

UNIVERSITÉ DE BOURGOGNE

MASTER OF COMPUTER VISION - ROBOTIC PROJECT

---

**KURAbot  
Kura-kura Restaurant Assistant Robot**

---

*Authors:*

Dennis CHRISTIE

Osama MAZHAR

Omid MORADTALAB

*Supervisor:*

Ralph SEULIN

Cansen JIANG

Raphael DUVERNE

Robotic Research Group

Centre Universitaire Condorcet - uB, Le Creusot

Dec 2015



UNIVERSITÉ DE BOURGOGNE

## *Abstract*

MSCV

Centre Universitaire Condorcet - uB, Le Creusot

MSCV - ROBOTIC Project

**KURAbot - Kura-kura Restaurant Assistant Robot**

by Authors

The purpose of this report is to introduce frame works of basic theoretical and practical parts of Turtlebot by using Linux, ROS. Here we will have complete consideration of robot navigation, mapping, localization, control and computer vision using ROS. To achieve this purpose we will consider many details of mentioned robot and a plan will be introduced to robot to do the scenario. We will consider this robot as Kitchen Assistance to deliver food from kitchen to customer. In this report we will pay to discuss more detail about how we implement the project and knowledge behind every steps from making map and localization to detect the goal and deliver the task.



# Contents

<b>Abstract</b>	iii
<b>Acknowledgements</b>	iv
<b>1 Introduction</b>	1
<b>2 General Concept of ROS</b>	3
2.1 What is ROS . . . . .	3
2.2 Why ROS . . . . .	3
2.3 How ROS Works . . . . .	4
<b>3 Setting Up</b>	7
3.1 Motion Control . . . . .	7
3.1.1 Motion Only with Linear and Angular Velocity . . . . .	7
3.1.2 Motion Control with Odometry . . . . .	9
3.1.3 Motion Control with JoyStick . . . . .	10
3.1.4 Motion Control with Path Planning (move_base) . . . . .	11
3.1.5 Conclusion of Motion Control . . . . .	13
3.2 Map Building . . . . .	13
3.3 Navigation and Localization . . . . .	15
3.4 Configurations . . . . .	16
3.4.1 base_local_planner_params.yaml . . . . .	17
3.4.2 costmap_common_params.yaml . . . . .	17
3.4.3 global and local_costmap_params.yaml . . . . .	18
<b>4 Implementation</b>	19
4.1 Scenario . . . . .	19
4.2 Robot Tasks . . . . .	21

4.2.1	Navigates . . . . .	22
4.2.2	AR Tags Detection . . . . .	22
AR Tags for Delivery Order	. . . . .	23
AR Tags for Table Indicator	. . . . .	24
4.2.3	Voice Control . . . . .	25
4.3	State Machine . . . . .	26
4.3.1	Creating a State . . . . .	27
4.3.2	Welcoming State . . . . .	27
4.3.3	Waiting Order State . . . . .	28
4.3.4	Navigation States . . . . .	29
4.3.5	Searching Table State (with Concurrence Class)	30
4.4	Final . . . . .	33
<b>5</b>	<b>Problems Faced</b>	<b>35</b>
5.1	Initialization Position Problem . . . . .	35
5.1.1	Description . . . . .	35
5.1.2	Solution . . . . .	35
5.2	AR Tag for Table Indicator Problem . . . . .	36
5.2.1	Description . . . . .	36
5.2.2	Solution . . . . .	36
5.3	Noisy Map Problem . . . . .	37
5.3.1	Description . . . . .	37
5.3.2	Solution . . . . .	37
<b>6</b>	<b>Conclusion</b>	<b>39</b>
<b>A</b>	<b>Environment Needed</b>	<b>41</b>
A.1	Map Building . . . . .	41
A.2	AR Tags . . . . .	41
A.3	Sounds . . . . .	42
<b>B</b>	<b>Source Code</b>	<b>43</b>
B.1	Main Program (delivering_food_10.py) . . . . .	43
B.2	Waiting Order State (computer_vision_beep.py) . . . . .	61

B.3	Rotate State (rotate360.py) . . . . .	64
B.4	Search State (search_table_beep.py) . . . . .	65
B.5	Waiting State (do_stuffs_beep.py) . . . . .	68
B.6	Workstation PC Launch File . . . . .	71
B.7	Turtlebot Notebook Launch File . . . . .	71
B.8	Full Smach Graph . . . . .	72
	<b>Bibliography</b>	<b>75</b>



# List of Figures

2.1 Register Nodes to ROS Master . . . . .	4
2.2 Publisher and Subscriber Nodes . . . . .	5
3.1 Motion RVIZ . . . . .	8
3.2 Motion Real . . . . .	8
3.3 Joy2twist Node Graph [4] . . . . .	10
3.4 Arena that built by MSCV6 . . . . .	14
3.5 Generated Map . . . . .	15
3.6 Navigation 1 . . . . .	16
3.7 Navigation 2 . . . . .	16
3.8 Noisy Map . . . . .	18
3.9 Low Raytrace . . . . .	18
4.1 Overall Scenario of KURAbot Design . . . . .	19
4.2 Order Delivery Task with No Temporary Obstacles . . . . .	20
4.3 Position of Checkpoints . . . . .	20
4.4 Order Delivery Task with Temporary Obstacles . . . . .	21
4.5 Marker from 1 until 6 . . . . .	23
4.6 Voice Control Graph . . . . .	26
4.7 Simple Navigation State . . . . .	30
4.8 Extended Navigation State . . . . .	31
B.1 Full Smach Graph from Smach Viewer . . . . .	73



# Chapter 1

## Introduction

Robotic technology is playing fundamental role in future life. Here, we want to consider knowledge behind of robotic system based on ROS which is theoretical and practical work on Robotic Lab. To achieve this purpose, we implement special plan and scenarios on the robot to do predefined tasks. The Robot Operating System (ROS) is a framework for writing robot software. It is a collection of tools, libraries, and conventions that aim to simplify the task of creating complex and robust robot behavior across a wide variety of robotic platforms [1].

We named the Turtlebot as KURAbot, that name came from Indonesian word for turtle, kura-kura. KURAbot is an abbreviation of Kura-Kura Restaurant Assistant and its task is to deliver food from the kitchen to the customers. It can be used in automatic restaurants and such places which someone needs to give the costumers order.

In this practical work we used different references like ROSwiki [1] tutorials and ROS by Examples under hydro distribution [7]. After learning the Professor Lectures and tutorials of basic practical of ROS, Linux and Turtlebot details we worked on implementation step by step after testing during the lab sessions and then we did Robotic Project Lab under our professor supervision and thanks a lot to helpful lectures and supervision of our professors.

The first Idea for our robot task came from automatic resturant idea to make a robot assistant in the kitchen to deliver food for costumers. And then we start to make a plan to do such tasks by KURAbot. After careful

consideration and many examples and tests, the idea finally became the reality.

The abilities of KURAbot are obstacle avoidance, Motion control, automatic navigation and localization, tag recognition, automatic noise removal, voice indication according to each scenarios and recognize the step to take order for costumer. So, it will introduce itself and start running to arrive goal. Then, it will start speaking to take order and if it is successful, it will come back to kitchen to bring the food and so on. When it faced with noisy environment, we suggest new scenarios to make clear the noise during. Each of AR Tags contains an id, so KURAbot can differentiate them. Since we have 6 points as customer table, and another 1 AR tag for indication of the table, we used 7 different AR Tags. We have two checkpoint location and when robot detect that the map is noisy it will come back to these points and by rotation it will clear the noisy map (by using laser scan).

In next step we will consider more details of robot working and overall structure of implementation, mapping, navigation, recognition, state machine and so on. We will consider all things behind algorithms from creating the map to detection the goal and finish the robot mission. Also, we will suggest some methods to make robot work better. The following chapters will be considered for our report:

- Chapter 1: Introduction to the basic topic
- Chapter 2: General Concept of ROS
- Chapter 3: Setting Up
- Chapter 4: Implementation
- Chapter 5: Problems Faced
- Chapter 6: Conclusion

## Chapter 2

# General Concept of ROS

### 2.1 What is ROS

The Robot Operating System (ROS) is a flexible framework for writing robot software. It is a collection of tools, libraries, and conventions that aim to simplify the task of creating complex and robust robot behavior across a wide variety of robotic platforms [1]. ROS system is comprised of a number of independent nodes, each of which communicates with the other nodes using a publish/subscribe messaging model [7]. For example, a particular sensor's driver might be implemented as a node, which publishes sensor data in a stream of messages. These messages could be consumed by any number of other nodes, including filters, loggers, and also higher-level systems such as guidance, path finding, etc.

