ME F266 STUDY-ORIENTED PROJECT

STUDY ORIENTED PROJECT REPORT

Design and Simulation of Mobile Manipulator in a Bio-Tech Facility

ISHAN NIGAM (2016B1A40858P)

UNDER THE SUPERVISION OF

Dr. B. K. Rout, Professor

DEPARTMENT OF MECHANICAL ENGINEERING

Abstract

The current designs of bioprocess and bio-manufacturing require excessive labour and manual interventions. The bio-manufacturing industry devotes 35-50% of total cost in labour, the highest among all, compared to 10-14% in the next highest industry. Besides economic reasons, there exists safety reasons to implement automation in the bio facilities especially in the wake of recent Covid-19 like situations where minimal contact is of paramount importance and the medical staff are in shortage. The proposed idea is to design and simulate a mobile manipulator (robotic system) which is capable of transporting material within the facility and performing the procedural tasks using the manipulator. For simulation purpose Gazebo platform, an open-source multi-robot simulator has been chosen which is interfaced with ROS(Robot Operating System) to control and programme the mobile manipulator which exists in the environment designed in Gazebo. The system can later be designed in the real world after testing its functionality in the simulation environment Gazebo.
`

Keywords: SLAM, Robot Operating System, Gazebo, mobile robot, Bioprocess facility.

Table of Contents

List of Figures and Tables

1. Introduction

The Biotechnology industry in India is still devoid of much automation. Economical solutions are required for the industry within India and the current crisis due to the pandemic calls for an immediate focus to reduce human interaction in bio-facilities. The targeted application of Robotics in automation can decrease the costs substantially. Another challenge the automation will solve is its ability to work in the clean rooms, handle hazardous wastes and material, and reduce wastes and the time taken for the completion of tasks. The automation will find its role in drug discovery and development, medicine quality control, protein synthesis, medical facilities, clean room work, etc.
The biomanufacturing or Biopharma industry demands the production of cell and their products. Traditionally it has been a manual process requiring multiple hours of repetitive and laborious work to maintain perfect sterility. The rate of every action and the time required to subject such conditions are very strict to keep up the cell line viable and free of contamination. To free up such mindless work, robotic automation is required so that biologists and researchers can utilize their time focussing on other productive work. In a setup within a country like India, complete automation may not be fruitful. A type of compact solution for low scale production is shown in figure 1.). The small to medium laboratories which are in majority often do not have the resources to conduct a high-throughput screening for a Drug Discovery campaign. Here a small area does all the work with the help of a central manipulator.

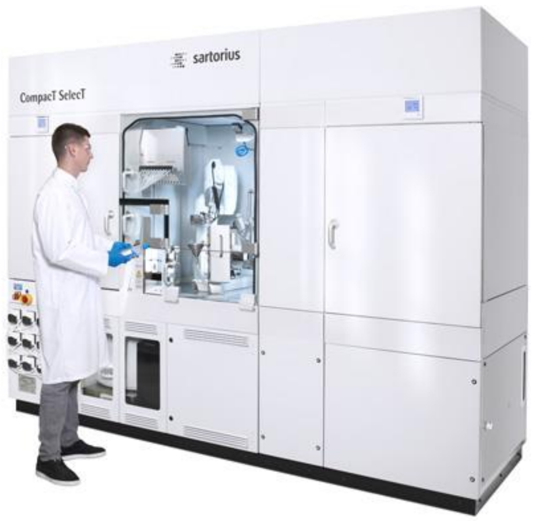


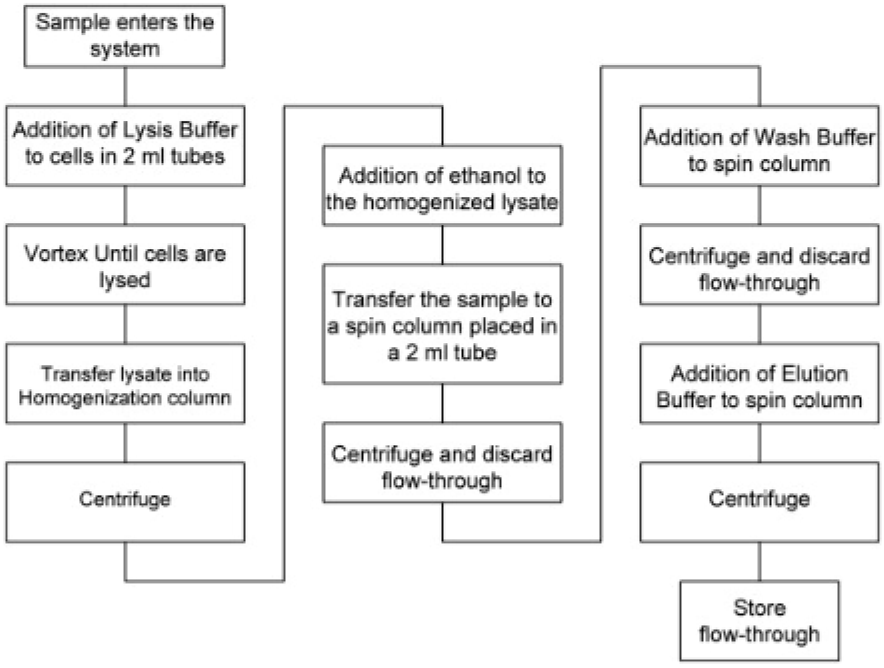
Figure 1.) A compact programmable workspace which performs automated cell culture for high quality cells and assay-ready plates – Sartorius of Tap Biosystems

The work in this report primarily focuses on the cell culture automation taking place in biomanufacturing industry or biotechnology labs. In this project robotic automation is explored. The basic idea is to divide the tasks in 2 parts.

A. One is to deliver materials across the facility using a mobile robot which can:
    a. Localize itself in the environment.
    b. Perform mapping to identify obstacles. (SLAM)
    c. Perform path planning and implement the algorithm.
    d. Receive input regarding its movement remotely. (not performed in this project)
B. A manipulator arm which can perform procedural tasks like handling test tube washing and managing, Dispense media, place glassware for autoclaving, pouring, centrifuging, incubation, purification, etc. These tasks have been described in Figure 5.). To perform such tasks, the forward and inverse Kinematics, Trajectory planning, control, etc is done using the MoveIt![15] Package of ROS. The in-depth assignment for all the aforesaid tasks is beyond the scope of this project but basic task of Grasp, Pick and place is attempted in Rviz and later will be simulated in gazebo environment to get an actual idea how the model will work in real world.

1.1 Literature Survey

An example of the area of work required in bio-technological labs is cell culture. Cell culture is a laboratory process used primarily for the production of cells and cell products. Traditionally, cell culture has been a manual process that requires many hours of repetitive, painstaking work in order to maintain absolute sterility. Furthermore, living cells require that a favourable environment be maintained throughout the cell culture cycle including cell growth, harvesting, reseeding, and analysis. Cell propagation requires careful control of media conditions and freezing and thawing rate in order to maintain viable and consistent cell lines. Proper training of cell culture professionals is essential to the success of maintaining viable cell lines. In order to improve the consistency of the cell culture process and to reduce the chances for cell contamination in laboratories that have high volume cell culture needs, pharmaceutical companies have turned to automated cell culture systems [3]. The current technologies revolve around a compact system such as that shown in Figure 1.), but in an Indian industry which are highly driven by economical feasibility and flexibility, the purchase and implementation of such machinery is an infeasible task. A type of semi automation is necessary for such an environment. This method involves breaking the compaction into many small parts where human intervention is minimally required. For example,


Sample enters the system
Addition of Lysis Buffer to cells in 2 ml tubes
Vortex Until cells are lysed
Transfer lysate into Homogenization column
Centrifuge
Addition of ethanol to the homogenized lysate
Transfer the sample to a spin column placed in a 2 ml tube
Centrifuge and discard flow-through
Addition of Wash Buffer to spin column
Centrifuge and discard flow-through
Addition of Elution Buffer to spin column
Centrifuge
Store flow-through

Download : Download full-size image

Figure 2.): The preparation of RNA from tissue-cultured cells. The set of process steps are first laid out conceptually as a manual-labor production line. Such a production line presentation, with its sequential stages, serves as a basis for a further deductive process as to how each station might be automated [25].

- Integra BioSciences, Inc. (www.integra-biosciences.com), offers a series of automated devices working together to fill, emboss, and stack multiple petri dishes.
- he AcuSyst-XCELL system (www.biovest.com) is a self-standing floor system comprised of an incubator and refrigerator section as well as control fixtures and a pump panel.
- Sterogene Bioseparations, Inc. (www.sterogene.com) integrate a hollow fibre bioreactor, which is a semi-automated system for culturing mammalian cells, with a protein purification system, comprised primarily of an 7mmune-adsorbent cartridge column [17].

This way 3 tasks-

1. managing petri dishes/plates,
2. storing/incubating them with their contents and
3. extracting proteins from them is accomplished.

A type of robotic automation system configuration is called "tower-based configuration" [18]. Tower configuration consists of two arms mounted on a common cylindrical base and is surrounded by stackers that carry laboratory processing instruments, lab-wares, and other accessories. Two major operations in automated laboratory systems are automated liquid handling and automated tube/plate transportation. The former is managed by a manipulator arm and the later by a mobile robot.

The first step is deployment of a mobile robot. For this purpose an Open Source Robotics Software-Robot Operating System (ROS) by https://www.openrobotics.org/ is used along with an open-source 3D robotics simulator- Gazebo http://gazebosim.org/. The Philosophy of ROS revolves around making a piece of software that could make a variety of robots programmable in a standard architecture. The robots could be programmed in C++ or python and their model can be developed or converted into urdf xml format such that they are defined in the form of linkages and joint. The details about working of ROS are unnecessary. The major concepts are beautifully described in [16].

This study is unique in the manner that very few papers have been published related to problems faced while automating bio-facilities. As discussed previously, there is a long way to go in automating this sector and in this project, I have tried to study and implement a basic framework which could later be built upon and optimized. The project involves using a mobile robot to map the environment, localize itself, pick a virtual object from a place and deliver the object to another position in the custom designed world while planning its path in a dynamic environment. The complete code for this process is viewed on Gazebo and the architecture for programming the system is provided by robot operating system (ROS).

