**Proof of Concepts & Technology demonstration**

**INTERNAL LOGISTICS**

**Team**

Shamsher Singh Thind

Suji Grena Edwin RajKumar

Change history

|  |  |  |  |
| --- | --- | --- | --- |
| **Version Number** | **Change Description** | **Author** | **Effective Date** |
| 0.1 | Initial draft | Suji Grena | Oct 25, 2018 |
|  |  |  |  |

Abstract

AGV for transportation of material within an manufacturing environment

- Mixed environment of robots, humans, and other machines

Continuous supply of tools and components are an essential aspect of any manufacturing plant for smooth production. Supply quantity and timings needs to be precise to keep assembly lines clutter free. This makes it extremely important for the plant operations team to ensure proper availability of such materials at all stations within manufacturing plant at all times. Delivering of tools and components to each and every station involves challenge of maneuvering in risky areas cluttered with heavy machinery with numerous moving parts. It becomes challenge to operate this manually.

Thus various line following robots are currently used which are programmed to follow a predefined set of paths between various stations. Although line following robot de-risks logistics operations within warehouse to quite some extent but they bring their own set of challenges. Line adherence limits the capability of robots to navigate to new areas without any infrastructural change. They also can’t bypass any obstacle on their way thus involving additional manual efforts to keep their pathway free.

This project involves in removing dependency of robots on any sort of fixed route guidance system and making them autonomous so as to navigate to various stations on assembly lines without sticking to any fixed paths. This project involves exploring low cost sensors with state of the art algorithms and techniques to localize within warehouse, planning a path to any desired station and navigating in a controlled manner. This project brings in amalgamation of robotics, mathematical modeling and machine learning to achieve precision in indoor autonomous navigation.

Table of Contents

[Problem Statement 7](#_Toc527962301)

[Solution Introduction 8](#_Toc527962302)

[AGV Architecture 9](#_Toc527962303)

[Hardware landscape 9](#_Toc527962304)

[Motors 9](#_Toc527962305)

[Arduino Uno based controller board 9](#_Toc527962306)

[Raspberry Pi 10](#_Toc527962307)

[Wheel Encoders 10](#_Toc527962308)

[10](#_Toc527962309)

[IMU 11](#_Toc527962310)

[RP Lidar A2 11](#_Toc527962311)

[Wi-Fi Modules 11](#_Toc527962312)

[AGV 12](#_Toc527962313)

[Software pipeline 13](#_Toc527962314)

[Mapping 16](#_Toc527962315)

[Localization 17](#_Toc527962316)

[Wi-Fi footprint using Machine learning model. 17](#_Toc527962317)

[Machine learning Modelling 17](#_Toc527962318)

[Odometry from Wheel Encoders and IMU 18](#_Toc527962319)

[Wheel encoders. 18](#_Toc527962320)

[IMU 18](#_Toc527962321)

[Odometry Fusion. 18](#_Toc527962322)

[AMCL 18](#_Toc527962323)

[Kinematic Vehicle model 19](#_Toc527962324)

[Path Planning 21](#_Toc527962325)

[Problem Formulation 21](#_Toc527962326)

[Proposed Approach 21](#_Toc527962327)

[Heuristics considered 22](#_Toc527962328)

[Lookup Table 22](#_Toc527962329)

[Implementation Details 23](#_Toc527962330)

[Motion Control 24](#_Toc527962331)

[MPC Formation: 24](#_Toc527962332)

[MPC Flow 27](#_Toc527962333)

[Environment Setup 28](#_Toc527962334)

[Wi-Fi Data Collection and Modelling 28](#_Toc527962335)

[AGV Sensor Connections 28](#_Toc527962336)

[Wheel Encoders 28](#_Toc527962337)

[Arduino Controller Setup 28](#_Toc527962338)

[Raspberry Pi Setup 28](#_Toc527962339)

[On Board Computer Setup 29](#_Toc527962340)

[Remote Computer Setup 29](#_Toc527962341)

[Setup Validation 30](#_Toc527962342)

[Appendix 32](#_Toc527962343)

[Challenges & solutions 32](#_Toc527962344)

[TF Tree 33](#_Toc527962345)

[Topics and Data Types 33](#_Toc527962346)

[Artifacts 34](#_Toc527962347)

[Future Enhancements 35](#_Toc527962348)

[References 35](#_Toc527962349)

[Index 36](#_Toc527962350)

**TABLE OF FIGURES**

[Figure 1Motor 8](file:///D:\AV\POCs\NonMagneticTransportation\Documentaion\POC_doc.docx#_Toc527551592)

[Figure 2Arduino Uno based controller board 8](file:///D:\AV\POCs\NonMagneticTransportation\Documentaion\POC_doc.docx#_Toc527551593)

[Figure 3Raspberry Pi 3 9](file:///D:\AV\POCs\NonMagneticTransportation\Documentaion\POC_doc.docx#_Toc527551594)

[Figure 4Rotatory Wheel encoder representation 9](file:///D:\AV\POCs\NonMagneticTransportation\Documentaion\POC_doc.docx#_Toc527551595)

[Figure 5Pixhawk PX4 10](file:///D:\AV\POCs\NonMagneticTransportation\Documentaion\POC_doc.docx#_Toc527551596)

[Figure 6RP Lidar 10](file:///D:\AV\POCs\NonMagneticTransportation\Documentaion\POC_doc.docx#_Toc527551597)

[Figure 7AGV architecture software pipeline 12](#_Toc527551598)

[Figure 8Hector SLAM flow 15](file:///D:\AV\POCs\NonMagneticTransportation\Documentaion\POC_doc.docx#_Toc527551599)

[Figure 9Map generated using hector SLAM 15](file:///D:\AV\POCs\NonMagneticTransportation\Documentaion\POC_doc.docx#_Toc527551600)

[Figure 103D plot of signal strength for 8 Wi-Fi modules 16](file:///D:\AV\POCs\NonMagneticTransportation\Documentaion\POC_doc.docx#_Toc527551601)

[Figure 11Wi-Fi field map 16](file:///D:\AV\POCs\NonMagneticTransportation\Documentaion\POC_doc.docx#_Toc527551602)

[Figure 12Kinematic model of vehicle 18](file:///D:\AV\POCs\NonMagneticTransportation\Documentaion\POC_doc.docx#_Toc527551603)

[Figure 13Model of AGV with trolley 19](#_Toc527551604)

[Figure 14A\* Searched Map 22](file:///D:\AV\POCs\NonMagneticTransportation\Documentaion\POC_doc.docx#_Toc527551605)

[Figure 15Input Map 22](file:///D:\AV\POCs\NonMagneticTransportation\Documentaion\POC_doc.docx#_Toc527551606)

[Figure 16Output Path from Hybrid A\* 22](#_Toc527551607)

[Figure 17Bottom view of AGV bot in rviz 29](#_Toc527551608)

[Figure 18Side view of AGV bot in rviz 30](#_Toc527551609)

[Figure 19Top view of AGV bot in rviz 30](#_Toc527551610)

[Figure 20TF Tree 31](file:///D:\AV\POCs\NonMagneticTransportation\Documentaion\POC_doc.docx#_Toc527551611)

# Problem Statement

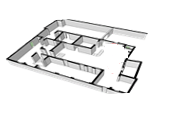
AGV’s can efficiently solve the problem of towing payloads between manufacturing stations but providing such autonomous capabilities to any vehicle is a challenging task. It requires highly synchronized set of nodes running state of the art algorithms and machine learning models , doing numerous tasks from state estimation, path planning, object detections and navigating vehicle to reach a desired goal and all that using cost effective sensors.

Purpose of this POC is to design and prototype a compact and cost effective AGV with sensor based localization and navigation system to tow a payload within closed ecosystem such as assembly lines in manufacturing plants.

AGV should adhere to the constrains and limitations imposed by an actual physical movement of a vehicle.

