MS Project Report

Indoor Navigation Robot

Submitted in partial fulfilment for the degree of
Master of Science (MS) in Mechatronics and Robotics. *by*
Abhimanyu Dhawan

To the department of
Mechanical and Aerospace Engineering
(Fall 2018)

Abstract:

The aim of this project was to develop a prototype solution for indoor navigation using robots which can help people in guiding humans in complex/large indoor environments. The need of indoor navigation is very well justified as Global Positioning System's (GPS) accuracy is very less inside closed walls. In areas like shopping malls, supermarkets, airports, museums etc. people have difficulty in deciding on correct path to their destination and needs assistance. For this purpose, robots can plan the path based on stored map of environment and goal location specified by the user. This simplifies the problem as now the human could just follow the robot to reach his destination. He can concentrate better on the task he has and not worry about an optimal path. This system is based on highly sophisticated algorithms of path planning and obstacle avoidance along with a state of the art depth sensing to give feedback to the robot. In addition, the mobile base is also very rugged and precise in control to ensure maximum reliability of the robot.

The other aim which was achieved through this robot was making the system most accessible and viable for the market by making the product modular and based on commercial off-the shelf products. Several algorithms were tested in both simulation and real-world scenarios and the results were highly encouraging. Robot was precisely able to map the surrounding with minor latency and localize itself to correct position. Although, there was an issue with some drift in odometry readings but that can be compensated as work on this system matures.

INDEX

Sr. Number	Title	Page Number
I.	Introduction	4
II.	Hardware	6
III.	Hardware Setup	12
IV.	Software Setup	14
V.	Overall Process	16
VI.	Software and Algorithms	17
VII.	Conclusion	21
VIII.	References	22

Introduction:

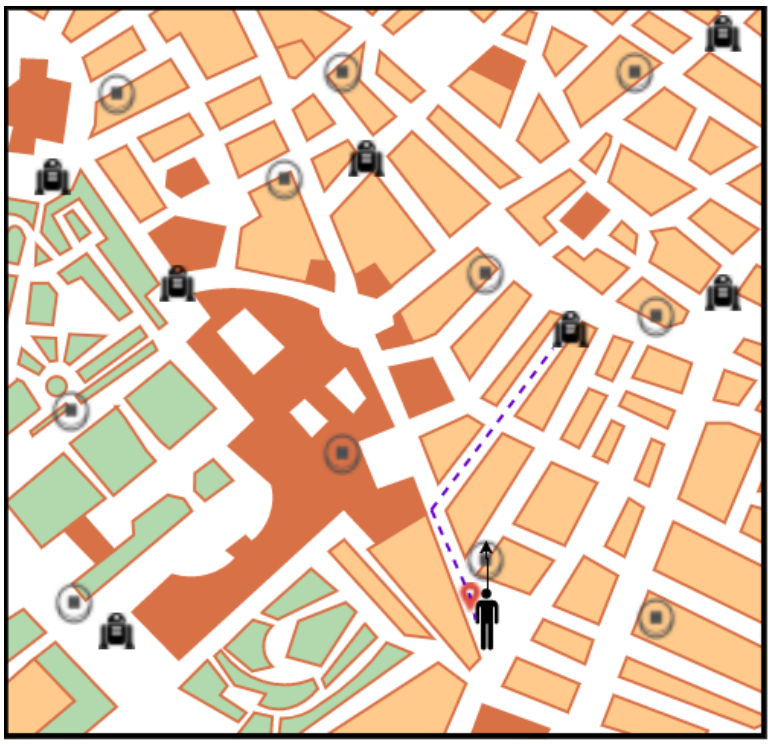
Big venues like airports, museums, parking lots, industries, shopping malls and hypermarkets want to improve their ease of access and navigation for humans indoors. People are interested in most convenient and user-friendly experience inside these venues. Many solutions are proposed in this area using technologies like Wifi triangulation, Bluetooth Beacons, Vision, RFID position and magnetic field positioning solutions by big companies like apple, google, Philips etc. But the major drawback of these was that they all are based on an app running on users mobile. These apps are susceptible to many failures like bad signal reception, limited hardware and software capability of media device they are running on or simply lack of enough battery to run the application for accurate guidance. And in many cases the users are not willing to or are able to install the required software being concerned about their technical abilities to use them. Hence this project was about developing Indoor navigation solution using an external robot agent which a human can follow to get guidance.

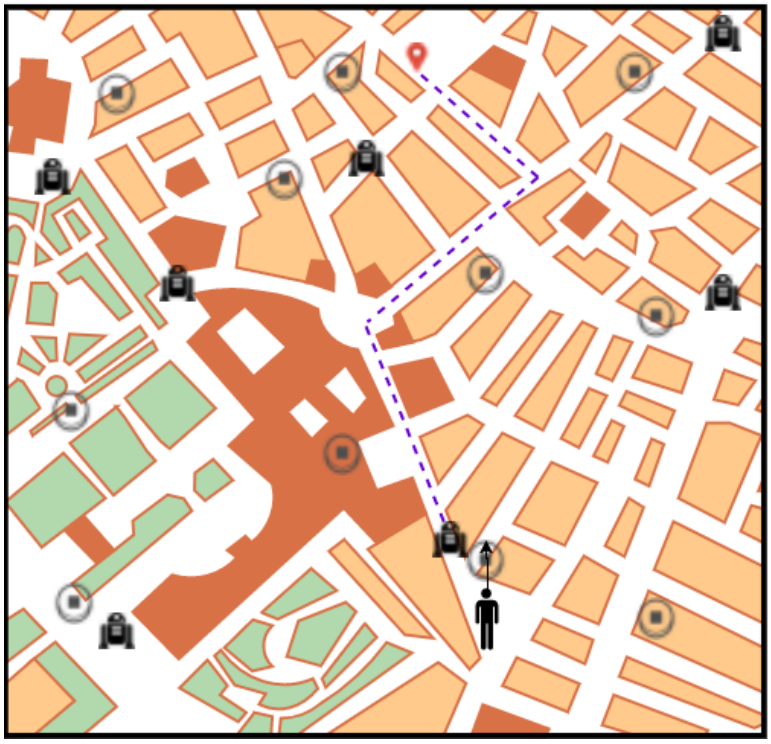
The basic idea was to implement path planning and obstacle avoidance outside the interface that user needs to carry along with him to make it more convenient for the task of navigation to be done easily and precisely.

Here a mobile base of a vacuum cleaning robot called Neato was taken and was interfaced with the mini-computer: Raspberry pi 3. Using depth sensing and point cloud generation capability of google tango device, a 3d map of surroundings was created and the whole indoor area was mapped in 2d based on obstacles around the Robot. This device was mounted on top of the robot using 3d printed stands and a holder with firm grip. The open source platform- Robot Operating System (ROS) was used for understanding and implementing advanced algorithm to localize the Robot based on the map and get it to a goal position avoiding the obstacle and taking the most optimal path.

Another focus was on system architecture and usability. Using Arcade buttons with raspberry pi zero placed at various key locations the user called the nearest bot to help him. He got feedback on robot's status based on blinking pattern of button's internal LED and once the robot reaches him it turns off to get ready for next call. The user interacts with a touch screen of Google tango device to feed the target location where he wants to reach in an easy to operate interface. Then he just follows this robot around to reach his destination in minimal time. Figure1. explains this architecture for an end-user.

In the end the robot is designed to be highly modular using custom 3d printed expandable stand for mobile and incorporating commercial products like a vacuum cleaner, mobile phone and power banks. They can be combined with affordable and readily available controller and open source software to generate superior solutions applicable on large scale indoor navigation problems. The whole system can work over the local network and does not require any external processor other than raspberry pi and tango device to do all complex computation and networking.