### 2.2 Why ROS

Creating truly robust general-purpose robot software is hard. From the robot's perspective, problems that seem trivial to humans often vary wildly between instances of tasks and environments. Dealing with these variations is so hard that no single individual, laboratory, or institution can hope to do it on their own. As a result, ROS was built from the ground up to encourage collaborative robotics software development [1].

Moreover, with the great concept of nodes, the tasks can be done in different system. ROS nodes allow user to communicate with any sensor drivers even if it is not in the same system, as long as they are connected in

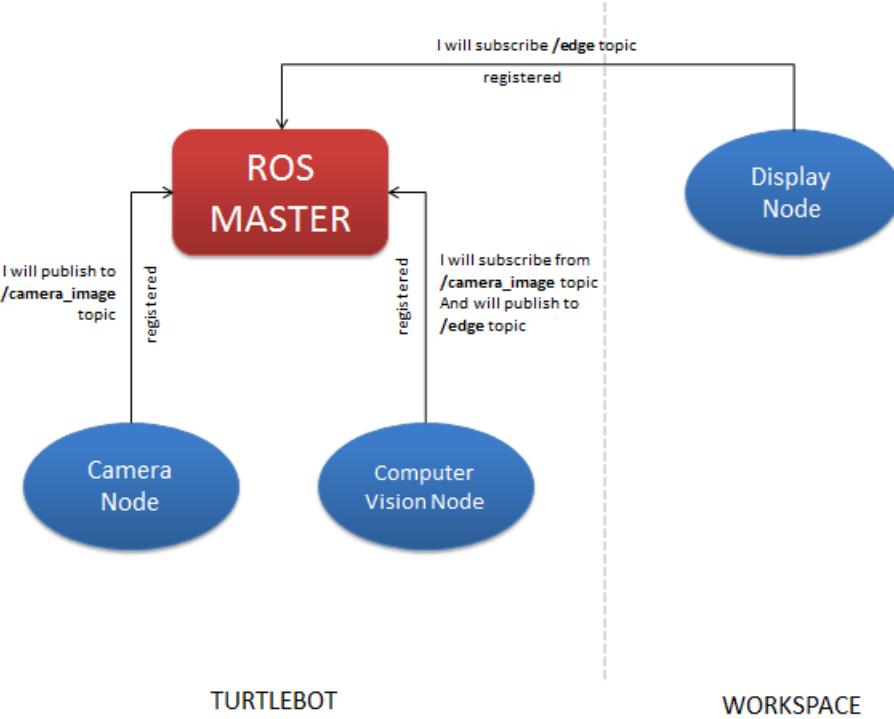


FIGURE 2.1: Register Nodes to ROS Master

the same wireless connection. This makes ROS really flexible and adaptable to the needs of the users.

## 2.3 How ROS Works

ROS Start with ROS Master. The ROS Master provides naming and registration services to the rest of the nodes in the ROS system. The role of the Master is to enable individual ROS nodes to locate one another. Once these nodes have located each other they communicate with each other peer-to-peer [7]. To make easier understanding, just consider ROS Master has somekind of lookup table where all the nodes go to find where exactly to send messages.

For example, in Figure 2.1, we create three nodes, there are Camera Node, Computer Vision Node, and Display Node. Camera Node will communicate with camera, Computer Vision node will process the image that come from Camera Node, Display Node will display image that has been processed by Computer Vision Node

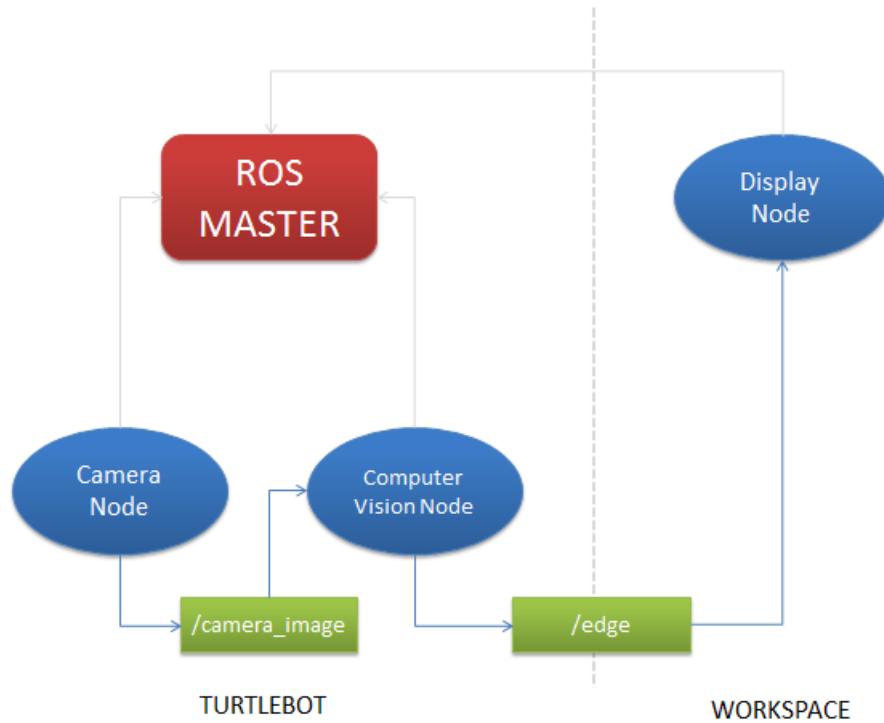


FIGURE 2.2: Publisher and Subscriber Nodes

All the nodes registered with ROS Master also with its behavior for being publisher or subscriber (Figure 2.2). Thus, once the Camera Node receives some data from the Camera, Camera Node will publish a camera stream data message called `/camera_image`, Computer Vision Node will subscribe that message and will publish a `/edge` message, Display Node will subscribe `/edge` message, and do stuffs whatever user want to do (through what is essentially TCP/IP).



# Chapter 3

## Setting Up

### 3.1 Motion Control

Motion Control are the core of this project. KURAbot must has a procedure to make it move as we want. There are a lot of ways to make the KURAbot move from one place to another. In this section several ways are introduced, and in the end of this section the solution provided.

#### 3.1.1 Motion Only with Linear and Angular Velocity

ROS using twist message type for publishing motion commands to be used by the base controller (it is usually called /cmd\_vel). From ROS documentation, twist message contains two submessages that is Vector3. Each of sub message contains three element: x, y, and z.

```

1 geometry_msgs/Vector3 linear
2   float64 x
3   float64 y
4   float64 z
5 geometry_msgs/Vector3 angular
6   float64 x
7   float64 y
8   float64 z

```

LISTING 3.1: Twist Message

Twist message basically contains information about linear velocity and angular velocity in x, y, and z axis of turtlebot. Since turtlebot doesn't have omni-direction wheel, we can only use x axis for linear velocity. And also,

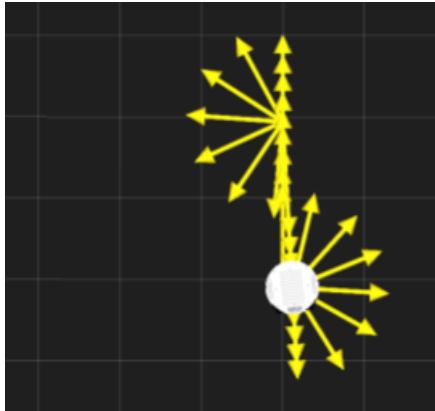


FIGURE 3.1:  
Motion RVIZ

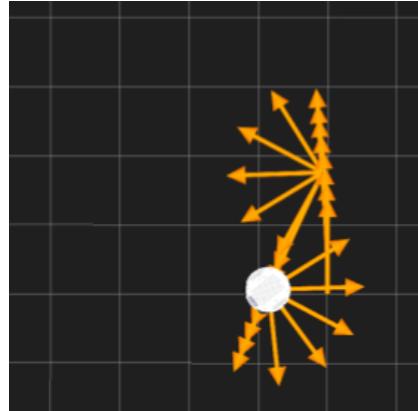


FIGURE 3.2:  
Motion Real

since turtlebot is a static ground robot (not a flying robot), we can only use z axis for angular velocity.

In the first motion command, we can simply do a basic calculation to make the turtlebot move from one point to another. With specified distance and linear velocity or specified angle and angular velocity, we can obtain amount of time we should publish this message. Turtlebot will move or rotate within this amount of time.

