h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbeit
FACHBEREICH ELEKTROTECHNIK
UND INFORMATIONSTECHNIK

# MASTER THESIS

# Optimization of the localization and control of a load-bearing omnidirectional platform

**Submitted by:** **Ankit Ankit**

**1st Academic Supervisor:** **Prof. Dr. Wolfgang Jürgen Weber**

**2nd Academic Supervisor:** **Prof. Dr.-Ing. Alexandra Weigl-Seitz**

**Industrial Supervisor:** **Dipl. Inf.Thomas Rühr**

**Completion Date:** **04th October 2019**

**Student:**

Ankit
First (Given) Name

Ankit
Last (Family) Name

Date of Birth: 18/12/1991

Matr.-No.: 757593

1st Academic Supervisor: Prof. Dr. Wolfgang Jürgen Weber

2nd Academic Supervisor: Prof. Dr.-Ing. Alexandra Weigl-Seitz

Title: Optimization of the localization and control of a load-bearing omnidirectional platform

Abstract:

This thesis report describes the optimization of the localization and control of a load-bearing omnidirectional platform. Firstly, the theoretical backgrounds are analyzed to provide an understanding of the Robot Operating System, Point Cloud Library, path planning and mobile robot kinematics. Then, it is described, how the implementation is performed to achieve the given objectives. An open-source robotics software, known as Robot Operating System (ROS) has been used for implementation. A motion control algorithm is designed to allow the mobile robot to reach the goal point. A controlled motion of the platform has been implemented to overcome the 180° reorientations of the caster wheels. Laser filters are designed for improving the precision and accuracy of the laser data. Moreover, a docking application for the mobile platform, to perform docking under a trolley cart, is carried out using the laser data. During this master thesis, I have gained knowledge about the Robot Operating System, mobile robotics, control of an omnidirectional drive, odometry and control, global laser localization using a particle filter and tracking of objects with laser sensor data in a ROS environment.

In partial fulfilment of the requirements of the **University of Applied Sciences Hochschule Darmstadt (h_da)** for the degree **Master of Science in Electrical Engineering** carried out in collaboration with **Industrial Enterprise**

Company: KUKA Deustchland GmbH

Address: Zugspitzstraße 140
86165 Augsburg, Deutschland

This Master Thesis is subject to a non-disclosure agreement between the University of Applied Sciences Hochschule Darmstadt (h_da) and the industrial partner.

(Signature)

1st Academic Supervisor:

Student:

Ankit
First (Given) Name

Ankit
Last (Family) Name

1st Academic Supervisor: Prof. Dr. Wolfgang Jürgen Weber

2nd Academic Supervisor: Prof. Dr.-Ing. Alexandra Weigl-Seitz

## Declaration

I hereby declare that this thesis is a presentation of my original research work and that no other sources were used other than what is cited.
I furthermore declare that wherever contributions of others are involved, this contribution is indicated, clearly acknowledged and due reference is given to the author and source.
I also certify that all content without reference or citation contained in this thesis is original work.
I acknowledge that any misappropriation of the previous declarations can be considered a case of academic fraud.

Darmstadt, 04th October 2019
(Date)

(Signature)

## Acknowledgment

I would like to thank my supervisor, Dipl. Inf. Thomas Rühr, for letting me be part of this project, as it was a very exciting and rewarding one. I appreciate his counsel, opinions and never-failing positive attitude. I would also like to thank Mr. Sebastian Jablonski for his support and engagement in the hardware setup. Further, I would like to thank my colleague Mr. Till Schulte Rebbelmund for providing the 3D printed cylindrical casting to be placed around the corner of the cart. This helped in better shape recognition of the object using laser sensor data.

I would like to thank Prof. Dr. Wolfgang Jürgen Weber for the valuable inputs, discussions and supervision during the entire tenure of this master thesis report.

I would also like to express my gratitude to all my colleagues at the Department of Corporate Research, for providing a relaxed as well as an inspiring environment. It was fun to have fruitful discussions during the coffee breaks. Finally, I would like to thank my family for always being there for their consistent support and motivation.

## Table of Contents

Abstract ........ 2

1. Directories ........ 7
   1.1. List of Abbreviations ........ 7
   1.2. List of figures ........ 8
   1.3. List of tables ........ 10
   1.4. Motivation ........ 11
2. Introduction ........ 12
   2.1. About the company ........ 14
3. Hardware and Software Modules Overview ........ 15
   3.1. Hardware Modules: ........ 15
   3.2. Software Modules: ........ 16
   3.3. Safety Requirements ........ 17
4. Background and Literature Survey ........ 18
   4.1. Mecanum Wheels ........ 18
   4.2. Caster Wheels ........ 19
   4.3. Omnidirectional Platform Configuration ........ 20
   4.4. Control of Omnidirectional Platform ........ 21
5. General Control scheme of the Mobile Robot System ........ 25
   5.1. Perception ........ 26
   5.2. Localization ........ 26
   5.3. Path Planning ........ 28
   5.4. Motion Control ........ 29
6. Introduction to Open Source Robotic Software ........ 30
   6.1. ROS Core ........ 32
   6.2. ROS Master ........ 33
   6.3. ROS Nodes ........ 34
   6.4. ROS Topic ........ 34
   6.5. ROS Messages ........ 34

6.6. ROS Parameter Server ..... 34
6.7. ROS Service ..... 34
6.8. ROS Action ..... 34
6.9. ROS Transformation Frames ..... 35
6.10. ROS Rviz ..... 35
6.11. ROS Gazebo ..... 35
6.12. ROS Plugins ..... 36
7. Creation of Environment Map ..... 37
8. Monte Carlo Localization ..... 38
8.1. Adaptive Monte Carlo Localization ..... 39
8.1.1. Subscribed Topics ..... 40
8.1.2. Published Topics ..... 40
8.1.3. Services ..... 40
8.1.4. Filter Parameters ..... 40
8.1.5. Laser Parameters ..... 40
8.1.6. Odometry Parameters ..... 41
8.1.7. AMCL Transforms ..... 41
8.2. RVIZ 2D Manual Pose Estimate ..... 44
9. Non-linear Behavior of Caster Wheels ..... 45
9.1. Motivation behind Endpoint Augmentation ..... 45
9.2. Ideas for compensating behavior: ..... 46
10. Realization of Spline Motion ..... 47
11. Laser Scan ..... 53
11.1. Implementation of the Laser Filter ..... 53
11.1.1. Laser Median Filter ..... 53
11.1.2. Laser Scan Shadow Filter ..... 55
11.1.3. Laser Scan Intensity Filter ..... 56
12. Line Extraction Algorithm using Laser ..... 57
12.1. Split-and-Merge Algorithm ..... 57

12.2. Hough Transform ....57
12.3. Line model with Expectation-Maximization ....57
12.4. RANSAC ....58
13. Implementation of Geometric Shapes Recognition Using Laser ....59
13.1. Clustering of Laser Point Cloud ....59
13.2. Splitting of Clusters ....60
13.3. Segmentation ....62
13.4. Merging of Segments ....62
13.5. Circle Extraction ....63
13.6. User-Defined Messages ....64
14. ROS Navigation Stack ....64
14.1. Cost Map ....66
14.2. Local Planner ....68
14.3. Global Planner ....68
14.4. Recovery Behavior ....68
14.5. Base Controller ....68
15. Implementation of Autonomous Platform Docking ....69
16. Experiment and Evaluation ....73
16.1. Test for Spline Motion Interpolation ....73
16.2. Test for Base Controller ....75
16.3. Test for Object Shape Recognition ....76
16.4. Test for Autonomous Docking Application ....77
17. Summary and Conclusion ....78
18. References ....79
19. Appendix ....81
19.1. Nomenclature ....81

# 1. Directories

## 1.1. List of Abbreviations

| Abbreviation | Meaning |
|---|---|
| TCP | Tool Center Point. |
| ROS | Robot Operating System. |
| PCL | Point Cloud Library. |
| AMCL | Adaptive Monte Carlo Localization. |
| ICP | Iterative Closest Point. |
| TF | Transformation Frames. |
| MCL | Monte Carlo Localization. |
| URDF | Unified Robot Description Format. |
| HT | Hough transforms. |
| RANSAC | Random Sample Consensus. |
| API | Application Programming Interface. |
| EM | Expectation-Maximization. |
| AGV | Automatic Guided Vehicle. |
| SLAM | Simultaneous Localization and Mapping. |
| SPA | Sparse Pose Adjustment. |

## 1.2. List of figures

Figure 2.1: REFILLS Project Overview....13
Figure 3.1: Virtual model of the omnidirectional platform....15
Figure 3.2: Mobile Platform, Scanning Unit and Trolley cart....16
Figure 3.3: Safety Remote Switch (Tyroremotes, 2000)....17
Figure 4.1: Mechanical structure of Mecanum wheel ( HENDZEL & RYKAŁA, 2017)....18
Figure 4.2: Caster wheel different angle views (SIEGWART & NOURBAKHSH, 2004)....19
Figure 4.3: Mobile platform with caster wheels (SIEGWART & NOURBAKHSH, 2004)....19
Figure 4.4: A few possible combinations for rotations of Mecanum wheels....20
Figure 4.5: Omnidirectional Platform kinematics....22
Figure 5.1: Control Scheme for Mobile Robot (SIEGWART & NOURBAKHSH, 2004)....25
Figure 5.2: Internal steps for Perception....26
Figure 5.3: Flowchart for mobile robot localization....27
Figure 5.4: Behavior-based navigation architecture....27
Figure 5.5: Map-based navigation architecture....28
Figure 5.6: Cascaded control of a motor....29
Figure 6.1: Layered Architecture with ROS as Middleware....30
Figure 6.2: ROS Communication Model....31
Figure 6.3: ROS Components....32
Figure 6.4: ROS Architecture with ROS Master and ROS Nodes....33
Figure 6.5: ROS Action Client-Server Interface (ROS.org, 2007)....35
Figure 7.1: Map Recorded with Laser Sensor....37
Figure 8.1: AMCL Map Localization (ROS.org, 2007)....41
Figure 8.2: AMCL depiction using RVIZ before localization....43
Figure 8.3: AMCL depiction using RVIZ after localization....44
Figure 9.1: Endpoint Augmentation....46
Figure 10.1: Interpolation Overview....47
Figure 10.2: Spline Visualization in RVIZ....52
Figure 11.1: Actual laser scan data without the median filter (visualization in RVIZ)....54
Figure 11.2: Laser scan with the median filter....55
Figure 13.1: Grouping of the point cloud (Przybyła, 2017)....60
Figure 13.2: Splitting procedure (Przybyła, 2017)....61
Figure 13.3: Iterative End Point Fit (Bresenham, 1962)....61
Figure 13.4: Depiction for neighboring and collinearity checks (Przybyła, 2017)....63
Figure 13.5: Circle Extraction (Przybyła, 2017)....64
Figure 14.1: Navigation Stack Nodes and Topics/Actions Overview (ROS ORG, 2007)....65
Figure 14.2: Nav_Core interface (ROS ORG, 2007)....66
Figure 14.3: 2D Costmap (ROS WIKI, 2007)....67

Figure 15.1: Flowchart for the docking application ....................................................................69
Figure 15.2: Circular trolley corners detection ............................................................................70
Figure 15.3: Docking with respect to dock_frame1 .....................................................................71
Figure 15.4: Docking with respect to dock_frame2 .....................................................................72
Figure 16.1: Monitoring the orientation and the position of the Mobile Robot base_link in the Map frame during the spline trajectory ...........................................................................................73
Figure 16.2: Deviation in the orientation and the position of the Mobile Robot base_link in the Map frame during the linear trajectory..........................................................................................74

## 1.3. List of tables

Table 9.1: Statistical data obtained from experiments without spline trajectory ......................... 45
Table 16.1: Statistical data obtained from experiments with spline trajectory ............................ 74
Table 16.2: Statistical data obtained from experiments of the position controller with large value of tolerance ........................................................................................................ 75
Table 16.3: Statistical data obtained from experiments of the position controller with the medium tolerance ........................................................................................................ 75
Table 16.4: Statistical data obtained from experiments of the position controller with low tolerance ........................................................................................................ 76
Table 16.5: Statistical data obtained from experiments for object shape recognition ................. 77
Table 16.6: Statistical data obtained from experiments for autonomous docking ....................... 77

## 1.4. Motivation

My passion for building innovative applications in Robotics was intensified during the lecture of Robotics in my master’s program. I was fortunate to get an opportunity to work at KUKA Deutschland GmbH, Augsburg as an intern for 19 weeks. After the internship, my eagerness to work and explore robotic applications increased. I was very delighted when I came to know that my master thesis task is for the same project in which I have worked during my internship.

Within the (H2020 REFILLS Research and Innovation Action, 2017) H2020 Project "Robotics Enabling Fully-Integrated Logistics Lines for Supermarkets — REFILLS" an omnidirectional small low-cost mobile platform was developed. The platform uses not only omnidirectional Mecanum wheels to drive but also has additional caster wheels (like the wheels of a shopping cart) to increase the maximum payload. The Mecanum wheels are suspended so that the contact to the ground is always maintained with similar pressure, independent of the load on the caster wheels. Ideally, the load-bearing caster wheels would act transparently, that is, they would not influence the motion of the platform in the plane but will help in increasing the maximum payload. This scenario is observed when the platform moves in a certain direction for a longer time and the casters have time to reorient themselves along with the motion direction.

However, there are some scenarios of motion where the casters influence the motion of the platform. Mostly, this happens when the mobile platform changes direction suddenly and the casters have to re-orient themselves to follow the new motion direction. The situation gets worse when the platform is first moved in one direction and then suddenly changes its orientation of motion by 180°. All 4 caster wheels will flip their orientation completely, but it is neither predictable nor observable whether the individual caster wheels would flip the direction clockwise or counterclockwise around their pivot axis. Since the load creates pressure on the casters, the friction on the casters that is orthogonal to their rolling direction will act as additional forces on the mobile platform and will slightly change its orientation and position.

# 2. Introduction

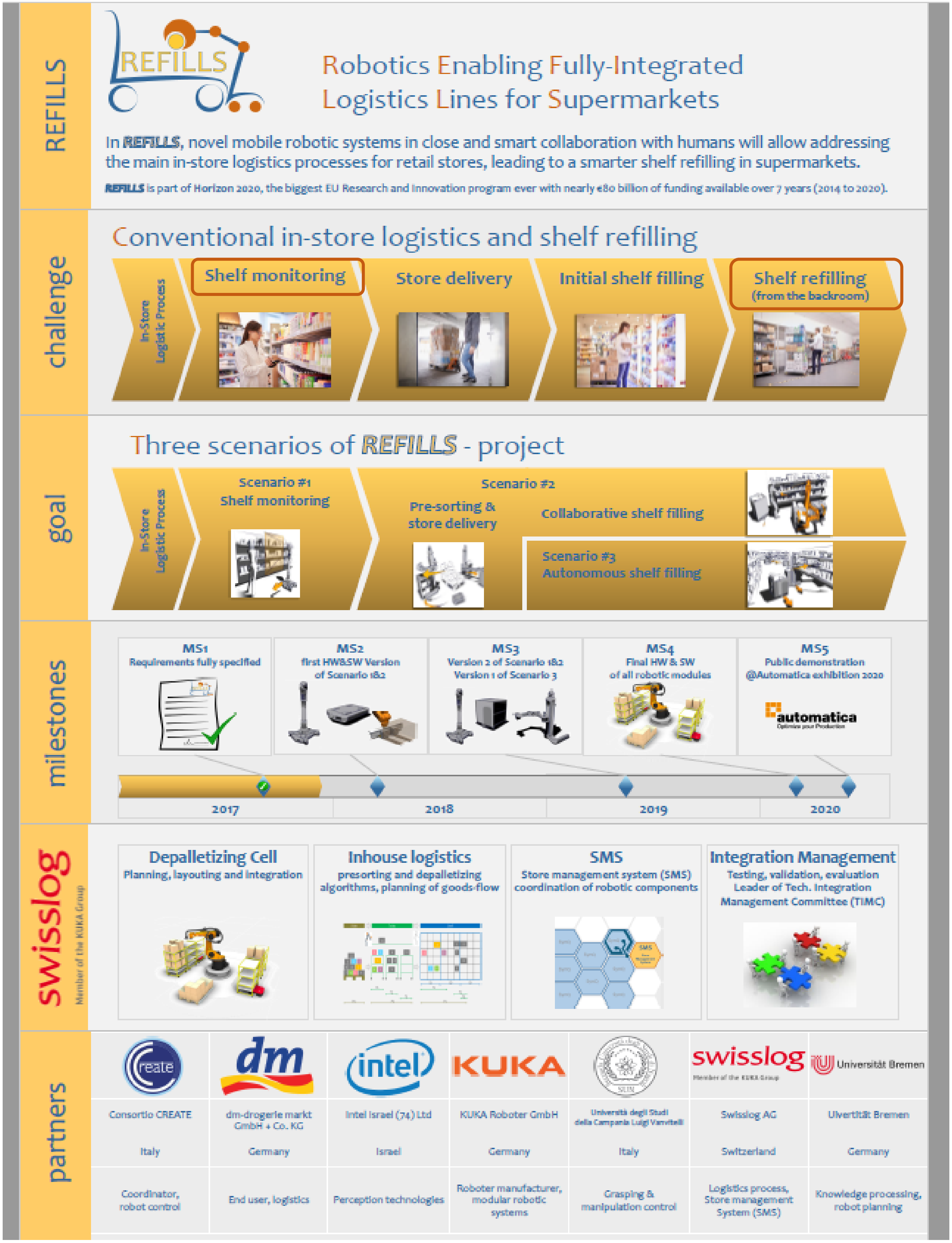
An Omnidirectional platform is a type of Automated Guided Vehicle (AGV). It is the kind of robot which is developed to carry out the task with ease. They are present for over more than 50 years. They lie under the category of the mobile robot. The omnidirectional platform is designed to carry out the task autonomously with a focus on transferring of commodities within the industry. Moreover, many commercial markets are utilizing omnidirectional platforms for lifting products and placing them at the desired position. Mobile robots are mostly suited for hazardous working environments and for environments where tasks are monotonous and boring along with an amount of the labor force requirement.

To meet the requirements of today's era of automation technology, the task performed by these omnidirectional platforms must be optimized and precise. This master thesis is focused on efficient localization of the omnidirectional platform and designing an application for docking the platform under a trolley cart. The entire implementation is based on the Robot Operating System (ROS). With reference to Figure 2.1, it is a part of the REFILLS project. This project includes autonomous refilling of the items in the stores. It involves different tasks such as shelf monitoring, store delivery, initial self-filling and self-refilling. This master thesis work will be used for shelf-scanning and shelf-refilling of the commodities.

Firstly, extensive research is carried out about all various available localization schemes for mobile robots. The behavior of the platform is studied for the case where it must return from the current position without changing its orientation. In this case, the analysis was done on the behavior of the caster wheel to make sure that the effect arising from the flipping of caster wheels can be nullified. This provides the solution for the consistent scanning of the products by the scanning unit, which is lifted by the omnidirectional platform.

Secondly, an algorithm for efficient docking has been developed. This involved extensive research on the various available algorithms for object detection using laser sensor data. Finally, the most effective methodology was selected and used for the implementation. This algorithm involves the usage of the laser sensor point cloud data and filters out the corners of the trolley. It uses the ROS navigation stack to move towards the goal point. This provides the solution to the automation transportation of the products loaded on the trolley cart.

The prerequisites for this thesis work require a good understanding of ROS and sound knowledge on mobile robotics.



REFILLS

Robotics Enabling Fully-Integrated Logistics Lines for Supermarkets

In **REFILLS**, novel mobile robotic systems in close and smart collaboration with humans will allow addressing the main in-store logistics processes for retail stores, leading to a smarter shelf refilling in supermarkets.

**REFILLS** is part of Horizon 2020, the biggest EU Research and Innovation program ever with nearly €80 billion of funding available over 7 years (2014 to 2020).

challenge

Conventional in-store logistics and shelf refilling

In-Store Logistic Process

Shelf monitoring

Store delivery

Initial shelf filling

Shelf refilling (from the backroom)

goal

Three scenarios of **REFILLS** - project

In-Store Logistic Process

Scenario #1
Shelf monitoring

Scenario #2
Pre-sorting & store delivery
Collaborative shelf filling

Scenario #3
Autonomous shelf filling

milestones

MS1
Requirements fully specified

MS2
first HW&SW Version of Scenario 1&2

MS3
Version 2 of Scenario 1&2
Version 1 of Scenario 3

MS4
Final HW & SW of all robotic modules

MS5
Public demonstration @Automatica exhibition 2020

2017 | 2018 | 2019 | 2020

swisslog
Member of the KUKA Group

Depalletizing Cell
Planning, layouting and integration

Inhouse logistics
presorting and depalletizing algorithms, planning of goods-flow

SMS
Store management system (SMS) coordination of robotic components

Integration Management
Testing, validation, evaluation
Leader of Tech. Integration Management Committee (TIMC)

partners

| | | | | | | |
|---|---|---|---|---|---|---|
| Consortio CREATE | dm-drogerie markt GmbH + Co. KG | Intel Israel (74) Ltd | KUKA Roboter GmbH | Università degli Studi della Campania Luigi Vanvitelli | Swisslog AG | Ulvertität Bremen |
| Italy | Germany | Israel | Germany | Italy | Switzerland | Germany |
| Coordinator, robot control | End user, logistics | Perception technologies | Roboter manufacturer, modular robotic systems | Grasping & manipulation control | Logistics process, Store management System (SMS) | Knowledge processing, robot planning |

Figure 2.1: REFILLS Project Overview

## 2.1. About the company

The corporate history of KUKA began in 1898 with Johann Joseph Keller and Jakob Knappich in Augsburg. For more than 100 years, KUKA has stood for ideas and innovations that have made it successful worldwide.

KUKA is one of the world’s leading suppliers of automation solutions. KUKA as a technology leader is known all over the world for maintaining high-quality standards. KUKA operates in four major divisions such as Division Industries, Division Automotive, Division Consumer Goods & Logistics Automation and Division Healthcare.

At KUKA Corporate Research and Development Department, currently, there are 40 employees. Every year around 50 students join the department as interns, working students or master thesis students to support the ongoing research and to gain knowledge in robotics-related domains. This department works in close collaboration with other research institutes. The main intention of this department is to work on new innovative ideas and to test use-cases with other ongoing robotics projects or applications.

KUKA stands for innovations in automation solutions for Industry 4.0.

# 3. Hardware and Software Modules Overview

This chapter provides an overview of the hardware and the software modules that have been used for the task in this master thesis. This chapter also illustrates the safety requirements to be followed while working in the lab with the robots.

## 3.1. Hardware Modules:

- Omnidirectional mobile platform.
- Laser sensor mounted on the mobile platform used for localization and object detection.
- Omnidirectional trolley cart with a cylindrical covering around the corners.
- Low-cost stepper motor driver with high torque.
- Self-developed USB-connected ultra-low-cost motor controllers.

The virtual model of the omnidirectional platform is been depicted in Figure 3.1. This model has a dimension of 788mmx480mmx140mm. Its design consideration includes a flat and small model and can bear a payload 100kgs.

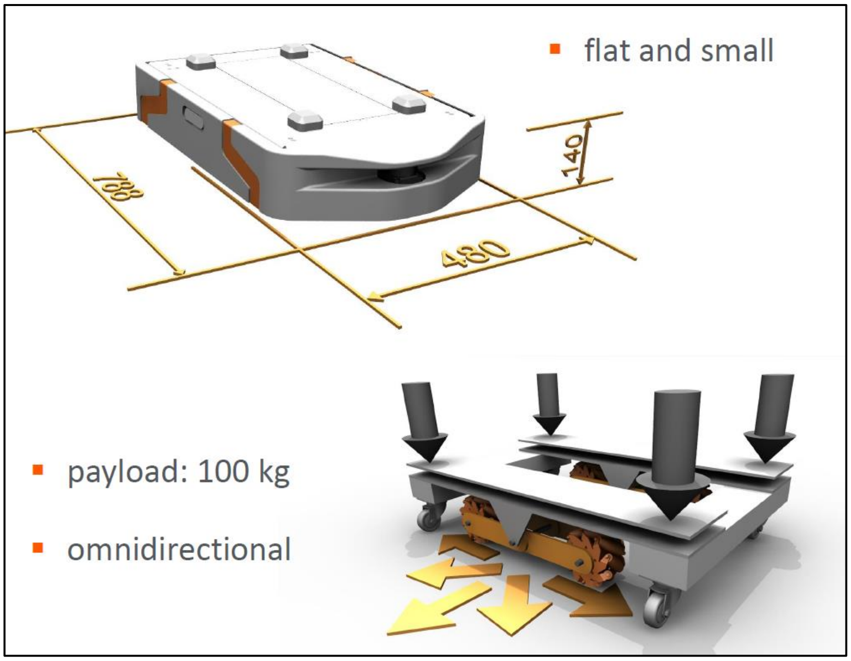


**Figure 3.1: Virtual model of the omnidirectional platform**

With reference to Figure 3.1, the wheels arrangements in the omnidirectional platform have been depicted. Mecanum wheels along the sides of the platform provide the mobility to the platform in all the direction. The passive caster wheels around the corner of the platform help in increasing its payload as the Mecanum wheels independently can't bear such high payload in the current application.

The real-world model of the omnidirectional platform and the trolley cart have been depicted in Figure 3.2. The omnidirectional platform has a laser mounter mounted at the front, which helps in the localization scheme and is also used for the docking application. Figure 3.2 also depicts the trolley along with the products. This trolley needs to be lifted by the platform so that the products can be delivered to the desired location.

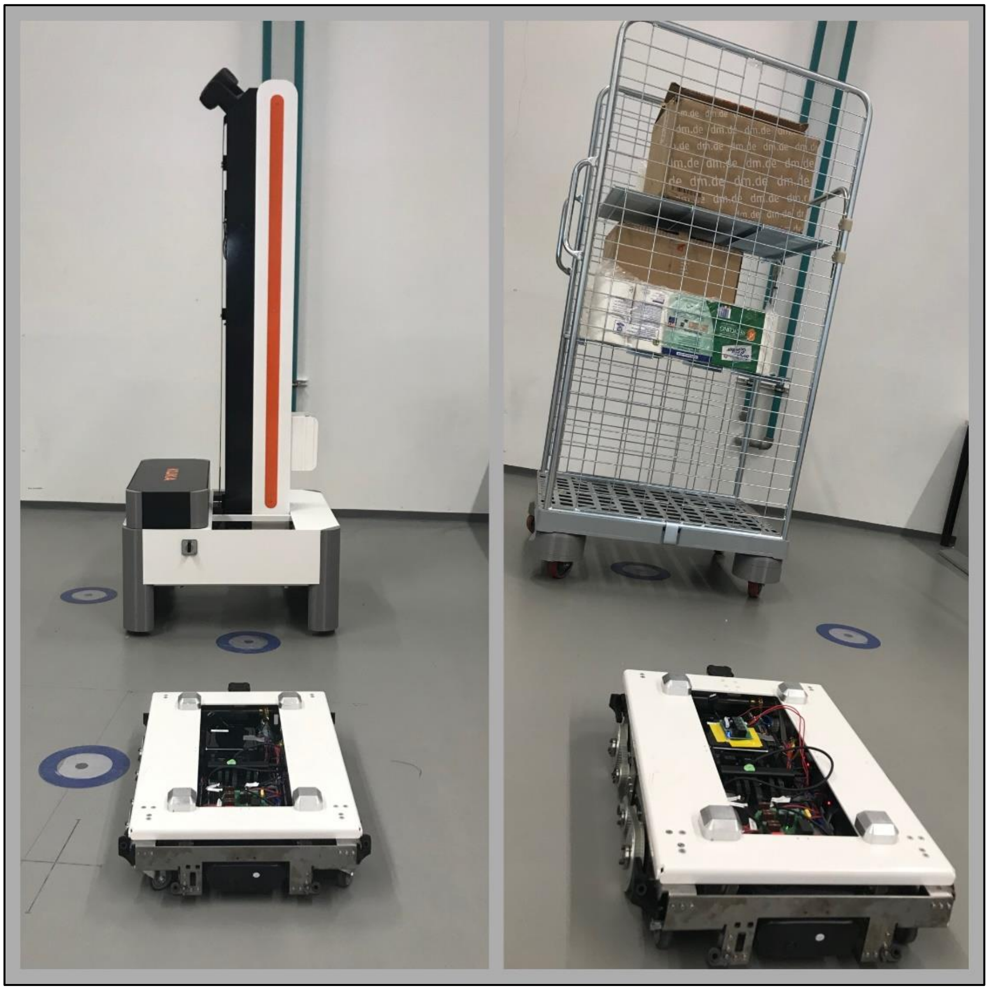


**Figure 3.2: Mobile Platform, Scanning Unit and Trolley cart**

## 3.2. Software Modules:

- Robot Operating System which acts as middleware between robot hardware and software.
- C++ Object-Oriented Language
- Point Cloud Library for 3D data processing

## 3.3. Safety Requirements

To ensure safety, an emergency stop remote is provided to the user. This is been depicted in Figure 3.3. Pressing the red knob will switch off the mobile platform controller and the mobile platform halts at that instant. The red knob must be rotated clockwise to release the emergency condition, and this will, in turn, restart the controller.

