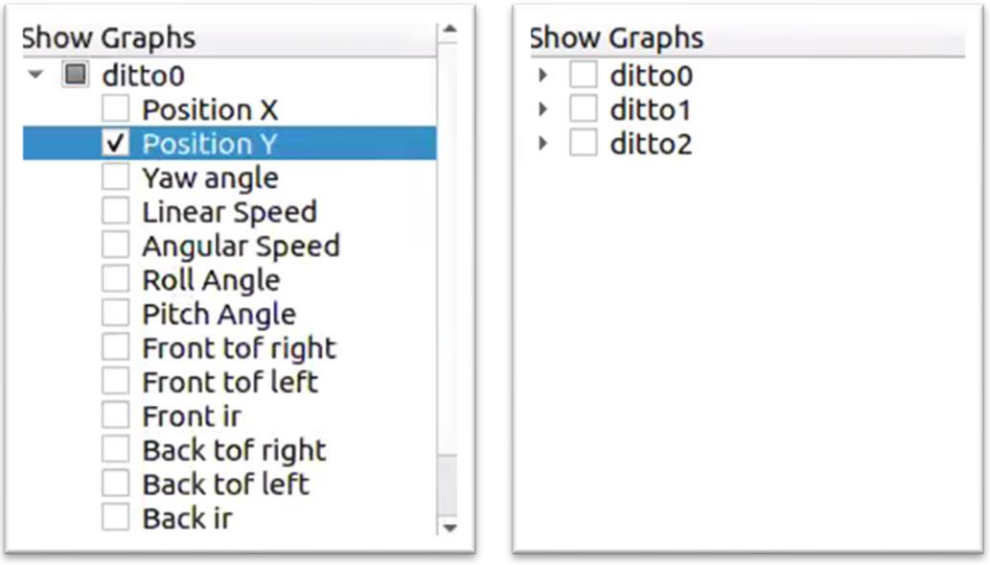


As each graph’s suggests, robot position graph represents the position of the robots within the boundaries of our camera. It may seem redundant to the camera stream, but actually this provides us with what our robots are seeing as each robot is seen as a point or if it is not even figured out correctly by our tag detection.

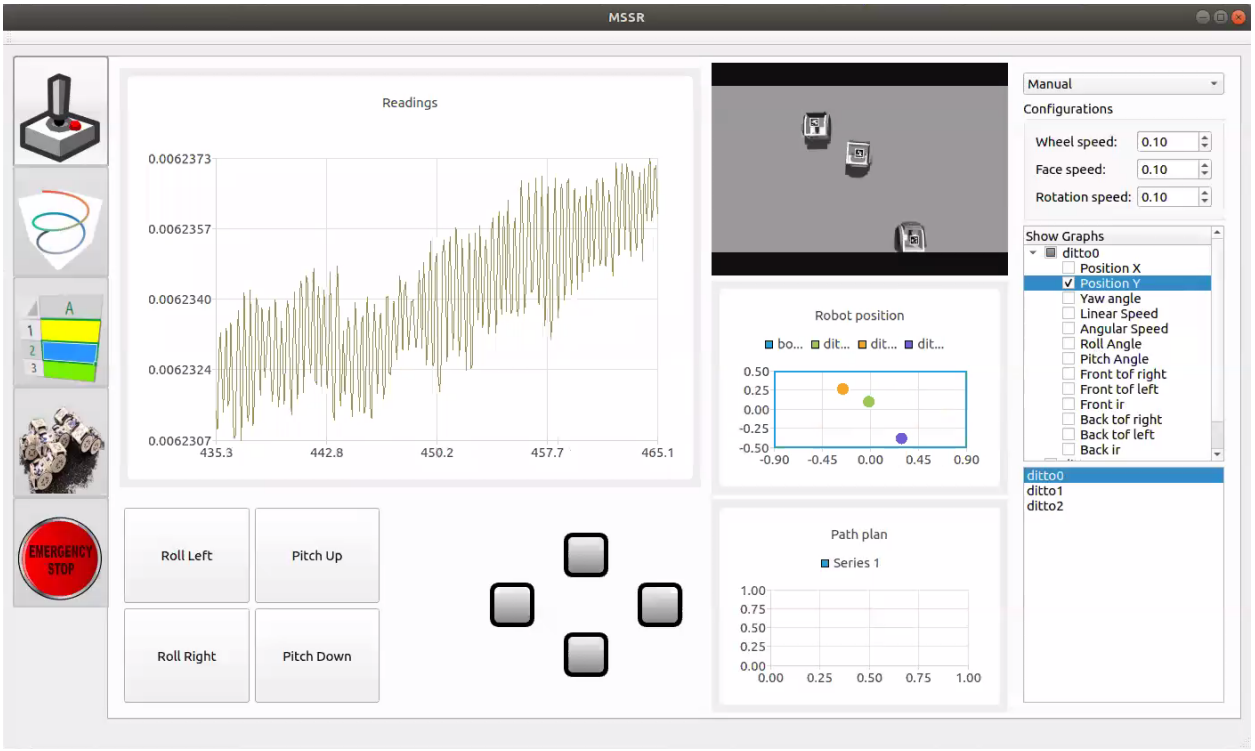


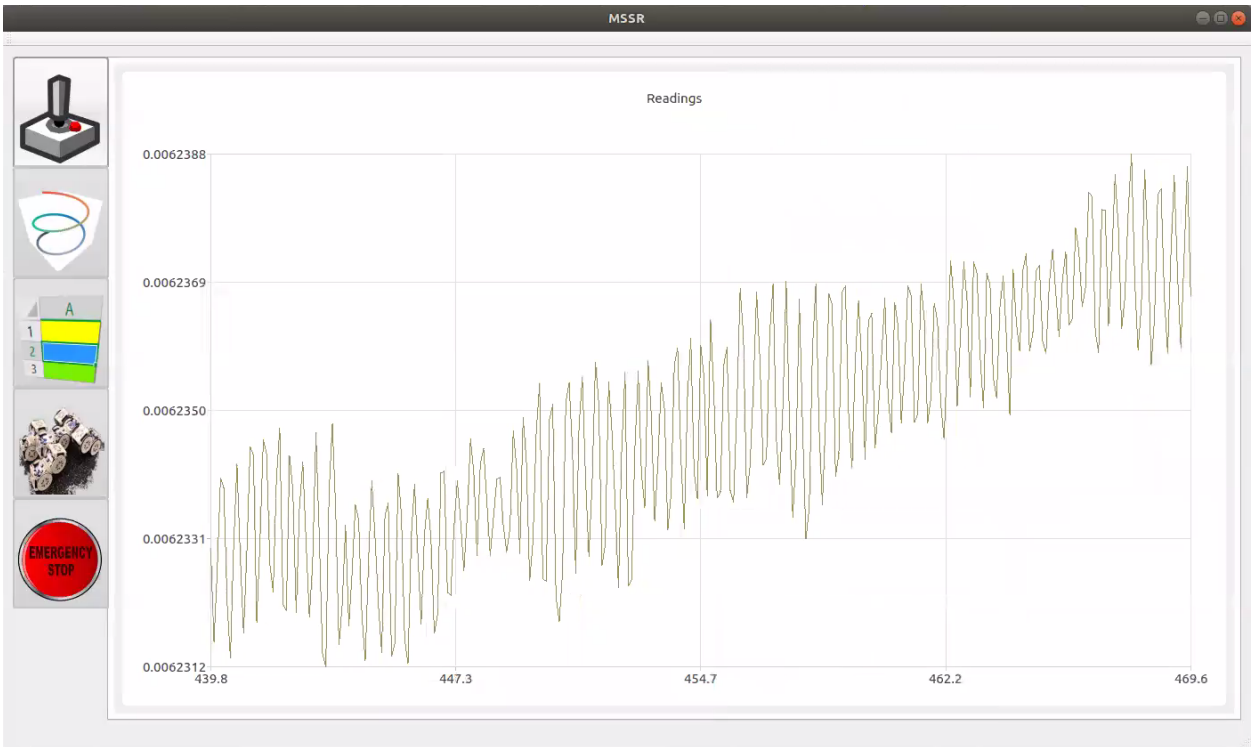
Next comes the Readings graph, where you can choose from ‘show graphs’ menu which sensor data you want to trace with respect to time. You just must pick the robot you want to track and which piece of information you want plotted.

Path plan graph was meant to provide the path to be taken by our robot after running the A* algorithm (not implemented).



You can click on any of the graphs to put it in an enlarged view to give you better view, or you could if it was already in the enlarged area click on it again to give it the full view click again and it will shrink again to the last size.

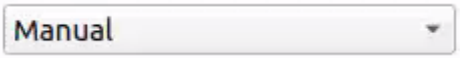




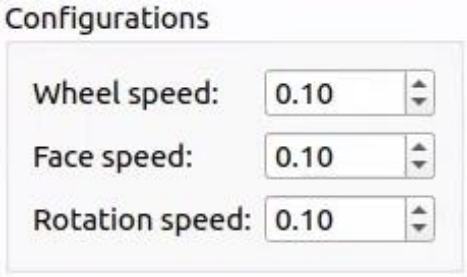
But due to a bug in the Image viewer you have to drag and drop the image in place for same effect (to put the view in the enlarged area or to give it full view.)

4.11.1.4 Important configurations

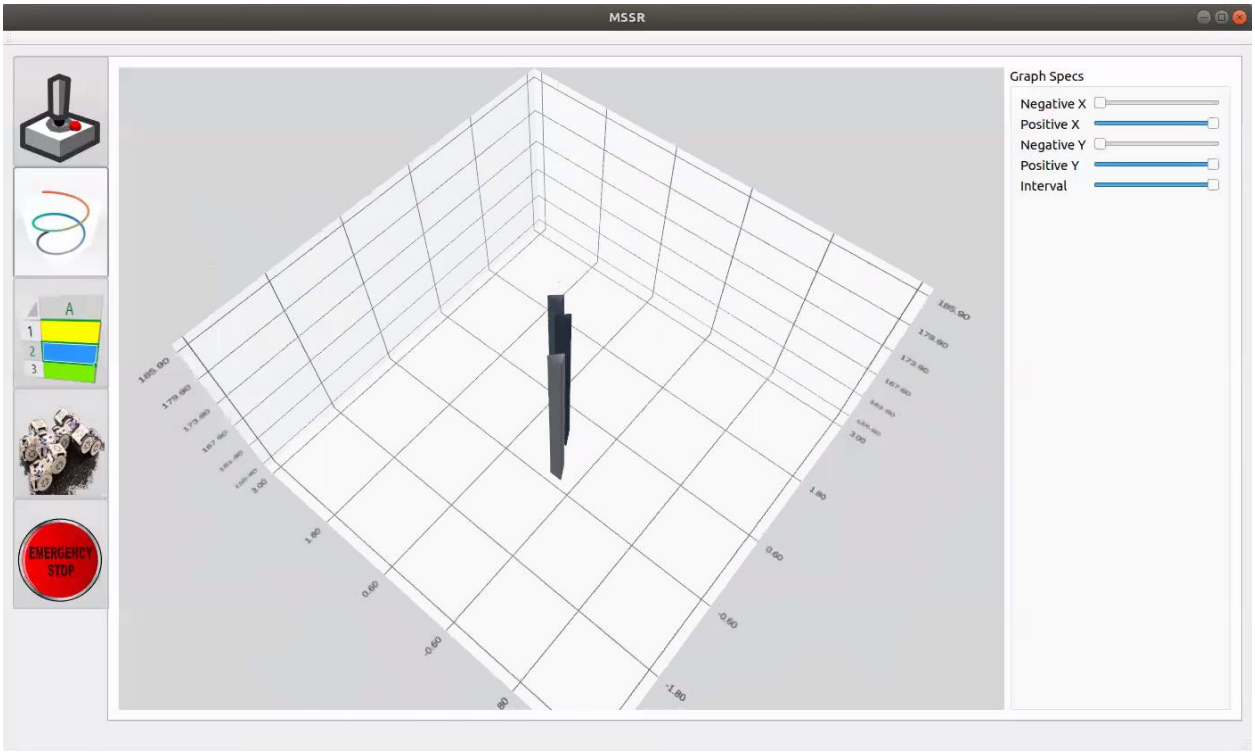
You can choose between two operating modes manual were you control the robots yourself and automatic were you can tell them were to go and what to do(not functional).



You can increase or decrease the speed of the robot by adjusting the wheel speed, the rate of the robot's rotation around it's center from rotation speed, and the speed at which the robot tilts and pans it's front face by changing the face speed with max speed of 1.0 m/s and min 0 m/s.



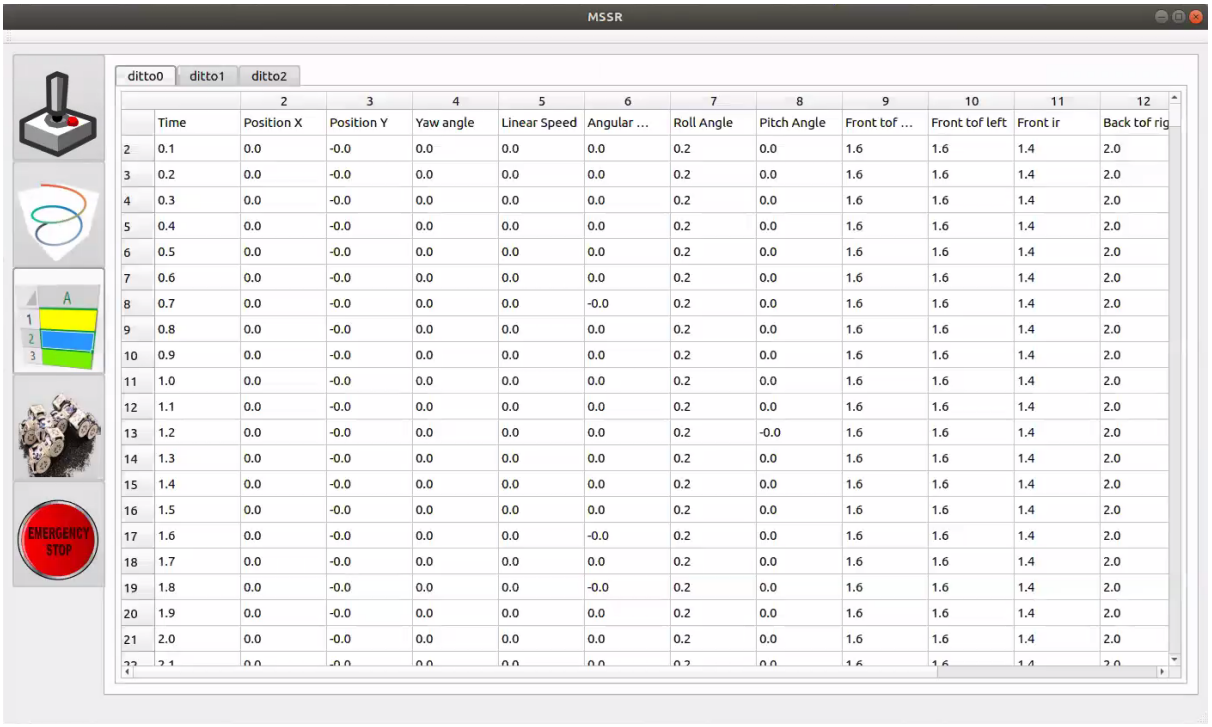
4.11.1.5 The Second Page: Monitoring Page



This page consists of 3d graph plotting the position and orientation of the robots found in the scene with the pointer pointing to where the front face of our robots are pointing to, the graph plots dimensions are the X position and Y position with Time being the third axis.

The scroll bars positioned in the left control the area covered by the graph where you can increase or decrease it to concentrate on specific robot(s), or to cover the whole space. It also controls how much tail effect by increasing or decreasing the interval as it shows older positions of the robots.

4.11.1.6 The Third Page: Sensors' Log



	Time	Position X	Position Y	Yaw angle	Linear Speed	Angular …	Roll Angle	Pitch Angle	Front tof …	Front tof left	Front ir	Back tof rig
2	0.1	0.0	-0.0	0.0	0.0	0.0	0.2	0.0	1.6	1.6	1.4	2.0
3	0.2	0.0	-0.0	0.0	0.0	0.0	0.2	0.0	1.6	1.6	1.4	2.0
4	0.3	0.0	-0.0	0.0	0.0	0.0	0.2	0.0	1.6	1.6	1.4	2.0
5	0.4	0.0	-0.0	0.0	0.0	0.0	0.2	0.0	1.6	1.6	1.4	2.0
6	0.5	0.0	-0.0	0.0	0.0	0.0	0.2	0.0	1.6	1.6	1.4	2.0
7	0.6	0.0	-0.0	0.0	0.0	0.0	0.2	0.0	1.6	1.6	1.4	2.0
8	0.7	0.0	-0.0	0.0	0.0	-0.0	0.2	0.0	1.6	1.6	1.4	2.0
9	0.8	0.0	-0.0	0.0	0.0	0.0	0.2	0.0	1.6	1.6	1.4	2.0
10	0.9	0.0	-0.0	0.0	0.0	0.0	0.2	0.0	1.6	1.6	1.4	2.0
11	1.0	0.0	-0.0	0.0	0.0	0.0	0.2	0.0	1.6	1.6	1.4	2.0
12	1.1	0.0	-0.0	0.0	0.0	0.0	0.2	0.0	1.6	1.6	1.4	2.0
13	1.2	0.0	-0.0	0.0	0.0	0.0	0.2	-0.0	1.6	1.6	1.4	2.0
14	1.3	0.0	-0.0	0.0	0.0	0.0	0.2	0.0	1.6	1.6	1.4	2.0
15	1.4	0.0	-0.0	0.0	0.0	0.0	0.2	0.0	1.6	1.6	1.4	2.0
16	1.5	0.0	-0.0	0.0	0.0	0.0	0.2	0.0	1.6	1.6	1.4	2.0
17	1.6	0.0	-0.0	0.0	0.0	-0.0	0.2	0.0	1.6	1.6	1.4	2.0
18	1.7	0.0	-0.0	0.0	0.0	0.0	0.2	0.0	1.6	1.6	1.4	2.0
19	1.8	0.0	-0.0	0.0	0.0	-0.0	0.2	0.0	1.6	1.6	1.4	2.0
20	1.9	0.0	-0.0	0.0	0.0	0.0	0.2	0.0	1.6	1.6	1.4	2.0
21	2.0	0.0	-0.0	0.0	0.0	0.0	0.2	0.0	1.6	1.6	1.4	2.0

In this page we a table with all the sensory data from all robots with respect to the time it was taken.

CHAPTER FIVE : ELECTRICAL DESIGN

Electrical components are always known to be the heart of any system, the link that connects the mechanical component to the software. And as the electrical has a very important role in the robot we did not take it any lightly especially that the electrical components in our reconfigurable robot are required to be significantly smaller and less in number as possible to fit the small size of the robot.

5.1 Component selection

As mentioned above size was a key factor in choosing most of our components as well as the weight of these components such that it will affect the robot dynamics.

5.1.1 STM32 (Black Pill)

It's the brain of the robot it is the low-level microcontroller that controls the robot.

We were mainly looking for small sized microcontrollers which made us eliminate Tiva C and Arduino UNO microcontrollers. We were comparing between Arduino Nano, STM32 and ESP 32. However, the STM32 was the one we chose as it had the greatest number of GPIO pins and four timers that were important for the PID control of the motor and for the beacon sensor.

Specification

- Core: Cortex-M3 32-bit
- Operating frequency: 72MHz
- Storage resources: 64K Byte Flash, 20KByte SRAM
- Interface Resources: 2x SPI, 3x USART, 2x I2C, 1x CAN, 37x I / O ports
- Analog-to-digital conversion: ADC (12-bit / 16-channel)
- PWM: 16-bit/15 channel
- Four Timers

5.1.2 ESP-01

It is a WIFI module that would be used to allow the communication between the modules and each other and between the user laptop and the modules to send commands. This specific WIFI module was chosen for its small size and cheap price compared to other modules providing the same functionality also this module is brought from Egypt making it easy to be obtained when needed.

Specification

Frequency range: 2.4GHz-2.5GHz (2400M-2483.5M)

Data interface: UART / HSPI / I2C / I2S / Ir Remote Contorl GPIO / PWM

Operating temperature: -40 ° ~ 125 °

5.1.3 TIME OF FLIGHT SENSOR (TOF)

This is a specific IR sensor that measures distance using the time of flight principle (ToF) that measure the time difference between the emission of a signal and its return and using the frequency it calculates the distance. This sensor is chosen obviously for its really small size making it possible to be placed on the outside frame of the robot.

The main purpose of this sensor is to proof the autonomous concept of the reconfigurable robots it is used alongside to the IR sensor to get the distance required by each robot to move to dock with the other robot or even help correct the orientation of the robot in case they are not aligned.