This way of motion control do not guarantee that the robot will go 1 meter forward, nor 180 rotation. Due to external condition, (such as the wheel, the area plane, and another disturbance) turtlebot sometimes not really going at the speed that we already specified. Since this way of motion control only depends on time, there is no guarantee that the turtlebot will go to the place we desired.

If we command the turtlebot to take a simple navigation by executing `odom_out_and_back.py` from `rbx1_nav` package that provided in ROS by Example Tutorials, the result shown in Figure 3.1 and Figure 3.2. Figure 3.1 shows how motion was really good in simulation with RVIZ, but when implemented in turtlebot, it is completely different, Figure 3.2 shows turtlebot can not return to its original position.

### 3.1.2 Motion Control with Odometry

Odometry is the use of data from motion sensors to estimate change in position over time. Odometry is used by some robots to estimate (not determine) their position relative to a starting location. From ROS documentation, odometry message contains a Header, a string identifying the child\_frame\_id, and two sub-messages, one for PoseWithCovariance and one for TwistWithCovariance.

```
1 Header header
2   uint32 seq
3   time stamp
4   string frame_id
5 string child_frame_id
6 geometry_msgs/PoseWithCovariance pose
7   geometry_msgs/Pose pose
8     geometry_msgs/Point position
9       float64 x
10      float64 y
11      float64 z
12     geometry_msgs/Quaternion orientation
13       float64 x
14       float64 y
15       float64 z
16       float64 w
17     float64[36] covariance
18 geometry_msgs/TwistWithCovariance twist
19   geometry_msgs/Twist twist
20     geometry_msgs/Vector3 linear
21       float64 x
22       float64 y
23       float64 z
24     geometry_msgs/Vector3 angular
25       float64 x
26       float64 y
27       float64 z
28     float64[36] covariance
```

LISTING 3.2: Odometry Message

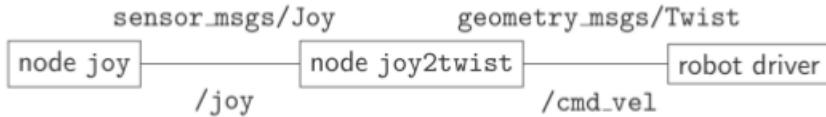


FIGURE 3.3: Joy2twist Node Graph [4]

The PoseWithCovariance sub-message records the position and orientation of the robot while the TwistWithCovariance component gives us the linear and angular speeds as we have already seen. Both the pose and twist can be supplemented with a covariance matrix which measures the uncertainty in the various measurements [5].

Before, we publish Twist message to /cmd\_vel topic by only using time as reference (it is just like guessing distances and angles based on time and speed). To make a better result of turtlebot's translation or rotation, we use our previous pose and compare it to the pose we desired, we will keep publish twist message to /cmd\_vel topic until our previous pose reach our desired position.

This way of motion control only can be performed by using odometry. If we command the turtlebot to take a simple navigation by executing nav\_square.py from rbx1\_nav package that provided in ROS by Example Tutorials, the turtlebot will move in a square shape and return to original position (which is almost impossible to do if we don't use odometry)

### 3.1.3 Motion Control with JoyStick

There is also another way to control turtlebot motion by manually. Manually here means user can control turtlebot in runtime by using another device, such as joystick. Figure 3.3 shows the general idea to do this motion control. We have to provide a node that will handle communication with joystick and publish a message in respect to linear and angular velocity message if there is some event from the joystick (when user press the button).

```

1 ...
2 def __init__(self):
  
```

```

3   rospy.init_node('joy2twist_node', anonymous=False)
4   self.cmd_vel = rospy.Publisher('/cmd_vel', Twist)
5   rospy.Subscriber("/joy", Joy, self.callback_function)
6   rospy.spin()
7
8 def callback_function(self, data):
9     msg = Twist()
10
11    # left analog button for forward-backward
12    msg.linear.x = data.axes[1]
13    # left analog button for left-right
14    msg.angular.z = data.axes[0]
15
16    self.cmd_vel.publish(msg)
17 ...

```

LISTING 3.3: Joy2Twist

In order to do that, we created a node that will subscribe to /joy topic, convert joystick message to twist message, and publish it to /cmd\_vel topic. After we run this program, turtlebot will start moving if user move the left analog button.

### 3.1.4 Motion Control with Path Planning (`move_base`)

Subsection 3.1.1, subsection 3.1.2, and subsection 3.1.3 basically can control the motion of turtlebot, but none of them can handle an obstacle. It is mandatory to consider about obstacle, because in real life, walls, tables, chairs, are an obstacle that our robot should not hit one of them.

ROS provides a much more elegant way to do the same thing using the `move_base` package [5]. Instead of using distance or angle as reference to do a motion, point to point movement was achieved by using the `move_base` package. This package helps in moving the turtlebot to any desired position only by specifying coordinate with (0,0) for its origin position. Since this motion control based on coordinates, it will help users a lot, because we will not concern anymore about the path, `move_base` package itself will handle this, which is called the Path Planning.

The move\_base package uses the MoveBaseActionGoal message type for specifying the goal [5]. From ROS Documentation [8], MoveBaseActionGoal message contains a lot of things to fill, but in practice, we only need few of the fields like the position and orientation.

```

1 Header header
2   uint32 seq
3   time stamp
4   string frame_id
5 actionlib_msgs/GoalID goal_id
6   time stamp
7   string id
8 move_base_msgs/MoveBaseGoal goal
9   geometry_msgs/PoseStamped target_pose
10  Header header
11    uint32 seq
12    time stamp
13    string frame_id
14   geometry_msgs/Pose pose
15   geometry_msgs/Point position
16     float64 x
17     float64 y
18     float64 z
19   geometry_msgs/Quaternion orientation
20     float64 x
21     float64 y
22     float64 z
23     float64 w

```

LISTING 3.4: Joy2Twist

The move\_base node requires four configuration files before it can be run. We need to specify a number related to how move\_base will behave such as radius of robot, tolerance of goal, maximum and minimum of velocity value, and others. We will discuss this in section 3.4

### 3.1.5 Conclusion of Motion Control

There are 4 ways to do a motion control, Motion Control only with Linear and Angular Velocity (subsection 3.1.1), Motion Control with Odometry (subsection 3.1.2), Motion Control with Joystick (subsection 3.1.3) and Motion Control with Path Planning or move\_base (subsection 3.1.4). Every motion control basically can be used depends on the situation, if we want a full control of the turtlebot, Motion Control with Joystick will be used, if we want to move by usind coordinates, Motion Control with Path Planning will be used, if we want to make an extraordinary moves (not like navigating) Motion Control with Odometry will be used. Every motion control except the first one, because the error is beyond the tolerance.

## 3.2 Map Building

Turtlebot will run in indoor environment. This situation leads that turtlebot supposed to able to avoid obstacles (such as walls). By motion control with move\_base, now turtlebot is able to avoid obstacles, the problem arise because turtlebot do not know what are the obstacles unless we specifiy it. Turtlebot have to know first what is around so that it can aware to the obstacles. To make the turtlebot know the environment around, we have to give an information about the map.

There are two solution to do this. First, we can create the map manually by looking at the real scene and sketch it in any graphics program by ourself, and second, we let the turtlebot to build its own map. Since the turtlebot already equipped with a laser scan, it is better to let the turtlebot create its own map. It will be more accurate than we create it by ourself by looking towards the real scene (if the users are not measuring the map appropriately, it will leads to miscalculation obstacles by the turtlebot).

