

Robotics Project

Part II

Navigation & Localization

By

*Chalikonda Prabhu Kumar*  
*Jampy Florian*  
*Sai Krishna Pathi*

Masters in Computer Vision, University of  
Burgundy

## Introduction:

The main aim of this part is Navigation and Localization. In this project, we navigate the robot in an unknown and known map by localizing the robot and then avoid the obstacles and reach the goal.

There were 4 goals proposed in this part. All these goals were completed using path planners and some of them required some more features.

The following were the goals proposed.

Goal Descriptions:

Goal 1: Navigation in a Blank Map

Goal 2: Navigation in a Map

Goal 3: Navigation in a Map with Obstacle Avoidance

Goal 4: Target Detection

## Procedure:

### Goal 1: Navigation in a blank map

In this part, the robot is made to move in an unknown map or a blank map. The robot is moved from Point A to Point B. This task is executed by using various path planners like *amcl*, *move\_base*, *robot\_pose\_ekf*. The pose of the robot should be the same as that of the starting pose. We executed the task with and without obstacles i.e. we use the blank map and the robot starts moving from Point A to B by avoiding the obstacles which are placed in between the robot and the goal and the second time we run the same thing and this time we don't place any obstacles so it goes plainly towards the goal.

There is a video provided in which the robot moves from Position A to B in blank map.

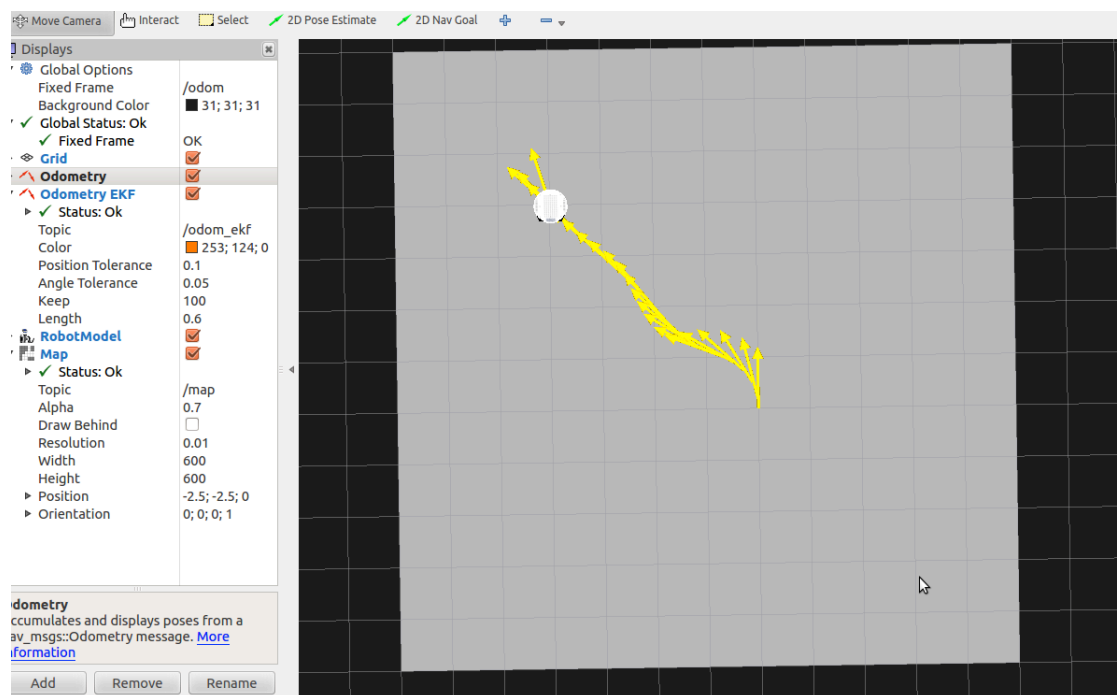
The link for the video is provided here.

<http://www.youtube.com/watch?v=oF43oZPVmiA>

## AMCL:

AMCL 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. In this, implementation is done on an empty map for doing AMCL.

The below figure shows the Rviz result of amcl in a Blank map.