The project mainly covers the mobile robotics part. The deployment of mobile robot is divided into following steps:

1. Design of facility layout and robot in Gazebo
2. Localization or determining the position of robot in that environment
3. Mapping the environment
4. Planning the path to go receive a parcel and then deliver it in a dynamic environment

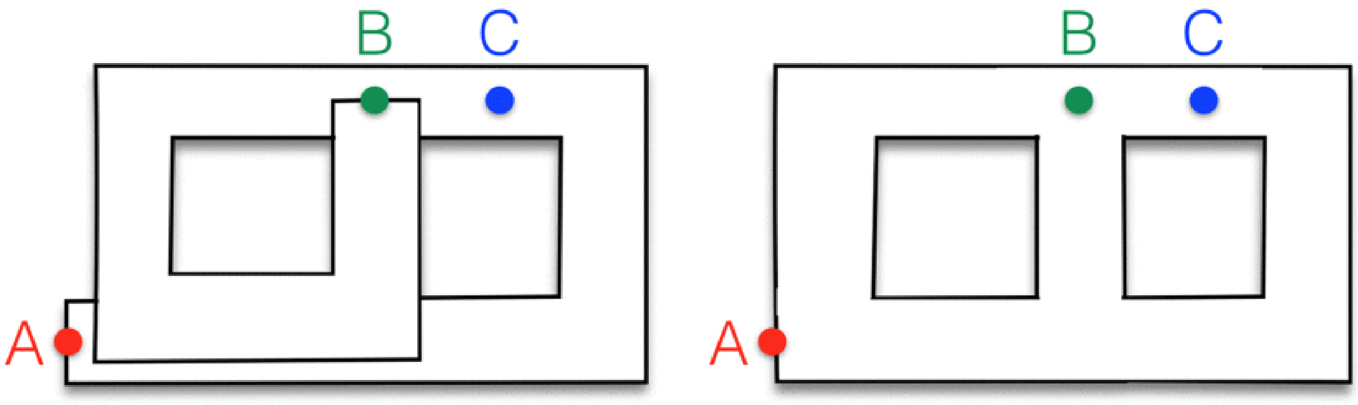
The 1st step is covered in methodology. The 3 steps are first studied and then implemented. They are as follows:

A. SLAM

The major issue that a mobile robot needs to address to be truly autonomous is whether it can be placed at an unknown location and an unknown environment where it can incrementally build a consistent map of the environment it is placed in while simultaneously establishing its accurate position within the map it creates. A solution to this problem is the dogma around which autonomous mobile robot systems revolve around. Multiple algorithms

Simultaneous Localization and Mapping (SLAM) is a technique for obtaining the 3D/2D structure of an unknown environment and sensor motion in the environment. This technique was originally proposed to achieve autonomous control of robots in robotics. SLAM is a tool commonly used for the accurate estimation of the robot's position relative to the environment while simultaneously mapping the environment around. The need to use a map of the environment is twofold. First, the map is often required to support other tasks; for instance, a map can inform path planning or provide an intuitive visualization for a human operator. Second, the map allows limiting the error committed in estimating the state of the robot. In the absence of a map, dead-reckoning would quickly drift over time; on the other hand, using a map, e.g., a set of distinguishable landmarks, the robot can "reset" its localization error by revisiting known areas (so-called *loop closure*). Therefore, SLAM finds applications in all scenarios in which a prior map is not available and needs to be built.

The question we ask is that "Do autonomous robots really need SLAM?". SLAM is different from a simple odometric method in the sense that SLAM aims at building a globally consistent environment representation using both the odometric/sensor readings along with a loop closure method. Earlier, only the odometry (reading from wheel/actuator motion) was utilized for this purpose and the inaccuracies as shown in figure 2.) pushed to finding new ways.



Firgure 3.) Left: map built from odometry. The map is homotopic to a long corridor that goes from the starting position A to the final position B. Points that are close in reality (e.g., B and C) may be arbitrarily far in the odometric map. Right: map build from SLAM. By leveraging loop closures, SLAM estimates the actual topology of the environment, and "discovers" shortcuts in the map [20].

The complete analysis of SLAM is beyond the scope of this project. However, the technique used in the project is Grid-based FastSLAM algorithm which is present in the ros gmapping package http://wiki.ros.org/gmapping. It uses a combination of Monte Carlo Localization and Occupancy Grid Mapping Algorithm.

Monte Carlo Localization: MCL algorithms represent a robot's belief by a set of weighted hypotheses (samples represented by particles as shown in Figure 4.)), which approximate the posterior under a common Bayesian formulation of the localization problem. A Sample based density concept, this algorithm uses the concept of particle filters where N particles are randomly distributed around the location and are assigned a weight and position (x, y, z, r, p, y). As the robot moves, the values of pose are updated in all the particles calculated from the movement control. Then new set of N particles are randomly chosen from the previous list of N particles with repetition. Each particle is randomly drawn after a motion of the robot such that the number of particles (N) remains constant. The particle with more weight is allotted more chance for random selection and the weight is determined based on the Probability of particle by Bayes filter ($P(X|Z_t)$), where X is particle state, and Z are measurements of robot up to time t.[1]

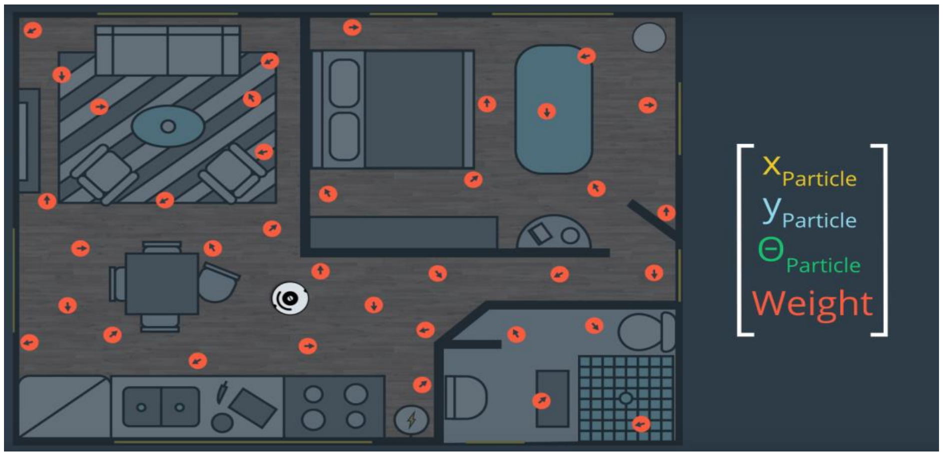
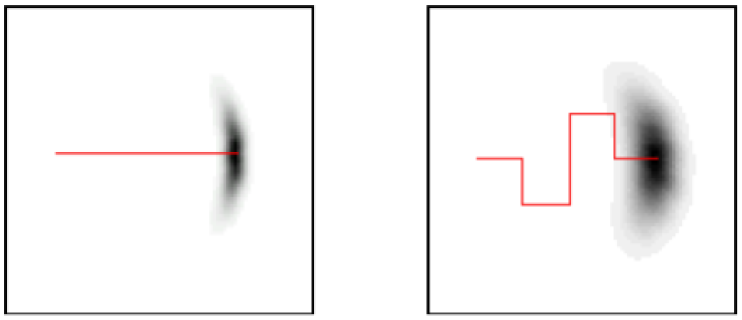


Figure 4.) Representation of randomly allotted particle filters. The right information in the brackets denote position and orientation.



The density $p(x'|x,a)$ after moving 40 meter (left diagram) and 80 meter (right diagram). The darker a pose, the more likely it is.

Figure 5.) A paper by Sebastian et al. [1] describes the visualization of working of algorithm.

Occupancy Grid Mapping Algorithm: This basically discretizes the environment in cells and assigns a probability of occupation to each cell. It is based upon dividing the space into a grid of 1s and 0s which represent occupied or unoccupied space. When a sensor such as lidar detects an occupied grid, it assigns the cells within its range of view a log odds ratio.[21] A visual description of such discretisation is shown in Figure 6.).

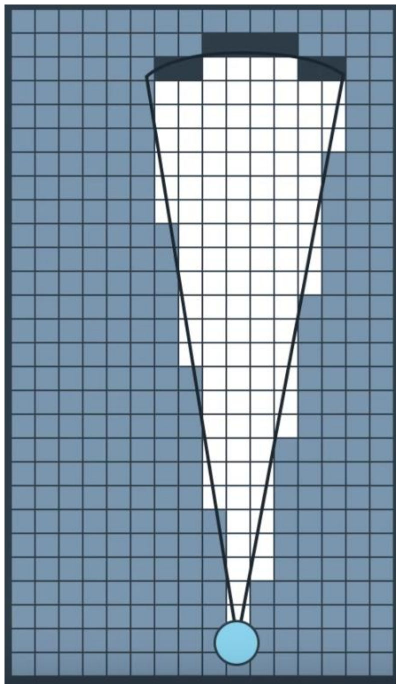


Figure 6.) Blue squares represent undiscovered area, black squares represent occupied area, white squares represent vacant area and blue circle represent the robot position. This figure displays the working of Occupancy Grid Mapping Algorithm.

Grid-based FastSLAM: FastSLAM, an algorithm that recursively estimates the full posterior distribution over robot pose and landmark locations yet scales logarithmically with the number of landmarks in the map. This algorithm is based on an exact factorization of the posterior into a product of conditional landmark distributions and a distribution over robot paths [22]. FastSLAM decomposes the SLAM problem into a robot localization problem, and a collection of landmark estimation problems that are conditioned on the robot pose estimate. This used the probability of weight of particles, the localization belief probability from MCL and the estimated map from occupancy grid mapping to find the product and thus the belief of our Map and our pose estimate with respect to the map. FastSLAM employs a particle filter for estimating the path posterior, using a filter that is similar (but not identical) to the Monte Carlo localization (MCL) algorithm [1].

B. Path Planning and Navigation

In this project, we will be using the ROS Navigation stack, which is based on the Dijkstra's, a variant of the Uniform Cost Search algorithm, to plan our robot trajectory from start to goal position. Uniform cost search is a type of Graph Search. This is an algorithm for finding the optimum path. Because this algorithm searches for the minimum-cost path among all paths in order, beginning from the starting point, the search region expands concentrically [23]. This method thus has the disadvantages of poor search efficiency and a long search time when the distance to the destination is large, but we aren't affected by this disadvantage because the distances we plan are relatively shorter
It is a type of uninformed search algorithms which are not provided with any information about the whereabouts of the goal, and thus search blindly. Here a cost is assigned to each edge of the web of node connections. A difficult path holds a higher cost. The working of this algorithm is mostly a black box since it utilizes the ROS Navigation stack [24], and no extra code has been written for this particular algorithm.

2. Design Methodology

The Method is divided into 6 parts:

A.) Designing a step by step comprehensive list of tasks which are procedural and are typically done by a lab worker which can be handled by our proposed manipulator arm.
B.) Designing a workplace layout which is efficient and covers the needs of a lab. It has to be based on configurations based on [18] and plan of tasks described in [25].
C.) Designing a mobile robot with 2 wheels which can use a differential plugin in gazebo and can navigate around. Although initially we used a turtlebot – http://wiki.ros.org/Robots/TurtleBot, We can soon convert to our own robot after equipping it with all necessary sensors.
D.) Choosing a manipulator arm for our work.
E.) Executing the mobile robot, we designed to work in the 3D layout.
F.) Execute the panda robotic arm.

E.) List of Tasks for the manipulator arm

.