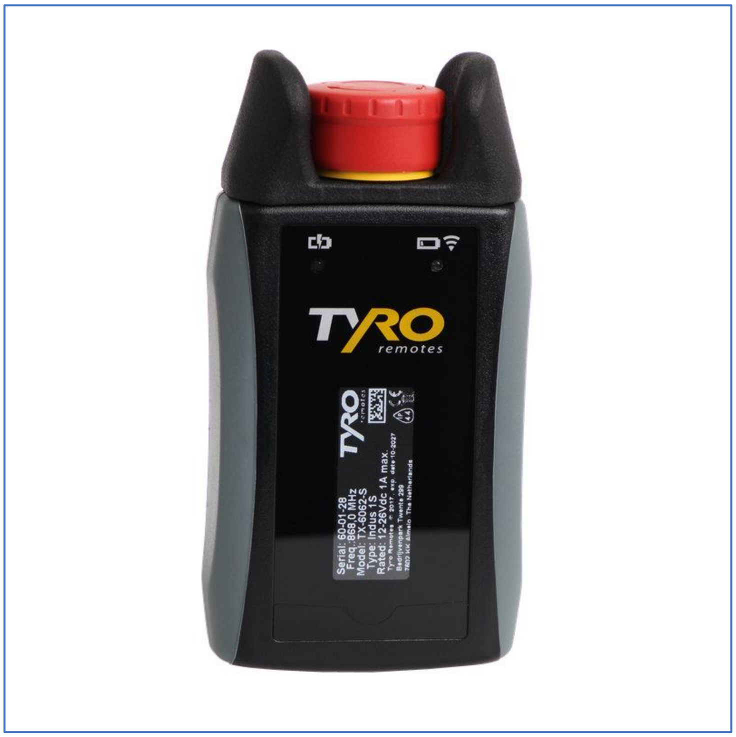


**Figure 3.3: Safety Remote Switch (Tyroremotes, 2000)**

Moreover, the operating area must be entered only with sturdy, closed footwear. The area used for testing must be fenced and no entry warning sign must be displayed at the workplace.

# 4. Background and Literature Survey

A minimum of three wheels is required for a mobile platform, to guarantee a stable balance. For mobile platforms with more than three wheels, a suspension system is required, to ensure contact to the ground is maintained in case of uneven or rough terrain.

## 4.1. Mecanum Wheels

Mecanum wheels are designed to allow motion of the mobile platform in any direction. Mecanum wheels have three degrees of freedom. It includes rotation around the wheel axle, around the rollers and around the contact point as depicted in Figure 4.1. Since the movement in the plane has three degrees of freedom, thus only three wheels can be independently controlled.

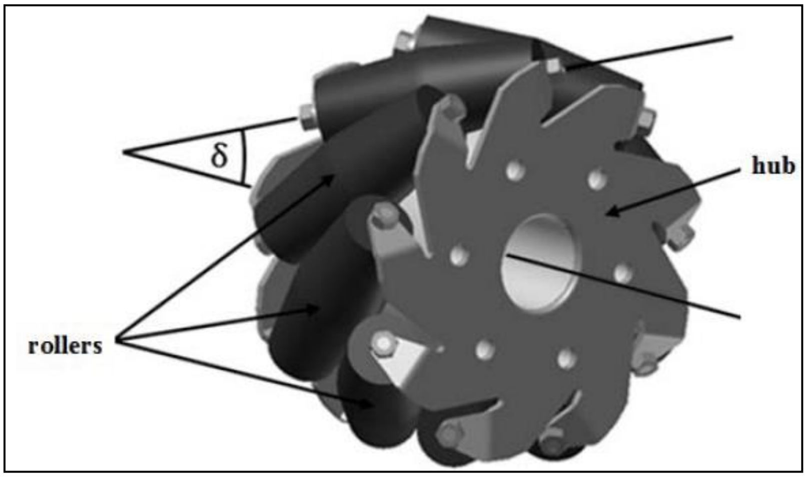


**Figure 4.1: Mechanical structure of Mecanum wheel ( HENDZEL & RYKAŁA, 2017)**

In a Mecanum wheel, a series of cylindrical rollers are attached to its circumference. The number of rollers varies depending on the wheel design. Each roller can be freely rotated around its own axis. The angle between the axis of rotation of the wheel and axis of rotation of the roller is denoted by $\delta$ as shown in Figure 4.1. The axes of rotation of these rollers are inclined at 45° with respect to the wheel plane. These rollers are passive. The primary axis of the wheel acts as an active powered joint. The main advantage of this design is that the vehicle can move in any trajectory instantaneously as opposed to a differential drive or Ackermann-steered platform.

## 4.2. Caster Wheels

A Caster wheel has two degrees of freedom, due to rotation around the wheel axle and the offset steering joint. This has been depicted in Figure 4.2. The caster wheel is highly directional due to the existence of the primary axis of rotation.

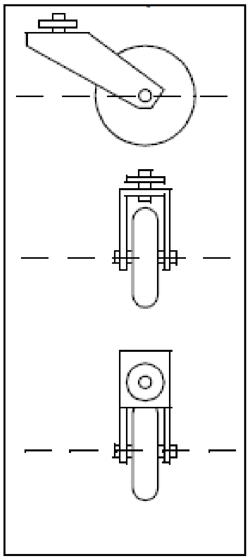


**Figure 4.2: Caster wheel different angle views (SIEGWART & NOURBAKHSH, 2004)**

It should be steered along a vertical axis, to move a caster wheel in a different direction. The force is distributed during steering on to the robot chassis because the caster wheel rotates around an offset axis. In robotics, caster wheels are used to provide mobility.

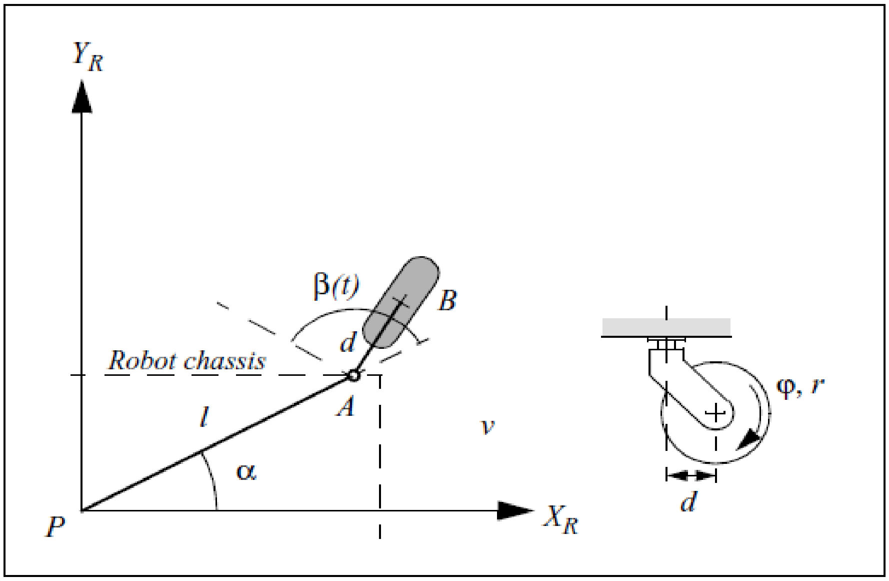


**Figure 4.3: Mobile platform with caster wheels (SIEGWART & NOURBAKHSH, 2004)**

With reference to (SIEGWART & NOURBAKHSH, 2004), Point B denotes the wheel contact point with the ground. Point B relates to the rigid rod AB of length d as depicted in Figure 4.3. The wheel plane is always aligned with AB. $\beta(t)$ represents the steering angle and orientation of AB with respect to time. $\varphi$ denotes the angular displacement of the caster wheel and $\dot{\varphi}(t)$ denotes the angular velocity of the wheel with respect to time. When the input force is applied to a stationary castor, the frictional force will act in the direction opposite to the applied force and it acts on the edge of the wheel that rests on the ground. This friction applies a torque to the wheels which help in starting its rotation. Therefore, static friction is necessary for the wheel to roll.

## 4.3. Omnidirectional Platform Configuration

The Omnidirectional platform has four identical Mecanum wheels which can be controlled by stepper motors. These Mecanum wheels are connected on the shaft and can turn with the help of these motors. Different combinations of rotation of the Mecanum wheels are depicted in Figure 4.4.

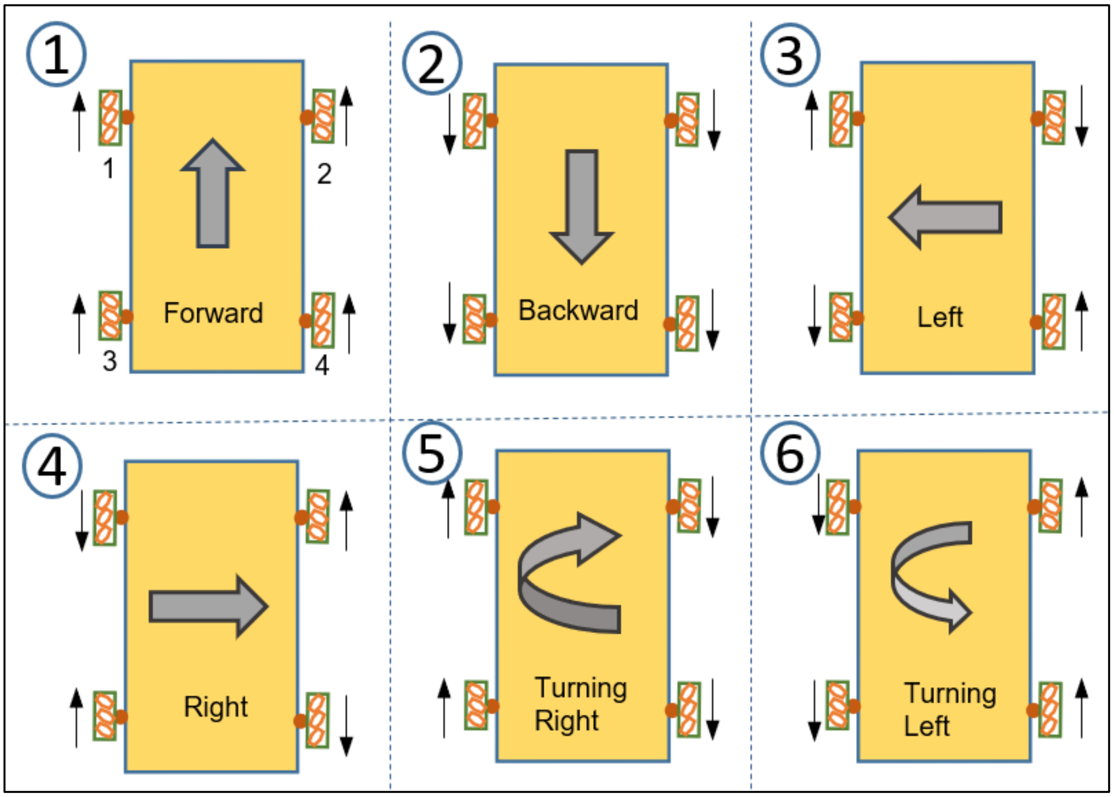


**Figure 4.4: A few possible combinations for rotations of Mecanum wheels**

The platform moves linearly when all Mecanum wheels are rotating in the same direction (can be clockwise or anticlockwise) as shown in the above Figure 4.4 with blocks 1 and 2.

However, if one pair of diagonally opposite wheels rotate in the one direction and the other pair of diagonally opposite wheels rotate in opposite direction, then platform traverses left or right as shown in the above Figure 4.4 with blocks 3 and 4.

Moreover, if a pair of wheels lying on the same side of the platform rotates in one direction while the other pair of wheels lying on another side of the platform rotates in another direction, the platform can be turned in the right or left direction as shown in the above Figure 4.4 with blocks 5 and 6.

The above-stated wheel motion combinations show that the translation and rotational motions of the platform can be combined arbitrarily.

## 4.4. Control of Omnidirectional Platform

The overall kinematics for the omnidirectional platform is been depicted in Figure 4.5. The equations for the motion of the complete platform have been derived using the individual angular speed of the wheels.

The omnidirectional platform is represented by the coordinate frame $X_R Y_R Z_R$ at the center of mass described by the characteristic point $\boldsymbol{T}$. The nomenclature is summarized in the appendix in chapter 19.1.

Following equations are referenced from ( HENDZEL & RYKAŁA, 2017) describe the motion constraints along the $X_R$ axis and $Y_R$ axis.

Let the angular velocity of the four Mecanum wheels of the omnidirectional platform be $\dot{\varphi}_1, \dot{\varphi}_2, \dot{\varphi}_3, \dot{\varphi}_4$. Let the radius of the Mecanum wheel and radius of rollers be denoted by $r_1$ and $r_2$ respectively. $\boldsymbol{\delta}$ denotes the rotation of the platform around the point $T$ in the 2-D plane. $T_X$ is the distance between the point $T$ and the front axle $A_1$. The same geometric distance is present between the point $T$ and the back axle $A_2$. The total width of the mobile platform is denoted by $2T_Y$.

$$v_{TX_R} - v_{TY_R} - \dot{\delta}(T_X + T_Y) - \dot{\varphi}_1(r_1 + r_2) = 0 \tag{4.1}$$

$$v_{TX_R} + v_{TY_R} + \dot{\delta}(T_X + T_Y) - \dot{\varphi}_2(r_1 + r_2) = 0 \tag{4.2}$$

$$v_{TX_R} + v_{TY_R} - \dot{\delta}(T_X + T_Y) - \dot{\varphi}_3(r_1 + r_2) = 0 \tag{4.3}$$

$$v_{TX_R} - v_{TY_R} + \dot{\delta}(T_X + T_Y) - \dot{\varphi}_4(r_1 + r_2) = 0 \tag{4.4}$$

Here, the projection of the velocity of the platform at the point $\boldsymbol{T}$ on the $X_R$ axis is denoted by $v_{TX_R}$ and the projection of the velocity of the platform at the point $\boldsymbol{T}$ on the $Y_R$ axis is denoted by $v_{TY_R}$.

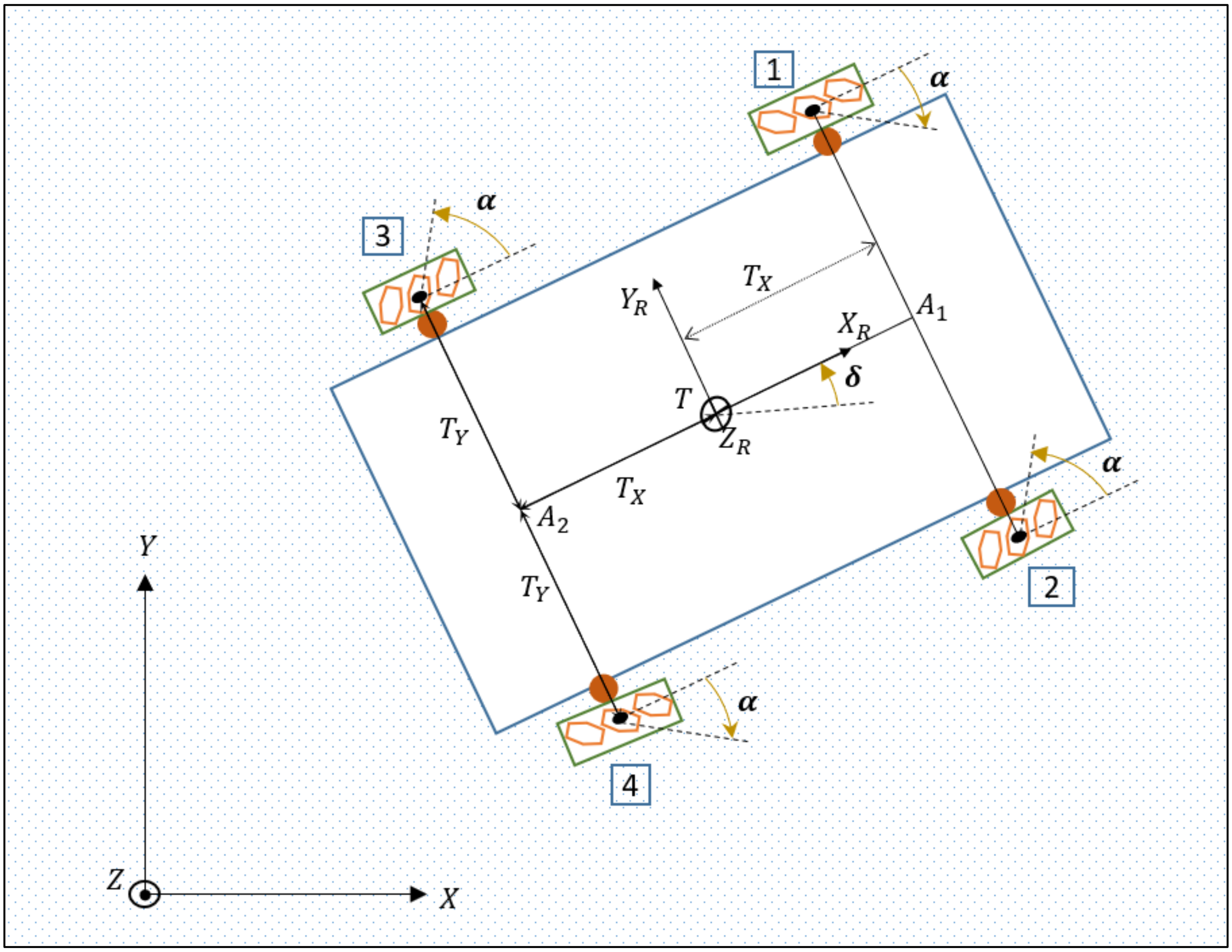


**Figure 4.5: Omnidirectional Platform kinematics**

The holonomic geometric constraints are represented by the equations (4.1), (4.2), (4.3) and (4.4).

Now, the solutions for the inverse kinematics are presented using the velocity $\overrightarrow{\boldsymbol{v_T}}$ as follows –

$$\dot{\varphi}_1 = \frac{1}{(r_1 + r_2)}\left[v_{TX_R} - v_{TY_R} - \dot{\delta}\,(T_X + T_Y)\right] \tag{4.5}$$

$$\dot{\varphi}_2 = \frac{1}{(r_1 + r_2)}\left[v_{TX_R} + v_{TY_R} + \dot{\delta}\,(T_X + T_Y)\right] \tag{4.6}$$

$$\dot{\varphi}_3 = \frac{1}{(r_1 + r_2)}\left[v_{TX_R} + v_{TY_R} - \dot{\delta}\,(T_X + T_Y)\right] \tag{4.7}$$

$$\dot{\varphi}_4 = \frac{1}{(r_1 + r_2)}\left[v_{TX_R} - v_{TY_R} + \dot{\delta}\,(T_X + T_Y)\right] \tag{4.8}$$

Equations (4.5), (4.6), (4.7) and (4.8) can be written in the form of

$$\boldsymbol{\dot{\varphi}} = \boldsymbol{J V_{T_R}} \tag{4.9}$$

where,

$$\boldsymbol{\dot{\varphi}} = \begin{bmatrix} \dot{\varphi}_1 \\ \dot{\varphi}_2 \\ \dot{\varphi}_3 \\ \dot{\varphi}_4 \end{bmatrix} \tag{4.10}$$

$$\boldsymbol{V_{T_R}} = \begin{bmatrix} v_{TX_R} \\ v_{TY_R} \\ \dot{\delta} \end{bmatrix} \tag{4.11}$$

Using a matrix inversion of $\boldsymbol{J}$, the forward kinematics problem can be solved. The Moore-Penrose theorem can be implemented for finding an inversion of rectangular matrices such as $J$ as shown below:

$$\boldsymbol{J_{od}} = \boldsymbol{J^{+}\dot{\varphi}J^{+}} = \left(\boldsymbol{J^{T}J}\right)^{\boldsymbol{-1}}\boldsymbol{J^{T}} \tag{4.12}$$

$\boldsymbol{J^{+}}$ is the Moore-Penrose pseudoinverse. $\boldsymbol{J_{od}}$ can be represented in the following form:

$$\boldsymbol{J_{od}} = \frac{1}{4}\begin{bmatrix} (r_1 + r_2) & (r_1 + r_2) & (r_1 + r_2) & (r_1 + r_2) \\ -(r_1 + r_2) & (r_1 + r_2) & (r_1 + r_2) & -(r_1 + r_2) \\ \frac{-(r_1 + r_2)}{(T_X + T_Y)} & \frac{(r_1 + r_2)}{(T_X + T_Y)} & \frac{-(r_1 + r_2)}{(T_X + T_Y)} & \frac{(r_1 + r_2)}{(T_X + T_Y)} \end{bmatrix} \tag{4.13}$$

The forward kinematics solution can be obtained by using the equation

$$\boldsymbol{V_{T_R}} = \boldsymbol{J_{od}\dot{\varphi}} \tag{4.14}$$

The projection of the platform velocity at point $T$ can be calculated using the below mentioned equations:

$$v_{TX_R} = \left(\frac{r_1 + r_2}{4}\right)[\dot{\varphi}_1 + \dot{\varphi}_2 + \dot{\varphi}_3 + \dot{\varphi}_4] \quad \textbf{(4.15)}$$

$$v_{TY_R} = \left(\frac{r_1 + r_2}{4}\right)[-\dot{\varphi}_1 + \dot{\varphi}_2 + \dot{\varphi}_3 - \dot{\varphi}_4] \quad \textbf{(4.16)}$$

$$\dot{\delta} = \left(\frac{r_1 + r_2}{4(T_X + T_Y)}\right)[-\dot{\varphi}_1 + \dot{\varphi}_2 - \dot{\varphi}_3 + \dot{\varphi}_4] \quad \textbf{(4.17)}$$

Here, $v_{TX_R}$ and $v_{TY_R}$ denotes the linear translational velocity of the omnidirectional platform along the $X_R$ axis and $Y_R$ axis. The rotation of the omnidirectional platform is denoted by $\dot{\delta}$.

The above equations state the derivation of the motion of the platform using the angular velocities of the individual Mecanum wheels.

# 5. General Control scheme of the Mobile Robot System

The basic principle behind the control scheme for a mobile robot is “See-Think-Act”. This overall control scheme has been depicted in Figure 5.1. This principle starts with perceiving and analyzing the raw data from the environment. Then, the input to the actuators is fed so that the mobile robot performs the desired action.

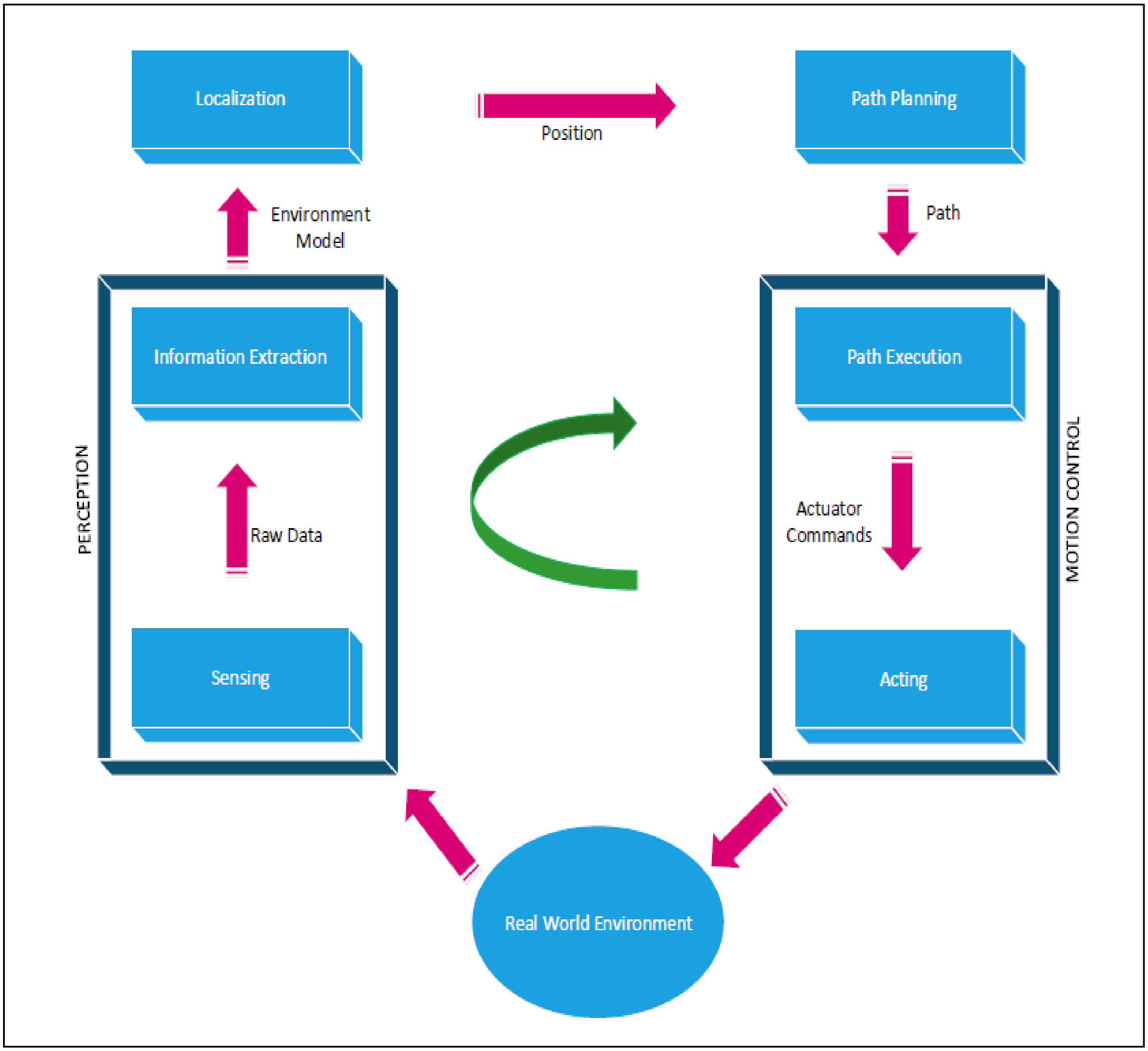


**Figure 5.1: Control Scheme for Mobile Robot (SIEGWART & NOURBAKHSH, 2004)**

The mobile robot control scheme includes four major overall steps –

## 5.1. Perception

Sensors are the key component for perceiving the environment. Perception is the initial step in the robot control scheme. Perception involves the sensing of the unknown and unstructured surrounding environment. Perception includes the collection of data from such an environment as depicted in Figure 5.2. Perception can be done in term of vision or laser, followed by feature extraction. The next step is the extraction and filtering of the relevant and desired data using all collected/sensed data. Various perception sensors that can be mounted on the mobile robot are heading sensors (which include compass, gyroscope) or range sensors (which include the laser sensor, sonar) or vision sensor (such as a camera).

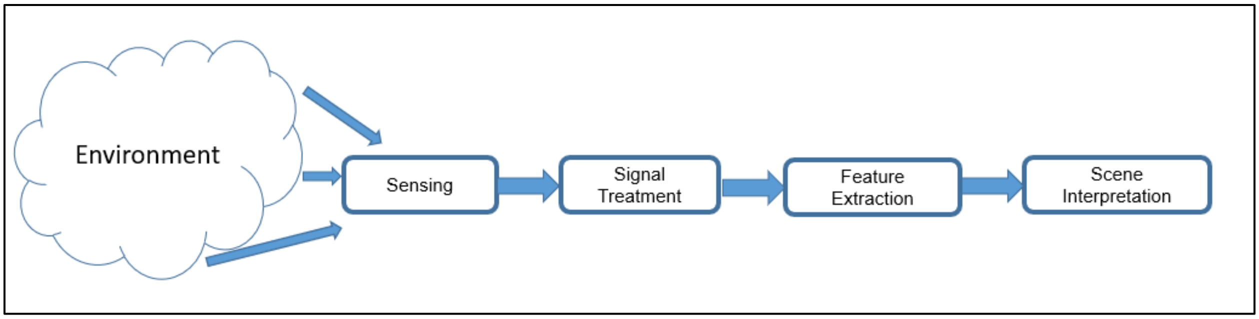


**Figure 5.2: Internal steps for Perception**

## 5.2. Localization

Once the data are perceived, it can be used further for localizing the mobile robot in the given environment. Localization includes knowledge about the position and orientation of the mobile robot in the current surrounding environment.

Localization helps the mobile robot in recovering its position in the map when it is lost while navigating to the goal point. Moreover, localization further provides the verification of whether the mobile robot has reached its goal position or not in the map coordinate frame. The localization of the mobile robot has been depicted in Figure 5.3.

The raw sensor data is the first input fed into the matching block. The map database is also used as input to the matching block. Further, with an output from the encoder and the stored map data, a prediction for the position of the mobile robot is estimated. This is fed as another input to the matching block. Since the current perception is read from the sensor, it is used to match with the predicted position of the mobile robot in the given map. If matching is successful, the latest value of the position is used as an input for the prediction of the position for the next cycle.

