REPORT TITLED

Simultaneous Localisation and Mapping for an Autonomous Mobile Manipulator (AVITRA)

submitted in fulfillment of the requirements of
B. TECH ELECTRONICS
By

VIRAJ SONAWANE	151060030
ATHARV KHADTARE	151060032
ATHARVA BHAVE	151060043
SHUBHAM PATIL	151060049
RISHABH SHAH	151060052
CHINMAY KHOPDE	151060054
SUYASH JUNNARKAR	151060055

BACHELOR OF TECHNOLOGY
in Electronics

Under the guidance of
Dr. Faruk S. Kazi
Electrical Department
Veermata Jijabai Technological Institute

STATEMENT OF THE CANDIDATE

I wish to state that work embodied in this report titled **Simultaneous Localisation and Mapping for an Autonomous Mobile Manipulator (AVITRA)** forms our groups contribution to the work carried out under the guidance of Dr. Faruk Kazi at Veermata Jijabai Technological Institute. This work has not been submitted for any other Degree or Diploma of any University/Institute. Wherever references have been made to the previous works of others, it has been clearly indicated.

Signature of Candidate
(VIRAJ SONAWANE)

Signature of Candidate
(ATHARV KHADTARE)

Signature of Candidate
(ATHARVA BHAVE)

Signature of Candidate
(SHUBHAM PATIL)

Signature of Candidate
(RISHABH SHAH)

Signature of Candidate
(CHINMAY KHOPDE)

Signature of Candidate
(SUYASH JUNNARKAR)

ACKNOWLEDGEMENT

The key aspects behind any successful endeavour is an astute sense of vision, unconditional perseverance and a strong sense of purpose. Right from the beginning, we intended to make this project a significant landmark of our lives. We shared a passion for robotics, and a strong desire to exploit it in a socio-economic effort. Here, we express our heartfelt gratitude to the people who directed this aspiration to its rightful destination.

We start by thanking our guide, Dr. Faruk Kazi. He has been a guiding light throughout the project, enriching us with his vast experience. He was always there to address our doubts and clear all the roadblocks that we faced. He provided us with all the resources at his disposal, thereby playing a key role in taking this project to a different strata altogether.

We would also like to thank the head of our department Dr. N.M. Singh and the entire Electrical department staff for being supportive of our aspiration and facilitating its completion.

We are eternally indebted to Society of Robotics and Automation, VJTI and CoE CNDS, VJTI for being such a wonderful platform for us to bridge the gap between theoretical knowledge and practical experience.

We reserve special thanks and deepest gratitude to our family and friends for their support and encouragement. They have generously put up with our fanatic attitude towards this project. We would like to thank every person who might have helped in making this. If, in the course of acknowledging the contributors to the project, we have forgotten any names, generously pardon us.

ABSTRACT

India and many other countries have faced situations where expedited human intervention could be deleterious for their health. Many workers who handle radioactive isotopes have to battle with the subsequent maladies. Despite tremendous progress in robotics and its allied fields over the decades, we have not been able to rely on robots without human intervention. We thereby propose an economic and efficient robotic solution implemented using the Robot Operating System(ROS) that combines the robustness of robots and intelligent sensing systems. The robot along with an onboard computational node will be capable of performing tasks such as pick and place and controlling manual valves. The bot will have full autonomous control over the factory. The mechanism enables the robot to navigate unknown terrains and interact with objects and obstacles of varied features. The main aim of the project is to build on the AVITRA humanoid by adding autonomous navigation and more degrees of freedom to the manipulator. This will allow the robot to work independently and effectively in new territories. Conducting research in this direction is justified by the calamatic events like the Fukushima Nuclear meltdown.

CONTENTS

LIST OF FIGURES **6**

1. About the project **9**

1.1 Introduction 9

1.2 Motivation 10

1.3 Literature Survey 11

1.4 Proposed Solution 12

1.5 Organisation of Thesis 12

2. Mathematical Modelling and Trajectory Planning **13**

2.1 Robot Operating System 13

2.2 Mathematical Modeling And Trajectory Planning 15

2.2.1 Intro to it's mathematical description 15

2.2.2 Kinematics of Manipulator 17

2.2.3 Trajectory Planning 20

3. Visual Perception and Trajectory Planning **28**

3.1 World Monitoring 28

3.1.1 Introduction 28

3.1.2 3D Perception 28

3.1.3 Collision Detection 31

3.1.4 Choosing a detector 32

3.2 Object Localization and Detection 33

3.2.1 Introduction 33

3.2.2 Multibox Single Shot Detector (SSD) 36

3.2.3 Depth Using Kinect 42

4. Mechanism **43**

4.1 Gripper Mechanism 43

4.2 Robot Locomotion Design 46

4.2.1 Types of mobile robot drives 46

5. Simultaneous Localisation and Mapping **51**

5.1 Autonomous Navigation 51

5.1.1 Introduction 51

5.2 Simultaneous Localization and Mapping 54

5.2.1 Methods to perform SLAM 54

5.2.2 Motion Planning 55

5.2.3 Achieving these Steps using ROS 57
5.2.4 Setting up the parameters for our Robot 61
5.2.5 Mapping In ROS 63
5.2.6 Localization In ROS 64
5.2.7 Autonomous Navigation in ROS 66

6. Future Scope and Conclusion **67**
6.1 Test Scenario 67
6.2 Probable Use-cases 67
6.3 Future Scope 68
6.3.1 Dexterous Gripper 68
6.3.2 Artificial Intelligence driven autonomous navigation 68
6.4 Conclusion 69

REFERENCES **70**

LIST OF FIGURES

- Chapter 2

Fig 2.1	14
Fig 2.2	16
Fig 2.3	20
Fig 2.4	22
Fig 2.5	24
Fig 2.6	26

- Chapter 3

Fig 3.1	28
Fig 3.2	30
Fig 3.3	30
Fig 3.4	31
Fig 3.5	33
Fig 3.6	34
Fig 3.7	34
Fig 3.8	35
Fig 3.9	36
Fig 3.10	36
Fig 3.11	37
Fig 3.12	38
Fig 3.13	39
Fig 3.14	40
Fig 3.15	40
Fig 3.16	42

- Chapter 4

Fig 4.1	43
Fig 4.2	45
Fig 4.3	46
Fig 4.4	47
Fig 4.5	48
Fig 4.6	50

- Chapter 5

Fig 5.1	53
Fig 5.2	57
Fig 5.3	58
Fig 5.4	59
Fig 5.5	59
Fig 5.6	60
Fig 5.7	61
Fig 5.8	61
Fig 5.9	62
Fig 5.10	62
Fig 5.11	63
Fig 5.12	64
Fig 5.13	65
Fig 5.14	65
Fig 5.15	66

- Chapter 6

Fig 6.1	68

1. About the project

1.1 Introduction

The ultimate goal of Autonomous Mobile Manipulator is the execution of complex manipulation tasks in unstructured and dynamic environments, in which cooperation with humans may be required. The goal of this project is to develop a mobile manipulator that can traverse a given mapped area and then perform manipulation on a given object using a 5 DOF manipulator.

Mobile manipulation systems must perform a variety of tasks, acquire new skills, and apply these skills towards the tasks. They must be able to continually adapt and improve their performance. The requirement to autonomously locomote uncertain terrain and address the random nature in task execution make it impractical to generalize the entire environment for the task. As a result, mobile manipulation systems have to address problems that arise due to the uncertainty of sensing, actuation and environment. Thus, requiring Simultaneous Localization and Mapping(SLAM) of the environment to locomote and successfully complete the tasks.

Every planning problem in robotics involves constraints. Whether the robot must avoid a collision or joint limits, there are always states that are not permissible. Some constraints are straightforward to satisfy while others can be so stringent that feasible states are very difficult to find. What makes planning with constraints challenging is that, for many constraints, it is impossible or impractical to provide the planning algorithm with the allowed states explicitly; it must discover these states as it plans. Mobile manipulation systems require the integration of a large number of hardware components for sensing, manipulation, and locomotion as well as the orchestration of algorithmic capabilities in perception, manipulation, learning, control, planning, etc.

1.2 Motivation

Exposure to ionizing radiation can induce the death of cells on a scale that can be extensive enough to impair the function of the exposed tissue or organ. At whole-body doses approaching 1 Gray (Gy) and above, acute health effects such as acute radiation syndrome may develop. The effect is more severe for a higher dose. Exposure to moderate levels can result in radiation sickness, which produces a range of symptoms. Nausea and vomiting often begin within hours of exposure, followed by diarrhoea, headaches and fever. This led us to explore many options which can be useful in such scenarios. Therefore we decided to start with a simple design which gave us the confidence that a robotic mobile manipulator can be a feasible solution in such situations.

The manipulator is a device used to manipulate materials without direct human contact with the material. We feel mobile manipulators can be used in such scenarios to protect the workers from these nuclear hazards. A manipulator is an arm-like mechanism that consists of a series of segments. Manipulators consist of nearly rigid links, which are connected by joints that allow relative motion of neighbouring links. These joints are usually instrumented with position sensors, which allow the relative position of neighbouring links to be measured. As this manipulator can give you a relative position with respect to the bot hence mounting it on the autonomous platform gives us an advantage for performing different tasks based on the tool attached.

We then went on to set an aim to develop an autonomous bot. The mobile manipulator will enter a room without any prior knowledge about its environment or its final task. The offboard visual computation mode looks for possible targets that can be manipulated such as a misplaced object, rotating valves, handling objects etc. Then using data from the onboard sensors and the off-board visual computational unit, the robot will navigate autonomously to the target of interest and perform the relevant task. The task to be performed and the target of interest will be selected by an operator sitting at a safe location. The manipulator will have the dexterity that could rival that of a human being. To achieve this the robot will have a gripping mechanism that can be used to negotiate a variety of tasks and work with undefined shapes. Just like a child can pick a ball and a hammer with the same pair of hands, the robot would need one gripper for all tasks.

1.3 Literature Survey

The current trends in robotics can be discerned into two categories: robots designed to solve a particular problem viz surveillance, transportation, prosthetics. A humanoid exoskeleton or a fire fighting robot can be put in this category. The second category is quite interesting; it involves robots that are build to study and emulate a natural phenomenon; which may or may not serve the society immediately. Robotic butterflies, humanoids and popular robots like Boston Dynamics Atlas & Cheetah can be put in this category.

With the decrease in cost of hardware components, more and more time and intellectual resources are being invested in developing intelligent systems for the service of the society. Mobile manipulator finds itself at the epicenter of this development. Mobile manipulators are increasingly being used in warehouses and factory floors for manipulation and transportation of components and loads within the factory floor. These tasks, once viewed as logistic nightmares, are now being carried out efficiently. Initial research in mobile manipulators was mainly related to the mechanical designing aspects to increase the robustness and utility of these mobile manipulators. Both mobility and manipulation were the two tasks being researched independently, and the combination of these generally resulted in huge systems. Applications like rescue operations and indoor pick and place generally require small systems. Thus further research was done to miniaturise these mobile manipulator systems.

As we probe through the available literature, we observe that teleoperated robots formed a major portion of the devices used in industrial infrastructure. Even with great advances in artificial intelligence, the need for supervision and controlled actions has always been felt. This is mainly due to the vast number of factors that need to be considered before any decision is made. This decision making is reserved for humans. Outdoor applications of robots usually involve a combination of robot AI (narrow spectrum intelligence) with human supervision (broad spectrum intelligence) where tasks like navigation and localization are handled by the robot and critical tasks are performed by the human operator. With advances in machine learning, robots are being trained to deal with a particular situation/stimulus. The training phase requires the robot to be programmed with a response to a specific stimulus. In case of humanoid robots, the response requires certain arm actions. These arm actions can be performed by joystick control or autonomously. Disaster management is one area where robots are perceived with great optimism. Disaster management can be seen as sequence of 4 events: prediction, mitigation, rescue and rehabilitation. Disaster mitigation and rescue operations are more appealing challenges as it involves negotiating inaccessible terrain in unforgiving conditions; making unmanned vehicles and robots a plausible and possible solution. Robots like the Quince, Packbot and warrior are used for landmine detection and disposal, and rescue operations. The design of these robots mainly include a robotic manipulator fitted on an all-terrain base. Various sensors are placed strategically placed on the robot for accurate measurements.

1.4 Proposed Solution

We propose to build a general purpose multi-application robotic platform to perform designated tasks, with the tasks being autonomous. To easily tackle the difficulties of a human optimised environment, we have designed an autonomous solution capable of working in unknown locations with articulated arms. As the robot navigates through unfamiliar terrain, a 2D map is generated using Simultaneous Localization and Mapping (SLAM) algorithm based on feedback from rotary encoders, a depth camera and a 2D LIDAR. The bot has an omni-directional platform to enable swift traversal through urban structures like a building. The bot has an underactuated gripper which can adapt to any shape of the object. The underactuated gripper simplifies the control system of the robot.

1.5 Organisation of Thesis

Chapter one introduces the background and motivation of the project idea and explains in brief the implementation of the same. The literature review for the same has been summarized in this section. The mathematical modelling of the bot is explained in chapter two. Also introduction to Robot Operating System (ROS) has been mentioned here. Integration of various hardware subsystems with ROS have been mentioned in the subsequent chapters. The advantages of ROS as an industrial standard has been justified. Chapter four explains the various features incorporated in the robot. The trade-offs between various mechanisms and drives have been studied along with the justification for the chosen ones. Also the gripper design has been demonstrated in two phases of development here. In chapter five we have described in detail how the concepts of SLAM, have been studied and applied in this project. The detailed description of the test bed setup to demonstrate the functionality of the robot is given in chapter six. We have also discussed the various possibilities that can be achieved with this platform.

2. Mathematical Modelling and Trajectory Planning

2.1 Robot Operating System

ROS is an open-source, meta-operating system for your robot. It provides the services you would expect from an operating system, including hardware abstraction, low-level device control, implementation of commonly-used functionality, message-passing between processes, and package management. It also provides tools and libraries for obtaining, building, writing, and running code across multiple computers. ROS is similar in some respects to 'robot frameworks,' such as Player, YARP, Orocos, CARMEN, Orca, MOOS, and Microsoft Robotics Studio.

The computation in ROS is done using a network of process called ROS nodes. This computation network can be called the computation graph. The main concepts in the computation graph are ROS Nodes, Master, Parameter server, Messages, Topics, Services, and Bags. Each concept in the graph is contributed to this graph in different ways.

The ROS communication related packages including core client libraries such as roscpp and rospython and the implementation of concepts such as topics, nodes, parameters, and services are included in a stack called ros_comm. This stack also consists of tools such as rostopic, rosparam, rosservice, and rosnode to introspect the preceding concepts.

The ros_comm stack contains the ROS communication middleware packages and these packages are collectively called ROS Graph layer.

ROS areas include:

- A master coordination node
- Publishing or subscribing to data streams: images, stereo, laser, control, actuator, contact ...
- Multiplexing information
- Node creation and destruction
- Nodes are seamlessly distributed, allowing distributed operation over multi-core, multi-processor, GPUs, and clusters
- Logging
- Parameter server
- Test systems

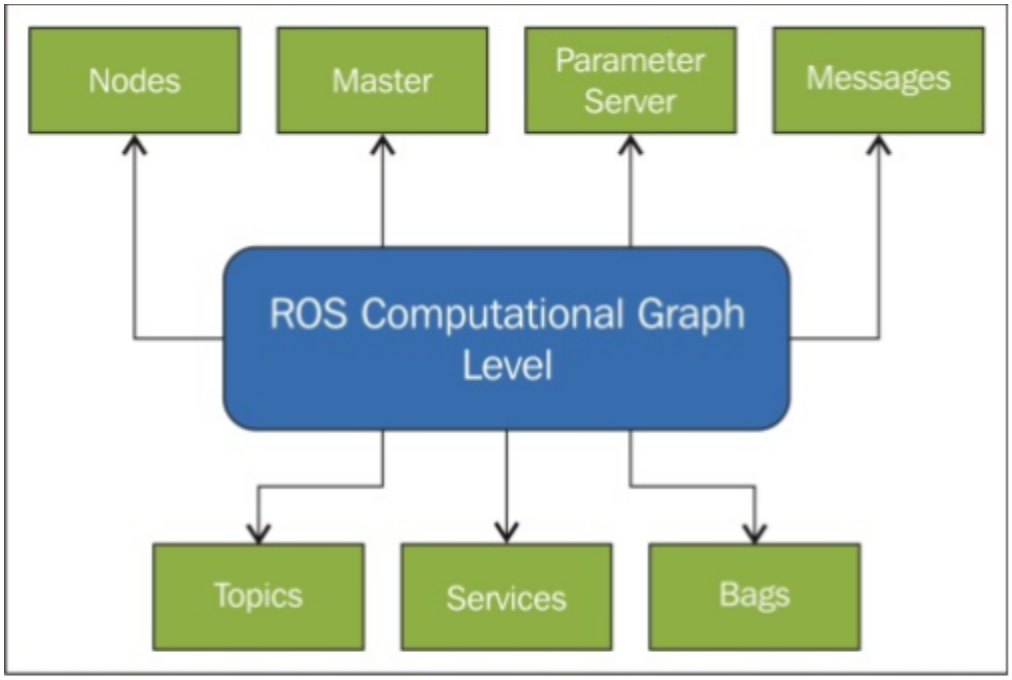

Nodes
Master
Parameter Server
Messages
ROS Computational Graph Level
Topics
Services
Bags

Fig 2.1

2.2 Mathematical Modeling And Trajectory Planning

2.2.1 Intro to it's mathematical description

The parameters

A manipulator may be thought of as a set of bodies connected in a chain by joints. These bodies are called links. Joints form a connection between a neighboring pair of links. For the purposes of obtaining the kinematic equations of the mechanism, a link is considered only as a rigid body that defines the relationship between two neighboring joint axes of a manipulator. Joint axes are defined by lines in space. Joint axis i is defined by a line in space, or a vector direction, about which link i rotates relative to link i 1. It turns out that, for kinematic purposes, a link can be specified with two numbers, which define the relative location of the two axes in space -

1. Link length- For any two axes in 3-space, there exists a well-defined measure of distance between them. The link length is the distance measured along a line that is mutually perpendicular to both axes. This mutual perpendicular always exists; it is unique except when both axes are parallel, in which case there are many mutual perpendiculars of equal
2. Link twist- If we imagine a plane whose normal is the mutually perpendicular line just constructed, we can project the axes i — 1 and i onto this plane and measure the angle between them. This angle is measured from axis i — 1 to axis i in the right-hand sense about α_{i-1}.

Neighboring links have a common joint axis between them. One parameter of interconnection has to do with the distance along this common axis from one link to the next. This parameter is called the link offset. The offset at joint axis i is called The second parameter describes the amount of rotation about this common axis between one link and its neighbor. This is called the joint angle.

Hence, any robot can be described kinematically by giving the values of four quantities for each link. Two describe the link itself, and two describe the link's connection to a neighboring link. In the usual case of a revolute joint, θ_i is called the joint variable, and the other three quantities would be fixed link parameters.

Convention for affixing frames to links

In order to describe the location of each link relative to its neighbors, we define a frame attached to each link. The link frames are named by number according to the link to which they are attached. That is, frame {i } is attached rigidly to link i.