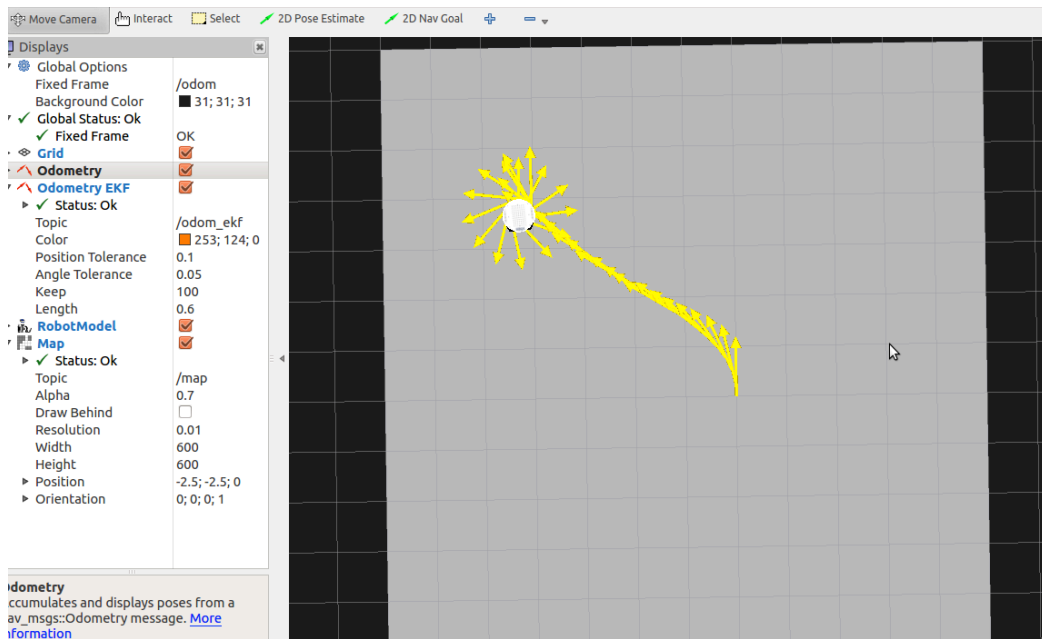


FIGURE

## Move Base:

The move\_base package provides an implementation of an action 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. The move\_base node also maintains two cost maps, one for the global planner, and one for a local planner that are used to accomplish navigation tasks. In this, the main theme is, first it will create a global path to reach goal and the robot follows this path and when it finds an obstacle in the path while moving towards the goal it uses local path planner to avoid obstacle and reach the goal.

The below figure shows the Rviz result of move\_base in a Blank map.

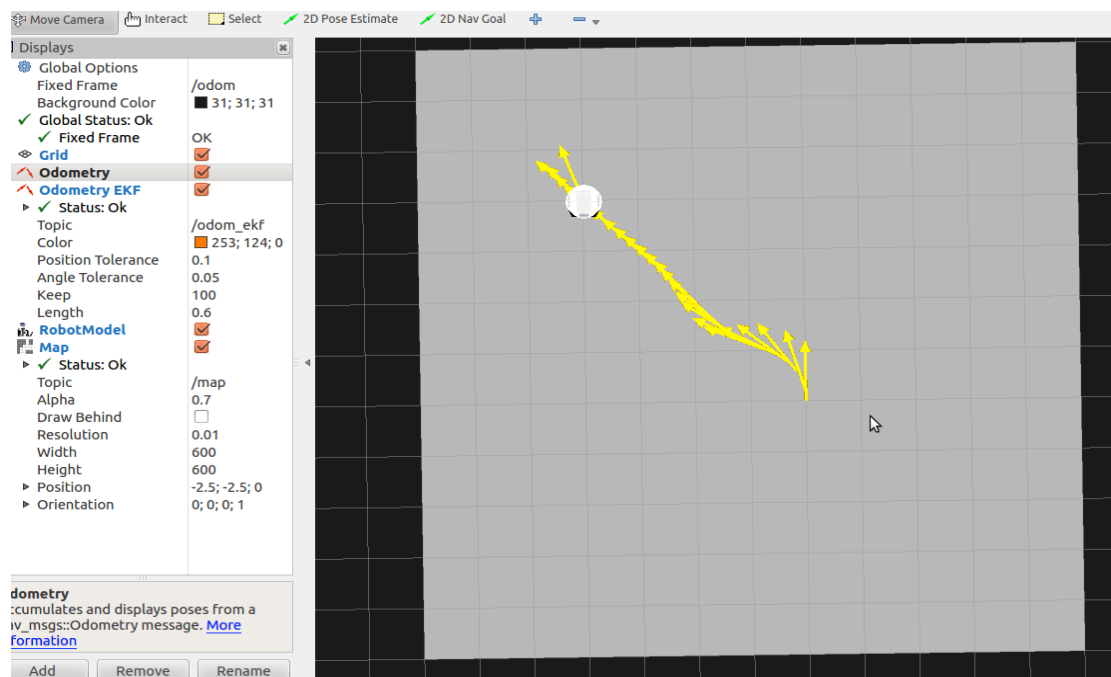


FIGURE

## Robot Pose EKF:

The Robot Pose EKF package is used to estimate the 3D pose of a robot, based on (partial) pose measurements coming from different sources. It uses an extended Kalman filter with a 6D model (3D position and 3D orientation) to combine measurements from wheel odometry, IMU sensor and visual odometry. The basic idea is to offer loosely coupled integration with different sensors, where sensor signals are received as ROS messages.

The below figure shows the Rviz result of robot\_pose\_ekf in a Blank map.



FIGURE

The difference between AMCL & EKF is AMCL is to localize (Laser Scan) the TurtleBot in 2D known map while EKF use 3D pose information coming from different sources. The EKF works by transforming the nonlinear models at each time step into linearized systems.

## Goal 2: Navigation in a Map

In this part, the robot is moved or driven around the arena using *keyboard\_teleop* or *Joystick* and the map is build-using *gmapping*. The *slam\_gmapping* node takes in *sensor\_msgs/LaserScan* messages and builds a map. Then the map is saved to file. There are some packages like exploration, interaction marker available in ros to build the map. Exploration package is used to move in the arena randomly to cover whole arena and build the make. In the case of Interaction marker package, rviz is used to move the robot by tapping the rviz in which direction does turtlebot should move etc.