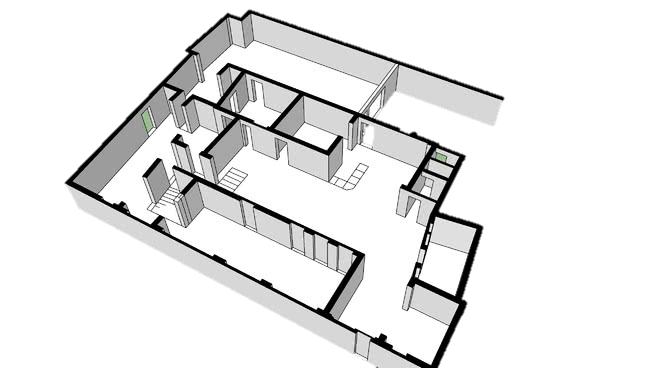
# Solution Introduction

Autonomous indoor navigation system involves usage of sensors and algorithms to create a map of the dynamic environment, localizing in the created map, plan a path to the desired goal Motion Control to reach the destination.

**Mapping**

Mapping is the process of creating a 2d representation of the given environment with indicates free path to navigate and the static obstacles that are to be avoided.

**Localization**



Localization is the process of identifying vehicles position involving x, y co-ordinates, orientation and other motion parameters on a given map.

**Path Planning**



Path planning involves determining an obstacle free and optimal path from current position of vehicle to a given goal.

**Motion Control**



Motion control drives vehicle to goal position by adhering to path planned while keeping all vehicle and path constraints in consideration.

Together these modules along with sensory inputs gives capability to a vehicle to navigate autonomously with in an indoor eco-system.

# Robot Architecture

## Hardware landscape

### Hokuyo UST 20LX

RP Lidar is used for getting laser scan of the environment around the vehicle. RPLIDAR A2 which is incorporated in the vehicle is a next generation low cost 360-degree 2D laser scanner (LIDAR). It can take up to 4000 samples of laser ranging per second with high rotation speed.

*Features*:

* Distance range – up to 20 meters
* 270-degree angle range
* 0.25° degree of angular resolution
* Sampling frequency of 2000 – 4000 Hz
* Scan rate of 40Hz
* Powered through 12V DC or 24V DC
* Ethernet 100BASE-TX Interface

Figure 6 Hokuyo Lidar

### Create Robot

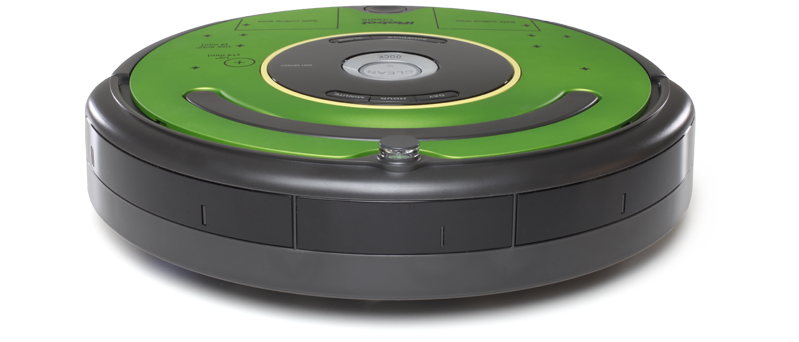


Figure 6AGV with trolley

Figure 7Cut sectional view of AGV

## Software pipeline

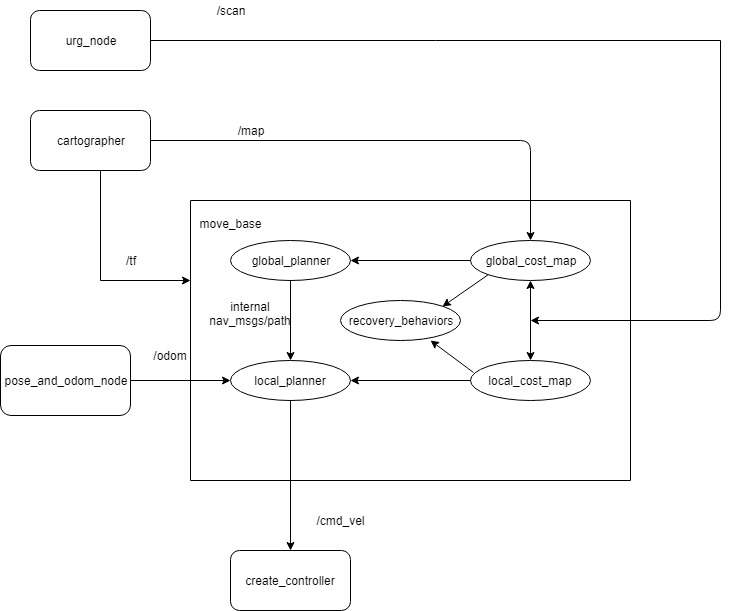


Figure 7AGV architecture software pipeline

* **urg\_node**

urg\_node is a ROS node running on laptop mounted on robot to get scan data using Hokuyo Lidar.

Subscribes : None

Publishes : /scan

* **cartographer**

cartographer is the important ROS node running on remote desktop connected to the on-board computer. It receives the scan from hokuyo lidar and creates the map of the environment in mapping node and gives the pose and odom estimate in localization mode.

Subscribes : /scan

Publishes : /map

* **pose\_and\_odom\_node**

pose\_and\_odom\_node is a custom written ROS node running on laptop mounted on robot to calculate and emit the pose and odometry of the robot using the laser scan matching emitted by cartographer node

Subscribes : None

Publishes : /odom\_carto

/pose\_carto

* **global\_costmap**

global\_costmap is a ROS node running on laptop mounted on robot to create the dynamic cost map which creates penalizes the obstacles in the global map.

Subscribes : None

Publishes : /plan

* **local\_costmap**

local\_costmap node running on laptop mounted on robot to create the dynamic cost map which penalizes the dynamic obstacles that occur in the environment that are not present in the global map.

Subscribes : /scan

/odom\_carto

Publishes : /global\_plan

/local\_plan

* **hector\_slam**

hector\_slam is a ROS package running on remote computer unit that fuses 2D lidar scan data to generate map. hector\_slam package is available at [Ref. Section:]

Subscribes : /scan

Publishes : /map

* **map\_server**

map\_server is a ROS node running on remote computer unit that publishes map generate by hector slam using 2D Lidar scan.

Subscribes : /None

Publishes : /map

* **amcl**

amcl is a ROS node running on remote computer unit that uses particle filter approach to generate current pose of the AGV on a given map.

Subscribes : /scan

/map

/tf

Publishes : /amcl\_pose

* **make\_plan**

make\_plan is a ROS node running on remote computer unit that consumes service from Path planner Server to generate trajectory from current position of AGV to a desired goal position published by rviz node.

Subscribes : /map

/movebase/simple/goal

/amcl\_pose

Publishes : /path

* **rviz**

rviz is a ROS visualizer that displays various type of sensor data and state information from ROS. It is also used to select a goal position for AGV.

Subscribes : /map

/amcl\_pose

/Odometry/filtered

Publishes : /movebase/simple/goal

* **mpc\_control**

mpc\_control is a ROS node running on remote computer unit that computes desired velocity of left and right wheels of AGV independently so as to make AGV follow trajectory generated by make\_plan node. It considers AGV’s state and vehicle model for generating desired velocities.

Subscribes : /path

/amcl\_pose

/acc/filtered

/Odometry/filtered

Publishes : /wheel\_velocity

* **PID**

PID is a controller node running on Arduino based controller that is used to generate PWM output based on given current state of AGV velocity and desired Velocity computed by MPC controller.

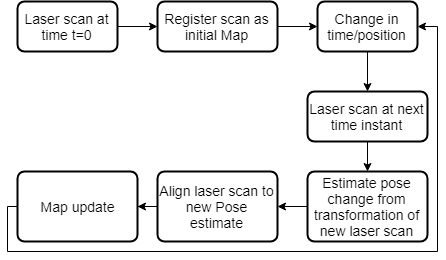
Subscribes : None

Publishes : None

# Mapping

Hector mapping is a SLAM approach that can work solely using Laser scan matching without odometry, or in platforms that exhibit roll/pitch motion or both. This can be attributed to the availability of high update rate of modern LIDAR systems. Hector SLAM provides 2D pose estimates at scan rate of the sensors.