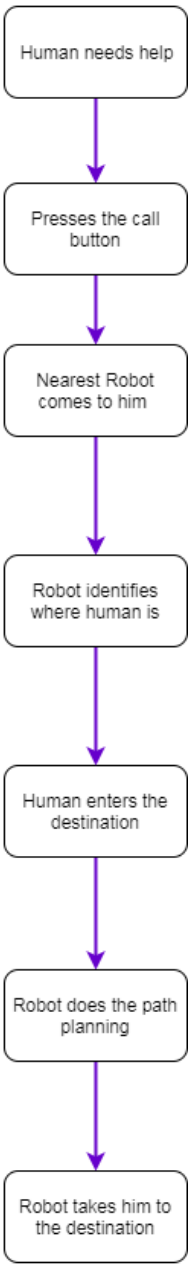


Figure 1: Flow diagram for end-user

Hardware:

Following is the list of electronic hardware components that made it possible for the system to deliver the navigation solution inside a closed space:

1. Mobile base: Neato Botvac D Series D3/D5 Robotic Vacuum
2. Server computer: Raspberry pi 2
3. Storage device: Memory Card
4. Client computer: Raspberry pi zero W
5. Power source: Power bank and AC source
6. Haptic sensors with feedback: Arcade buttons
7. External Laser Sensor, touch screen: Lenovo Phab 2 pro (google tango)

1. Neato Botvac D Series D3/D5 Robotic Vacuum

We required a rugged mobile base with differential drive system and a circular structure which can easily maneuverer in corners and small spaces. Additionally, this base was required to be an already commercial product which can be modified for our application. There were two main candidates which were being used in vacuum cleaning application and suited our requirement. They were Roomba and neato Botvac D3/D5. Figure2. is the comparison between two platforms.

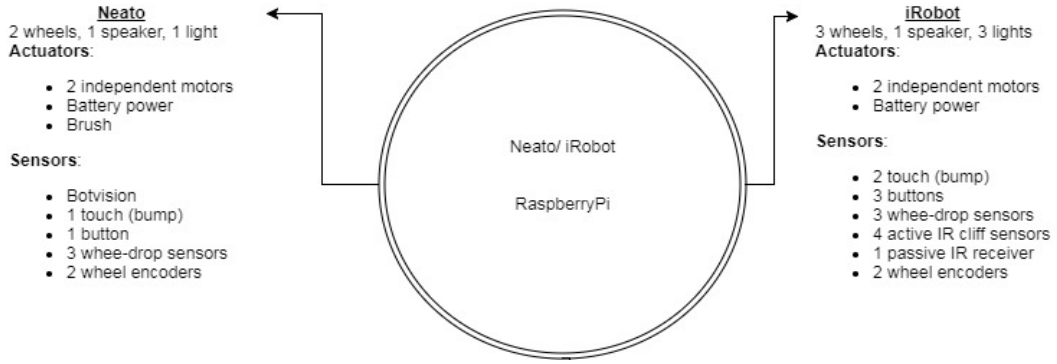


Figure 2

Although the programming of iRobot is easier and more elaborate in documentation and its

More widely used robot for vacuum cleaning but from technical side it has one major disadvantage. Neato robots have inbuilt Botvision (laser range finder) which is a cheaper version of advanced hokuyu laser range finders (Figure3) which iRobots does not have. Hence using this LIDAR a rough 360 laser scan of the environment has been done which compliments our actual more sophisticated scan from google tango device. This LIDAR inside is combination of a laser and a CMOS imager that the Lidar uses to detect distance to objects. It is a low-priced custom-made Lidar sensor which as of now can only be procured from inside of this vacuum cleaner (reference 1). It has 1 degree accuracy planar scanning with 10hz frequency. It has range of atleast 22 cm and operates at 115200 baud rate as found out experimentally.

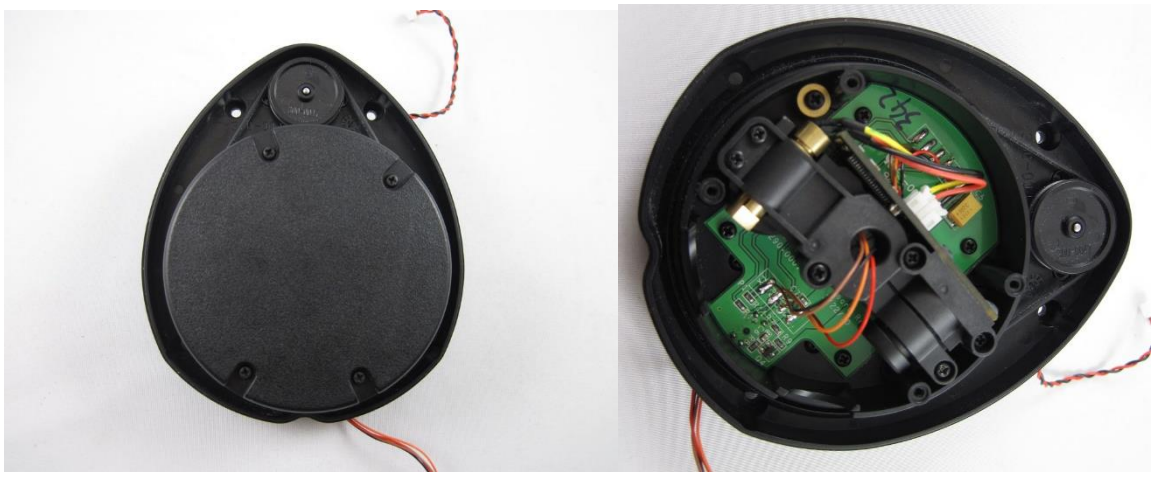


Figure 3

Other hardware specifications of Neato Botvac D3 that makes this base a very good choice for this application are as follows (reference 2):