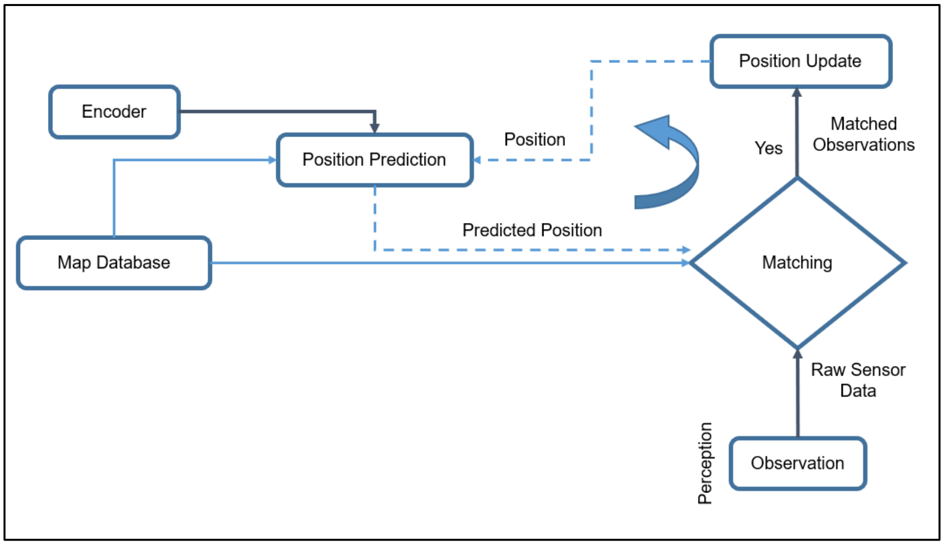


**Figure 5.3: Flowchart for mobile robot localization**

Figure 5.4 depicts the behavior-based navigation for the mobile robot. This emphasizes the fact that the creation of a geometric map should be often avoided for the mobile robot, as the sensor data and effectors are mostly noisy, and the information is limited. This approach avoids the explicit localization of the mobile robot as it is based on the combination of the set of behaviors.

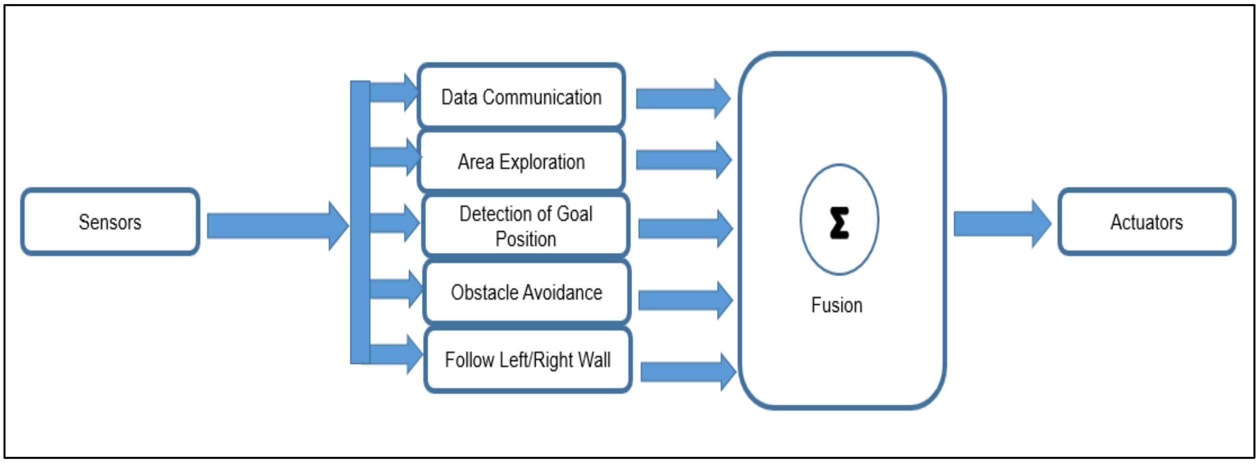


**Figure 5.4: Behavior-based navigation architecture**

However, there are a few drawbacks associated with it. This approach provides a solution very much specific to the environment and processing of data consumes a lot of time. The most crucial drawback is the fusion and rapid switching among all behaviors, which hampers the fine-tuning of the parameters for each behavior. To overcome this drawback, model-based navigation is implemented as depicted below in Figure 5.5. This approach involves explicit localization of the mobile robot using the sensor data and provides its position estimates in the given map.

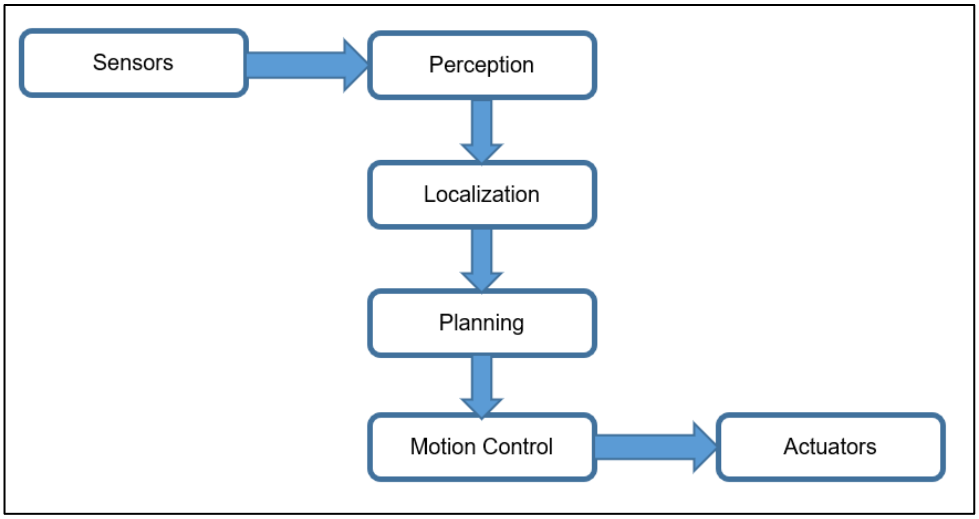


**Figure 5.5: Map-based navigation architecture**

There are a few advantages associated with this approach. The current position estimate of the mobile robot in the given map is readily available to the user. The map generated by the robot using the perception sensors can also be used later by the user. The key factor is that it is completely dependent on the map generated by the mobile robot. Therefore, the perception data needs both to be precise as well as accurate.

Belief representations are used to describe the belief of the robot position in the given map. It can be a single-hypothesis belief that works for a unique robot position or multi-hypothesis belief which describes the degree of uncertainty about the robot position. Further, the localization approach can be unimodal such as ‘Kalman Estimator’. In the unimodal approach, the mobile robot position estimate cannot be performed in case of the “kidnapped robot problem”. The “kidnapped robot problem” refers to the condition where the features cannot be matched correctly as the difference between estimate position and real position is large.

However, a grid-based approach can be implemented to overcome the problem of the “kidnapped robot problem”. This localization approach is multimodal in which the position estimate of the mobile robot is not lost. For example, in ‘Markov Localization’, multimodality is performed using small-sized grid cells. But this approach involves significant usage of processing power and memory to update the relevant cell information. In the 1990s, ‘Monte Carlo Localization’ was proposed which is based on sampling methodology and is very much efficient in calculating the estimated position of the mobile robot.

## 5.3. Path Planning

After the mobile robot is localized, the path for the mobile robot to be traversed can be planned. This will provide a plan for reaching the goal position. This can also include an obstacle avoidance algorithm to avoid an obstacle encountered during the movement of the mobile robot. Path planners use various decomposition strategies to reach a goal position. It can be a Road map (recognizes all possible routes in the free space), Cell Decomposition (identifies free space as free and occupied cells) or Potential field (implements potential gradient to direct the robot towards the goal).

## 5.4. Motion Control

Finally, the motion control can be applied to the mobile robot which allows it to travel to a goal position in the environment. In this project, we use a self-developed USB-connected ultra-low-cost motor controller that does vector control of the stepper motors. Each motor goes through a calibration where the position of the armatures with respect to the encoders is estimated. This is later used to control the commutation very precisely.

The cascaded control for the stepper motor is depicted in Figure 5.6. The desired position, desired angular speed and desired current are denoted by $\theta_d, \omega_d$ and $i_d$. The generated position, velocity and current are denoted by $\theta$ , $\omega$ and $i$ respectively. The current control loop helps in attaining the desired torque output of the stepper motor. The velocity control loop in return controls the velocity so that the desired speed can be achieved.

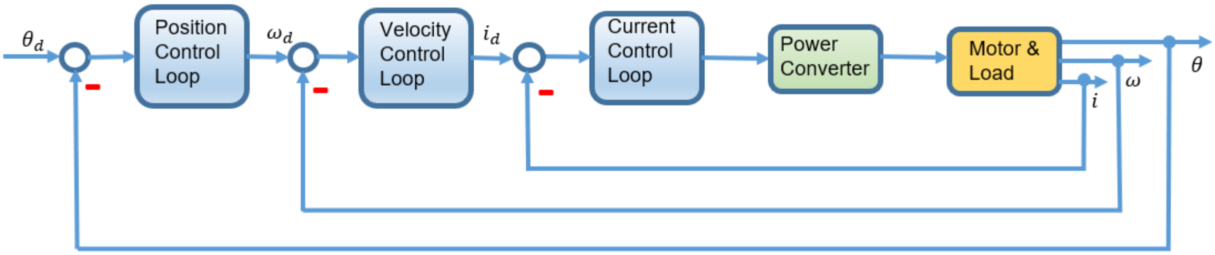


**Figure 5.6: Cascaded control of a motor**

# 6. Introduction to Open Source Robotic Software

The Robot Operating System (ROS) provides a framework for writing the software for the robot. ROS includes a collection of tools, libraries and conventions which help in simplifying complex tasks. ROS is applicable over a wide variety of robotics platforms and facilitates the robust behavior of robots.

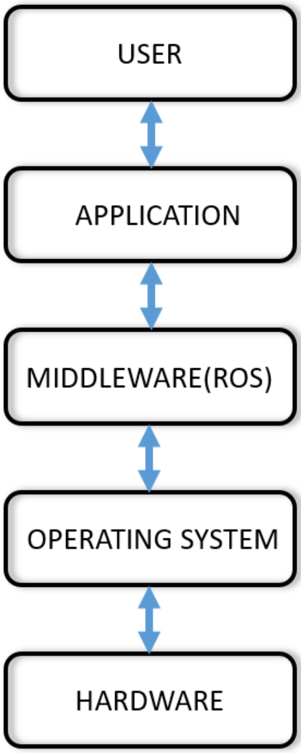


**Figure 6.1: Layered Architecture with ROS as Middleware**

Figure 6.1 shows the layered architecture with ROS as the middleware. ROS provides services of an operating system such as low-level device control, hardware abstraction and package management.

ROS is an open-source framework that aims at robotics software development in a collaborative manner.

The advantages of using ROS are as follows –

- **Easy Integration** – ROS provides easy integration between numerous hardware by considering them as ROS nodes. These ROS nodes can be interconnected, and data can be easily exchanged.
- **Easy Visualization** – ROS provides a lot of tools for easy visualization of the process and data flow. It includes visualization of the system interconnection as a tree diagram, plotting the data graphically and data logging.
- **Multilingual** – ROS implementation can be done using different programming languages such as C++, python.
- **Standalone Libraries** – ROS involves the inclusion of the user-defined standalone libraries suitable for different applications. These libraries can be used by various ROS nodes and communication can be easily established.
- **Free & Open Source** – ROS is a free and open-source framework. Its core uses the Berkeley Software Distribution (BSD) license and can be used for both commercial and non-commercial purposes.

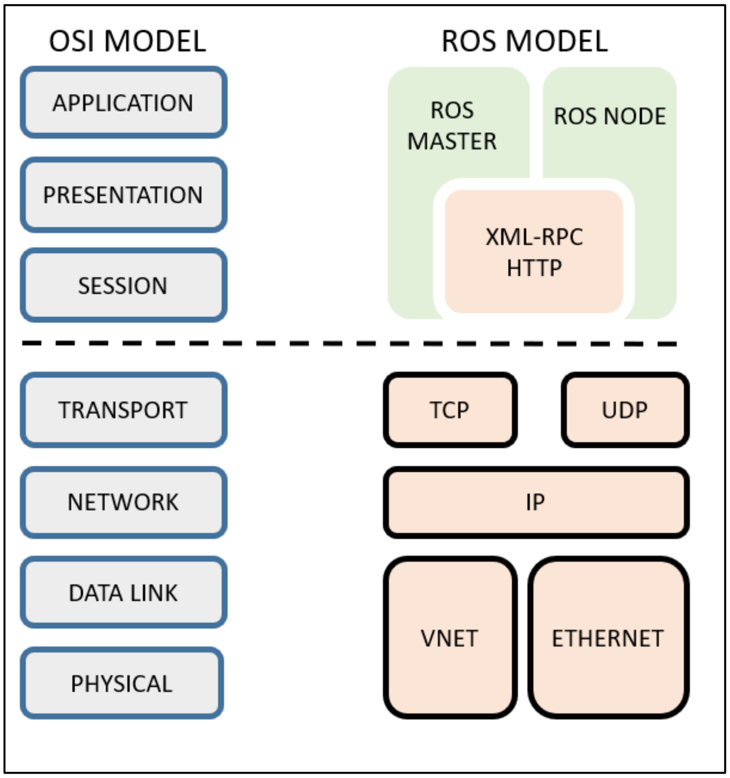


**Figure 6.2: ROS Communication Model**

With reference to Figure 6.2, the HTTP-based protocol known as XMLRPC which is used for communication with the ROS Master has been depicted. A Wi-Fi router is used to communicate with the robot from the ground control station (laptop). This helps in viewing the ROS topics on the master from other PCs. The registration APIs of the ROS Master allows the ROS nodes to register as publishers or subscribers.

ROS provides a lot of information for the user interface during the run-time. The data exchange between various components can be visualized and can even be plotted. Moreover, the user-defined data can also be sent to any component. All the data from various components can be visualized in three-dimensional (3D) space. The user can also define their own 3D model. Parallel processing is also achieved in ROS, for example, during the robot motion, images can be captured and stored and the feedback can be sent.

## 6.1. ROS Core

Roscore provides the service which further helps the ros nodes to communicate and transfer data streams with each other. Roscore must be running in every ROS environment to ensure that the ros nodes can interact with each other. Roscore helps in the registration of ros nodes during the startup. Each node informs roscore about the datatypes of the message which it will send or receive. Roscore, in turn, sends the address of the message producer or receiver.

Catkin in an open-source build system used for generating the executables. Catkin workspace can be created to set up a new project in ROS. After the creation of a catkin workspace, the package can be created mentioning the dependencies such as rospy or roscpp. They indicate the programming language which can be used for writing node implementation such as C++ or python.

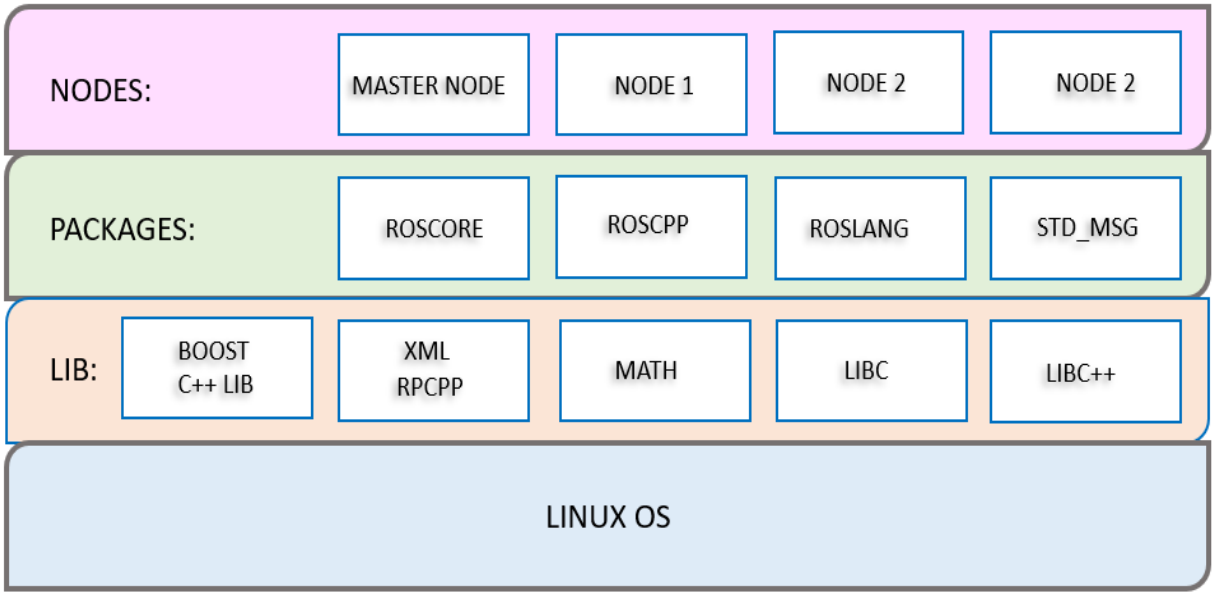


Figure 6.3: ROS Components

The above Figure 6.3 represents various ROS components such as ROS nodes, ROS packages and ROS libraries. ROS nodes include the master node and several other nodes which can be used for creating ROS service or ROS action. It also includes various ROS packages such as ROSCORE, ROSLANG. These packages are utilized to establish communication between various ROS nodes. Moreover, ROS libraries such as MATH, LIBC can be imported while creating new ROS nodes.

These components are further explained in detail below. There have been about 500 ROS packages available so far.

## 6.2. ROS Master

The ROS master is the server that provides the naming and registration services to the nodes. It also traces all the network addresses of the nodes. The ROS Master can be started by running ROS core.

Figure 6.4 depicts the architecture of the ROS system with communication between the ROS Master and several ROS nodes. All the ROS nodes must be registered to the ROS Master. The ROS Master must be notified before ROS Nodes publish any message on ROS Topics.

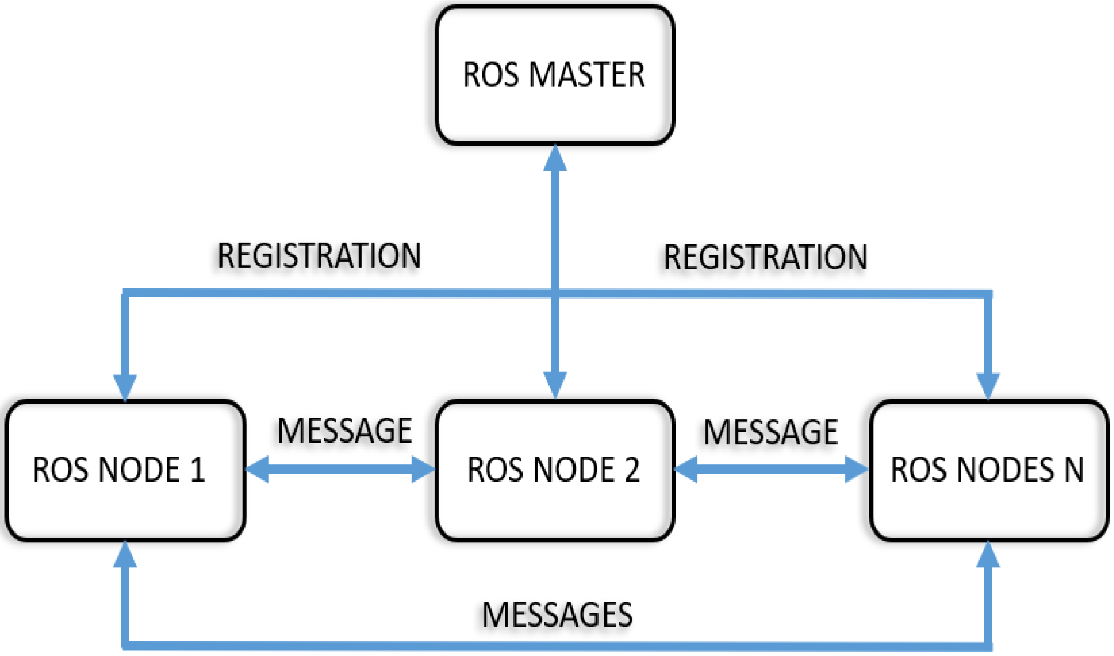


**Figure 6.4: ROS Architecture with ROS Master and ROS Nodes**

## 6.3. ROS Nodes

A ROS node is an executable that performs computation as per the user-defined function. By using a ROS action, ROS topic or ROS service, nodes can communicate with each other. A ROS node is called a publisher node when it publishes a data stream on the ROS topic. A ROS node is called a subscriber node when it retrieves data sent on a ROS topic. Data exchange between ROS nodes can take place via ROS topics, ROS services or ROS actions.

## 6.4. ROS Topic

ROS topics act as channels through which ROS nodes can communicate. ROS publisher nodes publish messages on these ROS topics. ROS subscriber nodes retrieve messages from these ROS topics. ROS topics are the simplest way of communication among ROS nodes.

## 6.5. ROS Messages

ROS messages are used to describe the data type which should be transmitted over the ROS topics. These message datatypes are referenced while creating a ROS publisher or ROS subscriber.

## 6.6. ROS Parameter Server

The ROS parameter server includes a dictionary where all the parameters can be stored. These stored parameters can be utilized by the ROS nodes during execution, to perform the desired functionalities, for example, robot model parameters, localizer parameters.

## 6.7. ROS Service

ROS services are like synchronous functionalities provided by the robot. ROS services are best suited for scenarios that are occasional and time-bounded. ROS service includes client-server-based communication. Service servers are used for implementing the service by specifying a callback to retrieve the service request. The ROS service server keeps the service running so that it can be called by ROS service client whenever required. The ROS service client can call the ROS service and utilize its functionalities. A ROS service type definition contains two parts: request and response. The request indicates how to call the ROS service. The response indicates how the service will respond after completing the functionality.

## 6.8. ROS Action

ROS actions are like asynchronous functionalities provided by the robot. ROS actions are best suited for scenarios where the exact time duration of the task cannot be calculated, and feedback is required to indicate task completion.

Similar to a ROS service, a ROS action also includes client-server-based communication as depicted in Figure 6.5. ROS action server is used to implement and initiate a ROS action. A ROS action client can be used to call and utilize a ROS action. ROS action defines three parts: goal, result and feedback. The goal describes the data which can be used as an input argument to perform a ROS action. Feedback is used to get the step by step result of each iteration when ROS service has been called. Result is the final data send when the complete ROS action is finished. The goal data is used by ROS action server to perform the ROS action.

A ROS action client along with sending the goal, can also be used to define the function taking place when ROS action starts, or ROS action terminates.

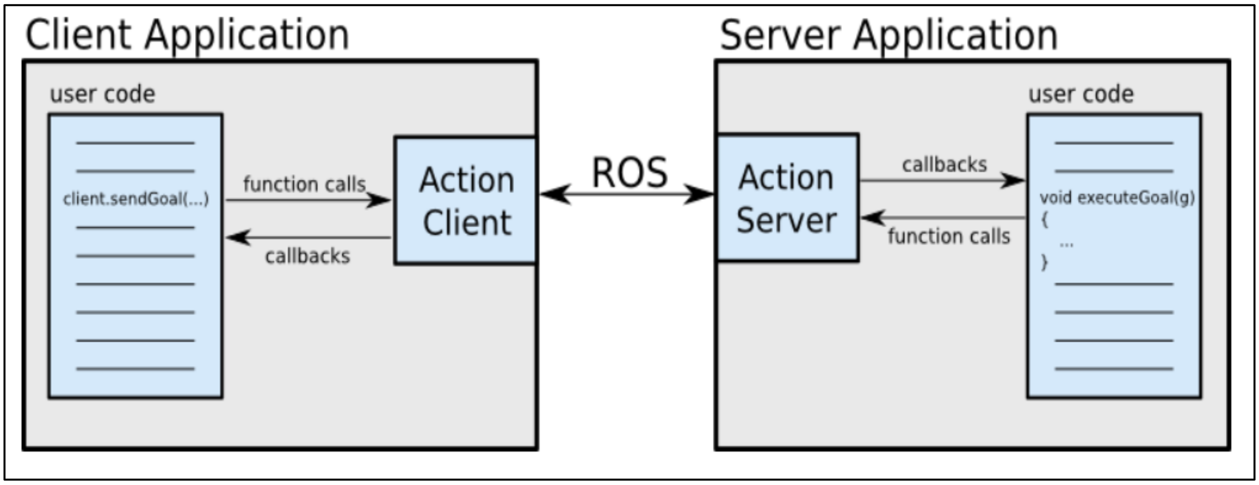


**Figure 6.5: ROS Action Client-Server Interface (ROS.org, 2007)**

## 6.9. ROS Transformation Frames

All the coordinate frames in ROS must be organized in a tree by parent/child relationships between them. ROS TF includes TF broadcaster and TF listener. TF broadcaster helps in broadcasting the transformation between two frames. TF listener can be used for monitoring the relationship between two transformation frames.

## 6.10. ROS Rviz

ROS rviz helps in visualizing the ROS messages that are transmitted through the ROS topic. For example, the laser data or data from a camera can be viewed using ROS rviz by selecting the appropriate ROS topic on which the data are published.

Using the ROS rviz, the different transformation frames such as base_link, sensor_frame can also be visualized.

## 6.11. ROS Gazebo

A ROS Gazebo provides the environment to simulate robots. The robot model can be created by writing a Unified Robot Description Format file. This URDF file includes location, shape and origin of the visual, collision and inertial properties of all the robot components. The URDF file also provides the support for defining the type and dimension of joints for joining the links. Moreover, sensors along with complete description can also be added further into the robot.

## 6.12. ROS Plugins

Following are the frequently used ROS plugins:

- **rqt_graph:** To display the structure of all nodes along with the connected topics.
- **rqt_plot:** To plot the ROS messages that are transmitted over the ROS topics.
- **rosbag:** To record and store ROS messages. This data can be later played in a time-synchronized fashion.
- **roscd:** To change directory to a package or stack.
- **roswtf:** For debugging the ROS system.
- **roscore:** To start the ROS master.
- **view_frames:** To visualize the tree structure of all coordinate transforms.
- **rqt_console:** This plugin provides a user interface to display and filter ROS messages.
- **rqt_reconfigure:** To view and edit the configuration parameters dynamically in the run-time.

# 7. Creation of Environment Map

In mobile robotics, autonomous navigation is largely dependent on the existing map. The mapping process plays a vital role since its output, that is, maps are used further as an input to navigate the mobile robot. Therefore, the map needs to be precise and accurate. Simultaneous Localization and Mapping (SLAM) is most commonly used in robotics for building and exploring the unknown environment. In this algorithm, the map is created by the mobile robot while it is in motion and the path to be traveled is found using the obstacle avoidance sensor data.

ROS slam_karto has been used for building the map of the environment. KartoSLAM is a graph-based SLAM approach that represents the map by means of the graph of nodes, storing poses and features. The constraints between the successive poses are indicated by the edges in the graph. In ROS slam_karto, the optimization problem is solved using the Cholesky matrix decomposition (Vincent, Limketkai, & Eriksen, April 2010.). Further, Sparse Pose Adjustment (SPA) (Konolige, et al., Oct 2010) is responsible for both scan-matching and the loop closure.

The initial step involves running the ROS slam_karto package. This step is followed by moving the robot in the map using the keyboard. It involves manually exploring the map of the robot environment using a laser sensor. The recordings can be visualized in RVIZ as shown in Figure 7.1.

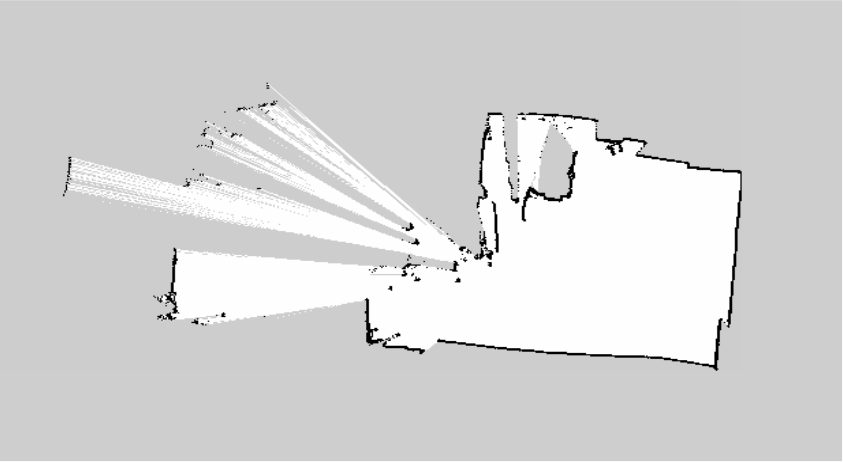


**Figure 7.1: Map Recorded with Laser Sensor**

Once the entire map has been explored, the map can be saved by running “rosrun map_server map_saver” command on the terminal. This will create two files map.pgm and map.yaml in our home directory. Then, the newly recorded map can be copied into our ROS node, so that map server can be started with our map. Then, the slam_karto node can be stopped as it is not needed anymore.

# 8. Monte Carlo Localization

Monte Carlo Localization (MCL) represents the belief about the robot's location using as a sample set. In our case, each sample – here also called particle - is describing a position as a x, y coordinate and orientation. The localizer can be initialized at the start with a uniform distribution over the whole map or by information through the user who can click and select an approximate position in the map.

The algorithm alternates between sensor updates, resampling and motion updates. These updates are explained further in detail below -

- **Motion update**

In the motion update, the odometry reading of the platform is used to predict the motion of each sample. The odometry model is probabilistic. The relative motion applied to each sample is drawn from a distribution that is formed by the actual odometry reading on the one hand and the configured parameters describing the expected error of the odometry on the other hand. In our model, there are values that denote the error in translation and orientation for forward, sideward and rotational motions. The odometry model should always overestimate the actual error in the odometry readings as the localization can otherwise be lost when the actual odometry error exceeds the modeled one.

- **Sensor update**

In the sensor update, the laser data is projected from each sample's position into the map.

The result of the projection is the map coordinates from where the robot has sensed a laser sensor hit assuming the respective sample represents the correct robot localization.