It is done by placing two ToF sensor placed on one face. These sensors are placed on the same horizontal line and each one of them will measure the distance and using the difference between the distances we get the angle the robot is inclined with and this angle is sent to the opposite robot to correct its orientation.

Specification

- Fully integrated miniature module
- 940nm Laser VCSEL
- VCSEL driver
- Ranging sensor with advanced embedded micro controller
- Fast, accurate distance ranging
- Measures absolute range up to 2m
- Reported range is independent of the target reflectance
- Operates in high infrared ambient light levels
- Easy integration– Single reflowable component
- No additional optics
- Single power supply
- I2C interface for device control and data transfer
- Xshutdown (Reset) and interrupt GPIO
- Programmable I2C address

5.1.4 INFRA-RED SENSOR (IR)

This component is also used for the autonomous concept of the robot by identifying which face of the robot the other robot is looking at.

It does so by using the beacon concept but instead of using radio waves we use the frequency of the IR blinking.

It will identify the face as each face will have an IR sensor which will emit a specific frequency to that face such that when the robot needs to dock with a specific face that face will start emitting that frequency to make it easy to the other robot to identify it.

Specification

- Detector type: phototransistor
- Peak operating distance: 2.5 mm
- Daylight blocking filter
- Emitter wavelength: 950 nm

5.1.5 MOTOR DRIVE (l293B)

Chosen based on the required current of the motor as well as the peak current. We used a 4-channel driver to drive four of our five motors.

Specification

- Contains four half H-Bridges that can operate as two full H-Bridges
- Operate 2 motors with direction and speed control
- Automatic Thermal shutdown is available
- Max peak current is 2A per channel

5.1.6 BATTERY

The battery was no difference to the rest of the components the size is a critical point to the selection not only that but the weight too was the main issue that's why we went with the lithium polymer option as it is much lighter in weight compared to the other batteries available in the market.

With the specific battery, we chose it since it contained both overcharging and overcurrent protection saving us the hustle of making our own protection circuits.

Specification

- 7.4V 2-cell pack
- 1000mAh of charge

- 25C continuous discharge rate

5.1.7 MOTOR

The motors used are dc motors with gearbox as they provide both high torque and small size making it the perfect option for us.

The specific motors chosen came with their encoders attached.

5.2 Battery sizing

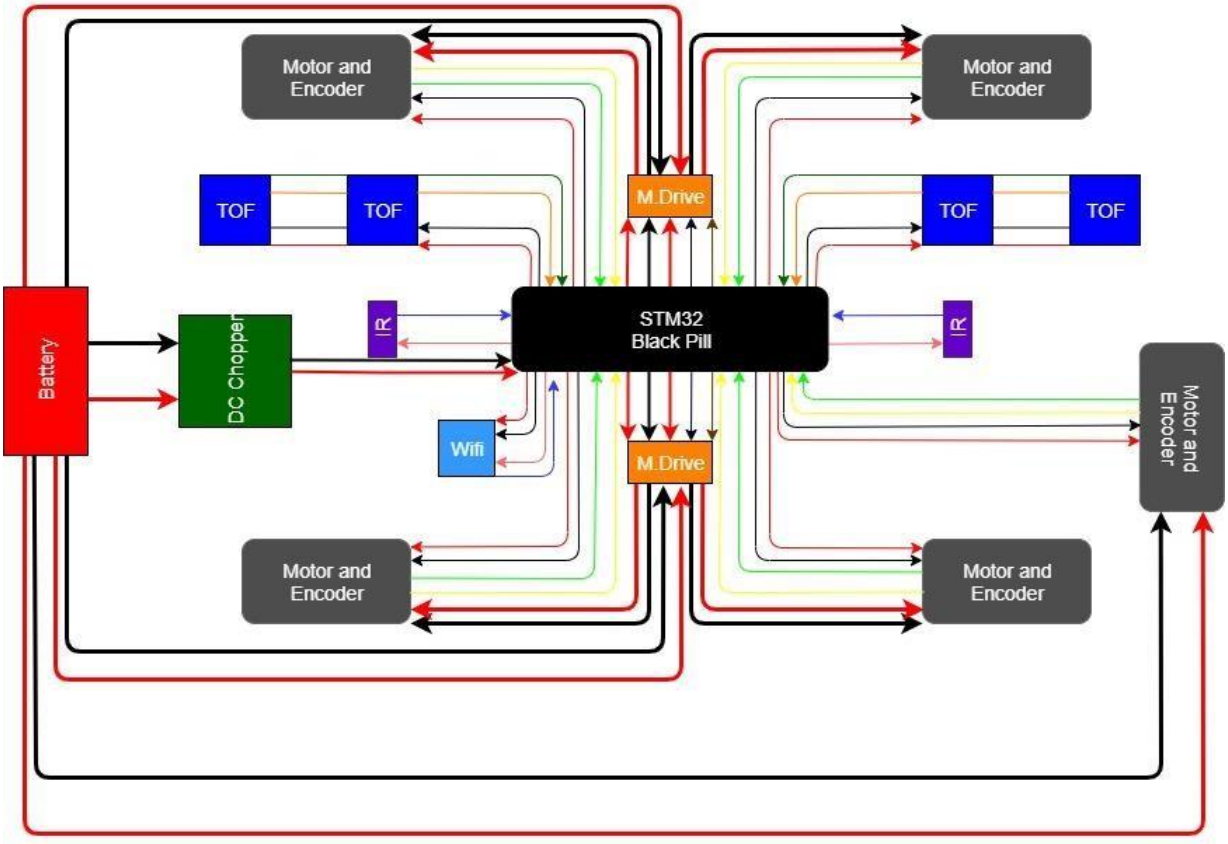
Component	Quantity	Current	Voltage	Power
STM32	1	0.15	5	0.75
ESP-01	1	0.17	3.3	0.561
DC Motor	5	0.55	6	16.5
Encoder	5	0.06	3.3	0.99
IR	4	0.06	5	1.2
ToF Sensor	4	0.06	3.3	0.792
Total power				20.793
Total current				2.809865
Supply voltage				7.4
Required time in hrs				0.4
Capacity				**1123.946**

Table 5.2.1 Battery sizing

In order to calculate the required battery size, we calculated the power consumption of each component and then summed up the total power consumption and divided it by the voltage of the battery required to get the capacity. To run for 24 minutes, the capacity required was

1123.946mAh but since our biggest limitation was the size, and we founda battery of 1000mAh capacity that fit all our size constrains we decided to compromise and use the smaller capacity.

5.3 System Interconnected Diagram



Line	Representation
	GROUND
	7.4V
	5V
	3.3V
	TX
	RX
	ENC1
	ENC2
	SDA
	SCL
	SIGNAL1
	SIGNAL2

Figure 5.2.1 SID

The above diagram shows the system diagram of all the electrical components and how they are connected in our system wiring.

Overall, we have the following components in each robot:

- STM32 controller

- Wi-Fi module
- 4 ToF sensors
- 2 IR sensors
- 5 motors with encoders
- 2 motor drivers
- 5v DC chopper
- A 7.4v Battery

5.4 Electromagnetic shielding

What is electromagnetic interference?

Before going into details with what the shielding in our robots is we need to address first what is electromagnetic interference.

It is a disturbance generated by an external source that affects an electrical circuit. The disturbance may degrade the performance of the circuit or even stop it from functioning. In the case of a data path, these effects can range from an increase in error rate to a total loss of the data.

In our case the external source are the magnets used in the docking of the robots and they are mainly affecting the motor encoders and the microcontroller making it an important issue to address and prevent.

Electromagnetic shielding is the practice of reducing the electromagnetic field in a space by blocking the field with barriers made of conductive or magnetic materials. Shielding is typically applied to enclosures to isolate electrical devices from the 'outside world'.

We had several options to choose which one is most compatible with us.

Firstly, and the most known one are faraday cage a conductive enclosure used to block electrostatic fields. The amount of reduction depends very much upon the material used, its

thickness, the size of the shielded volume and the frequency of the fields of interest and the size, shape and orientation of apertures in a shield to an incident electromagnetic field.

As this is the mostly used way however it's not applicable with our mechanism as it will affect the robot's movement also the size and weight is not compatible.

Secondly, we thought of placing a steel metal sheet behind the magnets to absorb the electromagnetic fields this is a good option for us as ease of implementation however, there is still other options much easier and less costly than the metal sheet.

Thirdly, upon searching we found a metallic ink that could be sprayed behind the walls of the magnets. The ink consists of a carrier material loaded with a suitable metal, typically copper or nickel, in the form of very small particulates.

This was a really interesting option however we could not search any further with its availability and its price.

Lastly, we covered the electrical components with aluminum foil which have the properties of reflecting the electromagnetic field this was the tested way due to the ease of availability and the cheap price making it a running competitor.

5.5 Conclusion

In conclusion we found that we have a lot of compatible methods for shielding like the metallic ink and the aluminium foil as both of them have proven that they are easy to implement however the aluminium foil has a higher compatibility as we have already tested this method.

5.6 LOW LEVEL CONTROL

Our low-level system had a few key requirements, it had to control all the motors in the robot to provide the motion required by the high-level control, and also provide means of communication between the robots and each other and the main controller (PC). As further development, we chose a few sensors to aid us in the docking and undocking of the modules from each other.

The system proposed will vary in conditions from the two extremes of structured or unstructured environment, starting with the working environment being seen via a camera and attempting to reach control in an unstructured environment using sensors alternative to the camera.

We will progress across the shown path, from the start of a fully manual system in a structured environment moving in the direction of fully autonomous in an unstructured environment.

In this section, we will go over the control algorithm that was followed, the communication system.

The high-level control will be sending commands to the low-level in order to control the robot, the low-level converts the high-level commands to executable actions controlling the motors and such.

5.6.1.1 Communication

The robots needed to be able to wirelessly communicate with the main controller (PC) while also communicating with each other when needed. The suggested communication systems ranged from Bluetooth to Wi-Fi.

Using a Bluetooth module would have proved easier in terms of coding, yet it could not provide us with the network capabilities needed. We needed all the robots to be simultaneously connected with each other and the main controller through a reliable network that also provided us with the ability to send messages to targeted addresses. Wi- Fi network was the best solution to our problem since it allowed all of the robots to be connected through an AP (Access point) and have access to each other through Static IP addresses.

5.6.1.2 System block diagram

The high-level code on the laptop publishes commands to the Wi-Fi network, the commands are received through the ESP-01 module and sent to the STM32 controller through UART. The STM32 sends the signals to the motor or reads from the sensors accordingly.

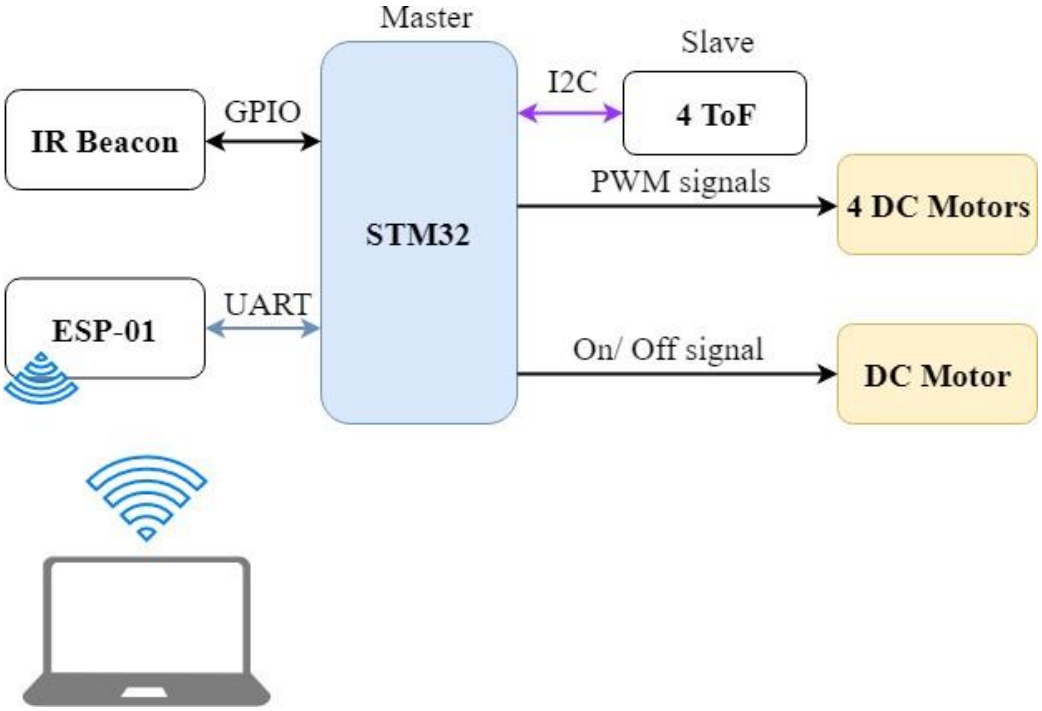


Figure 5.6.1 Control system block diagram

5.6.1.3 Finite state machine

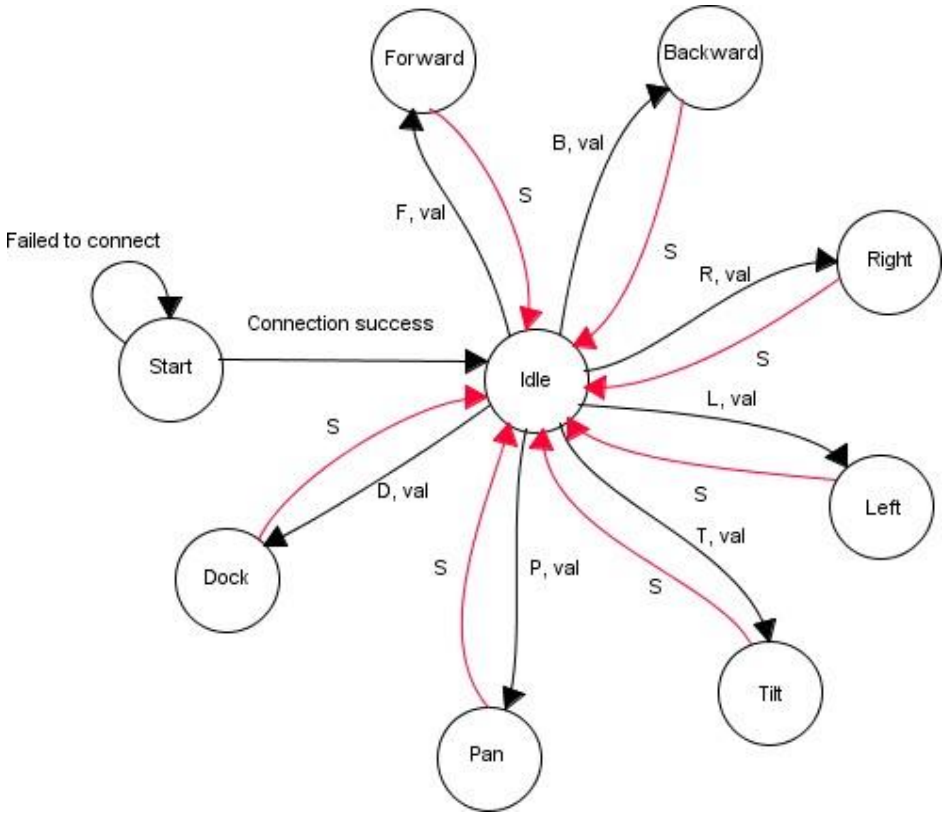


Figure 5.6.2 Finite State Machine

Since our robot will be continuously working, a finite state machine representation was needed to represent the main states of the robot.The following figure shows the finite state machine diagram of the robot, upon startup the robot is waiting for a connection to be valid, once it is connected to the network it is in the idle state and then waits for commands to perform one of the six main actions presented. Once an action is done, the robot immediately returns to the idle state waiting for the next command.

5.7 FLOW CHART

The flowchart is made to further specify the details in the code, specifically the connection method before reaching the idle state.

In the start of the code, the WIFI module searches for the connection. Once connected, the controller (ESP) waits for a command from the high level that then sends it to the main controller (STM).

The command specifies one of the ten possible actions along with a carried value or a symbol like in forward/backward forever, this value represents different things in the different pre-defined functions.

For the Forward and Backward function, the Val passed to the functions is the distance required in either direction. Those two functions loop until the desired distance is achieved.

For the left and right functions, the Val passed is the angle the robot will rotate around its center. The two functions loop until desired angle is achieved.

For the pan function, the Val passed is the angle required of the front face of the robot.For the tilt function, the Val passed is the tilt angle required.

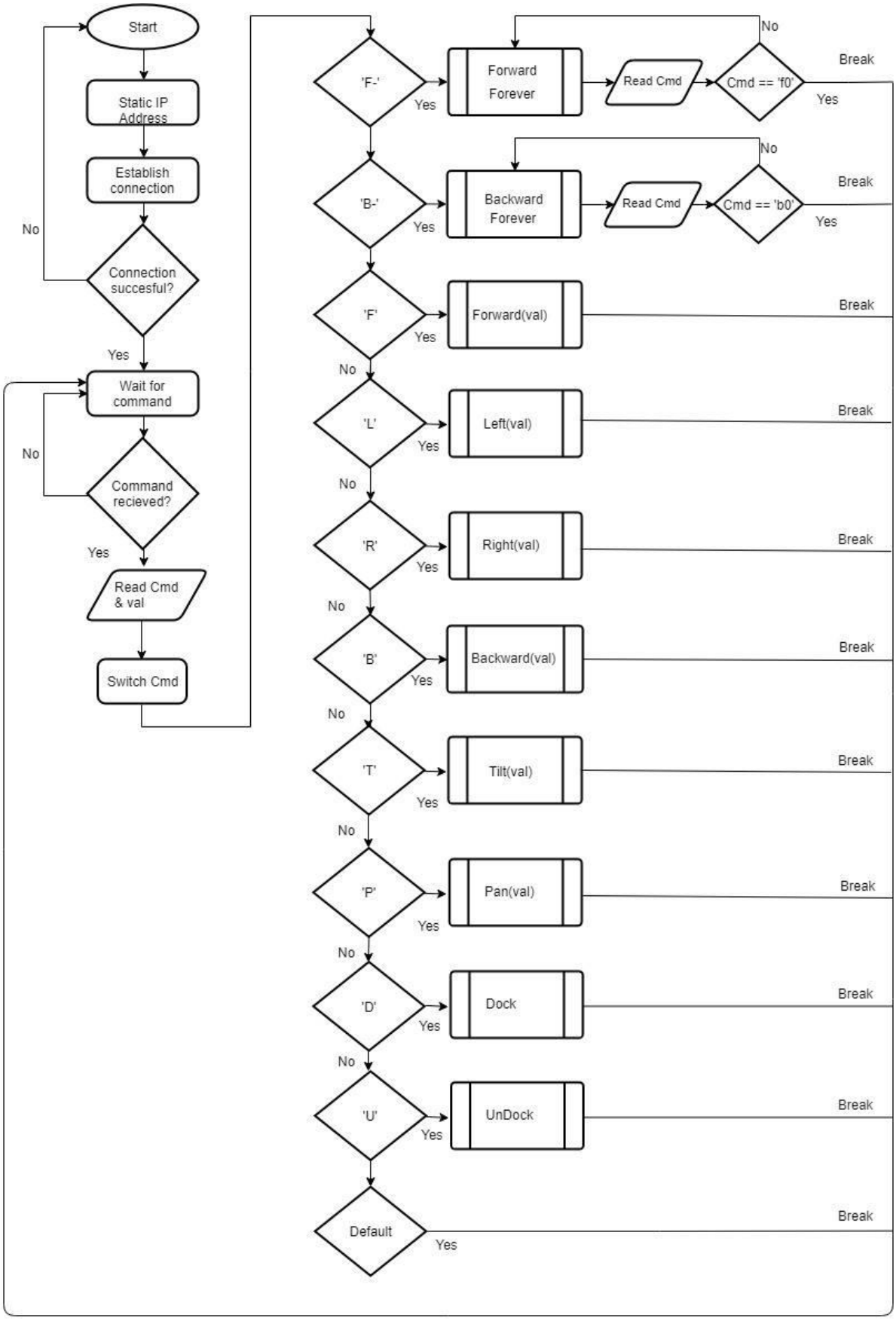


Figure 0.1 Control Flow Chart

5.7.1 Flow Chart of the Docking Logic with Tof

The tofs sensor will measure the angle and according to the sensors readings the robot will able to adjust itself in order to be aligned with the other robot in order to dock with the other robot and also the tof will be measure the distance between it and the 2nd robot that will make him able to go to the other robot and dock .