Stations no.	Role ID	Role definition
1	1.1.1	Grip individual glassware from pallet placed in ‘to be washed’ stack
	1.1.2	Use one arm to hold and the other arm to grip the cap and turn predefined rotations to open the lid
	1.1.3	Move arm towards Water jet with predefined motions to ensure complete rinse
	1.1.4	move the glassware towards rotating detergent laden brushes wait for some time
	1.1.5	Repeat step 1.1.3
	1.1.6	Use Visual sensor to check for leftover material on glassware
	1.1.7	if yes: Repeat from step 1.1.3; else: Scan and deregister the rfid code assigned to glassware, Place the glassware back into pallet move to step 1.2.1
	1.2.1	Place the pallet for the particular glassware in its respective stack
	1.3.1	When demanded, pick the pallet from the stack, register the glassware for the required task, carry it to the assigned station and place it at the allocated area.
2	2.1.1	Pick the flask or stock media holding glassware from its allocated area.
	2.1.1	Step 1.1.2
	2.1.2	Move the arm to a position under the dispenser of media assigned by worker and the pump remotely connected dispenses assigned quantity.
	2.1.3	If media dispensing left: Repeat step 2.1.2 else move to step 2.1.4
	2.1.4	Step 1.1.2
	2.2.1	move the glassware towards the shaker and place the glassware there and let it be for assigned time
3	3.1.1	Pick the flask or stock media holding glassware from its allocated area.
	3.1.2	Open the door of autoclave with one arm and place the pallets of filled glasswares within the autoclave. Close the door similarly.
	3.1.3	After prediefined time period remove the content similarly and proceed
	3.2.1	If pouring required: proceed; else: move to next station
	3.2.2	Pick the glassware to be poured in, repeat step 1.1.2 and place it in its allocated ‘pour area’
	3.2.3	Place its cap down in allocated area
	3.2.4	Place the glassware in allocated area
	3.2.5	Pick glassware with autoclaved media and repeat step 1.1.2 then steop 3.2.3
	3.2.6	Tilt the glassware with media on over the petridish by a predefined calculated amount and hence pour the media.
	3.2.7	Repeat step 1.1.2 for both the used glassware
	3.2.8	Place the glassware in pallet
	3.2.9	Move to next station
4.	4.1.1	To be decided as per the biotech speciality industry where robot system deployed
5	5.1.1	Open the door of incubator with one arm and place the pallets of coated dish laid with cells within the incubator. Close the door similarly.
6	6.1.1	Station where the workers work
	6.1.2	A mobile line following robot with cabinet which holds material placed by the worker or manipulator-at-stations-1-4.

Figure 7.): The workflow which was designed by me after carefully analysing all the procedure followed during cell culture which involves a lab worker moving about around a small space and working. The

data is useful for the future work where a manipulator arm needs to be programmed beyond the basic pick and place and actual execution is carried out.

The figure 7.) is compiled from the following set of tasks described in detail:

The main idea is to divide the complete set of tasks in the industry into 5 parts and automate 4 of these.

1.) Glassware station

This station Receives all the used glassware from the arm1. The arm1 Uses the following procedures. Brushes with wooden or plastic handles are recommended as they will not scratch or otherwise abrade the glassware's surface.

a.) Wash: When washing, soap, detergent, or cleaning powder (with or without an abrasive) may be used During the washing, all parts of the glassware should be thoroughly scrubbed with a brush. This means that a full set of brushes must be at hand, including brushes to fit large and small test tubes, burets, funnels, graduates, and various sizes of flasks and bottles. Motor driven revolving brushes must be used. The arm1 can place the glassware against the brushes after laying it in the detergent. Later after enough time coated with detergent, the glassware can be rinsed either by placing it in clean water or by using water jets. Imaging can be used to analyse if the glassware is clean or if it requires another cycle of the process. Other chemicals like acids or alcohol required specifically for certain cleaning can be used via dispensing system.

b.) Place: After the wash, the manipulator can stack the glassware in the pallet.

c.) Store: Pallet holders can be used to hold a set of the glassware. These pallets can be stacked over each other in a cabinet like system where the manipulator can reach out and pick as programmed.

d.) Register: The researcher or worker can select remotely what they want and the manipulator will then pick and register the glassware for the task. This glassware will undergo all the dispensing and other work as per the worker has programmed

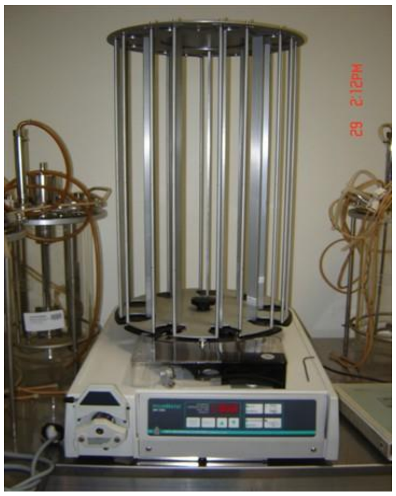


Figure 8.) Petri Dish holders

2.) Media Dispenser
   a.) De-capping: The glassware received from the previous stationed needs to be de-capped. The number of rotations or force required to decap for standard glassware like petridish, tubes, vials, etc. is already known and needs to be fed to the programme. This decapping is achieved via a gripper and holder combination to provide torque or force.
   b.) Dispensing media: Once the glassware is decapped, the arm1 moves the glassware to the platform with media dispensers. Such dispensers may use a peristaltic pump instead of a micropipette to dispense the media in a sterile and precise manner. Despite being expensive, very few such pumps are required vs the high number of micropipettes lying around a general laboratory. The other end of the pump tube can be attached to the container containing the media. Similarly automated powder dispensers can be used. Such a dispenser that is capable of dispensing solids such as powders, resins, microbeads, beads etc. It uses a metal plate containing holes to hold a specific amount of solid material to be dispensed. After the holes are filled with 'powder', a slider is pulled that opens the holes also called the 'dropdown method', letting the powder drop into a tube, well or microtiter plate.

Figure 9): A peristaltic pump, a type of positive displacement **pump** used for **pumping** a variety of fluids, without getting them in contact with any machinery and thus maintaining the purity.

   c.) Re-capping: Similar to decapping, recapping can be done
   d.) Shaker: Used to mix the contents of the media. Fig. below.

Figure 10.) A Shaker with a primary function to mix the content of the fluid in a gentle manner.

3.) Autoclave and Pouring:
This section is within a laminar flow so that a complete sanitized environment is maintained.

a. The media and the glassware needs to be sterilized and this is done by Autoclaving. Autoclave is a process of exposing the glassware and it’s content to steam to kill all bacteria so that the cell which need to be cultured are not affected by external microflora. This can be achieved by simply placing the glassware in the autoclave. The arm1 places the palette from the previous station into the autoclave and then there is waiting time as per the protocol.

b. Once the media is autoclaved, which is contained inside a flask or beaker, it needs to be poured in multiple petri dish or plate, where the required cell line would grow. The arm1 can tilt the flask with media over the petri dish and coat the dish/tube with the media over/within which the cell line would grow. This can be allowed to solidify or remain liquid as per the content.

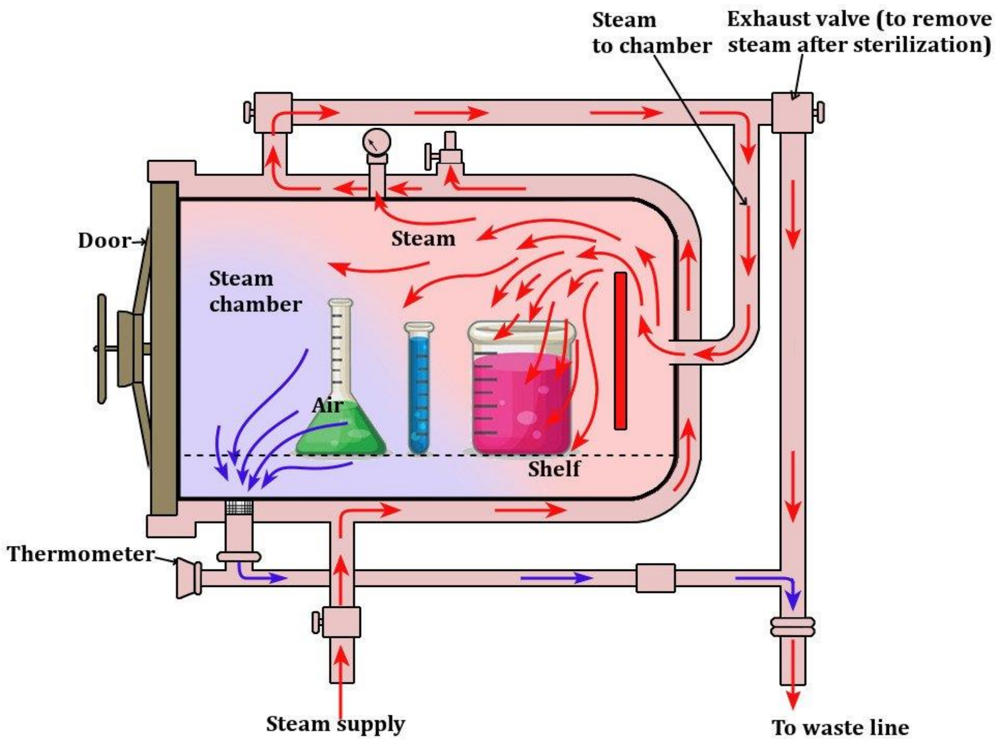


Figure 11.): Working of an autoclave which is used to disinfect any foreign bacteria/cell in a media by exposing the media to a constant 100 degree Celsius by maintaining the steam.

4.) Centrifuge, Chromatography Purification, incubate: This station may receive the pallet or glassware from the worker to extract the required protein which is the ultimate goal of cell culture.

   a. Centrifuge: Used to get all the proteins produced by the cell by forcing the content of the liquid to bottom via centrifugal force (forced sedimentation). The tube/vial can be placed via the arm1 into the centrifuge and the number of cycles and other settings can be set remotely by the research for the particular batch of sample sent by him/her.

   b. Purification: Different industries have different methods of purification and the task can be customized accordingly in the station.

Figure 12): Centrifuge: Used to vigorously separate the solutes from the fluid content in a test-tube by means of centrifugal motion.

5.) Incubator: An ideal environment for the cells to grow.

Figure 13.): Incubator maintains required temperature, humidity, pressure, etc.

6.) To the worker at their respective stations where they have the software system through which they can simply order what basic starting job for research/ sample testing they require. The work done by the worker is:
   a. Put the cells in the media coated dish they receive.
   b. Separate protein from the cells they got incubated.
   c. Programme the robot system what to do.

B.) **Design of workspace**

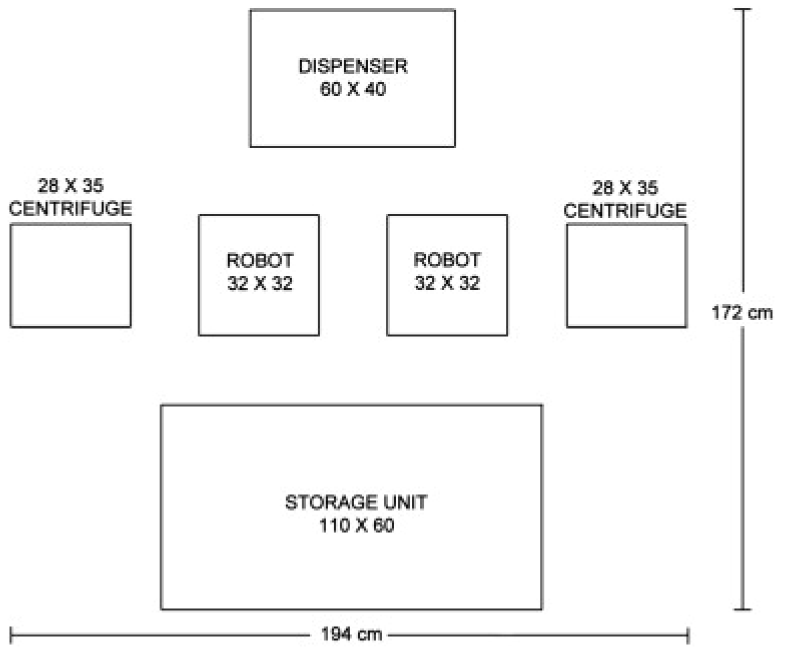


Figure 14.) One of the layouts shown in [25] which tries to achieve an efficient use of workspace.