The map of the Arena is shown in the below Figure

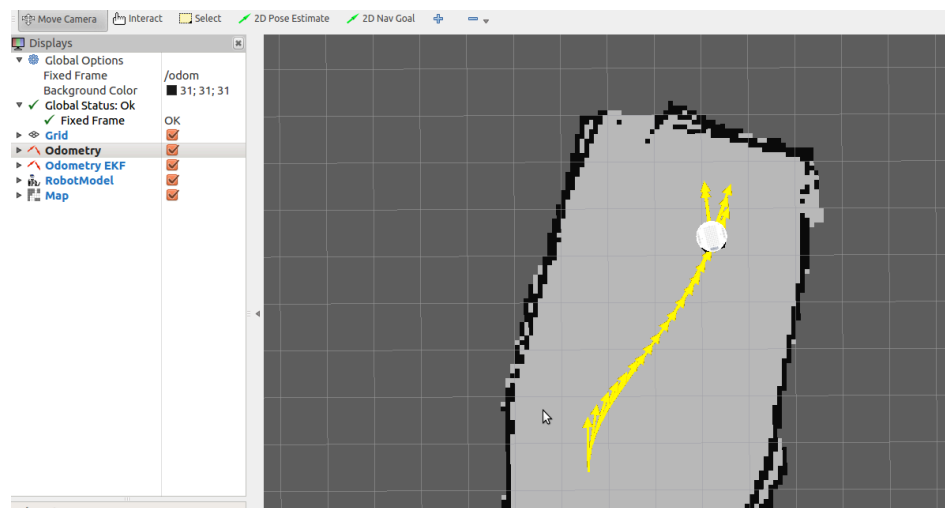


FIGURE

Then using this map, the robot is moved from Point A to Point B. The pose of the robot should be the same as that of the starting pose. Here we don't consider the obstacles.

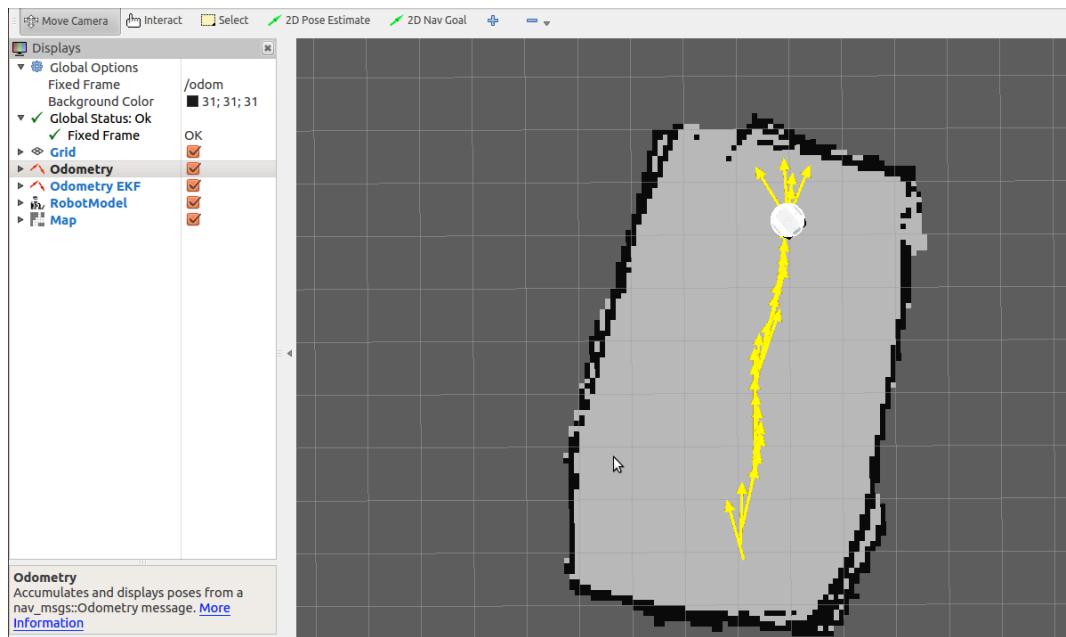
We also tried with the *move\_base*. The results of the rviz of these planners are provided below.

The below figure shows the Rviz result of *amcl* in a map without obstacle.



## FIGURE

The below figure shows the Rviz result of move\_base in a map without obstacle.



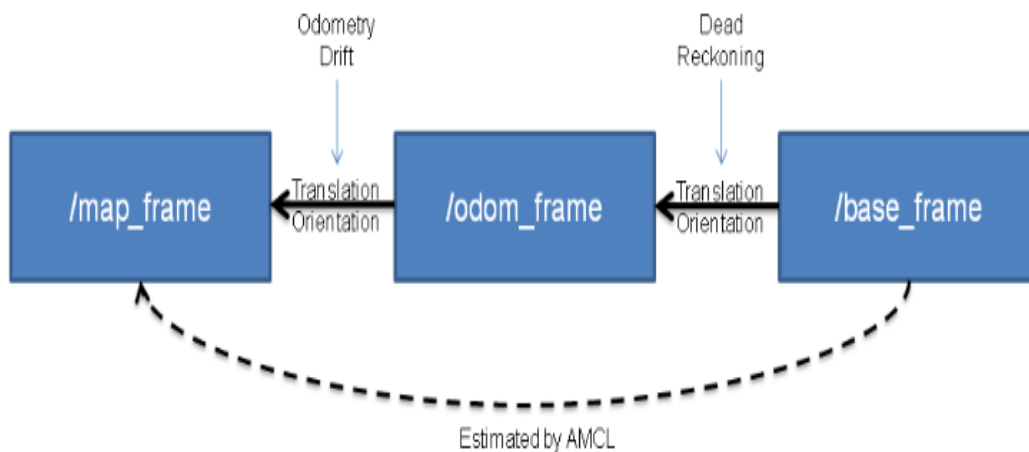
## FIGURE

To move From Point A to B we use the path planner *amcl*, which uses Global and Local Path Planners to plan the path. The Global path planner will plan a global path to reach the goal. The Local Planner will attempt to follow this global path. When there is an obstacle placed in the way to reach goal local planner will be used to avoid the obstacles. It will plan the path locally till it avoid obstacles and follow the global path, which is done initially. These planners are primarily used for Visualization.