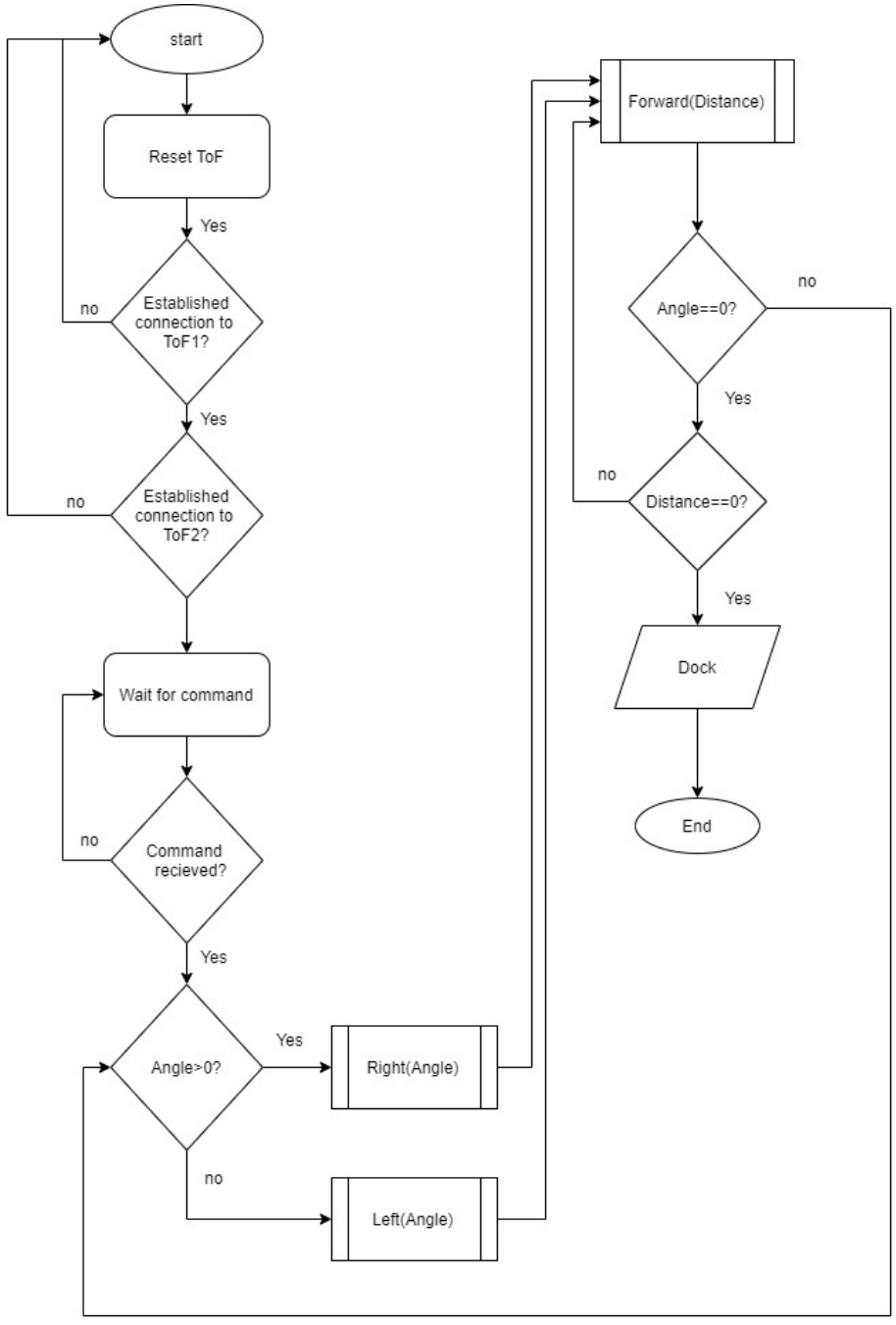


Figure 0.2 Flow chart of Tof logic

5.7.2 Functions

The forward and backward functions implemented PID speed control on both motors attached to the wheels, the loop was closed with the encoder as feedback. Since the function took distance as a parameter the function looped until the desired distance was achieved.

For the differential robot to move in a straight line, we needed to close the loop and since we were controlling one variable, PID control was used.

Several PIDs were implemented in our system based on the requirements. For instance, the wheels required PID to move in a straight line by maintaining the same speed to prevent swerving in either directions, for this we used a PI for speed control.

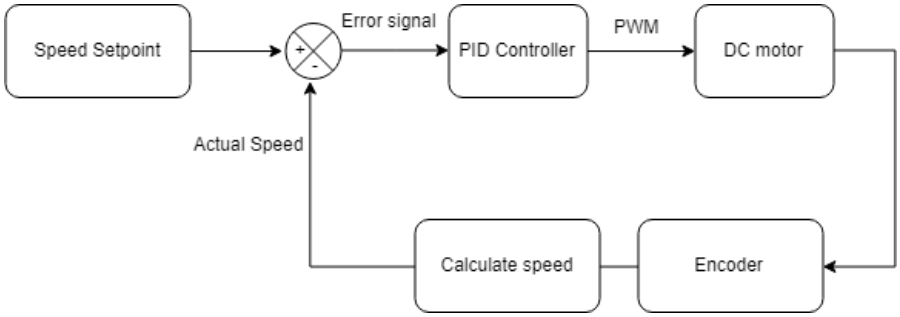
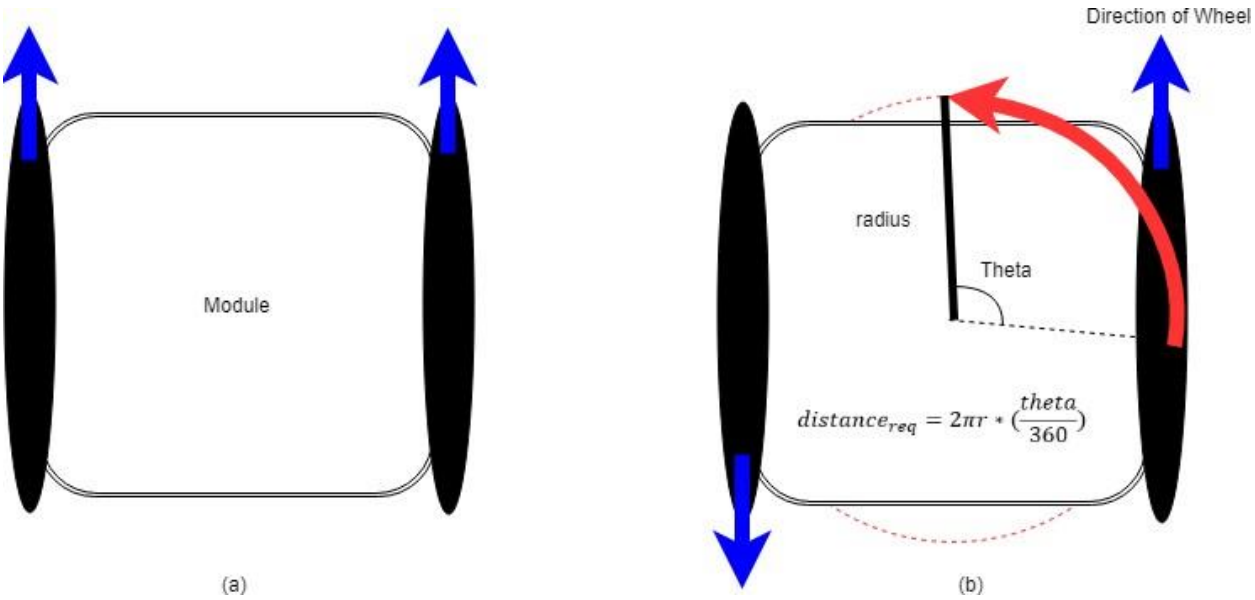


Figure 0.3 Feedback Control Loop

For the **forward forever and backward forever functions** they used the exact PID function as the forward and backward however they follow a different logic as these functions are required for the robot to move forward or backward till the operator sends a signal for the robot to stop.

So in our case there is a while function that keeps on checking if any signal is sent by the high level control for it to stop otherwise the robot will keep on moving.

These functions are usually used when no camera is used and the operator wants to operate the robot manually.



For the **functions of left and right**, it was used for the robot to turn a certain angle around itself received as theta. The function would give each wheel a different direction with the same setpoint of speed using a PID controller. The distance required is calculated as shown in figure xx(b) and the function loops until that distance is achieved meaning the required turning angle is also achieved.

For the **pan and tilt functions**, both speed and position control is applied. Speed control is applied as the both motors need to rotate with the exact same speed such that the gears in the differential mechanism won't break, where as for position control is used as reaching the exact position is critical for us especially in pan to facilitate the docking and undocking mechanism.

The PID function used is the cascade PID as we are using both speed and position control together. The control signal from the position control is feed in as the setpoint for the speed control function.

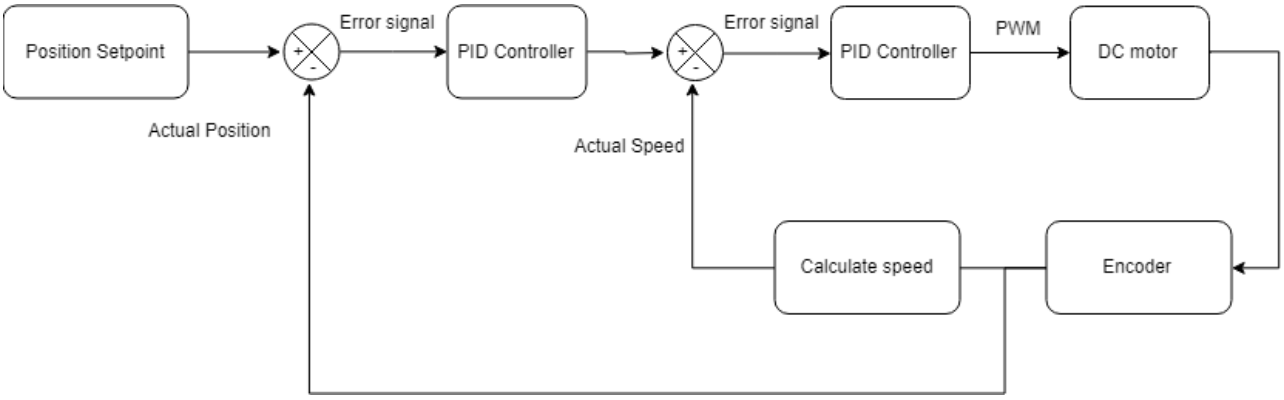


Figure 0.4 Feedback loop for pan & tilt

In the **tilt function** both motors move in the same direction to achieve the tilt motion. The Val given to the shows the angle at which the robot needs to tilt to. Also, it's bound by a limit from 5 to 85 degrees due to mechanical constraints.

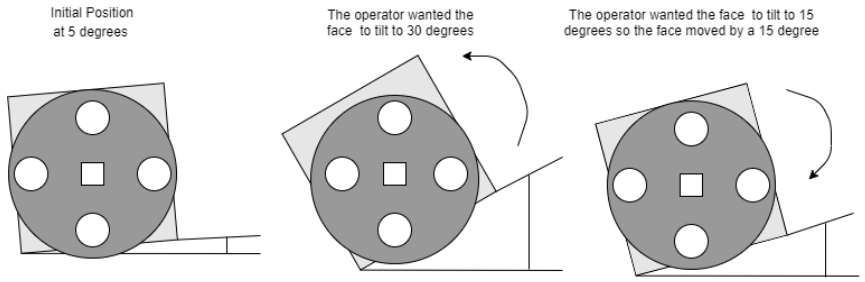
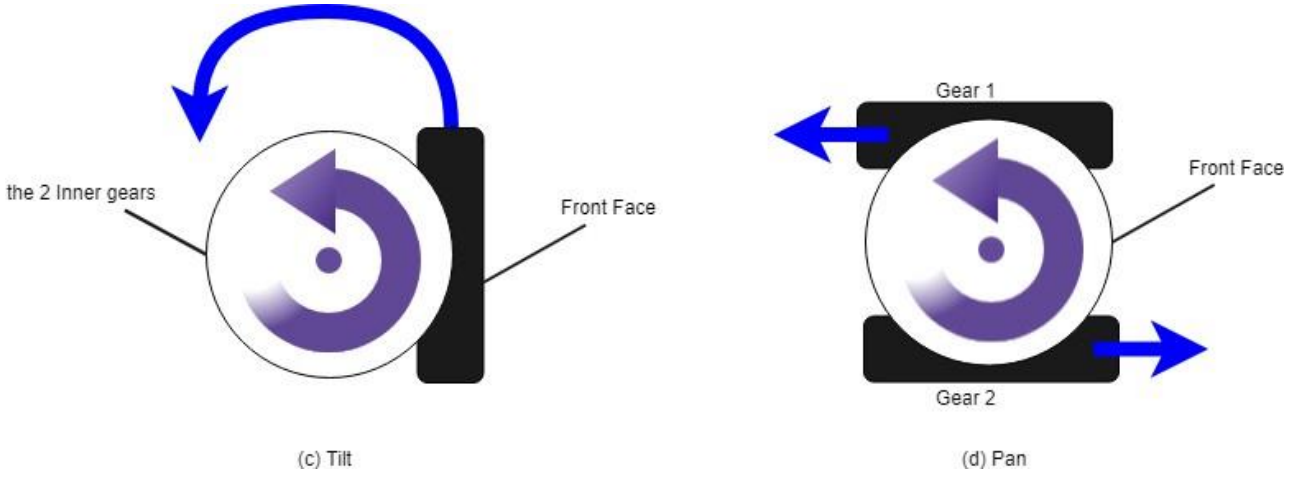


Figure 0.5 Fig Shows the Tilt Movements

This angle is an absolute angle meaning that when the robot is given a 45 degree it will tilt to 45 and if then given a 50-degree tilt it will move by 10 degree making the final tilting position at 50 degree. Unlike the forward and backward which is incremental we need the pan and tilt to be absolute to help in homing during docking/undocking which will be explained in more details in the docking functions.

We achieve this absolute behaviour by saving the previous value in a variable and when another tilt is sent to the robot this new Val will be subtracted from the previous one and the remaining degrees will cause the robot to tilt till it reach the desired position. If the new Val is less than previous Val it will tilt downwards and if it's higher it will move upwards.

For the **pan function** they follow the exact same logic as the tilt function however the motors are required to move in the opposite direction to achieve the pan motion.



The docking function it's a simple function that uses the previous Val in pan and tilt to return the robot to its initial position to perform the docking motion.

In here only one motor is used with a simple on and off mechanism so it doesn't require any PID control the motion adapted is a reciprocating motion were the motor will rotate for a specific distance and the two robots will start docking. This distance wasn't calculated we used the trail and error method using the motor counts to get that distance.

In here the pan and tilt are first homed using the previous Val and then the key will come out and docking takes place.

The undocking function is the exact same as the docking function due to the reciprocating mechanism however just after the key comes out the front face will pan by 90 degrees such that the magnets will start repelling each other and the undocking takes place.

5.8 Control Milestone

During the research period of our project our team have agreed upon the main milestones that need to be achieved and the steps on how to achieve it and the order of these milestones as well as the final target that we want to reach at the end. These milestones are explained bellow.

5.8.1 Manually operated

This is our first milestone that we wanted to achieve it's mainly to move the robot without any sensor or cameras meaning that no high-level control is taking part in this milestone.

In here we were aiming to test the low the low-level control mainly, to check whether the PID control is working and the functions of operation is working, the forward, backward, left, right movement as well as pan, tilt and docking.

5.8.2 Camera operated

This is the second step in here after checking that the low-level control and functions are working; we start to integrate it with the high-level control using the camera as the sensor.

In this step the high-level control will use the camera to check the position of the robots and decide which robot to dock with which in order to achieve certain reconfiguration it do so by the means of machine vision.

By the means of the camera the laptop will calculate the distance and the function that needs to be achieved by a certain robot to dock with the other robot then it sends that information to the robot through WIFI to the intended robot using it's unique IP address, after the robot receive

the function and the distance by the Wi-Fi module in it the robot will start to execute the movement.

5.8.3 Camera and sensors operated

This is the final milestone in here we wanted to proof the concept of the robots working autonomously without any cameras however due to our short timeline we will do so by using the sensors and the camera integrated together.

In here the high level will instead of sending distance and the movement function it will send just which face will need to dock with which face if the other robot.

5.8.4 Autonomous docking

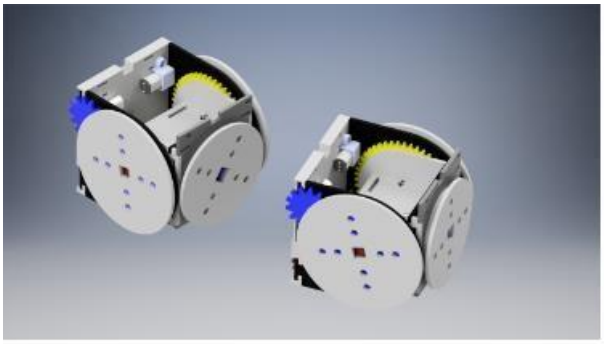
The autonomous docking between two modules is divided into five phases, namely finding, locating, communicating, navigating and lastly docking.

At the beginning, the module that receives the docking signal will be considered the current Master module and will initiate the phases listed above. This will also initiate the IR blink function in all the modules in the network, so that they all blink with their specified frequencies to prepare for docking.

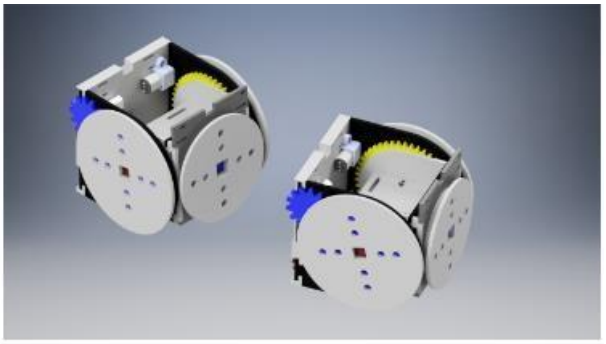
The Master module will begin to read the IR sensor to detect the blinking module closest to it. Based on the read frequency the module ahead is identified and based on the duty cycle the face of the Slave module is also identified. Then the Master module attempts to locate the slave module using the ToF sensors to calculate the distance and the difference in angle between them.

The Master module communicates with the slave module using the registered IP address in order to fetch its face angle, then the master changes its face angle accordingly to match the slave’s.

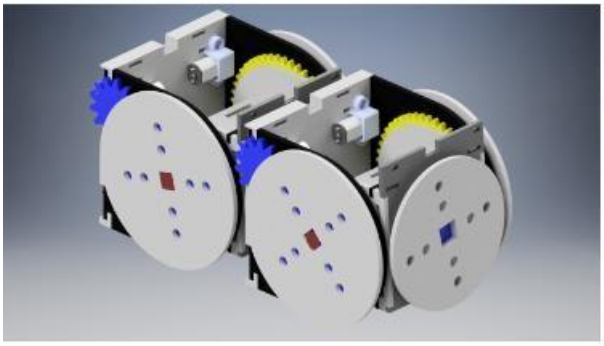
Based on the difference in distance and angle, the master moves to meet the face of the slave module until docking is achieved.



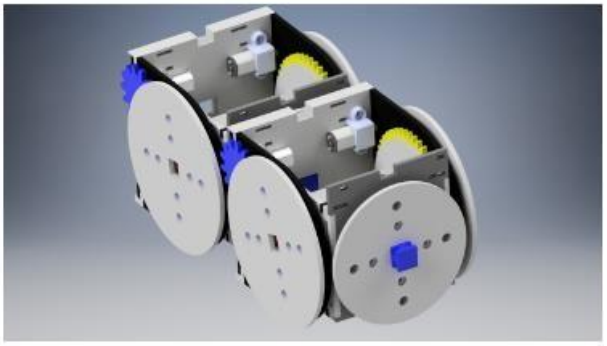
(a) finding & locating



(b) communicating



(c) navigating



(d) docking

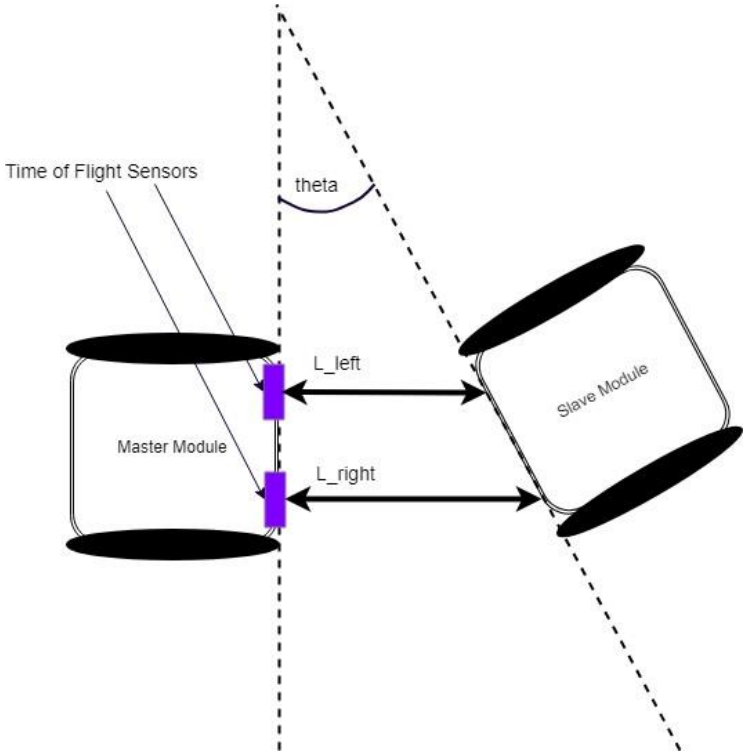


Figure 5.8.1 Tof angle measuring

5.8.5 Undocking

For the undocking of the modules, we have two options to release the two docked robots. The first option would be to rotate both faces in opposite directions simultaneously for 90 degrees causing the magnets that face each other to be of the same pole and thus repel each other and release.

