
UNIVERSITY OF BURGUNDY
MASTER IN COMPUTER VISION AND ROBOTICS

---

# ROBOTICS PROJECT
## Preliminary Survey

---

*Submitted To:*
Dr. Ralph SEULIN
Dr. Marc BLANCHON
Dr. David FOFI
Dr. Raphaël DUVERNE

*Submitted By :*
Mohamed Tawfik
François Legrand
Olivier Agbohoui
Vamshi Kodipaka

November 3, 2019

# Contents

# 1 Introduction

## 1.1 Context

This document is a technical survey about the implementation of our Robotic Project. The content will be focused on the implementation aspect of the tasks to perform, our strategies to construct them, and the overall organization. Here, we will also explain which programs and packages we will use, how we plan to make them work together, as well as explaining the motivations of our decisions.

The work will be performed on a second generation Turtlebot, and will be using a Microsoft Kinect V1. The main objectives of this project is to familiarize us with the different aspects and uses of the Robot Operating System (**ROS**) middle-ware, which include package creation, management, robot remote controlling, automated tasks and programming for embedded computer vision.

## 1.2 Organization

This technical survey will allow us to plan the organization of our project. It will consist into several tasks with gradually increasing difficulty. The four first ones being mandatory to cover all the previously given aspects of the course, the latter one as a challenge to complete. The objectives will allow us to further improve our knowledge and practice about ROS frame management and programming.

To do so, our technical survey will be divided into three parts, each one dealing with the different aspects and implementation ideas of a single task. The first part of this document will cover the map creation as well as the autonomous navigation of the Turtlebot while performing obstacle avoidance. We will describe our approach to perform Robot's displacement in an automated way.

In a second section, we will discuss about our first embedded computer vision objective to reach. It will depend on the success of the previously defined autonomous navigation part, as it will serve as a support for further development. For this task, we have chosen localization of AR tags as our framework.

From a similar concept, we will focus our third task on QR code detection and information extraction. The success of this task relies heavily on the quality of the implementation of the two previous ones.

Our fourth process which constitute the third section of this report, will be to parameter the content of the QR code to perform autonomous actions on the Turtlebot, from predefined instructions. In this part of the project, we will include the use of personal support to perform remote controlling (use of a micro-controller to transmit instructions and accesses).

In case our Gantt scheduling table is respected (**fig.1**), a final task will be implemented, based on QR code differentiation, assignation and multi-tasking. The time organization

will depend on a double flow, with the 4th process implemented independently from the rest of the planned workflow. This would lead us toward the **fig.2** representation.

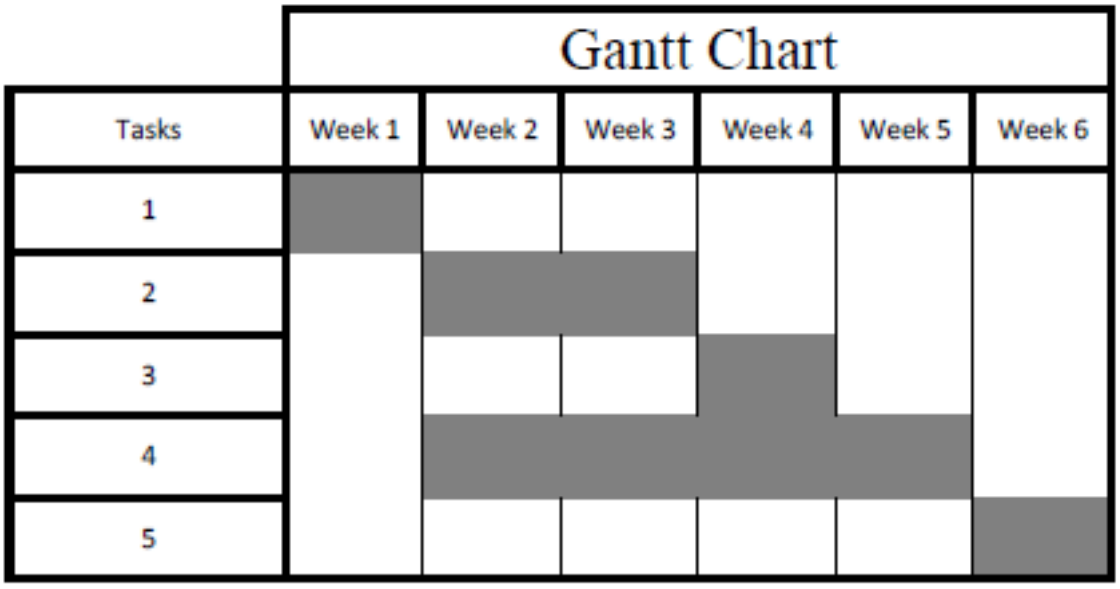


Figure 1: Gantt table determined for the Current Project

Our tasks can be sumarized as follow :