From the literature survey figure 2.) and figure 14.) I proposed a layout as shown in Figure 15.)

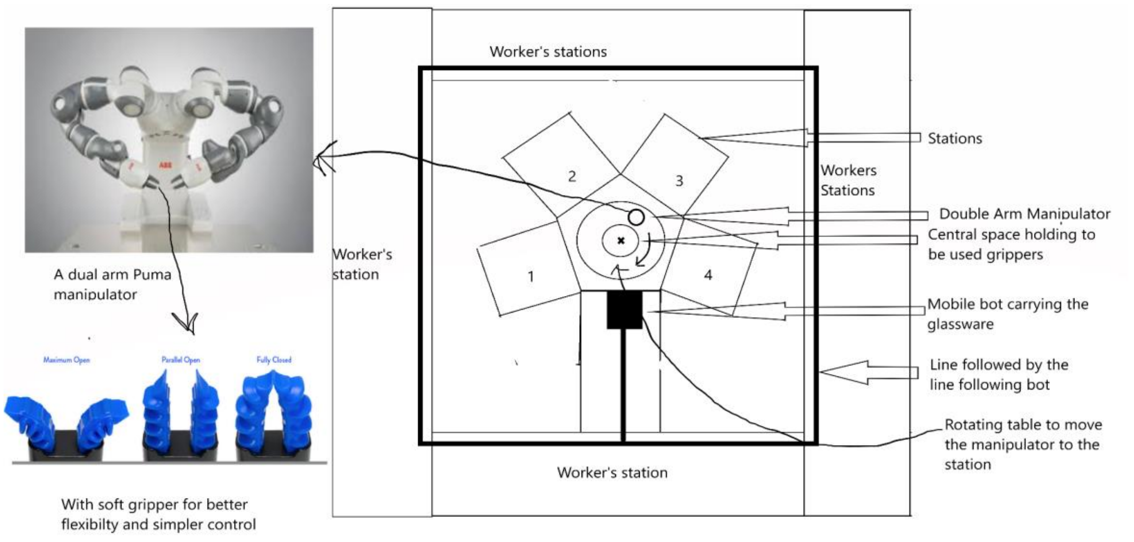

A dual arm Puma manipulator
Maximum Open
Parallel Open
Fully Closed
With soft gripper for better flexibilty and simpler control
Worker's stations
Worker's station
Worker's station
Workers Stations
1
2
3
4
Stations
Double Arm Manipulator
Central space holding to be used grippers
Mobile bot carrying the glassware
Line followed by the line following bot
Rotating table to move the manipulator to the station

Figure 15.) A sketch up of workspace and labels on its functionality

This layout tries to bring reduce the foot-space and condenses all the procedural task to the central workspace. The central pentagonal boxes will hold the "tower based configuration" discussed in [18]. It can hold materials like

- Centrifuge
- Incubator
- Media Dispensers
- Laminar airflow
- Pallet holder stack
- Refrigerator
- Protein purification setup
- Etc.

In each of its cubicles. The area surrounding (corridor) it can be used by employees for their work. The rest of the complete path can be used by the mobile manipulator to traverse. This was translated the world in Gazebo as shown in figure 16.)

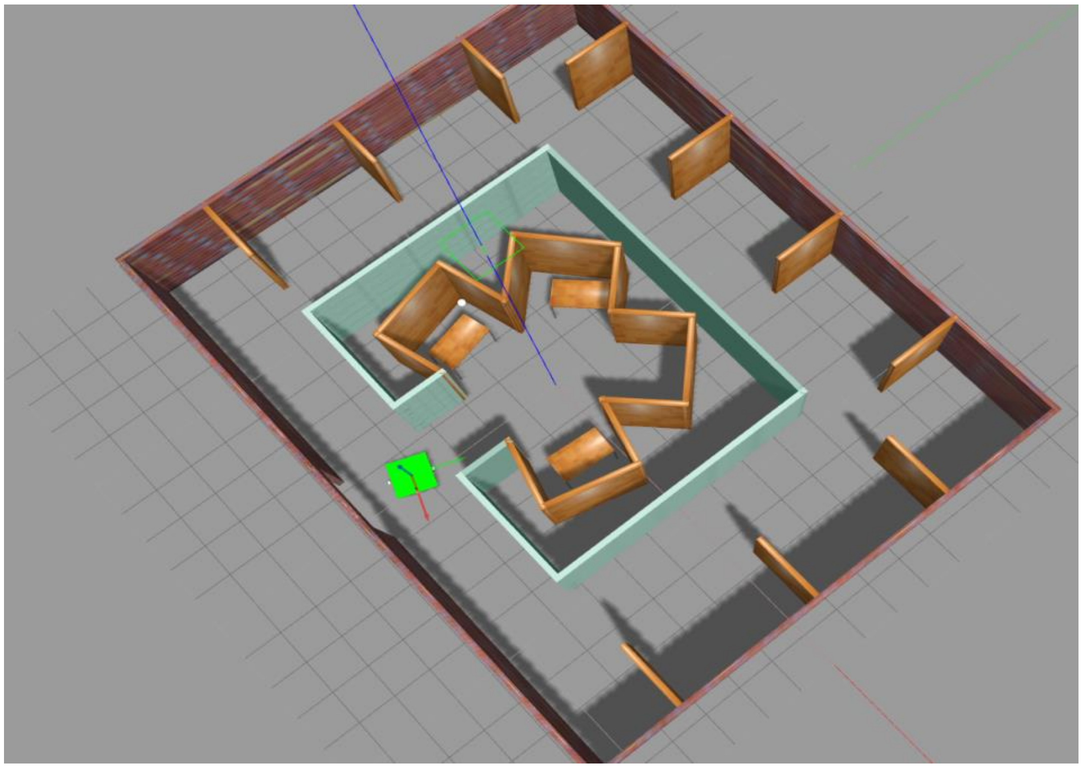


Figure 16.): Design of the initial workspace in Gazebo (green figure is the mobile robot from figure 19.))

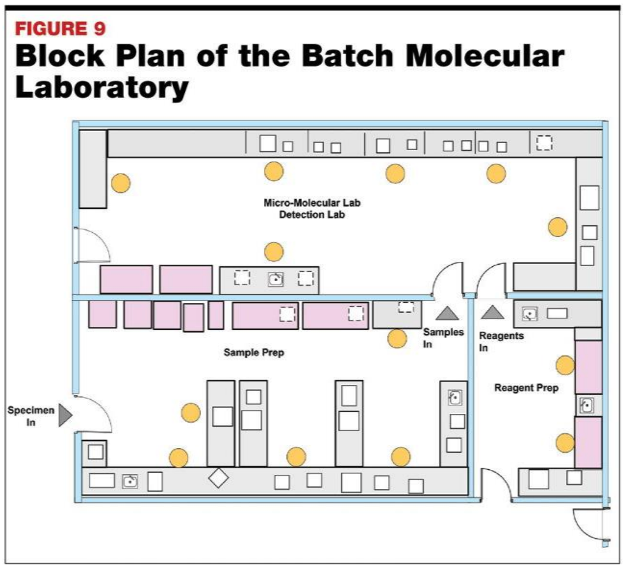


FIGURE 9
Block Plan of the Batch Molecular Laboratory
Micro-Molecular Lab Detection Lab
Sample Prep
Samples In
Reagents In
Reagent Prep
Specimen In

Figure 17.): Another layout from [26]

A few other layouts were studied later and another optimum layout which can heed the needs of various processes that take place in a bio manufacturing/research environment is given above.

This was translated into

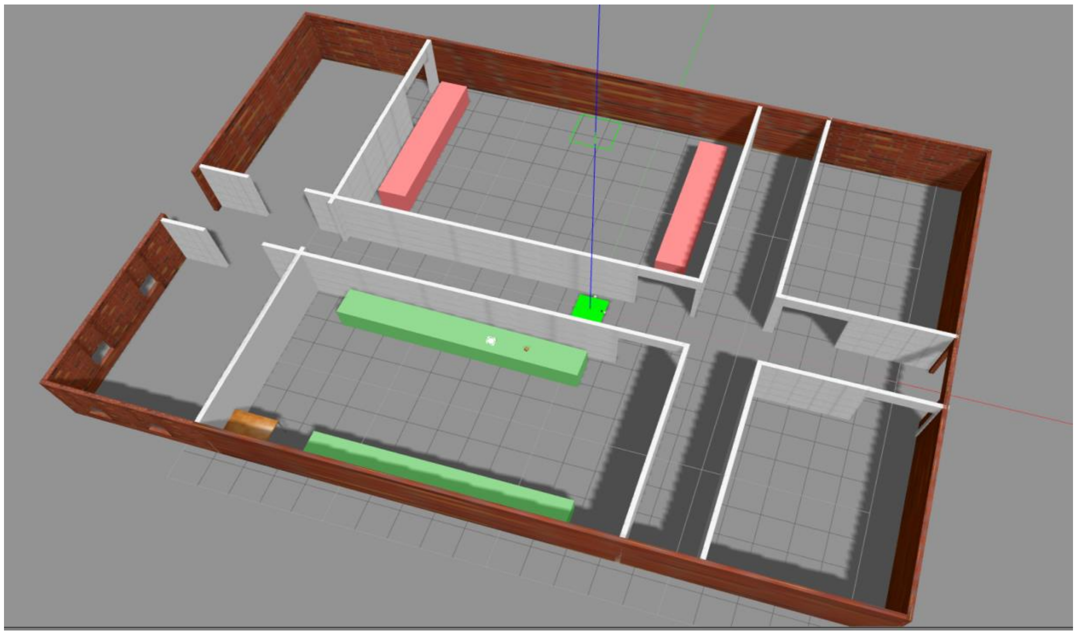


Figure 18): A Gazebo world for the layout in figure 17)

C.) **Design of mobile robot**

A simple 2 wheeled mobile robot with caster spheres for balance was used. This can be translated to a 4-wheel drive later. The robot was equipped with a camera and a lidar. This robot was designed in a .xacro format. The robot has a primary function to hold material on it and transport it to various places in the facility.

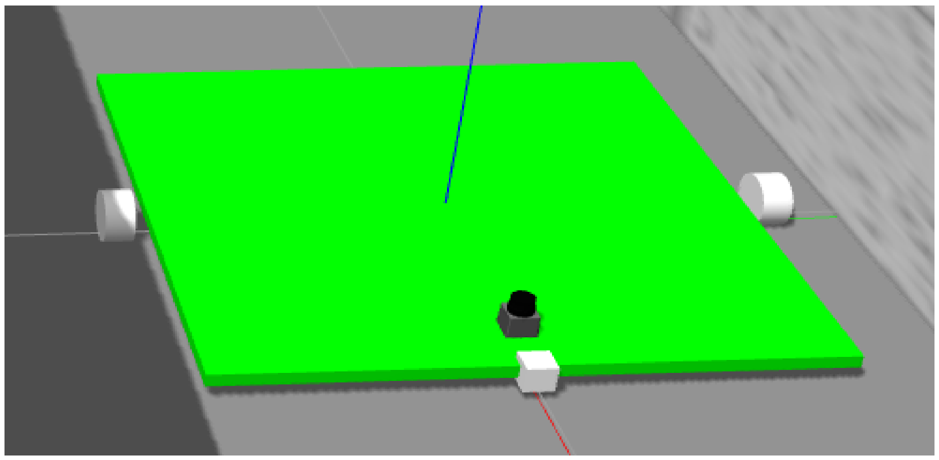
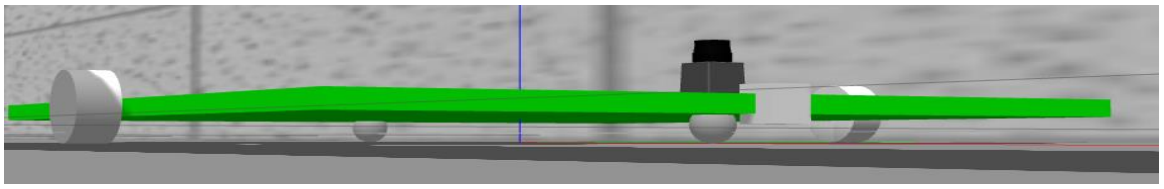



Figure 19.): (top) a 2 wheeled robot with differential drive and attached with lidar(black) and a camera (white cube). (bottom) Caster spheres for balance.