The convention we will use to locate frames on the links is as follows: The 2-axis of frame {i}, called $\hat{Z}_i$, is coincident with the joint axis i. The origin of frame {i} is located where the a_i perpendicular intersects the joint i axis. X_i points along a_i in the direction from joint i to joint i + 1.

In the case of $a_i = 0$, X_i is normal to the plane of Z_i and Z_{i+1} .We define α_i as being measured in the right-hand sense about X_i and so we see that the freedom of choosing the sign of α_i in this case

corresponds to two choices for the direction of X_i . Y_i is formed by the right-hand rule to complete the i^{th} frame.

Summary of the link parameters in terms of the link frames

If the link frames have been attached to the links according to our convention, the following definitions of the link parameters are valid:

1. a_i = the distance from Z_i to Z_{i+1} measured along X_i
2. α_i = the angle from Z_i to Z_{i+1} measured about X_i
3. d_i = the distance from X_{i-1} to X_i measured along Z_i
4. θ_i = the angle from X_{i-1} to X_i measured about Z_i

We usually choose a1 > 0, because it corresponds to a distance; however all other quantities are signed quantities.

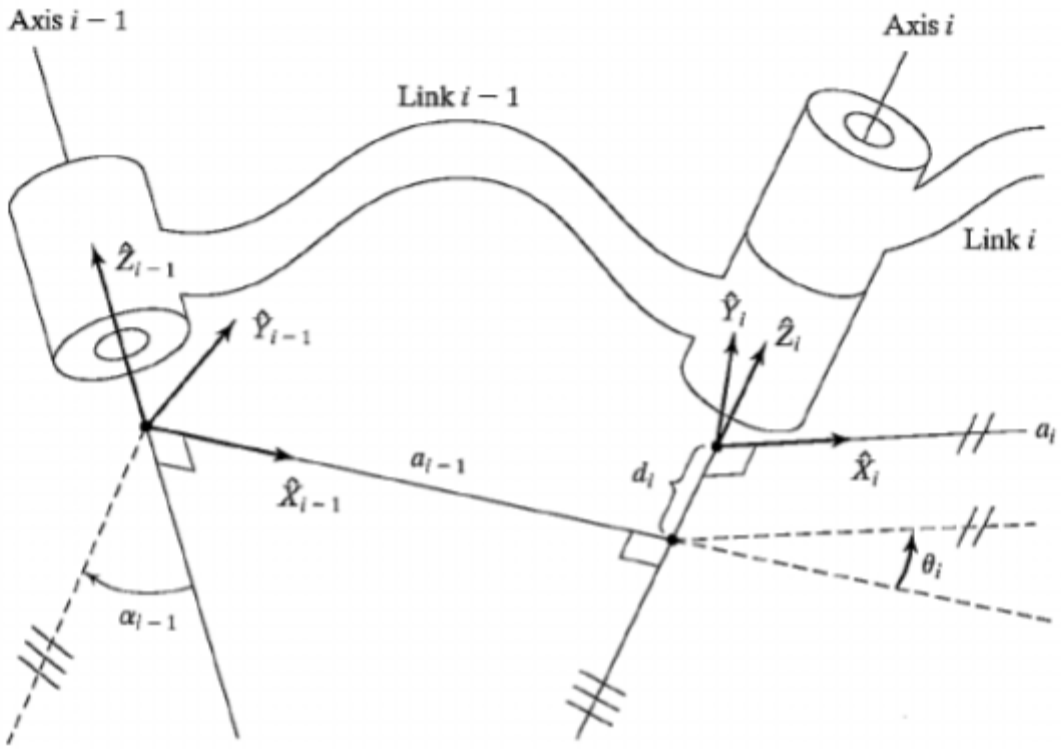


Fig 2.2

Link-frame attachment procedure

The following is a summary of the procedure to follow when faced with a new mechanism, in order to properly attach the link frames:

1. Identify the joint axes and imagine (or draw) infinite lines along them. steps 2 through 5 below, consider two of these neighboring lines (at axes i and i + 1).
2. Identify the common perpendicular between them, or point of intersection. At the point of intersection, or at the point where the common perpendicular meets the i^{th} axis, assign the link-frame origin.
3. Assign the Z_i axis pointing along the i^{th} joint axis.
4. Assign the X_i axis pointing along the common perpendicular, or, if the axes intersect, assign X_i to be normal to the plane containing the two axes.
5. Assign the Y_i axis to complete a right-hand coordinate system.
6. Assign {0} to match {1} when the first joint variable is zero. For {N}, choose an origin location and X_N direction freely, but generally so as to cause as many linkage parameters as possible to become zero.

2.2.2 Kinematics of Manipulator

In the study of robotics, we are constantly concerned with the location of objects in three-dimensional space. These objects are the links of the manipulator, the parts and tools with which it deals, and other objects in the manipulator's environment. At a crude but important level, these objects are described by just two attributes: position and orientation.

Kinematics is the science of motion that treats motion without regard to the forces which cause it. Within the science of kinematics, one studies position, velocity, acceleration, and all higher order derivatives of the position variables (with respect to time or any other variable(s)). Hence, the study of the kinematics of manipulators refers to all the geometrical and time-based properties of the motion. Manipulators consist of nearly rigid links, which are connected by joints that allow relative motion of neighboring links. These joints are usually instrumented with position sensors, which allow the relative position of neighboring links to be measured. In the case of rotary or revolute joints, these displacements are called joint angles.

The number of degrees of freedom that a manipulator possesses is the number of independent position variables that would have to be specified in order to locate all parts of the manipulator. At the free end of the chain of links that make up the manipulator is the end-effector.

Forward kinematics

A very basic problem in the study of mechanical manipulation is called forward kinematics. This is the static geometrical problem of computing the position and orientation of the end-effector of the manipulator. Specifically, given a set of joint angles, the forward kinematic problem is to compute the position and orientation of the tool frame relative to the base frame.

The following is the general form of the transformation that relates the frames attached to neighboring links. We then concatenate these individual transformations to solve for the position and orientation of link n relative to link 0.

$$ {}^{i-1}_{i}T = \begin{bmatrix} c\theta_i & -s\theta_i & 0 & a_{i-1} \\ s\theta_i c\alpha_{i-1} & c\theta_i c\alpha_{i-1} & -s\alpha_{i-1} & -s\alpha_{i-1}d_i \\ s\theta_i s\alpha_{i-1} & c\theta_i s\alpha_{i-1} & c\alpha_{i-1} & c\alpha_{i-1}d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

Once the link frames have been defined and the corresponding link parameters found, developing the kinematic equations is straightforward. From the values of the link parameters, the individual link-transformation matrices can be computed. Then, the link transformations can be multiplied together to find the single transformation that relates frame [N} to frame {0}:

$$ {}^{0}_{N}T = {}^{0}_{1}T\ {}^{1}_{2}T\ {}^{2}_{3}T \ldots {}^{N-1}_{N}T $$

This transformation, T, will be a function of all n joint variables. If the robot's. joint-position sensors are queried, the Cartesian position and orientation of the last. link can be computed by ${}^{0}_{n}T$.

Inverse Kinematics

This section is concerned with the inverse problem of finding the joint variables in terms of the end-effector position and orientation. This is the problem of inverse kinematics, and it is, in general, more difficult than the forward kinematics problem. The joint angles are calculated using the Denavit-Hartenberg convention (D-H parameters) for a n DOF manipulator. Iterative and Analytical methods were used for calculating the joint angles, but the latter was finalized as it resulted in faster and accurate results. The method of solving for joint angles (Inverse Kinematics) yielded multiple solutions for the joint angles as the end effector of the manipulator had the same position for different poses of the manipulator (singularities).

IKFast: The Robot Kinematics Compiler

To reduce the arm matrix of the 5DOF manipulator manually was a herculean task. So to generate the analytical inverse kinematics solutions for the manipulator, the ikfast open source plugin was used.

IKFast analytically solves robot inverse kinematics equations and generates optimized C++ files. The inverse kinematics equations arise from attempting to place the robot end effector coordinate system in the world while maintaining joint and user-specified constraints. User-specified constraints make up many different IK Types, each of them having advantages depending on the task. It is not trivial to create hand-optimized inverse kinematics solutions for arms that can capture all degenerate cases, having closed-form IK speeds up many tasks including planning algorithms, so it really is a must for most robotics researchers.

Closed-form solutions are necessary for motion planning due to two reasons:

- Numerical inverse kinematics solvers will always be much slower than closed form solutions. Planners require being able to process thousands of configurations per second. The closed-form code generated by ikfast can produce solutions on the order of ~4 microseconds! As a comparison, most numerical solutions are on the order of 10 milliseconds (assuming good convergence).
- The null space of the solution set can be explored because all solutions are computed.

2.2.3 Trajectory Planning

Trajectory planning and manipulator dynamics were used to provide a control over a trajectory that includes a number of via points to the end point and velocity control to all the via points using joint

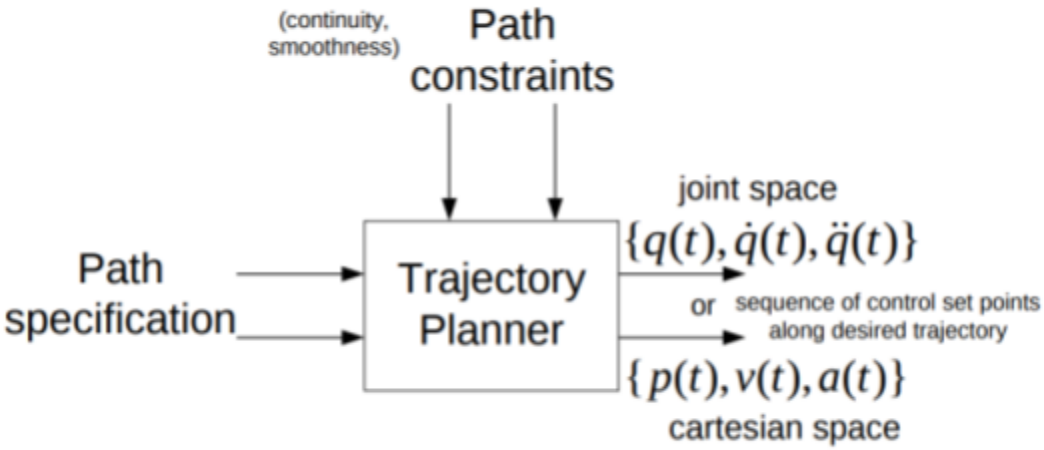


Fig 2.3

space mapping scheme.

In making a single smooth motion, at least four constraints on θ(t) are evident.Hence considering a cubic polynomial relation between theta (θ) and time (t) with four constants a_o , a_1, a_2 and a_3.

$$\theta(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3$$

On differentiating and double differentiating the above equation wrt time we get the equations for velocity and acceleration in the joint space.

$$\theta'(t) = a_1 + 2a_2 t + 3a_3 t^2$$

$$\theta''(t) = 2a_2 + 6a_3 t$$

Considering the four constraints for calculating the values four constraints in the above equations.

$$\theta(0) = \theta_0 ,$$

$$\theta(t_f) = \theta_f ,$$

$$\theta'(0) = 0 ,$$

$$\theta''(0) = 0 ,$$

Applying the above position and velocity constraints,we get.

$$\theta_0 = a_0,$$

$$\theta_f = a_0 + a_1 t_f + a_2 t_f^2 + a_3 t_f^3 ,$$

$$a_1 = 0 ,$$

$$a_1 + 2a_2 t_f + 3a_3 t_f^2 = 0,$$

Thus Solving the above four equations for four variables we the equations to be.

$$a_0 = \theta_0,$$
$$a_1 = 0,$$
$$a_2 = \frac{3}{t_f^2}(\theta_f - \theta_0),$$
$$a_3 = \frac{2}{t_f^3}(\theta_f - \theta_0),$$

This equation was used for point to point trajectory planning of the manipulator with velocity constraints at initial and final points.In order to provide a very smooth transitions between end points , similar equations were calculated for multiple points(i.e n number of via points).

Joint Space mapping helped us to complete tasks such as pick and place and drawing a straight line on a board using the marker as a tool. But the hardware interface was unable to provide a angle feedback hence shifted to the simulation of the hardware on gazebo simulation

Dynamixel

DYNAMIXEL (DXL) is a line of high performance networked actuators for robots developed by Korean manufacturer ROBOTIS. ROBOTIS is also the developer and manufacturer for OLLO, Bioloid and DARwIn-OP. DXL is being used by numerous companies, universities, and hobbyist due to its versatile expansion capability, powerful feedback functions, position, speed, internal temperature, input voltage, etc and its simply daisy-chain topology for simplified wiring connections. The RoboPlus software is available free to the public for use with the DXLs, and can be found on the ROBOTIS Download Page.

DXL can be used for multi-joint robot systems such as robotic arms, robotic hand, bipedal robot, hexapod robot, snake robot, scorpions, pan tilts, kinematic art, animatronics and automation, etc

MoveIt!

MoveIt! is a set of packages and tools for doing mobile manipulation in ROS. The official web page (http://moveit.ros.org/) contains the documentations. MoveIt! contains state of the art software for motion planning, manipulation, 3D perception, kinematics, collision checking, control, and navigation. Apart from the command line interface, MoveIt! has GUI to interface a new robot to MoveIt!. Also, there is a RViz plugin which enables motion planning from RViz itself. We have used the MOVEit Python APIs to control our robot.

MoveIt! architecture

Let's start with MoveIt! and its architecture. Understanding the architecture of MoveIt! helps to program and interface the robot to MoveIt!.

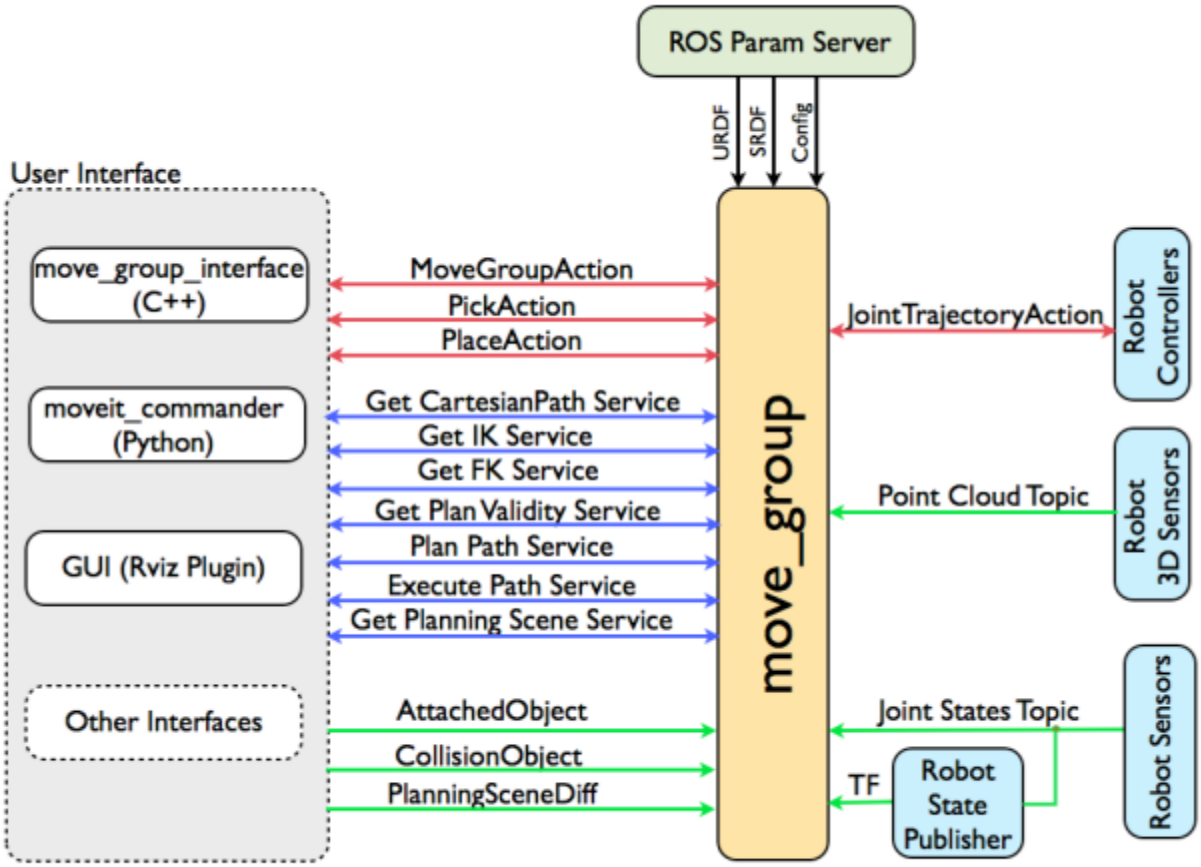


Fig 2.4

The move_group node

We can say that move_group is the heart of MoveIt! as this node acts as an integrator of the various components of the robot and delivers actions/services according to the user's needs.

From the architecture, it's clear that the move_group node collects robot information such as point cloud, joint state of the robot, and transform (T.F) of the robot in the form of topics and services.

From the parameter server, it collects the robot kinematics data, such as robot_description (URDF), SRDF (Semantic Robot Description Format), and the configuration files. The SRDF file and the configuration files are generated while we generate a MoveIt! package for our robot. The configuration files contain the parameter file for setting joint limits, perception, kinematics, end effector, and so on.

When MoveIt! gets all this information about the robot and its configuration, we can say it is properly configured and we can start commanding the robot from the user interfaces. We can either use C++ or Python MoveIt! APIs to command the move_group node to perform actions such as pick/place, IK, FK, among others. Using the RViz motion planning plugin, we can command the robot from the RViz GUI itself.

As we already discussed, the move_group node is an integrator; it does not run any kind of motion planning algorithms but instead connects all the functionalities as plugins. There are plugins for kinematics solvers, motion planning, and so on. We can extend the capabilities through these plugins.
After motion planning, the generated trajectory talks to the controllers in the robot using the Follow JointTrajectory Action interface. This is an action interface in which an action server is run on the robot, and move_node initiates an action client which talks to this server and executes the trajectory on the real robot/Gazebo simulator.

Motion planning using MoveIt!
Assume that we know the starting pose of the robot, a desired goal pose of the robot, the geometrical description of the robot, and geometrical description of the world, then motion planning is the technique to find an optimum path that moves the robot gradually from the start pose to the goal pose, while never touching any obstacles in the world and without colliding with the robot links.

Here the geometrical description of the robot is our URDF file and the geometrical description of the world can also be included in URDF and using laser scanner/3D vision sensor we can generate the world in 3D, which can help to avoid dynamic obstacles rather than static objects defined using URDF.
In the case of the robotic arm, the motion planner should find a trajectory (consisting of joint spaces of each joint) in which the links of the robot should never collide with the environment, avoid self-collision (collision between two robot links), and also not violate the joint limits.
MoveIt! can talk to the motion planners through the plugin interface. We can use any motion planner by simply changing the plugin. This method is highly extensible so we can try our own custom motion planners using this interface. The move_group node talks to the motion planner plugin via the ROS action/services. The default planner for the move_group node is OMPL.