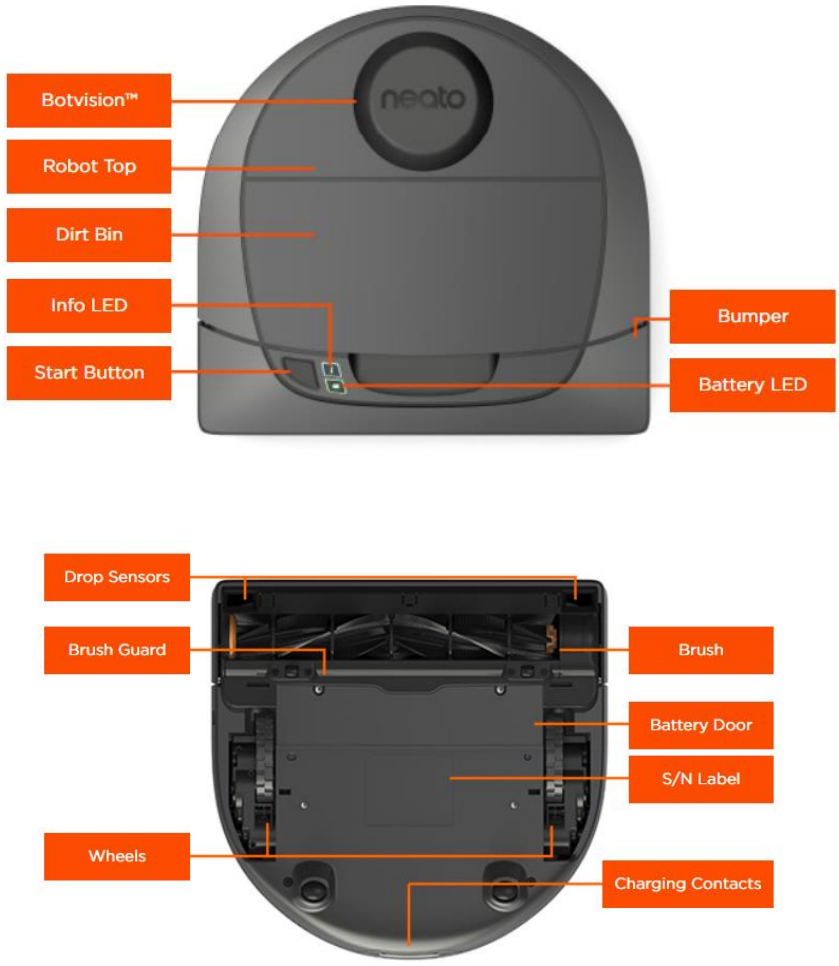


Figure 4: Hardware of Neato Botvac D3

Product Dimensions		
	Metric	*U.S. Standard*
Length	33.6 cm	13.2 in
Width	31.9 cm	12.6 in
Height	10 cm	3.9 in
Weight	3.4 kg	7.5 lbs

Technical	
Battery	Lithium-ion
Charger Voltage	110V, 220V

Figure 5: Technical specification of Neato Botvac D3

The software documentation of this Robot base is as follows (reference 4.):

Table of Robot Application Commands

Command	Description
Clean	Starts a cleaning by simulating press of start button.
DiagTest	Executes different test modes. Once set, press Start button to engage. (Test modes are mutually exclusive.)
GetAccel	Get the Accelerometer readings.
GetAnalogSensors	Get the A2D readings for the analog sensors.
GetButtons	Get the state of the UI Buttons.
GetCalInfo	Prints out the cal info from the System Control Block.
GetCharger	Get the diagnostic data for the charging system.
GetDigitalSensors	Get the state of the digital sensors.
GetErr	Get Error Message.
GetLDSScan	Get scan packet from LDS.
GetLifeStatLog	Get All Life Stat Logs.
GetMotors	Get the diagnostic data for the motors.
GetSchedule	Get the Cleaning Schedule. (24 hour clock format)
GetSysLog	Get System Log data.
GetTime	Get Current Scheduler Time.
GetVersion	Get the version information for the system software and hardware.
GetWarranty	Get the warranty validation codes.
Help	Without any argument, this prints a list of all possible cmds. With a command name, it prints the help for that particular command
PlaySound	Play the specified sound in the robot.
RestoreDefaults	Restore user settings to default.
SetDistanceCal	Set distance sensor calibration values for min and max distances.
SetFuelGauge	Set Fuel Gauge Level.
SetLCD	Sets the LCD to the specified display. (TestMode Only)
SetLDSRotation	Sets LDS rotation on or off. Can only be run in TestMode.
SetLED	Sets the specified LED to on,off,blink, or dim. (TestMode Only)
SetMotor	Sets the specified motor to run in a direction at a requested speed. (TestMode Only)
SetSchedule	Modify Cleaning Schedule.
SetSystemMode	Set the operation mode of the robot. (TestMode Only)
SetTime	Sets the current day, hour, and minute for the scheduler clock.
SetWallFollower	Enables/Disables wall follower
TestMode	Sets TestMode on or off. Some commands can only be run in TestMode.
Upload	Uploads new program to the robot.

Figure 6: Software Specification of neato

2. Raspberry pi 2

The Rapsberry pi 2 is a credit card sized mini computer designed and developed in United Kingdom by Raspberry Pi Foundation. It is a highly capable Linux operating system based single board computer being used with Ubuntu Mate platform on server side(robot) and Raspbian OS client side (call buttons).

Specifications: 1.2GHz 64-bit quad-core ARMv8 CPU, 802.11n Wireless LAN, 1GB RAM, 4 USB ports, 40 GPIO pins, Full HDMI port, Ethernet port, Combined 3.5mm audio jack and composite video, Camera interface (CSI), Display interface (DSI), Micro SD card slot (now push-pull rather than push-push), VideoCore IV 3D graphics core.

This was required to work as a portable Linux machine which can drive the Robot through serial communication over its USB ports and can easily be programmed with ROS environment and its libraries.

3. Memory Card

A high-speed class 10, 32 gb memory card was used to ensure good processing speed and space availability for Raspberry pi's operating system, ROS and scanned maps.

4. Raspberry pi Zero W

Raspberry pi Zero W (Wireless) is a very-small (as compared to Raspberry Pi 3), Ultra-low-cost single board computer which comes with built in wifi and Bluetooth. This was required to get signals from call button sensors scattered at key locations and pass it over to ROS network using native Raspbian OS that it runs on over the wifi. The super-small size of this device makes it perfect for internet of buttons applications just sending message to respective robots to call them to their locations.

Specifications: 1GHz, Single-core CPU, 512MB RAM, Mini HDMI and USB On-The-Go ports, Micro USB power, HAT-compatible 40-pin header, Composite video and reset headers, CSI camera connector, 802.11n wireless LAN 802.11n wireless LAN

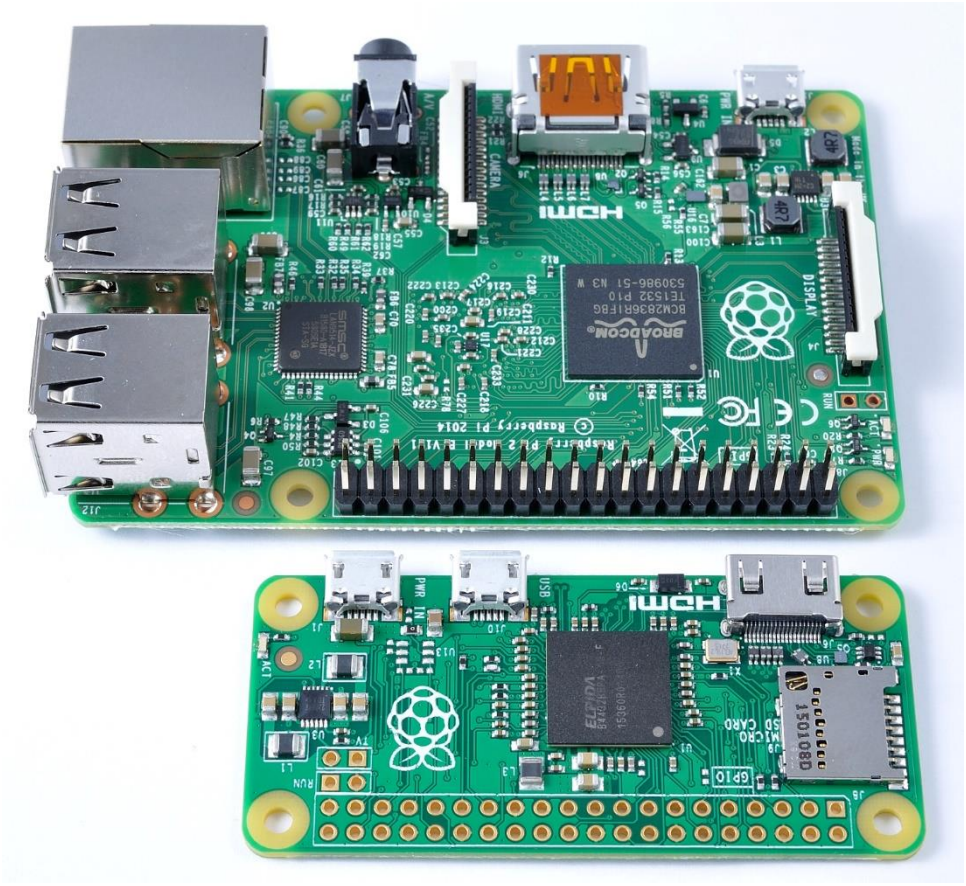


Figure 7: Raspberry pi 2 and Raspberry pi zero

5. Power bank and AC power

To make the system portable but long lasting powerful power sources were used. Power bank in use had a capacity of 12000 mAh with input of 5V/1A and output of 5v/3.4 A Max. This was able to support raspberry pi 2 for long durations approximately 8 hours of operation on continuous use after completely charged up. Raspberry pi zero W was sourced with AC power at 5V, 2A power supply USB cable.