What we want to know is how probable the location of the sample is, given the sensor reading. However, what is first calculated is the probability of the sensor reading given that location. Bayes theorem (Thrun, Burgard, & Fox, 2005) is then used to calculate the event with the condition.

In our case, a previously generated distance field is used to approximate the probability of the sensor readings. This is called the “likelihood field model” (Thrun, Burgard, & Fox, 2005). That is, for each laser reading, the MCL checks how far the projected reading is from the next known laser obstacle in the map.

Since we have a 2D map, this information can be precalculated and stored in a 2D array, so the algorithm only needs to look up a single value in a 2D array for each laser reading of each sample. This is considerably faster than other approaches such as following a line starting from the laser sensor in the direction of the laser ray and seeing when an obstacle is hit in the map and comparing that expected hit distance with the actual reading (so-called "beam model", (Thrun, Burgard, & Fox, 2005) can be further referred for more detail).

A probability for every single ray is calculated from the distance. All the ray's probabilities are combined for each sample to form the weight of the sample. After normalizing the sum of all samples' weights, each sample holds an approximation of its share of the overall probability mass.

If there were multiple sensors, a sensor update, and resampling would be done for each one. Alternatively, if the sensor readings arrive at the same frequency, the weights of the multiple sensor updates can be fused by multiplying the weight originating from the different sensor models.

- **Resampling**

In order for the distribution to converge, the samples with very low probability mass are reassigned to more probable positions. This is done by drawing a random number between zero and one. We can then walk through the list of all samples and add their weights up until we reach the drawn random number. Like that, more probable samples are chosen more often than less probable one. The pose of the sample, we have reached when the weights are added up to the random number is copied to the discarded sample.

Initially, the resampled samples are practically exact copies of their originals, but the next motion update will spread them out again.

- **Result**

The resulting location of the robot is usually not one of the currently best samples but a weighted average of a number of samples around the area of highest probability mass in the sample set. This is done to reduce noise and smoothen the location that is reported to other components of the system. Additionally, a Kalman filter is often used to further smoothen the estimated position in case the value is used for control of the platform.

## 8.1. Adaptive Monte Carlo Localization

Adaptive Monte Carlo Localization (AMCL) is an adaptive form of MCL in terms of the number of particles that are dynamically modified throughout the localization process. ROS AMCL utilizes the particle filter, laser data and odometry data to improvise the localization of the mobile robot. There are numerous parameters to be configured for AMCL as described in section 8.1.4, 8.1.5 and 8.1.6.

### 8.1.1. Subscribed Topics

AMCL subscribe to the scan topic to retrieve the laser scan readings. It also utilizes the transformation frame topic to fetch the latest transformation between the coordinate frames. To retrieve the map information, AMCL subscribes to the map topic as well.

### 8.1.2. Published Topics

AMCL publishes the pose estimation of the robot with covariance on the ROS topic “amcl_pose”. Moreover, the transformation from odometry to map frame is published by AMCL on the transformation frame topic.

### 8.1.3. Services

AMCL uses “global_localization” ROS service to randomly spread the particles for the initialization of the robot pose estimation in the map frame. Service “static_map” is also called by AMCL node to retrieve the map information.

### 8.1.4. Filter Parameters

Parameters such as “min_particles” and “max_particles” describe the range in which the particle for robot pose estimation can be dynamically varied in the run-time during localization. Initially, the localization algorithm starts with “max_particles” and finally “min_particles” is used to define the best likely estimation of robot position. “Kld_err” is another parameter that describes the maximum error allowed between the estimated and true distribution. “transform_tolerance” parameter is used to set the post-date by which the transform will be valid in the future and thus prevent lagging.

### 8.1.5. Laser Parameters

The laser range can also be set using the parameters “laser_min_range” and “laser_max_range”. This can be used to define the laser scan boundary. Depending on the laser sensor, “laser_max_beams” parameter allows the user to define the number of laser beams to be spread in each scan. The laser model can be defined with the “laser_model_type” parameter. The model can be a beam, “likelihood_field”, or “likelihood_field_probability”. While utilizing the likelihood_field algorithm, “laser_likelihood_max_dist” parameter can be set to allow obstacle inflation on the map. The beam model includes z_hit, z_short, z_max, and z_rand whereas “likelihood_field” model only includes “z_hit” and “z_rand”.

The mixture weight for z_hit and z_rand parts of the model can be set using the parameters “laser_z_hit” and “laser_z_rand” respectively. These parameters play a vital role in improving the localization accuracy.

### 8.1.6. Odometry Parameters

Odometry parameters help in configuring the robot movement. These parameters are utilized by AMCL to enhance the localization task. The parameter “odom_model_type” is used to define the odometry model which can either be "diff", "omni", "diff-corrected" or "omni-corrected".

In our scenarios, “omni-corrected” is used to define the model of the omnidirectional platform. This model includes parameters such as “odom_alpha1”, “odom_alpha2”, “odom_alpha3”, “odom_alpha4” and “odom_alpha5”. These parameters represent the noise expectation in the estimation of rotation and translation of the odometry. The parameter “base_frame_id” defines the coordinate frame used to represent the robot base and is set to “base_link”. The parameter “global_frame_id” defines the coordinate frame published by the localization. It is set to the “map” coordinate frame.

### 8.1.7. AMCL Transforms

A TF tree path is required from the incoming laser scan to the odometry frame for the AMCL to work. AMCL uses the transformation of the laser scans with respect to the /base_frame of the robot. AMCL latches this transformation with the first laser scan data and considered it as fixed in the future. Therefore, AMCL cannot be applied to a scenario where the laser is moving with respect to the /base_frame of the robot.

As depicted in Figure 8.1, the transformation of the /base_frame to the /map_frame is estimated by AMCL. The transformation of the /odom_frame to the /base_frame represents the pose of the mobile robot as calculated from the wheel encoders in the odometry frame.

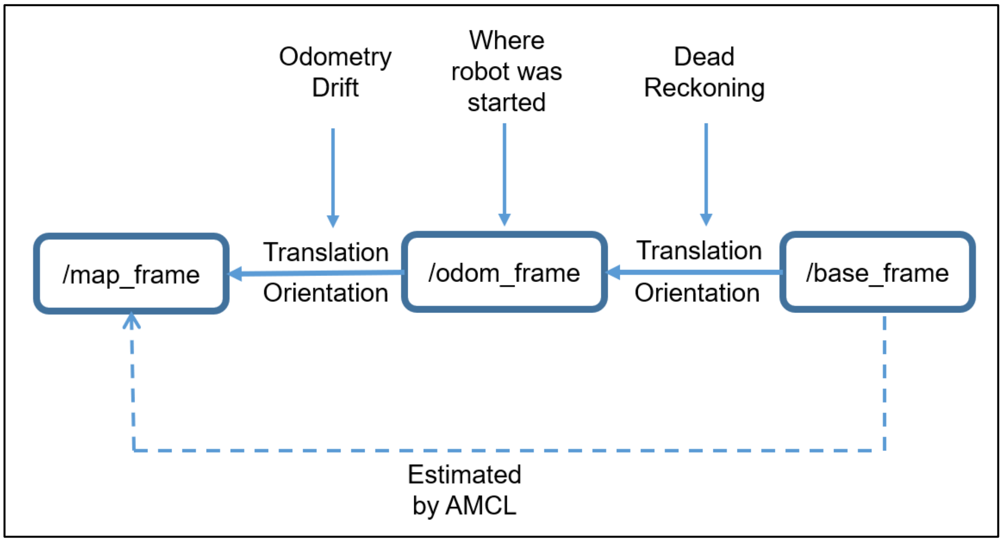


**Figure 8.1: AMCL Map Localization (ROS.org, 2007)**

AMCL introduces the correction for the drift in the odometry by publishing the transformation of the /map_frame to the /odom_frame. Therefore, the transformation of the /map_frame to the /base_frame represents the corrected pose of the mobile robot in the inertial world frame.

While AMCL calculates the transformation from /map_frame to /base_frame, it doesn’t publish this frame but the frame between the /map_frame and /odom_frame. This has several reasons. Firstly, depending on the number of particles and the performance of the computer, the localization update can consume more time. Especially in the initial phase when global localization with many particles is used, this was multiple seconds at the time AMCL was developed, but even today, the delay is non-trivial.

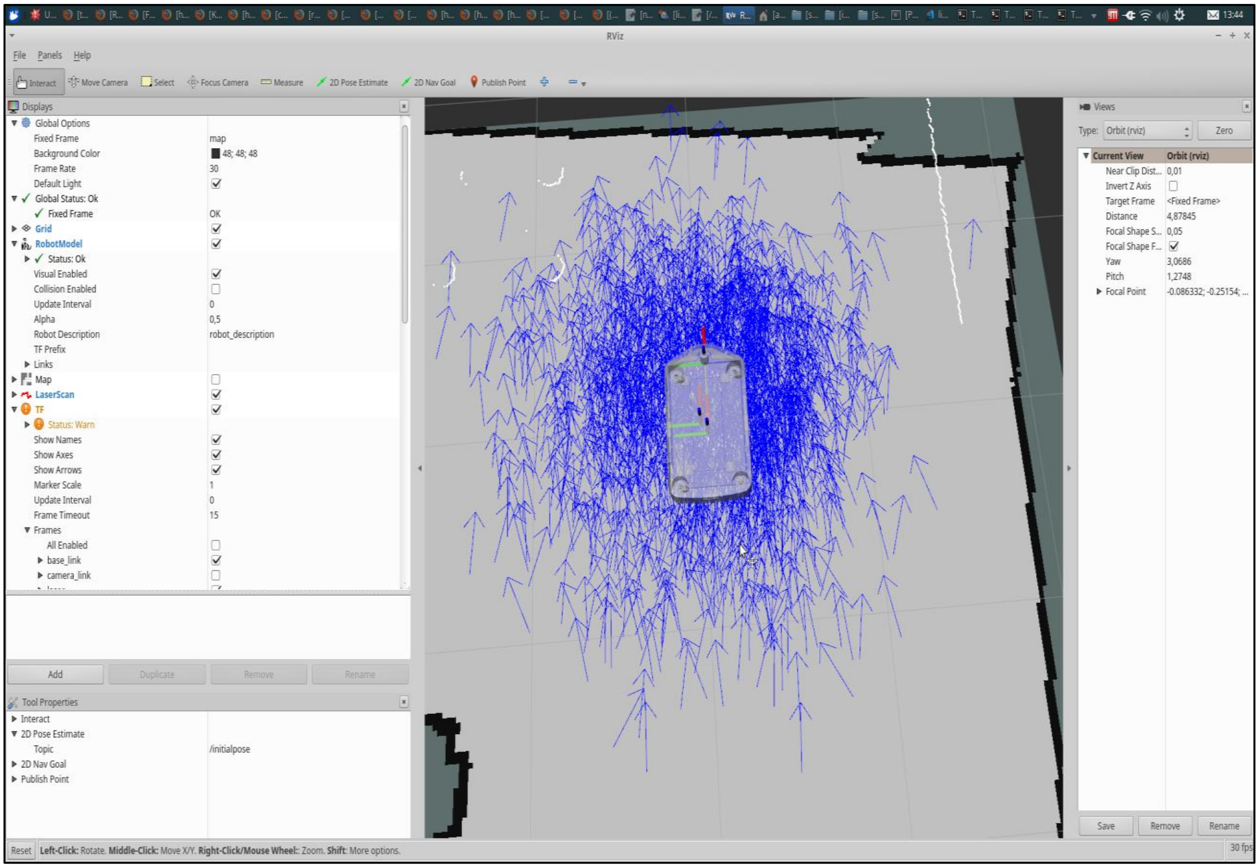
AMCL temporally matches the odometry data with the laser data, that is, it looks up the transformation from /odom_frame to /base_frame at the time the currently used laser scan was made. It then operates on those data while the robot might be moving somewhere else already. When the localization update is finally done, the freshly calculated transformation from /base_frame to /map_frame would already be outdated as the robot moved on.

For this reason, AMCL calculates the transformation from /map_frame to /odom_frame and publishes it. In this way, when other components calculate the transformation between /base_frame and /map_frame by concatenating the transformations from /map_frame to /odom_frame (given by AMCL) and the transformations from /odom_frame to the /base_frame (given by odometry “dead reckoning”) with up-to-date data for the odometry, the error in the overall transformation will be smaller. Namely, the remaining error would be the amount of odometry drift accumulated between the time when the scan was made the localizer worked on and the current time.

ROS transformation frames work with local caches in all components that use transformation data. When a component looks up the relative poses of two coordinate frames, a timestamp is part of the lookup API. When that requested timestamp is way ahead of the data that is stored in the local cache, an error occurs as the extrapolation of data is usually unsafe.

For this reason, AMCL doesn’t publish the transformation from /map_frame to the /odom_frame transformation with the timestamp of the laser scan it worked on but actually “future-dates” it for the transformation frames. The amount of future-dating is configurable, typically it is set to two seconds. This future dating of course also relies on the fact that the transformation from the /map_frame to the /odom_frame transformation only slowly changes. Theoretically, with perfect odometry, it would never change. But in reality, it changes with the rate that odometry error is accumulated. These above arguments provide the justification for the AMCL for not directly publishing the transformation from the /base_frame to the /map_frame, rather only publishing the transformation from /odom_frame to the /map_frame.

The initial step and the final step for the AMCL have been depicted in Figure 8.2 and Figure 8.3 using RVIZ tool. Below Figure 8.2 represents the initial step, that is, when the mobile robot is not localized. This figure describes the initial position of the mobile robot in the map. This visualization is performed in ROS rviz. The robot doesn't know its initial position in the map coordinate frame. Therefore, AMCL spreads the random particles indicating the pose estimate of the mobile robot. These initial pose estimates are indicated by the blue arrows.



| | |
|---|---|
| Global Options | |
| Fixed Frame | map |
| Background Color | ■ 48; 48; 48 |
| Frame Rate | 30 |
| Default Light | ☑ |
| Global Status: Ok | |
| Fixed Frame | OK |
| Grid | ☑ |
| RobotModel | ☑ |
| Status: Ok | |
| Visual Enabled | ☑ |
| Collision Enabled | ☐ |
| Update Interval | 0 |
| Alpha | 0,5 |
| Robot Description | robot_description |
| TF Prefix | |
| Links | |
| Map | ☐ |
| LaserScan | ☑ |
| TF | ☑ |
| Status: Warn | |
| Show Names | ☑ |
| Show Axes | ☑ |
| Show Arrows | ☑ |
| Marker Scale | 1 |
| Update Interval | 0 |
| Frame Timeout | 15 |
| Frames | |
| All Enabled | ☐ |
| base_link | ☑ |
| camera_link | ☐ |

| | |
|---|---|
| Interact | |
| 2D Pose Estimate | |
| Topic | /initialpose |
| 2D Nav Goal | |
| Publish Point | |

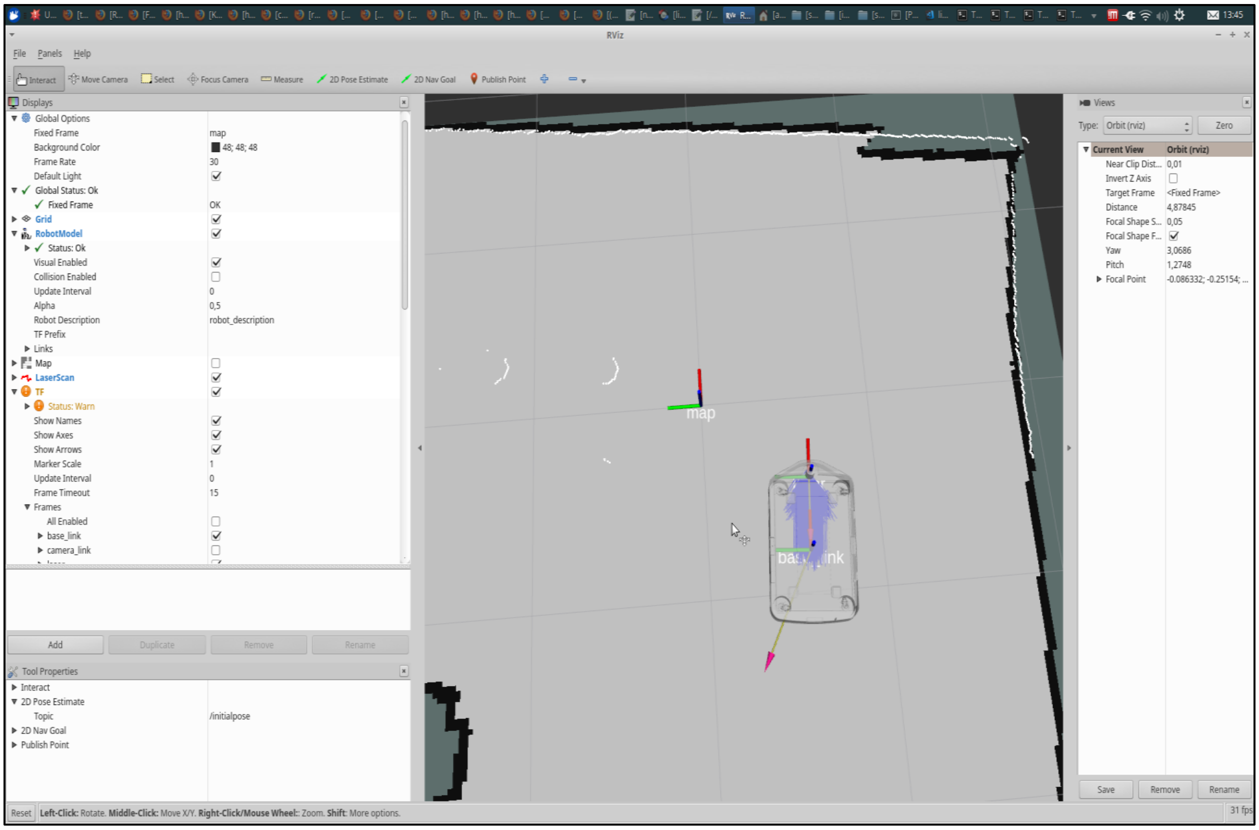
Type: Orbit (rviz)

| | |
|---|---|
| Current View | Orbit (rviz) |
| Near Clip Dist... | 0,01 |
| Invert Z Axis | ☐ |
| Target Frame | <Fixed Frame> |
| Distance | 4,87845 |
| Focal Shape S... | 0,05 |
| Focal Shape F... | ☑ |
| Yaw | 3,0686 |
| Pitch | 1,2748 |
| Focal Point | -0.086332; -0.25154; ... |

**Left-Click:** Rotate. **Middle-Click:** Move X/Y. **Right-Click/Mouse Wheel:**: Zoom. **Shift:** More options.

**Figure 8.2: AMCL depiction using RVIZ before localization**

Below Figure 8.3 represents the final step of the mobile robot localization. This step is performed by moving the robot in the environment so that the weights can be assigned to the particles and then re-sampling takes place. Therefore, after some time, all the randomly spread particles are assembled together. This is indicated by the cloud of the blue arrows in the figure. This condensed cloud indicates the best position estimate of the mobile robot.



| Displays | |
|---|---|
| Global Options | |
| Fixed Frame | map |
| Background Color | 48; 48; 48 |
| Frame Rate | 30 |
| Default Light | ☑ |
| Global Status: Ok | |
| Fixed Frame | OK |
| Grid | ☑ |
| RobotModel | ☑ |
| Status: Ok | |
| Visual Enabled | ☑ |
| Collision Enabled | ☐ |
| Update Interval | 0 |
| Alpha | 0,5 |
| Robot Description | robot_description |
| TF Prefix | |
| Links | |
| Map | ☐ |
| LaserScan | ☑ |
| TF | ☑ |
| Status: Warn | |
| Show Names | ☑ |
| Show Axes | ☑ |
| Show Arrows | ☑ |
| Marker Scale | 1 |
| Update Interval | 0 |
| Frame Timeout | 15 |
| Frames | |
| All Enabled | ☐ |
| base_link | ☑ |
| camera_link | ☐ |

| Tool Properties | |
|---|---|
| Interact | |
| 2D Pose Estimate | |
| Topic | /initialpose |
| 2D Nav Goal | |
| Publish Point | |

Type: Orbit (rviz)

| Current View | Orbit (rviz) |
|---|---|
| Near Clip Dist... | 0,01 |
| Invert Z Axis | ☐ |
| Target Frame | <Fixed Frame> |
| Distance | 4,87845 |
| Focal Shape S... | 0,05 |
| Focal Shape F... | ☑ |
| Yaw | 3,0686 |
| Pitch | 1,2748 |
| Focal Point | -0.086332; -0.25154; ... |

**Left-Click:** Rotate. **Middle-Click:** Move X/Y. **Right-Click/Mouse Wheel:**: Zoom. **Shift:** More options.

**Figure 8.3: AMCL depiction using RVIZ after localization**

## 8.2. RVIZ 2D Manual Pose Estimate

The ROS navigation stack also involves localization for the mobile robot. AMCL is commonly used for the mobile robot localization to get the best estimate of robot pose in the given map coordinate frame.

When AMCL is launched, some initial estimate should be provided about the pose of the mobile robot. This can be achieved by selecting the 2D Pose Estimate button in RVIZ. After choosing this option, the user can draw an arrow on the visualization map in RVIZ, indicating the rough estimate of the position and orientation of the robot, as seen by the user in the real world.

The next step involves moving the mobile robot with the help of the joystick or the controller. This will provide the mobile robot localization in the given map with respect to time and displacement. After certain time, the mobile robot will localize itself in the given map coordinate frame.

However, the option of using the “RVIZ 2D Manual Pose Estimate” is not mandatory. The mobile robot only be moved using the joystick or the controller in the map frames. This might consume more time for the localization to happen. That’s why, “RVIZ 2D Manual Pose Estimate” is used to roughly provide the initial pose to the mobile robot and it saves some time afterward for the localization.

# 9. Non-linear Behavior of Caster Wheels

In the REFILLS Project (H2020 REFILLS Research and Innovation Action, 2017), one of the objectives is to perform scanning of the products kept on the shelves in the supermarket. The omnidirectional platform lifts the scanning unit fitted with a camera, to perform product recognition using computer vision libraries. Images scanned by the camera needs to be clean to get the exact product knowledge. This can only be achieved when the motion of the omnidirectional platform is very steady while the scanning functionality is been performed.

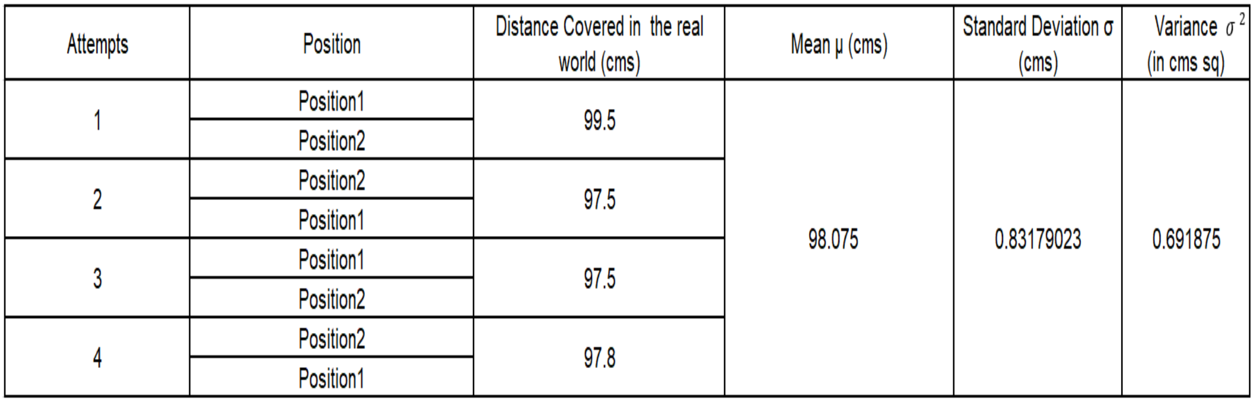
## 9.1. Motivation behind Endpoint Augmentation

The motivation behind implementing the endpoint augmentation at the end of the linear sideways motion is been illustrated in this section.

But while scanning the shelf, the mobile platform performs linear sideways motions. At the end of the linear motion, the platform changes direction by 180°. This leads to a nonlinear behavior as the passive caster wheels that carry the main part of the load, realign to the new movement direction at an undetermined point in time and they can either flip clockwise or counterclockwise. This leads to the mismatch in the camera scans and the knowledge about the product and its location cannot be estimated.

Another motivation that lies behind the endpoint augmentation implementation is the inaccuracy in the distance traveled by the mobile platform while lifting the scanning unit. In this scenario, the experiment is performed with linear side to side motion without the spline trajectory. The mobile platform has been programmed to perform the linear side to side motions for one meter. Position1 and Position2 indicate the extreme endpoints of the linear sideways motion of the mobile platform. The distance traveled by the mobile platform has been recorded for the four trials.

The experimented data illustrating the above scenario are tabulated in Table 9.1.



| Attempts | Position | Distance Covered in the real world (cms) | Mean μ (cms) | Standard Deviation σ (cms) | Variance $\sigma^2$ (in cms sq) |
|---|---|---|---|---|---|
| 1 | Position1<br>Position2 | 99.5 | 98.075 | 0.83179023 | 0.691875 |
| 2 | Position2<br>Position1 | 97.5 | | | |
| 3 | Position1<br>Position2 | 97.5 | | | |
| 4 | Position2<br>Position1 | 97.8 | | | |

**Table 9.1: Statistical data obtained from experiments without spline trajectory**

In Table 9.1, it has been observed that while commanding the omnidirectional platform motion for 1m, the linear distance covered was 98cms (approx.). This shows the inaccuracy and distance lag (by 2cms approx.) caused while performing linear sideways motion with 180° caster wheels flipping at the extreme endpoints of the trajectory.

## 9.2. Ideas for compensating behavior:

By adding a motion controller that forbids instantaneous 180° turns, it should be made sure that the caster wheels always track the motion and don't flip in a chaotic manner. Figure 9.1 depicts the normal sideways motion of the mobile platform. This motion causes undetermined flipping of the caster wheels. This has been compensated by implementing the endpoint augmentation as depicted in Figure 9.1.

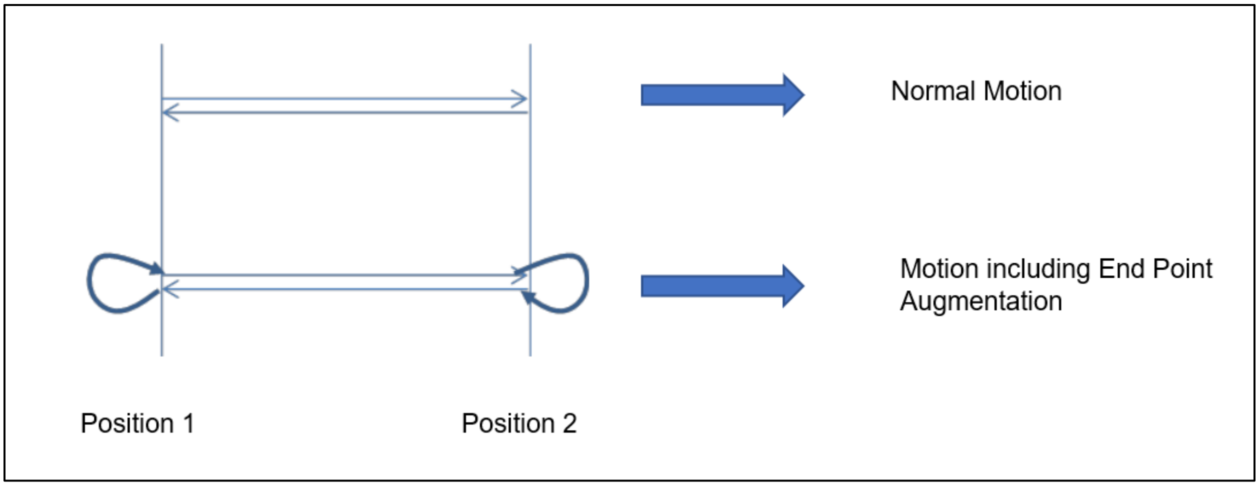


**Figure 9.1: Endpoint Augmentation**

The endpoint augmentation could be done using splines for motion planning. The type of spline shall be one that creates a differentiable curve and the control point shall be chosen so that the motion at the endpoint is large enough to make sure the caster wheels stay aligned but otherwise as small as possible in order to avoid loss of time.

The augmentation via spline shall be offered as an additional topic such as the move_base_simple/goal topic or as an action such as the move_base action and accept a goal point. The first motion commanded shall not be augmented at the start, but when multiple motions are commanded via the same interface. The augmentation shall bend the occurrence of the sharp turns to smoothen the trajectory.

The motion augmentation algorithm may store and access the odometry position to assess the motion that occurred between the last augmented motion and the current commanded one.

# 10. Realization of Spline Motion

Interpolation is a method of generating new data set when few data points in the range are known. There are two categories of interpolation:

- **Polynomial Interpolation:** It involves a unique polynomial of degree $n-1$ passing through $n$ data points. Few of these methods are Monomial Basis, Lagrange Basis and Newton's Basis (Leal, 2018). But there are drawbacks, especially when dealing with high degree polynomials. This approach may also induce some oscillating behavior as well.

- **Piece-wise Interpolation:** This method can overcome the above drawbacks by fitting many data points with a low degree of polynomials and thus helps in removing the oscillations.