Normally for Localization */base\_frame* is used which is by default "*base\_link*". But when we have a map we will be using */map\_frame* which uses "*map*" which is used for Localization.

Before using map, we publish the Position or Translation and Orientation that is considered as the distance, which is used by the robot to move. This is published using the "*base\_link*" which is used for odometry. But when we use the map, we publish the coordinates of Position or Translation and Orientation, which is the goal for the robot. This is published using the "*map*" which is used for odometry. From the below figure we can see how the map is used by the *amcl* for Localization.

We can use other navigation path planner like robot pose ekf, move\_base also to move in the arena.



**FIGURE**

There is a video provided in which the robot moves from Point A to B in a known map without any obstacles.

The link for the video is provided here.

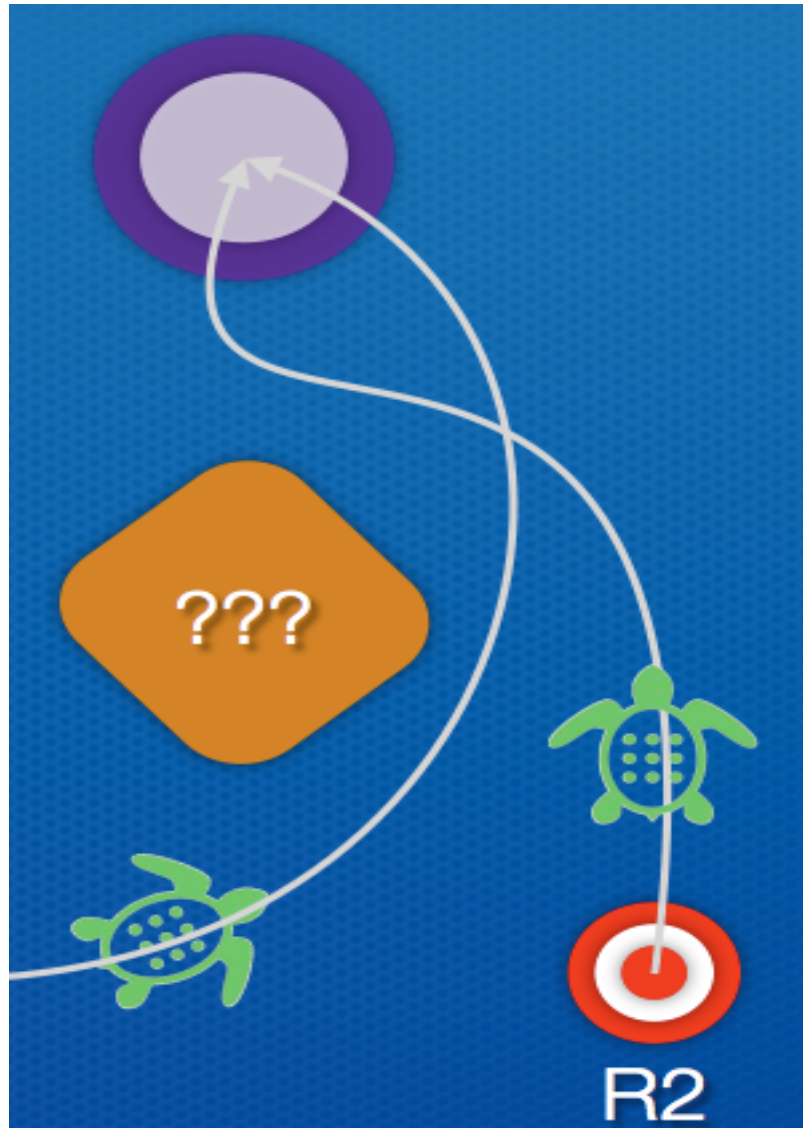
[http://www.youtube.com/watch?v=Rz\\_vUTZ2frY](http://www.youtube.com/watch?v=Rz_vUTZ2frY)

### Goal 3: Navigation in a Map with Obstacle Avoidance

In this part, the robot is moved from Point A to B. There are some obstacles in between the Starting Point and Goal Point. The robot moves towards the goal avoiding these obstacles and reaches the Goal Point.

Initially the global planner plans a path through the free space (avoiding the obstacles) in the map to the goal. Then the local planner starts to move towards the goal using this path. And if there are no obstacles placed during the movement then the robot moves in this path and achieves the goal. If there are any obstacles placed, then the Local Planner plans a path avoiding the obstacles and the robots moves in this new path planned by the local planner.

There is a video provided below, where the robot moves from Point A to B and there is an obstacle in the Arena but it is not exactly in the path between both the points A and B. The situation is like in the below figure and our robot is like the robot in R2 position.



**FIGURE**

The link for the video is provided here.