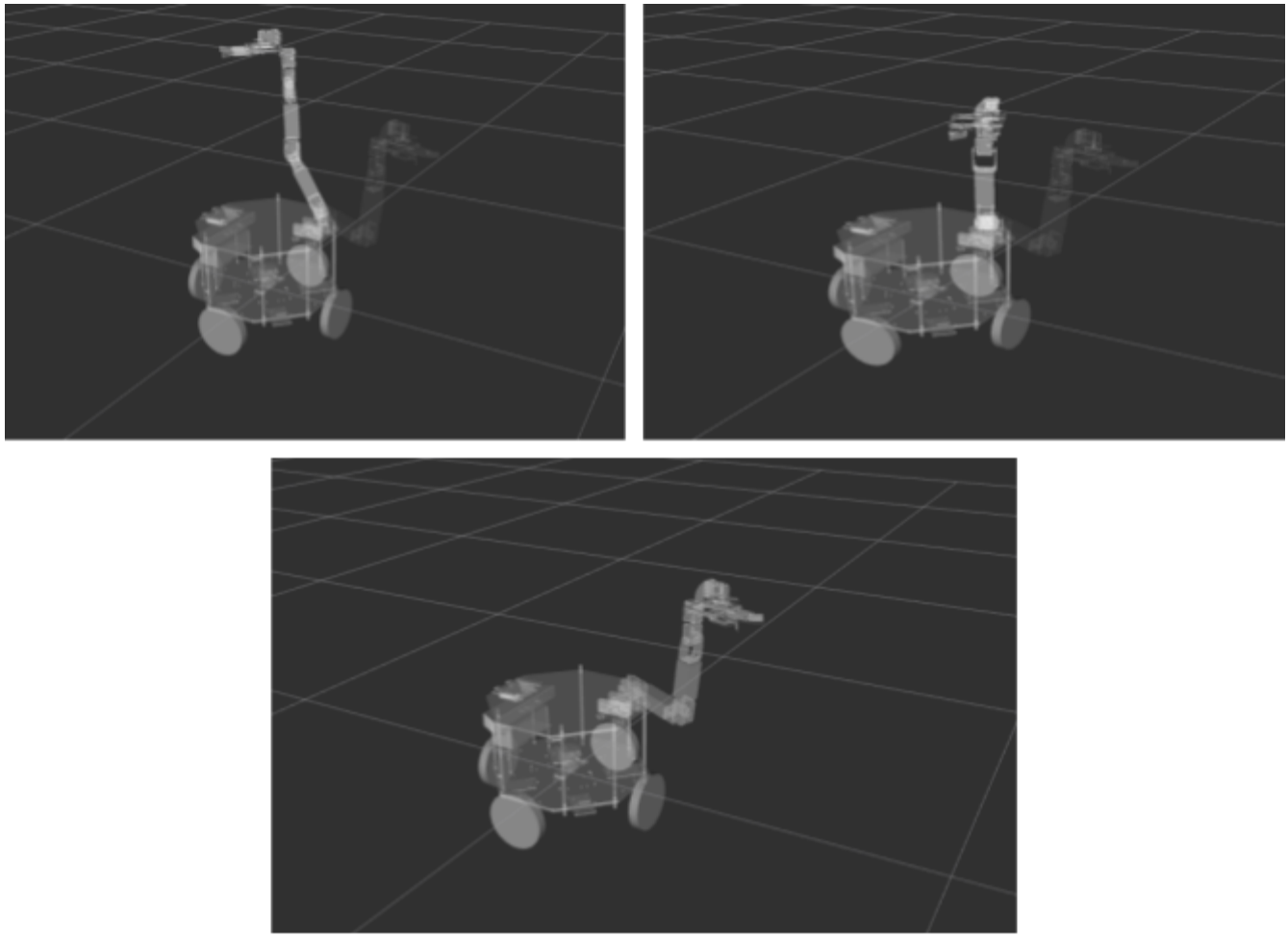


Fig 2.5

To start motion planning, we should send a motion planning request to the motion planner which specified our planning requirements. The planning requirement may be setting a new goal pose of the end-effector, for example, for a pick and place operation.

We can set additional kinematic constraints for the motion planners. Given next are some inbuilt constraints in MoveIt!:

- Position constraints: These restrict the position of a link
- Orientation constraints: These restrict the orientation of a link
- Visibility constraints: These restrict a point on the link to be visible in a particular area (view of a sensor)
- Joint constraints: These restrict a joint within its joint limits
- User-specified constraints: Using these constraints, the user can define his own constraints using the callback functions (These in our case help us to tell the planner to either create a cartesian path or a least effort path)

Using these constraints, we can send a motion planning request and the planner will generate a suitable trajectory according to the request. The move_group node will generate the suitable trajectory from the motion planner which obeys all the constraints. This can be sent to robot joint trajectory controllers.

Motion planning request adapters

The planning request adapters help to pre-process the motion planning request and post process the motion planning response. One of the uses of pre-processing requests is that it helps to correct if there is a violation in the joints states and, for the post processing, it can convert the path generated by the planner to a time-parameterized trajectory. Following are some of the default planning request adapters in MoveIt!:

- FixStartStateBounds: If a joint state is slightly outside the joint limits, then this adapter can fix the initial joint limits within the limits.
- FixWorkspaceBounds: This specifies a workspace for planning with a cube size of 10m x 10m x 10m.
- FixStar State Collision: This adapter samples a new collision free configuration if the existing joint configuration is in collision. It makes a new configuration by changing the current configuration by a small factor called jiggle_factor.
- FixStartStatePathConstraints: This adapter is used when the initial pose of the robot does not obey the path constraints. In this, it finds a near pose which satisfies the path constraints and uses that pose as the initial state.
- AddTime Parameterization: This adapter parameterized the motion plan by applying the velocity and acceleration constraints.

MoveIt! planning scene

The term planning scene is used to represent the world around the robot and also store the state of the robot itself. The planning scene monitor inside move_group maintains the planning scene representation. The move_group node consists of another section called the world geometry monitor, which builds the world geometry from the sensors of the robot and from the user input. The planning scene monitor reads the joint_states topic from the robot, and the sensor information and world geometry from the world geometry monitor. The world scene monitor reads from the occupancy map monitor, which uses 3D perception to build a 3D representation of the environment, called Octomap.

The Octomap can be generated from point clouds which are handled by a point cloud occupancy map update plugin and depth images handled by a depth image occupancy map updater. The following image shows the representation of the planning scene.

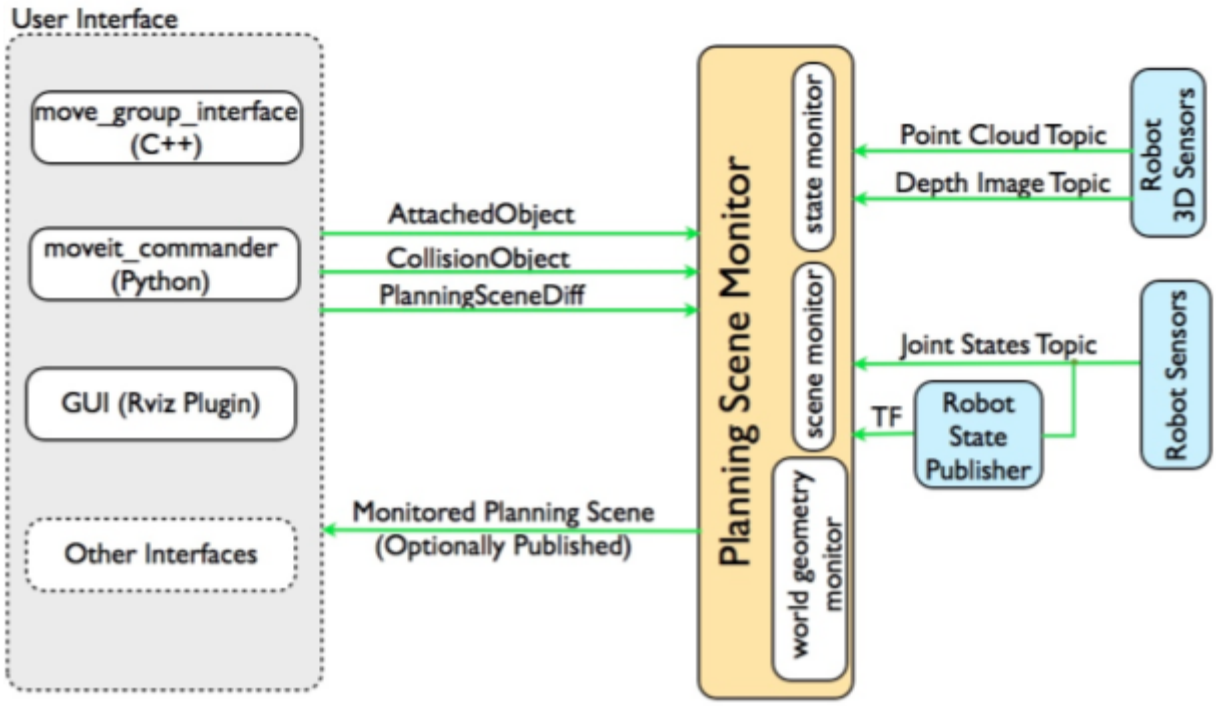


Fig 2.6

MoveIt! kinematics handling

MoveIt! provides a great flexibility to switch the inverse kinematics algorithms using the robot plugins. Users can write their own IK solver as a MoveIt! plugin and switch from the default solver plugin whenever required. The default IK solver in MoveIt! is a numerical jacobian-based solver.

Compared to the analytic solvers, the numerical solver can take time to solve IK. The package called IKFast can be used to generate a C++ code for solving IK using analytical methods, which can be used for different kinds of robot manipulator and perform better if the DOF is less than 6. This C++ code can also be converted into the MoveIt! plugin by using some ROS tool.

MoveIt! collision checking

The CollisionWorld object inside MoveIt! is used to find collisions inside a planning scene which is using the FCL (Flexible Collision Library) package as a backend. MoveIt! supports collision checking for different types of objects, such as meshes, primitive shapes such as boxes, cylinders, cones, spheres, and such, and Octomap.

The collision checking is one of the computationally expensive tasks during motion planning. To reduce this computation, MoveIt! provides a matrix called ACM (Allowed Collision Matrix). It contains a binary value corresponding to the need to check for collision between two pairs of bodies. If

the value of matrix is 1, it means collision of the corresponding pair is not needed. We can set the value as 1 where the bodies are always so far that they would never collide with each other. Optimizing ACM can reduce the total computation needed for collision avoidance. After discussing the basic concepts in MoveIt!, we can now discuss how to interface a robotic arm into MoveIt!. For interfacing a robot arm in MoveIt!, we need to satisfy the components that we saw in Figure 1. The move_group node essentially requires parameters such as URDF, SRDF, config files, and joint states topics along with TF from a robot to start with motion planning.

3. Visual Perception and Trajectory Planning

3.1 World Monitoring

3.1.1 Introduction

The world geometry monitor builds world geometry using information from the sensors on the robot and from user input. It uses the *occupancy map monitor* described below to build a 3D representation of the environment around the robot and augments that with information on the *planning_scene* topic for adding object information.

3.1.2 3D Perception

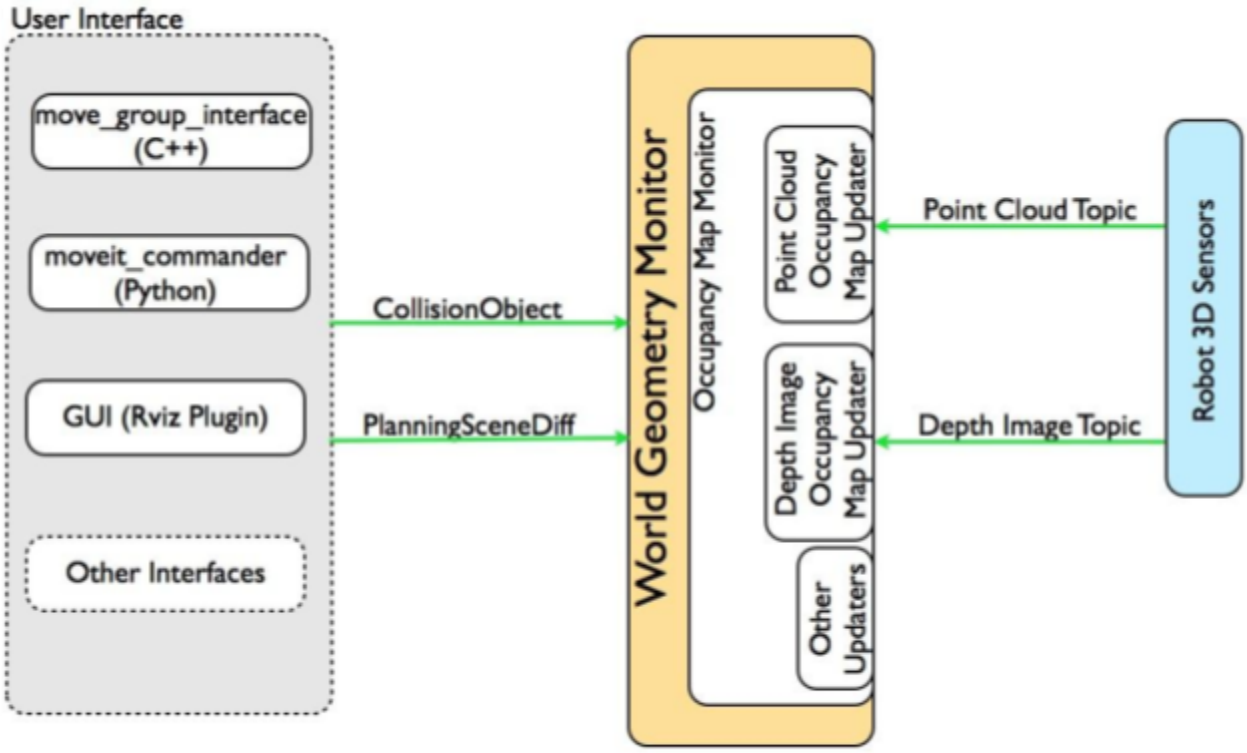


Fig 3.1

3D perception in MoveIt is handled by the occupancy map monitor. The occupancy map monitor uses a plugin architecture to handle different kinds of sensor input as shown in the Figure above. In particular, MoveIt has inbuilt support for handling two kinds of inputs:

- Point clouds: handled by the point cloud occupancy map updater plugin
- Depth images: handled by the depth image occupancy map updater plugin

Note that you can add your own types of updaters as a plugin to the occupancy map monitor.

OctoMap

The Occupancy map monitor uses an Octomap to maintain the occupancy map of the environment. The Octomap can actually encode probabilistic information about individual cells although this information is not currently used in MoveIt. The Octomap can directly be passed into FCL, the collision checking library that MoveIt uses.

OctoMap can be used with any kind of distance sensor, as long as an inverse sensor model is available. Since our real world datasets were mostly acquired with laser range finders, we employ a beam-based inverse sensor model which assumes that endpoints of a measurement correspond to obstacle surfaces and that the line of sight between sensor origin and endpoint does not contain any obstacles. The occupancy probability of all volumes is initialized to the uniform prior of P(n) = 0.5. To efficiently determine the map cells which need to be updated, a ray-casting operation is performed that determines voxels along a beam from the sensor origin to the measured endpoint. For efficiency, we use a 3D variant of the Bresenham algorithm to approximate the beam (Amanatides and Woo, 1987). Volumes along the beam are updated as described in Sect. 3.2 using the following inverse sensor model: L(n | zt) = (locc if beam is reflected within volume free if beam traversed volume (7) Throughout our experiments, we used log-odds values of locc = 0.85 and lfree = −0.4, corresponding to probabilities of 0.7 and 0.4 for occupied and free volumes, respectively

The clamping thresholds are set to lmin = −2 and lmax = 3.5, corresponding to the probabilities of 0.12 and 0.97. We experimentally determined these values to work best for our use case of mapping mostly static environments with laser range finders, while still preserving map updatability for occasional changes. By adapting these changeable thresholds, a stronger compression can be achieved. There is a trade-off between map confidence and compression. Discretization effects of the ray-casting operation can lead to undesired results when using a sweeping laser range finder. During a sensor sweep over flat surfaces at shallow angles, volumes measured occupied in one 2D scan may be marked as free in the ray-casting of following scans. Such undesired updates usually creates holes in the modeled surface, as shown in the example. To overcome this problem, we treat a collection of scan lines in a sensor sweep from the same location as single 3D point cloud in our mapping approach. Since measurements of laser scanners usually result from reflections at obstacle surfaces, we ensure that the voxels corresponding to endpoints are updated as occupied. More precisely, whenever a voxel is updated as occupied, it is not updated as free in the same measurement update of the map. By updating the map in this way, the described effect can be prevented and the environment is represented accurately, as can be seen in figure

A laser scanner sweeps over a flat surface at a shallow angle by rotating. A cell measured occupied in the first scan (top) is updated as free in the following scan (bottom) after the sensor rotated. Occupied cells are visualized as gray boxes, free cells are visualized in white.

A simulated noise-free 3D laser scan (left) is integrated into our 3D map structure. Sensor sweeps at

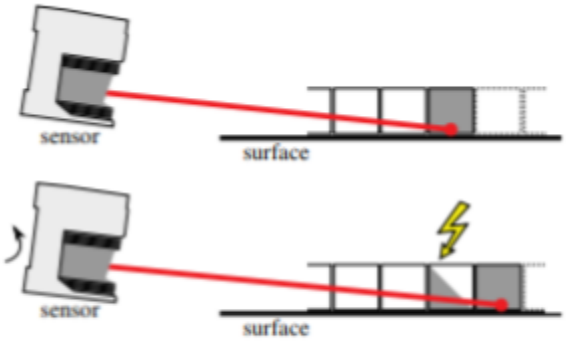


Fig 3.2

shallow angles lead to undesired discretization effects (center). By updating each volume at most once, the map correctly represents the environment (right). For clarity, only occupied cells are shown.

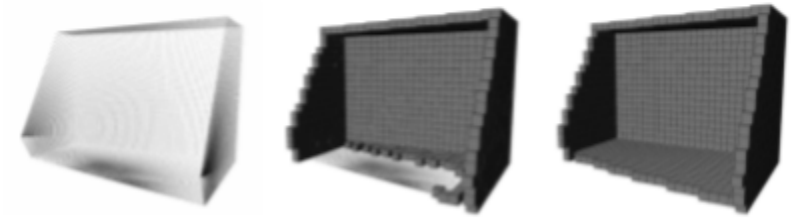


Fig 3.3

Depth Image Occupancy Map Updater

The depth image occupancy map updater includes its own self-filter, i.e. it will remove visible parts of the robot from the depth map. It uses current information about the robot (the robot state) to carry out this operation.