D.) **Design of manipulator arm**

A panda manipulator arm by *Franka Emika* was used to attempt manipulation to move objects like test-tubes. The individual path planning was successful on Rviz, a visual tool by ROS, but its integration with Gazebo and thus with our mobile robot for a grasp-pick-place-deliver task was not done.
Figure: Panda manipulator with 8 links.

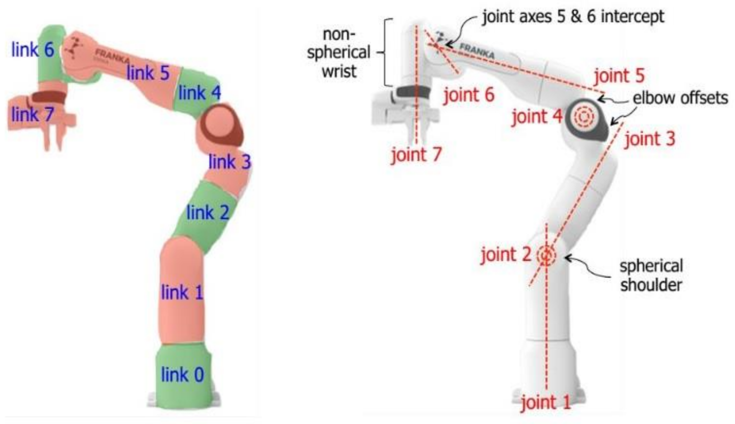


Figure 20.): The links and joints of Panda manipulator.

While manipulation was done using a set of pid controllers and the inverse kinematics was solved using the IKFast [27], the Robot Kinematics Compiler, which is a powerful inverse kinematics solver provided within Rosen Diankov’s OpenRAVE motion planning software. Unlike most inverse kinematics solvers, IKFast can analytically solve the kinematics equations of any complex kinematics chain and generate language-specific files (like C++) for later use. The result is extremely stable solutions that can run as fast as 5 microseconds on recent processors

E.) **Execution of Mobile Robot**

Due to lack of presence of all the sensors on the 2 wheeled differential drive robot designed, a turtlebot was used for the task. It is equipped with 3D cameras and laser sensors. Initially the original mobile bot was utilized to follow a ball using its camera for learning purposes. Later due to constraints of resources and time, the final task was made to be done by turtlebot. The following steps were taken for execution:

1.) Launching the the Gazebo world along with a turtlebot deployed at a known location.

2.) Adaptive Monte Carlo Localization algorithm package was launched which Localized the map while using Occupancy grid mapping algorithm. This combined step is called Grid based FastSLAM. After each control of movement the algorithm receives from the trajectory planning algorithm, a new set of particle filter is created and remapped. This makes this process suitable for a dynamic environment.

3.) For trajectory planning, ROS’ Navigation stack which uses Dijkstra’s shortest path algorithm is used and after each iteration where it provides the SLAM stack with controls to move in its trajectory, the area in vision is remapped and in presence of sudden appearance of an obstacle, a new map is generated which incorporates the changes in previous map. This way a coupled system is created between SLAM and trajectory planning so that it can work in a dynamic environment.

For SLAM the following map was developed for 1st workspace using pgm_map_creator package which would aid in the amcl. This amcl was combined with the algorithm of occupancy grid mapping to give the method of Grid-Based FastSLAM.

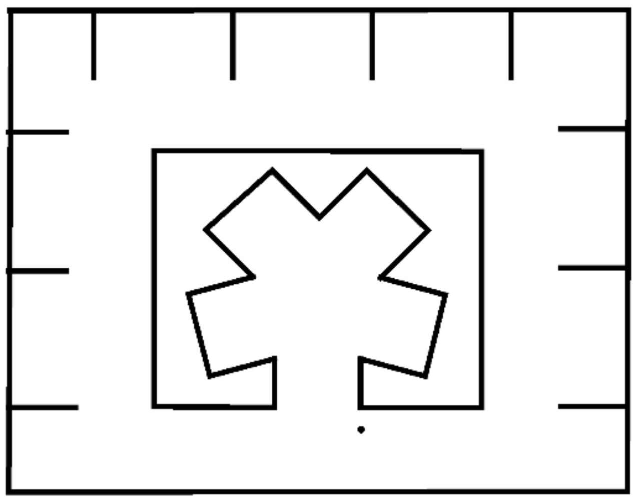


Figure 21.): Map generated of First workspace using pgm package in ROS.

2 nodes by the name of add_markers and pick_objects were written in C++ as separate packages as can be accessed from the github link [28]. Their function was as follows:

1.) add_markers: created a visual representation of parcels to be picked and delivered, denoted as a blue box. The blue box is visible until the bot arrives at the pick coordinates, once arrived waits for 5 seconds and then disappears denoting the parcel has been picked. Once the bot reaches its destination, the blue box lightens up again denoting the delivery of package
2.) Pick_objects: Its simple function is to use Dijkstra's algorithm from the Rviz navigation stack to traverse through using SLAM and a supplied map such as given above in the figure.

The files can be seen in Appendix A. The C++ code was added manually and the rest of the code for SLAM and Trajectory planning was cloned as a ros package.

The script which runs all these packages chronologically is given in Appendix A in the 3rd section.

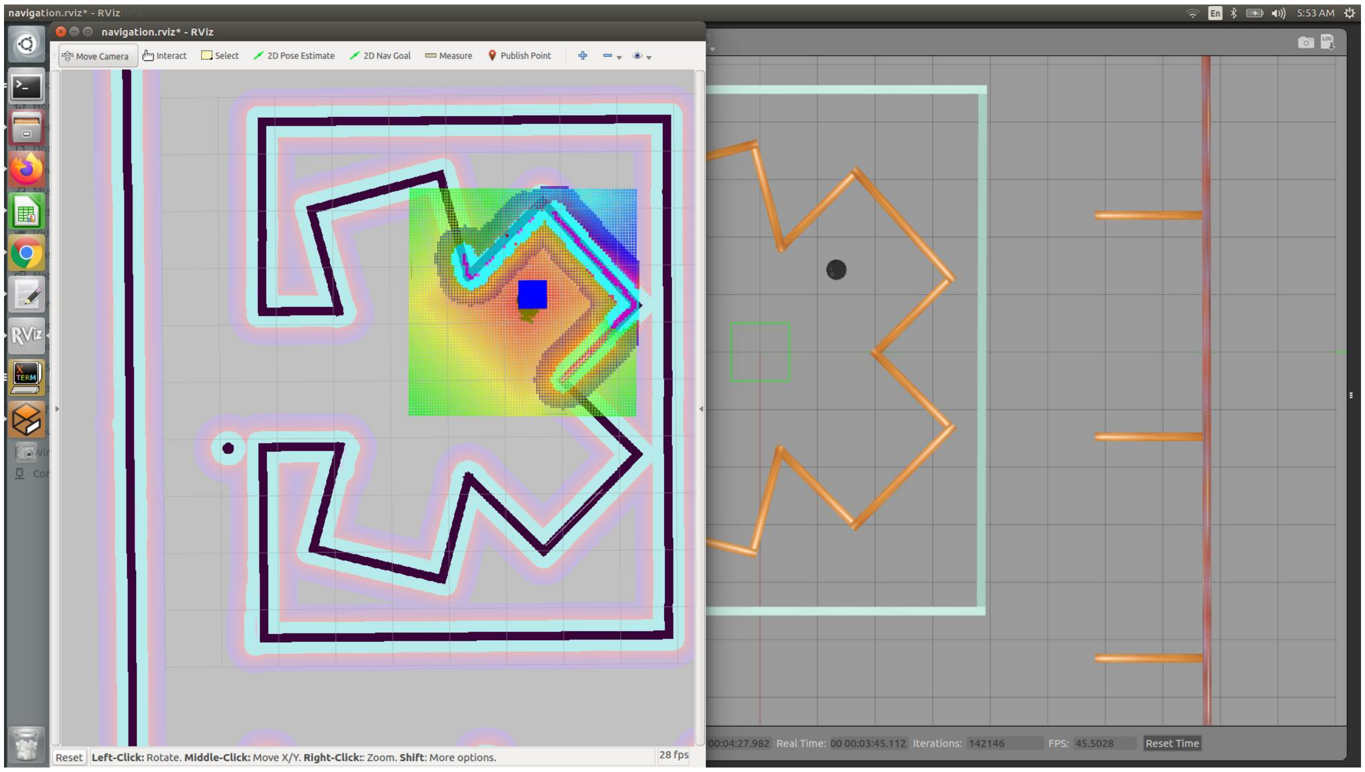


Figure 22.): Left is the Rviz which is a ROS visualization tool. It displays what the robot senses in the environment which is simulated by Gazebo. The blue box is the parcel visualization and the multicoloured square around it is the area where it performs SLAM. Right side is the Gazebo simulation environment.

E.) Execution of Panda Manipulator

The task of the arm is the most extensive. It needs to be covered in depth but a few constraints prevented the in-depth study. The main part of automation is to be carried out by the manipulator as discussed initially, but currently we restrict the study to its grasping abilities to pick and place only. A package by the name “Moveit!” [15].