- 1- Create a map and do autonomous navigation and obstacle avoidance.
- 2- Localization of AR tags.
- 3- Detect the QR frame and obtain the embedded code.
- 4- Build the web server and activate the embedded WIFI module.
- 5- Activate more than one QR frame (if time is left)

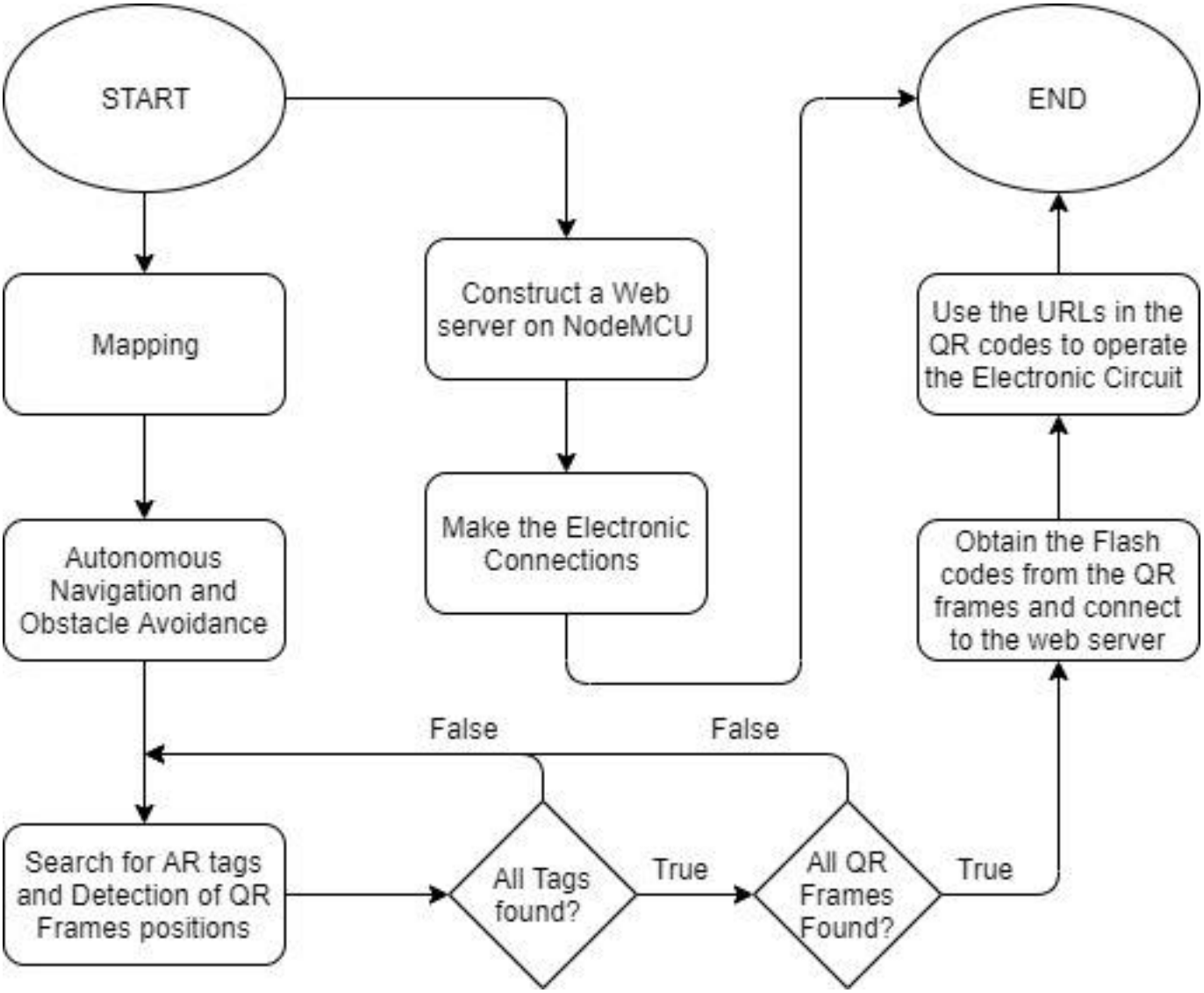


Figure 2: Ideal organization of our project

# 2 Autonomous navigation and localization of the Turtlebot

## 2.1 Implementation process

The first objective of this project is centered on the development and the setting of a robust autonomous navigation and localization ability on the robot. The building of our package will depend on a given workflow, composed by the following steps :

- Global map creation

- Setting the map as reference
- Localize the robot using navigation stack
- set obstacle avoidance
- Navigate the Turtlebot

Our general idea is to first drive the Turtlebot manually through the room through the *turtlebot_bringup* package, while activating a second one called *gmapping*, which will suscribe to the LiDAR topics to build our global map.

Once the global map is built, we plan to use it as our reference frame for further processes. In particular, we want to use navigation stack to localize the Turtlebot, by using the *AMCL* package, which will enable the creation of a local costmap then compare it to the global map, so the robot can be localized. This is an important step of the process as the costmap will allow us to perform obstacle avoidance by surrounding objects with a given radius of infinite cost (**fig3**).

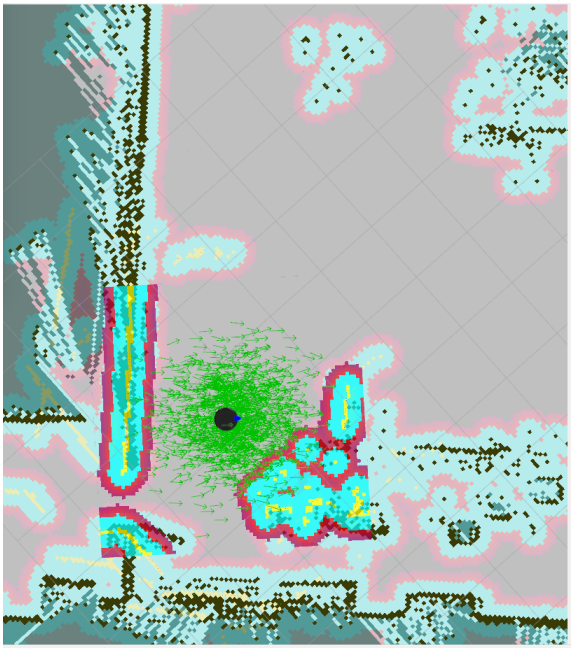

Figure 3: Illustration of the amcl package actions

To navigate autonomously the Turtlebot, we intend to use the *move_base* package, along AMCL. When running on move_base, the Turtlebot will move according to the surrounding local costmap (**fig.4**)