3.1.3 Collision Detection

The problems of collision and proximity computation are widely studied in various fields including robotics, simulated environments, haptics, computer games and computational geometry. The set of

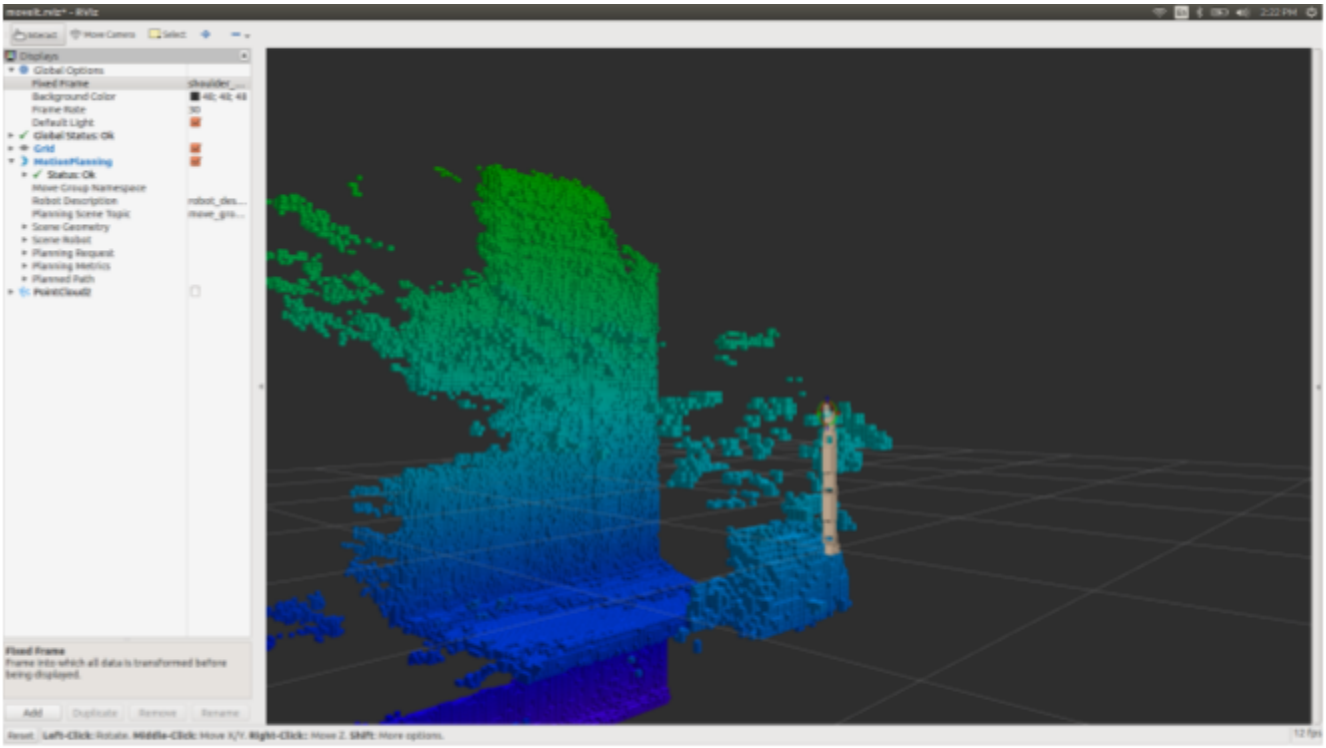


Fig 3.4

queries includes discrete collision checking, separation distance computation between two non overlapping objects, first point of contact computation between continuous moving objects, and penetration depth computation between overlapping objects. Furthermore, the underlying geometric representations may correspond to rigid objects (e.g., computer games), articulated models (e.g., mobile manipulators), deformable models (e.g., surgical or cloth simulators) or point-cloud datasets (e.g., captured using camera or LIDAR sensors on a robot). Many efficient algorithms have been proposed to perform collision and proximity queries on various types of models. At a broad level, they can be classified based on the underlying query or the model representation. Some of the commonly-used techniques for polygonal models are based on bounding volume hierarchies, which can

be used for collision and separation distance queries, and can be extended to deformable models. Moreover, many of these algorithms have been used to design widely used libraries such as I-COLLIDE, Bullet, ODE, RAPID, PQP, SOLID, OPCODE, V-Clip, Self-CCD, etc. However, these libraries have two main restrictions:

1. They are limited to specific queries (e.g., discrete collision checking or separation distance computation) on certain types of models (e.g., convex polytopes or rigid objects).
2. It is hard to modify or extend these libraries in terms of using a different algorithm or representation. For example, SOLID is designed to perform collision checking using axis-aligned bounding box (AABB) trees; RAPID is designed for collision detection using oriented bounding box (OBB) trees, and PQP performs separation distance queries using rectangular swept sphere (RSS) trees. It is hard to use a different bounding volume with each of these libraries or use a different hierarchy computation or traversal scheme. Many applications need to perform different collision and proximity queries. Continuous collision detection queries are useful for grasp planning executed by the robot to generate grasps for the objects. The robot uses sample-based motion planners to compute collision-free paths. It is well known that a high fraction of running time for sample-based planning is spent in collision/proximity queries, underlining the need for fast efficient proximity and collision queries.

Point Cloud Collision Detection: There has been relatively little work in terms of handling collisions between point clouds or between point clouds and unstructured meshes. With the recent advances in RGB-D cameras and LIDAR sensors, there is increased interest in performing various queries on noisy point-cloud datasets. FCL supports collision checking between triangle meshes/soups and point clouds as well as collision checking between point clouds. The former is useful for collision checking between robot parts and the environment, while the latter is used for collision checking between scanned objects (e.g. held by a robot gripper) and the environment. The point cloud collision checking algorithms in FCL also take into account noise in the point cloud data arising from various sensors as well as the inherent shap.

3.1.4 Choosing a detector

Which one to choose? Given the benchmarks, Faster R-CNN might look like the natural choice. Until recently however, it was hard to make Faster R-CNN's ROI Pooling layer work in Tensorflow — hence the custom C++ layers in the github repositories. In a talk from October 2016, Google hint that new tensorflow ops may have fixed this, but we only discovered this recently.

From the VOC2012 benchmark, the next-best model seems to be SSD512 — Single Shot Multibox. Although slightly less accurate than Faster R-CNN, it has some advantages, too. SSD512 runs at 22fps on a NVIDIA Titan X, compared to just 7fps for Faster R-CNN. An only slightly less accurate variant,

SSD300, runs at 59fps. In the end, we chose SSD because of its conceptual simplicity, because it is fast, and because it is competitive with the best detectors available.

3.2 Object Localization and Detection

3.2.1 Introduction

On this chapter we're going to learn about using convolution neural networks to localize and detect objects on images

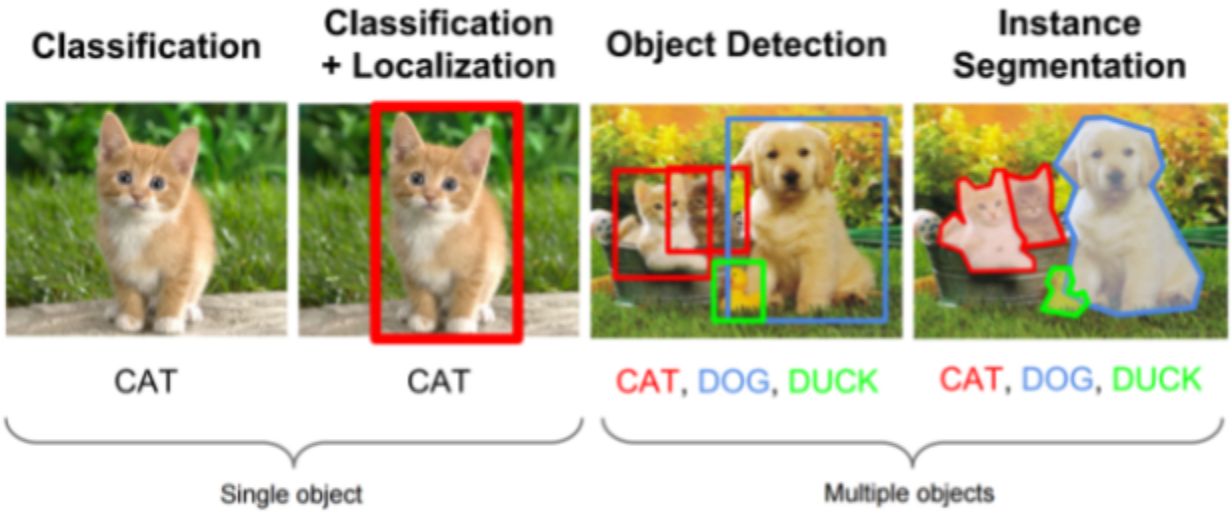


Fig 3.5

- RCNN
- Fast RCNN
- Faster RCNN
- Yolo
- SSD

Localize objects with regression

Regression is about returning a number instead of a class, in our case we're going to return 4 numbers (x0,y0,width,height) that are related to a bounding box. You train this system with an image an a ground truth bounding box, and use L2 distance to calculate the loss between the predicted bounding box and the ground truth.

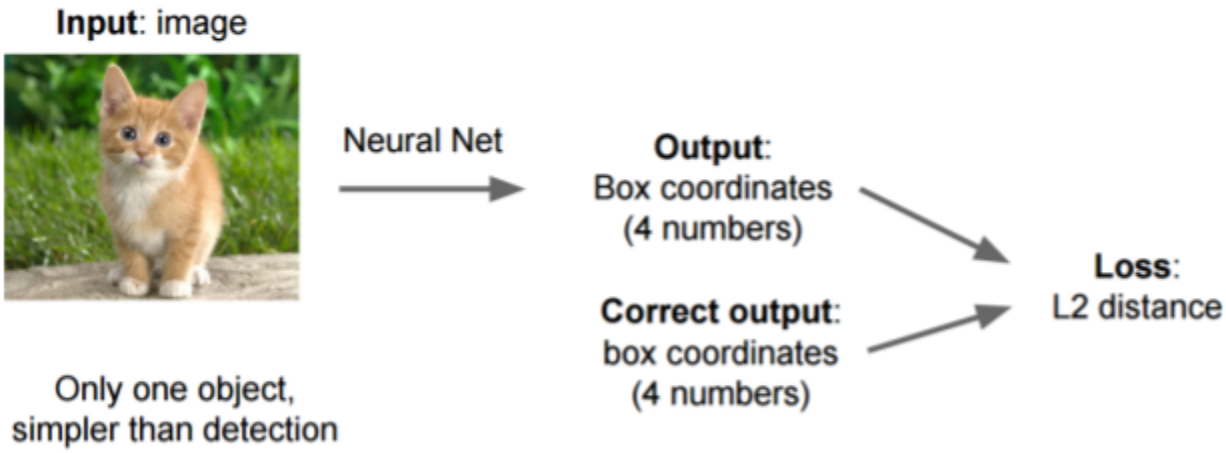


Fig 3.6

Normally what you do is attach another fully connected layer on the last convolution layer

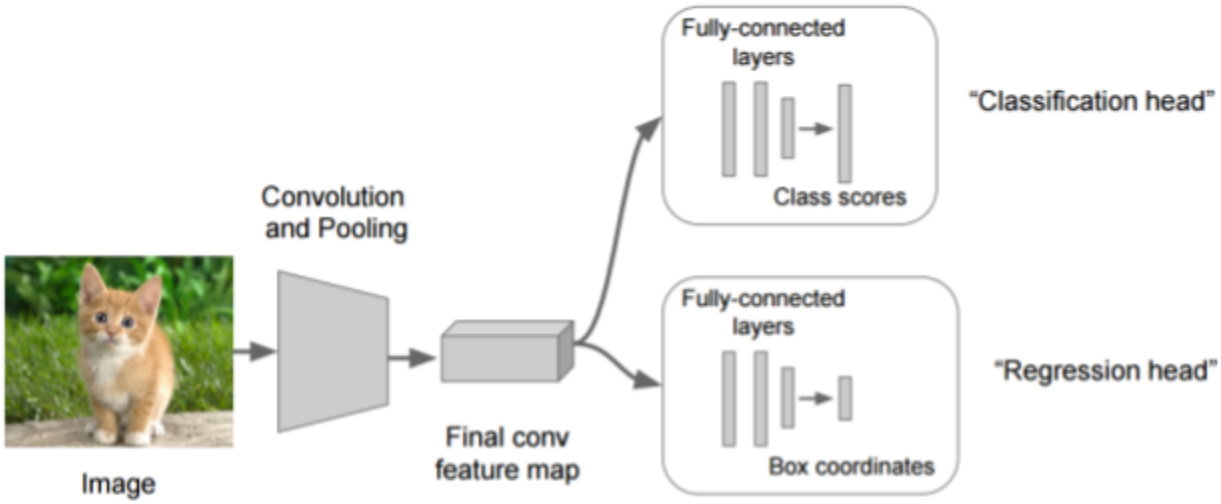


Fig 3.7

This will work only for one object at a time.

Some people attach the regression part after the last convolution (Overfeat) layer, while others attach after the fully connected layer (RCNN). Both works.

Comparing bounding box prediction accuracy

Basically we need to compare if the Intersection Over Union (ioU) between the prediction and the ground truth is bigger than some threshold (ex > 0.5)

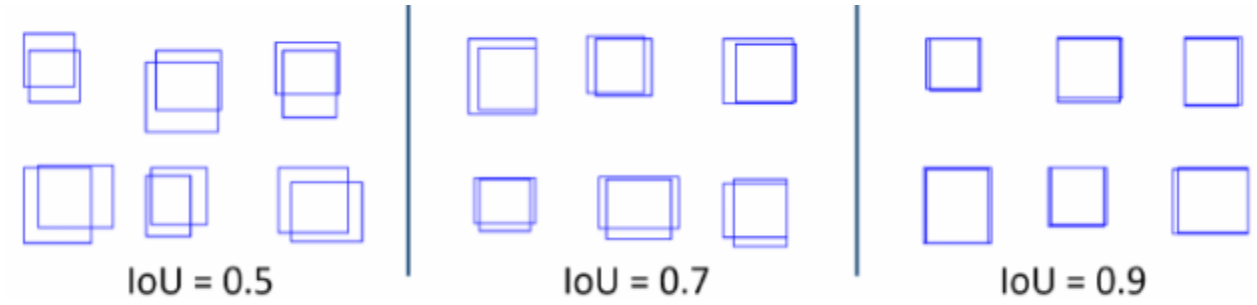


Fig 3.8

3.2.2 Multibox Single Shot Detector (SSD)

Localizing with Convolution neural networks

One way to reuse the computation that is already made during classification to localize objects is to grab activations from the final conv layers. At this point we still have spatial information but represented on a smaller version. For example an input image of size 640x480x3 passing into an inception model will have it's spatial information compressed into a 13x18x2048 size on it's final layers.

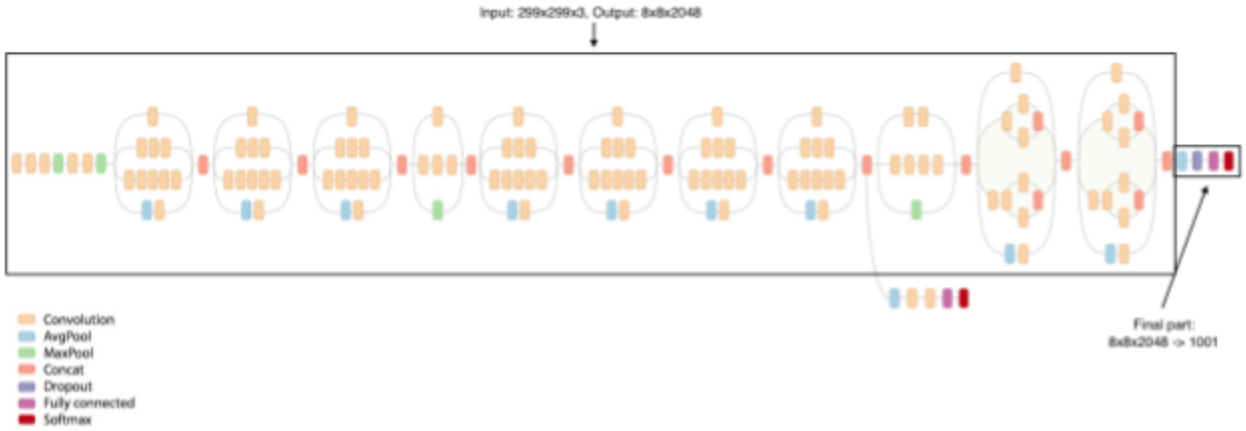


Fig 3.9

Fig 3.10

What happens is that on the final layers each "pixel" represent a larger area of the input image so we can use those cells to infer the object position. One thing to pay attention is that even though we are squeezing the image to a lower spatial dimension, the tensor is quite deep, so not much information is lost. (This is not entirely true when using pooling layers).

At this point imagine that you could use a 1x1 CONV layer to classify each cell as a class (ex: Pedestrian/Background), also from the same layer you could attach another CONV or FC layer to predict 4 numbers (Bounding box). In this way you get both class scores and location from one.

One common mistake is to think that we're actually dividing the input image into a grid, this does not happen! What actually happens is that each layer represent the input image with few spatial data but with bigger depth. On training time we will do some sort of matching between our ground truth and virtual cells. Also those cells will actually overlap they are not perfectly tiled.

Also regarding the number of detection, each one of those cells could detect an object. So the output of this model could be 13x18 detections.

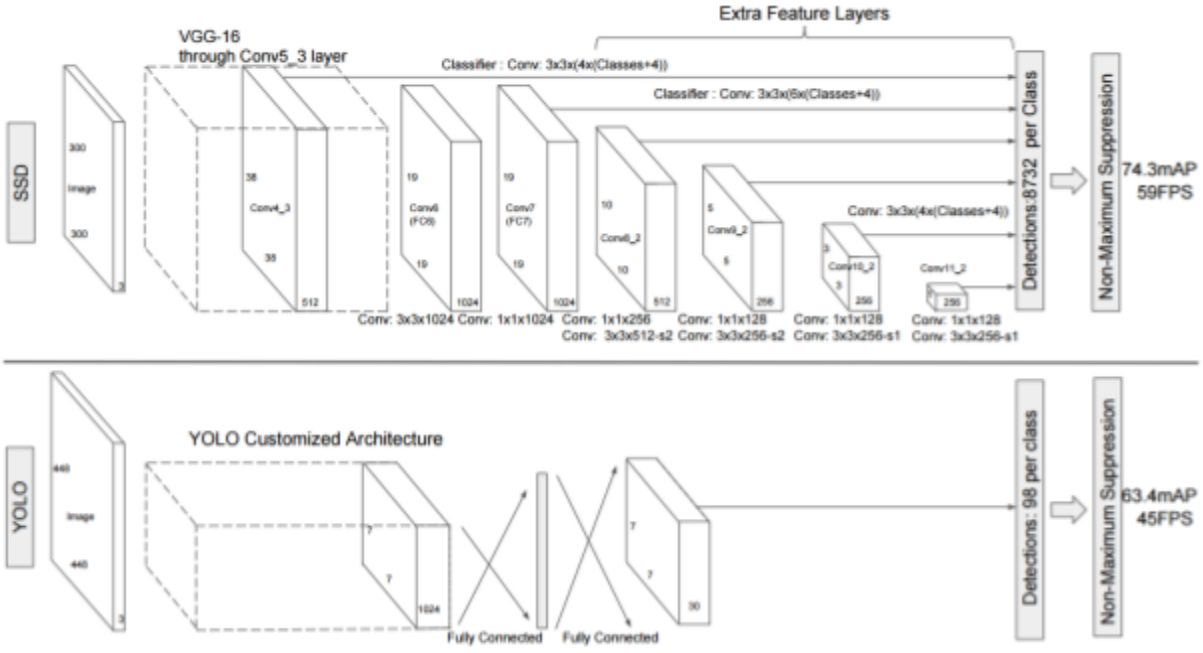


Fig 3.11

The SSD detector differs from others single shot detectors due to the usage of multiple layers that provide a finer accuracy on objects with different scales. (Each deeper layer will see bigger objects).