Is the most sought-after way to develop path planning of a manipulator. The moveit! Package was studied and implemented. This implementation however is restricted to Rviz only and hasn’t been extended to the Gazebo simulator which helps us realize the real life results. In MoveIt, grasping is done using the MoveGroup interface. In order to grasp an object we need to create moveit_msgs::Grasp msg which will allow defining the various poses and postures involved in a grasping operation

3. Simulation Results

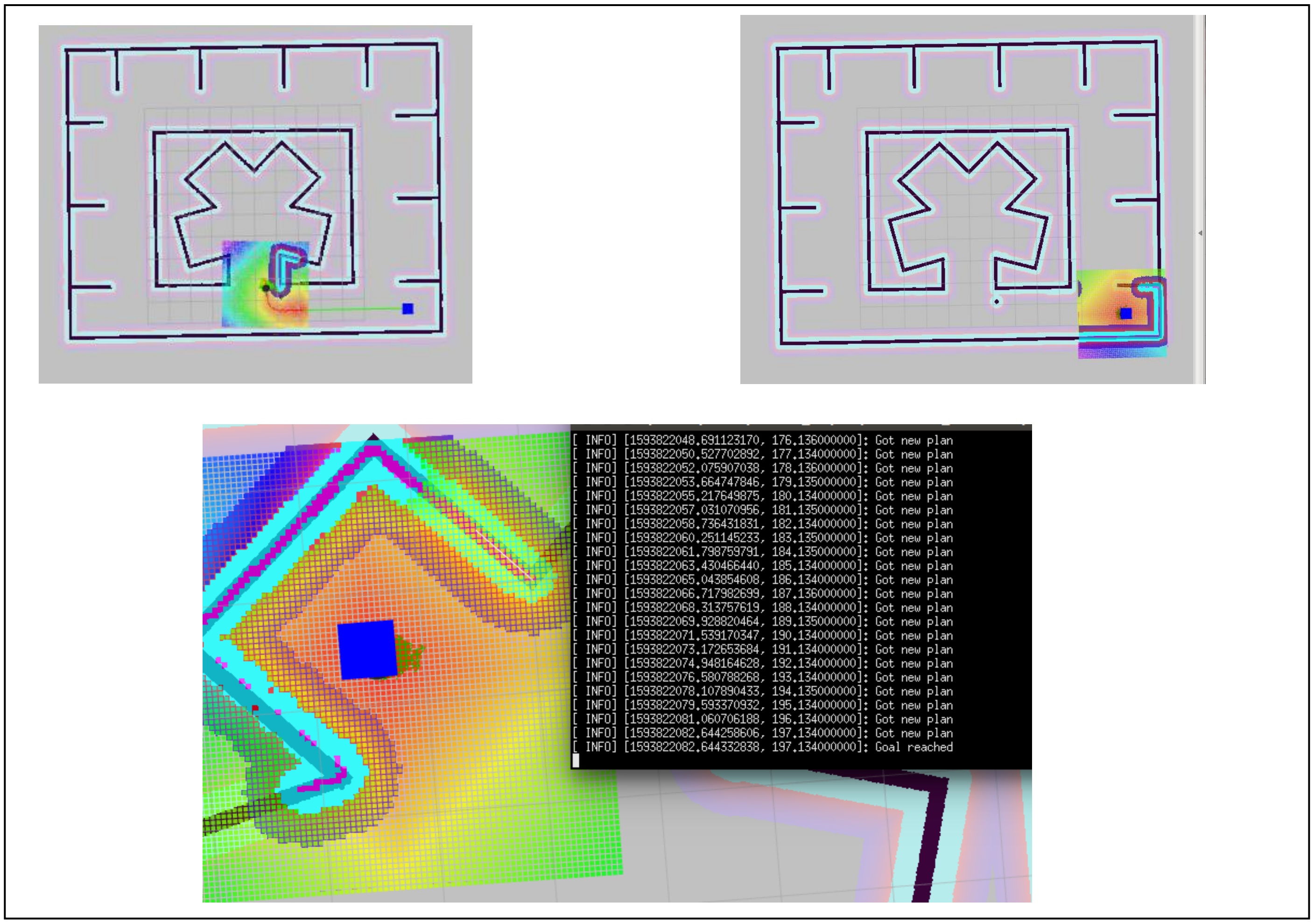
The results involve the working of the mobile robot in the workspace and performing Grid-based FastSLAM as discussed previously to localize itself and map the environment and the perform path planning. A simple comparison of 3 runs is done which can be seen in table 1.) below. The results are divided into

A.) Mobile robot

B.) Manipulator arm.

A simple visual information is provided here.

A. Mobile Robot



```
[ INFO] [1593822048.691123170, 176.136000000]: Got new plan
[ INFO] [1593822050.527702892, 177.134000000]: Got new plan
[ INFO] [1593822052.075907038, 178.136000000]: Got new plan
[ INFO] [1593822053.664747846, 179.135000000]: Got new plan
[ INFO] [1593822055.217649875, 180.134000000]: Got new plan
[ INFO] [1593822057.031070956, 181.135000000]: Got new plan
[ INFO] [1593822058.736431831, 182.134000000]: Got new plan
[ INFO] [1593822060.251145233, 183.135000000]: Got new plan
[ INFO] [1593822061.798759791, 184.135000000]: Got new plan
[ INFO] [1593822063.430466440, 185.134000000]: Got new plan
[ INFO] [1593822065.043854608, 186.134000000]: Got new plan
[ INFO] [1593822066.717982699, 187.136000000]: Got new plan
[ INFO] [1593822068.313757619, 188.134000000]: Got new plan
[ INFO] [1593822069.928820464, 189.135000000]: Got new plan
[ INFO] [1593822071.539170347, 190.134000000]: Got new plan
[ INFO] [1593822073.172653684, 191.134000000]: Got new plan
[ INFO] [1593822074.948164628, 192.134000000]: Got new plan
[ INFO] [1593822076.580788268, 193.134000000]: Got new plan
[ INFO] [1593822078.107890433, 194.135000000]: Got new plan
[ INFO] [1593822079.593370932, 195.134000000]: Got new plan
[ INFO] [1593822081.060706188, 196.134000000]: Got new plan
[ INFO] [1593822082.644258606, 197.134000000]: Got new plan
[ INFO] [1593822082.644332838, 197.134000000]: Goal reached
```

Figure 23.) (top left) Displays the mobile robot (in black) moving towards its goal to pick the virtual blue object to pick. (top right) Displays the robot reached its pick location. (bottom) Displays the bot has reached its drop location.

Run number	Total Distance(m)	Pick time(s)	Deliver time(s)
1	24.5	44	50
2	26.8	48	52
3	25.2	45	53
Average	25.5	47.33	51.67

Table 1.): It was noted after 3 runs from the odometry and simulation readings that the efficiency remained almost the same
This efficiency only arose from Grid-Based FastSLAM and Dijkstra's path planning algorithm based on uniform cost search.

B. Panda Manipulator

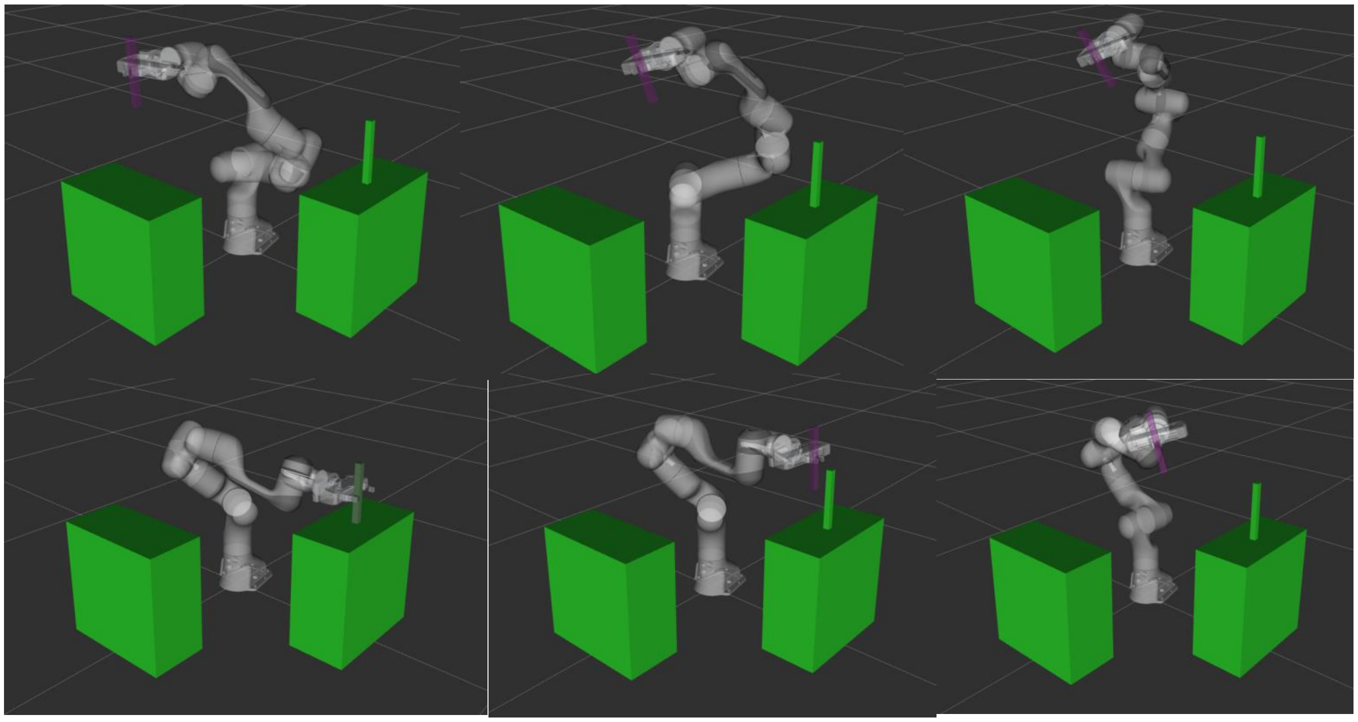


Figure 24): (see clockwise) is a trajectory planning of Panda manipulator via Moveit! package which demonstrates the grasping, picking, moving and then placing the parcel (like a test tube). The purple cylinder denotes it being picked up and the green coloured cylinder denotes that it has been placed.
As shown in figure 24.), the manipulation takes place via a move_group node which is responsible for the forward kinematics of the joints. The controller senses the joint orientation and actuation. Refer Appendix B for reference to the code.

4. Conclusion and Future Work

A lot of procedural work done by employees all across organizations and industries leads to a wastage of skilled manpower. Many debate that that automation will simply lead to increasing unemployment but, it is simply a transfer of employment from low skill to a higher skill labour. With improving education facilities around the world, a higher proportion of people are turning to be higher skilled than their predecessors. Moreover, automation leads to reduction in the cost of commodities.
Thus automation is imminent, and robotics is a huge area of automation which is not yet applied to all the areas where it can bring a change.
The current pandemic situation is a stark reminder that automation in biotechnological fields is lagging behind and needs to be worked upon. The bio-hazardous situations, depreciating supply and increasing endangerment of people from medical or biological professions call for a due automation in these areas. This project focuses on these very problems and intends to solve the issues via automation. There is a lot of further work to do and only the basics have been touched in this project which was mostly study oriented.
The basic algorithms of SLAM and trajectory planning worked fine in the workspace layout I designed but in the presence of a very dynamic environment like that of a lab will make it very difficult to work. Machine learning concepts need to be used to continuously make these bots smarter and tackle what's in their way in an efficient manner.
The Manipulator worked fine in gripping the test tube like cylinder which it later needs to function with. A basic understanding of Moveit! was developed.
A map for the 2nd workspace layout was having problems hence it was not able to be used for SLAM and path planning. The 2nd workspace layout is more practical and needs to be implemented. Overall a basic solid understanding of ROS and Gazebo and their working was developed which fulfilled the purpose of this study project

Future work which has to be done:

- Comparison of more algorithms of SLAM and Trajectory planning to find out the ones best suited for our Laboratory like environment.
- Integration of the Panda arm with Gazebo so that its response in a simulated environment can be judged.
- Integration of the 2 parts i.e. the mobile robot and the manipulator grasping so that a complete environment can be established where the two robots are working together.
- A longer way to go is usage of Data Science to optimize the working of the robots in varied environments
- Develop a prototype of these 2 robots and test them in real environment

5. Acknowledgment

I would like to thank Dr. B. K. Rout (Prof.) for his guidance and the insight he provided into scope the of this project.