Building the map can be perform by using gmapping package that provided by ROS. Gmapping package is a third party package that developed by Brian Gerkey [4]. The gmapping package provides laser-based SLAM (Simultaneous Localization and Mapping), as a ROS node called

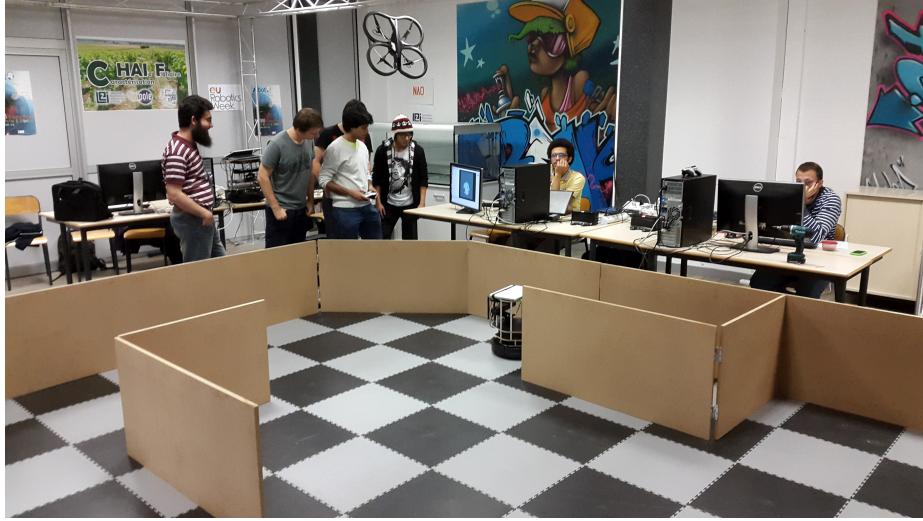


FIGURE 3.4: Arena that built by MScV6

`slam_gmapping`. This node does the work of combining the data from laser scans and odometry into an occupancy map. GMapping is a highly efficient Rao-Blackwellized particle filer to learn grid maps from laser range data [9]. There are several procedure to perform the map building

1. First we put our robot in the map. The place of the robot will be the origin of the map (coordinate 0,0), and also turtlebot initialization point that later use of the map. We can change the initial pose if we need it
2. Launch every launch files and node needed, there are rplidar and gmapping launch file for turtlebot notebook, RVIZ visualization and joy2twist node (subsection 3.1.3) for workstation PC. Appendix A shows every command that executed in terminal. RVIZ will help us to interact with the process result of gmapping.
3. Teleoperate the turtlebot around the area manually by the joystick. Teleoperating needs to be careful, sometimes if the turtlebot move to fast, the map will not generated correctly.
4. After every spot in the area generated into a map, then we save the map by using `map_saver` node. This will create two files, our map in pgm format, and configuration file of the map in yaml format. These

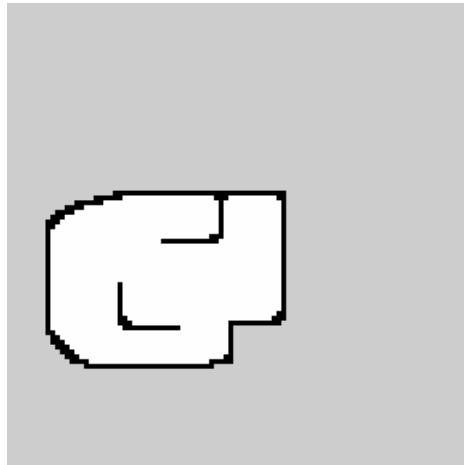


FIGURE 3.5: Generated Map

pairs of files should be together in the same directory, and will be used for Navigation and Localization, that will discussed more detail in section 3.3

5. Sometimes the map contains noises or errors. We used graphics program (such as GIMP), to remove these noises and errors.

Figure 3.4 is the real arena that built by MCSV6 and Figure 3.5 is the generated map that used in this project. As shown in Figure 3.5, the map is shifted, not centered, this is caused by our starting point when we were creating the map. The center of the picture shows where we started.

### 3.3 Navigation and Localization

Until this point, the turtlebot can move to one point to another, know its position relative to the origin position, avoid obstacles (such as walls) in the map, but turtlebot still can not determine where does its location in the map. This process called localization, knowing where we are in the known map.

In the real world, human can know his/her location in the map by looking a sign in the street, or another feature like houses, interesting statues. Robots does something similar, they use features around them to localize

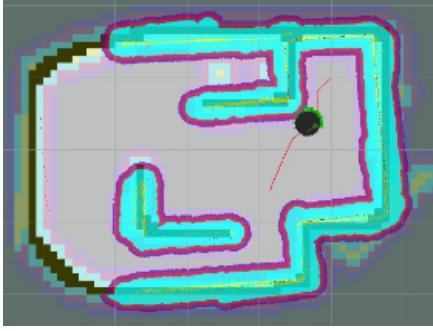


FIGURE 3.6:  
Navigation 1



FIGURE 3.7:  
Navigation 2

themselves in the map [15]. There are two ways to collect these features, we can use depth camera by Kinect or Laser Scan.

AMCL is a probabilistic localization system for a robot moving in 2D. It implements the adaptive (or KLD-sampling) Monte Carlo localization approach, which uses a particle filter to track the pose of a robot against a known map [2]. Each possible location is represented by a particle, and particles that do not match well to map are removed, which is resulting only a group of particles representing the location of the robot in the map. ROS uses the amcl package to localize the robot within an existing map using the current scan data coming from the robot's laser or depth camera [5].

Motion control by move\_base and AMCL with generated map are best combination, makes the turtlebot accurately moving from one location to another. Figure 3.6 and Figure 3.7 shows how the turtlebot know its location in the known maps (section 3.2)

### 3.4 Configurations

As already mentioned in subsection 3.1.4, motion control with move\_base requires four configuration files before it can be run. We have to give a specific value to some of the parameters, so that the turtlebot can behave as we desired.

### 3.4.1 base\_local\_planner\_params.yaml

We saw our behavior of our turtlebot. When we commanded it to move to certain pose in the map, sometimes it hardly got the exact location point. When the turtlebot arrived at the location area that we specified, the turtlebot tried to move forward, backward, rotate, etc. This is due to little tolerance specified. Turtlebot tried to make an exact orientation and an exact position point.

In order to remove this behavior, we increased parameter of the tolerances. yaw\_goal\_tolerance and xy\_goal\_tolerance from 0.1 into 0.25 (around 15 degrees of yaw tolerance, and around 25 cm of coordinate tolerance). Since we increased the tolerance, that means bigger possibility that the turtlebot is not going to the exact position and orientation, but 15 degrees of yaw tolerance and 25 cm of coordinate tolerance is not a big deal for our project purpose, so we choose this value for the tolerances.

### 3.4.2 costmap\_common\_params.yaml

Another behavior of our turtlebot is, sometimes it can not go to certain pose in the map due to radius of the obstacle. Figure 3.8 shows sometimes amcl leaves noisy particles in certain area. This situation will make our turtlebot cannot go to the yellow circled area, due to radius of the obstacle.

We tried to decrease inflation\_radius parameter a little bit from 3.0 to 2.5, but it is not giving any good result. Then we tried to decrease raytrace\_range parameter from 3.0 into 2.0, it will decrease the range of laser scan as shown in Figure 3.9, sometimes the range smaller than in the picture. At the beginning we thought this solution is good because we will not have noisy map in the far distance, then we realize that if we decrease the raytrace, that means we will decrease the particles obtained, and it will lead to miscalculation of localization by amcl. So at last we revert back to default value, but we solve this problem in another way, that will be discussed more detail in Chapter 4.



FIGURE 3.8:  
Noisy Map



FIGURE 3.9:  
Low Raytrace

### 3.4.3 `global and local_costmap_params.yaml`

We leave every value in this configuration files as default values. We did multiple time of testing and thought that every value that specified in this files already good enough.

# Chapter 4

## Implementation

### 4.1 Scenario

As stated earlier, KURAbot is designed such that it helps the restaurant representatives to deliver food to their customers. The scenario for the operation of KURAbot mainly includes a predefined map that is already scanned by the LiDAR laser sensor and considered “known” to the robot. The serving tables for the customers are assumed to be placed in fixed positions namely table1, table2 until table6 in our case.

The robot starts from the middle of the map, initialize its position and orientation with respect the known map and align itself with the data coming from LiDAR while it goes to the predefined position near the chef's counter. The robot waits for the order until chef puts the food on the tray