The major drawback of this approach is that the system does not provide explicit loop closing ability, it is sufficiently accurate for many real-world scenarios. This is because the mapping depends purely on successive scan matching without getting any feedback from the sensors in the robot.



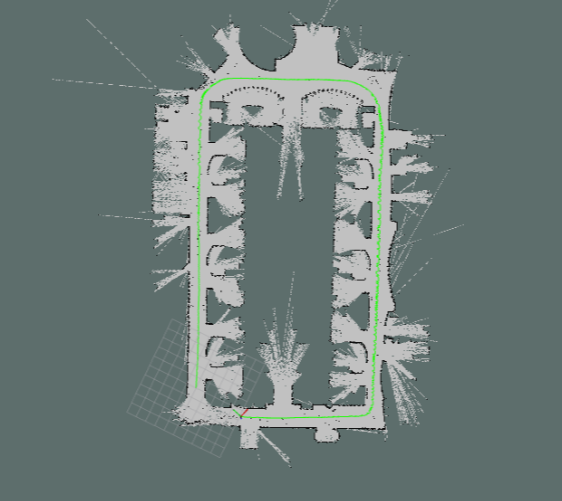
Hector SLAM works based on scan matching called ICP (Iterative Closest Point) matching. This is can explained as the matching of successive scan, by trying to match the closest points in the current scan with the scan in the previous time frame. It tries to find as much as points that are present in both the scans, thereby trying to find a correlation between two scans. Correlation here is determined using two matrices called the translation and rotation matrices respectively.

Figure 8Hector SLAM flow

Figure 9Map generated using hector SLAM

*Figure 9Map generated using hector SLAM* shows the results of RP LiDAR A2, 360-degree 2D laser scanner, with frequency 10hz (600rpm) thus, resolution will be 0.9° and within a 6-meter range yielded a sparse map after parameter tuning.

# Localization

AGV is using a two stage process for an efficient and cost effective indoor localization. In first stage Wi-Fi Signal strength footprints combined with machine learning is used to narrow down to a zone with ~4 meters of accuracy.

In stage two, sensory readings from wheel odometers and IMU are fused together to find precise pose on a map with initial approximate inputs from localization done during phase one.

## D:\AV\WiFi-RFP\fieldMap.pngMachine learning Modelling

For the purpose of this POC, entire region 15X30 meters was divided into 36 zones (4X9) and 8 Wi-Fi modules were installed at multiple locations. Multiple Wi-Fi signal strength readings along with demarcated zone ID’s were taken for all these 8 Wi-Fi modules using Raspberry Pi 3 and a custom *Wi-Fi data collection script***.**

This data was then cleansed to iron out any missed reading and randomization.

Collected and processed data was then used to train machine learning model. Signal strengths from all the readings were used as training i/p parameters and demarcated zone ID’s o/p. Multiple ML models were trained and best validation accuracy was achieved using Gaussian NB.

Figure 11Wi-Fi field map

Combined accuracy of top two zone predictions by trained model was more than 96%.

In Autonomous mode, Wi-Fi signal strengths from all evaluated using the trained model. Zone ID received from the trained model is then converted to zone center coordinates w.r.t Map frame using Zone Coordinates lookup table. These coordinates are then published to be consumed by AMCL to initiate particle filter process.

## Odometry from Wheel Encoders and IMU

During stage two of localization two sensors (Wheel encoders and IMU) are used to generate Odometry.

### Wheel encoders.

Two Wheel encoders present on right and left rear wheels of AGV generates ticks whenever wheel rotates. These ticks are converted into linear and angular velocity, orientation quaternion and X, Y co-ordinates. This is published as Odometry topic /odom *[Ref. Topics and Data Types].*

### IMU

One IMU was used to produce IMU readings including angular velocity, linear accelerations etc . Constant readings from IMU are published which in turn is subscribed to fuse along with readings produced from wheel odometers.

## Odometry Fusion.

Readings generated by any sensor are prone to some degree of variance. This error is introduced either by hardware limitations or environmental factors. Similarly, readings obtained from Wheel encoders include some amount of slippage in tires. This slippage makes readings polluted and thus can’t be used directly for any inferences.

Readings from both IMU and Wheel encoders are fused together using EKF[[1]](#footnote-1) (Extended Kalman Filter).A ROS package ***Robot Localization***[[2]](#footnote-2) provides a solution to fused multiple readings from IMU and Wheel encoders using either EKF or UKF. For the purpose of this POC, EKF mode was used to fused obtained readings.

EKF configurations were tweaked to adjust to the systems in use. A combined Odometry readings with lesser degree of uncertainty as compared to individual readings were published under topic ***/odometry/filtered.*** Additionally, ***/accel/filtered*** and transformation between ***base\_link*** and ***odom*** frame is also published under topic ***/tf.***

## AMCL

AMCL is a probabilistic localization system for a robot moving in 2D. It implements the adaptive Monte Carlo localization approach which uses a particle filter to track the pose of a robot against a known map.

A ROS package amcl [[3]](#footnote-3)provides AMCL for pose estimation. It uses **/tf** produced by Robot localization, 2D Lidar scan available under topic **/scan** from RP-Lidar and pre-computed 2D Lidar based map produced using Hector Slam *[Ref: Mapping****]*** to get pose estimate of AGV in map frame.

ACML uses initial pose estimate from Wi-Fi based localization to spread particles in conjunction with 2D Lidar points from **/scan** topic mapped against /**map**. With change in pose of AGV due to change in positon or orientation, particles are converged to get appropriate estimate of AGV’s pose. This localized is published under **/amcl\_pose** topic.

# Kinematic Vehicle model

Kinematic modeling is the study of the motion of mechanical systems without considering the forces that affect the motion. For the DDMR, the main purpose of kinematic modeling is to represent the robot velocities as a function of the driving wheels’ velocities along with the geometric parameters of the robot.

The linear velocity of each driving wheel in the Robot Frame is therefore, the linear velocity of the DDMR in the Robot Frame is the average of the linear velocities of the two wheels

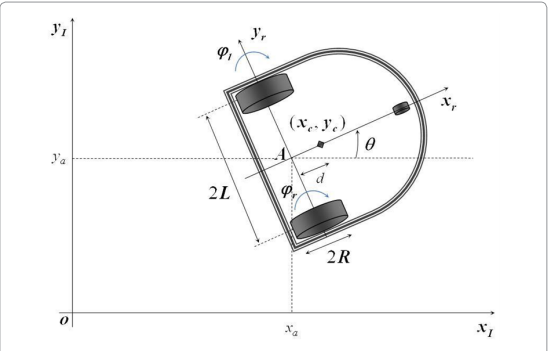
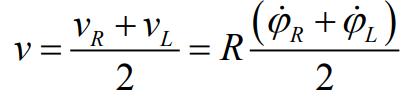
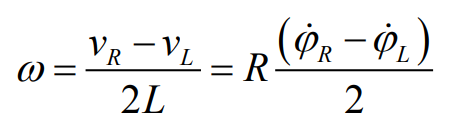