6. References

1. S. Thrun et al "Robust Monte Carlo localization for mobile robots" *Artificial Intelligence 128 (2001) 99–141*
2. INTELLIGENT SYSTEMS, CONTROL, AND AUTOMATION: SCIENCE AND ENGINEERING VOLUME 43 Editor Professor S. G. Tzafestas, National Technical University of Athens, Greece
3. A Review of Cell Culture Automation Maria E. Kempner, Associate Editor, JALA Robin A. Felder, Ph.D., Editor, JALA
4. Kowalski M P, Yoder A, Liu L and Pajak L. Controlling embryonic stem cell growth and differentiation by automation: enhanced and more reliable differentiation for drug discovery. J. Biomol. Screen. 17(9); 1171-9: (Oct 2012).
5. Solter, D, Knowles B B. Monoclonal antibody defining a stage-specific mouse embryonic antigen (SSEA-1). Proc Natl Acad Sci USA. 75(11); 5565-9: (1978). ©
6. High-throughput automation design considerations for biotechnology processes involving RNA purification protocols using multi-centrifuge bioseparation steps Aura-Maria Cardona a,n, Zvi Roth a, Chingping Han b.
7. A.-M. Cardonaetal./RoboticsandComputer-IntegratedManufacturing28(2012)285–293
8. Opportunities for Modern Robotic Systems in Biologics Manufacturing. RG Beri.
9. Robotics and Autonomous Systems Volume 60, Issue 10, October 2012, Pages 1340-1353
10. Trends in Biotechnology Volume 31, Issue 5, May 2013, Pages 287-294.
11. Trends in Biotechnology Volume 29, Issue 3, March 2011, Pages 127-135.
12. Udacity course on Robotics Software Engineer.
13. The Construct Youtube channel -https://www.youtube.com/channel/UCt6Lag-vv25fTX3e11mVY1Q
14. Robotics and control by R.K. Mittal and I.J. Nagrath
15. Moveit! Tutorials - http://docs.ros.org/melodic/api/moveit_tutorials/html/index.html
16. W. Qian et al., "Manipulation task simulation using ROS and Gazebo," 2014 IEEE International Conference on Robotics and Biomimetics (ROBIO 2014), Bali, 2014, pp. 2594-2598, doi: 10.1109/ROBIO.2014.7090732.
17. Grandics, P., Szathmary, S., Szathmary, Z., O'Neill, T. Integration of Cell Culture with Continuous, On-Line Sterile Downstream Processing Annals of the New York Academy of Sciences. V. 646, 1991, 322.

18. P. Najmabadi, A. A. Goldenberg and A. Emili, "A scalable robotic-based laboratory automation system for medium-sized biotechnology laboratories," *IEEE International Conference on Automation Science and Engineering, 2005.*, Edmonton, Alta., 2005, pp. 166-171, doi: 10.1109/COASE.2005.1506763.
19. Laboratory Robotics Handbook, Zymark Corporation, Hopkinton, MA, 1988, pp. 24-33.
20. C. Cadena *et al*., "Past, Present, and Future of Simultaneous Localization and Mapping: Toward the Robust-Perception Age," in *IEEE Transactions on Robotics*, vol. 32, no. 6, pp. 1309-1332, Dec. 2016, doi: 10.1109/TRO.2016.2624754.
21. A. Elfes, "Occupancy Grids: A probabilistic framework for robot perception and navigation", *J. Robot. Autom.*, vol. RA-3, no. 3, pp. 249-265, 1987.
22. *Michael Montemerlo and Sebastian Thrun*, "FastSLAM: A Factored Solution to the Simultaneous Localization and Mapping Problem", AAAI-02 Proceedings. Copyright © 2002, AAAI (www.aaai.org).
23. M. Noto and H. Sato, "A method for the shortest path search by extended Dijkstra algorithm," *Smc 2000 conference proceedings. 2000 ieee international conference on systems, man and cybernetics. 'cybernetics evolving to systems, humans, organizations, and their complex interactions' (cat. no.0*, Nashville, TN, 2000, pp. 2316-2320 vol.3, doi: 10.1109/ICSMC.2000.886462.
24. http://wiki.ros.org/Documentation.
25. *Aura-Maria Cardona, Zvi Roth, Chingping Han*, "High-throughput automation design considerations for biotechnology processes involving RNA purification protocols using multi-centrifuge bioseparation steps", Robotics and Computer-Integrated Manufacturing, Volume 28, Issue 3, 2012, Pages 285-293, ISSN 0736-5845.
26. "Bioprocess Facility Design — Layout Rules And Configuration" By *Herman Bozenhardt, Bozenhardt Consulting Services, and Erich Bozenhardt, IPS-Integrated Project Services.*
27. http://openrave.org/docs/0.8.2/openravepy/ikfast/
28. https://github.com/nigamishan/Bio_Facility_Automation.git

Appedix A

1.) add_markers.cpp

```
#include
<ros/ros.h>
              #include <visualization_msgs/Marker.h>
              #include "nav_msgs/Odometry.h"
              #include <tf/tf.h>
              #include <math.h>

              bool picked = false;
              bool dropped = false;
              float threshold = 0.25;
              float ends[2][3] = {{7.0, -4.5, 0.0}, {-1.5, 1.5, 0.0}};

              //recieves odometry values from subscribing

              void callback(const nav_msgs::Odometry::ConstPtr& msg)
              {
                     //current coordinates
                float x = 0;
                float y = 0;
                     //difference between current and the end x&y values
                float pick_x_diff = 0.0;
                float pick_y_diff = 0.0;
                float drop_x_diff = 0.0;
                float drop_y_diff = 0.0;
                     //distance between current and ends
                float pick_diff = 0.0;
                float drop_diff = 0.0;

                x = msg->pose.pose.position.x ;
                y = msg->pose.pose.position.y ;
                pick_x_diff = std::abs(x - ends[0][0]);
                pick_y_diff = std::abs(y - ends[0][1]);
                pick_diff = sqrt(pow(pick_x_diff,2) + pow(pick_y_diff,2));
                drop_x_diff = std::abs(x - ends[1][0]);
                drop_y_diff = std::abs(y - ends[1][1]);
                drop_diff = sqrt(pow(drop_x_diff,2) + pow(drop_y_diff,2));
```

```
  if (pick_x_diff < threshold && pick_y_diff < threshold)
  {
    picked = true;
    ROS_INFO("Object has been picked up!");
  }
  else if(drop_y_diff < threshold && drop_x_diff < threshold && picked)
  {
    dropped = true;
    ROS_INFO("Object is dropped");
  }
  else
  {

    ROS_INFO("curr_x: %f", x);
    ROS_INFO("curr_y: %f", y);
  }
}

int main( int argc, char** argv )
{
  ros::init(argc, argv, "add_markers");
  ros::NodeHandle n;
  ros::Subscriber odom = n.subscribe("/odom", 1000, callback);
  ros::Publisher marker_pub =
n.advertise<visualization_msgs::Marker>("visualization_marker", 1);
  visualization_msgs::Marker marker;

  marker.header.frame_id = "/map";
  marker.header.stamp = ros::Time::now();
  marker.ns = "add_markers";
  marker.id = 0;

  marker.type = visualization_msgs::Marker::CUBE;

  marker.scale.x = 0.5;
  marker.scale.y = 0.5;
  marker.scale.z = 0.5;
  marker.color.r = 0.0f;
  marker.color.g = 0.0f;
  marker.color.b = 1.0f;
  marker.color.a = 1.0;
  marker.lifetime = ros::Duration();
  while(ros::ok())
  {

    if(!picked)
    {
      marker.action = visualization_msgs::Marker::ADD;
      marker.pose.position.x = ends[0][0];
      marker.pose.position.y = ends[0][1];
      marker.pose.position.z = 0.5;
      marker.pose.orientation = tf::createQuaternionMsgFromYaw(ends[0][2]);
      marker_pub.publish(marker);
    }
    else
    {
      ros::Duration(5.0).sleep();
      marker.action = visualization_msgs::Marker::DELETE;
      marker_pub.publish(marker);
    }
    if(dropped)
    {
      ROS_INFO("dropped");
      marker.action = visualization_msgs::Marker::ADD;
      marker.pose.position.x = ends[1][0];
      marker.pose.position.y = ends[1][1];
      marker.pose.position.z = 0.5;
      marker.pose.orientation = tf::createQuaternionMsgFromYaw(ends[1][2]);
      marker_pub.publish(marker);
      ros::Duration(5.0).sleep();
    }
    ros::spinOnce();
  }
  return 0;
}
```

2.) pick_objects.cpp

```
#include
<ros/ros.h>
              #include <tf2/LinearMath/Quaternion.h>
              #include <move_base_msgs/MoveBaseAction.h>
              #include <actionlib/client/simple_action_client.h>

              // Define a client for to send goal requests to the move_base server through a
              SimpleActionClient
              typedef actionlib::SimpleActionClient<move_base_msgs::MoveBaseAction> MoveBaseClient;

              bool send(double x, double y, double theta, MoveBaseClient &ac)
              {

                     move_base_msgs::MoveBaseGoal goal;

                // set up the frame parameters
                goal.target_pose.header.frame_id = "map";
                goal.target_pose.header.stamp = ros::Time::now();

                // Define a position and orientation for the robot to reach
                goal.target_pose.pose.position.x = x;
                goal.target_pose.pose.position.y = y;


               tf2::Quaternion orientation;
                  orientation.setRPY(0, 0, theta);
                  goal.target_pose.pose.orientation.w = orientation.getW();
                  goal.target_pose.pose.orientation.x = orientation.getX();
                  goal.target_pose.pose.orientation.y = orientation.getY();
                  goal.target_pose.pose.orientation.z = orientation.getZ();

                 // Send the goal position and orientation for the robot to reach
                     ROS_INFO("Sending goal (x, y, theta)=(%f, %f, %f).", x, y, theta);
                ac.sendGoal(goal);

                // Wait an infinite time for the results
                ac.waitForResult();
```

```
  // Check if the robot reached its goal
  if(ac.getState() == actionlib::SimpleClientGoalState::SUCCEEDED)
 {
    ROS_INFO("Hooray, the base moved to desired goal");
       return true;
 }
  else
 {
    ROS_INFO("The base failed to move to the goal for some reason");
       return false;
 }
}



int main(int argc, char** argv){
  // Initialize the simple_navigation_goals node
  ros::init(argc, argv, "pick_objects");

  //tell the action client that we want to spin a thread by default
  MoveBaseClient ac("move_base", true);

  // Wait 5 sec for move_base action server to come up
  while(!ac.waitForServer(ros::Duration(5.0))){
    ROS_INFO("Waiting for the move_base action server to come up");
  }
  bool next =send(7.0, -4.5, 0.0, ac);
       ros::Duration(5).sleep();
  if(next)
  {
       send(-1.5, 1.5, 0.0, ac);
  }

  return 0;
}
```

3.) Scripts to run all packages at once

```
#!/bin/sh

export TURTLEBOT_GAZEBO_WORLD_FILE=/home/ishan/catkin_ws/src/map/Ishan.world
export TURTLEBOT_GAZEBO_MAP_FILE=/home/ishan/catkin_ws/src/map/map.yaml

xterm  -e  " roslaunch turtlebot_gazebo turtlebot_world.launch " &
sleep 8
xterm  -e  " roslaunch turtlebot_gazebo amcl_demo.launch" &
sleep 5
xterm  -e  " roslaunch turtlebot_rviz_launchers view_navigation.launch" &
sleep 5
xterm  -e  " rosrun pick_objects pick_objects" &
sleep 5
xterm  -e  " rosrun add_markers add_markers"
```