<http://www.youtube.com/watch?v=Nq9YvUaA9Kc>

<http://www.youtube.com/watch?v=gSyl-qz5gtM>

<http://www.youtube.com/watch?v=oYFRQYNcs5c&feature>

#### **Goal 4: Target Detection**

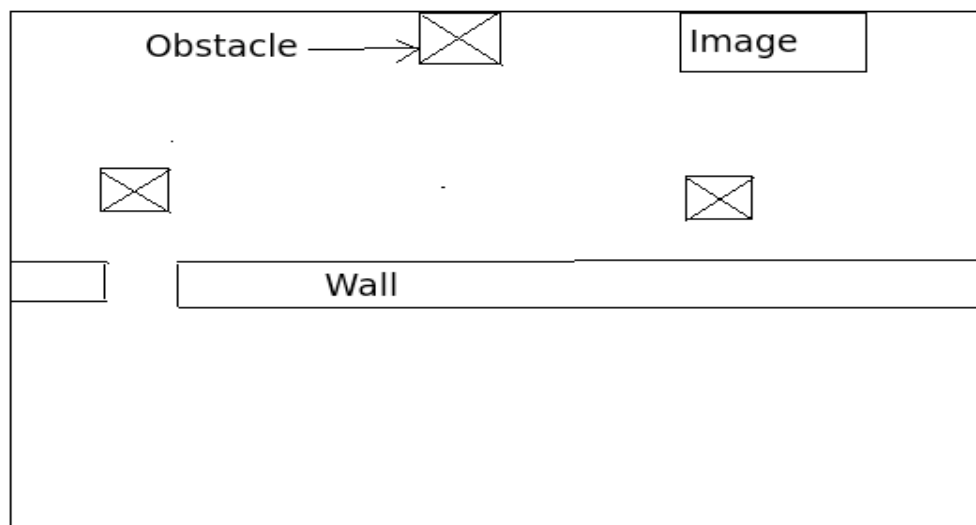
In this part, the robot is supposed move in the map and detect the QR Code (Picture/Image) using the position and orientation coordinates of the QR Code. The robot should move towards the picture and stand in front of image this is the goal.

This goal is divided in two parts. First detect the image by moving around the arena and then when the image is found stop moving around and the second



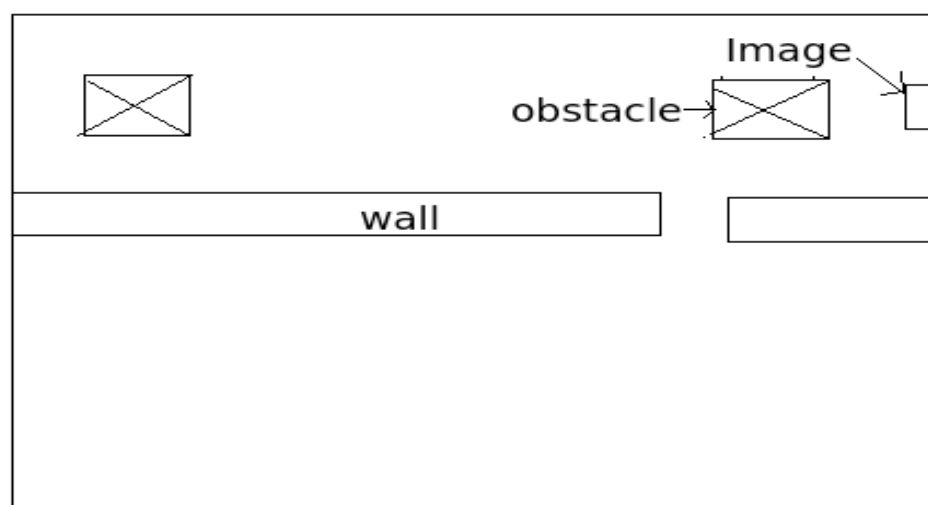
part is moving towards the image and stop in front of it with certain distance between turtlebot and target image. For the first part we have developed a code to move around the arena in a specified path to detect the code as the image may be placed any where in the arena. For the second part we have developed another code to move in front of the image using the position and orientation coordinates which we obtained when the image is found using the library.

For the first part we assumed different possibilities or chances of the placing the image in the arena. For all these options we assumed different paths, which can be suitable for achieving the goal. The below figure 1 shows 1<sup>st</sup> option which shows the placement of the image in the arena.



**FIGURE**

If this is the situation then the detection and moving towards the image becomes easy as the image is openly placed. In this situation, the robot is supposed to move to the center of the search area and rotate 360 degrees to find the image and then move towards it.



**FIGURE**

In this situation the robot has to move in the search area to find the image because the option mentioned above (rotate 360 degrees) will not be useful in this case as the obstacle may be in front of it and it may block the vision of the robot. So for this case, we came up with another solution i.e. move in the search area to detect the image. For this, we set a path for the robot to move so that it can cover the whole search area with minimum time. The specified path is a square. By moving in a square around the search area the robot can detect the image on the wall or on the obstacle, which is the opposite side to the wall. The robot moves with the wall so that the image placed anywhere in the search area on the wall can be identified.

For this Visual Servoing Part, we tried with different libraries and packages. Finally, we found one library and one package, which are working for this part. They are the *VISP* and the package is *ar\_recog*. There are others too but they have some impediment on real time implementation.

### **VISP:**

VISP is a complete cross-platform library mainly used for Visual Tracking and Visual Servoing. We first tried with *Visp* and it is working fine and the detection and tracking are robust.

The tracking is implemented using the *visp\_auto\_tracker*. This node publishes the topics, which are helpful in the tracking. The topics helpful in our case are *object\_position*, which gives the 3D pose of the model and *klt\_points\_positions*, which gives the Position and id of the key points used for monitoring purpose.

Subscribing to the above-mentioned topics we were getting the position and orientation values of the image in reference to the world. So using this information we can easily send the robot to the image.