6. Arcade buttons

For sensing and feedback at call button translucent arcade buttons with LEDs built inside it were used. They had diameter of 24 mm and thickness of 15 mm. As noted they had two surface mount LEDs with resistors built in, buried in the button body. Next to the switch contacts were two additional contacts for powering/controlling the LEDs. The two LEDs were connected in parallel with ~1K resistors each, so it could be powered by 5V (say USB) with 10mA draw. It could go down to 3.3V power, only 2mA per button, but they were dimmer.

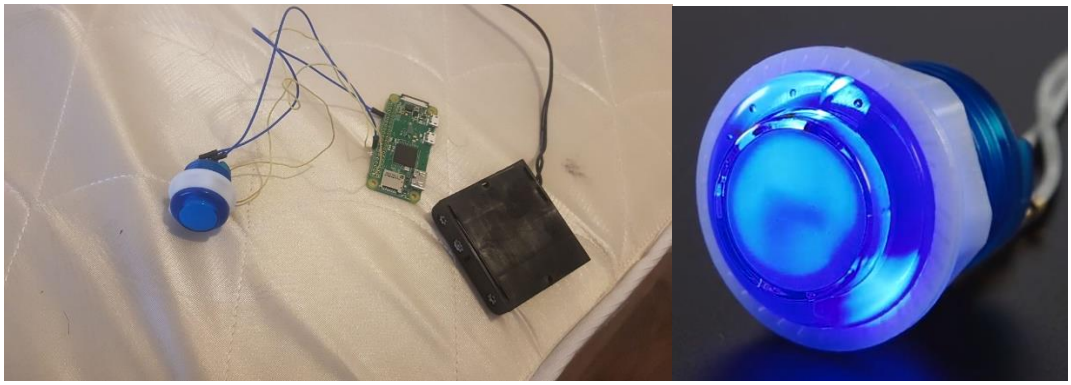


Figure 8: Arcade button

7. Lenovo Phab 2 pro (Google tango)

Tango is an emerging 3D-sensing computer vision product by google which run on standalone mobile phones and tablets. Using orientation and position information combined with vision sensors like RGB camera, fisheye camera and IR camera with IR emitters this device can generate six degree of freedom information about phone's motion and generate 3d scans of environment using it.

The software works by integrating three types of functionality:

- Motion-tracking: using visual features of the environment, in combination with accelerometer and gyroscope data, to closely track the device's movements in space
- Area learning: storing environment data in a map that can be re-used later, shared with other Tango devices, and enhanced with metadata such as notes, instructions, or points of interest
- Depth perception: detecting distances, sizes, and surfaces in the environment

Applications on mobile devices use Tango's C and Java APIs to access this data in real time. In addition, an API is also provided for integrating Tango with the Unity game engine; this enables the rapid conversion or creation of games that allow the user to interact and navigate in the game space by moving and rotating a Tango device in real space. These APIs are documented on the Google developer website.[8]

Along with Google tango's hardware and software this device offers a very big and fluid screen with size of 6.4 inches which was very suitable in our application for a custom designed interface for users to interact with. It has a powerful processor and large battery along with GPU and storage which aids in working of our project.

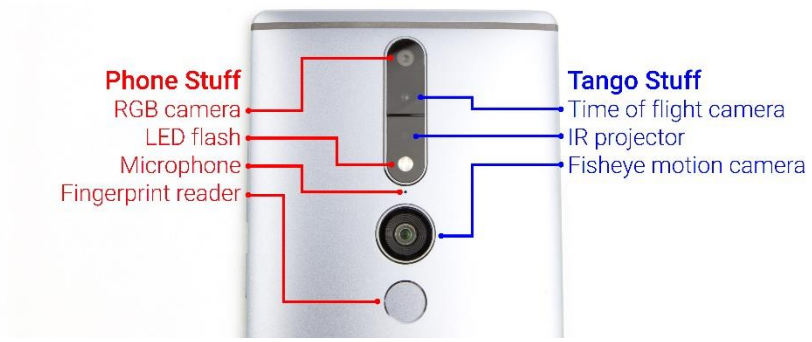


Figure 9: Lenovo Phab 2 Pro

SPECS AT A GLANCE: LENOVO PHAB 2 PRO	
SCREEN	2560×1440 6.4" (459ppi) IPS LCD
OS	Android 6.0
CPU	Octa-core Qualcomm Snapdragon 652 (four 1.8 GHz Cortex A72 cores and four 1.4 GHz Cortex A53 cores)
RAM	4GB
GPU	Adreno 510
STORAGE	64GB
NETWORKING	802.11b/g/n/ac, Bluetooth 4.0, GPS, NFC
PORTS	MicroUSB, 3.5mm headphone jack
CAMERA	16MP rear camera, 8MP front camera
SIZE	179.8 x 88.6 x 11 mm (7.08 x 3.49 x 0.43 in)
WEIGHT	259 g (9.12 oz.)
BATTERY	4050 mAh
STARTING PRICE	$499
OTHER PERKS	Fingerprint sensor, notification LED, Tango sensors

Figure 10: Phab 2 hardware specifications

2. Hardware Setup

Using CAD modelling and 3d printing components were printed to support mobile device on a stand above the base at the human height. There were 4 t-joints along with a holder to support the mobile and many aluminium rods with pvc joints were used to achieve the required height while maintaining centre of mass at the geometric centre of the robot.