The other option would be to depend on the latching/unlatching key, but firstly the faces would have to rotate in opposite directions for 45 degrees for the key to then extend and push the other module away.

5.9 CHALLENGES

5.9.1 ESP

This model of ESP were not widely used so we were limited with the number of references available to it especially with the STM not only that but also we were not able to program it using the STM so we had disconnect it and connect it to the Arduino the connect it back to the STM making the testing process really dreadful and time consuming, not only that we also had difficulty in making it’s IP static that was crucial for our project.

5.9.2 STM32

5.9.2.1 STM32 libraries

Until now we are unable to connect the ESP WIFI module successfully to the STM32, since the libraries found are so far not compatible and would not compile for our STM version.

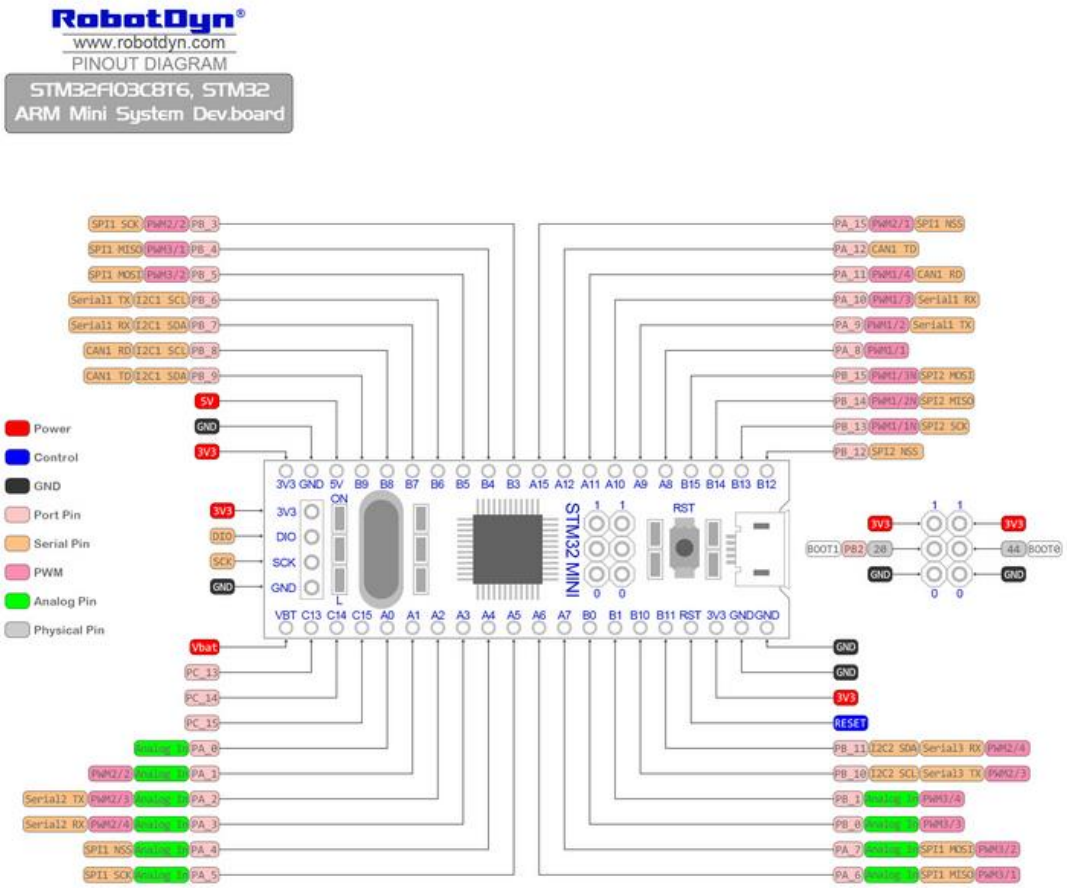
We faced the same problem with the ToF sensors, the only library that worked was hard setting the i2c address of the sensor when we need the ability to change the address freely to initialize our 4 sensors.

5.9.2.2 Pinout

The official page that we brought our STM from have shown that the STM have around 15 PWM pins however in the pinout diagram it shows 20 pins and sadly we discovered this mistake late in the project after manufacturing the PCB that caused us to change the tracks manually and edit it to make it work.

Pinout and interface

With **32 GPIO – 15 PWM pins, 10 analog inputs**, 3 UARTs (hardware serial ports), 2 I2C and 2 SPI interface, and a larger memory space 64/128 /256KB for your code.



5.9.2.3 Examples

The STM32 is not that widely used microcontroller which unfortunately meant that there were fewer references to other people's work which made it hard on us when it came to debug an issue or understand how the board pins work.

5.9.3 Sensor Accuracy and delays

The sensor was a very vague area the Tof sensor was not available Egypt so it was not tested before and after numerous testing on several MCU's the results was not very encouraging there was fluctuating errors in the sensors , moreover integrating the sensor with the STM32 as mentioned earlier the library and sources of stm32 was very limited.

5.9.4 PCB

As we initially manufactured a prototype PCB we realized that the manufacturer never had via for there double layer PCB so they had to make changes in the original design that made it harder to track when issues would appear and at the end it lead us to loose a motor and an STM.

After the new PCB were manufactured things went smoother however we were having the issue of the solder melting and grounding the signal pins to the ground that caused us a lot of issues because it was hard to debug what happened.

5.9.5 Code

The initial code was written without being tested that was a real drawback as we had to go line by line to start and debug it which costed us a lot of time and effort.

5.9.6 Mechanical

Due to the laser cut clearances the current robot is smaller than the cad which lead to the new PCB not being able to fit, not only that but also we were challenged as some of the mechanical caused issues during testing like in the forward function one wheel would be looser than the other making it hard for the robot to move in a straight line or like during the tilt movement whenever the robot tilt downward it will tilt more due to the front face weight that is helping the motor go mor down than the specific value required.

CHAPTER SIX: PROTOTYPE MANUFACTURING

The manufacturing process we mostly depended on is Laser cutting and 3d printing process Almost all of the parts is manufactured from 3 mm Acrylic sheet.

The shelves were fixed using T-slots and bolts, nuts.

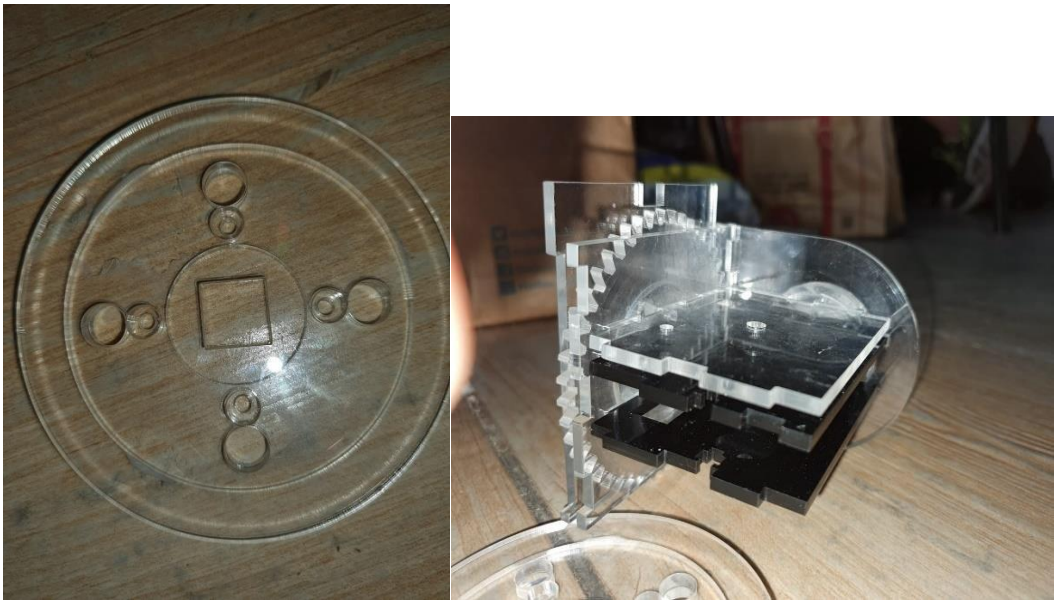


Figure 5.9.1 Manufactured Front wheel

Figure 5.9.2 Motor bracket SLA and 3D printed key

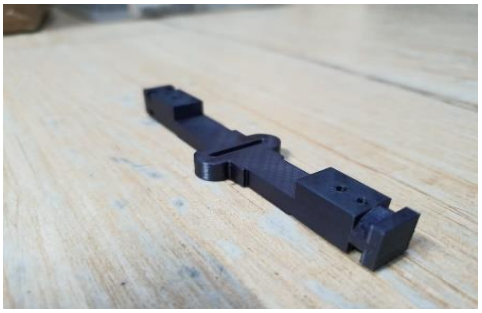


Figure 5.9.3 2nd part of the key

6.1 FINAL ASSEMBLY MANUFACTURED

The following pictures represents the Final Assembly of the 1st prototype manufactured and assembled

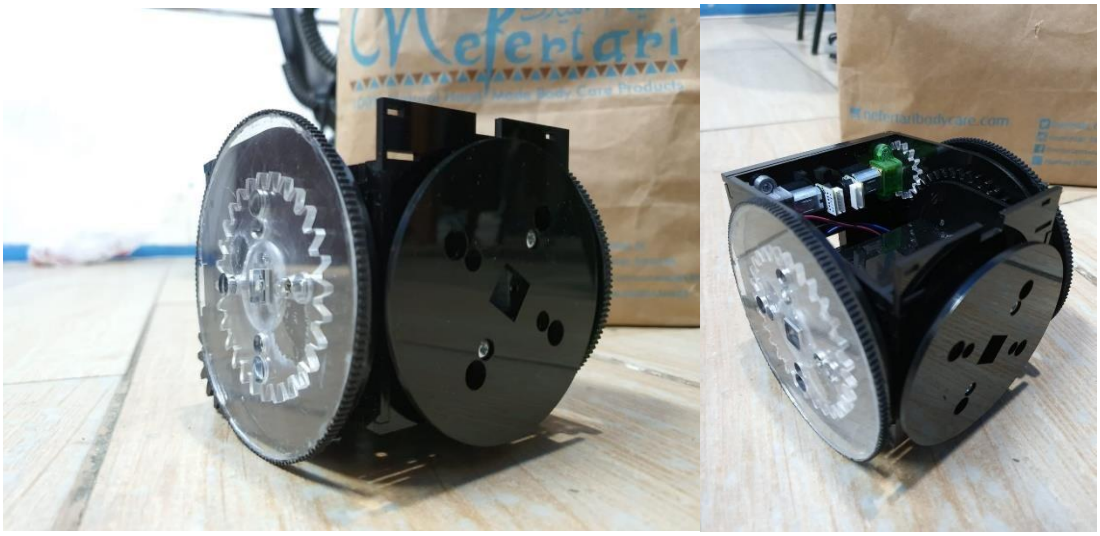


Figure 6.1.1 Final Manufactured Assembly

During the manufacturing process however we faced a couple of the problems which included

The Acrylic sheet was not 3 mm, there was error that caused clearance in the body.

The laser beam caused issues in the clearance and interference of the parts

The Hubs has clearance during the 3D printing.

6.2 ELECTRICAL TESTING AND IMPLEMENTATION

6.2.1 PCB

We manufactured the PCB prototype taking the following into consideration:

- The PCB size was a major factor as it had small space to fit on the shelf
- Minimum components as much as possible

Track width consideration, we calculated the track width based on the current of each component.

Figure 6.2.1 Manufactered PCB

The components on the PCB added:

- The blue Rosetta was used for Power
- One is used for the motors and the other is for the other components
- The Female pin headers it is for the STM 32
- The 2 seats is For the motor drivers
- The male pin header is for the rest of the components which is the sensors , motors
- Mosfet IR P 12
- Resistors

The second PCB manufactured were bigger as we had to change the motor driver due to the motors changed leading us to add 16 diodes to the previous PCB shown above.

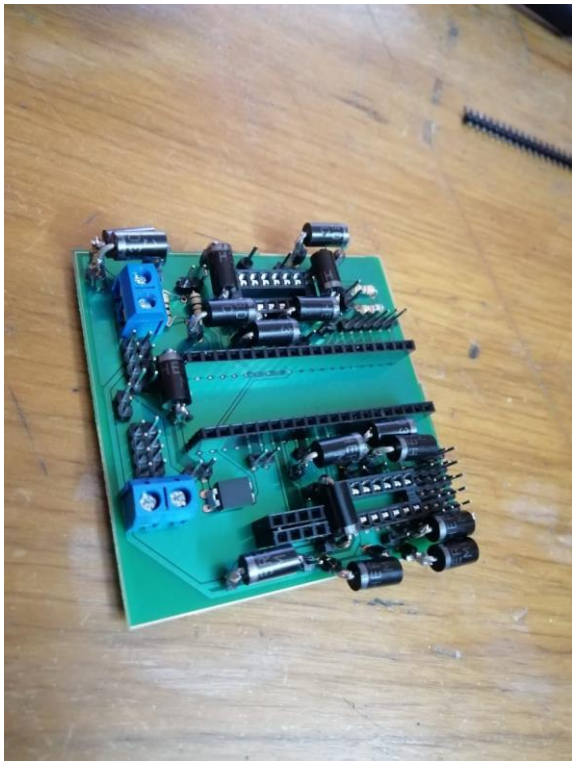


Figure 6.2.2 Another PCB View

This was done to prevent the motor drive from getting and back lash from the motors as the previous motor driver had built in diodes the new one did not.

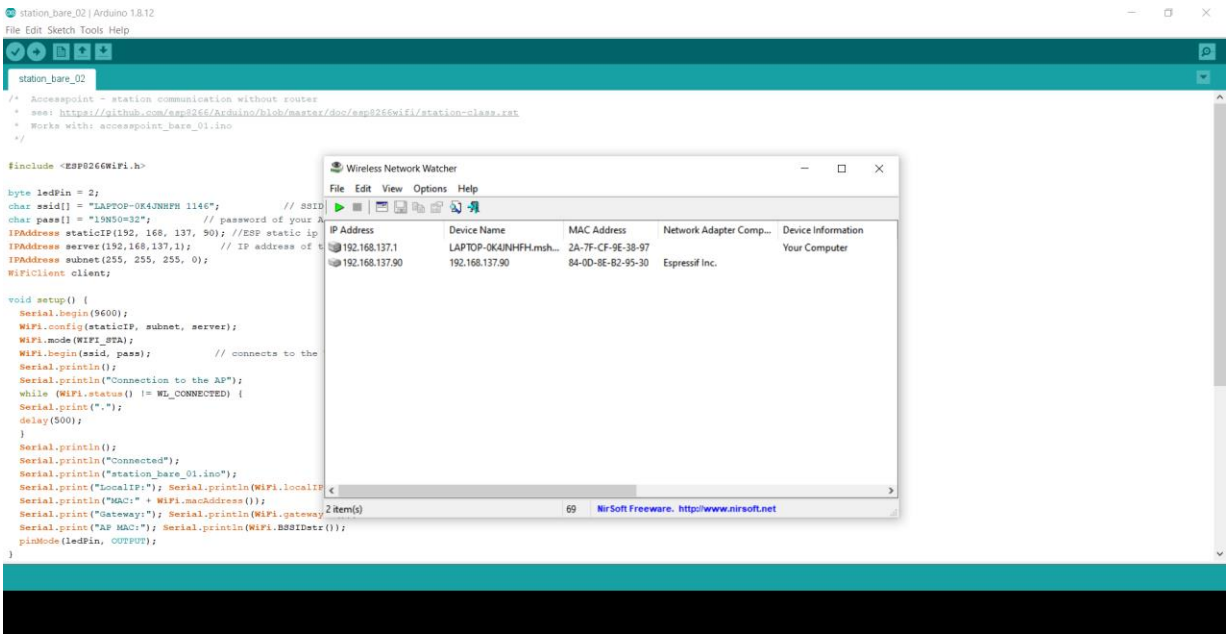
After making the PCB we started testing it checking if all the components are connected and working by using simple codes to eliminate any electrical error while debugging.

6.2.2 ESP

We started testing the ESP first by a simple communication between it and the Arduino by just sending a simple hello from the ESP.

Then we followed by sending simple text from the computer then to the ESP and then sending these texts to the Arduino. Also sending back text to the computer using the ESP.

After establishing two way communication we started to tackle the fixed IP address issue as it was required for each robot to have a specific IP address to ease the communication between the robots and the high level control.



After making sure everything were working on the Arduino we moved on to the STM and started with exact steps as the Arduino however unlike the Arduino we were not able to program the ESP using the STM so when any changes were needed in the ESP we had to connect it back to the Arduino then after making sure everything is going smooth we would return it back to the Arduino.

6.2.3 MOTORS

The motors were first tested using an Arduino we started first by ensuring the ppr given by the company is the right one as its crucial in our calculations, we have done so by setting a specific number of counts and seeing how many revolution have the motor performed.

After the ppr is checked we move to the next step of checking a simple forward and backward function using the motors without being attached to the robot to check if the motors are working properly and motor drivers are not broken.

After doing so we start testing using the STM to ensure its well-integrated before connecting them to the robot

6.2.4 ROBOT

After making sure every component is working properly, we started attaching the components to the robot to start testing.

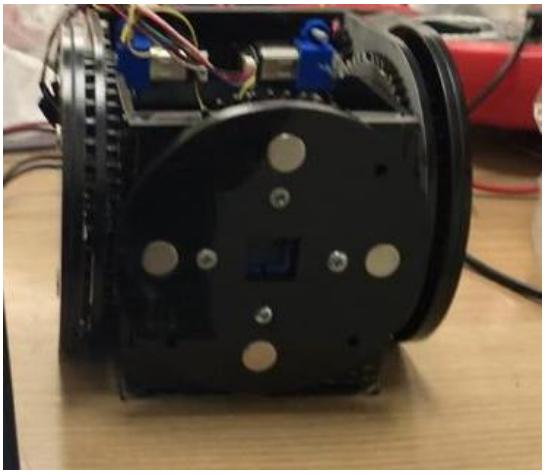


Figure 6.2.3 Robot Testing

We first started with the forward and backward functions first as they were the simplest functions to check. We checked first by just giving the robots high and low the we proceeded by upping the PID speed control to make sure the robot is moving in a straight line by adjusting the PID parameters.

Then we moved on to the left and right function they did not take much time to test as they were using the same PID function as the forward and backward.

After ensuring that these movements are fully functional we moved to the pan and tilt and this is where we had most of our time in testing as we had to start directly with the PID control such that the differential mechanism won't be damaged nor the motors attached to it also we faced some complication in the accuracy of the functions.

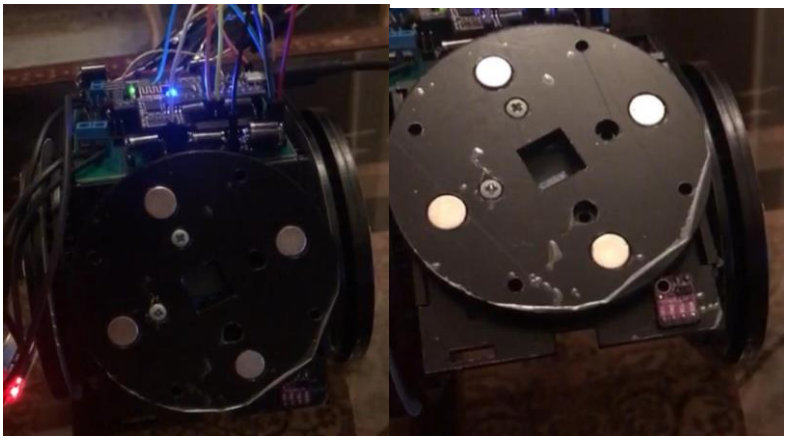


Figure 6.2.4 Robot Tilting

Figure 6.2.5 Robot Panning

After the pan and tilt we moved on to the docking and undocking mechanism we tested the mechanism first by a simple high and low to make sure that the mechanism is working smoothly before including the logic to it.