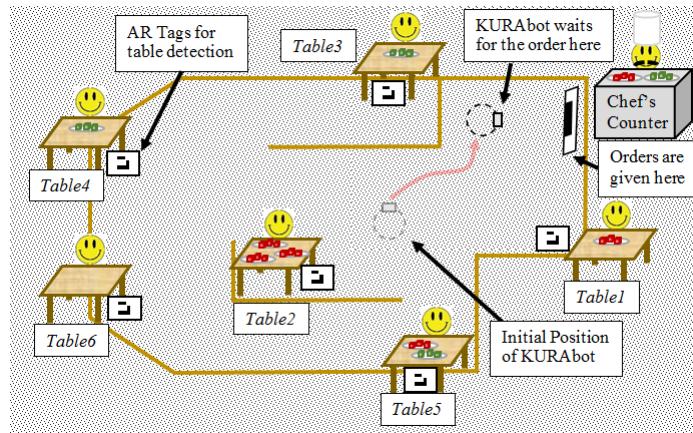


FIGURE 4.1: Overall Scenario of KURAbot Design

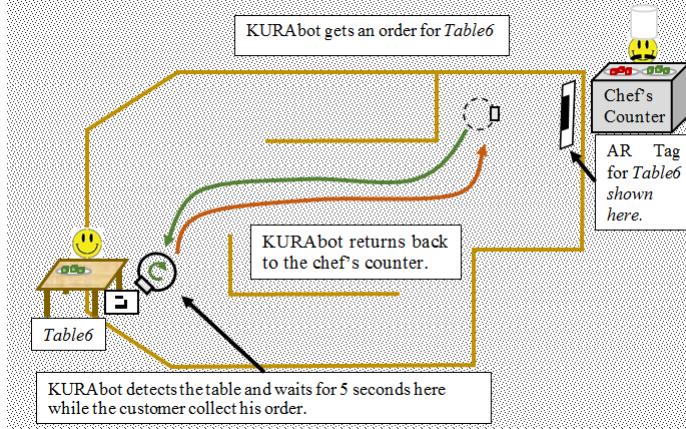


FIGURE 4.2: Order Delivery Task with No Temporary Obstacles

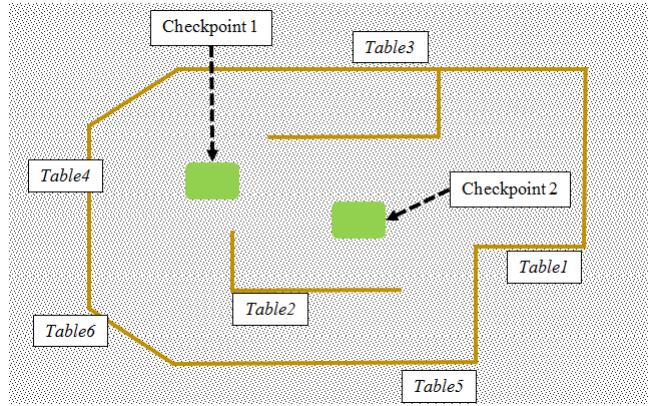


FIGURE 4.3: Position of Checkpoints

over the robot and assert a command by showing a specified AR Tag ranging from one to six corresponding to the table number. KURAbot detects the tag and starts moving towards the corresponding table. The pose for each table (waypoint) is already defined with the specified position and orientation. As soon as the robot reaches the table, it starts rotating until it detects a table with another unique AR Tag or the rotation exceeds 360 degrees.

When the table is detected the robot beeps a sound to indicate the detection, wait until 5 seconds so that the customers pick their order, thanks the customer with good wish and returns to the kitchen/chef's counter again. If the robot could not detect the table, it goes back to the chef's counter without any delay.

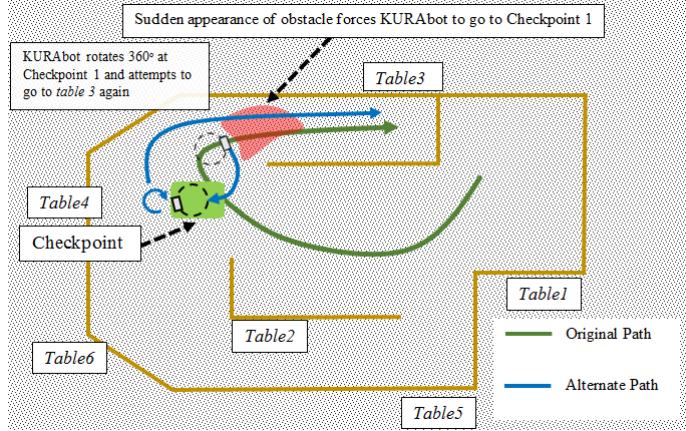


FIGURE 4.4: Order Delivery Task with Temporary Obstacles

The design of KURAbot inherently takes care of the fixed obstacles (walls in the arena) while estimating its trajectory to move towards the specified waypoint/table. However if the robot detects any other obstacle on its way to the service, it is still capable to deal with them. For this purpose two checkpoints are defined in the map as shown in Figure 4.1.

If the robot detects an obstacle it adopts another route towards one of the nearest checkpoint, rotates for 360 degree to scan and wait until the obstacle has moved, it goes back to its desired destination to serve the order. If the obstacle do not move while the robot has come for the second time, the robot will move back to the chef's counter while raising an unsuccessful flag signal.

In order to make the provision of service robust, a series of events are defined in the algorithm. KURAbot moves from one checkpoint to the other and back to the first checkpoint while it continuously scans for the obstacles to make sure that the service is successfully completed before it returns to the chef.

## 4.2 Robot Tasks

In order to become a good restaurant assistant, KURAbot supposed to be able to do, at least, three general task. These task are, Navigation, Tags Detection, and Voice Indication. There are a lot more details when performing

each task. These details will explained as follows

### 4.2.1 Navigates

First main task of restaurant assistant is able to deliver food from the kitchen to the the customers. It is mandatory for KURAbot to be able to navigates to points in the map. With map provided (section 3.2), motion control with move\_base (section 3.1.4), localization with AMCL (section 3.3), and tuned configuration files (section 3.4), navigation task become an easy task to do. With the existing scenario, KURAbot will do the navigation task perfectly as we desired.

It has been mentioned in section 4.1, we already specified there are one kitchen room and six table of the costumers, and two checkpoint location. These two checkpoint location were made in order to solve the problem that arise when there are noises in the map (subsection 3.4.2). In order to avoid cancellation of navigation to certain point that caused by noises in the map, some addition points called checkpoints are used, so that KURAbot has higher possibility to arrive at the point we desired from those checkpoints rather than from the origin of the position, kitchen.

All the navigation task considered as a state, and being called by a state machine. The details of navigation with state machine will be discussed more detail in subsection 4.3.4.

### 4.2.2 AR Tags Detection

Second main task of restaurant assistant is able to recognize the delivery order and adjust its position towards the customer table. After the chef done with the food, he will put the food on the top of KURAbot and give a command to go to the table of the customer who order that food. When KURAbot arrive to the table, it supposed to be able to face the customers for better service.

AR Tags can be used as delivery order and table indicator. Each of AR Tags contains an id, so KURAbot can differentiate them. Since we have 6

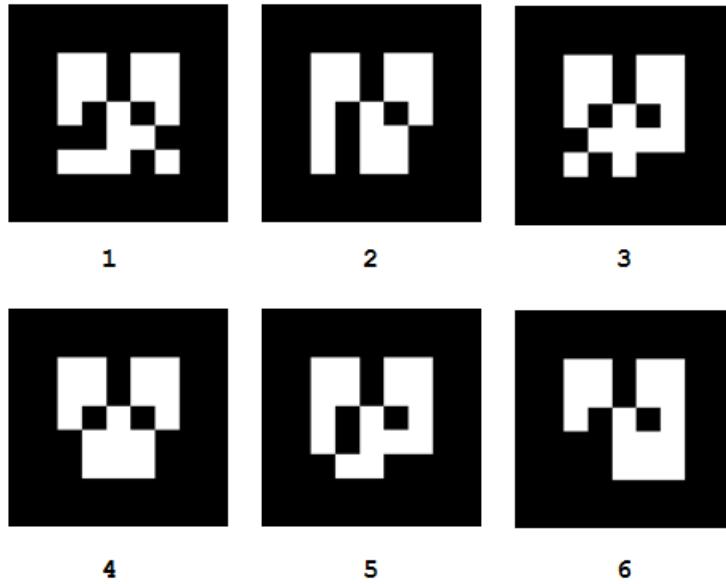


FIGURE 4.5: Marker from 1 until 6

points as customer table, and another 1 AR tag for indication of the table, we used 7 different AR Tags.

ROS provide ar\_track\_alvar package that will handle AR Tags detection. ar\_track\_alvar has several functionalities, two of them are generating AR tags of varying size, resolution, and data/ID encoding and identifying and tracking the pose of individual AR tags [3]. ar\_track\_alvar package can generate an AR Tags which has id from 0 until 65535.

### AR Tags for Delivery Order

Since we only have six customer table, we generated 6 AR Tags from 1 until 6. Appendix A.2 shows command that we used to generate the AR Tags. We generate an AR Tags which has an ID number 0 that has a size 30 cm x 30 cm. Larger tags are useful for navigation and localization since they are more easily recognized from a greater distance. Smaller tags can be used to label objects that will be viewed at a closer range [5]. Figure are the tags that considered to become a delivery order from table 1 to table 6

## AR Tags for Table Indicator

Similar to AR Tags for delivery order, we generated one more tags, which has ID number 7, and will consider this last AR tag as a Table. We attached this AR tag in each of the table location. So we have 6 AR tags which has ID number 7.