But the problem we faced was it couldn't detect the target if the distance between turtlebot and target is greater than 20cm. We tried to increase the distance of detection but it didn't work for us.

So we tried to build own code using OpenCV and SURF but here also the problem is same. So we tried with other packages.

### **CMVISION:**

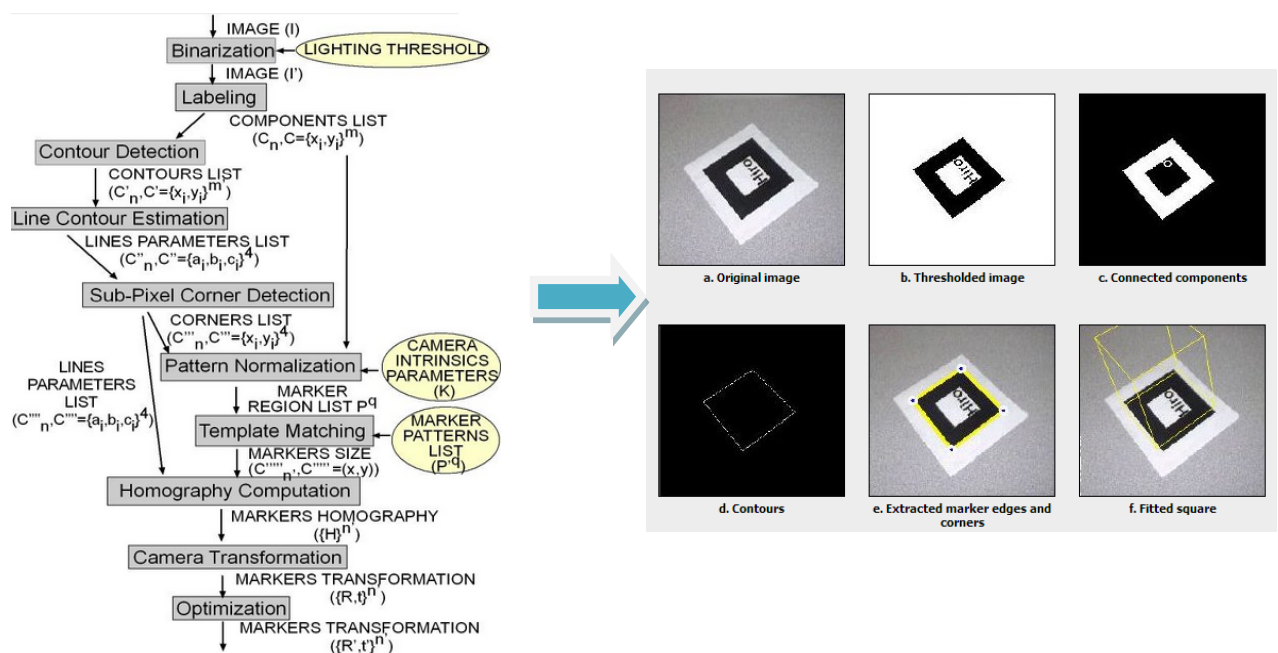
Then we plan to use *cmvision* package in ros by using *blobfinder*. The idea of the blob finder is detecting the pixels, which are having mostly same value. Even though it is able to detect the blobs, which we specified in the image from, long distance but false positives (unwanted blobs) are more than actual one. *cmvision* package is successfully but unwanted blobs are more this is happen because of light changes in the environment. Illumination factors dominate the package while detecting the actual blobs, small light changes also makes to increase false positives rate. Apart from this the useful information is not published like distance from target etc. So we came to conclusion that this package is not much useful for accomplishing the target.

## AR\_RECOC:

Then we came up with the solution to use *ar\_recog* package in ros. There are some advantages of this package especially in recognition of QR code. By using this we can detect the image from long distance this is the great advantage of this package. *ar\_recog* publishes two messages tags contains the Tags message, the information about the found tags. The tags information consists of distance, position of image, corners of the image etc., which is really helpful for robust detection.

### How AR\_RECOC works?

Based on the *ARToolkit* augmented reality library, *ar\_recog* recognizes augmented reality tags in an image. *ar\_recog* publishes various information about recognized tags, such as its corners in image space and relative 6DOF pose in camera space.



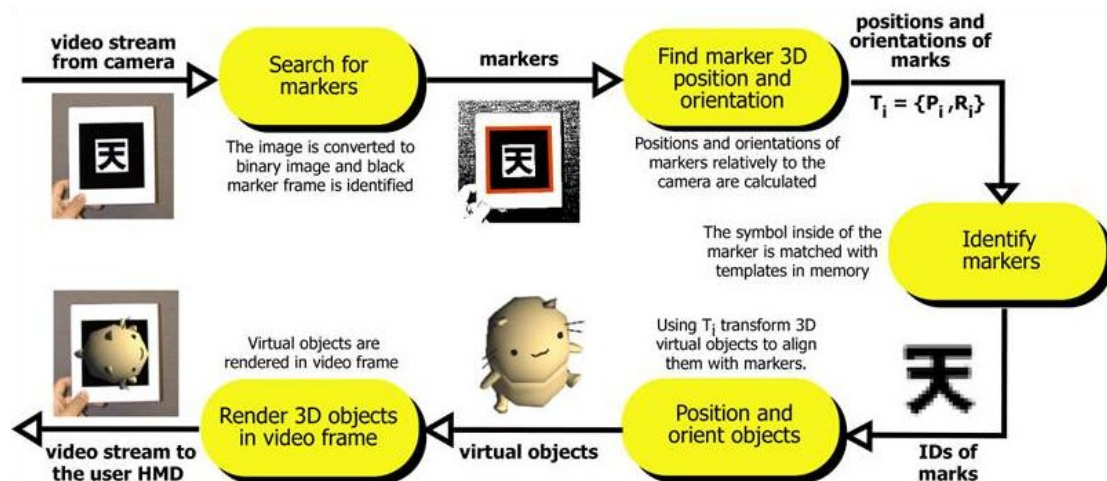
FIGURE

ARToolKit applications allow virtual imagery to be superimposed over live video of the real world. Although this appears magical it is not. The secret is in the black squares used as tracking markers. The ARToolKit tracking works as follows:

1. The camera captures video of the real world and sends it to the computer.
2. Software on the computer searches through each video frame for any square shapes.
3. If a square is found, the software uses some mathematics to calculate the position of the camera relative to the black square.

4. Once the position of the camera is known a computer graphics model is drawn from that same position.
5. This model is drawn on top of the video of the real world and so appears stuck on the square marker.
6. The final output is shown back in the handheld display, so when the user looks through the display they see graphics overlaid on the real world.

ARToolKit is able to perform this camera tracking in real time, ensuring that the virtual objects always appear overlaid on the tracking markers.



FIGURE

With this package we are able to detect the QR code robust. The markers are stored in the package with the help of online website in 16x16 pattern (<http://flash.tarotaro.org/blog/2009/07/12/mgo2/>). From this package, we are subscribing to the "Tags" topic.

From this topic, we are getting the information of the *distance* between the QRCode and Robot and the angle i.e. *yRot* with this information we are able to keep the robot in the same angle of that of Image.

Here we build a state machine code for the coordination purpose between all the processes or nodes.

The state machine is here to schedule each task and avoid any interference between them. Each State corresponds to an action, Wait a command or the actual performed task. It's also used to centralize the information each node subscribed to the State machine and the State machine subscribed to each useful topic.

There is a video provided in which the robot detects the QRCode or Image and then moves towards it and stops in front of it.

The link for the video is provided here.

[http://www.youtube.com/watch?v=tvpz037xU\\_E](http://www.youtube.com/watch?v=tvpz037xU_E)

## Conclusion:

In this project, we learnt how to navigate and localize the robot using different Path Planners. And in the Visual Servoing part, we learnt about the detection and tracking.

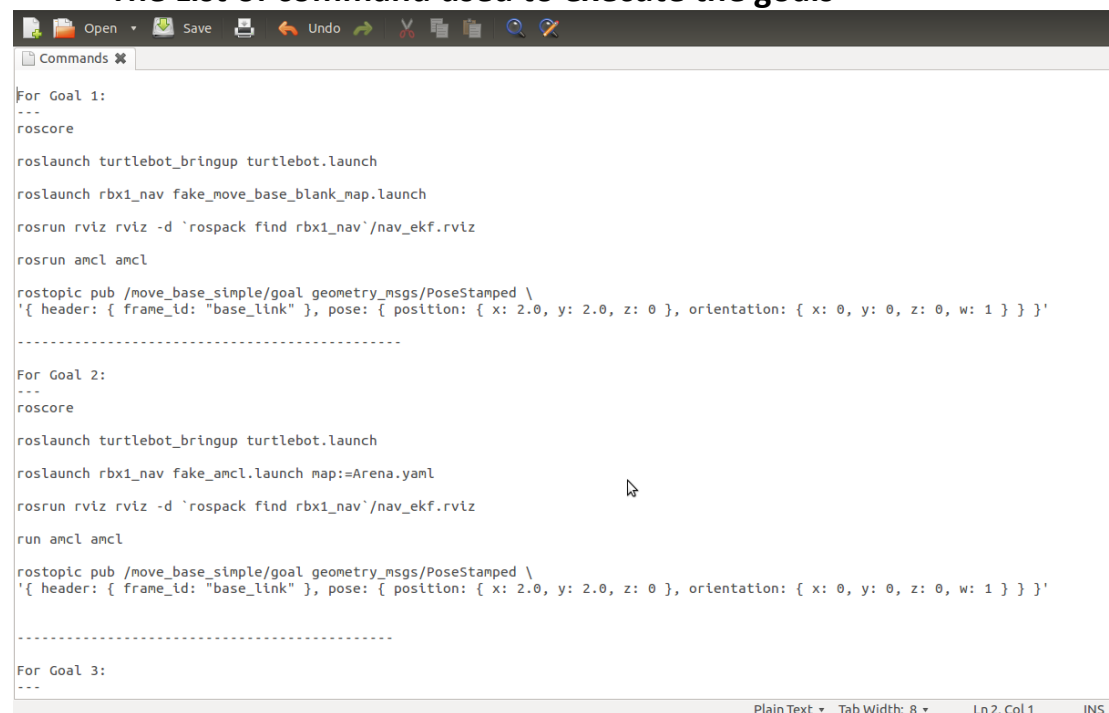
To summarize, move\_base and amcl are used for getting optimal path. Besides this we use amcl because particle filter is used to track the pose of the robot against the known map. For target recognition, ar\_recog is more robust to achieve the task with necessary information. This is also used to detect multiple targets in the map. Visp also is robust but the distance is the limitation.