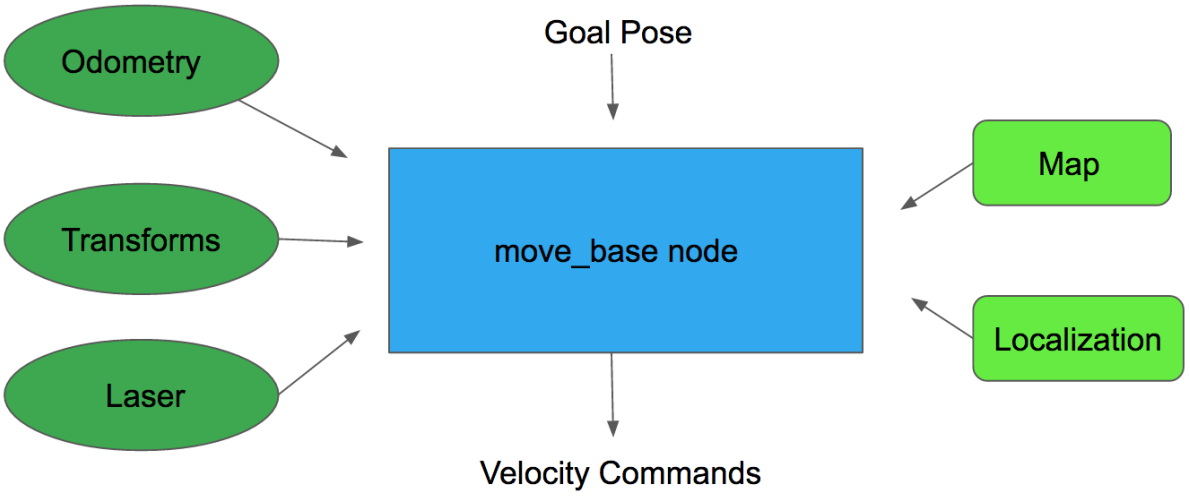


Figure 4: Working principle of the *move_base* package

Those packages will be framed and called by a provided one which will also allows us to export and rely on LiDAR and Kinnect data (*turtlebot_vibot* metapackage).

## 2.2 Package choice motivations

As given previously, we intend to mainly rely on the following package list for autonomous navigation and obstacle avoidance :

- *turtlebot_vibot* (and its sub-packages)
- *gmapping*
- *AMCL*
- *move_base*

The first package mentioned in the list contains demonstration files which can be useful for us to set-up the task, in particular by duplicating and modifying them to perform the wanted actions.

*Gmapping* package has been chosen here since it is already present in the available ROS set-up, has an acceptable stability, and is called as a dependency of the base meta-package. Other alternatives include mapping through *hector_slam* (less stable) and *rtabmap_ros* (more complex, different system).

*AMCL* package is our next choice as it is the most known one for local map creation

and stacking to run along move_base navigation package. Its function can be summarized through fig.5.

*Move_base* is used here to perform autonomous navigation, as it is the most complete package of its kind (for example *cb_base_navigation* is simpler but less flexible).