The SSD normally start with a VGG on Resnet pre-trained model that is converted to a fully convolutional neural network. Then we attach some extra conv layers, which will actually help to

handle bigger objects. The SSD architecture can in principle be used with any deep network base model.

One important point to notice is that after the image is passed on the VGG network, some conv layers are added producing feature maps of sizes 19x19, 10x10, 5x5, 3x3, 1x1. These, together with the 38x38 feature map produced by VGG's conv4_3, are the feature maps which will be used to predict bounding boxes.

There the conv4_3 is responsible to detect the smallest objects while the conv11_2 is responsible for the biggest objects.

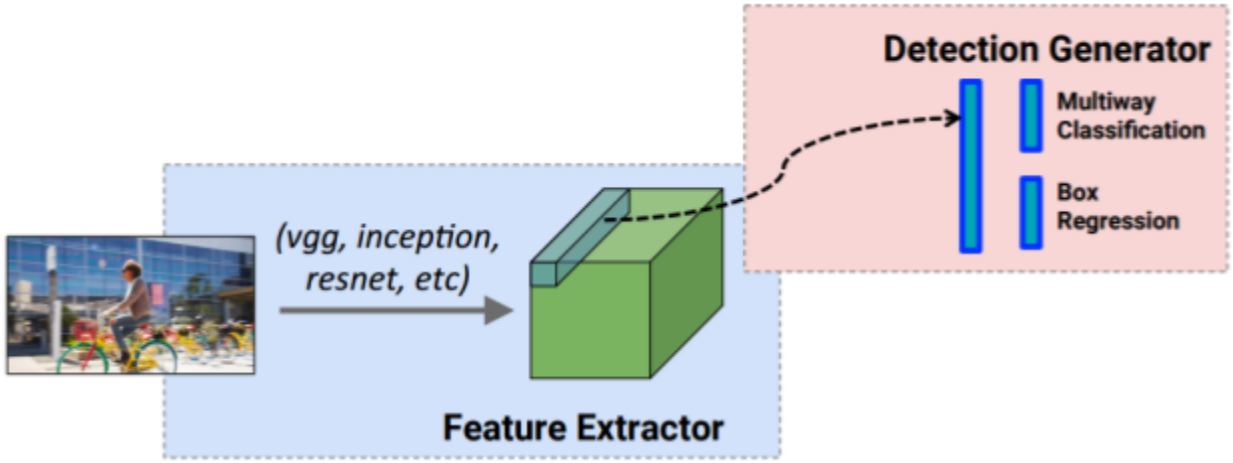


Fig 3.12

As shown on the diagram some activations are "grabbed" from the network and passed to a specialized sub-network that should work as a classifier and localizer. During prediction we use a Non-maxima suppression algorithm to filter the multiple boxes per object that may appear.

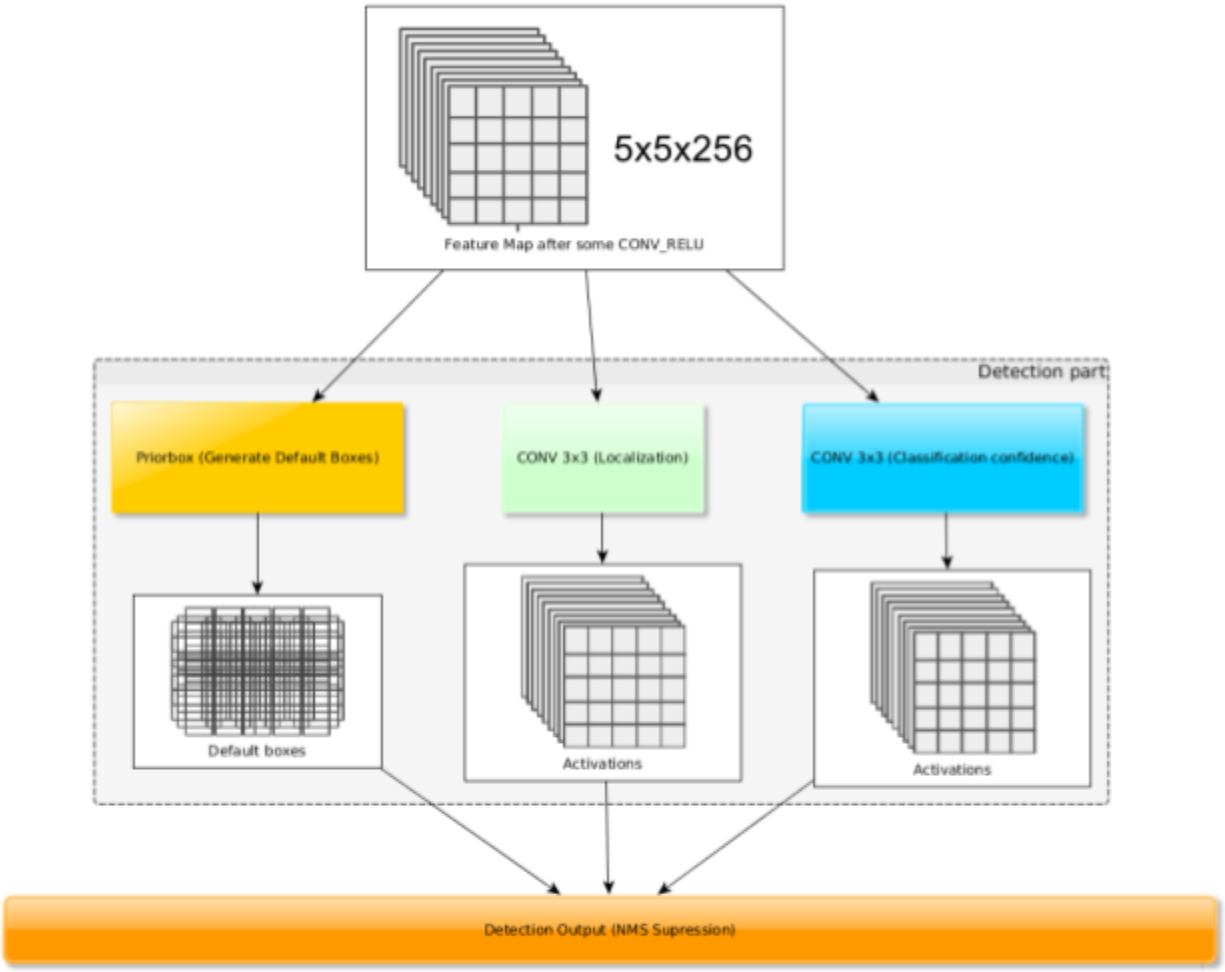


Fig 3.13

Anchor(Priors or Default boxes) concept

Anchors are a collection of boxes overlaid on the image at different spatial locations, scales and aspect ratios that act as reference points on the ground truth images. It's like the Yolo idea where each cell on the activation map has multiple boxes.

A model is then trained to make two predictions for each anchor:

1. A discrete class prediction for each anchor

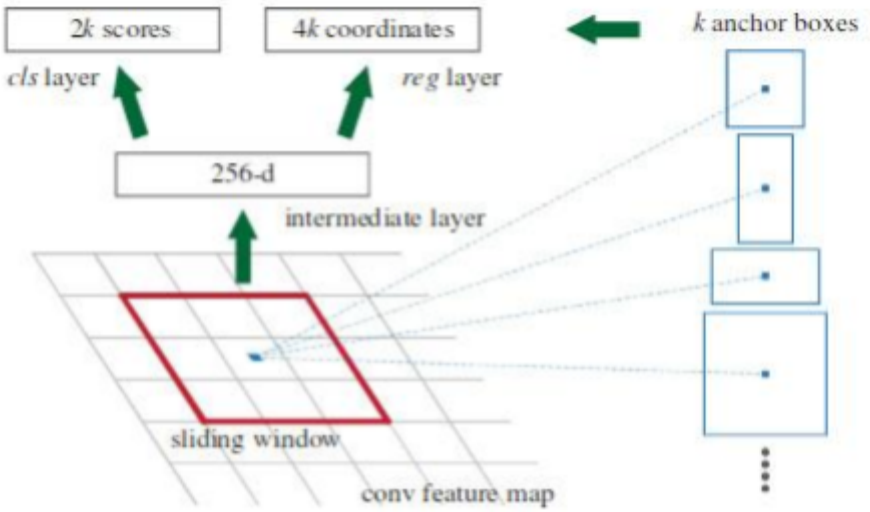


Fig 3.14

2. A continuous prediction of an offset by which the anchor needs to be shifted to fit the ground-truth bounding box

During training SSD matches objects with _default boxes _of different aspects. Each element of the feature map (cell) has a number of default boxes associated with it. Any default box with an IoU (Jaccard index) greater than 0.5 is considered a match.

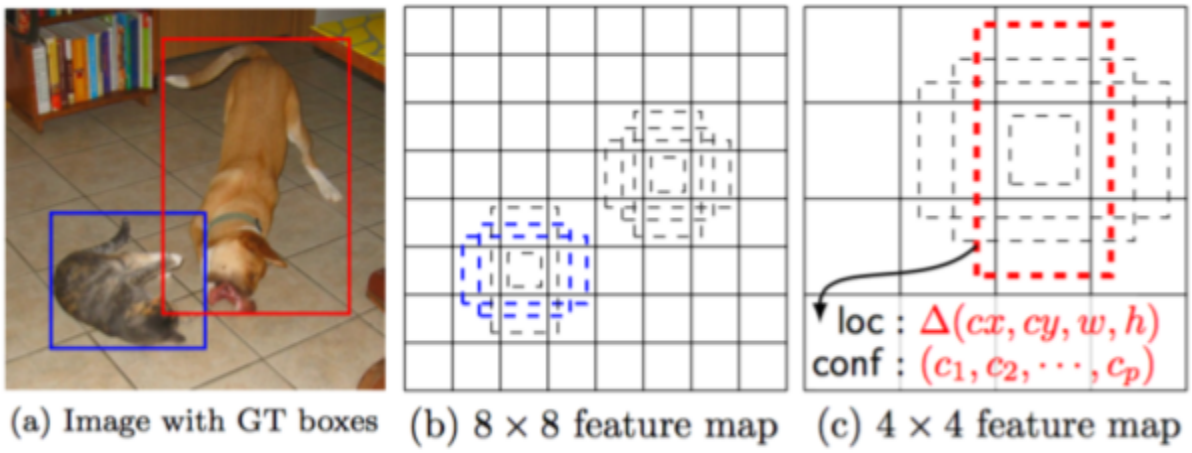


Fig 3.15

Consider the image above, observe that the cat is has 2 boxes that match on the 8x8 feature map, but none on the dog. Now on the 4x4 feature map there is one box that matches the dog.

It is important to note that the boxes in the 8x8 feature map are smaller than those in the 4x4 feature map: SSD grab some feature maps, each responsible for a different scale of objects, allowing it to identify objects across a large range of scales.

For each default box on each cell the network output the following:

- A probability vector of length c, where c are the number of classes plus the background class that indicates no object.
- A vector with 4 elements (x,y,width,height) representing the offset required move the default box position to the real object.

Multibox Loss Function

During training we minimize a combined classification and regression loss.

$$L(x, c, l, g) = \frac{1}{N}(L_{conf}(x, c) + \alpha L_{loc}(x, l, g))$$

As Yolo the SSD loss balance the classification objective and the localization objective.

Localization Loss

$$L_{loc}(x, l, g) = \sum_{i \in Pos}^{N} \sum_{m \in \{cx, cy, w, h\}} x_{ij}^{k} \text{smooth}_{\text{L1}}(l_i^m - \hat{g}_j^m)$$

$$\hat{g}_j^{cx} = (g_j^{cx} - d_i^{cx})/d_i^w \qquad \hat{g}_j^{cy} = (g_j^{cy} - d_i^{cy})/d_i^h$$

$$\hat{g}_j^w = \log\left(\frac{g_j^w}{d_i^w}\right) \qquad \hat{g}_j^h = \log\left(\frac{g_j^h}{d_i^h}\right)$$

$$\text{smooth}_{L_1}(x) = \begin{cases} 0.5x^2 & \text{if } |x| < 1 \\ |x| - 0.5 & \text{otherwise,} \end{cases}$$

Classification Loss

$$L_{conf}(x, c) = -\sum_{i \in Pos}^{N} x_{ij}^p log(\hat{c}_i^p) - \sum_{i \in Neg} log(\hat{c}_i^0) \quad \text{where} \quad \hat{c}_i^p = \frac{\exp(c_i^p)}{\sum_p \exp(c_i^p)}$$

3.2.3 Depth Using Kinect

The 'world coordinates' of a point cloud are represented in the transform data. If your pointcloud is in the sensor frame, and you have a tf transform (link) available to a map or world frame, you can use the tf library to transform the point cloud into the desired frame, which will transform the positions of the points in the pointcloud

Ros Numpy package is used to access the centroid pixel coordinates [x,y,z] from the pointcloud made available through Kinect v2.But for a better estimation of the of the coordinate the median of all the pixel coordinates [x,y,z] present in the bounding box is taken.

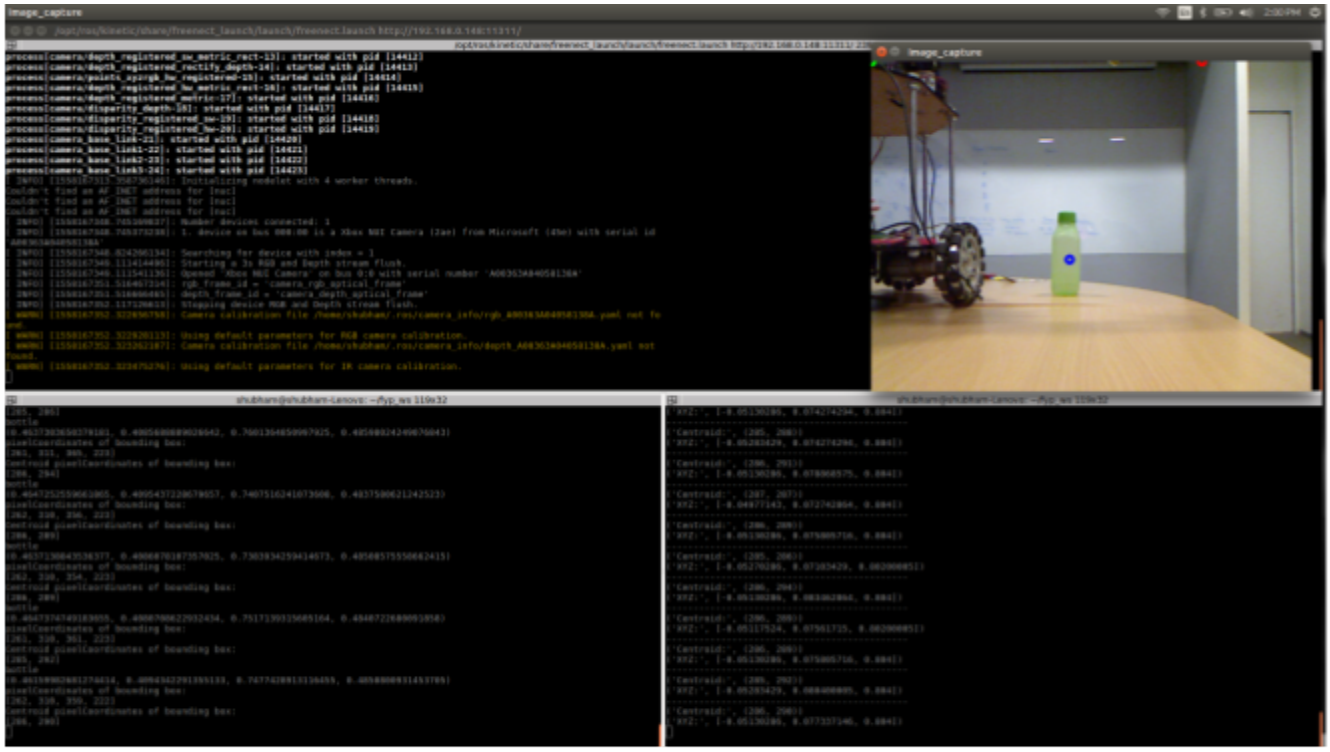


Fig 3.16

4. Mechanism

4.1 Gripper Mechanism

The gripper mechanism was made in two design phases. In the first phase we made a parallel gripper which had the capability of picking objects with a set geometry and material composition. In the second phase we made an underactuated gripper which mimicked the gripping mechanism of human fingers with the use of only one actuation. Let us look at each design in depth:

Phase 1

Parallel Gripper design

This structure is designed to grip objects with parallel straight faces only. The structure can be seen below. The entire design was printed using the PLA material and extra ribs were added to increase the traction of gripping. However this design had a lot of limitations as to what it can pick and place.


Front View

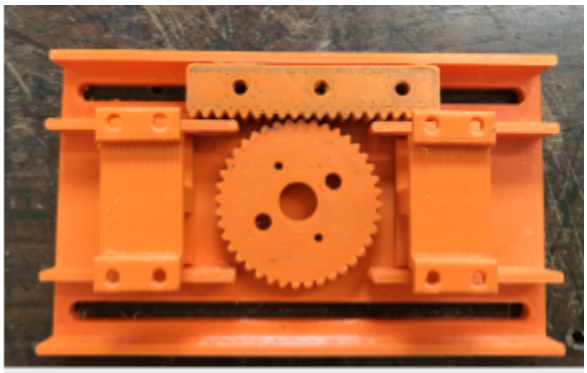

Bottom View

Fig 4.1

Thus in the second phase we went for a new design based on the principles of underactuated robotics. This system was designed to be adaptable according to the shape of the object and yet work with just one actuator.

Phase 2
Two Phalanx underactuated gripper

Underactuation
Underactuation expresses the property of a system to have an input vector of smaller dimensions than the output vector. In robotics, it means, having fewer number of actuators than degrees of freedom (DOF). The idea behind using underactuation in grasping is to be able to grasp an object by adapting to its shape automatically using simple control rather than having to control and coordinate several actions. This mechanical intelligence embedded in the hand is based on the principle of differential systems. These devices distribute one input to several outputs and the ratio between the outputs is determined by the design parameters like link lengths and spring stiffness. The same philosophy of intelligent design is commonly found in mechanical linkages where the different link lengths and joint types are determined at the design stage in order to follow a particular trajectory. If this trajectory is fixed in time only one DOF is needed to build the mechanism. So, the postulate behind the use of underactuation in grasping can be stated as follows: if the task to be performed is grasping, it should be possible to accomplish this one action using one single actuator.