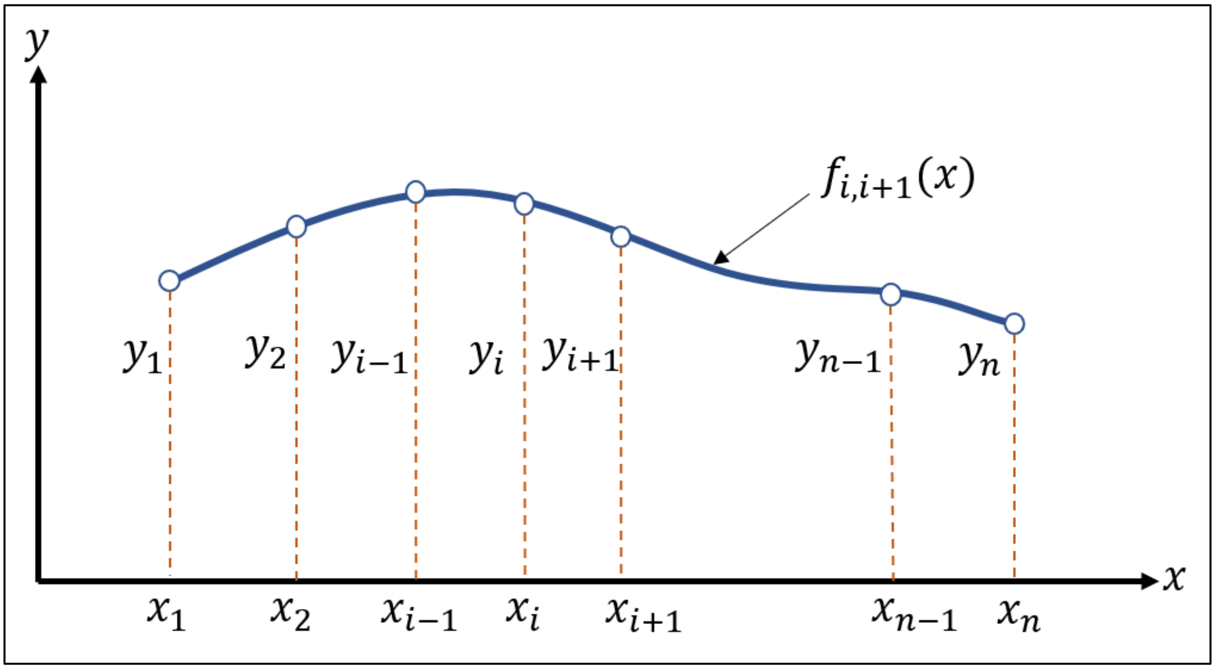


**Figure 10.1: Interpolation Overview**

In Figure 10.1, there are n knots which act as data points $(x, y)$ and $f$ denotes the interpolant between two knots.

A spline is a piecewise polynomial of degree $n$ that is continuously differentiable into $n-1$ times. A cubic spline interpolator is implemented for avoiding a sharp turn during linear side to side motion.

A cubic spline is a piecewise cubic polynomial. At the interpolation nodes, the function, its derivative and second derivates all are continuous. It has fewer tendencies to oscillate between data points. The natural cubic spline will provide zero values for the second derivative at the endpoints.

A cubic spline is the smoothest of all possible interpolating curves because it minimizes the integral of the square of the second derivative.

The derivation is referenced using the sources (Hoschek, Lasser, & Schumaker, 1993), (Jens, Karina, Torsten, & Manfred, 2003) and (Farin, 1996). The reference points $\mu_0, \mu_1, \mu_2 \dots \mu_n$ will be interpolated by using the cubic spline curve.

Let $X(u)$ defines the spline curve with the index as $u$ which varies in the range $u \epsilon [0,1]$. To calculate the parameters of the reference points $u_0, u_1, u_2 \dots u_n$ referring to the interpolation points $P_0, P_1 \dots P_n$ with the condition as stated in equation (10.1).

$$0 = u_0 < u_1 < u_2 < \cdots < u_n = 1 \quad and \quad X(u_i) = P_i \tag{10.1}$$

Different solutions can be obtained by varying $i = 0,1,2,3 \dots n$. The parameters are defined below by equation (10.2).

$$u_i = \frac{1}{s} \sum_{j=0}^{i} d_j, \quad i = 0,1,2 \dots n \tag{10.2}$$

Whereas, the distance between reference points is given below by equation (10.3).

$$d_0 = 0, \; d_i = \sqrt{(P_i - P_{i-1})^2}, \; i = 1,2 \dots n \tag{10.3}$$

The total length of the polygon corresponding to the reference points is given below by equation (10.4).

$$s = \sum_{i=1}^{n} d_j \tag{10.4}$$

$n$ cubic polygon segment $X_i \, where \; i = 0,1,2 \dots n-1$ describe the spline curve connecting neighboring points $P_i and \; P_{i+1}$ with each other.

The length of the interval will be denoted below by equation (10.5).

$$\Delta u_i = u_{i+1} - u_i, \quad i = 0,1,2 \dots n-1 \tag{10.5}$$

Following equation (10.6) gives the parameter description of the curve segment $X_i$

$$X_i(u) = a_i(u-u_i)^3 + b_i(u-u_i)^2 + c_i(u-u_i) + d_i \tag{10.6}$$

With $u \in [u_i, u_{i+1}]$ can be chosen as desired. Due to double differentiation, the function values, the gradients and second derivatives need to match the following equations (10.7) and (10.8) as shown below:

$$\left.\begin{array}{c} X_i(u_i) = X_{i-1}(u_i) \\ X'_i(u_i) = X'_{i-1}(u_i) \\ X''_i(u_i) = X''_{i-1}(u_i) \end{array}\right\} \qquad \textbf{(10.7)}$$

Or

$$\left.\begin{array}{c} X_i(u_{i+1}) = X_{i+1}(u_{i+1}) \\ X'_i(u_{i+1}) = X'_{i+1}(u_{i+1}) \\ X''_i(u_{i+1}) = X''_{i+1}(u_{i+1}) \end{array}\right\} \qquad \textbf{(10.8)}$$

Substituting the above equations (10.7) and (10.8) in the equation (10.6) provide the below equation (10.9).

$$\left.\begin{array}{c} X_i(u_i) = d_i \\ X_i(u_{i+1}) = (\Delta u_i)^3 a_i + (\Delta u_i)^2 b_i + \Delta u_i c_i + d_i \\ X'_i(u_i) = c_i \\ X'_i(u_{i+1}) = 3(\Delta u_i)^2 a_i + 2\Delta u_i b_i + c_i \end{array}\right\} \qquad \textbf{(10.9)}$$

Where the interpolation points are denoted as shown in equation (10.10),

$$X_i(u_i) = P_i\ ,\ X_i(u_{i+1}) = P_{i+1}\ ,\ \ X'_i(u_i) = P'_i \quad and \quad X'_i(u_{i+1}) = P'_{i+1} \qquad \textbf{(10.10)}$$

The unknown gradient in the interpolation points will be calculated and will be denoted by $P'_i\ where\ i = 0,1,2 \ldots n$.

The coefficients are calculated by equations (10.8) and (10.9) and are depicted by the equation (10.11).

$$\left.\begin{array}{c} a_i = \dfrac{\left(2(P_i - P_{i+1}) + \Delta u_i(P'_i + P'_{i+1})\right)}{(\Delta u_i)^3} \\ b_i = \dfrac{\left(-3(P_i - P_{i+1}) - \Delta u_i(2P'_i + P'_{i+1})\right)}{(\Delta u_i)^2} \\ c_i = P'_i \\ d_i = P_i \end{array}\right\} \qquad \textbf{(10.11)}$$

Using Equation (10.6), the cubic splines curve between $P_i$ and $P_{i+1}$ in the interval $[0,1]$ with $i = 0,1,2 \ldots n-1$ is derived as below:

$$\begin{aligned} X_i(u) = & \left(2\frac{(u-u_i)^3}{(\Delta u_i)^3} - 3\frac{(u-u_i)^2}{(\Delta u_i)^2} + 1\right)P_i \\ & + \left(-2\frac{(u-u_i)^3}{(\Delta u_i)^3} + 3\frac{(u-u_i)^2}{(\Delta u_i)^2}\right)P_{i+1} \\ & + \left(\frac{(u-u_i)^3}{(\Delta u_i)^2} - 2\frac{(u-u_i)^2}{\Delta u_i} + u - u_i\right)P'_i \\ & + \left(\frac{(u-u_i)^3}{(\Delta u_i)^2} - \frac{(u-u_i)^2}{\Delta u_i}\right)P'_{i+1} \end{aligned} \quad \textbf{(10.12)}$$

The cubic spline curve is described by equation (10.13) as shown below:

$$X(u) = \begin{cases} X_0(u) & for \quad u \in [u_0, u_1[ \\ & . \\ & . \\ & . \\ X_{n-2}(u) & for \quad u \in [u_{n-2}, u_{n-1}[ \\ X_{n-1}(u) & for \quad u \in [u_{n-1}, u_n[ \end{cases} \quad \textbf{(10.13)}$$

The gradient at the starting point and the endpoint is predefined to avoid discontinuities and ensure the feasibility of the planned trajectory.

The gradient at the starting point and the endpoint is given by the tangents at these points by equation (10.14) as shown below:

$$\left.\begin{aligned} X'(0) = X'_0(u_0) = P'_0 \\ X'(1) = X'_{n-1}(u_n) = P'_n \end{aligned}\right\} \quad \textbf{(10.14)}$$

Using the above-stated description, a package **CubicSplineInterpolator** has been implemented in ROS with C++ as object-oriented programming.

The reference points to be interpolated with the spline curve are defined using the position $x, y$ and orientation $\theta$ around $Z$-axis of the robot base. In this scenario, the robot base orientation must be the same during this application. So, the value of orientation can be set to a constant angle for all reference points.

Different ROS topics are used to analyze and visualize the data stream. These are -

- **initial_pose:** Defines the initial position and orientation of the robot base at the beginning of the trajectory.

- **final_pose:** Defines the final position and orientation of the robot at the end of the trajectory.
- **initial_path:** Depicts the initial trajectory formed by joining all reference points
- **smoothed_path:** Depicts the final smoothened spline trajectory formed by joining all interpolated points

This package **CubicSplineInterpolator** has various configurable parameters:

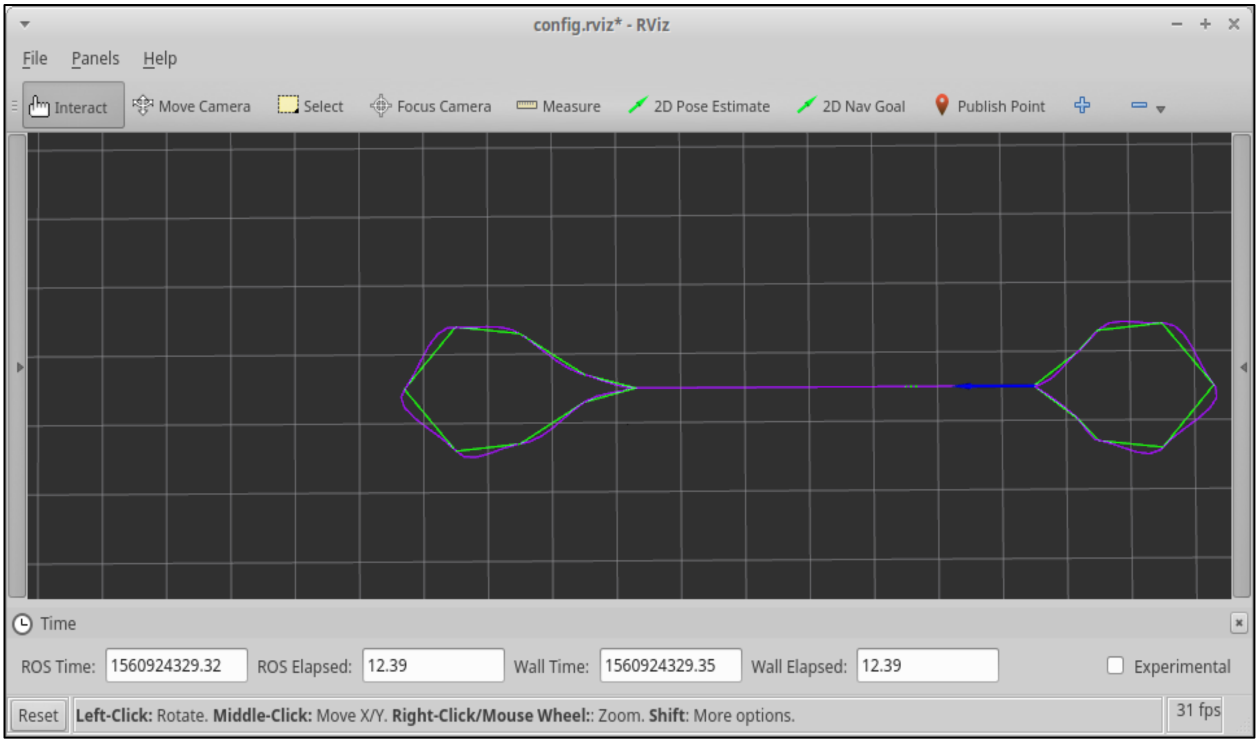
- **pointsPerUnit:** Indicates the number of points to be interpolated between various reference points. The Default value is set to 5.
- **skipPoints:** Represents the number of allowed points that can be skipped to trigger spline trajectory. The Default value is set to 0
- **Condition:** Different conditions can be set for the beginning and the end of the trajectory generation.

The Gradient of points are calculated using interpolation or “pose.yaw” value. Orientations are interpolated to get the intermediate poses of the robot. Finally, by interpolating the reference points, smoothed path points are calculated.

The implementation starts by calculating the cumulating distances between the given reference input points using the equations (10.2), (10.3) and (10.4). Afterward, the coefficients for the cubic equation are calculated using the equation (10.11). This is followed by point gradient calculation as shown in equations (10.7) and (10.8).

These calculated data are further utilized to calculate the input to position X and position Y for the pose of the mobile robot. The reference points are fed as inputs to "/move_base_simple/goal" topic in ROS. This allows the mobile robot to move using the interpolated points of the smooth trajectory, as goal input.

Experiments have been conducted with this spline interpolation package and testing in the real-world environment is performed. Few reference points on the robot trajectory are chosen and are given as inputs to the CubicSplineInterpolator package. This package, in turn, generates a smooth trajectory using the reference points.



ROS Time: 1560924329.32 ROS Elapsed: 12.39 Wall Time: 1560924329.35 Wall Elapsed: 12.39 ☐ Experimental

**Left-Click:** Rotate. **Middle-Click:** Move X/Y. **Right-Click/Mouse Wheel:**: Zoom. **Shift**: More options.

**Figure 10.2: Spline Visualization in RVIZ**

The experiment for the sideways motion of the mobile robot is been depicted in Figure 10.2. This represents the visualization of the actual and the interpolated trajectory in ROS rviz tool. In this figure, the green curve indicates the original trajectory formed by joining the reference points. This curve includes sharp turns which can cause the caster wheels to reorient randomly.

Using the CubicSplineInterpolator package, a cubic spline trajectory is been interpolated between each pair of two reference points. The gradients at the start point and the endpoints are predefined. To ensure the curvature of the trajectory is pretty much smooth, the function value, its first derivative (slope) and its second derivatives must match at the point of the intersection.

Using this approach, CubicSplineInterpolator package generates a new trajectory as shown by the purple color. This package detects the sharp turns in the original trajectory and turns them into smooth curvature ensuring that the caster wheels are not flipped randomly. The blue arrow indicates the initial pose of the mobile robot.

# 11. Laser Scan

In ROS, laser scan data can be read from the /scan topic.

The laser scan data set is an array of geometric points in the polar coordinate system. Each consecutive point differs by a constant angle difference. The major advantage of such a system is that by calculating the average, the measurement noise can be counteracted.

## 11.1. Implementation of the Laser Filter

The laser filters are implemented as ROS nodes in C++. These filters use the actual laser scan as input and perform filter implementation and the new filtered data are published on the different ROS topics. During the later stage, the data are read using this ROS topic and object shape recognition is performed. The laser filters are implemented as plugins. There are two nodes which can be used to implement multiple laser filters:

- **scan_to_scan_filter_chain**: This node can be used to apply a series of laser filters to the laser scan data of type sensor_msgs/LaserScan.

- **scan_to_cloud_filter_chain:** The laser filter takes the laser scan as an input argument, applies laser filters and provides the result in the form of the point cloud. The point cloud data are stored in the form of sensor_msgs/PointCloud.

### 11.1.1. Laser Median Filter

This filter helps in filtering the flickering of the laser scan data while the omnidirectional platform is in motion.

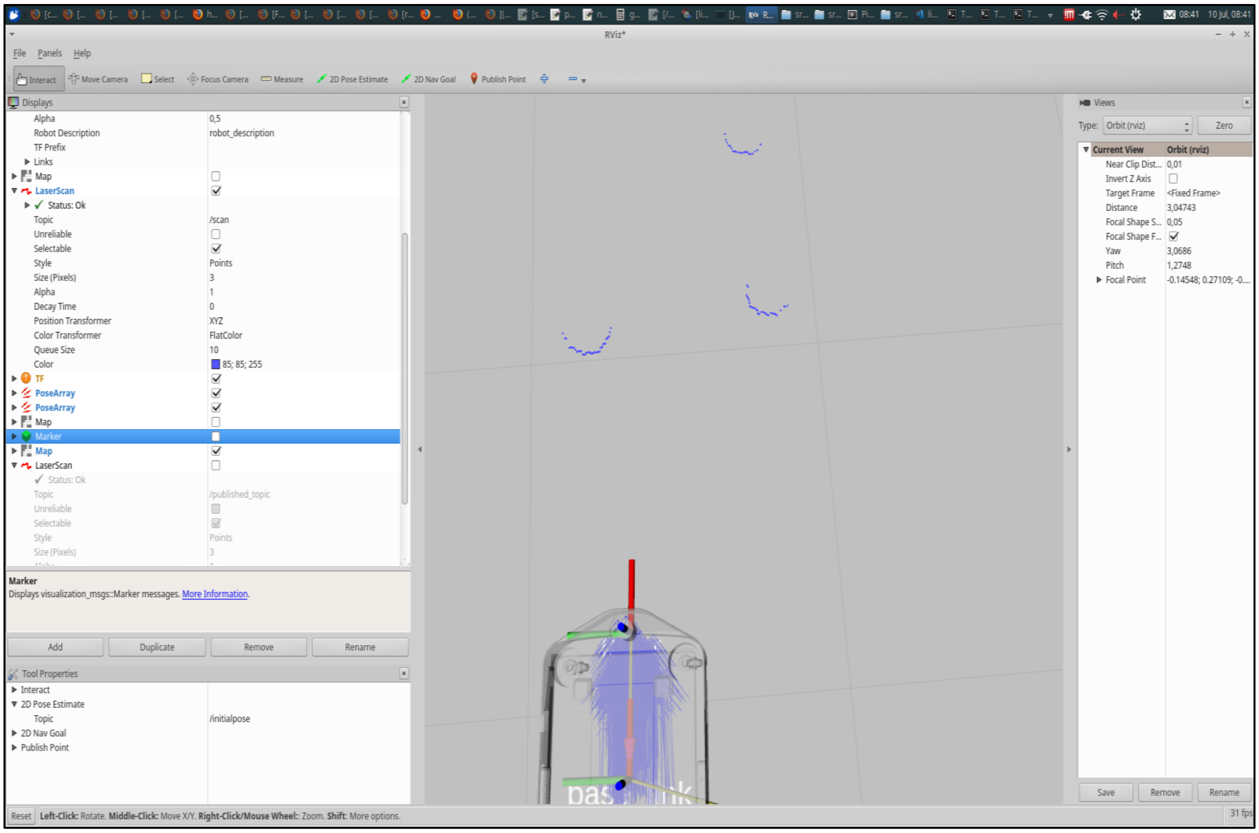
It also helps in retrieving the median value for the entire array of laser scan data. At each array element, the median value is calculated based on the most recent laser scan readings N. In our case, N=10.

For each array element, the median value is given by equation (11.1) as shown below.

$$\left.\begin{array}{c} \mathrm{Y}[\mathrm{i}_1] = \mathrm{median}(\mathrm{X}[\mathrm{i}_{11}], \mathrm{X}[\mathrm{i}_{12}], \mathrm{X}[\mathrm{i}_{13}] \ldots \mathrm{X}[\mathrm{i}_{1\mathrm{N}}]) \\ \mathrm{Y}[\mathrm{i}_2] = \mathrm{median}(\mathrm{X}[\mathrm{i}_{21}], \mathrm{X}[\mathrm{i}_{22}], \mathrm{X}[\mathrm{i}_{23}] \ldots \mathrm{X}[\mathrm{i}_{2\mathrm{N}}]) \\ . \\ . \\ \mathrm{Y}[\mathrm{i}_\mathrm{T}] = \mathrm{median}(\mathrm{X}[\mathrm{i}_{\mathrm{T}1}], \mathrm{X}[\mathrm{i}_{\mathrm{T}2}], \mathrm{X}[\mathrm{i}_{\mathrm{T}3}] \ldots \mathrm{X}[\mathrm{i}_{\mathrm{TN}}]) \end{array}\right\} \quad \textbf{(11.1)}$$

Where, $\mathrm{i}_1, \mathrm{i}_2, \mathrm{i}_3 \ldots \mathrm{i}_\mathrm{N}$ indicates the consecutive readings of the laser scan for one individual array index. This is repeated for all $\mathrm{T}$ elements of the laser scan.

The below Figure 11.1 depicts the visualization of the data from the laser sensor in ROS rviz. The laser sensor is used to indicates the corners of the trolley cart as depicted by blue arcs in the figure. But these laser detection data are not smooth and shows the non-uniformity. To overcome this problem, the median filter for laser data has been implemented.



| | |
|---|---|
| Alpha | 0,5 |
| Robot Description | robot_description |
| TF Prefix | |
| Links | |
| Map | ☐ |
| LaserScan | ☑ |
| Status: Ok | |
| Topic | /scan |
| Unreliable | ☐ |
| Selectable | ☑ |
| Style | Points |
| Size (Pixels) | 3 |
| Alpha | 1 |
| Decay Time | 0 |
| Position Transformer | XYZ |
| Color Transformer | FlatColor |
| Queue Size | 10 |
| Color | 85; 85; 255 |
| TF | ☑ |
| PoseArray | ☑ |
| PoseArray | ☑ |
| Map | ☐ |
| Marker | ☐ |
| Map | ☑ |
| LaserScan | ☐ |
| Status: Ok | |
| Topic | /published_topic |
| Unreliable | ☐ |
| Selectable | ☑ |
| Style | Points |
| Size (Pixels) | 3 |

Marker
Displays visualization_msgs::Marker messages. More Information.

| | |
|---|---|
| Interact | |
| 2D Pose Estimate | |
| Topic | /initialpose |
| 2D Nav Goal | |
| Publish Point | |

Type: Orbit (rviz)

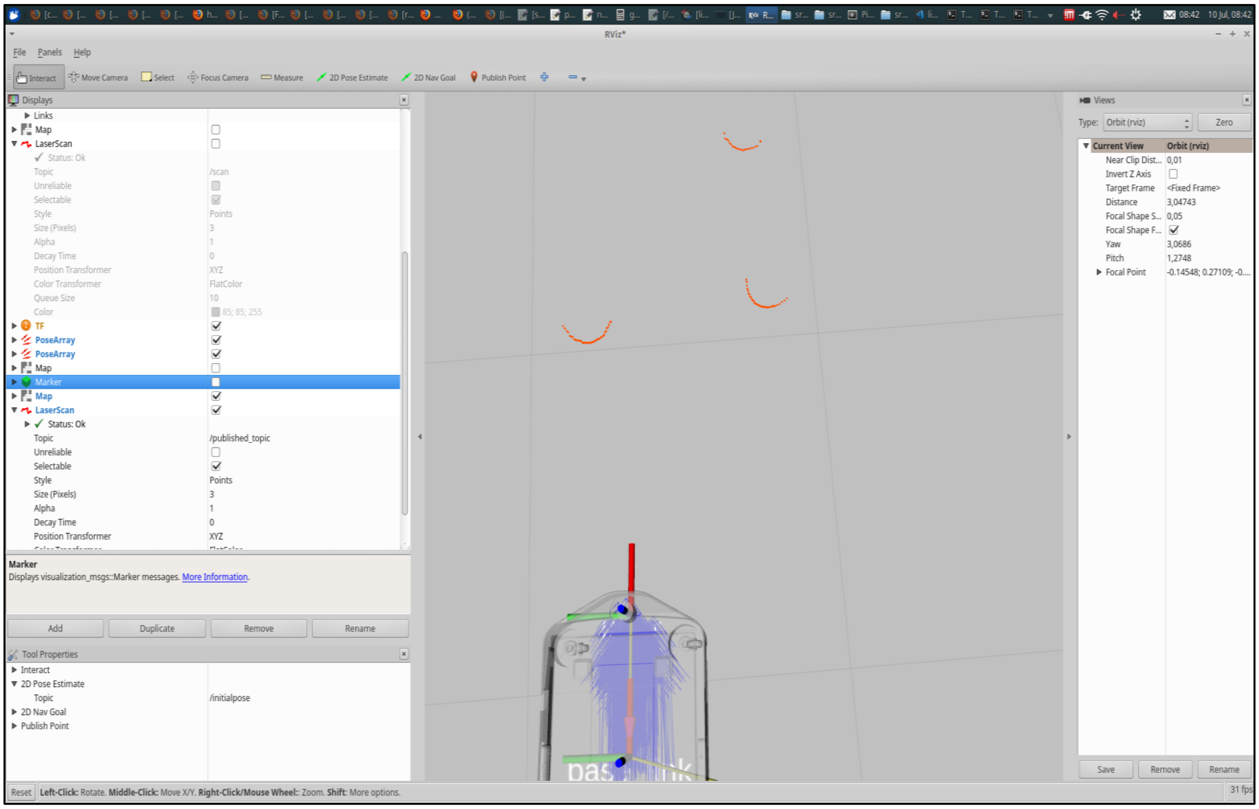
| | |
|---|---|
| Current View | Orbit (rviz) |
| Near Clip Dist... | 0,01 |
| Invert Z Axis | ☐ |
| Target Frame | <Fixed Frame> |
| Distance | 3,04743 |
| Focal Shape S... | 0,05 |
| Focal Shape F... | ☑ |
| Yaw | 3,0686 |
| Pitch | 1,2748 |
| Focal Point | -0.14548; 0.27109; -0.... |

Reset Left-Click: Rotate. Middle-Click: Move X/Y. Right-Click/Mouse Wheel:: Zoom. Shift: More options.

**Figure 11.1: Actual laser scan data without the median filter (visualization in RVIZ)**

The laser median filter implementation yields a more consistent laser reading as depicted in Figure 11.2. This visualization is in ROS rviz. The laser sensor is projected again on the trolley cart to visualize the corners. Here, it can be seen that the laser readings are more consistent and are better aligned. This is been depicted as orange arcs in Figure 11.2.

A time delay occurs when the median filter is been used for the laser sensor data. This time delay causes a lag in the data appearance, but this lag is very minimal as the median calculation is performed after every 10 scans from the laser sensor. This analysis is further useful in the object shape recognition as explained in chapter 13.



| Displays | |
|---|---|
| Links | |
| Map | ☐ |
| LaserScan | ☐ |
| ✓ Status: Ok | |
| Topic | /scan |
| Unreliable | ☐ |
| Selectable | ☑ |
| Style | Points |
| Size (Pixels) | 3 |
| Alpha | 1 |
| Decay Time | 0 |
| Position Transformer | XYZ |
| Color Transformer | FlatColor |
| Queue Size | 10 |
| Color | 85; 85; 255 |
| TF | ☑ |
| PoseArray | ☑ |
| PoseArray | ☑ |
| Map | ☐ |
| Marker | ☐ |
| Map | ☑ |
| LaserScan | ☑ |
| ✓ Status: Ok | |
| Topic | /published_topic |
| Unreliable | ☐ |
| Selectable | ☑ |
| Style | Points |
| Size (Pixels) | 3 |
| Alpha | 1 |
| Decay Time | 0 |
| Position Transformer | XYZ |

**Marker**
Displays visualization_msgs::Marker messages. More Information.

| Tool Properties | |
|---|---|
| Interact | |
| 2D Pose Estimate | |
| Topic | /initialpose |
| 2D Nav Goal | |
| Publish Point | |

Type: Orbit (rviz)

| Current View | Orbit (rviz) |
|---|---|
| Near Clip Dist... | 0,01 |
| Invert Z Axis | ☐ |
| Target Frame | <Fixed Frame> |
| Distance | 3,04743 |
| Focal Shape S... | 0,05 |
| Focal Shape F... | ☑ |
| Yaw | 3,0686 |
| Pitch | 1,2748 |
| Focal Point | -0.14548; 0.27109; -0.... |

**Left-Click:** Rotate. **Middle-Click:** Move X/Y. **Right-Click/Mouse Wheel:**: Zoom. **Shift:** More options.

Figure 11.2: Laser scan with the median filter

### 11.1.2. Laser Scan Shadow Filter

The laser scanner used in our scenario uses a time-of-flight approach, that is, the distance is measured by recognizing the time required by the signal to travel from the scanner laser to the obstacle back to the sensor's photodiode.