There is minor problem when we tried to implement this scenario. Let's consider there is delivery order to table number 1, KURAbot will navigate itself to location number one. Once it is arrived, now it tried to face itself to the table (AR tag number 7), problem arise when sometimes it can detect the table in another table location, for example location table 2, since we also use AR tag number 7 as table indicator number 2. KURAbot will end up facing that direction instead of facing to table number 1.

We discussed to solve this problem. The easiest solution is generating another AR Tags, so that each position has its own table indicator. The drawbacks of this solution is we will have a lot of AR Tags IDs. It is better if we just use 1 AR tags for table indicator.

Fortunately, ar\_track\_alvar node publishes AlvarMarkers message that contains not only the ID, but also the pose of each detected marker on the /ar\_pose\_maker topic. From ROS Documentation [10], details of Alvar-Markers message listed as follows.

```

1 std_msgs/Header header
2   uint32 seq
3   time stamp
4   string frame_id
5 uint32 id
6 uint32 confidence
7 geometry_msgs/PoseStamped pose
8   std_msgs/Header header
9     uint32 seq
10    time stamp
11    string frame_id
12   geometry_msgs/Pose pose
13     geometry_msgs/Point position
14       float64 x

```

```

15     float64 y
16     float64 z
17     geometry_msgs/Quaternion orientation
18         float64 x
19         float64 y
20         float64 z
21         float64 w

```

LISTING 4.1: AlvarMarkers message

With position of AR Tags provided, we can use this information to differentiate table indicator AR tags from different customer table. We assumed that once KURAbot arrive to the location, the table must be near KURAbot right now. With the information of x position from AlvarMarker message (which is the distance of the marker relative to the camera) we can set the threshold, so that AR Tags from far distance will not be considered as the table in this delivery order task.

All the AR detection task considered as a state, and being called by a state machine. Details of how KURAbot manage to detect the delivery order and adjust itself to facing towards the table will discussed more detail in subsection 4.3.5.

### 4.2.3 Voice Control

The third main task of restaurant assistant is able to give voice message. It is not mandatory, but as a good restaurant assistant, KURAbot suppose to give a voice indicator what tasks that is performed right now.

ROS provide sound\_play package that provides a ROS node that translates commands on a ROS topic into sounds. The node supports built-in sounds, playing OGG/WAV files, and doing speech synthesis via festival [11]. To synthesize a voice, we can simply use say function that already provided by sound\_play class. It will give a result of (likely terrible) robot sound. We want to make KURAbot become user friendly that has friendly sound, so instead of using sound from the system, we record our own voices.

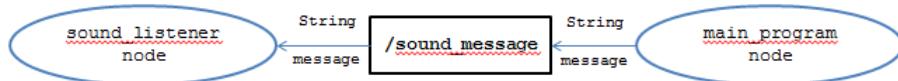


FIGURE 4.6: Voice Control Graph

We made a complete list of voices that will be used by KURAbot, the list is provided in Appendix A.3. Sound files are in WAV format and stored in KURAbot notebook.

To make a voice control, we can use a simple program called talker and listener [12] and modify it a little bit, as described in Figure 4.6. We published some strings from workstation PC to KURAbot notebook depend on task that is performed right now to a topic called /sound\_message. In KURAbot side, we created a node that subscribed string message from /sound\_message topic, with a condition specified, KURAbot will play sound file correspond to the strings obtained.

### 4.3 State Machine

Sometimes, when developing a robot, the programs will become complicated and more complicated, especially when robot have do a difficult task. If the developer still using regular programming methods, the code will become a mess. More complex the task, more complex the code, more difficult to continue the developing process. Therefore, another method of programming supposed to be invented, a method that can manage difficult task, divide them into small-simple-pieces of task. There comes state machine.

SMACH or State Machine usually used in order to help user to perform complicated tasks. SMACH syntax makes it easy to quickly prototype a state machine and start running it. SMACH allows you to design, maintain and debug large, complex hierarchical state machines. SMACH also gives you full introspection in your state machines, state transitions, data flow, etc. [13].

ROS provides SMACH package to make a state machine for our program, and also provides smach\_viewer pakage This package let us visualize the current state, all possible state transitions, data passed between the states and many more giving a complete freedom to the user to introspect and debug the system.

### 4.3.1 Creating a State

SMACH states are Python classes that extend the smach.State class by overriding the execute() method to return one or more possible outcomes [6]. We can build our own class, but there are a number of predefined state types that can save a lot of unnecessary code. In particular, the SimpleActionState class turns a regular ROS action into a SMACH state. Similarly, the MonitorState wraps a ROS topic and the ServiceState handles ROS services.

### 4.3.2 Welcoming State

KURAbot start from Welcoming State. This state only contain voice control, to make KURAbot speak and welcoming the user. This state also give us indication that everything started correctly.

```
1 class Welcome(State):
2     def __init__(self):
3         State.__init__(self, outcomes=['succeeded'])
4         self.soundstring_publisher = rospy.Publisher('sound_message',
5             String, queue_size=10)
6
7     def execute(self, userdata):
8         rospy.sleep(1)
9         rospy.loginfo("Hello, I am kura-kurabot")
10        rospy.loginfo("I will help you to deliver food to your
11        customers")
12
13        self.soundstring_publisher.publish("welcome")
14        rospy.sleep(8)
```

```
14     return 'succeeded'
```

LISTING 4.2: Welcoming State

### 4.3.3 Waiting Order State

KURAbot will go to the kitchen and waiting until delivery order comes. This state will subscribe to /ar\_pose\_maker topic, it will listen to that topic, so every event that happen in that topic, this state will run some function that required to achieve Waiting Order Process.

We will take only the most important point in this state. Basically, a state machine will execute the execute function in a state, and if it reached to returning outcome statement, the state is terminated. We do not want this state to be terminated if it is still not detect any delivery order (AR Tags). So we create a loop and the only condition that can break this loop is when this state detect at least one AR Tag. Since we are using a callback function to manage an event in /ar\_pose\_maker topic (behavior of callback function is behaving like an interrupt, will be called where ever the line of code being executed), it is possible to make this kind of process.

When the AR Tags detected, it will store its ID, breaks the loop, and right after the loop breaks, the state will know what is the delivery order. State will terminated by returning the outcomes detect1, or until detect6.

```
1 ...
2 # while there is no marker, hold the execution of execute function
3 while(self.n_markers==0):
4
5     if self.preempt_requested():
6         rospy.loginfo("State ComputerVision is being preempted!!!")
7         self.service_preempt()
8         return 'preempted'
9
10    rospy.loginfo("waiting for delivery command...")
11    rospy.sleep(1)
12
13 # right after there is marker return an outcome
14 rospy.loginfo("number detected! it is: " + str(self.my_tagid) )
```

15 . . .

LISTING 4.3: Waiting Order State

#### 4.3.4 Navigation States

After KURAbot knows what is the delivery order, KURAbot has to navigate itself to the table position correspond to the ID of AR Tag detected. We separate our navigation task into two group, Simple Navigation State (Figure 4.7), and Extended Navigation State (Figure 4.8). From figure 4.1, position of table number 1, number 2, and number 5, are easy to reach. In contrast of those table position, table number 3, number 4, and number 6, are more difficult to reach. This problem is already mentioned in subsection 3.4.2, and the general idea of solution also already mentioned in subsection 4.2.1, which is creating checkpoints.

As we can see from figure 4.1, there are two checkpoints that specified. We tried several positions to become our checkpoints, and these two locations are the chosen locations that considered as the best positions that can reach every area in the map.

Let's take an example of KURAbot is delivering food to table number 4. if KURAbot failed to go to table number 4, it will try to go to checkpoint number 2 (KURAbot will try to go to the farthest checkpoint first). If it is succeeded then proceed to go to table number 4, but if it is failed, it will try to go to checkpoint number 1 (nearest checkpoint). If it is succeeded then go to the next checkpoint and go to table number 4, but if it is still failed, KURAbot will cancel the delivery order. Similar things also applied when KURAbot want to go back to the kitchen after delivering the food.

KURAbot also have Clearing Noise State, it is a state that only contains 360 degrees rotation. In some rare case, noises in the map still not removed when KURAbot go to checkpoints, this state make sure noises removed from the map. Details of this extended navigation state can be seen in Figure 4.8.