Need for underactuation
The robotic arm is mounted on a mobile platform that imposes a lot of restrictions on the manipulator weight and size. Moreover the control system for a multi-actuated gripper becomes complicated and requires additional systems to synchronization. This forms the motivation behind designing an underactuated gripper.

Hardware Specifications
The gripper is actuated using the Dynamixel AX-12 motor. This choice was made to ensure maximum compatibility with the rest of the manipulator and make precision control possible.

Skeletal Design

Fig 4.2

The term phalanx stands for the individual segments of the human fingers. In the design shown above the white segment is the first phalanx. The second phalanx is the segment after the first one that comes in contact with the object collectively. The lengths of the phalanx are decided as per certain rules of stable gripping.

This design is created on the Autodesk Fusion 360 software using parametric modelling. The design had been simulated for possible configurations of the phalanx.

The skeletal design is used to 3D print the required parts, which are then assembled.

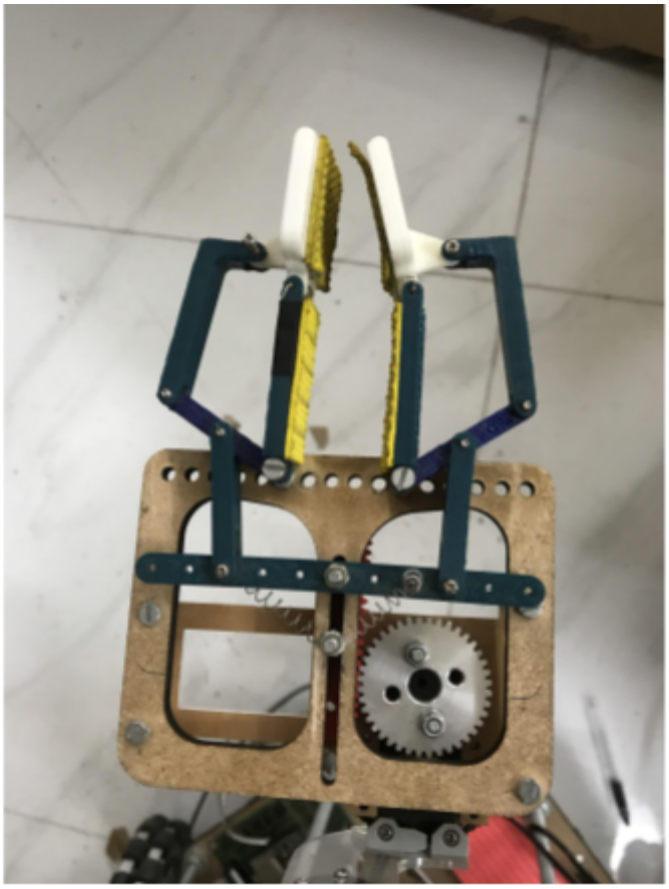


Fig 4.3

4.2 Robot Locomotion Design

4.2.1 Types of mobile robot drives

Different types of robot bases/drives are used in different applications depending on the conditions in which the robot has to traverse. Here are a few robot drives, which are commonly used:

Differential Drive

This is the most common control mechanism for roboticists. The concept is simple; Velocity difference between two motors drive the robot in any required path and direction. Hence the name Differential drive. Differential wheeled robot can have two independently driven wheels fixed on a common horizontal axis or three wheels where two independently driven wheels and a roller-ball or a castor attached to maintain equilibrium.

There are three fundamental cases that can happen in a differential wheeled robot:

1. If the angular velocities are identical in terms of both values and direction, i.e. if both the wheels are driven at the same speed and same direction (either clockwise or anticlockwise) then the robot tends to spin around its vertical axis. This complete turn capability is one of the greatest advantages of a differentially driven robot.

2. If the angular velocities are identical in terms of values and opposite in direction, i.e. if both the wheels are driven in the same speed but in the opposite direction (One clockwise and other anticlockwise) then the robot is more likely to follow a linear path, either forward or backward based on the motors spin.

3. If the angular velocities are different in terms of values (same or different direction), i.e. if the wheels are driven at different speeds in the same direction or opposite direction, then the robot makes a curve motion. Lastly, if one of the wheels rotate and the other stays still then the robot almost makes a 90°turn. Manipulating the drive speed and direction can give some interesting drive paths.

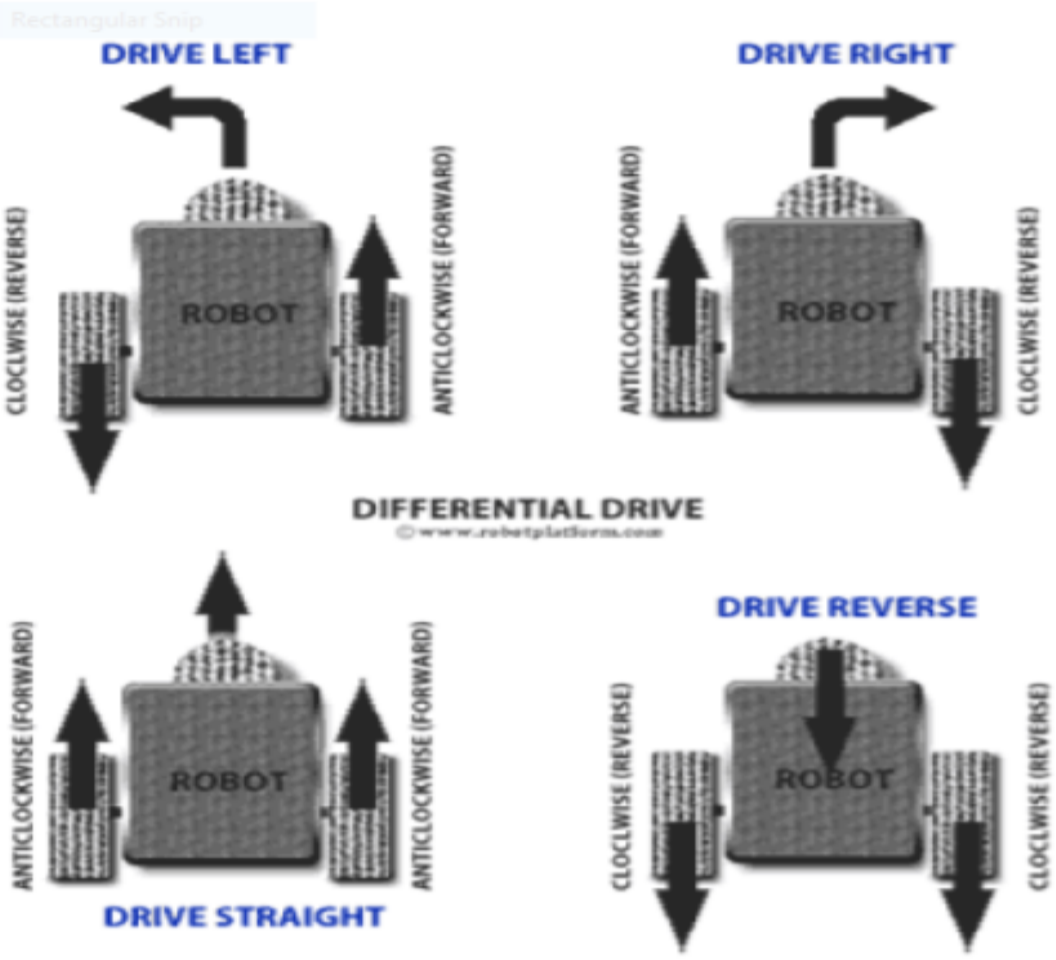


Fig 4.4

Design, mechanical construction and control algorithm can never get any simpler than this driving technique, and the concept can be incorporated in almost any kind of robots including legged robots.
One of the major disadvantages of this control is that the robot does not drive as expected. It neither drives along a straight line nor turn exactly at expected angles, especially when we use DC motors. This is due to difference in the number of rotations of each wheel in a given amount of time. To handle this problem, we need to add correction factor to the motor speed. For example if you intend to drive your robot in a linear path and feel that the robot is turning towards one side, then a correction factor can be added to reduce the speed of the other wheel.

Skid Steering

Skid steering is another driving mechanism implemented on vehicles with either tracks or wheels , which uses differential drive concept. Most common Skid steered vehicles are tracked tanks and bulldozers.
This method engages one side of the tracks or wheels and turning is done by generating differential velocity at opposite side of a vehicle, as the wheels or tracks in the vehicle are non-steerable.
In differentially driven robot, there is a castor, which balances the robot, and in Skid Steer drive, the castor is replaced with two driving wheels. Suppose you need your robot to turn left; then the right wheels or tracks are driven forward and the left wheels or tracks are driven backward until the robot turns right. If the drive continuous in the same way, then the robot will have a 360 turn with almost 0

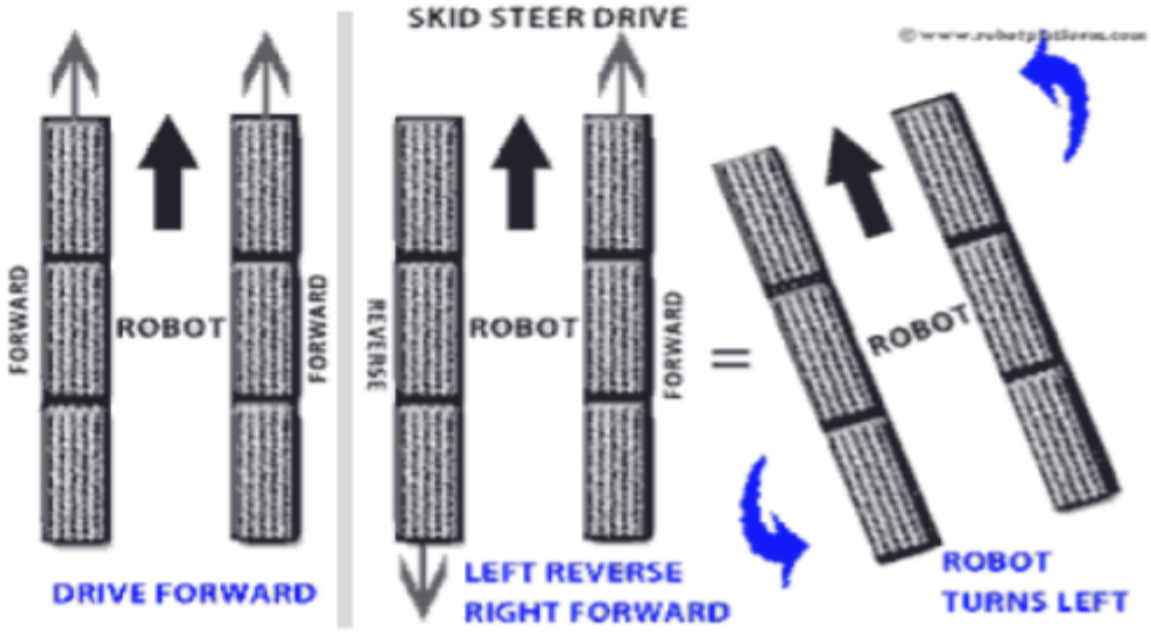


Fig 4.5

radius. Suppose if there are four wheels attached on each side, then the front and rear wheels rotate more and the centre wheels almost skid to turn. Thus the name Skid steer.

Some of the advantages of skid steer drive are:
1. They have greater traction and especially good for rough terrain
2. Same concept can be used on both tracked robots and wheeled robots
3. Since there are no explicit steering wheels, steering mechanism is not required
4. Since all wheels on each side drive in the same direction, only two motors are enough for driving and steering the robot
5. This method does not require caster wheels and hence eliminates the problems caused by casters
Few drawbacks of Skid steering are:
1. Since it uses skidding or slipping technique, increases wheel / track wear and tear thereby reducing its life
2. Like differentially driven robots, driving in a straight path is a hard to achieve task as both the motors are expected to drive at exactly the same speed. This can still be taken care by encoder feedback, but it adds to the cost and control mechanism complexity.

Omni Directional Drive

Omni directional robots are built using Omni wheels and/or casters. Since Omni wheels have smaller wheels attached perpendicular to the circumference of another bigger wheel, they allow wheels to move in any direction instantly. The major advantage is that they do not need to rotate or turn to move in any direction unlike other designs. In other words, they are Holonomic robots and can move in any direction without changing the orientation. Generally, Omni wheeled robots use either a three-wheeled platform or a four wheeled platform. Each design has its own advantages and disadvantages.
Wheeled omnidirectional platform design:
In this design, 4 Omni wheels are attached at 90°to each other. This means any two wheels are parallel to each other and other two wheels perpendicular. The first and the major benefit is the simplified calculation. Since there are two pairs of wheels, each pair requires only one calculation and all four wheels require only two calculations. Also at any point there are two driving wheels and two free wheels. This makes the two driving wheels 100% efficient and drives the robot at higher speed compared to 3-wheeled design.

The only drawback is that a four-wheeled Omni robot does not balance on irregular terrain and not all four wheels are guaranteed to stay on the same plane. Additional wheel might incur an extra cost, but the advantage makes this seem a minor concern. Irrespective of three, four or any number of wheels, Omni-wheeled robots have few drawbacks:
1. Omni-wheels are expensive
2. They are less efficient since not all wheels are fully utilized for driving and controlling the robot

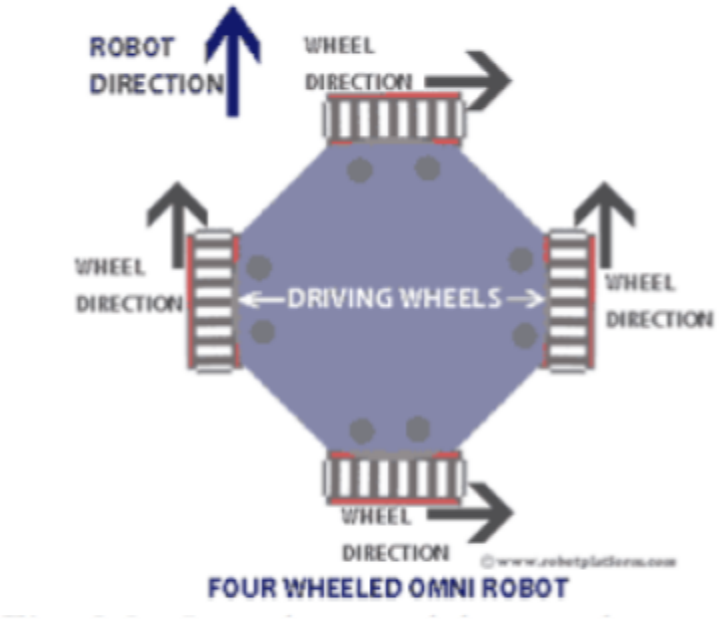


Fig 4.6

3. Since Omni wheels are a combination of many wheels / rollers into one, there is a greater resistance to rotation which leads to greater loss of energy; i.e. Loss due to friction.
4. Since Omni-wheeled robot works on the principle of slippage, position control is difficult

Decision: Reasoning and Conclusion

In order to facilitate maneuverability in compact indoor environments we decided to use a four wheeled Omni drive. Ratings and dimensions of the motors and wheels are as follows:

- DC Planetary Gear motor (encoded motor : 7 pulse per rotation)
- Voltage: 24 Volts
- Rated Speed: 450 RPM
- Torque: 15 kg-cm
- Omni wheel with bearings and 15 cm diameter

5. Simultaneous Localisation and Mapping

5.1 Autonomous Navigation

5.1.1 Introduction

Autonomous navigation of robots in uncontrolled environments is a challenge because it requires a set of subsystems to work together. It requires building a map of the environment, localizing the robot in that map, making a motion plan according to the map, executing that plan with a controller, and other tasks; all at the same time. On the other hand, the autonomous navigation problem is very important for the future of robotics. There are many applications that require this problem to be solved, such as package delivery, cleaning, agriculture, surveillance, search & rescue, construction, and transportation. Note that all these applications occur in uncontrolled environments.

Why is it important to develop autonomous navigation robots? Basically, there are three kinds of robots: operated, automatic and autonomous. Operated robots are those that require control by a human, e.g. teleoperated robots for surgery or army exploration. Automatic robots do preprogrammed and repeated activities in controlled environments, e.g. robotic arms in car production lines or line follower robots. In contrast, autonomous robots do tasks in unstructured environments and make their own decisions as a function of the given goal, e.g. courier robots in hospitals and driverless cars in cities. The tendency is to give robots more autonomy, which means robots do tasks with as little human assistance as possible.

Autonomous robots could catapult the productivity and quality of various human activities. Some applications are package delivery, cleaning, agriculture, surveillance, search & rescue, building, and transportation. However, a basic task that robots must do is to navigate in natural and human environments in order to achieve these applications. If someday we wish to have robots that build our highways, clean our streets, and grow and harvest our food, it is crucial they be able to navigate in unstructured environments.

The aim is to contribute to this goal: solving some problems of autonomous navigation in unstructured environments.

To successfully navigate autonomously, the robot must be capable of performing the following tasks:

Perception

Much like a human, a robot must be aware of its environment in order to successfully and efficiently perform any task. The human perceives information of the environment using its sensory organs. The robot is enabled with sensors to allow it to perceive this information.

A robot must interpret the data sent by sensors to recognize objects, places, and events that occur in the environment.

Simultaneous Localization and Map generation

A robot can make better-informed decisions if previously perceived information is stored and fused with freshly perceived data. The most appropriate example is a map of the environment. A map allows the robot to make appropriate decisions and avoid damage.

If we want to navigate through an environment, we need a map for two reasons:

1. To localize the robot
2. To plan a path between the robot's position and the goal position. But in unstructured, there are no maps that can be useful for navigation. Consequently, the robot must create its own map. In such cases, the map building process requires knowing the robot's position, and the robot's position estimation requires a map. So we need to locate the robot and generate the map at the same time.

This is why the map-building problem is so interesting and also problematic: errors in position and map estimation affect each other exponentially, producing inconsistent maps. The effectiveness and efficiency of navigation depend on having a map without large errors. Most map building methods are based on the Gaussian assumption (i.e. estimation errors are described by a Gaussian probability distribution). But this assumption is not entirely accurate because the real world error distributions are not Gaussian.

SLAM will be discussed in detail in later sections.

Planning

Motion planning decides what and how to move. It has the purpose of generating a trajectory (path) in state space to reach the goal, given an environment model (map), the current state of the robot, and a goal state. It must avoid the obstacles in the environment and must take into account the kinematic and dynamic constraints of the robot and its size. Motion planning is a difficult task because the algorithms should search for a solution on a continuous and high-dimensional state space. They must discretize the state space to make decisions as soon as possible.

In these types of environments, the problem of partial observability is important because planning has to make certain assumptions about the environment that cannot be or has not been observed. And these assumptions might be wrong. Even if we have a perfect mapping method, the map might have errors relative to the real environment. Why is this possible? The reason is simple: environments are dynamic

and all sensors have a limited range of observation. This type of problem is important to solve because it affects the optimality of the path length and completeness of navigation task (i.e. to be able to reach the goal when a solution exists).
A few planning algorithms are discussed in later sections.

Control
Ensures the planned movements are executed, despite unexpected disturbances.