The advantage with respect to cheaper laser sensors that use triangulation is, that the precision does not directly depend on the distance. A triangulating sensor would have a linear array of the sensors on the side of the laser emitter and recognize the projection of the laser beam onto the obstacle in a similar way like a camera with only a single line. The problem with that is, the error/uncertainty in the distance rises with the square of the distance. This is not the case with the time of flight approach.

The ray of the laser is not a perfect line but rather a cone with a certain opening angle, so with the distance, the diameter of the laser cone increases slightly. Now when the ray hits the edge of an object, there will be two return signals that the sensor can't discriminate.

The modulation of the laser is a sinus signal. The phase shift of the signal between sending and receiving will define the distance (recognized by something similar to a fast phase-locked-loop). Now if there are two signals coming back are both sinusoidal and of the same frequency. The photosensor will see the “mixture” or the addition of the two signals. The two sinuses are added up. And as they are of the same frequency, the addition signal will just shift in phase but will be of the same frequency. The amplitude will of course differ, with the extreme of the signals “canceling out” at a phase of $\pi$, where there is still a positive amplitude but the signal collapses to a straight line. This is how the shadow points in the middle appear in more detail.

In the shadow filter, the angle between the two points with respect to the laser origin is calculated. If the two points are presented as $T_1$ and $T_2$ and origin of laser scan is $X$. If the perpendicular $\angle XT_1T_2$ violates the minimum and maximum boundary condition, then the neighbor points are ignored. The filter parameters “min_angle” and “max_angle” are used to configure the minimum and maximum boundary conditions for the laser shadow filter respectively.

### 11.1.3. Laser Scan Intensity Filter

This filter is implemented to filter the laser scan depending on the intensity values. The minimum and maximum threshold values can be defined as an input argument. The parameters “lower_threshold” and “upper_threshold” are configured to define the minimum and maximum range for the laser intensity readings.

# 12. Line Extraction Algorithm using Laser

Perception is an important factor allowing mobile robots to drive autonomously. The environment for the mobile robot may be irregular and varying with time. The obstacles are detected through laser sensor reading. But to avoid the obstacle accurately, this knowledge is not enough. We need to know the exact shape and size of the obstacle. Knowledge of the position and orientation of the obstacle helps in avoiding it precisely while traversing the trajectory. The literature survey has been carried out to find the most accurate and suitable segment detection algorithm based on the laser sensor data. The different surveyed algorithms are described below:

## 12.1. Split-and-Merge Algorithm

This is the most popular line extraction method. The most successful implementation of Split-and-merge in the mobile robotics is **Iterative-End-Point-Fit**. This method is a recursive algorithm. The line is defined by the first and last point of the given segment. Afterward, the point which is at maximum separation from the line is determined. This point leads to the generation of two smaller segments. This algorithm is run recursively until the threshold for the maximum distance is attained. This algorithm is explained in detail in chapter 13.2.

## 12.2. Hough Transform

The Hough transform (HT) algorithm involves the extraction of linear features from sequences of consecutive measurements. This is mostly applicable for clusters of collinear points. This is followed by an approximation to retrieve a line out of this cluster. This method is applicable to the global structure. Hough Transform involves selecting the elements with maximum votes from point cloud clusters. This is followed by the calculation of inliers. Then, a line fitting is created using these inliers.

There are a few drawbacks associated with this approach. It is difficult to select an accurate grid size for the accumulator array. It does not basically include noise or uncertainty in the environment.

## 12.3. Line model with Expectation-Maximization

Expectation-Maximization (EM) is a very popular probabilistic approach for detecting the line model using laser scan point clouds. This is used to extract linear as well as a planar structure using 3D data. It starts by randomly generating the line parameters and initializing weights for the points. Using the line model, the weights of the points are assigned, and the line model parameters are recomputed. Meanwhile, there is a consistent check for a number of steps and number of trials. The goal is to provide an iterated sequence of models with increased likelihood estimation.

There are a few drawbacks with the implementation of EM. It can get stuck in local minima and is challenging to select correct initial values.

## 12.4. RANSAC

RANSAC stands for Random Sample Consensus. RANSAC algorithm was introduced for fitting models robustly using the outliers. The main advantage of this method is that it involves generic segmentation. With the preparation of the feature model, this algorithm can be used with several types of features. RANSAC is initiated by selecting two random sample points from a group of points. Then, a line fitting is performed with these two sample points. The distance of other points with respect to this line is computed, an Inlier set of data points is created, and line parameters are recomputed. This procedure is repeated until the maximum number of iterations is reached.

There are a few drawbacks associated with this algorithm. It involves a large number of iteration steps. Moreover, if the error bound is set too high, the inaccuracy of the calculated model increases.

# 13. Implementation of Geometric Shapes Recognition Using Laser

The Split-and-Merge algorithm runs at a faster pace as compared to the Hough Transform. The Spit and Merge algorithm is applicable for structures locally. So, the results produced by this algorithm are locally more consistent as compared to the Hough Transform. Here, the approach for obstacle detection uses the Split-and-Merge algorithm using laser scans. Later it is computed in terms of the global map frame. The Split-and-Merge algorithm also assures the order in which measurements were recorded.

Hough Transform, RANSAC and EM do not utilize the sequential property of the scan angle of the points. These algorithms most often perform line fitting incorrectly with respect to the scan map. This results in the inaccuracy of extracted lines and the complexity of the algorithm increases.

Obstacle extraction is performed in terms of geometric shapes of an object such as a circle. This is done in an asynchronous manner using the data provided by the laser sensor. The polar coordinates obtained from the laser sensor is transformed into the “point cloud”. This point cloud is published in the map coordinate frame and is further used throughout this algorithm. The extraction of geometric shapes using laser data point cloud provides advantages such as simplifying the data structure and reducing the noise measurement.

This chapter is combinedly referenced using the references (Abel, Luis, & Urbano, 2004), (A. Azim and O. Aycard, 2012), (Przybyła, 2017) and (Siegwart, Nguyen, Martinelli, & Tomatis, 2005).

The laser data is used to extract the geometric feature of the circle in terms of the circle radius and the coordinates of the center points. This is shown in the form of equation (13.1).

$$c \triangleq \{r, \overrightarrow{\boldsymbol{p_0}}\} \triangleq \{r, (x_0, y_0)\} \qquad \textbf{(13.1)}$$

## 13.1. Clustering of Laser Point Cloud

Clustering is performed to assemble the neighboring point cloud laser data together. This step ensures that an individual object can be depicted as a point cloud cluster. The target is to create an abstract subset of point cloud data representing an object. The threshold condition for forming a new cluster is given by the equation (13.2).

The distance condition used is:

$$d_{i-1}^{i} < d_{group} + R_i d_p \qquad \textbf{(13.2)}$$

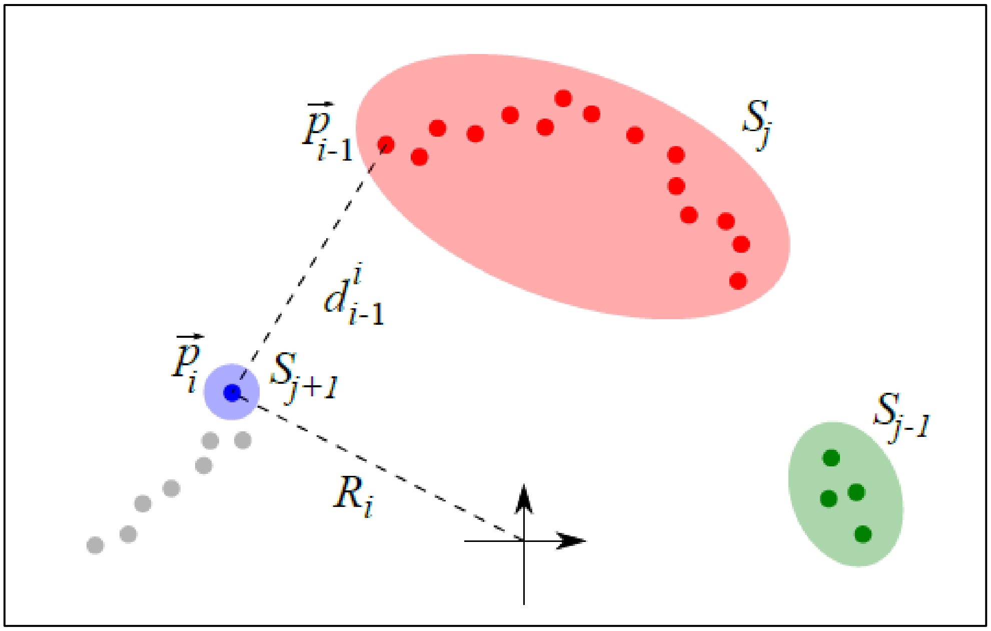


**Figure 13.1: Grouping of the point cloud (Przybyła, 2017)**

With reference to Figure 13.1, the distance between two points $\vec{\boldsymbol{p}}_{\boldsymbol{i}}$ and $\vec{\boldsymbol{p}}_{\boldsymbol{i-1}}$ is given by $d^i_{i-1}$. The variable $d_{group}$ is the user-defined threshold. The variable $R_i$ is the distance of the point in the given map coordinate frame and $d_p$ is the distance proportion. The term $R_i d_p$ indicates the varying threshold check and it depends on the distance of the point cloud data from the reference coordinate frame. This is needed to counteract on the calculation because with far away laser scans, the distance threshold will have more tolerance. A new cluster is generated when the (13.2) equation is not fulfilled.

## 13.2. Splitting of Clusters

The condition for the split between two data subsets is been provided by the user as a threshold value. No data group procedure is carried out when the point count is less than $N_{min}$. The splitting algorithm is based on Iterative End Point Fit. This criterion can be defined by equation (13.3).

$$\boldsymbol{d}_j > d_{split} + R_j d_p \tag{13.3}$$

With reference to the above equation (13.3), the variable $d_{split}$ represents the user-defined threshold for the maximum allowed distance $\boldsymbol{d}_j$ between the leading line and the point lying in the set $S_j$. This is been depicted in Figure 13.2. The range of the furthermost point is denoted by $R_j$. The term $R_j d_p$ indicates the varying threshold check and it depends on the distance of the point cloud data from the reference coordinate frame. This is needed to counteract on the calculation because with far away laser scans, the distance threshold will have more tolerance.

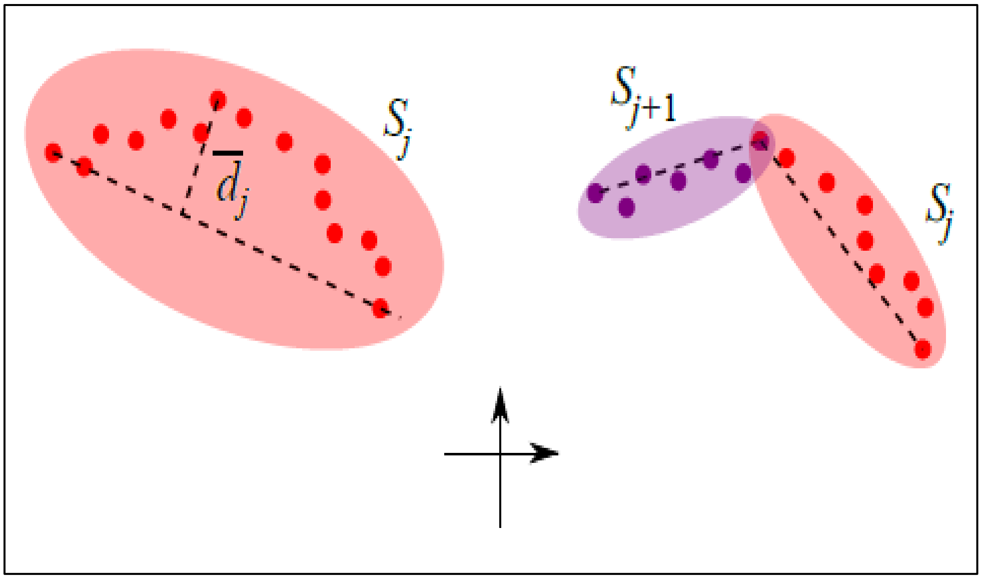


**Figure 13.2: Splitting procedure (Przybyła, 2017)**

In Figure 13.2, there are two subsets sharing the common point of division. The procedure of splitting is continued based on equation (13.3) until there is no further spit possible.

The Iterative End Point Fit algorithm uses the recursion technique to generates the best fitting segment for most of the given points by iteratively splitting the segment. It is depicted in Figure 13.3.

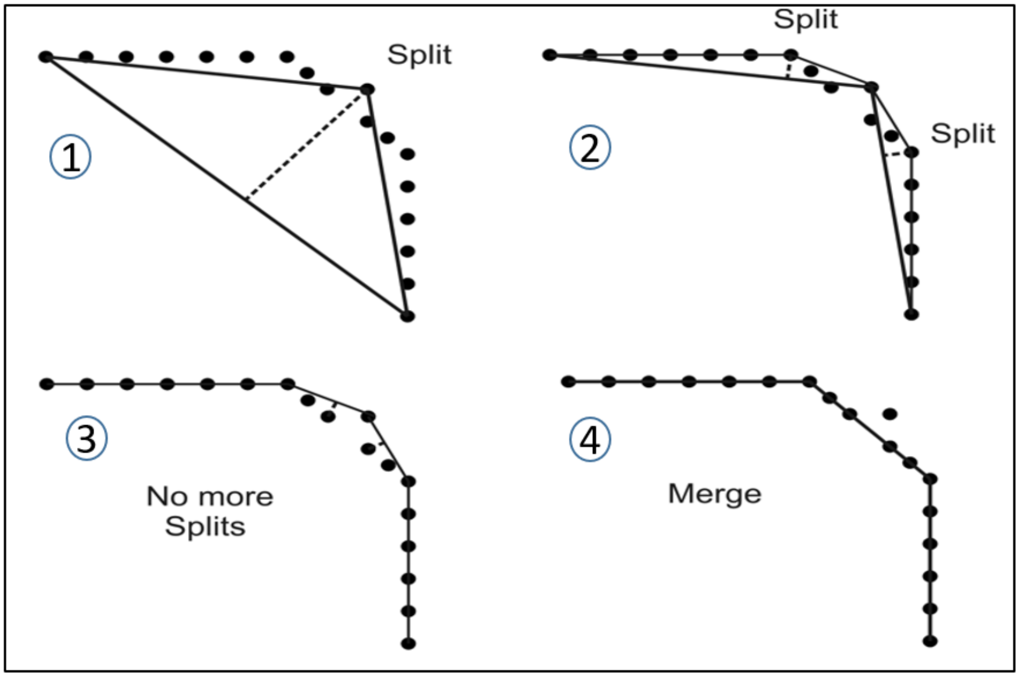


**Figure 13.3: Iterative End Point Fit (Bresenham, 1962)**

With reference to Figure 13.3, Iterative End Point Fit algorithm has depicted in four phases. This algorithm is initiated in step 1. Step 2 and step 3 are the iterative steps performed to promote more splitting within the segment. Finally, step 4, shows the merged segment fitting most of the given points on the line.

This algorithm is initiated by joining the leading line with the extreme endpoints in the point cloud data. Then, the distances of the remaining points from this line are calculated. The farthest point from this line considered, and the two new lines are created by joining this farthest point. With the extreme ends of the original line Again, the same procedure is applied individually to each of the two newly created lines. This algorithm is continued recursively till a new obtained line cannot be further divided depending on the threshold value assigned by the user.

## 13.3. Segmentation

The next step involves the approximation of each subset of point cloud data as a segment $l_m$. The leading line obtained from point cloud data is modeled using the straight-line equation given by:

$$A_m x + B_m y + C = 0 \tag{13.4}$$

Here the variables are presented by $x, y$. The parameters are described $A_m, B_m and\ C_m$ are calculated using the least square regression algorithm. The parameters $A_m\ and\ B_m$ can be solved arbitrarily considering $C_m \neq 0$. Moore-Penrose matrix pseudoinverse is presented by symbol **†** and is used to calculate the parameters $A_m\ and\ B_m$.

$$\begin{bmatrix} A_m \\ B_m \end{bmatrix} = \begin{bmatrix} x_1 & y_1 \\ \vdots & \vdots \\ x_{N_m} & y_{N_m} \end{bmatrix}^{\dagger} \begin{bmatrix} -C_m \\ \vdots \\ -C_m \end{bmatrix} \tag{13.5}$$

## 13.4. Merging of Segments

The next steps involve merging procedure for segments which are very close to each other. This will result in merging the segments as a single segment.

The first check is the **neighboring check** based on the below criterion. $d_{merge}$ is the threshold value defined by the user for merging. $d_0$ is the distance between the closest points on two different segments.

$$d_0 < d_{merge} \tag{13.6}$$

The second check is **collinearity check** based on if equation (13.6) is satisfied and two segments are within the collinearity condition given by equation (13.7). Here the variable $d\_spread$ defines the threshold value defined by the user. The parameters $d_1, d_2, d_3, d_4$ represents the distance between the extreme points of the segment and the leading line.

$$d_{spread} > \max(d_1, d_2, d_3, d_4) \tag{13.7}$$

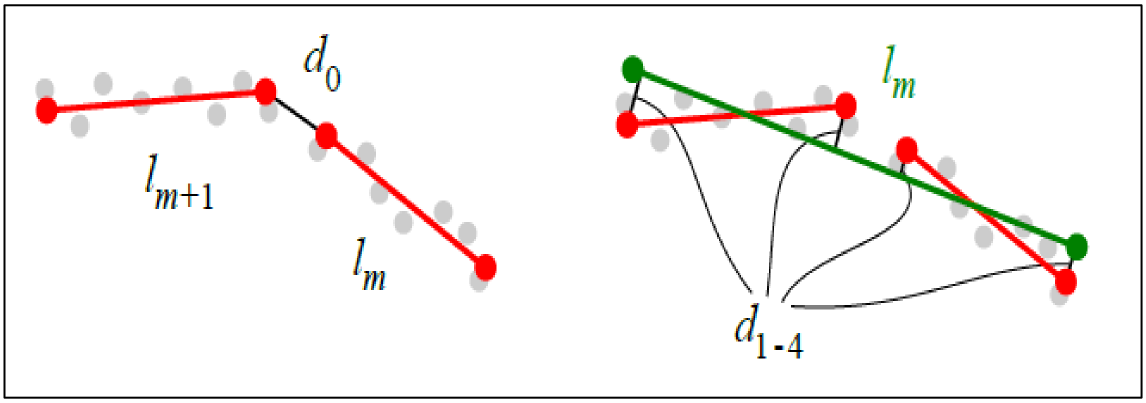


**Figure 13.4: Depiction for neighboring and collinearity checks (Przybyła, 2017)**

## 13.5. Circle Extraction

The information from the segment is utilized to extract the circular obstacles in the robot workspace. The Iterative End Point Fit algorithm is followed by least square regression. This results in the segment fitting most of the laser point cloud data. Then, the merging of the segments takes place according to the neighborhood check along with the collinearity check. The neighborhood check ensures the merging of the two neighboring segments and it depends on the threshold defined by the user. The collinearity check is performed along with the neighborhood check to ensure that the merging segments are collinear as well.

For each of the obtained segments, the extreme points are joined together to get the base of an equilateral triangle, with its center in the direction opposite to the center of the segment. Cross product is used to find the orientation of the segment. The points must be defined in the laser sensor coordinate frame. The radius of the circle is given by equation (13.8). $r_n$ denotes the radius and $\overrightarrow{\boldsymbol{l_n}}$ denotes the segment vector.

$$r_n = \frac{\sqrt{3}}{3}\overrightarrow{\boldsymbol{l_n}} \tag{13.8}$$

Equation (13.9) describes the calculation of the radius vector for the circumcircle of the triangle. $\vec{\boldsymbol{n}}_{\boldsymbol{n}}$ represents the segment normal vector towards the origin.

$$\vec{\boldsymbol{p}}_{\boldsymbol{0n}} = \frac{1}{2}(\vec{\boldsymbol{p}}_{\boldsymbol{1n}} + \vec{\boldsymbol{p}}_{\boldsymbol{2n}} - r_n\vec{\boldsymbol{n}}_{\boldsymbol{n}}) \tag{13.9}$$

The calculation of the circumcircle radius and the center coordinates is depicted in Figure 13.5 as shown below.

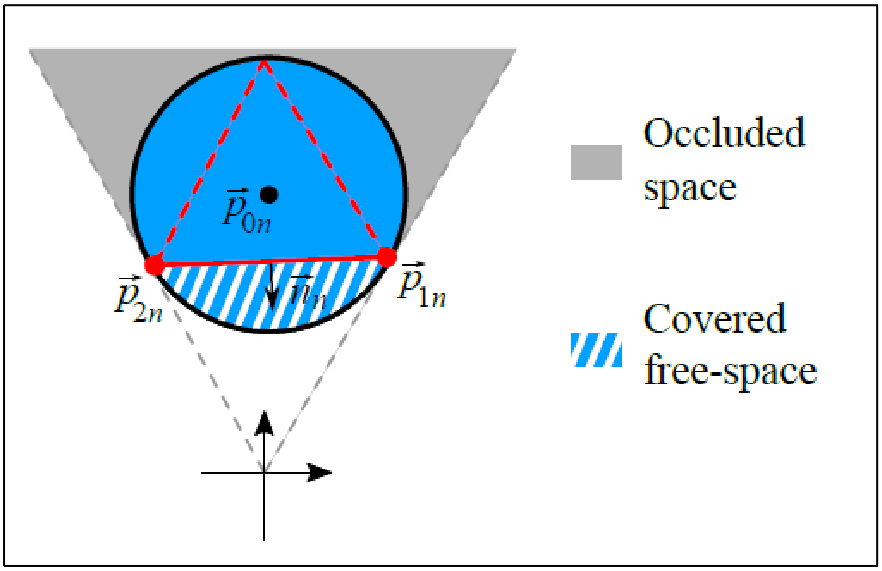


**Figure 13.5: Circle Extraction (Przybyła, 2017)**

The user can also define the margin for the inflation of the circle. The maximum and minimum circle radius can also be set by the user to filter and visualize the desired objects.

### 13.6. User-Defined Messages

There are three types of user-defined messages. They are the CircleObstacle.msg, Obstacles.msg and SegmentObstacle.msg files.

CircleObstacle.msg file contains variables such as center which stores the center of the circle as a Point. It also has the variable true_radius to store the circle radius of type Float.

SegmentsObstacle.msg file contains first_point and last_point as variables of Point type.

Obstacles.msg contain the variables segments and circles. They are an array of SegmentObstacle and CircleObstacle type.

## 14. ROS Navigation Stack

The ROS Navigation stack helps in autonomously driving the mobile robot in the real world. It accepts odometry, laser sensor data and a goal pose of the robot as inputs and returns the safe velocity commands to be sent to the robot. A motion controller must be implemented for navigating the robot in the given environment.

The overview of the ROS navigation stack is depicted in Figure 14.1. The ROS move_base module is used at a high-level to perform robot navigation. It is implemented using ROS actions. A Global and a local planner are combined by move_base to navigate to a goal position and to the desired orientation. ROS move_base also maintains the costmap (refer section 14.1) for the local planner and the global planner. The robot can also be directly sent to the goal position using RVIZ. RVIZ provides an option “2D Nav Goal”, where the user can directly indicate the goal position and orientation of the robot by drawing an arrow in the visualization map.

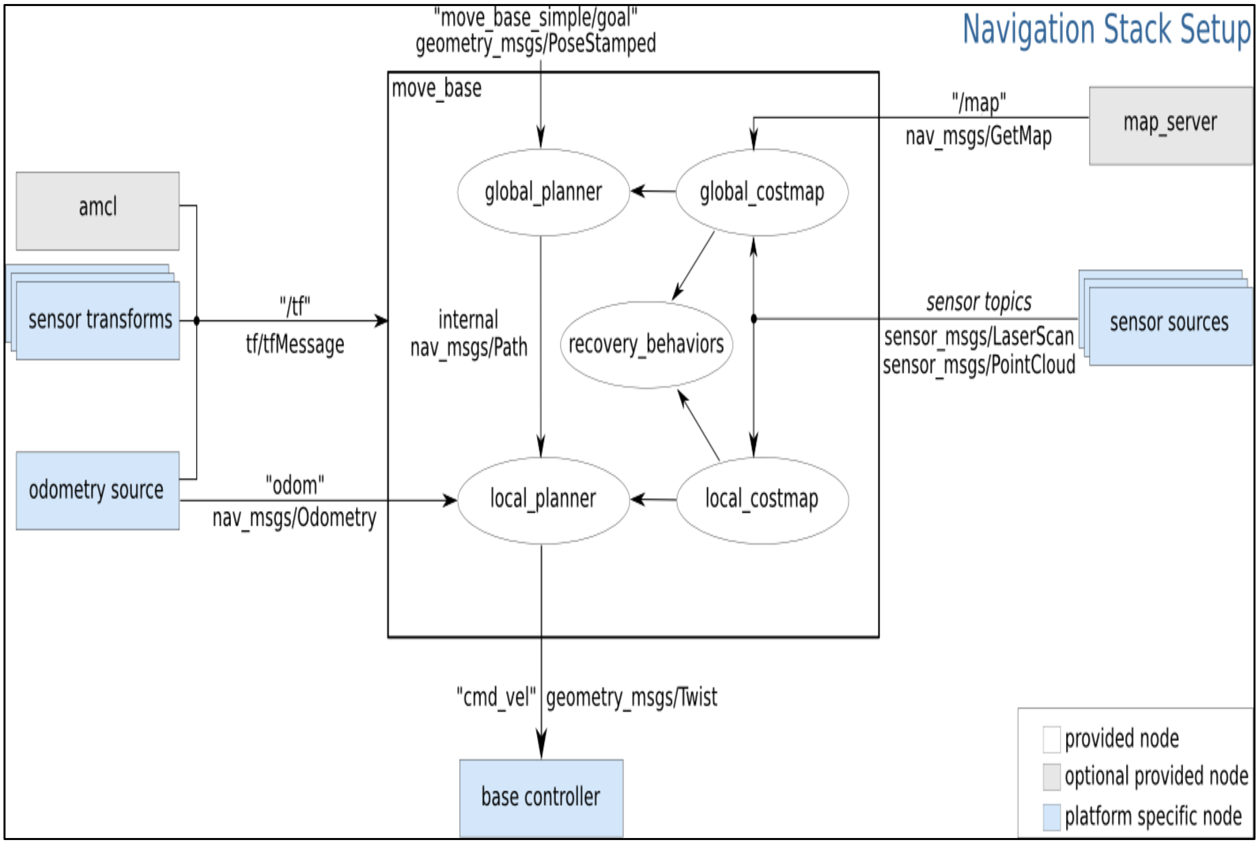


**Figure 14.1: Navigation Stack Nodes and Topics/Actions Overview (ROS ORG, 2007)**

Figure 14.2 represents the interface for the nav_core. ROS package “nav_core” is used by the move_base node for navigating to the goal position. This ROS package utilizes the global_planner, local_planner and recovery_behaviours. These interfaces are described in detail below.

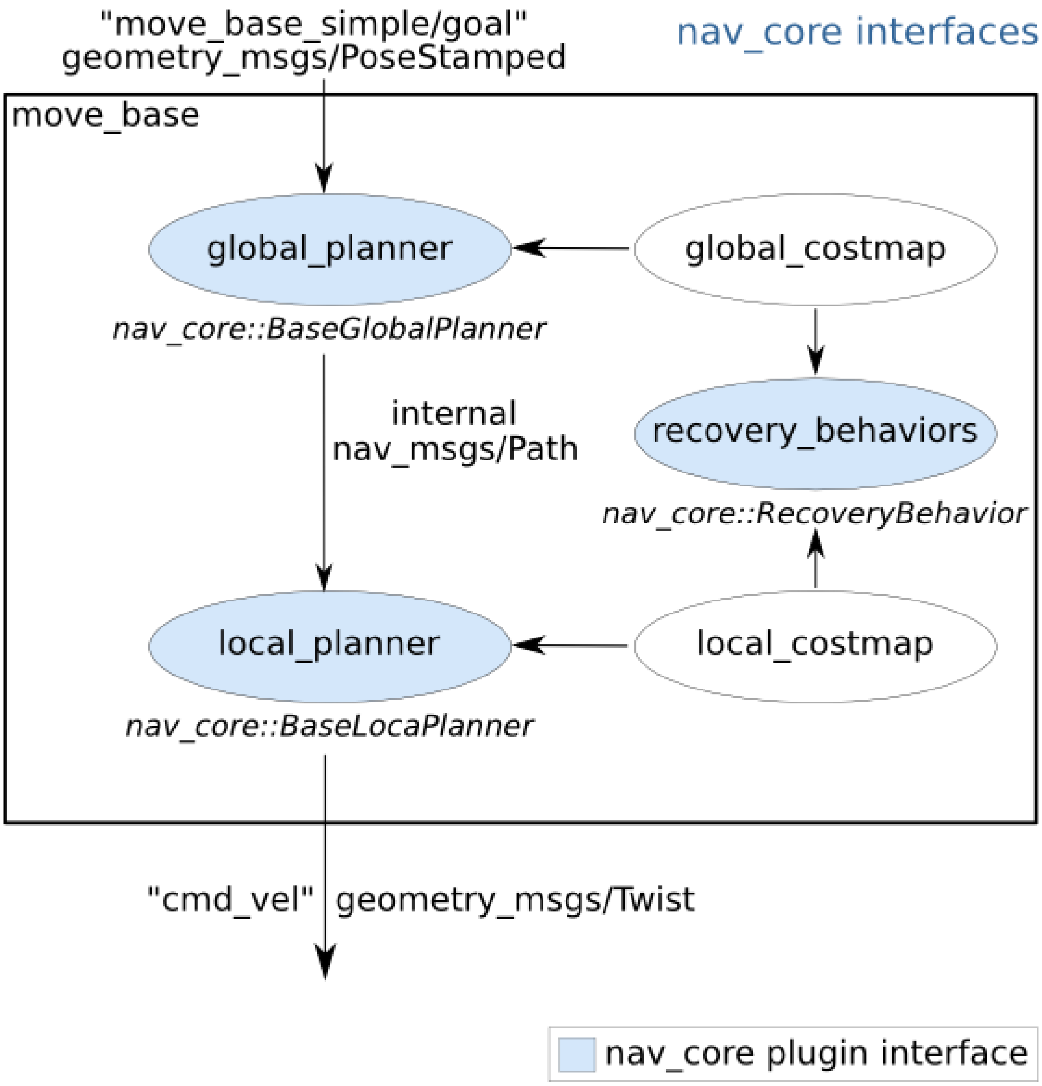


Figure 14.2: Nav_Core interface (ROS ORG, 2007)

## 14.1. Cost Map

Using the sensor data, a 2D/3D occupancy grid of the data is built. Then costs in the 2D costmap are inflated based on the occupancy grid and inflation radius as defined by the user. The cost value of the cell in the costmap can be one among 255 different values. Each cell can be either occupied, free or unknown.