6.2.5 Tof Testing and TCRT

The Tof testing we tested 1 sensor first and then we placed two sensors on the same I2C bus in order to calculate the angle as well as measure the distance.

The TCRT sensors blinks with certain frequencies in order that we will be able to identify the front face from the back face of the robots

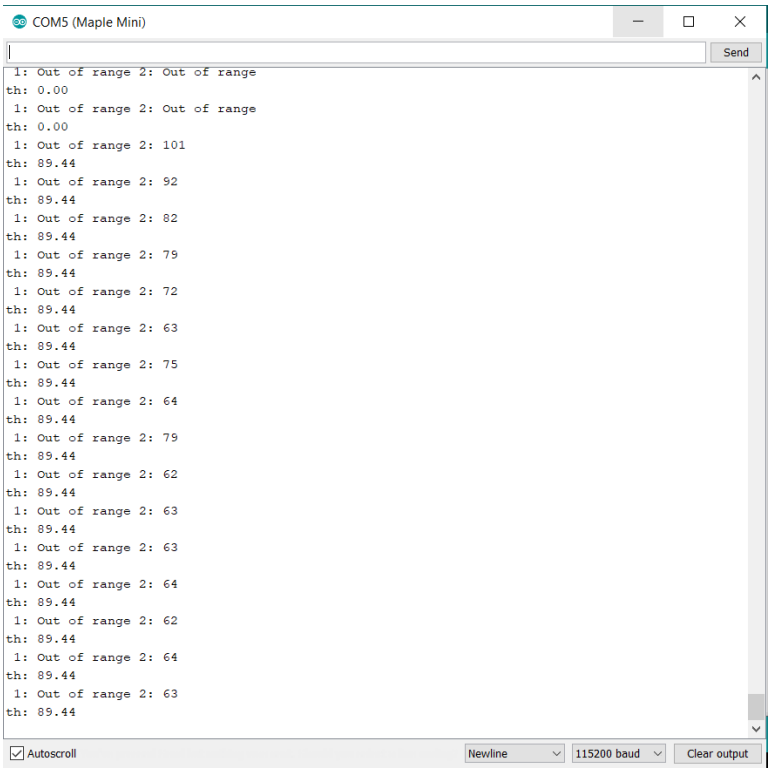


Figure 6.2.6 Tof sensor readings

The next step was to integrate the sensor with the full code on the stm 32 and place the sensor on the robot .

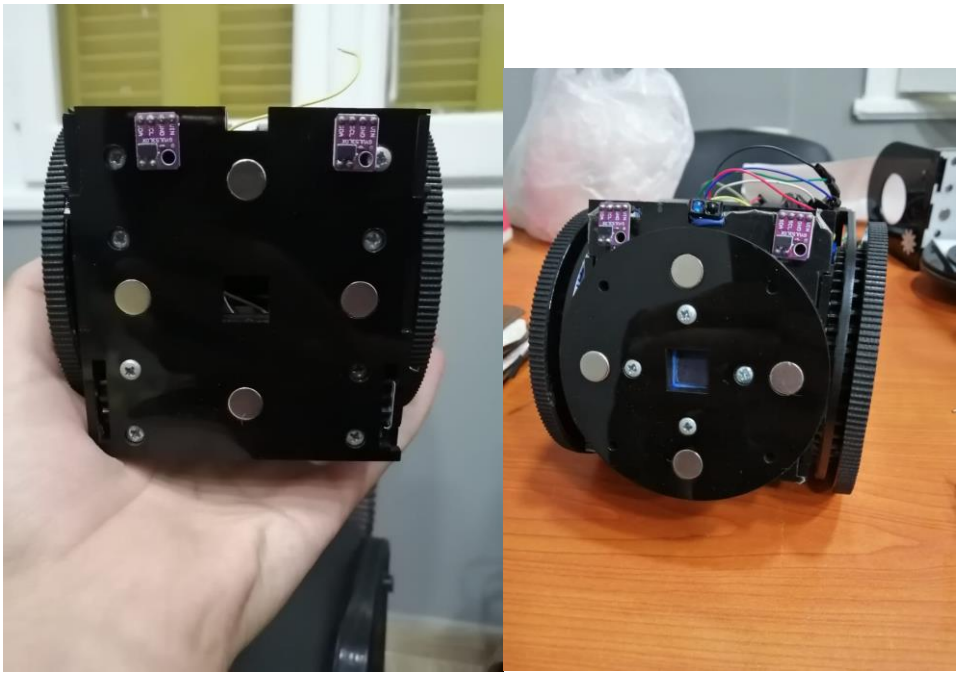


Figure 6.2.7 Sensor placed on the robots

There are 4 tofs sensors placed on each robot that measures the angle and distance both infront and the back of the robot. the 2 TCRT sensors are placed one in front and back ward as shown and of course it blinks with a certain frequency, so we are able to identify the face and the back of the robot.

CHAPTER SEVEN: FUTURE WORK:

After adding the sensors and the camera, we aim to semi-automate the modules so they can find each other by the help of the camera and the different sensors and use them to dock to each other and build different structures and test different algorithms in automation and swarm control. Having digital model of our module opened for us infinite opportunities as we can have any member in the team test, and try sensors out, and algorithms without the need for the physical ones. Also by specifying the motors max torque, and giving the modules realistic masses we can test and visualize the max limits of our system of robots. We are preparing for releasing the code, the model CADs, components, electronics and instructions for the assembly of our module making it easier for anyone interested in taking part in this project as an Academic project or even for hobbyists, Also having digital model can give an opportunity for even machine learning and training a model for a specific task.

This is the final milestone in here we wanted to proof the concept of the robots working autonomously without any cameras however due to our short timeline we will do so by using the sensors and the camera integrated together.

In here the high level will instead of sending distance and the movement function it will send just which face will need to dock with which face if the other robot.

REFERENCES

[1] “The Industrial Robotic Arm -- An Engineering Marvel,” [Online]. Available: https://www.robots.com/articles/the-industrial-robotic-arm-an-engineering-marvel.

[2] W. a. H. E. Golnazarian, “Intelligent Industrial Robots,” in *IMTS 2002 Manufacturing Conference, Sep. 9, 2002* , 2002.

[3] A. Robotics, “IRB 120 data - Industrial Robots (Robotics) - IRB 120 - Industrial Robots (Robotics) (Industrial Robots From ABB Robotics),” [Online]. Available: https://new.abb.com/products/robotics/industrial-robots/irb-120/irb-120-data. [Accessed 21 January 2020].

[4] “Camera blog,” [Online]. Available: https://www.e-consystems.com/blog/camera/what-is-a-stereo-vision-camera/. [Accessed 2020 January 26].

[5] “Setting Up a 3D sensor - Industrial Training documentation,” [Online]. Available: https://industrial-training-master.readthedocs.io/en/melodic/_source/demo3/Setting-up-a-3D-sensor.html. [Accessed 26 January 2020].

[6] “What Is Camera Calibration? - MATLAB & Simulink,” [Online]. Available: https://www.mathworks.com/help/vision/ug/camera-calibration.html.

[7] R. S. Sutton and A. G. Barto, Reinforcement Learning: An Introduction, MIT Press, 2015.

[8] A. Ademovic. [Online]. Available: https://www.toptal.com/robotics/introduction-to-robot-operating-system#blog_post-article-author.

[9] D. T. L. H. Carol Fairchild.

[10] [Online]. Available: http://wiki.ros.org/Robots/Care-O-bot/Tutorials/Care-O-bot%20environments%20%28Gazebo%20and%20Rviz%29.

[11] [Online]. Available: http://gazebosim.org/tutorials/?tut=ros_urdf.

[12] t. foote. [Online]. Available:
http://wiki.ros.org/Papers/TePRA2013_Foote?action=AttachFile&do=get&target=TePRA2013_Foote.pdf.

[13] R. Zhao., “Trajectory planning and control for robot manipulations.”.

[14] R. Katzschmann, “Dynamic Online Trajectory Generation”.

[15] “Reflexxes,” [Online]. Available: http://www.reflexxes.ws/.

[16] “Application manual controller software IRC5”.

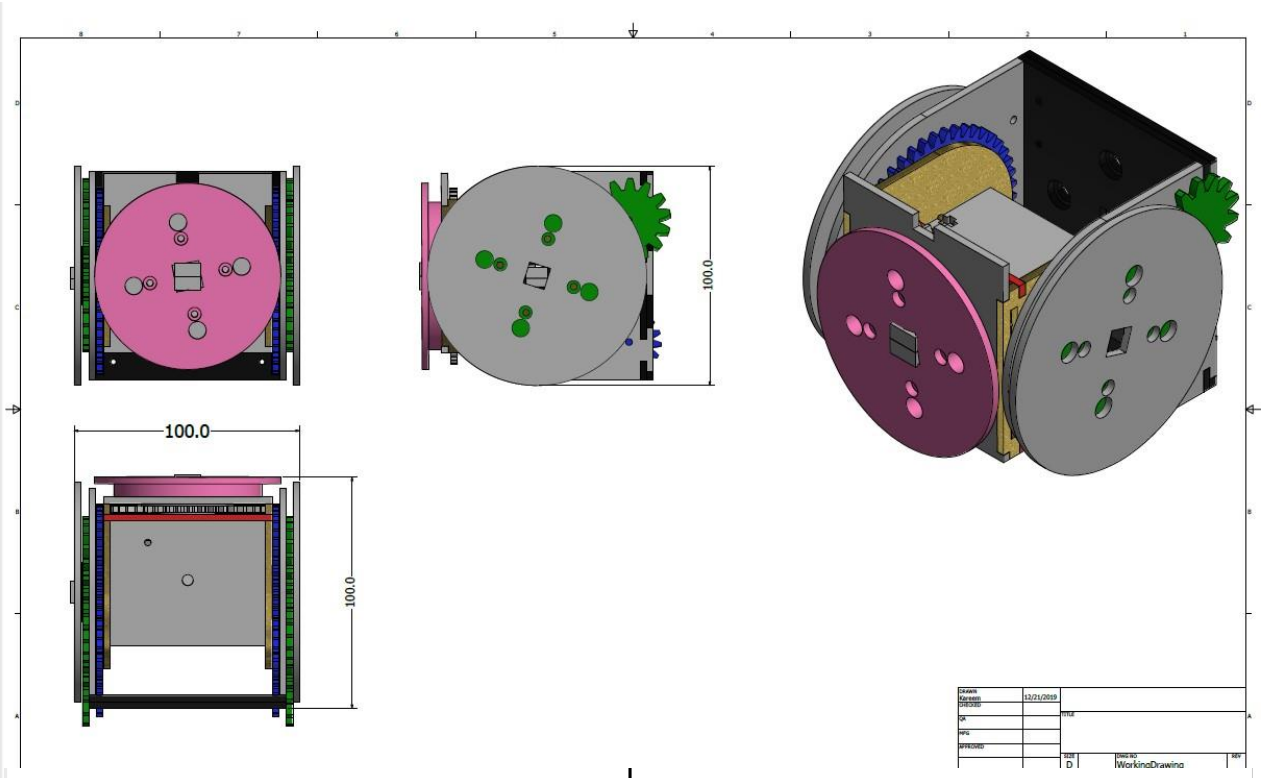
[17] “Ease of use packages between EGM and ROS,” [Online]. Available:
https://rosindustrial.squarespace.com/s/20181212_Jon_Tjerngren.PDF.

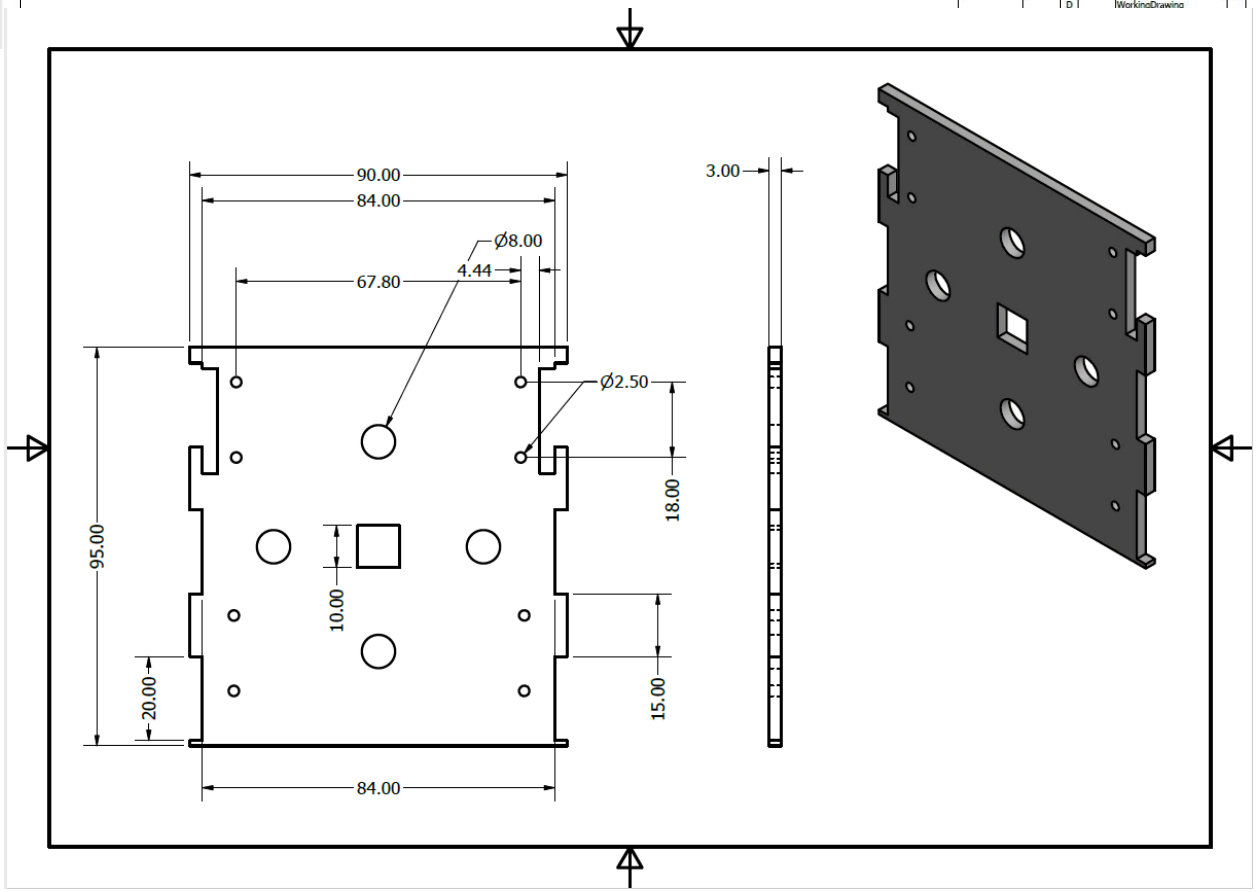
[18] “Github,” [Online]. Available: https://github.com/ros-industrial/abb_libegm.

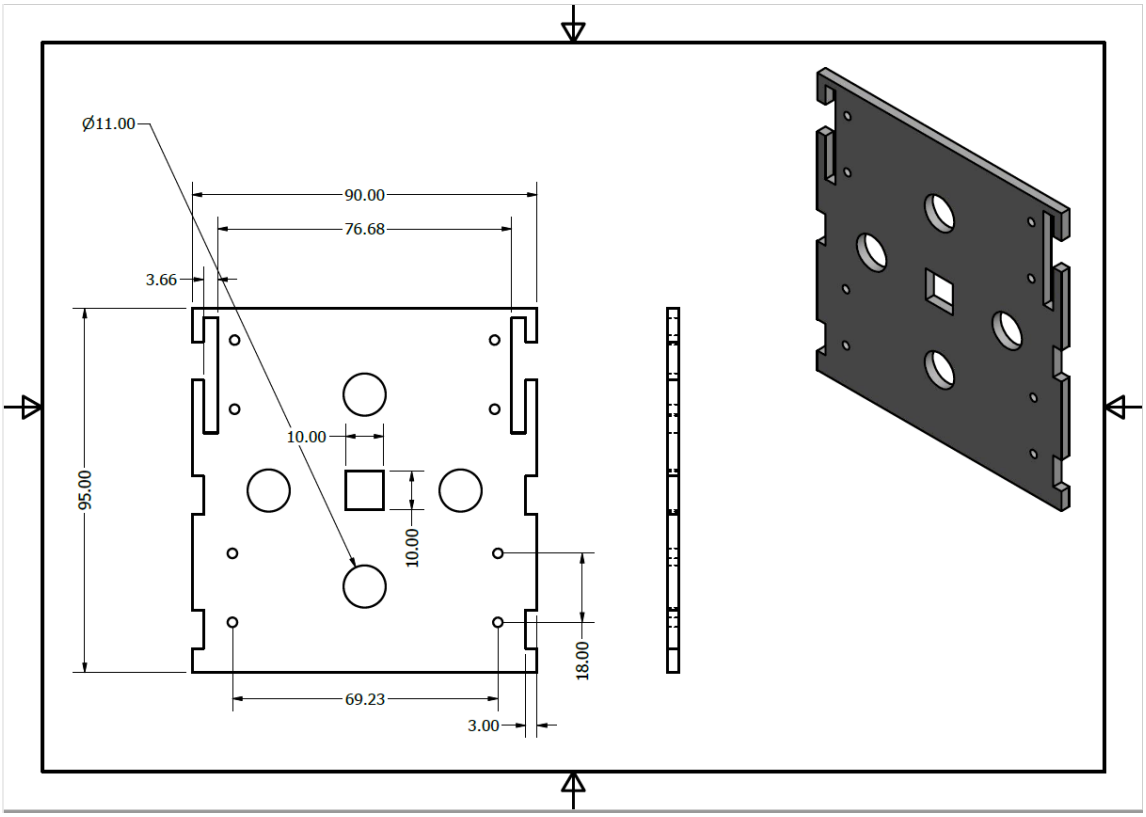
[19] “K,” [Online]. Available: https://www.mas.bg.ac.rs/_media/istrazivanje/fme/vol43/1/8_bkaran.pdf.