Obstacle avoidance
Avoids crashing into moving objects, such as people, animals, doors, furniture or other robots that are not on the map.

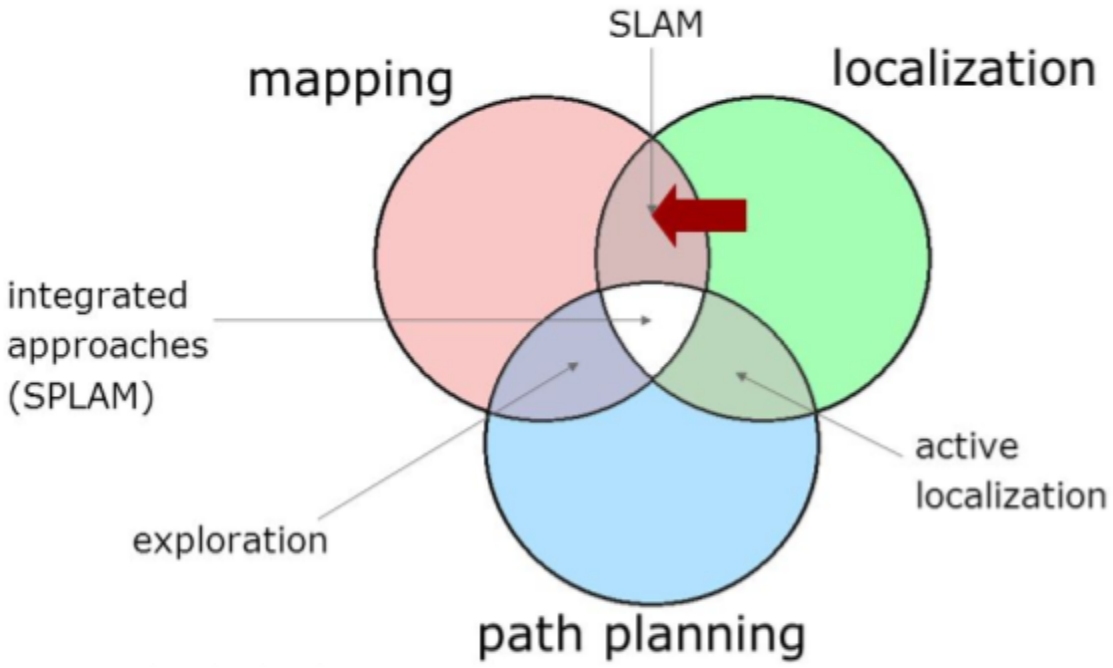


Fig 5.1

5.2 Simultaneous Localization and Mapping

The objective of simultaneous localization and mapping (SLAM) is to build a map and to locate the robot in that map at the same time. For the SLAM problem, it does not matter if the robot moves autonomously or is controlled by a human. The important thing is to build the map and locate the robot correctly.

The most basic way to locate a robot is using odometry. For a terrestrial mobile robot, odometry consists of integrating the displacements measured by encoders to estimate the position and orientation of the robot. The problem is that odometer error accumulates as the robot moves. Eventually, the error is so large that odometry no longer gives a good estimate of the state of the robot. Then the robot must make observations of some references in the environment to correct the odometer error, assuming the references are static. When the robot returns to observe, these references can reduce the accumulated error. The main work of the SLAM method is to correct the estimation of the robot state and the map. On the other hand, the SLAM is a chicken-and-egg problem because the robot needs a map to locate itself and the map needs the localization of the robot to build a consistent map. Thus SLAM methods must be recursive. This is why the SLAM problem is so difficult because errors in robot and map states affect each other, producing inconsistent maps. In the following section, we will give an overview of several SLAM methods.

5.2.1 Methods to perform SLAM

In the last decade, many types of SLAM have been developed. They can be categorized by different criteria, such as state estimation techniques, type of map, real-time performance, the sensors, etc. However, we classify the techniques according to the type of problem they solve into the following three categories: Feature-based SLAM, Pose-based SLAM, and Appearance-based SLAM.

Feature-based SLAM

Odometer error can be corrected by the use of landmarks in the environment as references. There are three tasks that any feature-based SLAM method must do. (1) Landmark detection. The robot must recognize some specific objects in the environment; they are called landmarks. It is common to use a laser range finder or cameras to recognize landmarks, such as corners, lines, trees, etc. (2) Data association. Detected landmarks should be associated with the landmarks on the map. Because landmarks are not distinguishable, the association may be wrong, causing large errors on the map. Besides, the number of possible associations can grow exponentially over time; therefore, data association is a difficult task. (3) State estimation. It takes observations and odometry to reduce errors. The convergence, accuracy, and consistency of the state estimation are the most important properties

Pose-based SLAM

This method estimates only the robot's state. It is easier than feature-based SLAM because the landmark positions are not estimated. However, it must maintain the robot path, and it uses the landmarks to extract metric constraints to compensate for the odometer error. Therefore, the high dimensionality limitation arises from the dimension growing by robot states rather than by the landmark states. Landmark detection, data association, loop closure, and dynamic environment still are relevant problems. Most pose-based SLAMs employ a laser range finder and the laser scans that form the occupancy grid maps of the environment (if the odometry is corrected, the grid map is right). Instead of using the landmark detector, the laser scan transforms the data association problem as a matching of laser scans for extracting the constraints. This problem is also called the front-end problem and is typically hard due to potential ambiguities or symmetries in the environment. Path estimation can be performed by optimization techniques, information filters, and particle filters.

Appearance-based SLAM

This method does not use metric information. The robot path is not tracked in a metric sense; instead, it estimates a topological map of places. Topological maps are useful for global motion planning, but not for obstacles avoidance. The configurations of the landmarks of a given place can be used to recognize the place; visual images or spatial information are also utilized to recognize the place. The metric estimation problem is avoided, but data association is still an important problem to solve. Loop closure is performed in the topological space. High dimensionality is translated into the number of places in the topological map. It is very common that these appearance techniques are used complementary to any metric SLAM method to detect loop closures.

5.2.2 Motion Planning

The motion planning method must generate a state trajectory, avoiding damage to the robot and reaching the goal state.

The motion planning problem can be stated as follows: given an environment model (an occupancy grid map, in our case), an initial state x_1 , and a goal state x_G, the planning algorithm must calculate automatically a sequence of states x_1, x_2, $x_{3,}$, …, x_G which conforms to a collision-free trajectory, with minimal length, and satisfies the kinematic and dynamic constraints of the robot.

In general, there are two kinds of planners: Graph Search, where the trajectory is searching in a graph of nodes that discreetly represent the state space of the robot; and Controllers, where instead of obtaining a path, they determine the control policy for every location in the state space. We briefly review the most important algorithms for each category.

Graph Search algorithms

These algorithms discretize the state space by creating a graph with nodes that represent robot states and edges that represent the robot actions. The graph formation can be simple, like dividing the physical space into a grid, or more complicated, like randomly sampling the state space. The most common Graph Search algorithms are the following:

1. Breadth-First Search

   The nodes are explored in order of proximity. Proximity is defined as the shortest number of edge transitions. The generated path could be the minimum-cost path (i.e., it is the optimal path) if the cost of edges is a nondecreasing function of the depth of the node. Its large disadvantage is that the memory requirement is bigger as the depth of the graph increases.

2. Depth-First Search

   In contrast to the previous algorithm, this explores each node up to the deepest level of the graph. The advantage is that its memory requirement is less than Breadth-First Search because it stores only a single path during execution. However, this search does not guarantee to find the optimal solution.

3. A* search

   It is an improvement of Dijkstra's algorithm, adding a heuristic function that encodes knowledge about the cost of reaching the goal. The optimal solution is guaranteed, if the heuristic is always an underestimated the cost. This heuristic reduces the number of node explorations with respect to Breadth- and Depth-First. Its main drawback is the high memory requirement because it keeps all generated nodes in memory. Some ways to overcome this difficulty is by applying an Iterative Deepening strategy (IDA*), Recursive Best-First Search (RBFS), or Memory Bounded A* (MA* or simplified MA*).

5.2.3 Achieving these Steps using ROS

Introduction to Navstack

The Navigation Stack is fairly simple on a conceptual level. It takes in information from odometry and sensor streams and outputs velocity commands to send to a mobile base. Use of the Navigation Stack on an arbitrary robot, however, is a bit more complicated. As a prerequisite for navigation stack use, the robot must be running ROS, have a tf transform tree in place, and publish sensor data using the correct ROS Message types. Also, the Navigation Stack needs to be configured for the shape and dynamics of a robot to perform at a high level.

While the Navigation Stack is designed to be as general purpose as possible, there are three main hardware requirements that restrict its use:

1. It is meant for both differential drive and holonomic wheeled robots only. It assumes that the mobile base is controlled by sending desired velocity commands to achieve in the form of: x velocity, y velocity, theta velocity.
2. It requires a planar laser mounted somewhere on the mobile base. This laser is used for map building and localization.
3. The Navigation Stack was developed on a square robot, so its performance will be best on robots that are nearly square or circular. It does work on robots of arbitrary shapes and sizes, but it may have difficulty with large rectangular robots in narrow spaces like doorways.

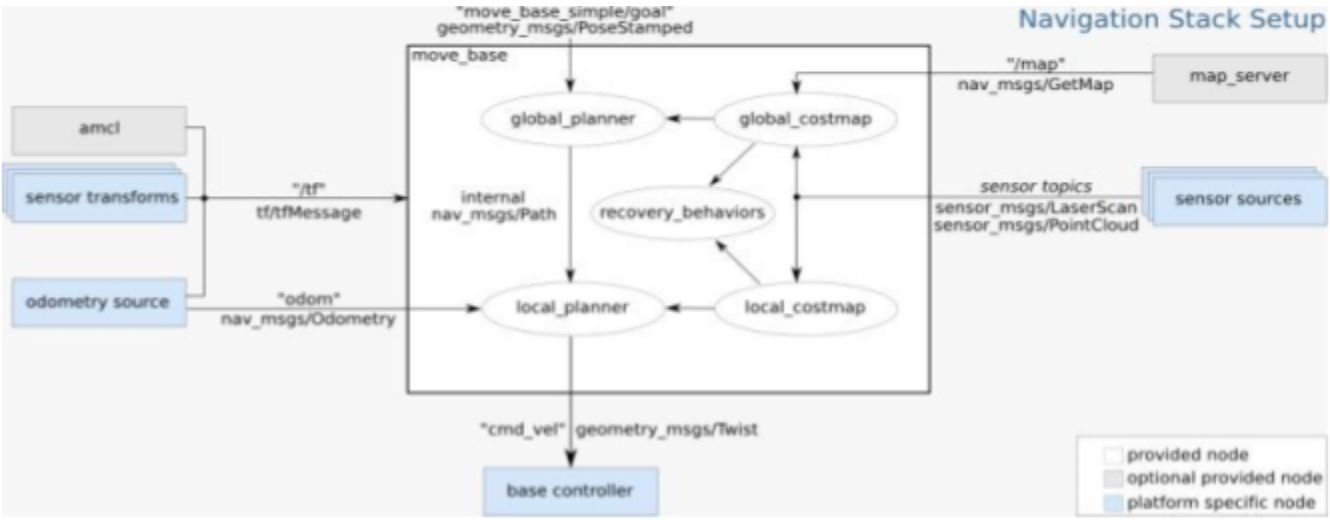


Fig 5.2

Terms used in implementing Autonomous Navigation

Map

A map is a representation of the environment where the robot is operating.It should contain enough information to accomplish a task of interest.

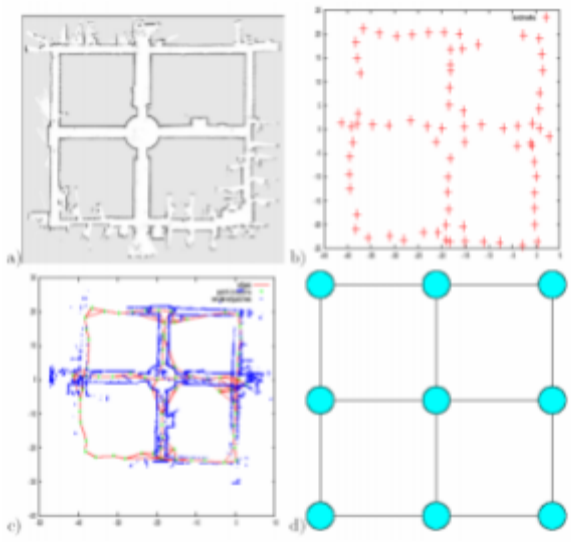


Fig 5.3

Representations:

1. Metric
    a. Grid Based
    b. Feature Based
    c. Hybrid
2. Topological
3. Hybrid

Robot Pose and Path

A metric map defines a reference frame.To operate in a map, a robot should know its position in that reference frame. A sequence of waypoints or of actions to reach a goal location in the map is a path.

Fig 5.4

Localization

Determine the current robot position, the measurements up to the current instant and a map.

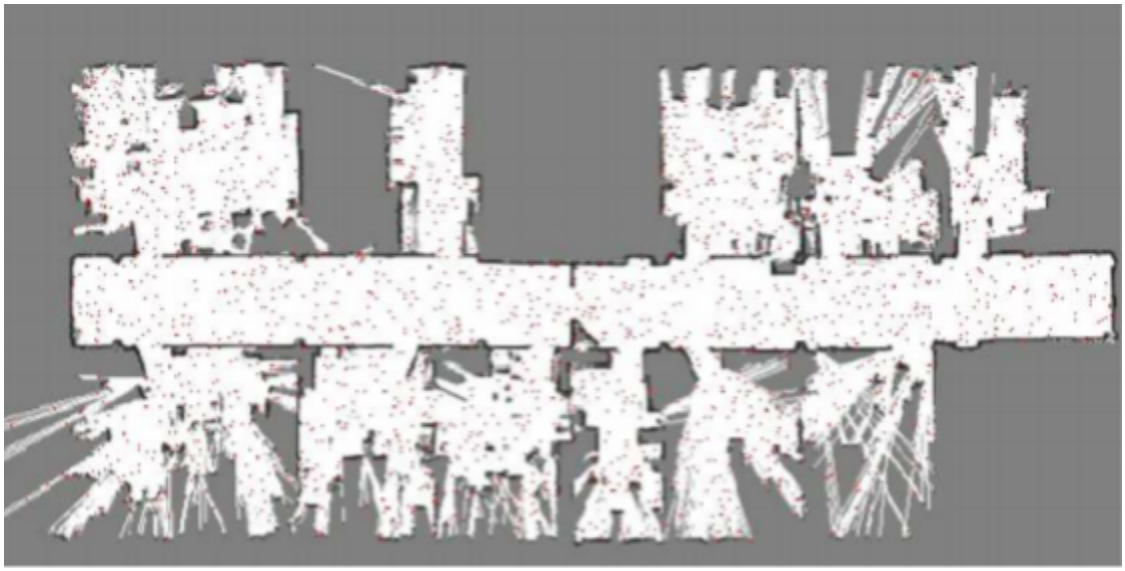


Fig 5.5

Path planning

Determine (if it exists) a path to reach a given goal location given a localized robot and a map of traversable regions.

Fig 5.6

Mapping

Given a robot that has a perfect ego - estimate of the position, and a sequence of measurements, determine the map of the environment. A perfect estimate of the robot pose is usually not available.

SLAM

SLAM= Simultaneous Localization and Mapping. It is used to estimate:

- the map of the environment
- the trajectory of a moving device using a sequence of sensor measurement

5.2.4 Setting up the parameters for our Robot

TF

Many ROS packages require the transform tree of a robot to be published using the tf software library. At an abstract level, a transform tree defines offsets in terms of both translation and rotation between different coordinate frames. To make this more concrete, consider the example of a simple robot that has a mobile base with a single laser mounted on top of it. In referring to the robot let's define two coordinate frames: one corresponding to the center point of the base of the robot and one for the center point of the laser that is mounted on top of the base

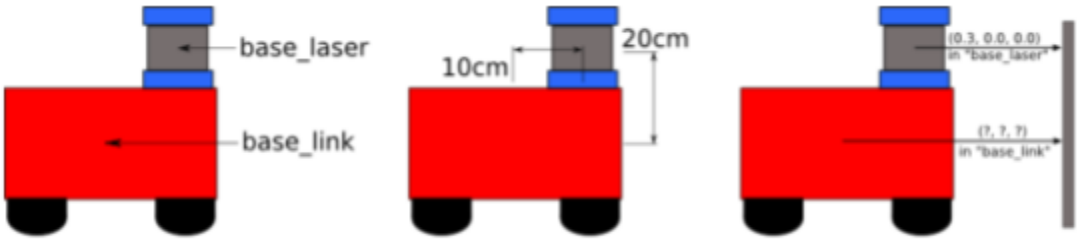
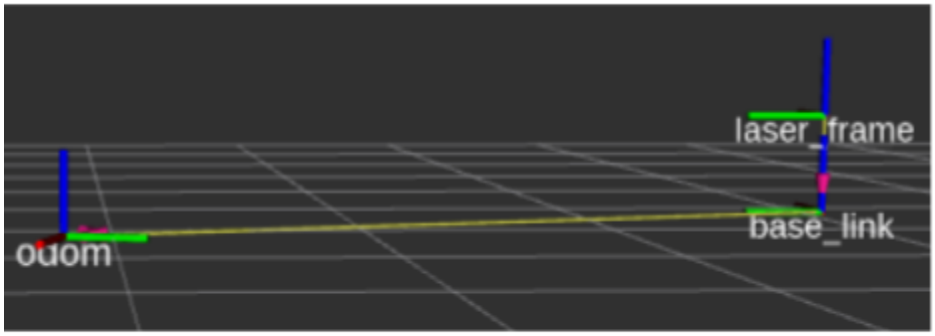


Fig 5.7

TF for AVITRA is continuously published using a rosnode named tf_publisher



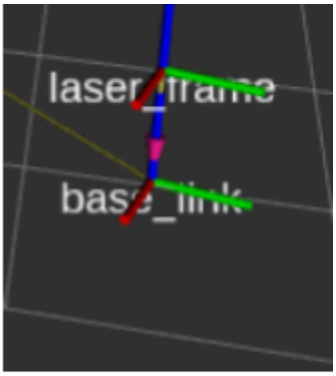


Fig 5.8

Navigation Tuning Guide

This includes setting up the odometry and range sensors. Avitra has encoded motors mounted on a differential driven base for getting odometry values and a sweep scanse lidar as range sensors.