For accurately navigating the mobile robot, the costmap must be set and configured in the correct manner. The configuration of the costmap involves varied parameters such as:

- **footprint:** The Footprint defines the contour of the mobile robot. In general, the footprint is kept slightly larger than the real contour of the mobile robot.
- **expected_update_rate:** Defines the rate at which the data are published by the sensor.

- **transform_tolerance:** Defines tolerance allowed while transforming between two coordinates frames.
- **map_update_rate**: Configures the map update rate. It must be configured low if the robot lacks processing speed.
- **publish_frequency:** Involves the visualization of a large map in RVIZ by setting it to a lower value.

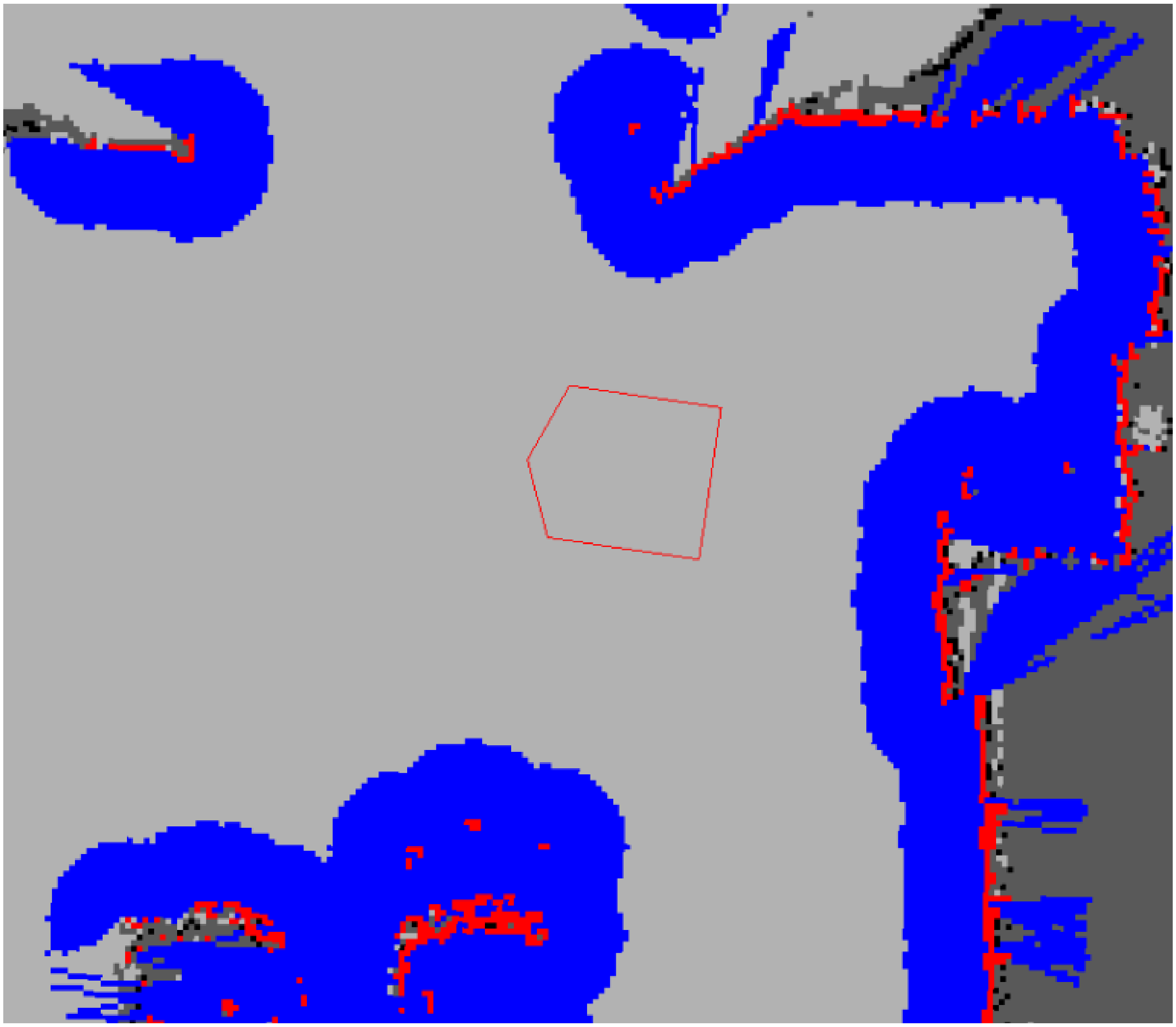


**Figure 14.3: 2D Costmap (ROS WIKI, 2007)**

The above Figure 14.3 depicts the visualization of the cost2D maps using the ROS rviz tool. The obstacles in the costmap are represented by the red cells, the blue cells show the inflation of the obstacle and the red polygon represents the footprint of the robot. To avoid the collision, it is made sure that the robot footprint should not intersect with the red cells and the center of the robot should not cross the blue cell.

## 14.2. Local Planner

The local planner such as dwa_local_planner is used when the robot has a reasonable limit for the acceleration. But if the robot has low limits on acceleration, the base_local_planner can be used. Acceleration limits must be correctly configured while using these local planners. The parameter “vx_samples” must be configured between 8 and 15 while using dwa_local_planner due to reasonable acceleration limits. The parameter “sim_granularity” can be increased to a few cycles if the CPU has low resolution.

## 14.3. Global Planner

It helps in computing a trajectory, given the goal pose of the robot in the world frame.

## 14.4. Recovery Behavior

ROS plugins are used to implement different recovery behaviors. The various recovery behaviors are:

- **clear_costmap_recovery:** It is a recovery behavior, the costmaps revert to the static maps and in return clears out space in the navigation stack’s costmap.
- **rotate_recovery:** This behavior provides the complete rotation by 360° of the robot causing space clearance.

## 14.5. Base Controller

The ROS message type geometry_msgs/Twist is used to send the velocity command by publishing the desired velocity in the cmd_vel topic. The commands are accepted in the form of $x, y \; and \; \theta$ indicating the position and orientation for the mobile robot base. These commands are converted into motor commands to allow the motion of the mobile robot as per the desired command.

ROS nav_pcontroller includes a simple P controller for the motion of the holonomic mobile base platform. This controller uses several configuration parameters such as “xy_tolerance” and “th_tolerance” to define the permissible tolerance in the position and orientation respectively while reaching the goal point. Moreover, maximum velocity and acceleration limits can be set using the parameters “vel_ang_max”, “vel_lin_max”, “acc_ang_max” and “acc_lin_max” respectively.

# 15. Implementation of Autonomous Platform Docking

One of the objectives of REFILLS Project (H2020 REFILLS Research and Innovation Action, 2017) involves autonomous refilling of items. In this case, the items must be transported to the shelves using the trolley cart. This trolley cart has to be lifted by the omnidirectional platform so that the products can be later organized at the desired location.

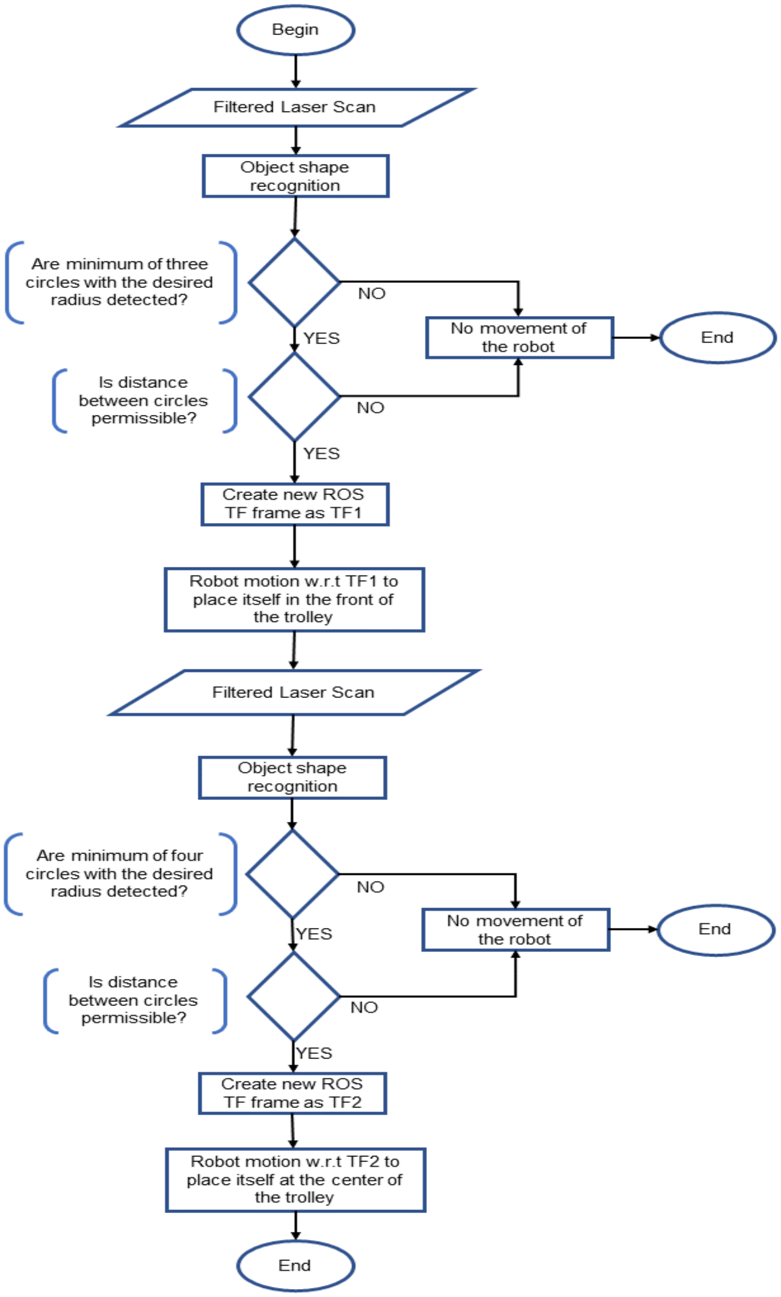
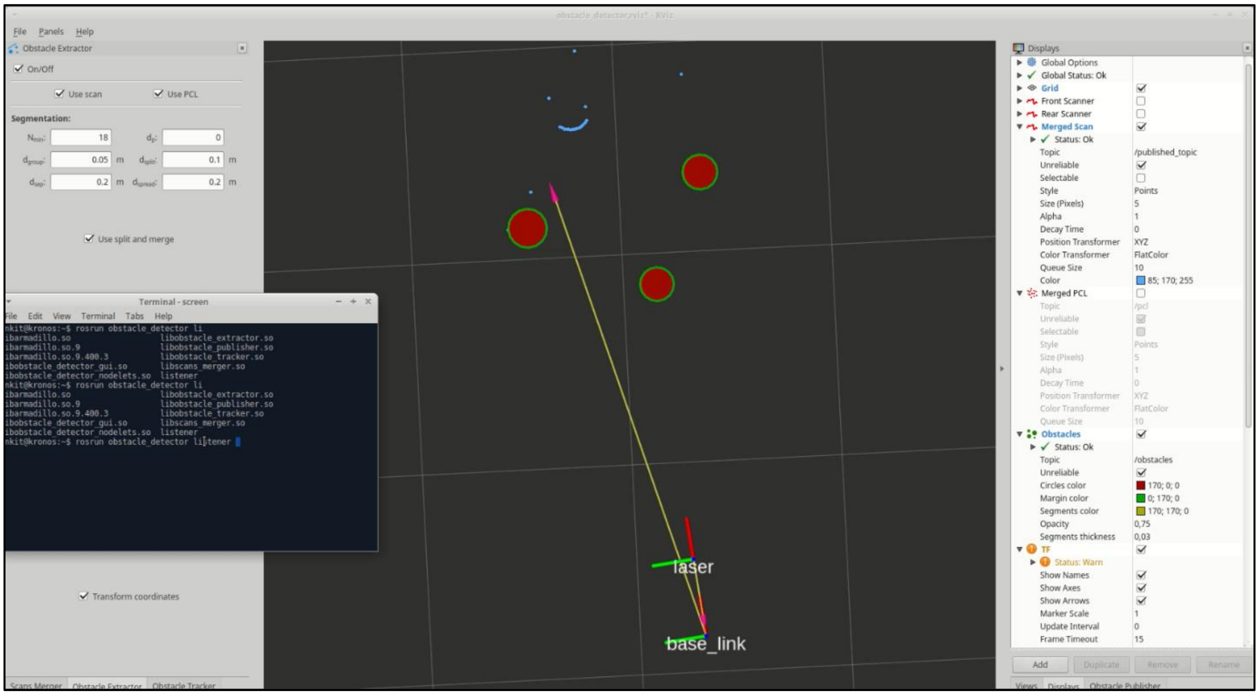

Begin
Filtered Laser Scan
Object shape recognition
Are minimum of three circles with the desired radius detected?
NO
YES
No movement of the robot
End
Is distance between circles permissible?
NO
YES
Create new ROS TF frame as TF1
Robot motion w.r.t TF1 to place itself in the front of the trolley
Filtered Laser Scan
Object shape recognition
Are minimum of four circles with the desired radius detected?
NO
YES
No movement of the robot
End
Is distance between circles permissible?
NO
YES
Create new ROS TF frame as TF2
Robot motion w.r.t TF2 to place itself at the center of the trolley
End

Figure 15.1: Flowchart for the docking application

In order to perform lifting of the trolley cart, a docking application has been implemented in ROS using C++ as object-oriented programming. This platform is detected using the laser sensor mounted in front of the mobile robot. The above flowchart Figure 15.1 represents the systematic flow of the control for the complete application. This implementation makes use of the object shape recognition as explained in chapter 13 to recognize the corners of the trolley base as circles. Cylindrical 3D printed hardcases are mounted at four corners (around the wheels) of the trolley base so that it can be easily recognized by the laser.

The cylindrical surfaces from the trolley corner are recognized as circles along with the radius and center coordinates. These data are then used to calculate the distance between the circles indicating the distance between the corners of the trolley in the real world. Once the calculations are suited as per the trolley dimensions, the ROS TF is created which acts as a reference for the motion of the mobile platform. In the case where there is a mismatch between the calculated data and real-world trolley dimension, the mobile platform stays still and no further motion is triggered. In run time, the circular shape of the cylindrical object is detected, as shown in Figure 15.2. Then their position is calculated with respect to the map frame.

In the below Figure 15.2, the three red circles indicate the three corners of the trolley base. The laser cannot be projected onto the fourth corner at this position and orientation of the mobile robot and hence it is not visible yet. The corners of the trolley base are detected as a circle with respect to the map coordinate frame.



☑ On/Off

☑ Use scan ☑ Use PCL

Segmentation:

$N_{min}$: 18 $d_p$: 0
$d_{group}$: 0.05 m $d_{split}$: 0.1 m
$d_{sep}$: 0.2 m $d_{spread}$: 0.2 m

☑ Use split and merge

☑ Transform coordinates

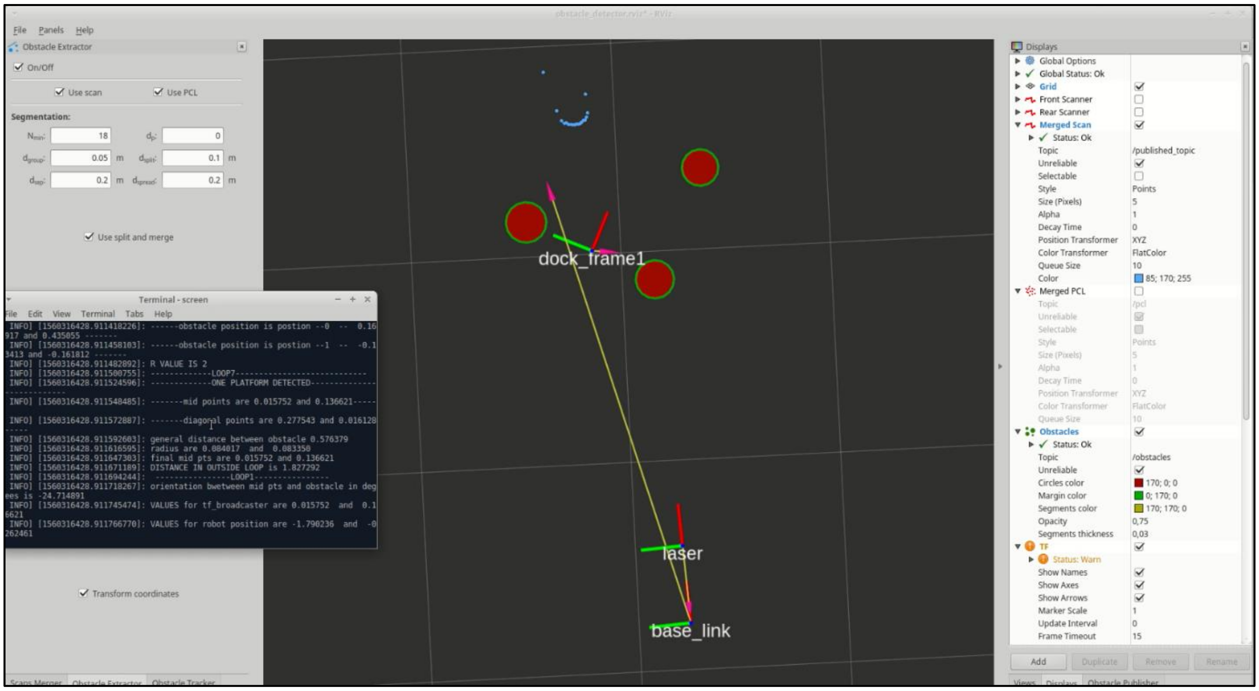
```
nkit@kronos:~$ rosrun obstacle_detector li
ibarmadillo.so                  libobstacle_extractor.so
ibarmadillo.so.9                libobstacle_publisher.so
ibarmadillo.so.9.400.3          libobstacle_tracker.so
ibobstacle_detector_gui.so      libscans_merger.so
ibobstacle_detector_nodelets.so  listener
nkit@kronos:~$ rosrun obstacle_detector li
ibarmadillo.so                  libobstacle_extractor.so
ibarmadillo.so.9                libobstacle_publisher.so
ibarmadillo.so.9.400.3          libobstacle_tracker.so
ibobstacle_detector_gui.so      libscans_merger.so
ibobstacle_detector_nodelets.so  listener
nkit@kronos:~$ rosrun obstacle_detector listener
```

| | |
|---|---|
| Global Options | |
| Global Status: Ok | |
| Grid | ☑ |
| Front Scanner | ☐ |
| Rear Scanner | ☐ |
| Merged Scan | ☑ |
| Status: Ok | |
| Topic | /published_topic |
| Unreliable | ☑ |
| Selectable | ☐ |
| Style | Points |
| Size (Pixels) | 5 |
| Alpha | 1 |
| Decay Time | 0 |
| Position Transformer | XYZ |
| Color Transformer | FlatColor |
| Queue Size | 10 |
| Color | 85; 170; 255 |
| Merged PCL | ☐ |
| Topic | /pcl |
| Unreliable | ☑ |
| Selectable | ☐ |
| Style | Points |
| Size (Pixels) | 5 |
| Alpha | 1 |
| Decay Time | 0 |
| Position Transformer | XYZ |
| Color Transformer | FlatColor |
| Queue Size | 10 |
| Obstacles | ☑ |
| Status: Ok | |
| Topic | /obstacles |
| Unreliable | ☑ |
| Circles color | 170; 0; 0 |
| Margin color | 0; 170; 0 |
| Segments color | 170; 170; 0 |
| Opacity | 0,75 |
| Segments thickness | 0,03 |
| TF | ☑ |
| Status: Warn | |
| Show Names | ☑ |
| Show Axes | ☑ |
| Show Arrows | ☑ |
| Marker Scale | 1 |
| Update Interval | 0 |
| Frame Timeout | 15 |

Figure 15.2: Circular trolley corners detection

The trolley base is rectangular in shape and can be docked only from two edges out of all four edges, that is, the longer edge of the rectangular base. The distance between the detected object is verified with the permissible distance allowed for docking under the trolley base. If the distance matches, the midpoint coordinates of the docking edge is calculated. The mid-point coordinates of the diagonally opposite circular object are also calculated.

Now, with reference to Figure 15.3, a TF coordinate frame in ROS is created and published, with the origin at this midpoint and orientation along the vector from the midpoint of the detected docking edge to the midpoint of the detected diagonal. A ROS node is created for publishing the TF of base_link frame with respect to the map, to monitor the current position of the robot in map frame. This newly created TF is named "dock_frame1" and is created with respect to "map" frame. This frame has X-axis pointing towards the inside of the trolley base.



☑ On/Off

☑ Use scan ☑ Use PCL

Segmentation:

$N_{min}$: 18 $d_p$: 0
$d_{group}$: 0.05 m $d_{split}$: 0.1 m
$d_{sep}$: 0.2 m $d_{spread}$: 0.2 m

☑ Use split and merge

☑ Transform coordinates

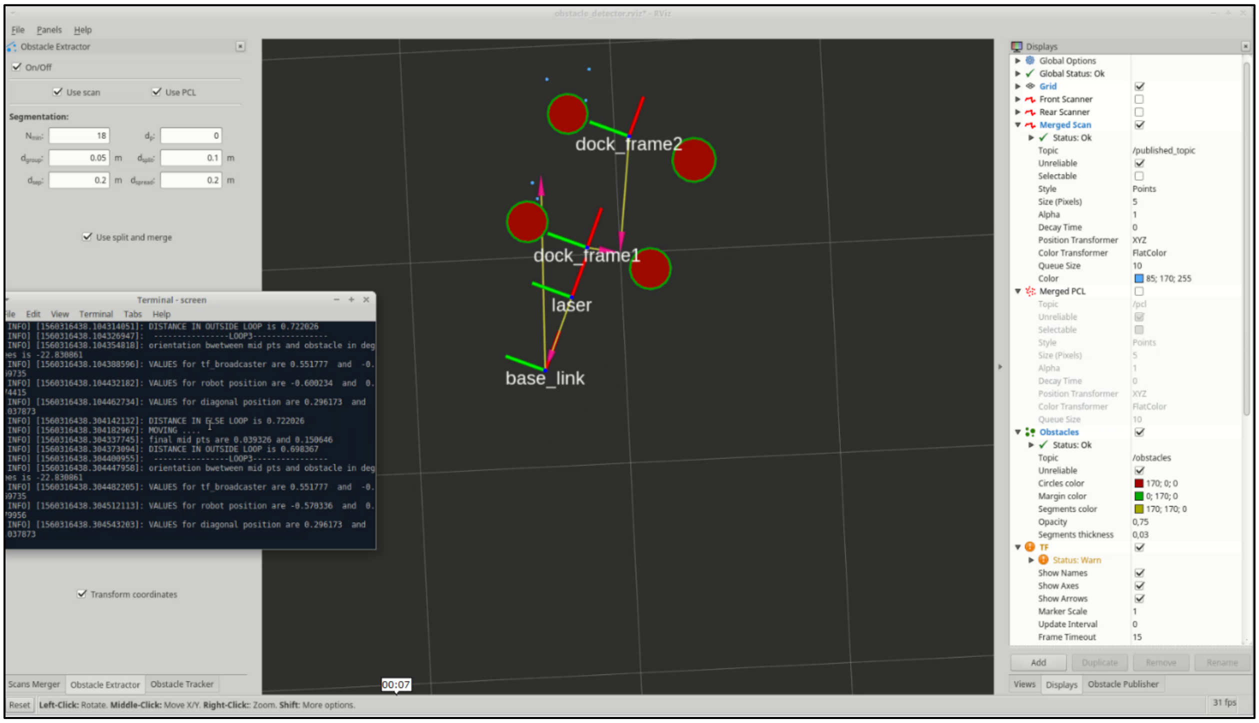
```
 INFO] [1560316428.911418226]: ------obstacle position is postion --0  --  0.16
917 and 0.435055 -------
 INFO] [1560316428.911458103]: ------obstacle position is postion --1  --  -0.1
3413 and -0.161812 --------
 INFO] [1560316428.911482892]: R VALUE IS 2
 INFO] [1560316428.911500755]: -------------LOOP7----------------------------
 INFO] [1560316428.911524596]: -------------ONE PLATFORM DETECTED-------------
------------
 INFO] [1560316428.911548485]: -------mid points are 0.015752 and 0.136621-----

 INFO] [1560316428.911572887]: -------diagonal points are 0.277543 and 0.016128
-----
 INFO] [1560316428.911592603]: general distance between obstacle 0.576379
 INFO] [1560316428.911616595]: radius are 0.084017  and  0.083350
 INFO] [1560316428.911647303]: final mid pts are 0.015752 and 0.136621
 INFO] [1560316428.911671189]: DISTANCE IN OUTSIDE LOOP is 1.827292
 INFO] [1560316428.911694244]:  ---------------LOOP1---------------
 INFO] [1560316428.911718267]: orientation bwetween mid pts and obstacle in deg
ees is -24.714891
 INFO] [1560316428.911745474]: VALUES for tf_broadcaster are 0.015752  and  0.1
6621
 INFO] [1560316428.911766770]: VALUES for robot position are -1.790236  and  -0
262461
```

Global Options
Global Status: Ok
Grid ☑
Front Scanner ☐
Rear Scanner ☐
Merged Scan ☑
  Status: Ok
  Topic: /published_topic
  Unreliable ☑
  Selectable ☐
  Style: Points
  Size (Pixels): 5
  Alpha: 1
  Decay Time: 0
  Position Transformer: XYZ
  Color Transformer: FlatColor
  Queue Size: 10
  Color: 85; 170; 255
Merged PCL ☐
  Topic: /pcl
  Unreliable ☑
  Selectable ☐
  Style: Points
  Size (Pixels): 5
  Alpha: 1
  Decay Time: 0
  Position Transformer: XYZ
  Color Transformer: FlatColor
  Queue Size: 10
Obstacles ☑
  Status: Ok
  Topic: /obstacles
  Unreliable ☑
  Circles color: 170; 0; 0
  Margin color: 0; 170; 0
  Segments color: 170; 170; 0
  Opacity: 0,75
  Segments thickness: 0,03
TF ☑
  Status: Warn
  Show Names ☑
  Show Axes ☑
  Show Arrows ☑
  Marker Scale: 1
  Update Interval: 0
  Frame Timeout: 15

**Figure 15.3: Docking with respect to dock_frame1**

Now, to move the robot in this new TF, the calculated information is published on the "/move_base_simple/goal" topic and "dock_frame1" is used as the reference frame. A value of pose position as 0.9m is assigned in the negative X-axis of "dock_frame1" so that the robot can move in front of the trolley base. The quaternion as 0,0,0,1 is fed as an input, so that robot aligns itself to the X-axis of the "dock_frame1". The position of the robot in the "map" frame is constantly monitored. Then, the robot moves with respect to "dock_frame1" until the distance between the midpoint and the robot is greater than 1.1m. When the robot reaches closer than this distance, then the robot moves linearly along the negative X-axis of "dock_frame1" for 0.3m. At this point, the robot is close enough to the trolley base and this small linear motion allows all caster wheels to align straight along the X-axis of the "dock_frame1" as shown in Figure 15.3.

Now the laser mounted on the robotic platform can detect all four cylindrical objects. Then, a new ROS TF is created at the midpoint of the farther docking edge as depicted in Figure 15.4. The TF is aligned along the vector from the midpoint of the farther docking edge towards the midpoint of the detected diagonal. This TF is denoted as "dock_frame2". The calculated data is published on the "/move_base_simple/goal" topic and it uses "dock_frame2" as a reference frame. Now, the robot is moved at the at x=-0.2m with respect to "dock_frame2". At the end of the motion, the omnidirectional platform docks itself under the trolley base. The coordinate frames are depicted below in Figure 15.4.