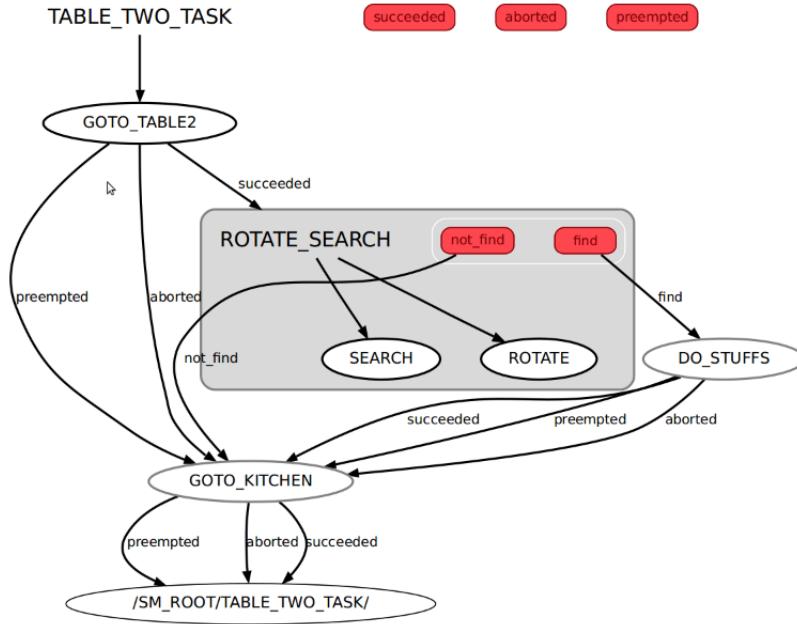


FIGURE 4.7: Simple Navigation State

#### 4.3.5 Searching Table State (with Concurrence Class)

After KURAbot arrived to table position, KURAbot has to adjust itself to face towards the table. This state actually contains two states that are running together simultaneously, Rotate State, and Searching Tag State (gray part in the state machine describe by Figure 4.7 and Figure 4.8). This process can be performed by using Concurrence Class, instead of State Machine Class. Basically, Concurrence Class is similar to State Machine Class. In fact, State Class inherit from Concurrence Class [14].

Since Concurrence Class will execute states simultaneously, there will be a minor problem, we will do not know which state will terminate first, that leads to ambiguity of outcomes in Concurrence Class. Concurrence class already provide a way to solve this problem, is to implement two callback functions called child termination callback, and outcome callback. Child termination callback function called every time one of the child states terminates. In the callback function you can decide if the state machine should keep running (return False), or if it should preempt all remaining running states (return True). Outcome callback function is called once when

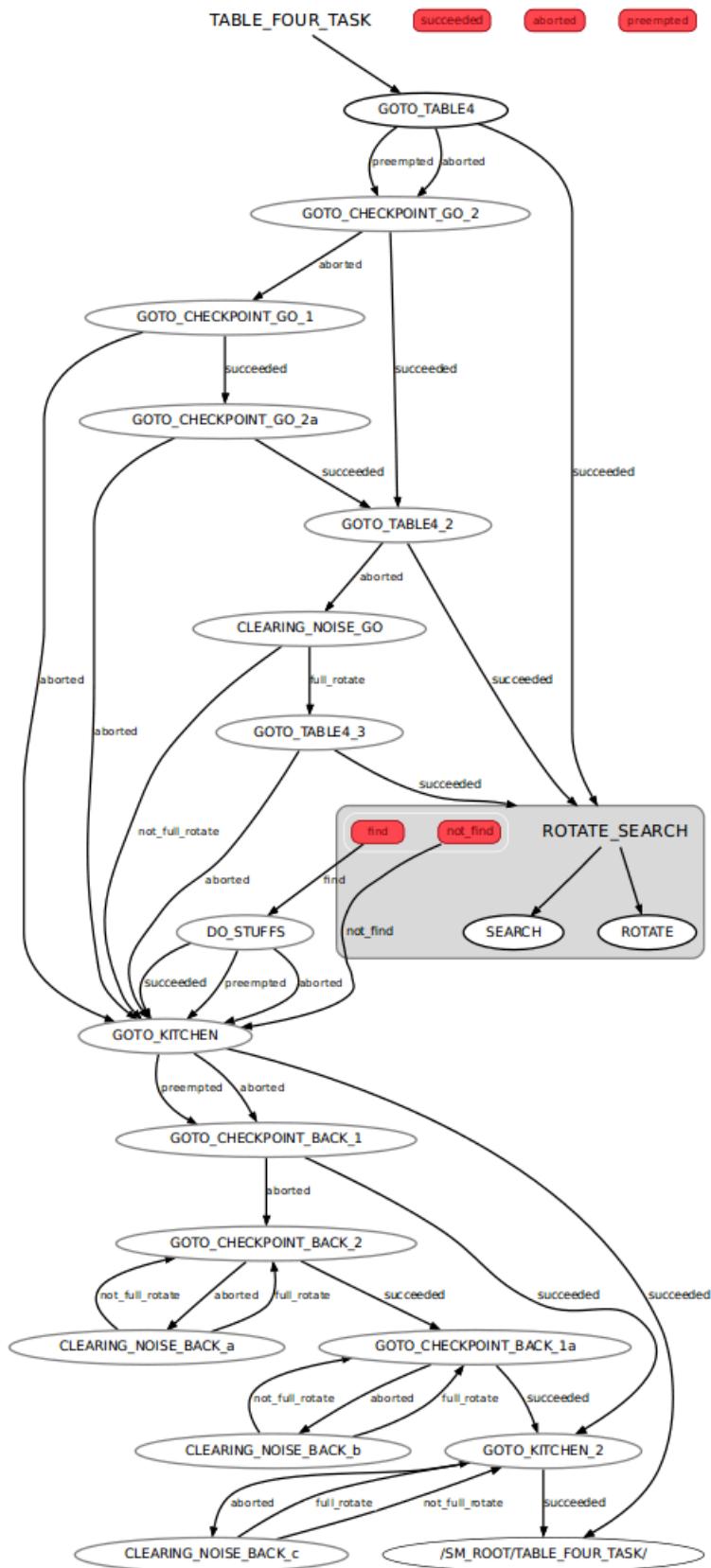


FIGURE 4.8: Extended Navigation State

the last child state terminates. This callback returns the outcome of the concurrence state machine. If we implement these two callback functions, that means we will have full control of Concurrence Class.

Let's take example of KURAbot executing Rotate and Search State simultaneously. KURAbot tried to rotate until 360 degrees, and while it is rotating, it keep searching for AR Tag nearby (below some threshold, mentioned in subsection 4.2.2). Rotate state will give full\_rotate outcome if KURAbot finish to rotate in 360 degrees, and if it doesn't finish Rotate state will give not\_full\_rotate outcome. Search state will give detect outcome if KURAbot detect AR Tag nearby, and if it didn't detect any AR Tag nearby Search state will give not\_detect outcome.

If Rotate state terminated first, it means KURAbot finish to rotate in 360 degrees, or we can say KURAbot is not detecting any AR Tag nearby. If Search state terminated first, it means KURAbot successfully detect AR Tag nearby, or we can say KURAbot is not rotating full 360 degrees. We can we can take advantage of these condition, if one of the state terminate, we terminated the other states, because we already know what is the outcomes depends on what state that terminate first. The implementation of the two callback function is written as follows

```

1 def concurrence_child_termination_callback(self, outcome_map):
2     if outcome_map['SEARCH'] == 'detect':
3         return True
4
5     elif outcome_map['ROTATE'] == 'full_rotate':
6         return True
7
8     else:
9         return False
10
11 def concurrence_outcome_callback(self, outcome_map):
12     if outcome_map['SEARCH'] == 'detect':
13         return 'find'
14
15     elif outcome_map['ROTATE'] == 'full_rotate':
16         return 'not_find'
```

```
17  
18     else:  
19         return 'not_find'
```

LISTING 4.4: Concurrence Callback Functions

## 4.4 Final

Finally, KURAbot is complete. We arranged all of those state machine to become one big state machine. The picture is too big to shown here, we put our graph of state machine in Appendix.



# Chapter 5

## Problems Faced

This is a chapter of collection of every problem that we faced when we developing KURAbot. Some already mentioned in previous chapters, some are not. We summary everything in sections that contain the description and solution of the problem

### 5.1 Initialization Position Problem

#### 5.1.1 Description

We built the map by running command that written in Appendix A.1. The place of the robot will be the origin of the map (coordinate 0,0), and also KURAbot initialization point. Our starting position of KURAbot when we built the map is in the top right of the arena. Instead of using that position, we want to make center of arena as our initial position.