Fig 5.9

Publishing sensor messages over ROS

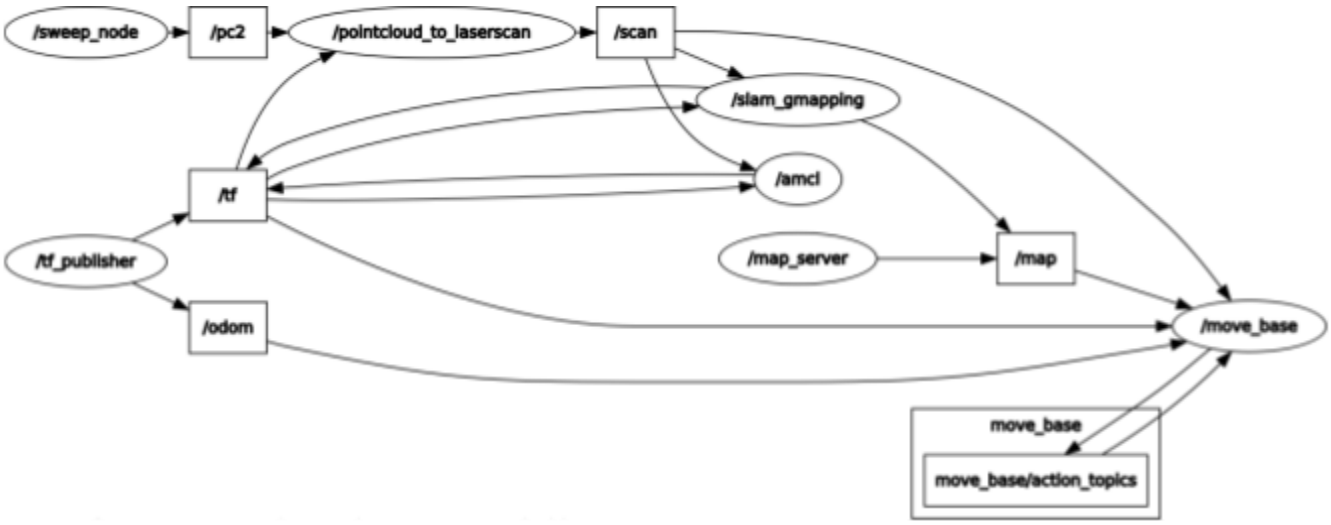


Fig 5.10

5.2.5 Mapping In ROS

We use ROS-SLAM-GMAPPING, which makes use of particle filter for estimating robot trajectories.

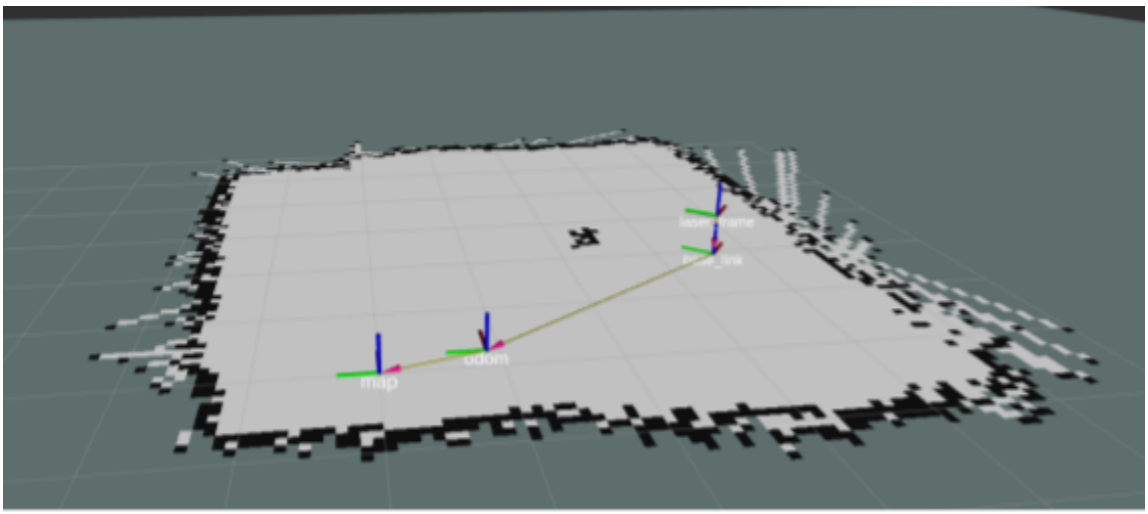


Fig 5.11

To build a map we-

1. Record a bag with data on topics /odom, /scan/ and /tf while driving the robot around in the environment(/teleop) it is going to operate in.
2. Then play the bag and the slam-gmapping-node, which generates the map of the environment using particle filter on data from the encoder(/odom) and lidar(/scan).The map generated is published on a topic /map which is saved using map_server service.

The map is an occupancy map and it is represented as-

1. An image showing the blueprint of the environment
2. A configuration file (yaml) that gives meta information about the map (origin, size of a pixel in real world)

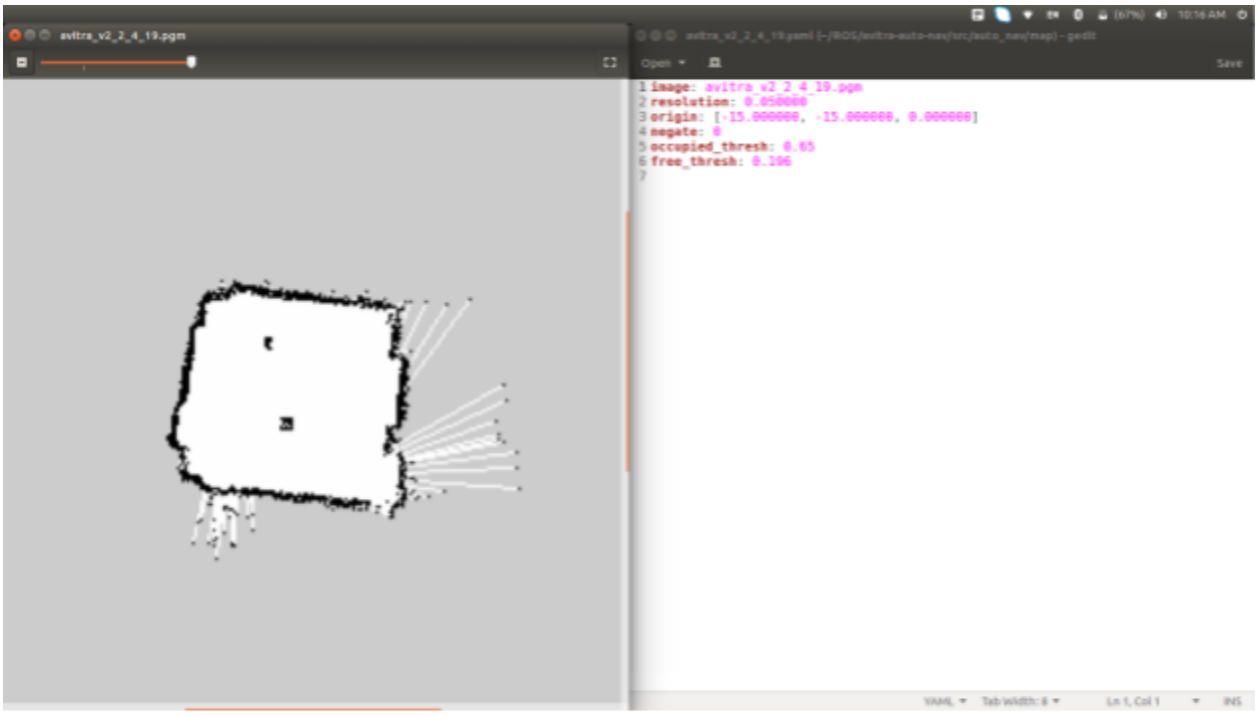


Fig 5.12

5.2.6 Localization In ROS

ROS implements the Adaptive Monte Carlo Localization algorithm. AMCL uses a particle filter to track the position of the robot. Each pose is represented by a particle.

Properties of particles are- Moved according to (relative) movement measured by the odometry

1. Suppressed/replicated based on how well the laser scan fits the map, given the position of the particle.
2. The localization is integrated in ROS by emitting a transform from a map-frame to the odom frame that “corrects” the odometry.

3. In order to localize
    a. The map saved in mapping is loaded.
    b. Now using a 2d pose estimate tool in rviz the estimate of the robot is given in on map.

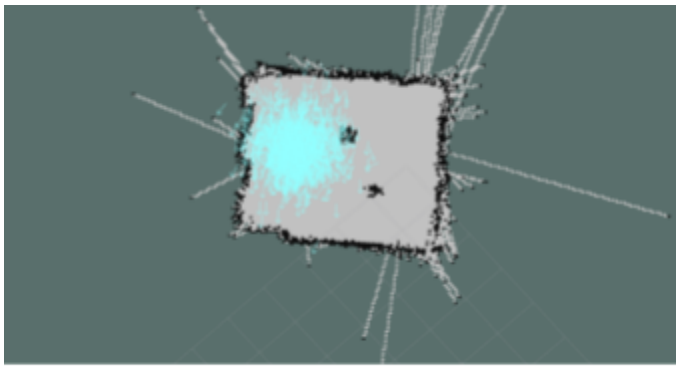


Fig 5.13

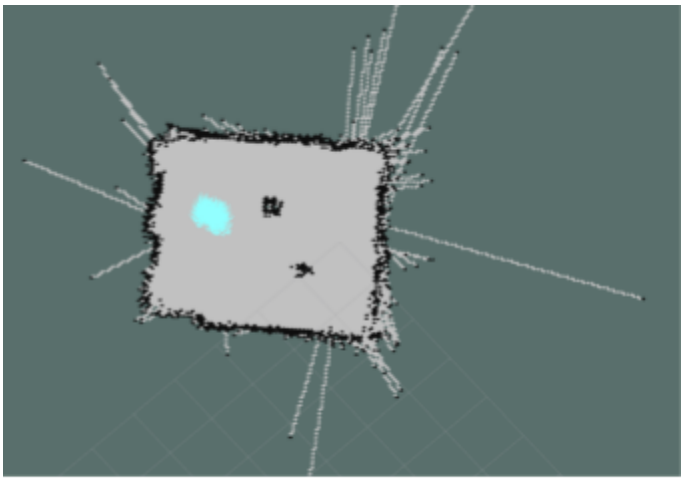


Fig 5.14

    c. Particles get evenly distributed about the initial pose estimate.
    d. Now using the teleoperation the bot is made to navigate in environment till the number of particles reduces and converges within the footprint of the robot.
    e. Once there are less than 100 particles left on the map we can say we are done with localization.

5.2.7 Autonomous Navigation in ROS

The velocities (linear and angular) are computed and the corresponding PWM signal is sent to drive the robot.

The robot while traversing to goal generates a local costmap in order to calculate a local path to its immediate next point to the goal.If obstacles are dynamically added in map they are considered in the local costmap but are not added in actual static map .
But the path to destination is made considering the obstacles.

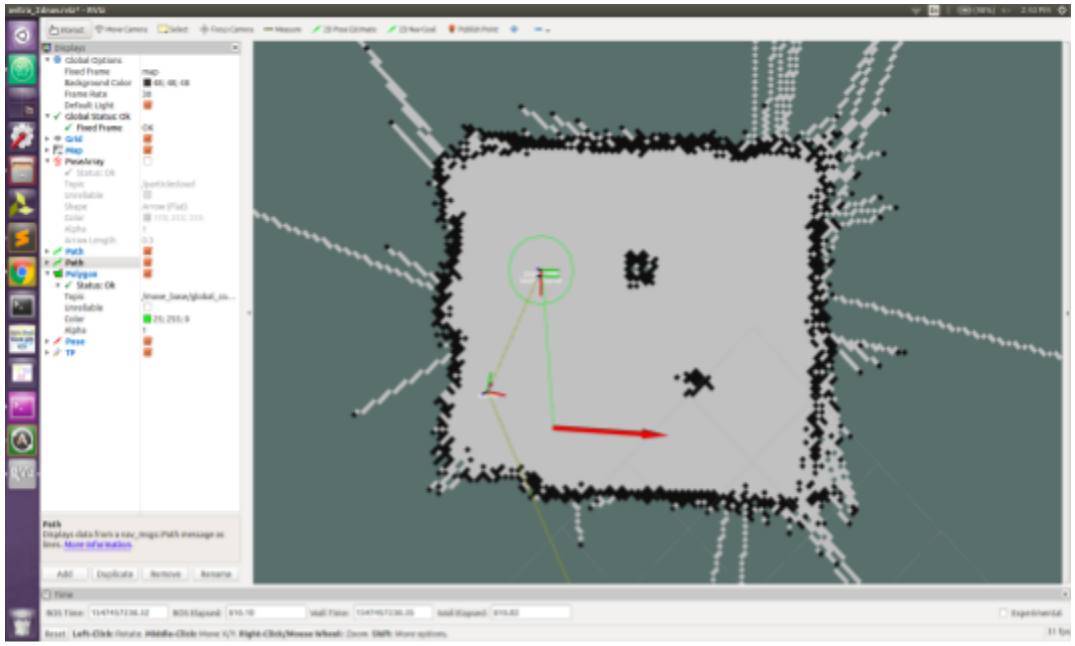


Fig 5.15

6. Future Scope and Conclusion

6.1 Test Scenario

Navigate, pick and place an object autonomously!

To test the robotic platform, we prepared a test environment akin to the real-life situation that we envisaged while designing the robot. An object is to be picked up from a region detrimental to the health of humans. The robot navigates through the unknown location, negotiating through the obstacles in the non-linear path. The robot provides live feed to the operator throughout the mission. A 2D map of the traversed path are simultaneously formed. The manipulator is controlled using the processing done by Kinect. Once the object is picked and the bot has left the room, the objective is achieved. Such scenarios are common in industrial sites where it is not advisable for humans to go.

6.2 Probable Use-cases

Besides being used as a disaster mitigation robot, the robotic platform can serve the following applications:

1. Serve as a testbench for further research into autonomous navigation.
2. Package delivery system.
3. Autonomous garbage collection

6.3 Future Scope

While a significant amount of time and effort has been invested in learning and implementing the various structural and functional aspects implemented in this robotic platform, we believe that the platform is capable of some more additions which shall aid to achieve the desired use cases. Some of the ideas that can be implemented are enlisted below.

6.3.1 Dexterous Gripper

The current robot has a two finger two phalanx design. To facilitate gripping operations, we propose a three finger three phalanxed dexterous gripper. The design of the gripper is a challenge because the dimensions and weight of the gripper should be such that the torque acting on the shoulder joint should not exceed the rated torque. But this ensures great efficacy and reliability of the gripping mechanism.

Fig 6.1

6.3.2 Artificial Intelligence driven autonomous navigation

Currently, the system is driven based on a pre mapped location. But in case of disaster or an unknown location we need a solution to be adaptable. For this purpose we can add artificial intelligence to make the system adaptable to unknown locations.

6.4 Conclusion

We realized that there are many tasks which are deleterious for humans to do. Keeping this issue in mind we brainstormed on possible solutions and came up with the idea of autonomous mobile manipulators. The solution would enable us to autonomously achieve tasks of pick and place and opening and closing valves. The manipulator is able to draw a straight line on a whiteboard autonomously. On working for this project we have learnt a plethora of new technologies that have helped in our overall development as engineers and young innovators. This project is an amalgamation of computing, mathematical modelling and mechanical design innovations. The wheelbase and the wheels are chosen in such a way that the robot has maximum mobility. Taking such decisions has helped us streamline our design thinking process. Human mimicry is the ultimate application of robotics and we aim to achieve a respectable leap in that direction. In realizing our vision we are introduced to many new fields of engineering.

REFERENCES

1. Yoshio Yamamoto, Xiaoping Yun, ”Coordinating Locomotion and Manipulation of a Mobile Manipulator”, University of Pennsylvania Department of Computer and Information Science Technical Report No. MS-CIS-92-18,March 1992.
2. Vedran Vajnberger, Tarik Terzimehić, Semir Silajdžić, Nedim Osmić, “Remote control of robot arm with five DOF”, 2011 Proceedings of the 34th International Convention MIPRO, 29 July 2011.
3. Lars Petersson, David Austin, Danica Kragic, “High-level Control of a Mobile Manipulator for Door Opening”, International Conference on Intelligent Robots and Systems, April 10, 2000.
4. Ramish, Syed Baqar Hussain, Farah Kanwal,”Design of a 3 DoF robotic arm”,2016 Sixth International Conference on Innovative Computing Technology (INTECH), 09 February 2017.
5. Nguyen, Viet; Harati, Ahad; Siegwart, Rol and, “A Lightweight SLAM algorithm using Orthogonal Planes for Indoor Mobile Robotics”, Zurich Research Collection, 2007.
6. S. Dubowsky, E. E. Vance, “Planning mobile manipulator motions considering vehicle dynamic stability constraints”, The Society of Electrical and Electronics Engineers Inc.
7. Ayrton Oliver, Steven Kang, Burkhard C. Wünsche, Bruce MacDonald, ”Using the Kinect as a Navigation Sensor for Mobile Robotics”, November 26 - 28 2012.
8. Alif Ridzuan Khairuddin, Mohamad Shukor Talib, Habibollah Haron, “Review on simultaneous localization and mapping (SLAM)”, 2015 IEEE International Conference on Control System, Computing and Engineering (ICCSCE), 02 June 2016.
9. Giorgio Grisetti, Rainer Kummerle, Cyrill Stachniss, Wolfram Burgard, “A Tutorial on Graph-Based SLAM”, IEEE Intelligent Transportation Systems Magazine (Volume: 2, Issue: 4, winter 2010)
10. Rodrigo Munguia, Antoni Grau, “Monocular SLAM for Visual Odometry”, IEEE International Symposium on Intelligent Signal Processing,08 February 2008.
11. Luigi D'Alfonso, Andrea Griffo, Pietro Muraca, Paolo Pugliese, “A SLAM algorithm for indoor mobile robot localization using an Extended Kalman filter and a segment based environment mapping”,2013 16th International Conference on Advanced Robotics (ICAR),13 March 2014.
12. https://leonardoaraujosantos.gitbooks.io/artificial-inteligence/content/
13. Making Things See by Greg Borenstein Published by O’Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.
14. A. Hornung,. K.M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard, "OctoMap: An Efficient Probabilistic 3D Mapping Framework Based on Octrees" in Autonomous Robots, 2013; DOI: 10.1007/s10514-012-9321-0. Software available at http://octomap.github.com.
15. Mastering ROS for Robotics Programming - By Lentin Joseph
16. Introduction to Robotics - John Craig
17. The MoveIt Documentation : https://moveit.ros.org/documentation/

18. Safdar Zaman, Wolfgang Slany, Gerald Steinbauer Institute for Software Technology Graz University of Technology, Austria ”ROS-based Mapping, Localization and Autonomous Navigation using a Pioneer 3-DX Robot and their Relevant Issues” - https://www.researchgate.net/profile/Safdar_Zaman2/publication/224242103_ROS-based_mapping_localization_and_autonomous_navigation_using_a_Pioneer_3-DX_robot_and_their_relevant_issues/links/00b49520b2fc4b0945000000/ROS-based-mapping-localization-and-autonomous-navigation-using-a-Pioneer-3-DX-robot-and-their-relevant-issues.pdf
19. Dereck Wonnacott ,Matias Karhumaa James and Walker “AUTONOMOUS NAVIGATION PLANNING WITH ROS” - http://pages.mtu.edu/~jwwalker/files/cs5881.pdf
20. Mohan Kumar N, Shreekanth T Department of Electronics and Communication Engineering Shri Jayachamarajendra College of Engineering Mysuru, India “Autonomous Robot Based on Robot Operating System (ROS) for Mapping and Navigation ” - https://pdfs.semanticscholar.org/4c7e/bbfc044545cbfd38c3195093cc402f6181c9.pdf