Figure 12Kinematic model of vehicle

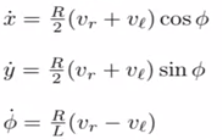
**Linear Velocity:**

****

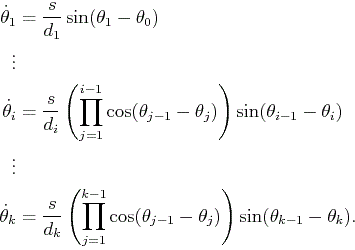
**Angular Velocity:**

****

**Vehicle Model for Differential Drive AGV:**

****

**Vehicle Model for Differential Drive AGV with trolleys:**



x˙= {R (VR + VL) Cos Ø } / 2

y˙= {R (VR + VL) Sin Ø } / 2

Ø˙= {R (VR - VL )}/L

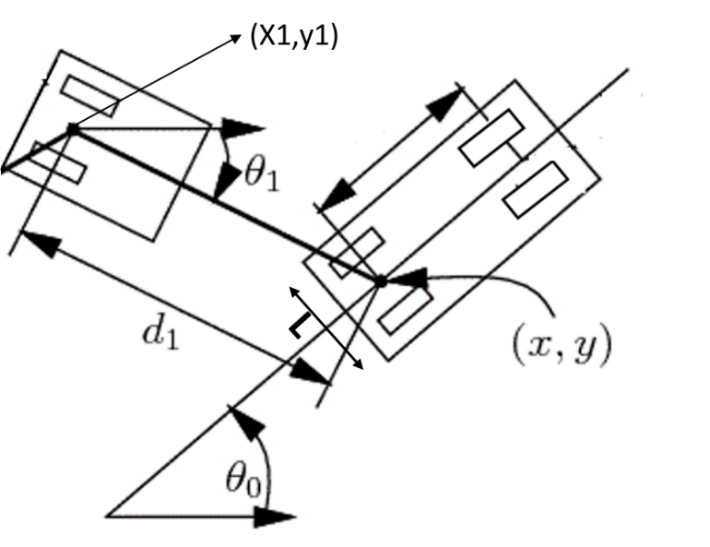


Figure 13Model of AGV with trolley

* v=linear velocity
* ω = angular Velocity
* r= radius of wheel
* l=distance between the left and right wheel of the vehicle
* d1=distance between AGV and trolley

AGV Kinematics:

* V=r/2\*(vr+vl)
* ω =(r/l)\*(vr-vl)
* X0t+1=X0t + Vt \* dt \* CosΘ0(t)
* Y0t+1=Y0t + Vt \* dt \* SinΘ0(t)
* Θ0(t+1)= Θ0(t) + ωt \* dt

Trolley Kinematics

* Θ1’= V/d1 \* sin(Θ0-Θ1 )
* Θ1= Θ1 + Θ1’
* X1=X0 -d1 \* Cos(Θ1)

Y1 =Y0-d1 \* Sin(Θ1)

# Path Planning

Path planning is the method of finding the optimal path from one point to another in space. It optimizes the path between source and destination by determining shortest path between them. It is sometimes also termed as motion planning as it helps to decide the motion of any object in an environment. An object can be a robot which is autonomous in nature as it makes use of the path finding algorithm to determine its traversing points in space. Such a robot is referred to as mobile robot. Path planning can also be defined as the process of breaking down a desired path into number of iterative steps to make discrete motions to optimize some entities. Here we address Path Planning in a static environment for non-holonomic car like robot, with trolleys.

## Problem Formulation

The Path Planning problem can be stated as follows: Given a map ***M****,* find the cost-optimal path from a vehicle’s current location ***robot\_initial*** including the pose*,* to a goal location ***goal*** subject to the constraints given by ***Vehicle and the Trailer Kinematics***. The discrete grid map ***M*** is an occupancy grid of resolution 0.05 obtained from /map\_server node in ROSconsisting of values – [0, -1, 100] where, 0 – free space, -1 – unexplored region and 100 – obstacle.

The Path generated should be optimal (least cost), traversable (considering vehicle constraints), collision free (from static obstacles) and inbound collision free (collision among trolleys and robot).

## Proposed Approach

1D Occupancy grid

map dimensions

num\_trolley

Input

motion\_matrix

A\* search

Is

Goal Reachable

no

Exit

no

robot & Trolley kinematics

yes

yes

Path

Output

Is

Goal Reachable

Hybrid A\*

***A\**** algorithm, which is limited to discrete state spaces is used for generating a *Cost Map* i.e. a grid, with each cell having a cost to reach the goal. Along with generating a cost map it’s also used to identify whether a goal is ‘Reachable’ or ‘Not Reachable’. The exploration of neighboring cells is determined by ***motion\_matrix.***

***Hybrid A\**** algorithm is used for generating a continuous path in the region searched by A\*.

The continuous path is a result of the kinematic equations []

Each node *n* of the search graph i.e. the state of the robot is completely defined by,

*robo\_state = (x, y, psi (orientation), g, h, f, v, w, vr, vl, num\_trolley, trolley\_list)*

trolley\_list – is a list of trolley states, of length equal to num\_trolley

*trolley\_state = (x, y, psi, index)*

Hybrid A\*

motion

A\* motion

Parameters considered for path planning here are,

* Robot : height, width, wheel radius
* Trolley : height, width, distance between trolley and robot
* v - linear velocity (considered as constant)
* w - angular velocity (augmented for n values in range ‘min angle’ and ‘max angle’)

## Heuristics considered

* A\* is used to search for the start position from the goal in a grid, having the cost of goal equal to zero and the neighboring expansions incremented by unit value. This helps in validating whether a goal is *Reachable* or Not.
* If ‘Reachable’ then every cell in the grid is stored with value needed to reach the goal from that cell (***h***).
* ***g*** – distance between the start to the current cell (Euclidean considered)
* ***heading\_cost*** – cost added based on the magnitude of difference in the *psi* value of current and previous states.
* ***collision\_cost*** – cost added in case of boundary collision and inbound collision
* ***forward\_cost*** – cost added by calculating if the future states are collision free
* ***f*** – total cost (***g + h + heading\_cost + collision\_cost + forward\_cost***)

On every iteration of calculating next states, the list containing these states is sorted based on ***f*** value.

## Lookup Table

Here we consider the map to have a set of anchor points (i.e. is predefined set of locations where the robot have to pass from). Since the *Map* is static the search algorithms are used and the paths are generated between these anchor points. The paths are then stored in a table and used for future references, hence reducing the computation for generation of paths on every job.

The table here is a directional weighted graph, with nodes as the Anchor points and weights as the path cost. The path returned would consider the least cost using ***Dijkstra’s Algorithm*** to reach the end node and the orientation of the start node.

## Implementation Details

The search algorithms are implemented using the ***Numpy*** operations in Python (*version- 3.5*) for reducing the computation cost. Framework used for coding is ***Jupyter Notebook/ Google Colab.*** Output is visualized using the ***Matplotlib*** package.

Input Parameters : *map (1*D occupancy gridfrom /map\_server node in ROS), *map\_width, map\_height,*