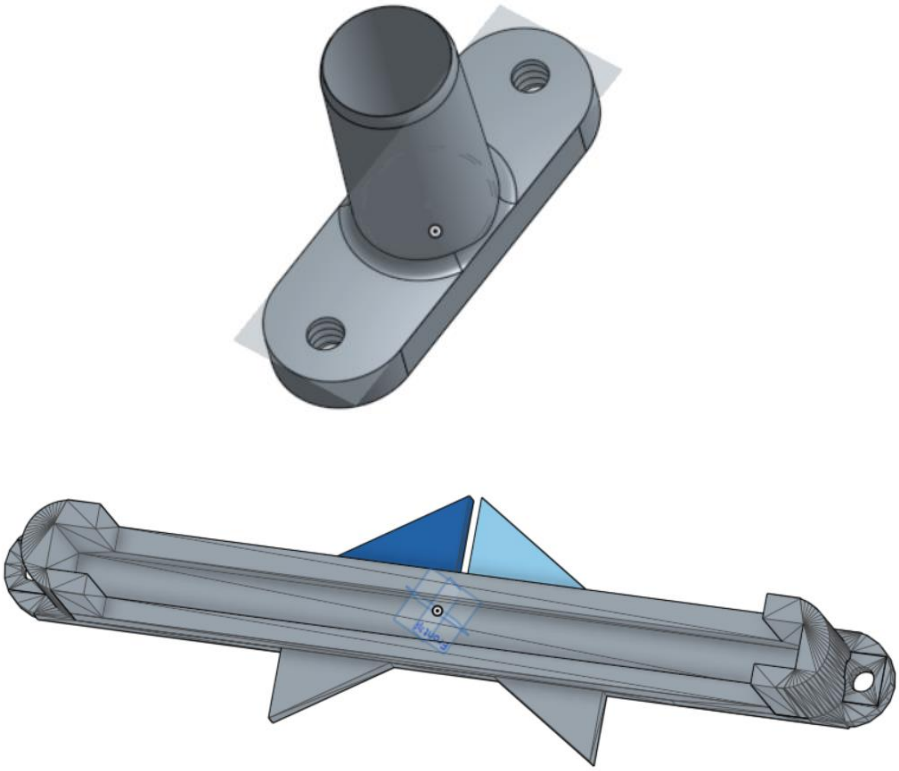


Figure 11: CAD of holder and T-joint

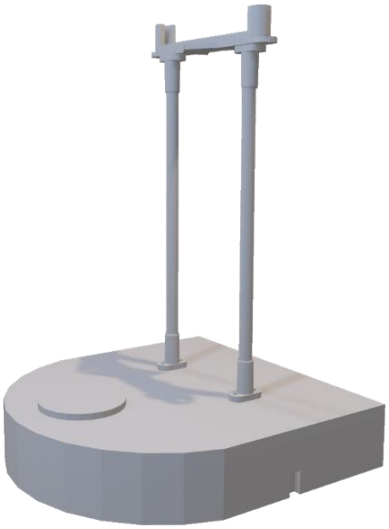


Figure 12: Final hardware structure

3. Software Setup

First step was to configure ubuntu and Raspbian OS in Raspberry pi 2 and Raspberry pi Zero W respectively. After this ROS kinetic was installed in both.

On server (robot) side following major libraries were installed (reference 3,4):

1. Move Base: The move_base package provides an implementation of an action (see the actionlib package) that, given a goal in the world, will attempt to reach it with a mobile base. The move_base node links together a global and local planner to accomplish its global navigation task.
2. AMCL: AMCL (adaptive monte-carlo) is a probabilistic localization system for a robot moving in 2D. It implements the adaptive (or KLD-sampling) Monte Carlo localization approach (as described by Dieter Fox), which uses a particle filter to track the pose of a robot against a known map.
3. Gmapping: This package contains a ROS wrapper for OpenSlam's Gmapping. The gmapping package provides laser-based SLAM (Simultaneous Localization and Mapping), as a ROS node called slam_gmapping. Using slam_gmapping, you can create a 2-D occupancy grid map (like a building floorplan) from laser and pose data collected by a mobile robot.
4. Lase_scan_matcher: An incremental laser scan matcher, using Andrea Censi's Canonical Scan Matcher (CSM) implementation.
5. Nav_core: This package provides common interfaces for navigation specific robot actions. Currently, this package provides the BaseGlobalPlanner, BaseLocalPlanner, and RecoveryBehavior interfaces, which can be used to build actions that can easily swap their planner, local controller, or recovery behavior for new versions adhering to the same interface.
6. Map_server: map_server provides the map_server ROS Node, which offers map data as a ROS Service. It also provides the map_saver command-line utility, which allows dynamically generated maps to be saved to file.
7. Global_planner: A path planner library and node.
8. Base_Local_planner: This package provides implementations of the Trajectory Rollout and Dynamic Window approaches to local robot navigation on a plane. Given a plan to follow and a costmap, the controller produces velocity commands to send to a mobile base
9. Navfn : navfn provides a fast interpolated navigation function that can be used to create plans for a mobile base. The planner assumes a circular robot and operates on a costmap to find a minimum cost plan from a start point to an end point in a grid.
10. Navigation: A 2D navigation stack that takes in information from odometry, sensor streams, and a goal pose and outputs safe velocity commands that are sent to a mobile base.
11. Neato_robot: Metapackage for drivers for the Neato XV-11 robot.
12. Tango_ros_streamer: This package wraps Tango Ros Streamer application

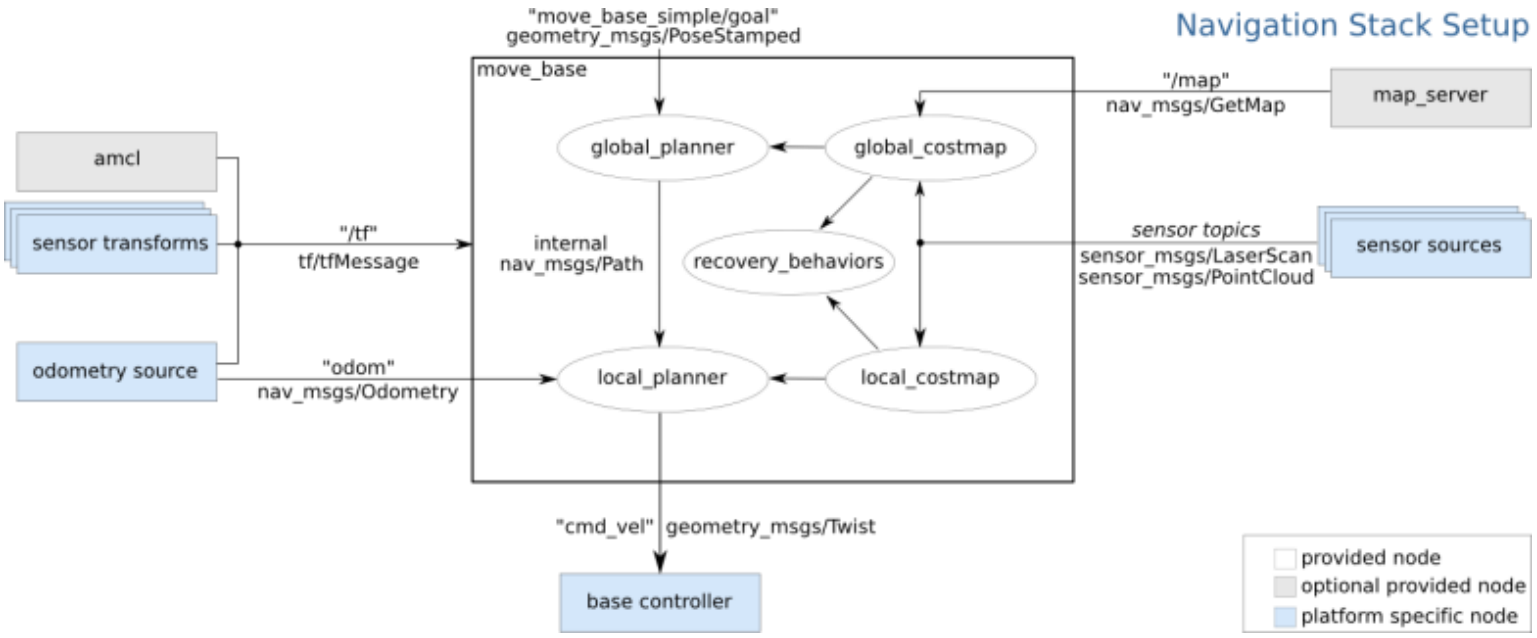


Figure 13: Navigation stack

It was required to make some changes in driver for Neato XV-11 in set motor commands commands to make it compatible with Neato Botvac D3.

driver.py (/opt/ros/kinetic/src/neato_robot/neato_driver/src/neato_driver) - gedit