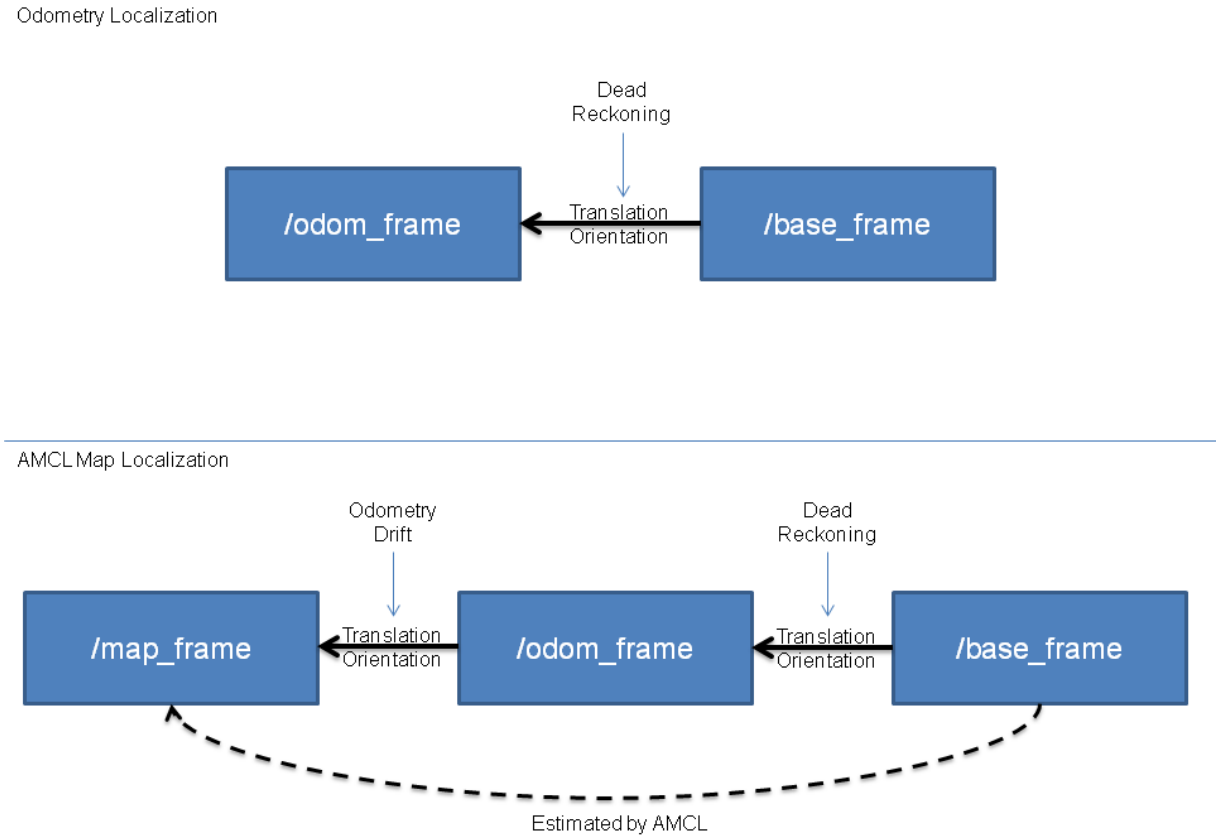


Figure 5: Schematized function of the *AMCL* package

# 3 AR tags detection & tracking

## 3.1 Real-time

To perform real time tracking, we will teleoperate the Turtlebot from the workstation. We intend to conduct the following process to achieve real time so to use the AR Tags we have *ar_track_alvar* package which allows us to generate, identify, track the tags, automatically calculate the spatial relationships between the markers.

*ar_track_alvar* has four main functionalities:

*G*enerating AR tags of varying size, resolution, and data/ID encoding. Identifying and tracking the pose of individual AR tags, optionally integrating kinect depth data (when a kinect is available) for better pose estimates. Identifying and tracking the pose of "bundles" consisting of multiple tags. This allows for more stable pose estimates, robustness to occlusions, and tracking of multi-sided objects. Using camera images to automatically calculate spatial relationships between tags in a bundle, so that the user does not have to manually measure and enter tag locations in an XML file to use the bundle functionality.

*A*lvar is significantly newer and more advanced than the ARToolkit, which has been the basis for several other ROS AR tag packages. Alvar features adaptive thresholding to handle a variety of lighting conditions, optical flow based tracking for more stable pose estimation, and an improved tag identification method that does not significantly slow down as the number of tags increases.

*Installation of AR Tags:*

sudo apt-get install ros-indigo-ar-track-alvar

### 3.2 Generating AR tags

*T*wo pdf files are in the markers directory containing tags 0-8 and 9-17, respectively. The markers are 4.5 cm (although when printed and measured, came out to 4.4 cm for me). If you want to generate your own markers with different ID numbers, border widths, or sizes, run: *r*osrun *ar_track_alvar* createMarker. and instructions will appear describing the various options.We download the aforementioned two marker files here:

AR Tags

*Detecting individual tags:* The first use case for this package is to identify and track the poses of (possibly) multiple AR tags that are each considered individually. The node individualMarkers takes the following command line arguments:

- *marker_size(double) – The width in cms. of one side of the black square marker border*
- *max_new_marker_error (double) – A threshold determining when new markers can be detected under uncertainty*
- *max_track_error (double) – A threshold determining how much tracking error can be observed before an tag is considered to have disappeared*
- *camera_image (string) – The name of the topic that provides camera frames for detecting the AR tags. This can be mono or color, but should be an Unrectified image, since rectification takes place in this package*
- *camera_info (string) – The name of the topic that provides the camera calibration parameters so that the image can be rectified*
- *output_frame (string) – The name of the frame that the published Cartesian locations*

*of the AR tags will be relative to individualMarkers assumes that a kinect being used as the camera, so that depth data can be integrated for better pose estimates. If you are not using a kinect or do not desire to use depth data improvements, use individualMarkersNoKinect instead.*

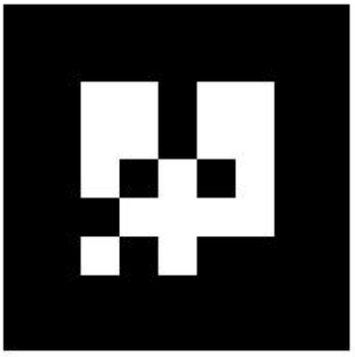

Figure 6: Same image of the AR marker

*W*e have to develop nodes to do the different tasks and communication between nodes is achieved by flags.

### 3.3 Developing the following nodes

*A*. 'gotoPoint' Node(python file):

- It has to publish the intial pose of turtlebot.
- Then it publishes the centre position of the map.
- After it reaches the centre of the map, it has to publish a flag on 'centerReached_flag'

*B*. 'Localization' Node(python file): *T*he task for detecting two markers and calculating the pose of markers with respect to map and publishing flag on 'goToOn_flag' topic.