Obstacle Extractor

☑ On/Off

☑ Use scan ☑ Use PCL

**Segmentation:**

$N_{min}$: 18 $d_p$: 0
$d_{group}$: 0.05 m $d_{split}$: 0.1 m
$d_{sep}$: 0.2 m $d_{spread}$: 0.2 m

☑ Use split and merge

☑ Transform coordinates

Terminal - screen

```
 INFO] [1560316438.104314051]: DISTANCE IN OUTSIDE LOOP is 0.722026
 INFO] [1560316438.104326947]:  ----------------LOOP3----------------
 INFO] [1560316438.104354818]: orientation bwetween mid pts and obstacle in deg
ees is -22.830861
 INFO] [1560316438.104388596]: VALUES for tf_broadcaster are 0.551777  and  -0.
9735
 INFO] [1560316438.104432182]: VALUES for robot position are -0.600234  and  0.
4415
 INFO] [1560316438.104462734]: VALUES for diagonal position are 0.296173  and
037873
 INFO] [1560316438.304142132]: DISTANCE IN ELSE LOOP is 0.722026
 INFO] [1560316438.304182967]: MOVING ....
 INFO] [1560316438.304337745]: final mid pts are 0.039326 and 0.150646
 INFO] [1560316438.304373094]: DISTANCE IN OUTSIDE LOOP is 0.698367
 INFO] [1560316438.304400955]:  ----------------LOOP3----------------
 INFO] [1560316438.304447958]: orientation bwetween mid pts and obstacle in deg
ees is -22.830861
 INFO] [1560316438.304482205]: VALUES for tf_broadcaster are 0.551777  and  -0.
9735
 INFO] [1560316438.304512113]: VALUES for robot position are -0.570336  and  0.
9956
 INFO] [1560316438.304543203]: VALUES for diagonal position are 0.296173  and
037873
```

Displays

| | |
|---|---|
| Global Options | |
| Global Status: Ok | |
| Grid | ☑ |
| Front Scanner | ☐ |
| Rear Scanner | ☐ |
| Merged Scan | ☑ |
| Status: Ok | |
| Topic | /published_topic |
| Unreliable | ☑ |
| Selectable | ☐ |
| Style | Points |
| Size (Pixels) | 5 |
| Alpha | 1 |
| Decay Time | 0 |
| Position Transformer | XYZ |
| Color Transformer | FlatColor |
| Queue Size | 10 |
| Color | 85; 170; 255 |
| Merged PCL | ☐ |
| Topic | /pcl |
| Unreliable | ☑ |
| Selectable | ☐ |
| Style | Points |
| Size (Pixels) | 5 |
| Alpha | 1 |
| Decay Time | 0 |
| Position Transformer | XYZ |
| Color Transformer | FlatColor |
| Queue Size | 10 |
| Obstacles | ☑ |
| Status: Ok | |
| Topic | /obstacles |
| Unreliable | ☑ |
| Circles color | 170; 0; 0 |
| Margin color | 0; 170; 0 |
| Segments color | 170; 170; 0 |
| Opacity | 0,75 |
| Segments thickness | 0,03 |
| TF | ☑ |
| Status: Warn | |
| Show Names | ☑ |
| Show Axes | ☑ |
| Show Arrows | ☑ |
| Marker Scale | 1 |
| Update Interval | 0 |
| Frame Timeout | 15 |

Figure 15.4: Docking with respect to dock_frame2

# 16. Experiment and Evaluation

To evaluate the implementations done in this master thesis, several experiments have been conducted and the statistical recorded data have been depicted in the tabular charts.

## 16.1. Test for Spline Motion Interpolation

The verification for the spline trajectory interpolation is performed by monitoring the deviation in the position and orientation of the base_link frame of the mobile robot. In this scenario, the mobile platform is commanded to move along the X-axis on the map coordinate frame. Spline motion is performed when the robot has reached the extreme endpoint. The following graphs are obtained using the rqt_plot in ROS.

The plot in Figure 16.1 depicts the orientation and the position of the base_link frame of the mobile robot, while it is moving in the spline trajectory. The red line shows the X-axis of the map coordinate frame.

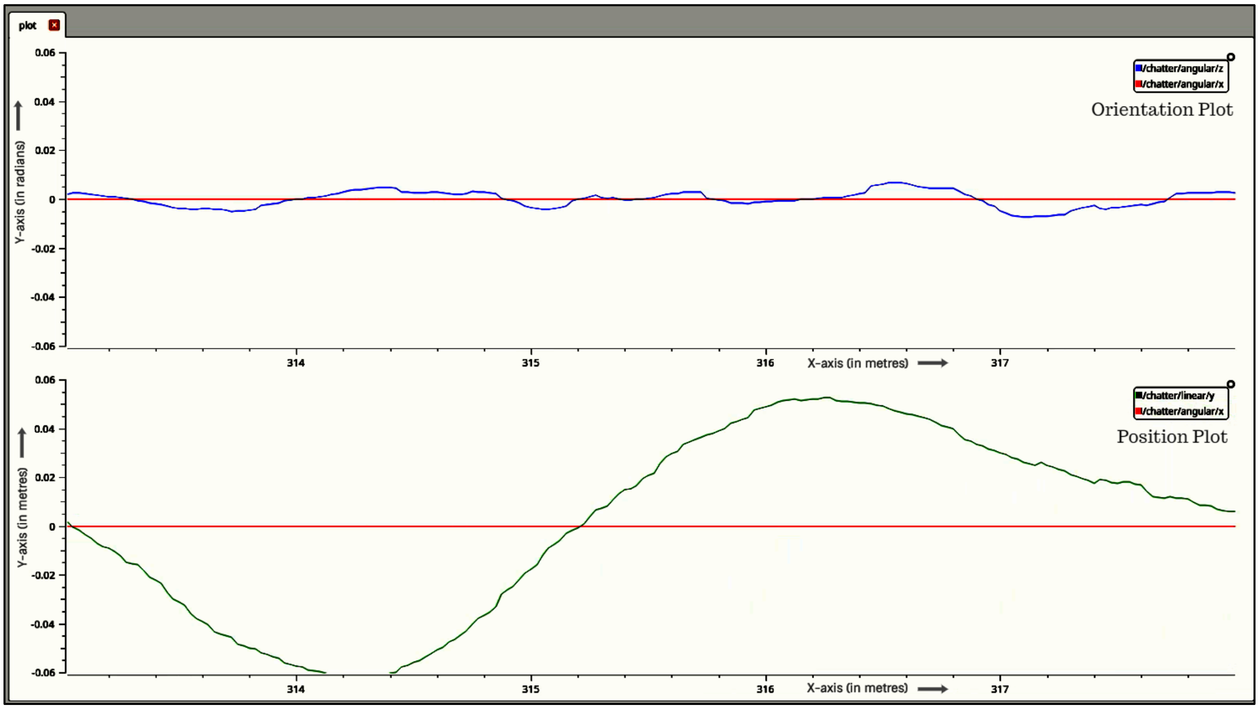


**Figure 16.1: Monitoring the orientation and the position of the Mobile Robot base_link in the Map frame during the spline trajectory**

The graph in Figure 16.2 depicts the deviation in the orientation and the deviation in the position of the base_link of the mobile robot while it is moving in the linearly. the red line shows the X-axis of the map coordinate frame. The maximum deviation in the orientation observed was 0.01 radians (0.57 degrees). The maximum deviation in the position observed was 0.01 meters.

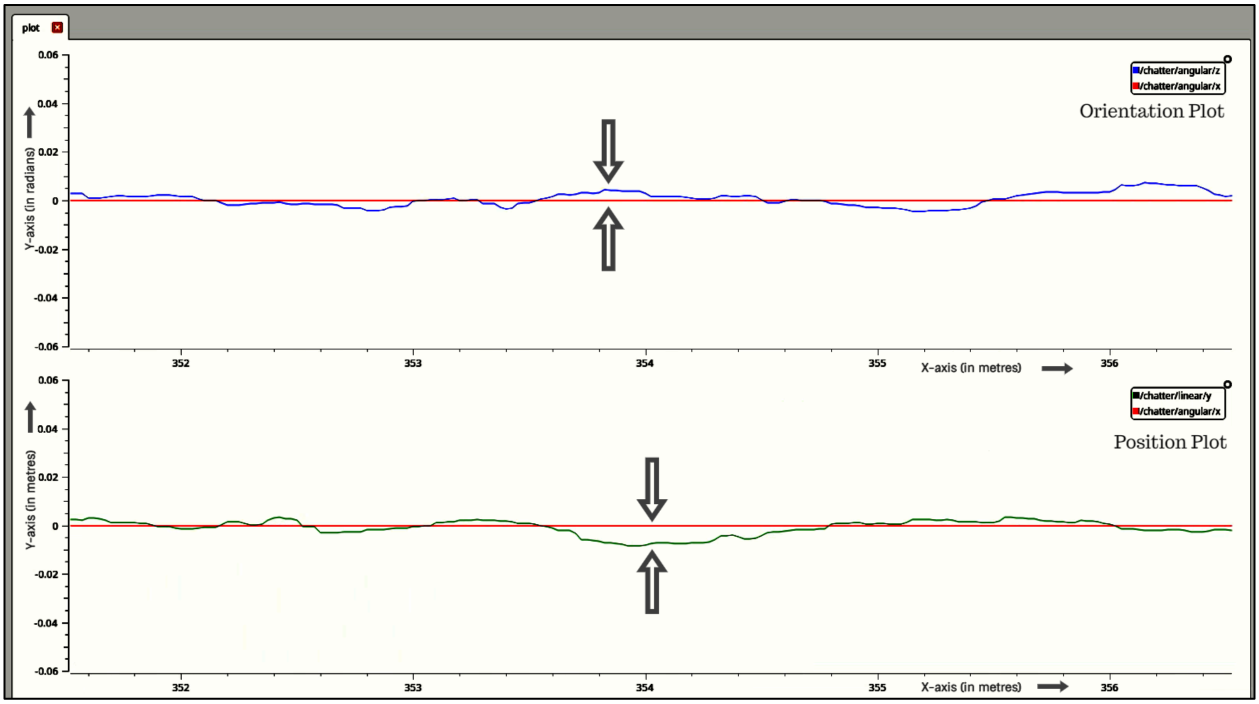


**Figure 16.2: Deviation in the orientation and the position of the Mobile Robot base_link in the Map frame during the linear trajectory**

In the above Figure 16.2, the red line shows the X-axis of the map coordinate frame. The green line shows the trajectory followed by base_link of the mobile platform. At the beginning of the graph 1 and at the extreme end of the graph 2 shows the spline motion curve.

In this scenario, the spline trajectory is performed at the end of linear side to side motion. This experimental data is tabulated in Table 16.1.

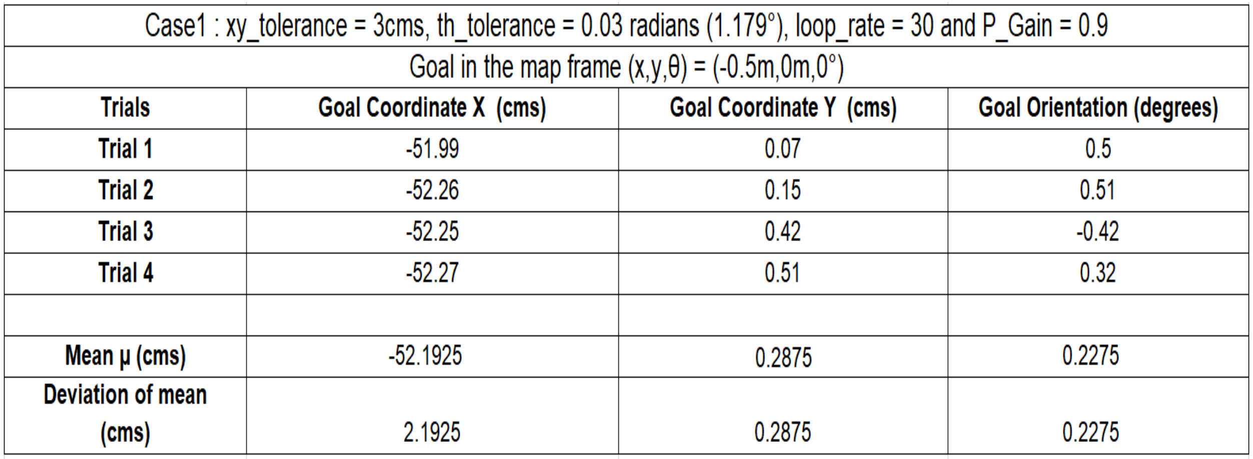
| Attempts | Position | Distance Covered in the real world (cms) | Mean µ (cms) | Standard Deviation σ (cms) | Variance $\sigma^2$ (in cms sq) |
|---|---|---|---|---|---|
| 1 | Position1 | 99 | 99.125 | 0.21650635 | 0.046875 |
|  | Position2 |  |  |  |  |
| 2 | Position2 | 99 |  |  |  |
|  | Position1 |  |  |  |  |
| 3 | Position1 | 99 |  |  |  |
|  | Position2 |  |  |  |  |
| 4 | Position2 | 99.5 |  |  |  |
|  | Position1 |  |  |  |  |

**Table 16.1: Statistical data obtained from experiments with spline trajectory**

As a result, it is observed that the standard deviation in the earlier scenario without spline trajectory (as explained in 9.1), is higher than this current scenario with the spline. This also concludes that the distance traveled by the mobile platform is more stable and accurate when a spline trajectory is performed at the end of linear side to side motion.

## 16.2. Test for Base Controller

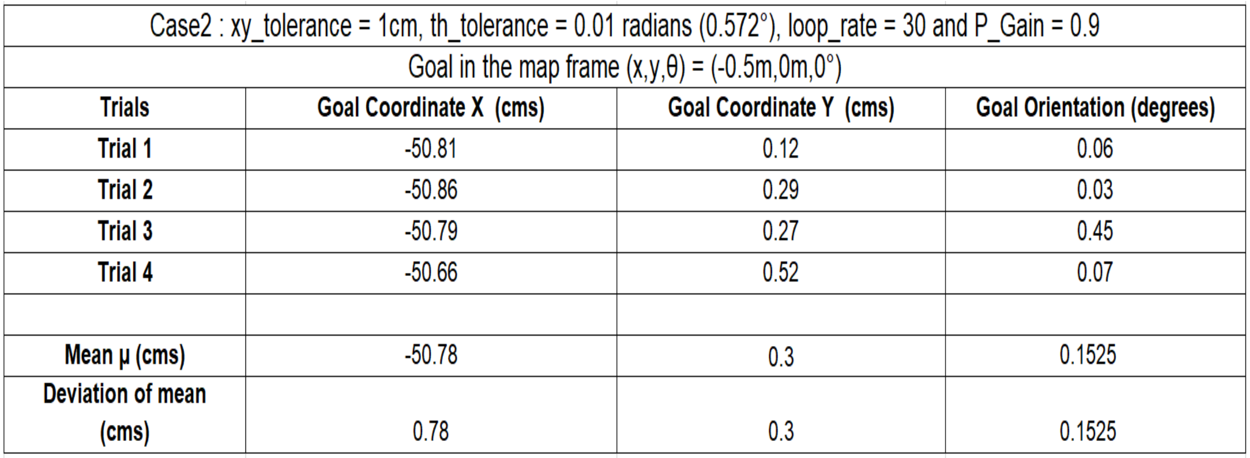
Various experiments are conducted in the real-world environment, to enhance the accuracy of the mobile platform, for reaching towards the goal point by varying the configuration parameters of the "nav_pcontroller".



| Case1 : xy_tolerance = 3cms, th_tolerance = 0.03 radians (1.179°), loop_rate = 30 and P_Gain = 0.9 | | | |
|---|---|---|---|
| Goal in the map frame (x,y,θ) = (-0.5m,0m,0°) | | | |
| **Trials** | **Goal Coordinate X (cms)** | **Goal Coordinate Y (cms)** | **Goal Orientation (degrees)** |
| **Trial 1** | -51.99 | 0.07 | 0.5 |
| **Trial 2** | -52.26 | 0.15 | 0.51 |
| **Trial 3** | -52.25 | 0.42 | -0.42 |
| **Trial 4** | -52.27 | 0.51 | 0.32 |
| | | | |
| **Mean μ (cms)** | -52.1925 | 0.2875 | 0.2275 |
| **Deviation of mean (cms)** | 2.1925 | 0.2875 | 0.2275 |

**Table 16.2: Statistical data obtained from experiments of the position controller with large value of tolerance**

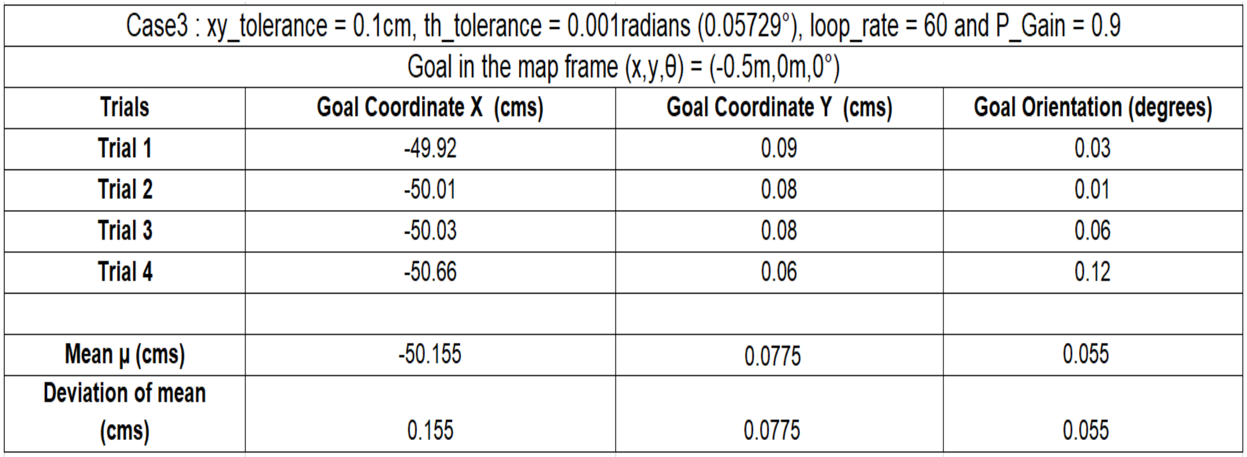
With the reference to Table 16.2, the position tolerance was set to 3cm, orientation tolerance was set to 0.03 radians, loop_rate was set to 30 and the controller gain was set to 0.9. It was found that the deviation from the goal point along X-axis was found to be 2.1925cms, along Y-axis was found to be 0.2875cms and orientation was 0.2275°.



| Case2 : xy_tolerance = 1cm, th_tolerance = 0.01 radians (0.572°), loop_rate = 30 and P_Gain = 0.9 | | | |
|---|---|---|---|
| Goal in the map frame (x,y,θ) = (-0.5m,0m,0°) | | | |
| **Trials** | **Goal Coordinate X (cms)** | **Goal Coordinate Y (cms)** | **Goal Orientation (degrees)** |
| **Trial 1** | -50.81 | 0.12 | 0.06 |
| **Trial 2** | -50.86 | 0.29 | 0.03 |
| **Trial 3** | -50.79 | 0.27 | 0.45 |
| **Trial 4** | -50.66 | 0.52 | 0.07 |
| | | | |
| **Mean μ (cms)** | -50.78 | 0.3 | 0.1525 |
| **Deviation of mean (cms)** | 0.78 | 0.3 | 0.1525 |

**Table 16.3: Statistical data obtained from experiments of the position controller with the medium tolerance**

With the reference to Table 16.3, the position tolerance was set to 1cm, orientation tolerance was set to 0.01 radians, loop_rate was set to 30 and the controller gain was set to 0.9. It was found that the deviation from the goal point along X-axis was found to be 0.78cms, along Y-axis was found to be 0.3cms and orientation was 0.1525°.



| Case3 : xy_tolerance = 0.1cm, th_tolerance = 0.001radians (0.05729°), loop_rate = 60 and P_Gain = 0.9 | | | |
|---|---|---|---|
| Goal in the map frame (x,y,θ) = (-0.5m,0m,0°) | | | |
| **Trials** | **Goal Coordinate X (cms)** | **Goal Coordinate Y (cms)** | **Goal Orientation (degrees)** |
| **Trial 1** | -49.92 | 0.09 | 0.03 |
| **Trial 2** | -50.01 | 0.08 | 0.01 |
| **Trial 3** | -50.03 | 0.08 | 0.06 |
| **Trial 4** | -50.66 | 0.06 | 0.12 |
| | | | |
| **Mean µ (cms)** | -50.155 | 0.0775 | 0.055 |
| **Deviation of mean (cms)** | 0.155 | 0.0775 | 0.055 |

**Table 16.4: Statistical data obtained from experiments of the position controller with low tolerance**

With the reference to Table 16.4, the position tolerance was set to 0.1cm, orientation tolerance was set to 0.001 radians, loop_rate was set to 60 and the controller gain was set to 0.9. It was found that the deviation from the goal point along X-axis was found to be 0.155cms, along Y-axis was found to be 0.0775cms and orientation was 0.055°.

Hence, it can be seen that by decreasing the position and orientation tolerance and increasing the loop_rate, for a constant P_Gain(0.9), the precision and accuracy of the mobile robot motion to reach a given goal point can be improved.

## 16.3. Test for Object Shape Recognition

The experiments have been conducted considering one of the corners of the trolley cart as an object. Table 16.5 represents the experimental data obtained from the implementation of the shape recognition of an object. During the experiments, the platform was kept at different positions and the object shape recognition for the same object has been performed.

| Trials | Obstacle Coordinate X (cms) | Obstacle Coordinate Y (cms) | Obstacle Radius (cms) |
|---|---|---|---|
| **Trial 1** | -44.42 | -32.83 | 9.23 |
| **Trial 2** | -44.35 | -32.52 | 9.28 |
| **Trial 3** | -44.86 | -33.03 | 9.17 |
| **Trial 4** | -44.03 | -33.17 | 9.78 |
| **Trial 5** | -44.16 | -33.23 | 8.97 |
| | | | |
| **Mean μ (cms)** | -44.364 | -32.956 | 9.286 |
| **Standard Deviation σ (cms)** | 0.3172223 | 0.288236 | 0.300216 |
| **Variance $\sigma^2$ (in cms sq)** | 0.10063 | 0.083079 | 0.09013 |

**Table 16.5: Statistical data obtained from experiments for object shape recognition**

The X coordinate, Y coordinate and radius of the detected object are tabulated below. It is observed that the standard deviation for center coordinates and the radius calculation is less than 0.5cms. This shows that the object detection algorithm is quite accurate.

## 16.4. Test for Autonomous Docking Application

In the real environment, several trials were conducted to test the above-implemented docking application. The trolley cart was fixed at a particular point in the map coordinate frame. The docking application was performed by considering the different initial position of the platform. Table 16.6 represents the data collected while performing the trials. Four different trails for the docking application were conducted. Finally, the standard deviation was calculated. It was found to be less than 0.5 cms. This proves that the docking application is quite precise in a real-world application.

| Trials | Docking Coordinate X (cms) | Docking Coordinate Y (cms) |
|---|---|---|
| **Trial 1** | -16.21 | -59.27 |
| **Trial 2** | -17.51 | -59.04 |
| **Trial 3** | -17.25 | -58.74 |
| **Trial 4** | -16.89 | -59.81 |
| | | |
| **Mean μ (cms)** | -16.965 | -59.215 |
| **Standard Deviation σ (cms)** | 0.48833902 | 0.39156736 |
| **Variance $\sigma^2$ (in cms sq)** | 0.238475 | 0.153325 |

**Table 16.6: Statistical data obtained from experiments for autonomous docking**

# 17. Summary and Conclusion

In this master thesis, an emphasis was laid on the research to optimize the localization and control of the omnidirectional platform. Adaptive Monte Carlo Localization was used for localizing the mobile platform in the given map coordinate frame. This localization scheme uses laser sensor data and odometry data to perform localization.

The spline motion interpolation trajectory has been implemented to overcome the non-linear flipping effect of the caster wheels. This spline trajectory made sure the flipping of the caster wheels is controlled smoothly and precisely. The maximum deviation in the orientation was observed to be within $[-0.57,0.57]$ degrees for the straight-line motion of the mobile platform. The maximum linear deviation observed was within $[-1,1]$ cm for the straight-line motion of the mobile platform.

An algorithm for object shape recognition using the laser sensor data was also developed. Various tests were performed to improve its precision. Final achieved precision was with a standard deviation less than 0.5cms.

Object Shape recognition algorithm was further used to perform the docking application. This application involves moving the omnidirectional platform under the trolley cart. In this application, the performance was estimated to have a standard deviation of 0.5cms.

As a future scope, the localization algorithm can be enhanced by the implementation of the sensor fusion, for example, Inertial Measurement Unit (IMU) sensor data can be fused with current AMCL algorithms. This will improve the precision of the mobile platform localization. Visual SLAM can be implemented to perform simultaneous localization and mapping of an unknown environment using computer vision.

The passive caster wheels can be replaced with active powered caster wheels to improve its position, orientation and velocity control. Further, computer vision libraries (along with the ArUco marker) can be implemented to perform the docking application more precisely.

# 18. References

[1] HENDZEL, Z., & RYKAŁA, Ł. (2017). Modelling of dynamics of a wheeled mobile robot with mecanum wheels with the use of langrange equations of the second kind. *Int. J. of Applied Mechanics and Engineering, 22*(10.1515/ijame-2017-0005), 82-84.

[2] A. Azim and O. Aycard. (2012). Detection, classification and tracking of moving objects in a 3D environment. *Proceedings of IEEE Intelligent*(pp. 802–807).

[3] Abel, M., Luis, C. B., & Urbano, N. (2004). Multi-target detection and tracking with a laserscanner. *Proceedings of IEEE Intelligent Vehicles Symposium*(pp. 796–801).

[4] Bresenham, J. E. (1962). *revolvy.* Retrieved May 24, 2019, from https://www.revolvy.com/page/Bresenham%27s-line-algorithm

[5] Farin, G. (1996). Curves and Surfaces for Computer Aided Geometric Design: A Practical Guide. (London: Academic Press).

[6] H2020 REFILLS Research and Innovation Action. (2017, January). *REFILLS Project*. Retrieved April 2019, from REFILLS Project: http://www.refills-project.eu/

[7] Hoschek, J., Lasser, D., & Schumaker, L. L. (1993). Fundamentals of Computer Aided Geometric Design. *Wellesley, Massachusetts*.

[8] Jens, H., Karina, H., Torsten, B., & Manfred, H. (2003). Emergency Path Planning for Autonomous Vehicles Using Elastic Band Theory. *IEEE/ASME*(0.7803-7759-1/03), 1392-1393.

[9] Konolige, K., Grisetti, G., Kmmerle, R., Burgard, W., Limketkai, B., & Vincent, R. (Oct 2010). Efficient sparse pose adjustment for 2d mapping. *In: 2010 IEEE/RSJ International Conference on Intelligent Robots and Systems.pp. 22-29.*

[10] Leal, L. A. (2018, Dec 2). *Towards Data Science*. Retrieved June 5 May 2019, 2019, from https://towardsdatascience.com/numerical-interpolation-natural-cubic-spline-52c1157b98ac

[11] Paleja, R. (2017). Adaptive Monte Carlo Localization for Mobile Robots. 1-2.

[12] Przybyła, M. (2017). Online Detection and Tracking of 2D Geometric Obstacles from LRF Data. (Faculty of Computing, Chair of Control and Systems).

[13] ROS ORG. (2007). *ROS nav_core*. Retrieved June 2019, from https://wiki.ros.org/nav_core

[14] ROS ORG. (2007). *ROS Navigation.* Retrieved May 2019, from http://wiki.ros.org/navigation/Tutorials

[15] ROS WIKI. (2007, May). *costmap_2d*. Retrieved July 2019, from http://wiki.ros.org/costmap_2d

[16] ROS.org. (2007). *ROS WIKI*. Retrieved May 2019, from http://wiki.ros.org/amcl

[17] ROS.org. (2007, May). *ROS WIKI*. Retrieved May 2019, from http://wiki.ros.org/actionlib

[18] SIEGWART, R., & NOURBAKHSH, I. R. (2004). *Introduction to Autonomous Mobile Robots.* Massachusetts Institute of Technology.

[19] Siegwart, R., Nguyen, V., Martinelli, A., & Tomatis, N. (2005). A comparison of line extraction algorithms using 2D laser rangefinder for indoor mobile robotics. *Proceedings of IEEE/RSJ International Conference*(pp. 1929–1934.).

[20] Thrun, S., Burgard, W., & Fox, D. (2005). *Probabilistic Robotics.* MIT Press.

[21] Tyroremotes. (2000). *Tyroremotes*. Retrieved May 2019, from https://www.tyroremotes.co.uk/o/radio-remote-controls/indus-gemini

[22] Vincent, R., Limketkai, B., & Eriksen, M. (April 2010.). Comparison of indoor robot localization techniques in the absence of GPS. *In Proc. of SPIE: Detection and Sensing of Mines, Explosive Objects, and Obscured Targets XV of Defense, Security, and Sensing Symposium*.

[23] Wei, H., Yu, Q., Huang, Z., & Guan, Y. (2014). RGMP-ROS: a Real-time ROS Architecture of Hybrid RTOS and GPOS on Multi-core Processor. *IEEE International Conference on Robotics & Automation*, 2483.

# 19. Appendix

## 19.1. Nomenclature

| | |
|---|---|
| $r_1, r_2$ | –radius of Mecanum wheel and roller |
| $T_X, T_Y$ | –length and width of the omnidirectional platform |
| $\delta$ | –the angle of rotation of the platform |
| $\alpha$ | –construction angle of Mecanum wheel |
| $v$ | –linear velocity |
| $\varphi$ | –the angle of rotation of Mecanum wheel |