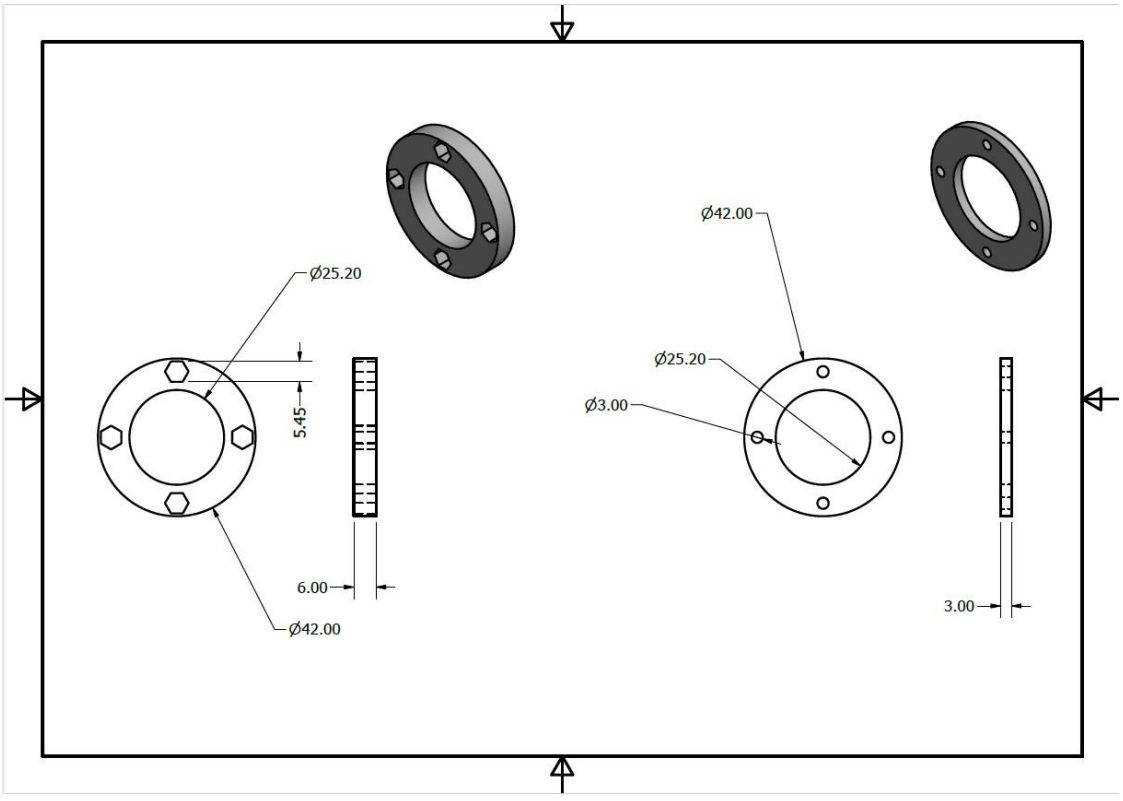
APPENDIX

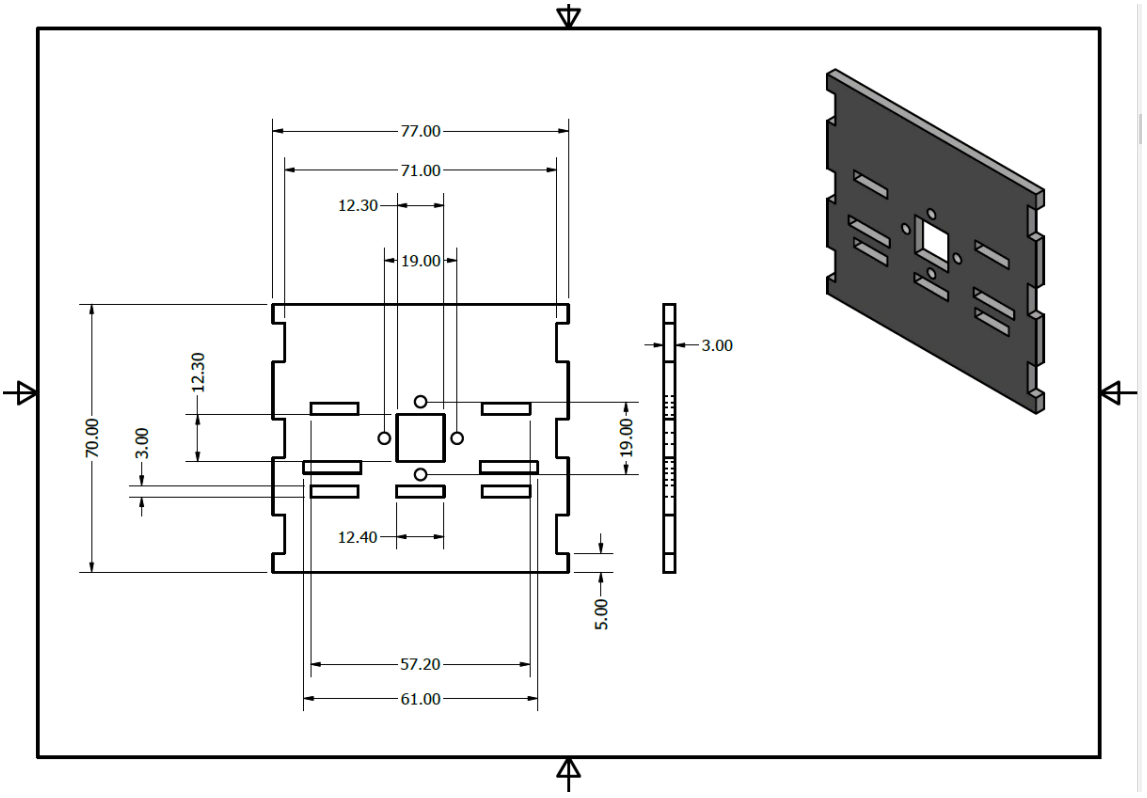
8.1 WORKING DRAWINGS

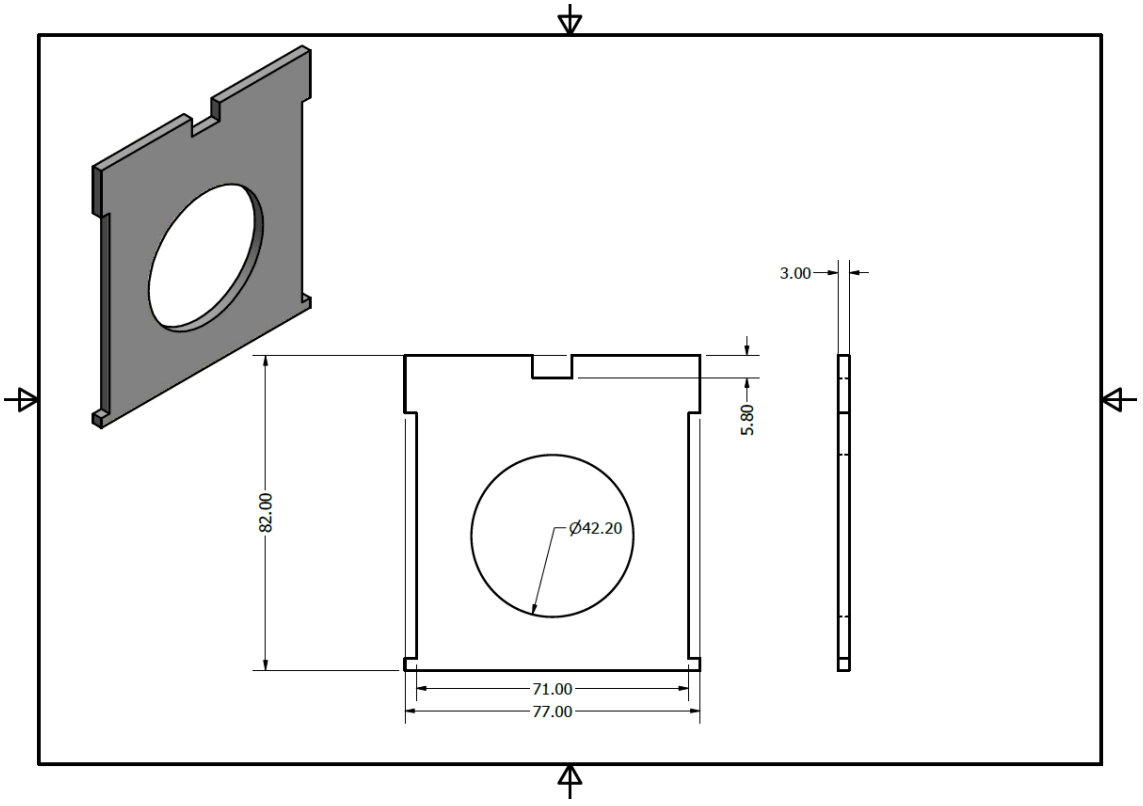


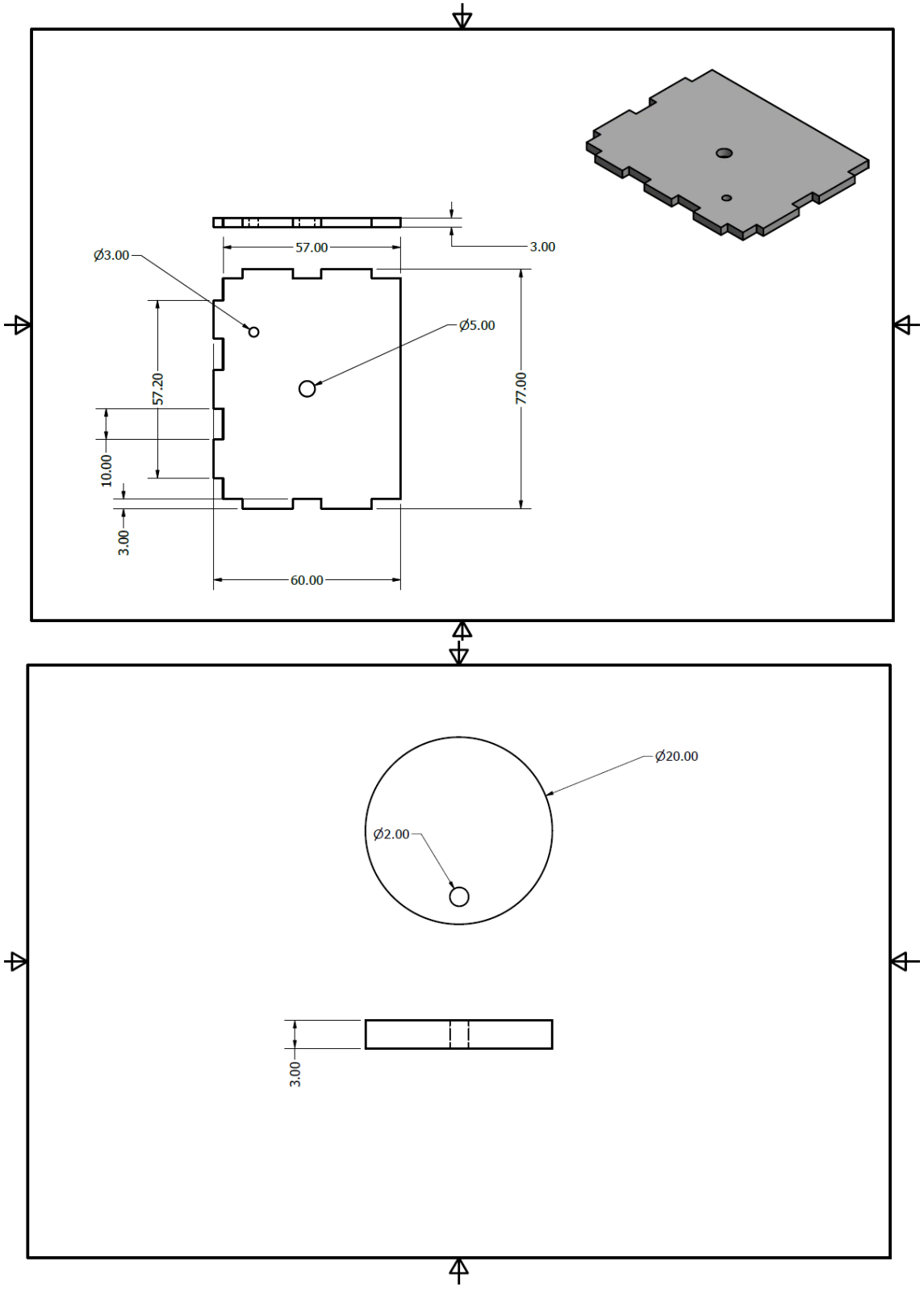


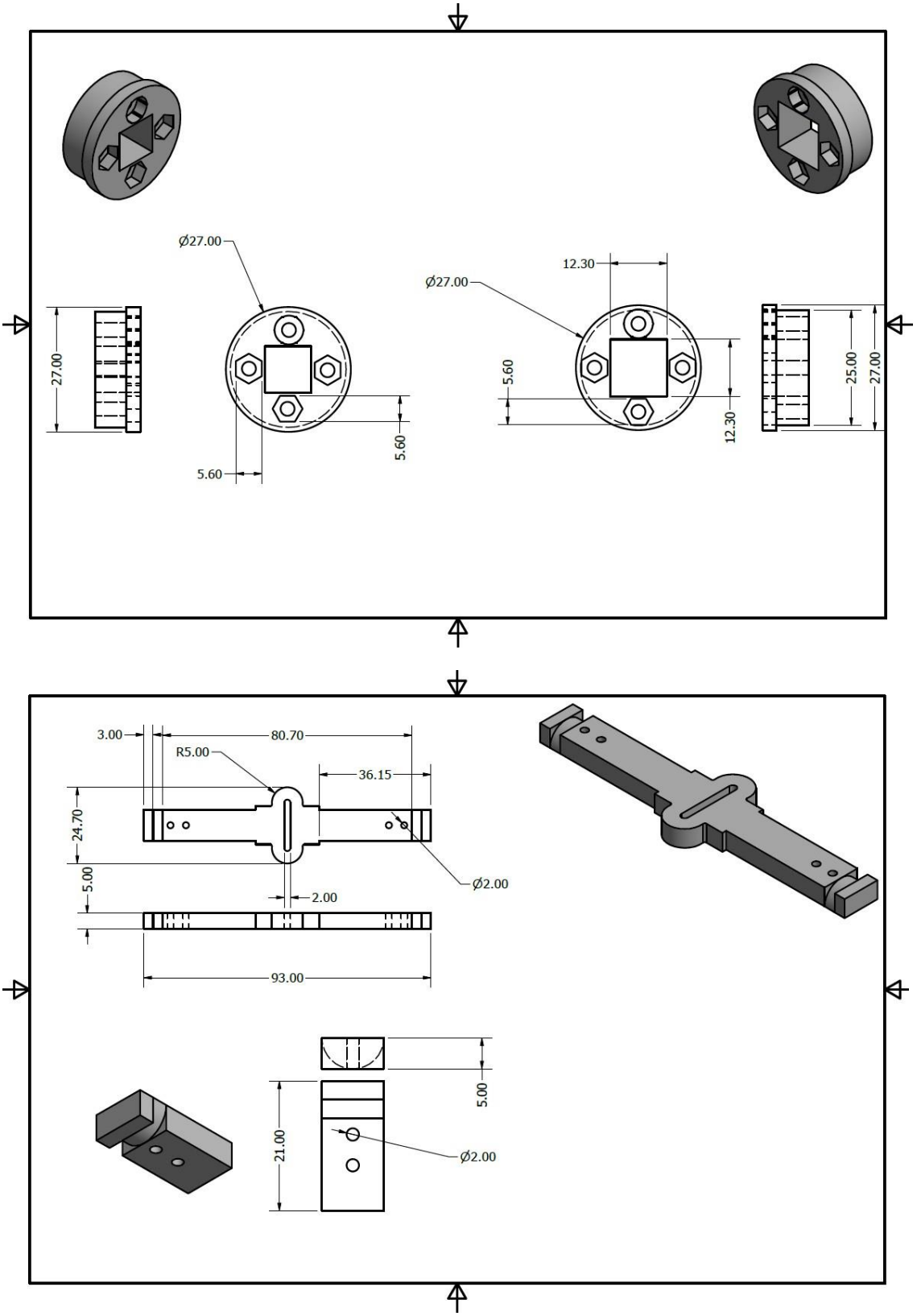

Ø11.00
90.00
76.68
3.66
95.00
10.00
10.00
18.00
69.23
3.00

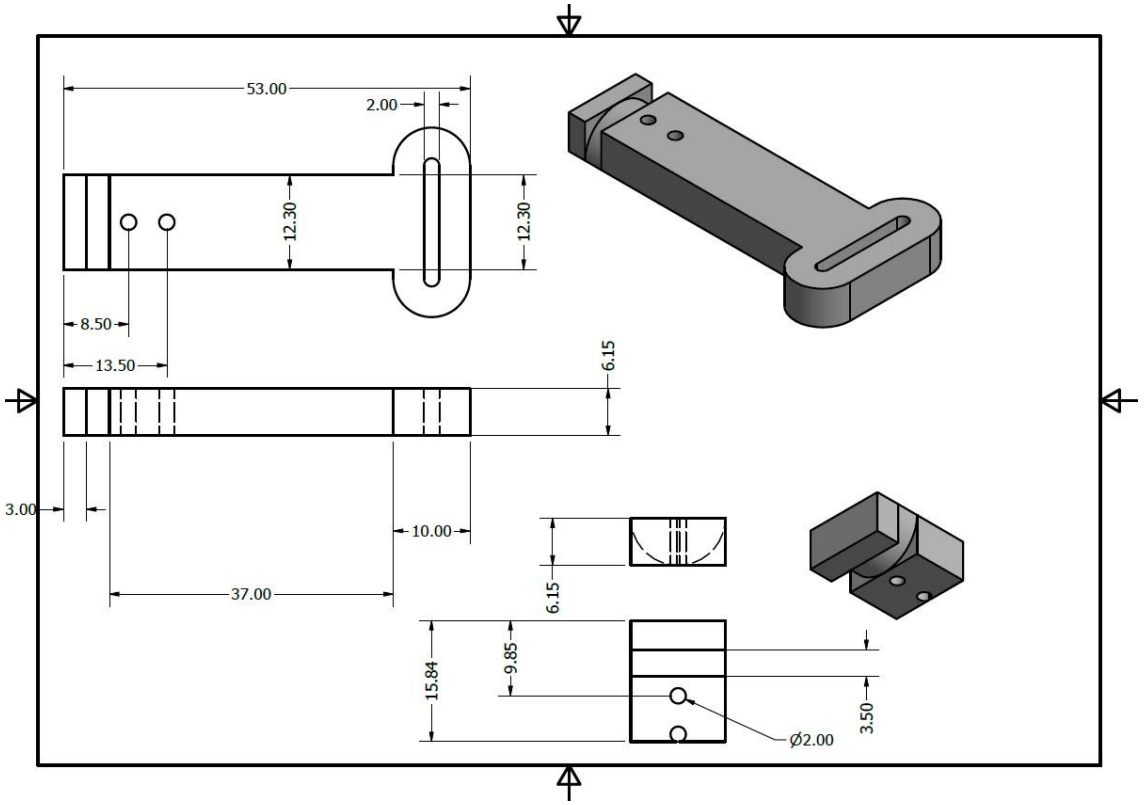

Ø25.20
5.45
6.00
Ø42.00
Ø42.00
Ø25.20
Ø3.00
3.00

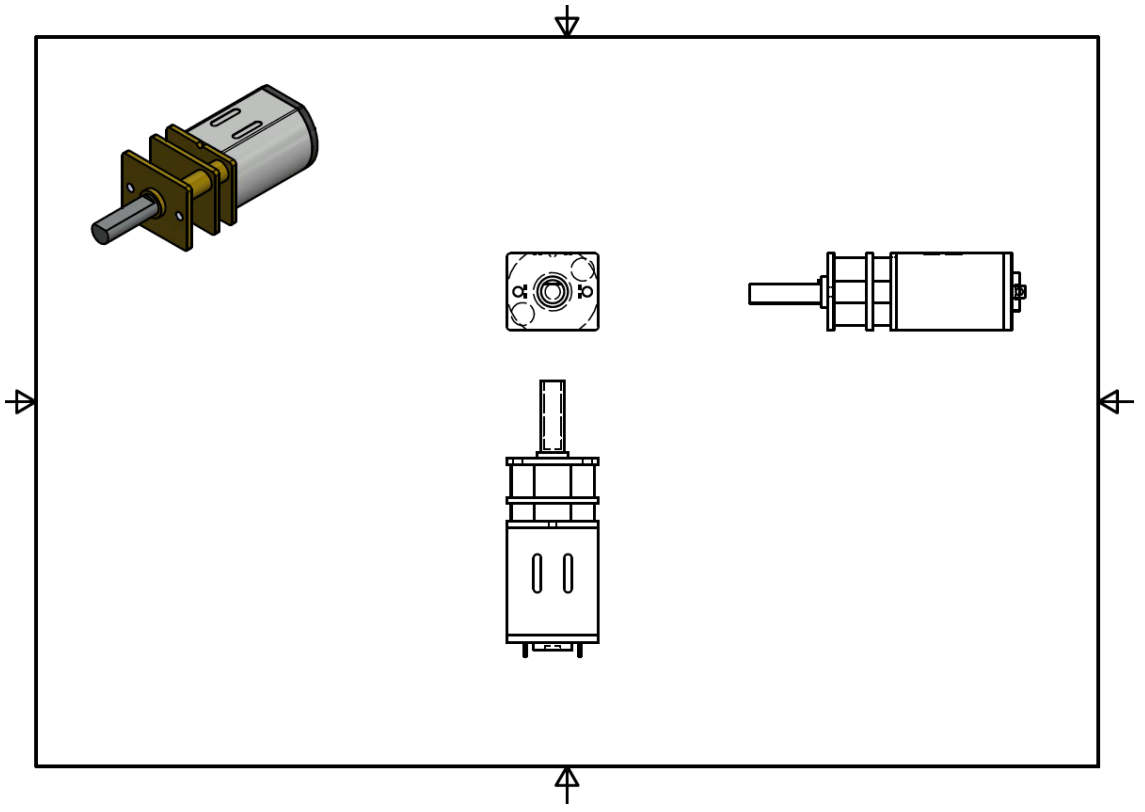

77.00
71.00
12.30
19.00
70.00
3.00
12.30
19.00
12.40
5.00
3.00
57.20
61.00

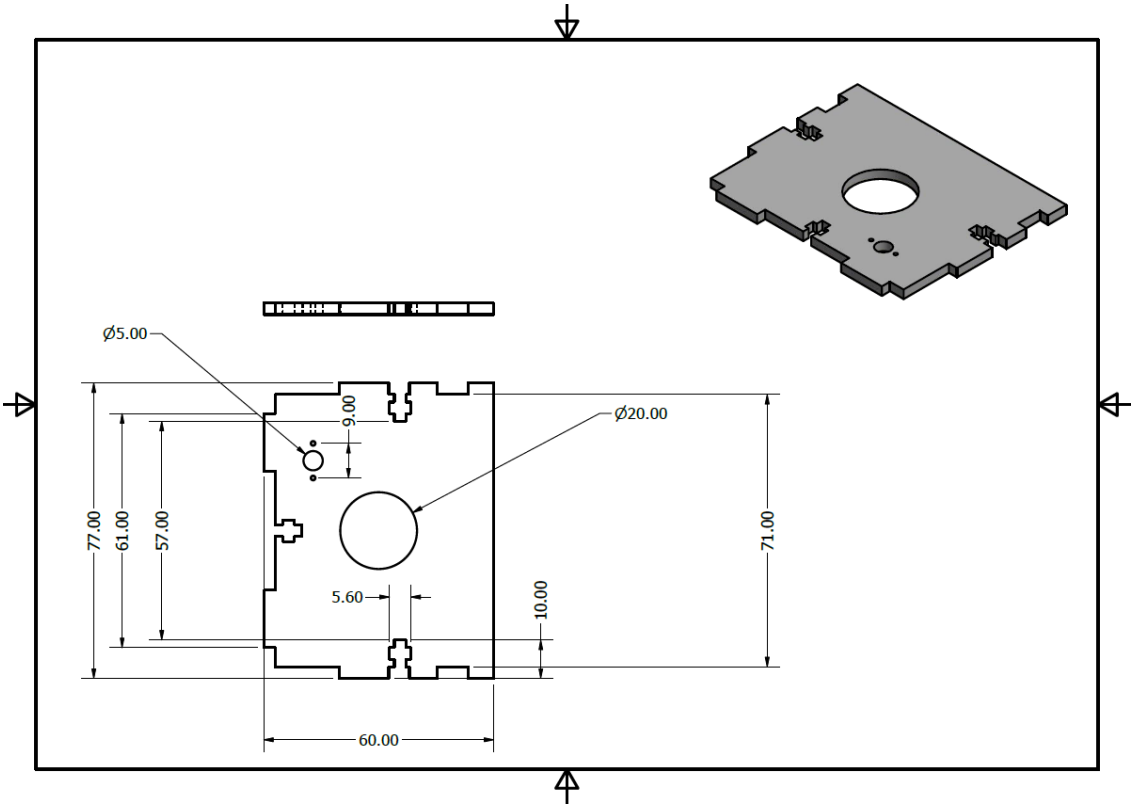

3.00
5.80
82.00
Ø42.20
71.00
77.00

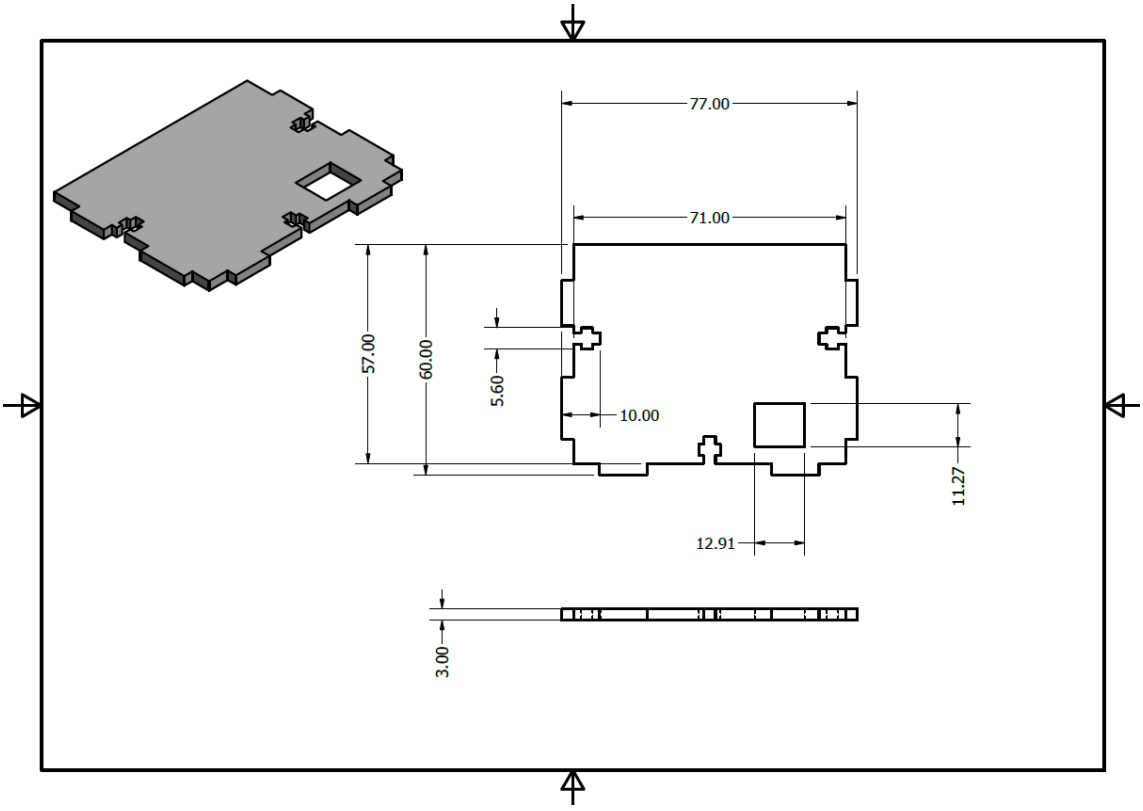

Ø3.00
57.00
3.00
Ø5.00
57.20
77.00
10.00
3.00
60.00
Ø20.00
Ø2.00
3.00

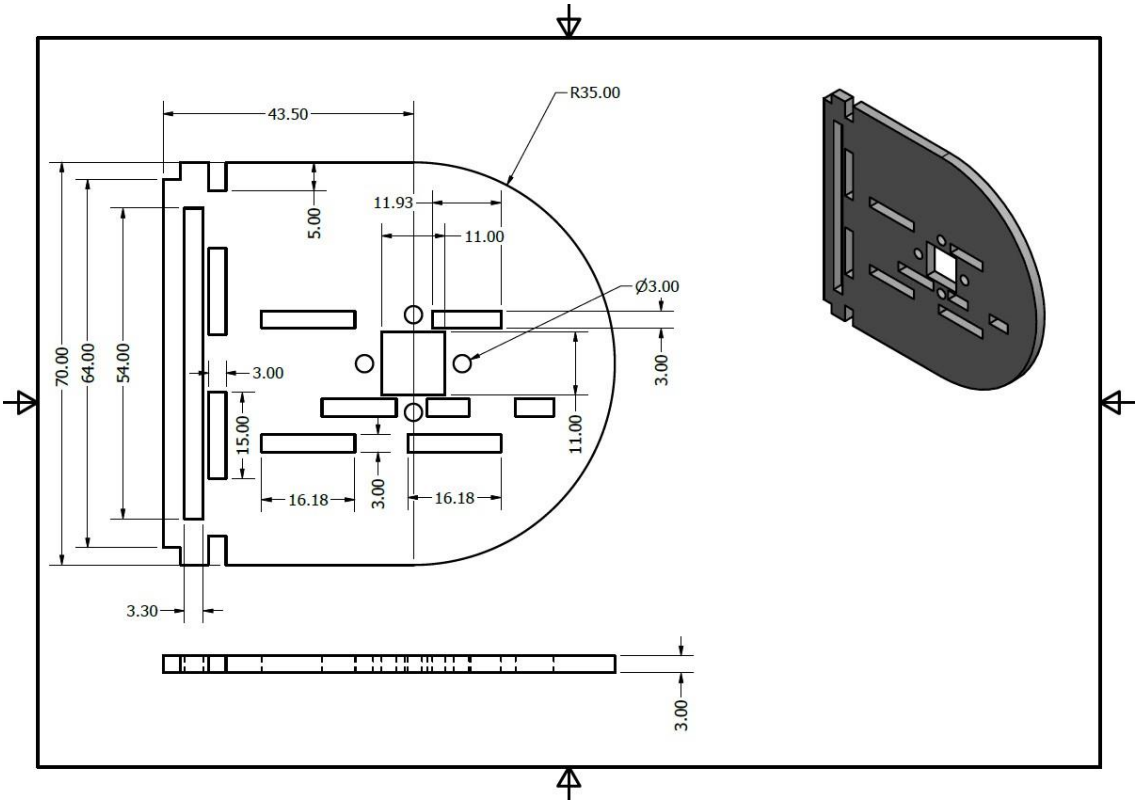

Ø27.00
27.00
5.60
5.60
12.30
Ø27.00
5.60
12.30
25.00
27.00
3.00
80.70
R5.00
36.15
24.70
5.00
2.00
Ø2.00
93.00
5.00
21.00
Ø2.00