Open

```
        #first time a 0-velocity is sent in, a velocity of 1,1,1 is sent. Then,
        #the zero is sent. This effectively causes the robot to stop instantly.
        if (int(l) == 0 and int(r) == 0 and int(s) == 0):
            if (not self.stop_state):
                self.stop_state = True
                l = 1
                r = 1
                s = 1
        else:
            self.stop_state = False
        self.port.write("setmotor lwheeldist "+str(int(l))+" rwheeldist "+str(int(r))+" speed "+str(int(s))+"\n")
        #self.port.write("setmotor "+str(int(l))+" "+str(int(r))+" "+str(int(s))+"\n")

    def readResponseAndUpdateState(self):
        """ Read neato's response and update self.state dictionary.
            Call this function only after sending a command. """
        response = self.readResponseString()
        for line in response.splitlines():
            #print(line)
            vals = line.split(",")
            if len(vals) >= 2 and vals[0].replace('_', '').isalpha() and vals[1].isdigit():
                self.state[vals[0]] = int(vals[1])
                #print(vals[0] , vals[1])

    def getMotors(self):
        """ Update values for motors in the self.state dictionary.
            Returns current left, right encoder values. """
        self.port.flushInput()
        self.port.write("getmotors\n")
        self.readResponseAndUpdateState()
        return [self.state["LeftWheel_PositionInMM"],self.state["RightWheel_PositionInMM"]]

    def getAnalogSensors(self):
```

Figure 14: Neato driver changes

The updated 3d model of neato with stand was defined in ROS using appropriate transforms and making changes in sdf files for RVIZ visualisation.

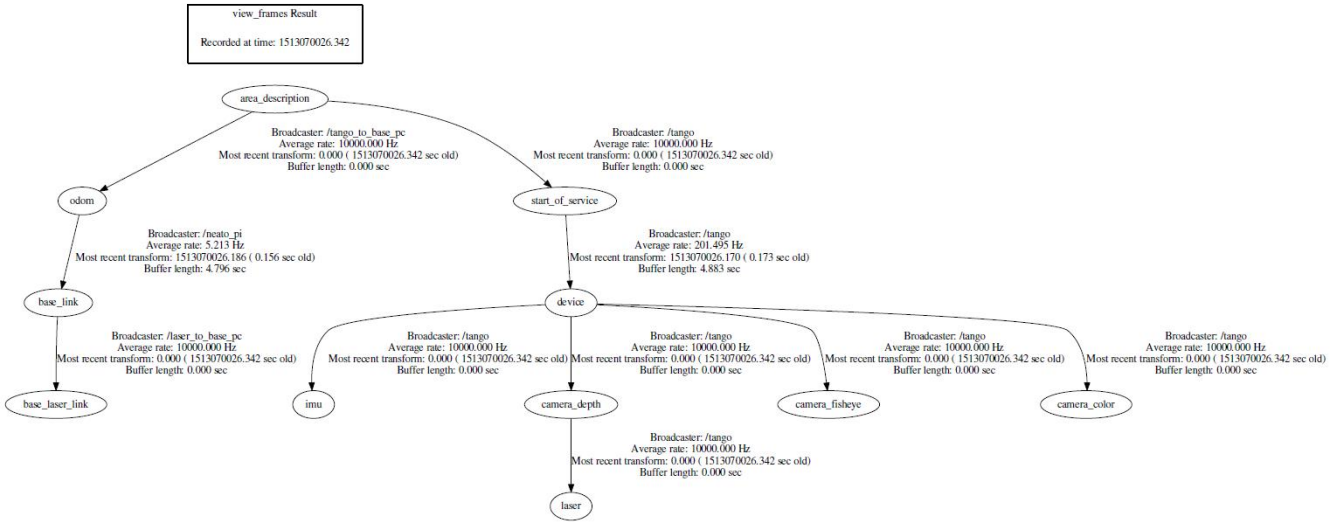


Figure 15: Transform tree

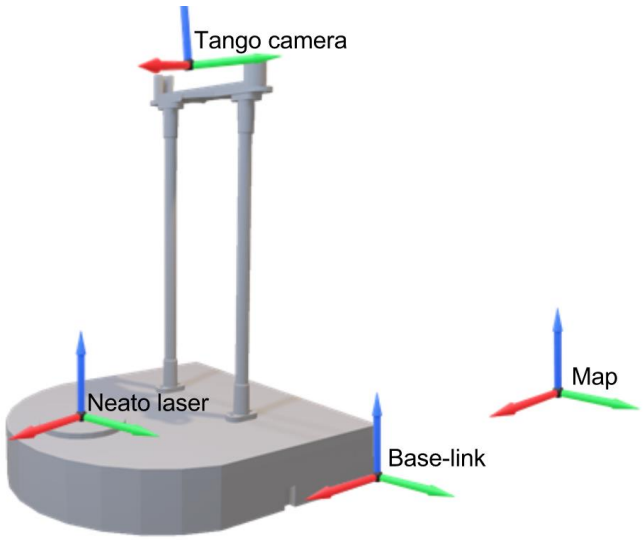


Figure 16: Transform Axis

4. Overall Process

Now to achieve the requirements the Robot had to first go around and map the surroundings using laser scanning and localisation. This was done through teleoperation. This was a very crucial step as the lesser accuracy in mapping obstacles would have given errors accumulating forward to make the system ineffective.

Next was to store this static map in yaml and pgm file and make it available for future use with a distinct name and id. Then the complete stored map was loaded back into the tango device once the robot was ready to navigate humans.
Now the LIDAR inside Neato was also turned on to detect new dynamic obstacles that would be created as the robot is moving on its calculated path.

The Neato Lidar's output was given to the local planner which in turn request the global planner to find new viable paths to destination if some new obstacles arise in the path made by global planner based on static map stored by the google tango device. Hence the path will adjust as per the changes in surroundings and google tango will register it in its static maps.