- This node has to subscribe to 'amcl_pose' topic to get the pose of turtlebot with respect to map and 'ar_marker_pose' topic to get the pose of marker with respect to base link of turtlebot.
- When the turtlebot detects markers 0 and 1, the 't0_flag' and 't1_flag' is checked which is initially set to 'False'. If both flags are true a flag is published on 'goToOn_flag'.
- When a marker is detected, the pose of turtlebot w.r.t map is subscribed and the quaternion angles of rotation of frames of map and turtlebot is converted to rotation angle 'yaw' of turtlebot with respect to map according to given formula.

*C*. 'rotate' Node(python file):

- It has to rotate the turtlebot around its center of axis at center position of map for detecting the markers.
- When the 'centerReached_flag' is 'True', the turtlebot has to start rotating, it rotates 360 degrees and then it publishes a flag on 'rotate_flag'.

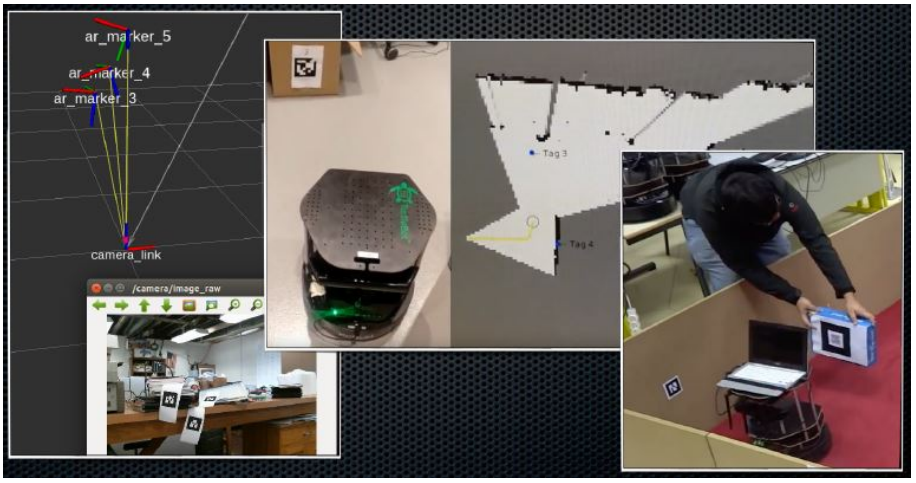


Figure 7: Example work for AR Tag Detection

*D.* 'goToTarget' Node(python file):

- When the two markers are detected, 'goToOn_flag' has to set to 'True' and
- after 'roate_flag' is set to 'True' after rotation of turtlebot, the turtlebot moves to the pose computed for marker 0, which is the location of pickup of the object.

*E.* 'goalStatus' Node(python file):

- After the turtlebot reaches the marker 0 location, we perform QR code detection.
- This task of handshake with the QR detection is done by this node. It publishes on topic 'nav_status'

*F.* 'goToTarget1' Node(python file):

- After QR code detected manually. They handshake a flag on topic 'arm_status'. This node has to subscribe to this topic and it publishes goal of marker 1 pose on 'move_base_simple/goal'.

*G.* 'goalStatus1' Node(python file):

- It has to subscribe to 'target1_published' flag, amcl pose and goal pose to check whether the turtlebot reached the goal or not.

  *T*he communication between nodes is achieved by using different flags. For node structure and tag detection flowchart; check Ref.No.4

*N*ecessary python libraries:

*move_base_msgs, actionlib, actionlib_msgs, geometry_msgs, Pose, Point, Quaternion, PoseWithCovarianceStamped,std_msgs, rospy, numpy, math, os, ar_track_alvar_msgs*

# 4 QR Detection for URL and Transmitting to Wifi Module

## 4.1 Description

QR codes often contain data for a locator, identifier, or tracker that points to a website or application. A QR code uses four standardized encoding modes (numeric, alphanumeric, byte/binary, and kanji) to store data efficiently; extensions may also be used. Applications include product tracking, item identification, time tracking, document management, and general marketing, and in particular instruction embedding, which is an interesting feature for our project. We intend to perform an autonomous process in relation with a custom QR code.