*robot\_initial (x, y, psi), goal (x, y), num\_trolley*

Output : *Path (List of Robot states)*



Figure 14A\* Searched Map

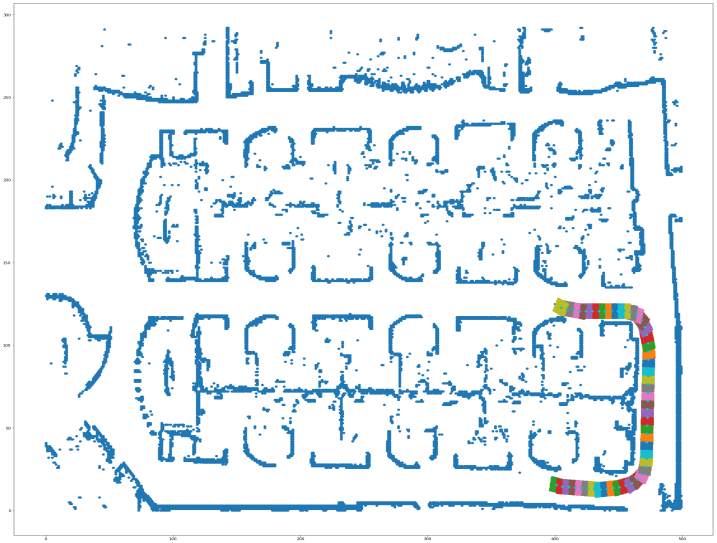
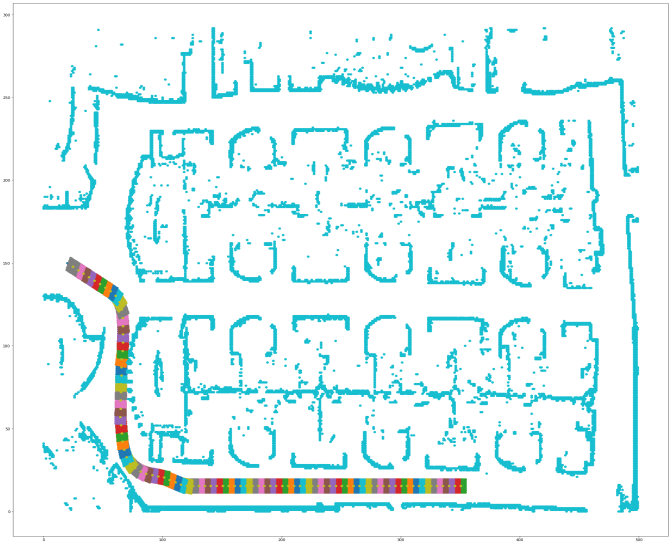
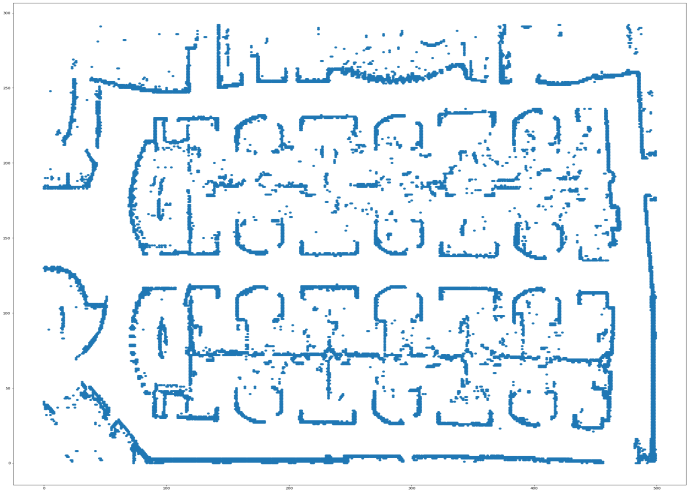


Figure 15Input Map

Figure 16Output Path from Hybrid A\*

# Motion Control

AGV used for the indoor navigation has four wheels, each of which is driven by its own motor and steers by adjusting the rotational speed of the motors, this configuration is known as differential drive motors. To control the motion of the AVG, the control reference would be the velocity of the robot and the rotation of the wheels is required to move the robot

## MPC Formation:

Predictive Control is composed by the following components:

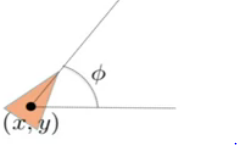
Predictive Model

The control of the AGV contains a model of the relationship between the velocity of the robot and the rotation of the wheels. Before describing the kinematics of the AGV, one need to know the dimensions of the AGV. For this experiment we have wheel separation as **\*\*\*\*** and radius of the wheels as **\*\*\*\*.**

Differential drive kinematic equations:

Assuming that

* VR – Velocity of right wheel
* VL – Velocity of left wheel
* R – Radius of the wheel
* L – Wheel separation (Distance between two wheel)
* (x,y) – Current position of the AGV
* Ø – Orientation of the AGV (Heading direction)



State transition equation (as discussed in the Vehicle model) is:

x˙= {R(VR + VL )Cos Ø } / 2

y˙= {R(VR + VL )Sin Ø } / 2

Ø˙= {R(VR - VL )}/L

As above equations are terms of right & left wheel velocity doesn’t seem very intuitive to think in terms of the rates of the various wheels, due to which we went ahead with the unicycle model which implements the differential drive equation underneath it.

The final equation used in the MPC vehicle model:

Taking into the consideration of the translational and the angular velocities

* V – Translational velocity
* Ω – Angular velocity

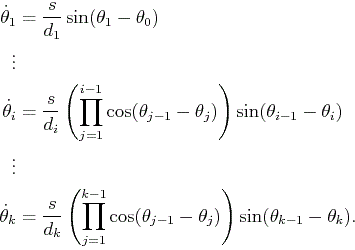
Dynamics:

* x˙= V Cos Ø
* y˙= V Sin Ø
* Ø˙= Ω

And getting the VR and VL as:

* VR = {2V + ΩL}/(2 \* R)
* VL = {2V - ΩL}/(2 \* R)

Since we have trolley attached to the AGV, equation of that one is



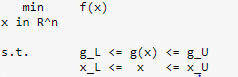
Objective cost function:

This function is used as the criteria to optimize by the optimization problem associated to the predictive controller. It functions expose the cost of a certain evolution along the predictive horizon N. Usually, its form is:



L Stage (·) is the stage cost and V Term (·) is the terminal cost. ¯u(k) represent the future control action along the predictive horizon calculated at instant k.

For our purpose we have used the Ipopt (Interior Point OPTimizer) for large-scale ​nonlinear optimization It is designed to find (local) solutions of mathematical optimization problems of the from



where f(x): R^n --> R is the objective function, and g(x): R^n --> R^m are the constraint functions. The vectors g\_L and g\_U denote the lower and upper bounds on the constraints, and the vectors x\_L and x\_U are the bounds on the variables x. The functions f(x) and g(x) can be nonlinear and nonconvex, but should be twice continuously differentiable. Note that equality constraints can be formulated in the above formulation by setting the corresponding components of g\_L and g\_U to the same value.

**Constraints:**

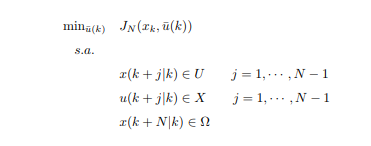
This equation expresses a set of additional conditions which must be satisfied along the predictive horizon. Normally this additional condition is a group of belonging sets associated to the optimization problem variables. For example, it’s typical that the state and the control actions move themselves inside of a limit or a region. This constraint must be imposed due to physical limits or operational limits of the real system. This constraint is usually expressed as closed and bounded sets X and U where states and control actions must be inside.

For the current scenario we are introducing the following constraints in the MPC system:

1. Maximum & minimum affordable transactional velocity is 8 m/sec, 0m/sec respectively
2. Maximum affordable angular velocity is
3. Maximum & minimum linear acceleration is 1, -1 respectively
4. Penalizing the cost when

* Vehicle orientation is not aligned with the predicted path orientation
* Vehicle is very far from the predicted path
* Vehicle is not following the reference velocity
* Any abrupt change in the acceleration and the angular velocity.

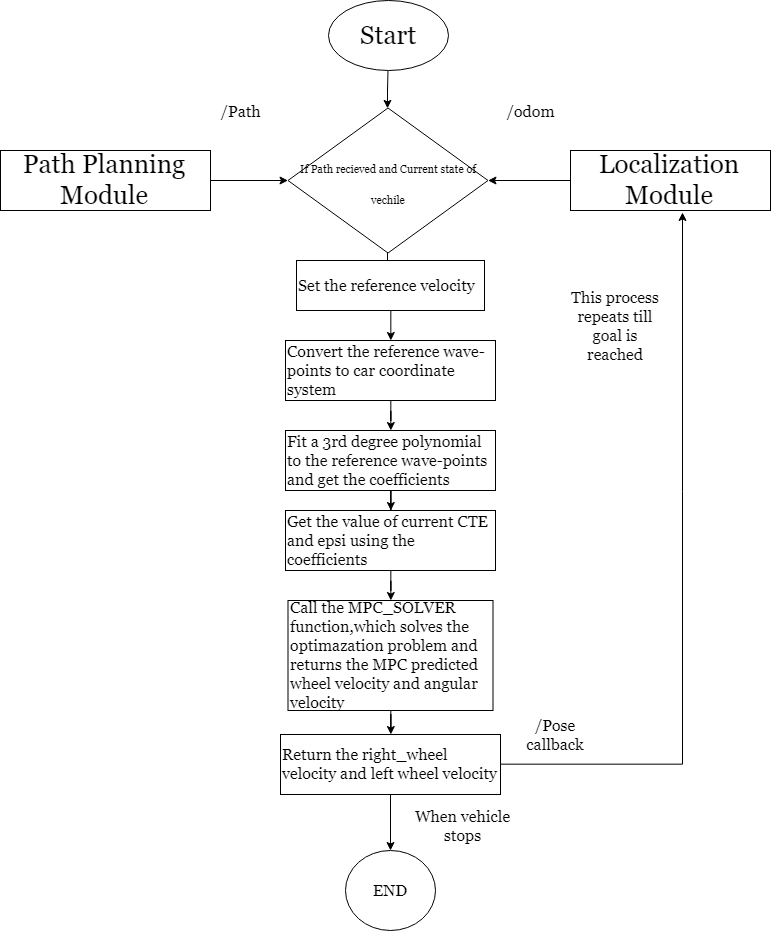
A typical formulation for the optimization problem associated with the MPC could be:



Inputs required by MPC is the current state of the AGV which includes the following quantities:

1. Current co-ordinates of AGV along with the orientation
2. Current translational velocity
3. Current angular velocity
4. Current acceleration

## MPC Flow



# Environment Setup

## Wi-Fi Data Collection and Modelling

* Use multiple Wi-Fi hotspot modules kept at eight different places with each having unique ID.
* Use *Wi-Fi data collection script* to collect Wi-Fi signal strengths of all the hotspots.
* Annotate signal strength collected from each zone needs with the zone ID’s.
* Train machine learning model using *iPython notebook for training ML model.*
* Save trained model to a pickled file.

## AGV Sensor Connections

### Wheel Encoders

* Wheel encoder has four output cables. Voltage, Ground and two data pins.
* Voltage of 5V is supplied from raspberry pi's pin no.2.
* Ground is connected with the pin no.6.
* The only data pin from wheel encoder that is used for data collection is connected with pin no.3 (left wheel's encoder) and pin 11(right wheel's encoder).

## Arduino Controller Setup

The Arduino based controller board is connected with raspberry pi through USB port with a USB to USB connector to aid serial communication. The code that runs on the controller board can be downloaded from following *Arduino Controller Code build.* It can be uploaded into the controller board through latest version of the Arduino IDE.

## Raspberry Pi Setup

Before setting up Raspberry Pi ensure there is at-least 8 - GB of external memory card available.

Setup Raspberry Pi with following OS and software

* + RASPBIAN STRETCH WITH DESKTOP - v4.9
  + kinetic-ros-base for debian os

You can use ***Script to setup ROS on raspbian*** script to setup ROS on raspberry Pi.

*After installation, RP Lidar package was built from source. As this is a base version of ros-kinetic distro, some packages might be missing for a successful build of RPLidar package.*

*You can make use of ROSINSTALL or ROSDEP to have a successful build.*

*-------with ROSINSTALL-------*

*->rosinstall\_generator <missing\_package\_name> --rosdistro kinetic --deps --wet-only --tar > kinetic-<missing\_package\_name>-wet.rosinstall*

*->wstool init src kinetic-<missing\_package\_name>-wet.rosinstall*

*-------with ROSDEP----------*

*navigate bash to the top dirctory of workspace*

*->rosrun install --from-paths src --ignore-src -r -y*

Once it is installed and raspberry pi is booted, download *Raspberry Pi code build* package which contains required ROS nodes to a catkin workspace.

Once downloaded , build packages using catkin\_make.

Navigate to catkin workspace and launch AGV package using following command:-

*roslaunch agv agv.launch*

## On Board Computer Setup

* Install following dependencies on onboard Computer:-
  + Ubuntu version: 16.0+
  + ROS version : kinetic
* Once on-board computer is ready with ROS setup, download Mavros package from *Mavros*
* Connect pixhawk Px4 IMU with on-board computer using USB port connection.
* Follow instructions provided with package to build and launch mavros node.

## Remote Computer Setup

1. Download *AGV Code build*
2. Create a workspace and copy all the files present in the src folder to the newly created workspace.
3. Replace the .yaml and .pgm file with the your file (as done in preprocessing).
4. Replace machine learning model file with the one trained above.
5. Do catkin\_make.
6. Run roslaunch astar avg.launch.
7. Set the intial position and the goal position in rviz.
8. AGV should start moving to the goal position.

## Setup Validation

1. Once all the nodes are up and running. Check if all required topics are being published using following command:-

rostopic list

All topics listed in Topics and Data Types should be present.