## References:

1. R. Patrick Gabriel. ROS by examples volume 1.
2. <http://wiki.ros.org/gmapping>
3. <http://wiki.ros.org/visp>
4. <http://wiki.ros.org/cmvision>
5. [http://wiki.ros.org/ar\\_recog](http://wiki.ros.org/ar_recog)

## APPENDIX

### The List of command used to execute the goals



```
Commands ✖
For Goal 1:
---
roscore

roslaunch turtlebot_bringup turtlebot.launch
roslaunch rbx1_nav fake_move_base_blank_map.launch
roslaunch rviz rviz -d 'rospack find rbx1_nav'/nav_ekf.rviz
roslaunch amcl amcl

rostopic pub /move_base_simple/goal geometry_msgs/PoseStamped \
'{"header": {"frame_id": "base_link"}, pose: {"position": {"x": 2.0, "y": 2.0, "z": 0}, orientation: {"x": 0, "y": 0, "z": 0, "w": 1}}}'

-----

For Goal 2:
---
roscore

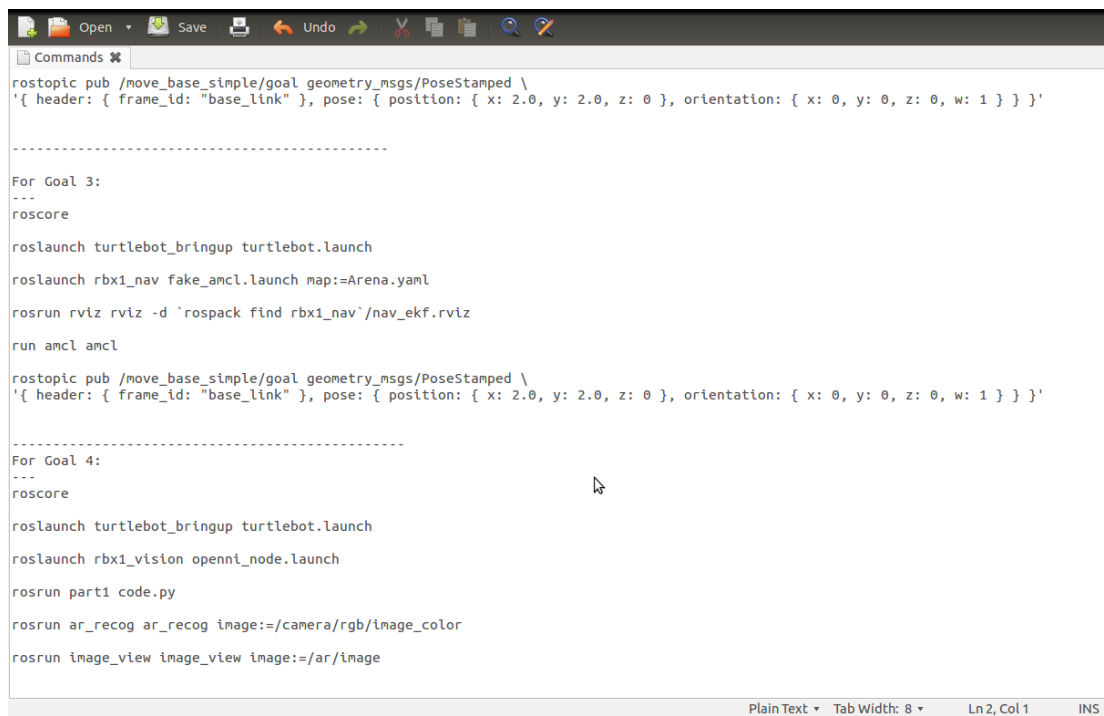
roslaunch turtlebot_bringup turtlebot.launch
roslaunch rbx1_nav fake_amcl.launch map:=Arena.yaml
roslaunch rviz rviz -d 'rospack find rbx1_nav'/nav_ekf.rviz
run amcl amcl

rostopic pub /move_base_simple/goal geometry_msgs/PoseStamped \
'{"header": {"frame_id": "base_link"}, pose: {"position": {"x": 2.0, "y": 2.0, "z": 0}, orientation: {"x": 0, "y": 0, "z": 0, "w": 1}}}'

-----

For Goal 3:
---
```

Plain Text ▾ Tab Width: 8 ▾ Ln 2, Col 1 INS



```
Commands
rostopic pub /move_base_simple/goal geometry_msgs/PoseStamped \
'header: { frame_id: "base_link" }, pose: { position: { x: 2.0, y: 2.0, z: 0 }, orientation: { x: 0, y: 0, z: 0, w: 1 } }'
```

-----

For Goal 3:

```
---
roscore

roslaunch turtlebot_bringup turtlebot.launch
roslaunch rbx1_nav fake_amcl.launch map:=Arena.yaml
roslaunch rviz rviz -d 'rospack find rbx1_nav'/nav_ekf.rviz
run amcl amcl

rostopic pub /move_base_simple/goal geometry_msgs/PoseStamped \
'header: { frame_id: "base_link" }, pose: { position: { x: 2.0, y: 2.0, z: 0 }, orientation: { x: 0, y: 0, z: 0, w: 1 } }'
```

-----

For Goal 4:

```
---
roscore

roslaunch turtlebot_bringup turtlebot.launch
roslaunch rbx1_vision openni_node.launch
roslaunch part1 code.py
roslaunch ar_recog ar_recog image:=/camera/rgb/image_color
roslaunch image_view image_view image:=/ar/image
```

Plain Text ▾ Tab Width: 8 ▾ Ln 2, Col 1 INS