All these can be accessed from: *https://github.com/nigamishan/Bio_Facility_Automation.git*

Appendix B

Pick and Place cpp (from MoveIt documentation):

```
//
ROS
        #include <ros/ros.h>

        // MoveIt
        #include <moveit/planning_scene_interface/planning_scene_interface.h>
        #include <moveit/move_group_interface/move_group_interface.h>

        // TF2
        #include <tf2_geometry_msgs/tf2_geometry_msgs.h>

        void openGripper(trajectory_msgs::JointTrajectory& posture)
        {
          // BEGIN_SUB_TUTORIAL open_gripper
          /* Add both finger joints of panda robot. */
          posture.joint_names.resize(2);
          posture.joint_names[0] = "panda_finger_joint1";
          posture.joint_names[1] = "panda_finger_joint2";

          /* Set them as open, wide enough for the object to fit. */
          posture.points.resize(1);
          posture.points[0].positions.resize(2);
          posture.points[0].positions[0] = 0.04;
          posture.points[0].positions[1] = 0.04;
          posture.points[0].time_from_start = ros::Duration(0.5);
          // END_SUB_TUTORIAL
        }

        void closedGripper(trajectory_msgs::JointTrajectory& posture)
        {
          // BEGIN_SUB_TUTORIAL closed_gripper
          /* Add both finger joints of panda robot. */
          posture.joint_names.resize(2);
          posture.joint_names[0] = "panda_finger_joint1";
          posture.joint_names[1] = "panda_finger_joint2";

          /* Set them as closed. */
```

```
  posture.points.resize(1);
  posture.points[0].positions.resize(2);
  posture.points[0].positions[0] = 0.00;
  posture.points[0].positions[1] = 0.00;
  posture.points[0].time_from_start = ros::Duration(0.5);
  // END_SUB_TUTORIAL
}

void pick(moveit::planning_interface::MoveGroupInterface& move_group)
{
  // BEGIN_SUB_TUTORIAL pick1
  // Create a vector of grasps to be attempted, currently only creating single grasp.
  // This is essentially useful when using a grasp generator to generate and test multiple
grasps.
  std::vector<moveit_msgs::Grasp> grasps;
  grasps.resize(1);

  // Setting grasp pose
  // ++++++++++++++++++++++
  // This is the pose of panda_link8. |br|
  // From panda_link8 to the palm of the eef the distance is 0.058, the cube starts 0.01 before
5.0 (half of the length
  // of the cube). |br|
  // Therefore, the position for panda_link8 = 5 - (length of cube/2 - distance b/w panda_link8
and palm of eef - some
  // extra padding)
  grasps[0].grasp_pose.header.frame_id = "panda_link0";
  tf2::Quaternion orientation;
  orientation.setRPY(-M_PI / 2, -M_PI / 4, -M_PI / 2);
  grasps[0].grasp_pose.pose.orientation = tf2::toMsg(orientation);
  grasps[0].grasp_pose.pose.position.x = 0.415;
  grasps[0].grasp_pose.pose.position.y = 0;
  grasps[0].grasp_pose.pose.position.z = 0.5;

  // Setting pre-grasp approach
  // ++++++++++++++++++++++++++
  /* Defined with respect to frame_id */
  grasps[0].pre_grasp_approach.direction.header.frame_id = "panda_link0";
  /* Direction is set as positive x axis */
  grasps[0].pre_grasp_approach.direction.vector.x = 1.0;
  grasps[0].pre_grasp_approach.min_distance = 0.095;
  grasps[0].pre_grasp_approach.desired_distance = 0.115;
  // Setting post-grasp retreat
  // ++++++++++++++++++++++++++
  /* Defined with respect to frame_id */
  grasps[0].post_grasp_retreat.direction.header.frame_id = "panda_link0";
  /* Direction is set as positive z axis */
  grasps[0].post_grasp_retreat.direction.vector.z = 1.0;
  grasps[0].post_grasp_retreat.min_distance = 0.1;
  grasps[0].post_grasp_retreat.desired_distance = 0.25;

  // Setting posture of eef before grasp
  // +++++++++++++++++++++++++++++++++++
  openGripper(grasps[0].pre_grasp_posture);
  // END_SUB_TUTORIAL

  // BEGIN_SUB_TUTORIAL pick2
  // Setting posture of eef during grasp
  // +++++++++++++++++++++++++++++++++++
  closedGripper(grasps[0].grasp_posture);
  // END_SUB_TUTORIAL

  // BEGIN_SUB_TUTORIAL pick3
  // Set support surface as table1.
  move_group.setSupportSurfaceName("table1");
  // Call pick to pick up the object using the grasps given
  move_group.pick("object", grasps);
  // END_SUB_TUTORIAL
}

void place(moveit::planning_interface::MoveGroupInterface& group)
{
  // BEGIN_SUB_TUTORIAL place
  // TODO(@ridhwanluthra) - Calling place function may lead to "All supplied place locations
failed. Retrying last
  // location in
  // verbose mode." This is a known issue and we are working on fixing it. |br|
  // Create a vector of placings to be attempted, currently only creating single place
location.
  std::vector<moveit_msgs::PlaceLocation> place_location;
  place_location.resize(1);

  // Setting place location pose
  // +++++++++++++++++++++++++++
  place_location[0].place_pose.header.frame_id = "panda_link0";
  tf2::Quaternion orientation;
  orientation.setRPY(0, 0, M_PI / 2);
  place_location[0].place_pose.pose.orientation = tf2::toMsg(orientation);

  /* While placing it is the exact location of the center of the object. */
  place_location[0].place_pose.pose.position.x = 0;
  place_location[0].place_pose.pose.position.y = 0.5;
  place_location[0].place_pose.pose.position.z = 0.5;

  // Setting pre-place approach
  // ++++++++++++++++++++++++++
  /* Defined with respect to frame_id */
  place_location[0].pre_place_approach.direction.header.frame_id = "panda_link0";
  /* Direction is set as negative z axis */
  place_location[0].pre_place_approach.direction.vector.z = -1.0;
  place_location[0].pre_place_approach.min_distance = 0.095;
  place_location[0].pre_place_approach.desired_distance = 0.115;

  // Setting post-grasp retreat
  // ++++++++++++++++++++++++++
  /* Defined with respect to frame_id */
  place_location[0].post_place_retreat.direction.header.frame_id = "panda_link0";
  /* Direction is set as negative y axis */
  place_location[0].post_place_retreat.direction.vector.y = -1.0;
  place_location[0].post_place_retreat.min_distance = 0.1;
  place_location[0].post_place_retreat.desired_distance = 0.25;

  // Setting posture of eef after placing object
  // +++++++++++++++++++++++++++++++++++++++++++
  /* Similar to the pick case */
  openGripper(place_location[0].post_place_posture);

  // Set support surface as table2.
  group.setSupportSurfaceName("table2");
  // Call place to place the object using the place locations given.
  group.place("object", place_location);
  // END_SUB_TUTORIAL
}

void addCollisionObjects(moveit::planning_interface::PlanningSceneInterface&
planning_scene_interface)
{
  // BEGIN_SUB_TUTORIAL table1
```

```
//
// Creating Environment
// ^^^^^^^^^^^^^^^^^^^^
// Create vector to hold 3 collision objects.
std::vector<moveit_msgs::CollisionObject> collision_objects;
collision_objects.resize(3);

// Add the first table where the cube will originally be kept.
collision_objects[0].id = "table1";
collision_objects[0].header.frame_id = "panda_link0";

/* Define the primitive and its dimensions. */
collision_objects[0].primitives.resize(1);
collision_objects[0].primitives[0].type = collision_objects[0].primitives[0].BOX;
collision_objects[0].primitives[0].dimensions.resize(3);
collision_objects[0].primitives[0].dimensions[0] = 0.2;
collision_objects[0].primitives[0].dimensions[1] = 0.4;
collision_objects[0].primitives[0].dimensions[2] = 0.4;

/* Define the pose of the table. */
collision_objects[0].primitive_poses.resize(1);
collision_objects[0].primitive_poses[0].position.x = 0.5;
collision_objects[0].primitive_poses[0].position.y = 0;
collision_objects[0].primitive_poses[0].position.z = 0.2;
// END_SUB_TUTORIAL

collision_objects[0].operation = collision_objects[0].ADD;

// BEGIN_SUB_TUTORIAL table2
// Add the second table where we will be placing the cube.
collision_objects[1].id = "table2";
collision_objects[1].header.frame_id = "panda_link0";

/* Define the primitive and its dimensions. */
collision_objects[1].primitives.resize(1);
collision_objects[1].primitives[0].type = collision_objects[1].primitives[0].BOX;
collision_objects[1].primitives[0].dimensions.resize(3);
collision_objects[1].primitives[0].dimensions[0] = 0.4;
collision_objects[1].primitives[0].dimensions[1] = 0.2;
collision_objects[1].primitives[0].dimensions[2] = 0.4;

/* Define the pose of the table. */
collision_objects[1].primitive_poses.resize(1);
```

```
  collision_objects[1].primitive_poses[0].position.x = 0;
  collision_objects[1].primitive_poses[0].position.y = 0.5;
  collision_objects[1].primitive_poses[0].position.z = 0.2;
  // END_SUB_TUTORIAL

  collision_objects[1].operation = collision_objects[1].ADD;

  // BEGIN_SUB_TUTORIAL object
  // Define the object that we will be manipulating
  collision_objects[2].header.frame_id = "panda_link0";
  collision_objects[2].id = "object";

  /* Define the primitive and its dimensions. */
  collision_objects[2].primitives.resize(1);
  collision_objects[2].primitives[0].type = collision_objects[1].primitives[0].BOX;
  collision_objects[2].primitives[0].dimensions.resize(3);
  collision_objects[2].primitives[0].dimensions[0] = 0.02;
  collision_objects[2].primitives[0].dimensions[1] = 0.02;
  collision_objects[2].primitives[0].dimensions[2] = 0.2;

  /* Define the pose of the object. */
  collision_objects[2].primitive_poses.resize(1);
  collision_objects[2].primitive_poses[0].position.x = 0.5;
  collision_objects[2].primitive_poses[0].position.y = 0;
  collision_objects[2].primitive_poses[0].position.z = 0.5;
  // END_SUB_TUTORIAL

  collision_objects[2].operation = collision_objects[2].ADD;

  planning_scene_interface.applyCollisionObjects(collision_objects);
}

int main(int argc, char** argv)
{
  ros::init(argc, argv, "panda_arm_pick_place");
  ros::NodeHandle nh;
  ros::AsyncSpinner spinner(1);
  spinner.start();

  ros::WallDuration(1.0).sleep();
  moveit::planning_interface::PlanningSceneInterface planning_scene_interface;
  moveit::planning_interface::MoveGroupInterface group("panda_arm");
  group.setPlanningTime(45.0);
  addCollisionObjects(planning_scene_interface);

  // Wait a bit for ROS things to initialize
  ros::WallDuration(1.0).sleep();

  pick(group);

  ros::WallDuration(1.0).sleep();

  place(group);

  ros::waitForShutdown();
  return 0;
}
```