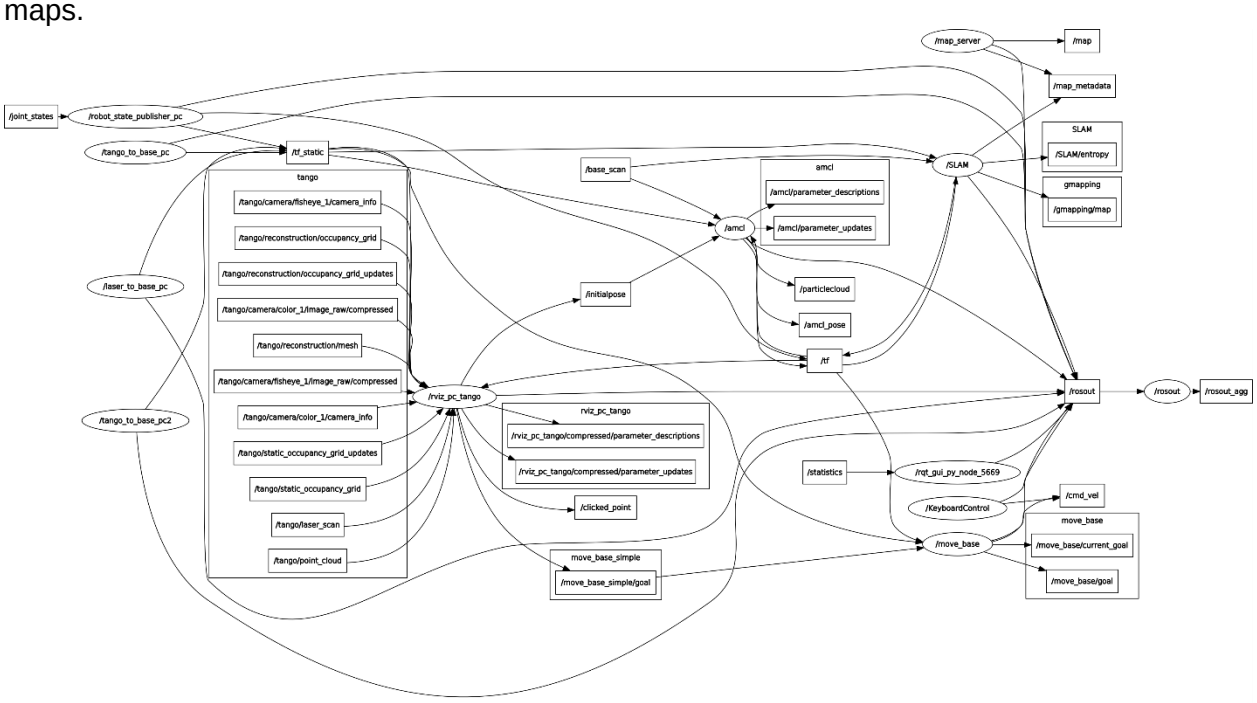


Figure 17: Software Architecture

5. Software and Algorithms

In developing this project a number of API's, opensource libraries and algorithms were tried and tested to select the one with best results. There were 2 main areas where these libraries were extensively utilized to get superior outputs:

- Mapping and Localization
- Navigation

1. Mapping and Localization

There are two approaches of doing it: online or offline. If the robot is localising itself as it moves in the surroundings and simultaneously and mapping its surroundings also its an online approach called as SLAM. There are various algorithms refered to using SLAM (refer

4). Based on results of this paper we found Hector SLAM and Gmapping as most valuable candidates. The hector slam doesnot require odometry reading which would be advantageous as our robot's odometry is very much susceptible to drift over time. On the other hand gmapping is found to be more robust and with least CPU load which would be major factor in giving desired outcome within least amount of time and error.
So we utilized the results stated in (refer 4) and experimentally compared the results in our hardware setup.

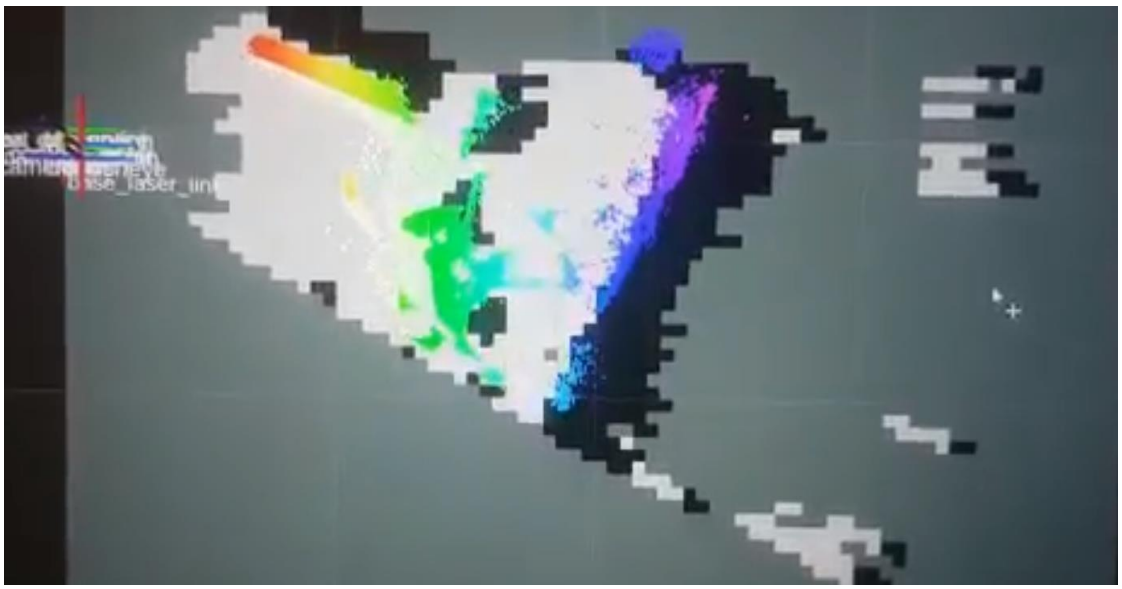


Figure 18: Hector Slam output

Figure 19: Gmapping output

There was one more approach to mapping using Google Tango's visual odometry and drift correction called as area learning. Tango device remembers the visual features of the area it has visited and uses them to correct errors in its understanding of its position, orientation, and movement. This memory allows the system to perform drift corrections (also called loop closures). After trying it out in our experiment the results were even better than gmapping. So, we decided to use the tango's approach for mapping and localization.

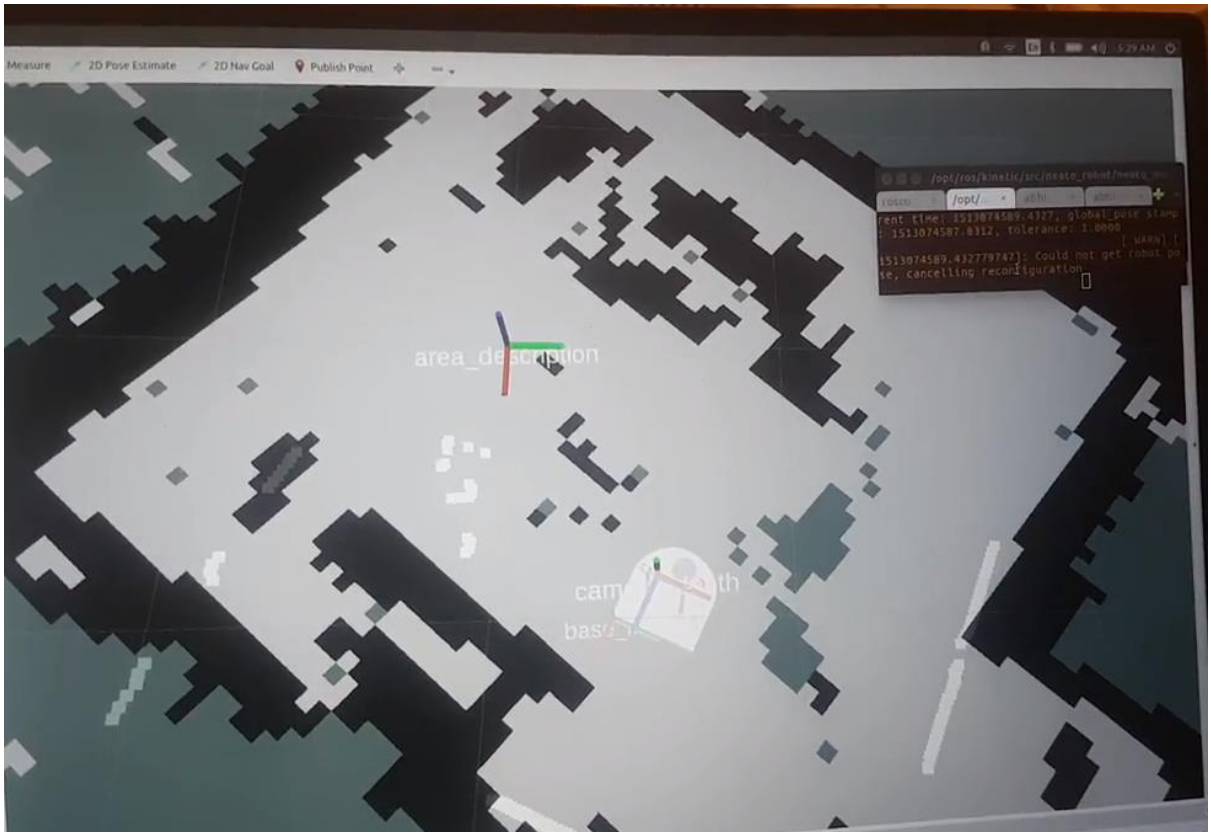


Figure 20: Tango VOI output

But even doing SLAM online was computationally very expensive that our small computer raspberry pi could not handle. Hence using map-server we stored the static maps to allow offline mapping and localisation. Using services and parameters definition of Tango API we were able to accomplish the task.