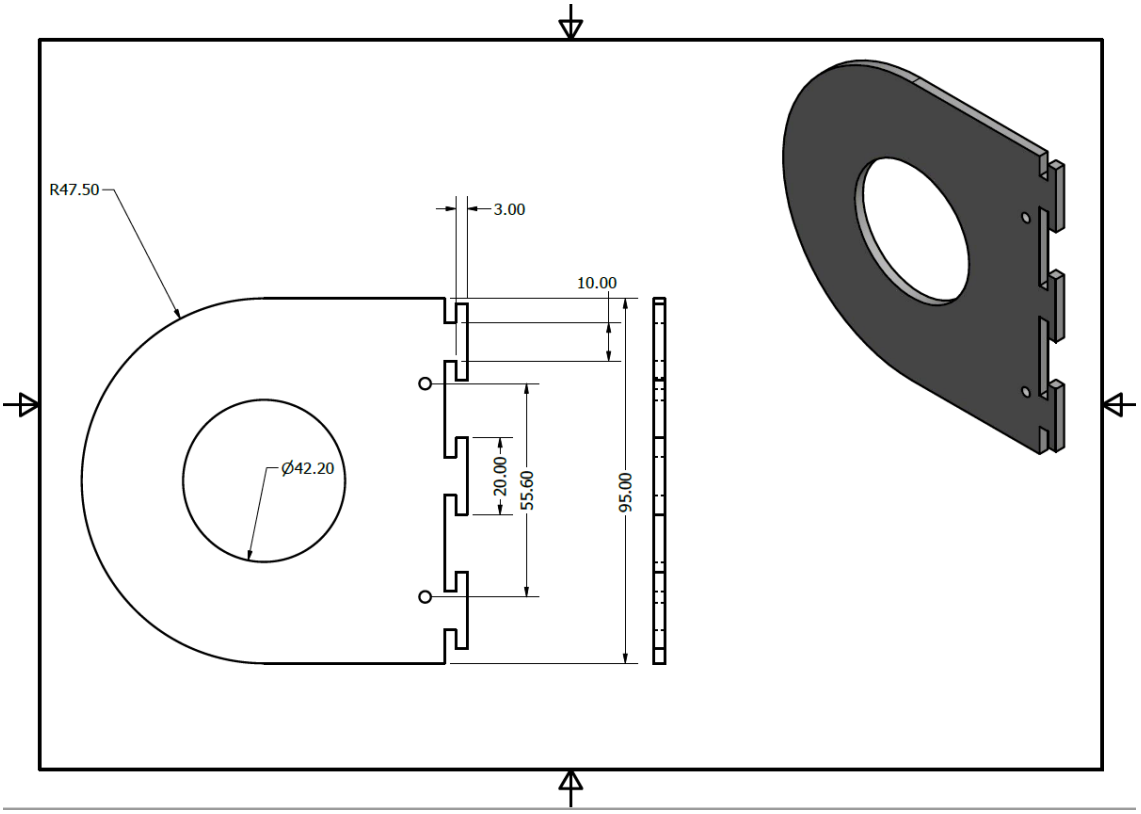

53.00
2.00
12.30
12.30
8.50
13.50
6.15
3.00
10.00
37.00
6.15
15.84
9.85
3.50
Ø2.00

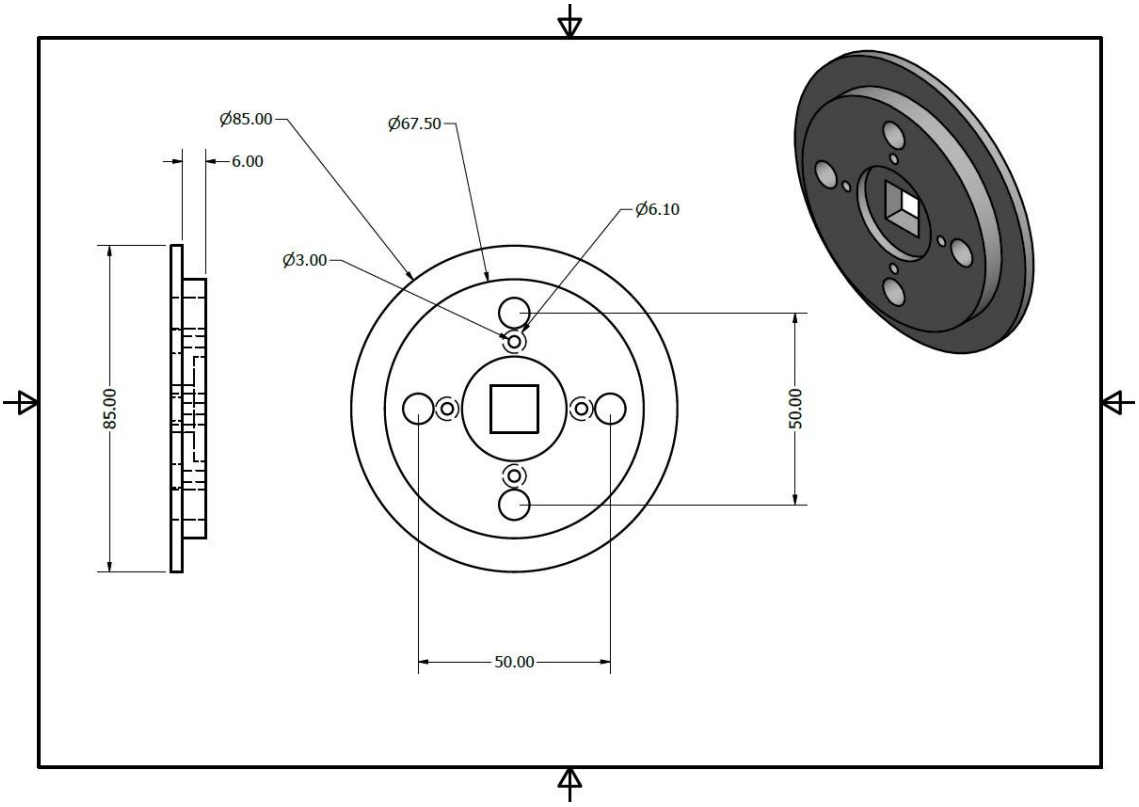


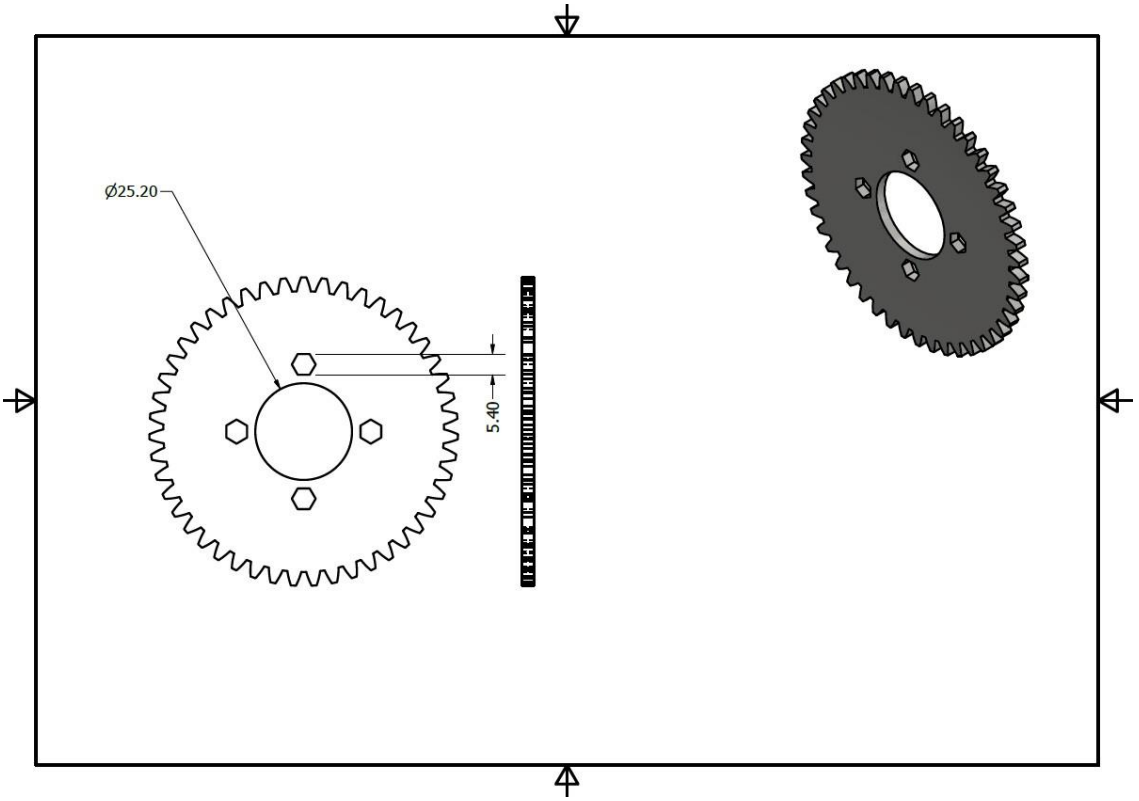

Ø5.00
9.00
Ø20.00
77.00
61.00
57.00
71.00
5.60
10.00
60.00

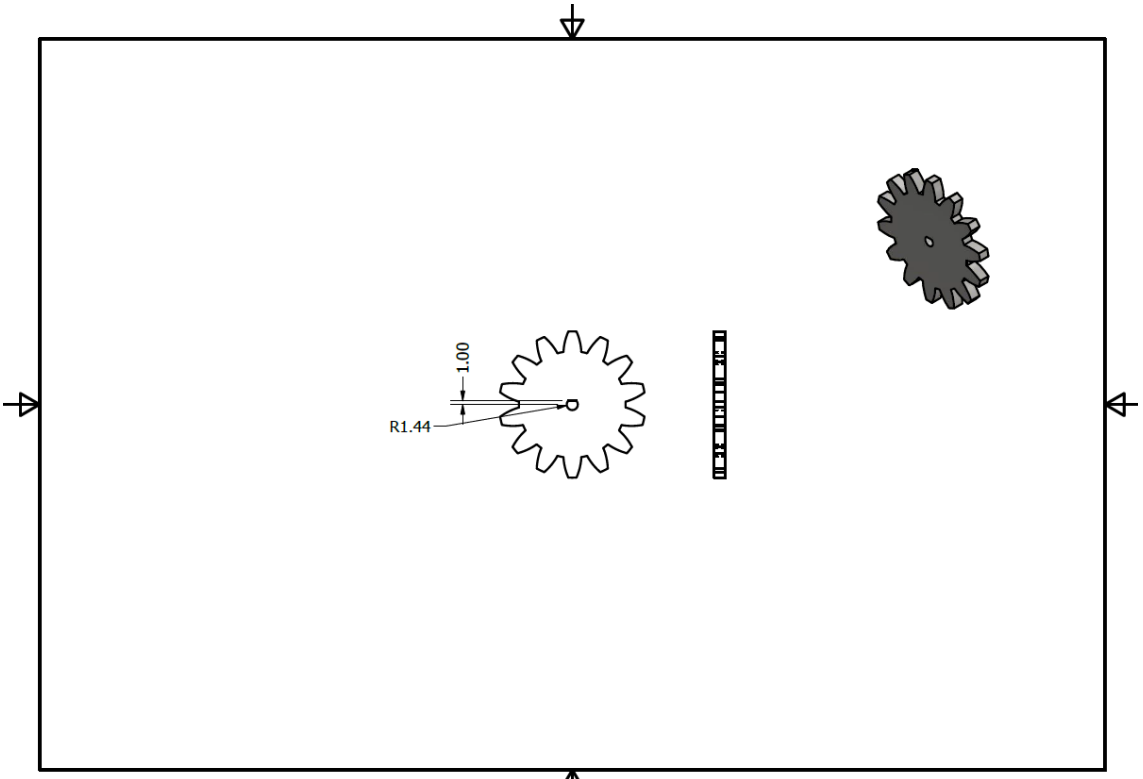

77.00
71.00
57.00
60.00
5.60
10.00
11.27
12.91
3.00

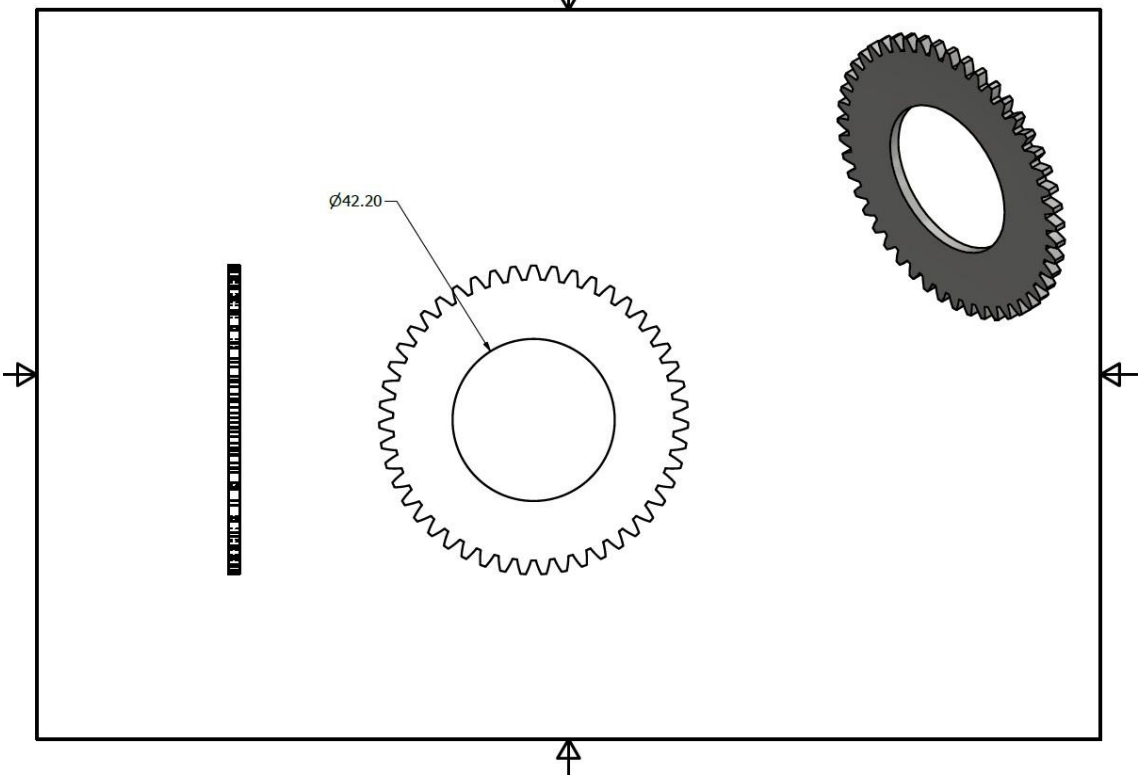

R35.00
43.50
11.93
11.00
5.00
Ø3.00
70.00
64.00
54.00
3.00
15.00
11.00
16.18
3.00
16.18
3.30
3.00