1. Run following command to generate tf tree. It should match one referred in TF *Tree*

rosrun rqt\_tf\_view rqt\_tf\_view

1. After running all the nodes, rviz should display current pose of AGV .

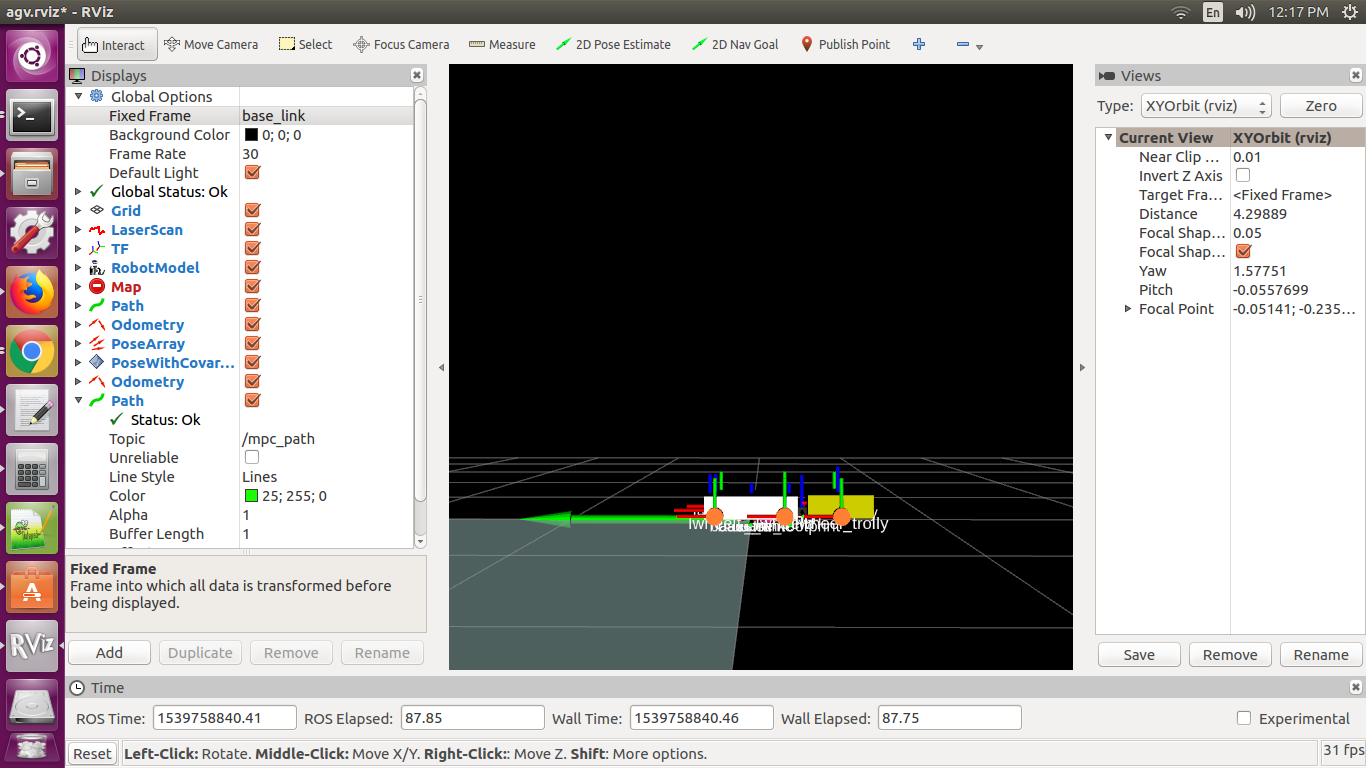


Figure 17Bottom view of AGV bot in rviz

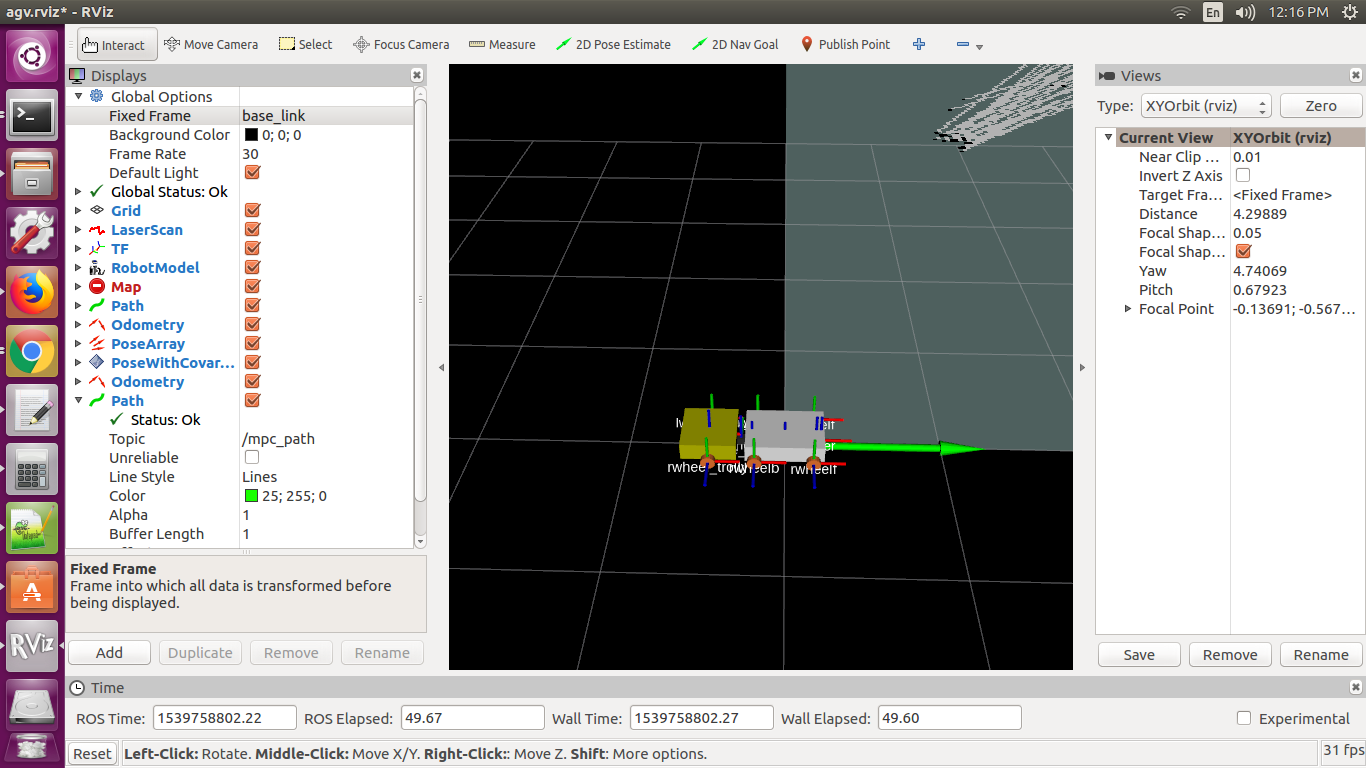


Figure 18Side view of AGV bot in rviz

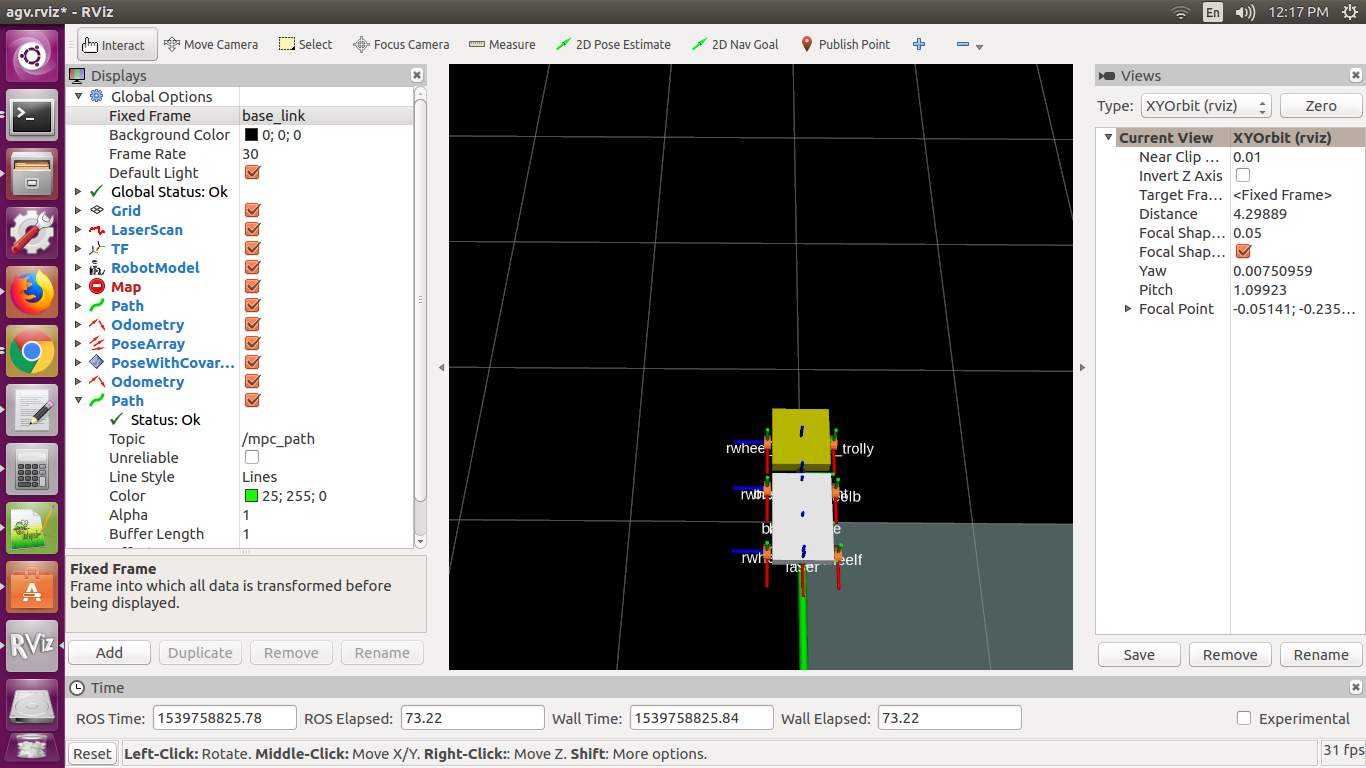


Figure 19Top view of AGV bot in rviz

1. After selecting a goal node via rviz, AGV should start moving towards that node.

# Appendix

## Challenges & solutions

Challenge 1: Wheel Odometry.

Solutions:

* Check if there is any wheel with slippage or lesser friction with floor.
* Check if any of the wheel with wheel encoders have more friction in gears than others.
* Make sure all motors are of same rating.
* Try switching wheel encoders from front to rear wheels or vice versa.
* Try adding wheel odometery to all four wheels.
* Make sure wheel ticks per second, distance between wheels is properly configured.

Challenges 2: Path Planning

* While converting Occupancy grid from /map\_server to a Numpy array of shape equivalent to that of Rviz map, take a transpose after reshaping.
* While generating the trajectory after searching, make sure the functionality is ended after the trajectory is within a specific radius of goal point.
* Make sure that the start and goal x, y values are converted to ‘int’ for grid based search

Challenges 3: Wi-fi Localization

* Try adding new wifi points
* Try relocating the wifi points.
* Try using other ML algorithms inorder to improve accuracy.

Challenges 4: Sensor Fusion

* Try switching between UKF & EKF based on outcomes
* Try configuring filter parameters in Robot Localization Package.
* Try using different combinations of sensors to check for minimal errors.
* Try calibrating sensors.
* Reconfigure sensor co-variance matrices.