2. Navigation

Using ROS navigation Core library, we were able to do the proper path planning based on static maps following the local targets efficiently. But as the Robot moved due to problem of drift in odometry reading of robot and visual odometry of tango device results were skewed

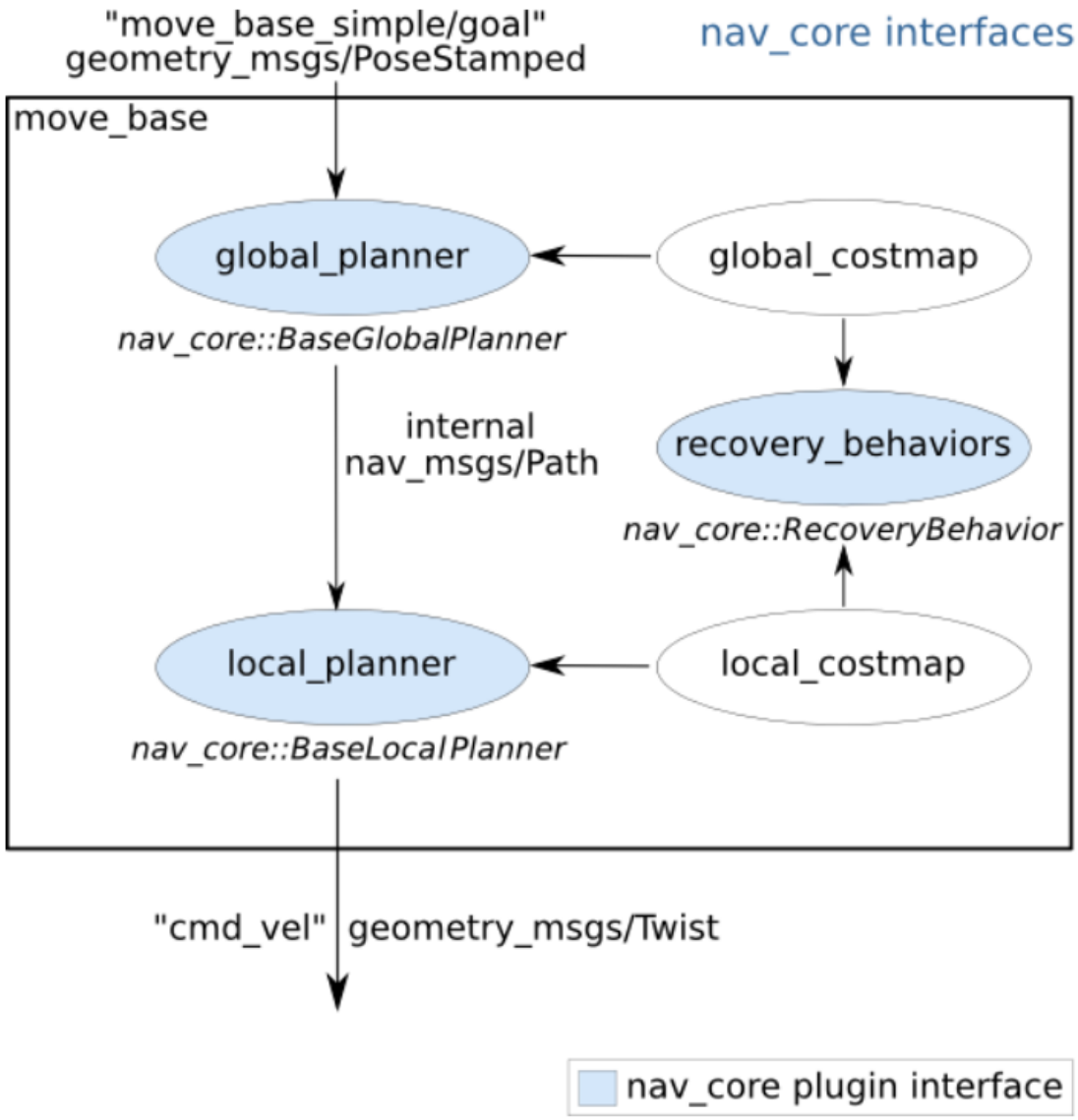


Figure 21: Navigation block diagram

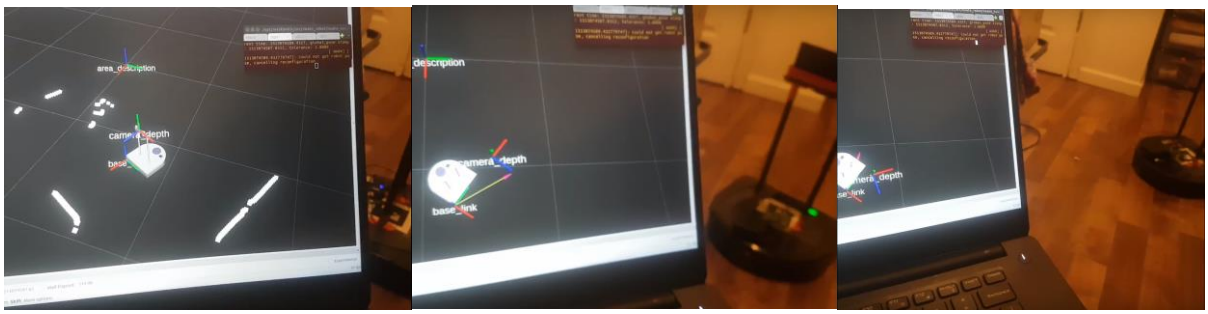


Figure 22: Drift in odometry

More work needs to be done to counter this error by converting visual odometry to base odometry and hence obtaining simulated precise odometry based on Tango vision and IMU sensor readings.

VII. Conclusion:

After testing the prototype multiple times, the system seems to function well. It has very low latency and accurate mapping and localization in static maps. Also it is able to adopt the planned path based on obstacles coming in real time and has fast rate of response. Although there are errors occurring due to drift in odometer readings between the base and sensor but there are many approaches with promising results which are yet to be applied and would solve this issue. Using highly robust Tango scanning and optimized mobile GPU and IMU raspberry pi is accurately able to understand its environment and do the path planning with rugged mobile base of neato. Moreover, the laser scanner of the neato can detect obstacles at high enough rate and take corrective action in all directions

References

1. https://www.sparkfun.com/news/490
2. https://www.neatorobotics.com/robot-vacuum/botvac-connected-series/botvac-d3-connected/?gclid=Cj0KCQiA38jRBRCQARIsACEqIeuv_Qz0A_ucGAPPLkzP4vITpVZj35tOj6MxJOFPUtmztBefig75zRMaAqI4EALw_wcB
3. http://wiki.ros.org
4. *https://pdfs.semanticscholar.org/6b9c/afcf9aef5b4c0c338c44a581236d54caddbd.pdf*