After moving to the desired marker location, the Turtlebot starts to move towards the QR code frame and obtain the embedded codes inside, which contain the URL to connect to the web server. It will build a web server on the NodeMCU and make the electronic connections for the Light bulb which will indicate the control signals. Using the turtlebot computer, we will open the GET request URLs and connect the turtlebot to the web server which will automatically switch on/off the light bulb through the NodeMCU. We have to detect the QR code and transmit. More libraries are studied in later part of the project.

## 4.2 Libraries for ESP8266 Wifi Module on NodeMCU

*T*he ESP8266 is a low-cost Wi-Fi microchip with full TCP/IP stack and microcontroller capability which is embedded on a NodeMCU. Both are used to make a web server which is mobile responsive and it can be accessed with any device with a browser in your local network.

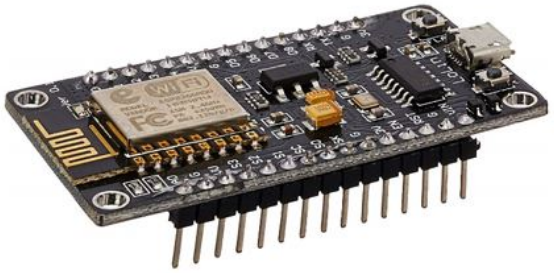

Figure 8: Sample Wifi Module

*N*odeMCU provides access to the GPIO (General Purpose Input/Output) and a pin mapping table is part of the API documentation.

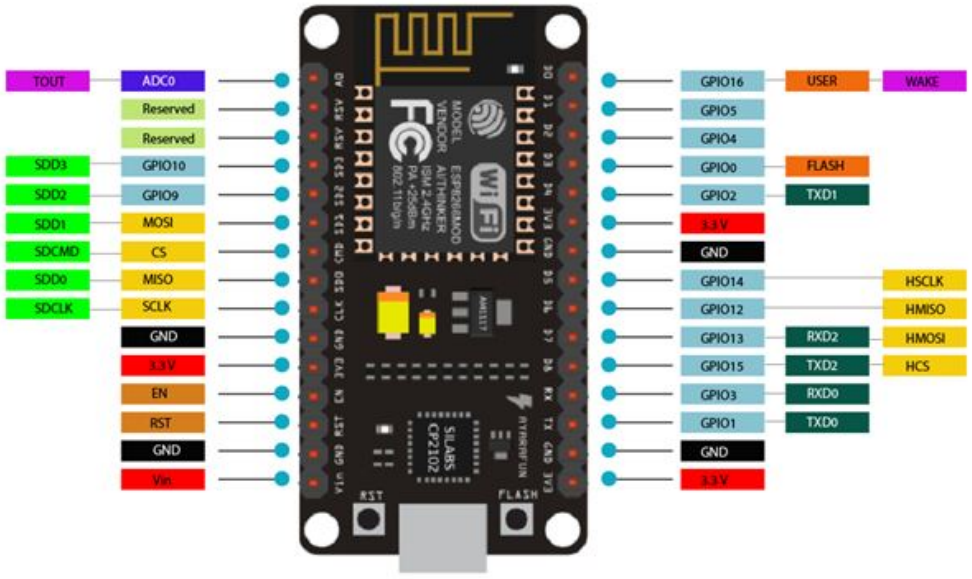


Figure 9: sample Wifi Module - Pin diagram

## 5 Conclusion

*F*irstly, we localize the Turtlebot with the AR tags. Next, we detect the QR tag. Finally we operate a Wifi module with the reference link decoded from the QR image which can be further processed for embedded applicaiton. In this way, we can explore the numerous applications of IOT.

## 6 References

1. **ROS Materials:** `All supporting notes from Dr.Ralph Seulin`
2. **Textbooks:** `ROS by Example Volume-1 and Volume-2`
3. **Github:** `https://github.com/francoislegd/BSCV`$_R OS$
4. **Github:**

   *https://github.com/gopi231091/ Mapping_LocalizationOfARMarkers_Navigation_ROS_Turtlebot/*
5. **Github:** `https://github.com/krishnan793/qrcode`$_z bar$
6. **ROS:** `http://wiki.ros.org/ar`$_t rack_a lvar$
7. **ROS:** `https://github.com/ekstrand/ESP8266wifi`