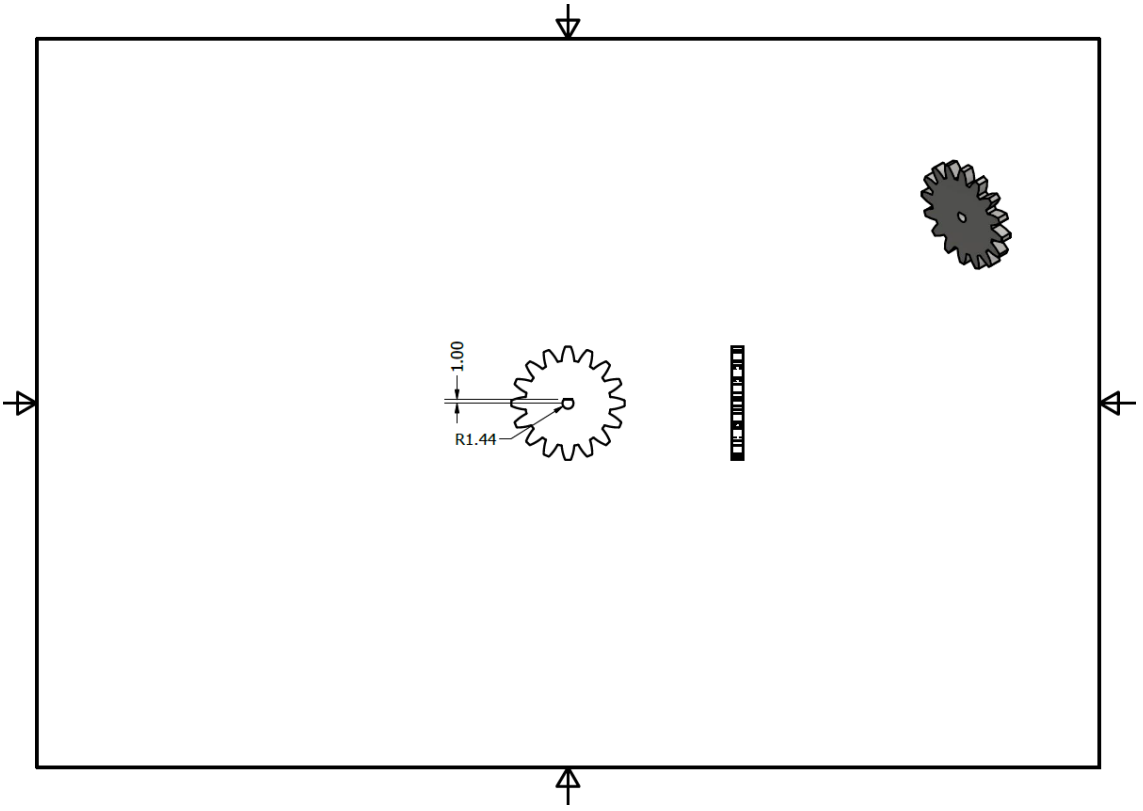

R47.50
3.00
10.00
Ø42.20
20.00
55.60
95.00

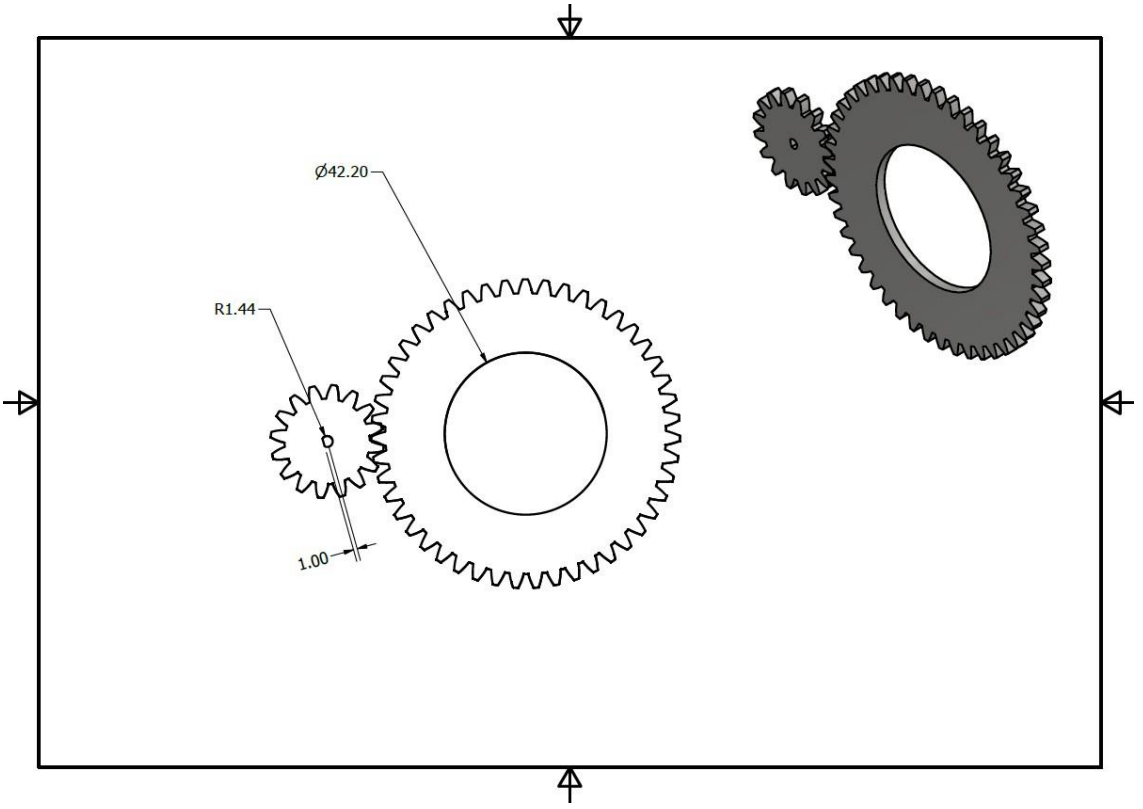

Ø85.00
Ø67.50
6.00
Ø6.10
Ø3.00
85.00
50.00
50.00

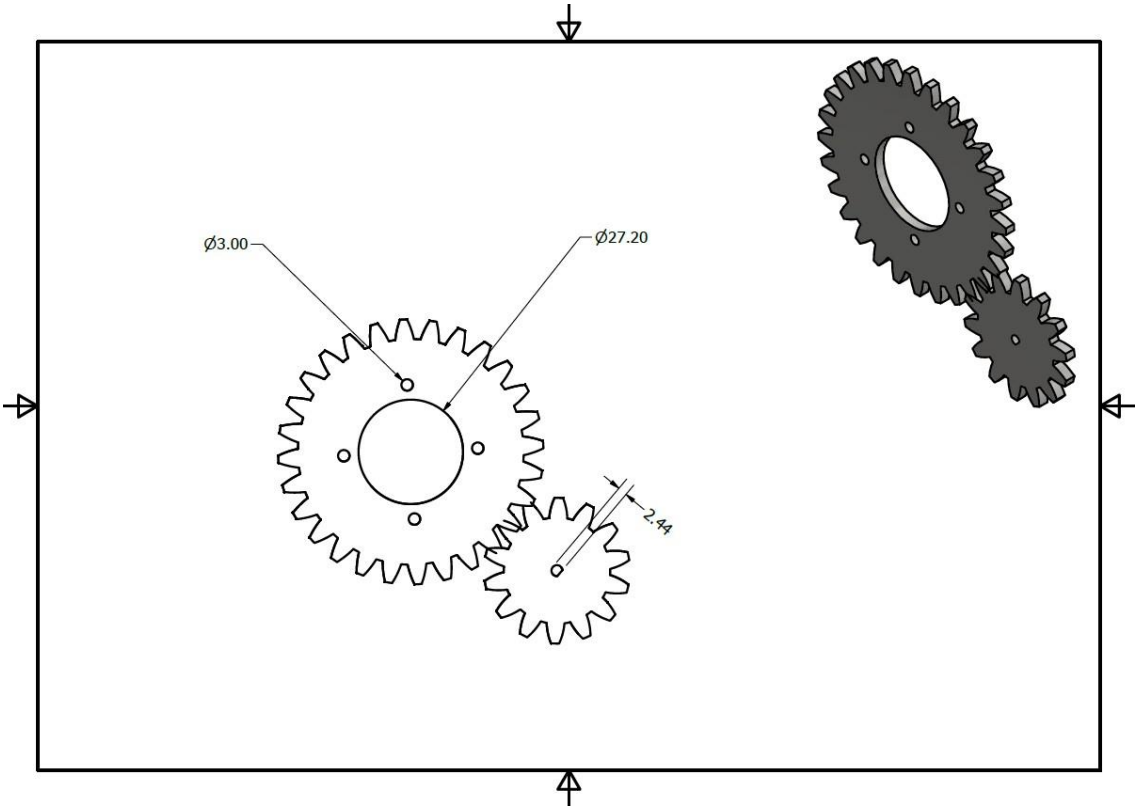

Ø25.20
5.40

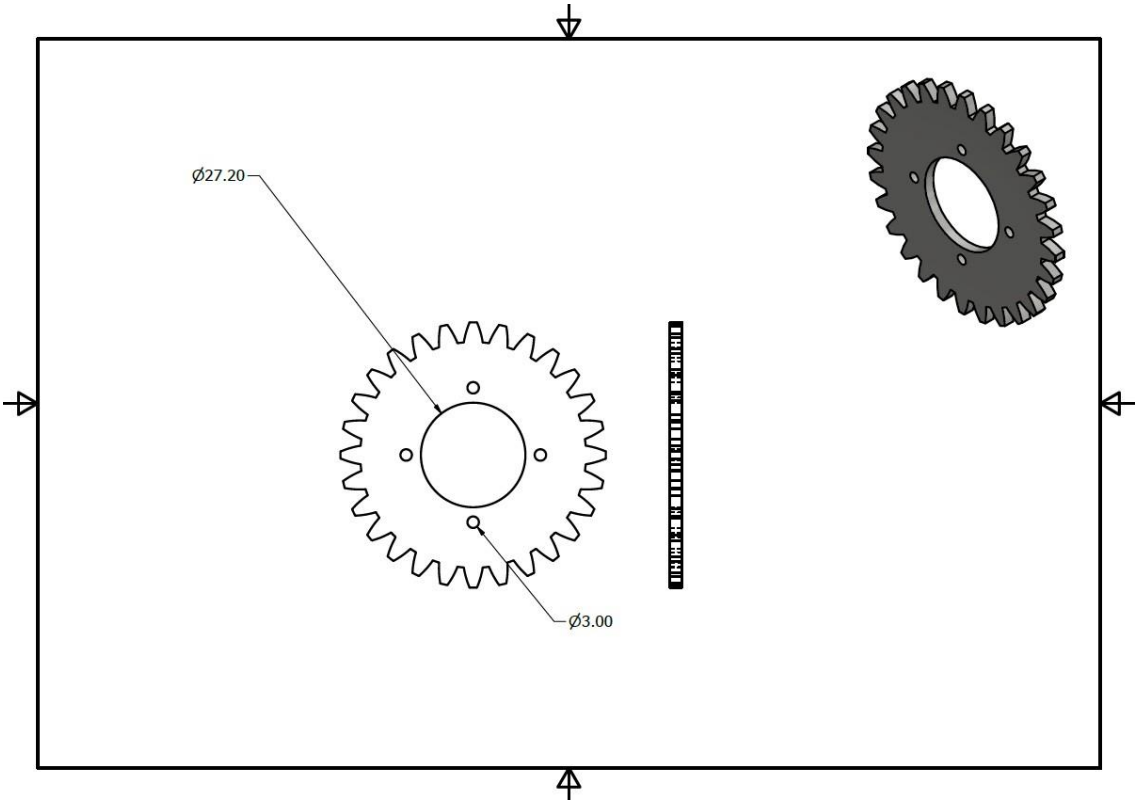

1.00
R1.44

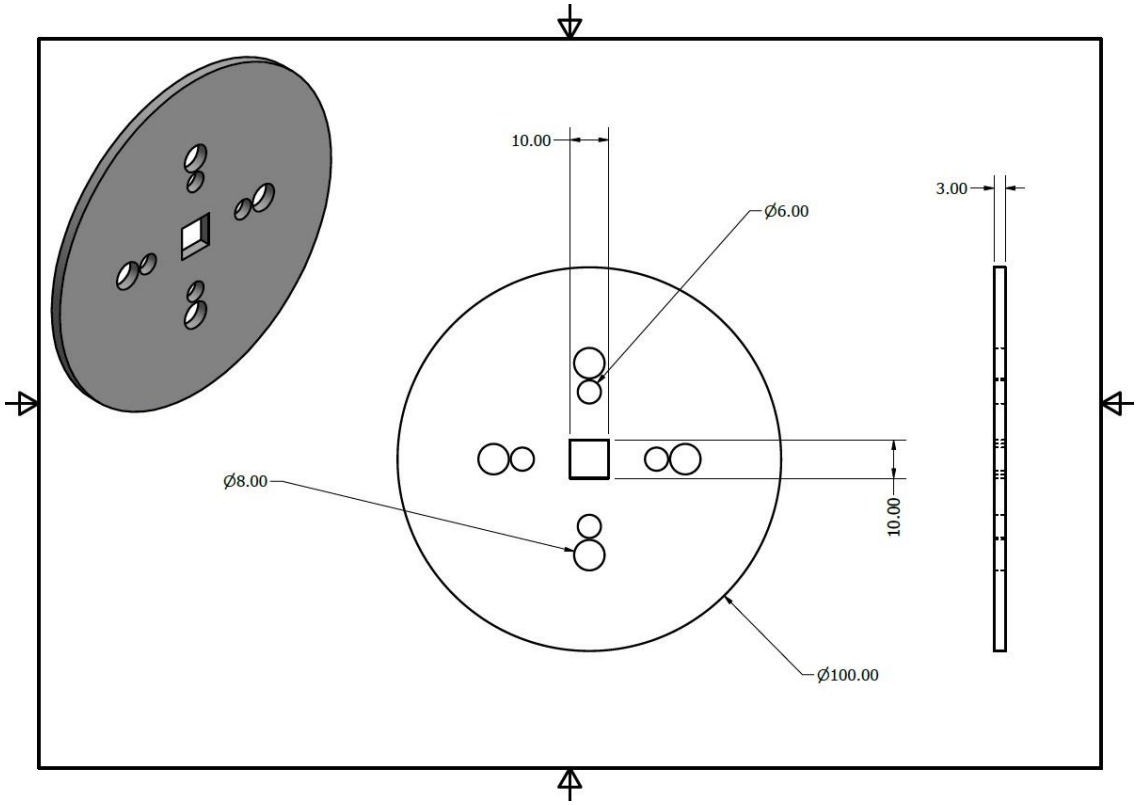

Ø42.20

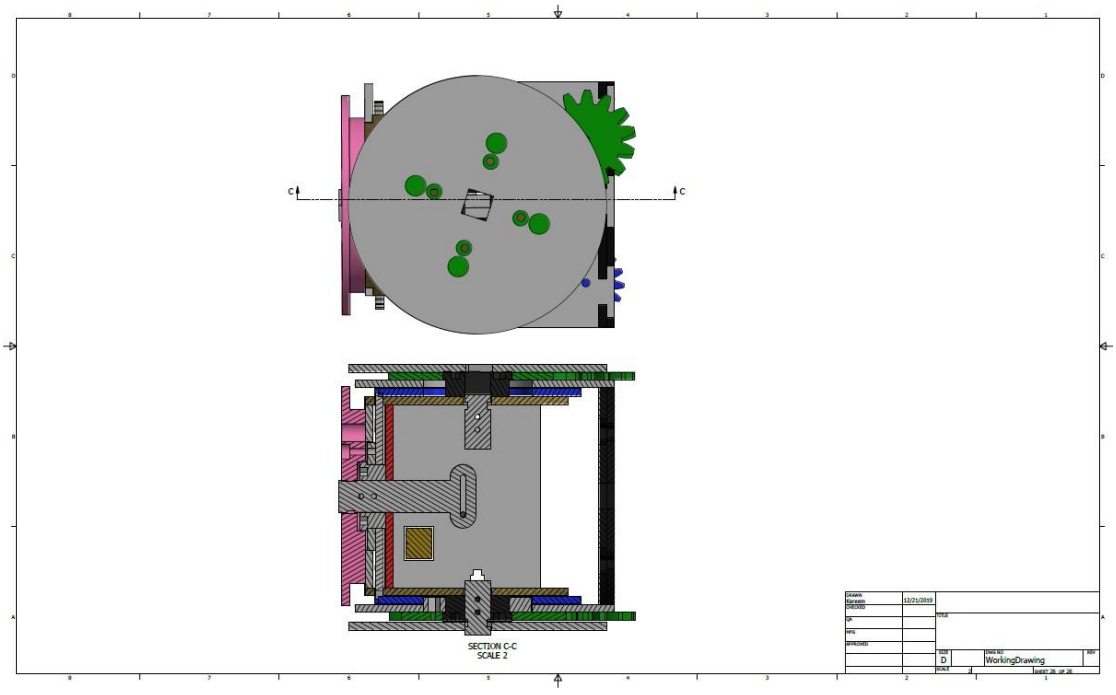

1.00
R1.44


Ø42.20
R1.44
1.00


Ø3.00
Ø27.20
2.44


Ø27.20
Ø3.00


10.00
Ø6.00
3.00
Ø8.00
10.00
Ø100.00


C
C
SECTION C-C
SCALE 2
12/21/2019
D
WorkingDrawing