## TF Tree

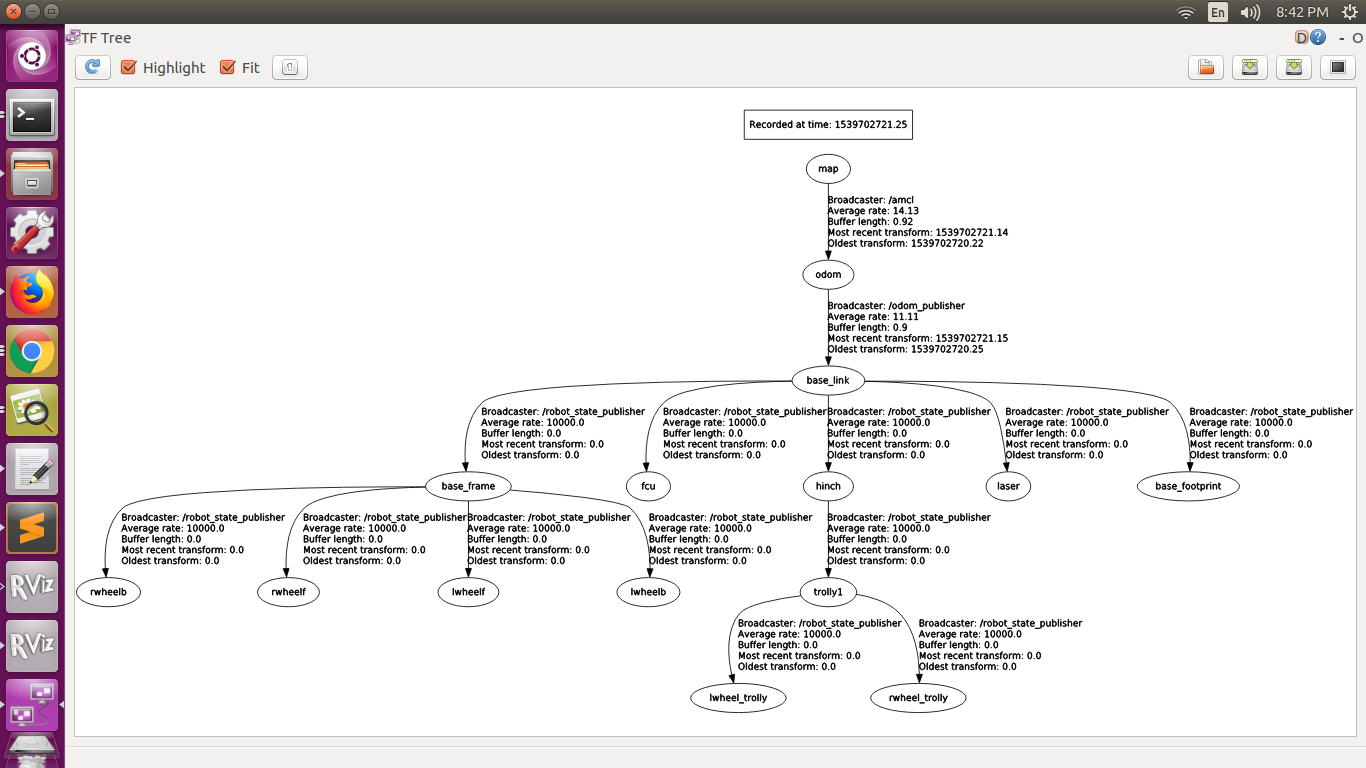
Detailed TF tree can be found at ***Tf Tree***

Figure 20TF Tree

## Topics and Data Types

|  |  |
| --- | --- |
| Topic | Message DataType |
| /odom | nav\_msgs/Odometry |
| /mavros/imu/data | nav\_msgs/Odometry |
| /odometry/filtered | nav\_msgs/Odometry |
| /accel/filtered | geometry\_msgs/AccelWithCovarianceStamped |
| /amcl\_pose | geometry\_msgs/AccelWithCovarianceStamped |
| /path | nav\_msgs/Path |
| /move\_base\_simple/goal | geometry\_msgs/PoseStamped |
| /wheel\_veocity | geometry\_msgs/Quaternion |
| /map | nav\_msgs/OccupancyGrid |
| /scan | sensor\_msgs/LaserScan |
| /lwheel\_ticks | std\_msgs/Int32 |
| /rwheel\_ticks | std\_msgs/Int32 |
| /zone | geometry\_msgs/Point |

## Artifacts

|  |  |
| --- | --- |
| Wi-Fi data collection script |  |
| Hector Slam | http://wiki.ros.org/hector\_slam |
| Raspberry Pi code build |  |
| Arduino Controller Code build |  |
| Mavros | http://wiki.ros.org/mavros |
| AGV Code build |  |
| iPython notebook for training ML model |  |
| Tf Tree |  |
| Script to setup ROS on raspbian |  |

## Future Enhancements

## References

|  |
| --- |
| <https://www.omicsonline.org/open-access/dynamic-modelling-of-differentialdrive-mobile-robots-using-lagrange-and-newtoneuler-methodologies-a-unified-framework-2168-9695.1000107.pdf> |
| <http://planning.cs.uiuc.edu/node661.html> |
| <https://www.coursera.org/lecture/mobile-robot/differential-drive-robots-GnbnD> |
|  |
|  |
|  |
|  |
|  |

# Index

A

**AGV · 3, 5, 7, 10, 11, 12, 13, 14, 17, 18, 19, 20, 22, 23, 24, 26, 27**

**amcl · 13, 19, 28**

**Arduino · 3, 7, 12, 14, 26, 28**

**Autonomous · 6, 17**

C

**controller · 3, 7, 12, 14, 23, 26**

D

**diff\_drive · 12**

E

**encoders · 3, 11, 18**

H

**Hector · 15, 18, 28**

**hector\_slam · 12, 28**

I

**IMU · 3, 9, 12, 17, 18, 26**

L

**localization · 5, 6, 10, 12, 17, 18**

**Localization · 6, 17, 18**

M

**Machine learning · 3, 17**

**make\_plan · 13**

**map\_server · 12**

**mapping · 15**

**mavros · 12, 26, 28**

**Motion Control · 6, 22**

**MPC · 3, 12, 14, 22, 23, 24, 25**

**mpc\_control · 13**

N

**navigate · *See***

**navigation · 2, 5, 6, 22**

O

**object detections · 5**

**odometry · 12, 15, 18, 28**

**Odometry · 3, 12, 13, 18, 28**

P

**particle filter · 13, 17, 18**

**path planning · 5, 6**

**Path planning · 6**

**Path Planning · 6, 21**

**PID · 13, 14**

**Pixhawk · 9**

**POC · 5, 17, 18**

R

**raspberry Pi · 11, 12**

**Raspberry Pi · 3, 9, 17, 26, 28**

**ROS · 11, 12, 13, 18, 26**

**ros\_arduino\_bride · 12**

**RP lidar · 11**

**RP Lidar · 3, 10**

**RPLIDAR · 10**

**RP LiDAR · 15**

**rviz · 13, 27**

S

**scan · 10, 11, 12, 13, 15, 18, 28**

**sensors · 2, 5, 6, 9, 15, 18**

**SLAM · 15, 18**

**Software · 3, 11**

V

**vehicle · 5, 6, 7, 9, 10, 13, 20, 23**

**Vehicle · 3, 19, 20, 22, 24**

W

**Wheel Encoders · 3, 9, 18, 26**

**Wi-Fi · 3, 9, 10, 11, 17, 18, 26**

**Wi-Fi Modules · 3, 10**

1. <https://en.wikipedia.org/wiki/Extended_Kalman_filter> [↑](#footnote-ref-1)
2. EKF: Robot localization package provides EKF and UKF mode of fusion. More details on this can be found at http://wiki.ros.org/robot\_localization [↑](#footnote-ref-2)
3. http://wiki.ros.org/amcl [↑](#footnote-ref-3)