#### 5.1.2 Solution

We have to publish a PoseWithCovarianceStamped message to /initial\_pose topic. We can specify our desired initial location and orientation with that message. Code below is how do we publish PoseWithCovarianceStamped message to /initial\_pose topic.

```

1 #initialize position
2 initial_publisher = rospy.Publisher('/initialpose',
3                                     PoseWithCovarianceStamped, queue_size=20)
4 rospy.sleep(3)
5 pose_msg = PoseWithCovarianceStamped()

```

```

5 pose_msg.pose.pose= self.room_locations['checkpoint1']
6 initial_publisher.publish(pose_msg)

```

LISTING 5.1: Publish message to /initialpose topic

Important note, we put sleep 3 second after we instantiate Publisher class. This is due instantiating Publisher class need some times, if we do not wait a little amount of time, initial\_publisher cannot publish anything since it is not well initialize.

## 5.2 AR Tag for Table Indicator Problem

### 5.2.1 Description

Let's consider there is delivery order to table number 1, KURAbot will navigate itself to location number one. Once it is arrived, now it tried to face itself to the table (AR tag number 7), problem arise when sometimes it can detect the table in another table location, for example location table 2, since we also use AR tag number 7 as table indicator number 2. KURAbot will end up facing that direction instead of facing to table number 1.

Something that make it worse is, when far distance AR Tag are detected in the center of our camera, it will give us good result of distance, but since in this situation KURAbot is rotating, eventually far distance AR Tag move to the edge of the camera, in this situation (in rarely times), the distance is detected close enough to the camera (sometimes below 0.7), and the result, KURAbot will end up facing that direction instead of facing to another table.

### 5.2.2 Solution

Since this situation rarely encountered (but quite terrible if it is happening), instead of just using one value of distance, we collect the distances values and take a mean of the distances. If that condition happen, since we collect more than value and most of the value are true distance, it will not change mean of distances much.

```
1 self.sum_x = self.sum_x + tag.pose.pose.position.x  
2 self.mean_distance = self.sum_x/self.counter
```

LISTING 5.2: collecting and compute mean of distances

## 5.3 Noisy Map Problem

### 5.3.1 Description

KURAbot sometimes can not go to certain pose in the map due to radius of the obstacle. Figure 3.8 shows sometimes amcl leaves noisy particles in certain area. This will leads cancelation of navigation to certain point.

### 5.3.2 Solution

In order to avoid cancelation of navigation to certain point that caused by noises in the map, some addition points called checkpoints are used, so that KURAbot has higher possibility to arrive at the point we desired from those checkpoints rather than from the origin of the position. Subsection 4.3.4 explained in detail how these checkpoints works.



## Chapter 6

# Conclusion

In this Lab Project session, we faced with a big practical working to analysis, test and implement algorithms on robot. After finishing final tests, the robot was successful for each predefined tasks.

We considered special scenarios implemented on robot. Tag detection does works well and is good enough to recognize delivery order. After each step, robot speak clearly to inform goal costumer and will be prepared for next steps. We have tested the tag detection with 6 different tags which robot gives correct output with high accuracy. All tasks done successfully. After creating and implementing mapping part, it does work but some times in some special places which the map becomes noisy to make navigation and localization more robust, we implement a scenario that make the noise clear well and will increase robot steady state. It works well also after different testing and experiments which has been implemented in the state machine.

In this lab, we learned many technical methods and applications of computer vision like servo vision, different detection schemes and so on, and also robotic part like mapping, navigation, localization and so on and both of these two parts made a strong robot to do many stuffs in real words. Also, we learned how to implement different tasks on robot by ROS to make it flexible to do any special works.



## Appendix A

# Environment Needed

### A.1 Map Building

In turtlebot notebook

```
$ rosrun turtlebot_le2i rplidar_minimal.launch
$ rosrun rbx1_nav rplidar_gmapping_demo.launch
```

In workstation PC

```
$ rosrun turtlebot_rviz_launchers view_navigation.launch
$ rosrun turtlebot_teleop keyboard_teleop.launch
```

### A.2 AR Tags

Generate AR Tags

```
$ rosrun ar_track_alvar createMarker -s 30 1
$ rosrun ar_track_alvar createMarker -s 30 2
$ rosrun ar_track_alvar createMarker -s 30 3
$ rosrun ar_track_alvar createMarker -s 30 4
$ rosrun ar_track_alvar createMarker -s 30 5
$ rosrun ar_track_alvar createMarker -s 30 6
$ rosrun ar_track_alvar createMarker -s 30 7
```

### A.3 Sounds

```
.... /new_sound_wav/welcome.wav  
"Hello, I'm KURAbot, I will help you to deliver food to the customers"  
.... /new_sound_wav/waiting_order.wav  
"Waiting for the order"  
.... /new_sound_wav/01.wav  
"Order to table one detected"  
.... /new_sound_wav/M1.wav  
"Moving to table one"  
.... /new_sound_wav/table_search.wav  
"Searching for the table, please wait"  
.... /new_sound_wav/thanks.wav  
"Thank you for the order, have a nice meal, bye.."  
.... /sound/smb_powerup.wav  
"[mario power up sound (table found)]"  
.... /sound/smb_mariodie.wav  
"[mario die sound (table not found)]"
```

## Appendix B

### Source Code

#### B.1 Main Program (delivering\_food\_10.py)

```

1 #!/usr/bin/env python
2
3 import rospy
4 import smach
5 import actionlib
6 from smach import State, StateMachine, Concurrence
7 from smach_ros import SimpleActionState, IntrospectionServer
8 from geometry_msgs.msg import Twist
9 from move_base_msgs.msg import MoveBaseAction, MoveBaseGoal
10 from geometry_msgs.msg import Pose, PoseWithCovarianceStamped,
11     Point, Quaternion, Twist
12 from visualization_msgs.msg import Marker
13 from ar_track_alvar.msg import AlvarMarkers
14
15 import easygui
16 import datetime
17 from collections import OrderedDict
18 from tf.transformations import quaternion_from_euler
19 from math import radians, pi
20 import random
21
22 import tf
23 from rbx1_nav.transform_utils import quat_to_angle,
24     normalize_angle
25
26 from custom_state_class.computer_vision_beep import ComputerVision
27 from custom_state_class.search_table_beep import SearchTable

```

```
26 from custom_state_class.do_stuffs_beep import DoStuffs
27 from custom_state_class.rotate360 import Rotate360
28
29 from std_msgs.msg import String
30
31 class Welcome(State):
32     def __init__(self):
33         State.__init__(self, outcomes=['succeeded'])
34         self.soundstring_publisher = rospy.Publisher('
35             sound_message', String, queue_size=10)
36
37     def execute(self, userdata):
38         rospy.sleep(1)
39         rospy.loginfo("Hello, I am kura-kurabot")
40         rospy.loginfo("I will help you to deliver food to your
41             customers")
42
43         self.soundstring_publisher.publish("welcome")
44         rospy.sleep(8)
45
46
47 class MyMainClass():
48     def __init__(self):
49         rospy.init_node('deliver_food', anonymous=False)
50         self.initialize_destination()
51
52         # Track success rate of getting to the goal locations
53         self.n_succeeded = 0
54         self.n_aborted = 0
55         self.n_preempted = 0
56
57         nav_states = {}
58
59         for room in self.room_locations.iterkeys():
60             nav_goal = MoveBaseGoal()
61             nav_goal.target_pose.header.frame_id = 'map'
62             nav_goal.target_pose.pose = self.room_locations[room]
```

```

63         move_base_state = SimpleActionState('move_base',
64             MoveBaseAction, goal=nav_goal, result_cb=self.
65             move_base_result_cb,
66             exec_timeout=rospy.
67             Duration(60.0),
68
69             server_wait_timeout=rospy.Duration(10.0))
70
71             nav_states[room] = move_base_state
72
73             rospy.loginfo(room + " -> [" + str(round(self.
74             room_locations[room].position.x,2)) + ", " + str(round(self.
75             room_locations[room].position.y,2)) + "]")
76
77
78             sm_rotate_search = Concurrence(outcomes=['find', 'not_find'],
79             ''],
80
81             default_outcome='not_find'
82
83             ,
84
85             child_termination_cb=self.
86             concurrence_child_termination_callback,
87
88             outcome_cb=self.
89             concurrence_outcome_callback)
90
91
92             with sm_rotate_search:
93
94                 Concurrence.add('ROTATE', Rotate360(0.4, 2*pi) )
95
96                 Concurrence.add('SEARCH', SearchTable() )
97
98
99             sm_table1 = StateMachine(outcomes=['succeeded','aborted','
100 preempted'])
101
102             with sm_table1:
103
104                 StateMachine.add('GOTO_TABLE1', nav_states['table1'],
105 transitions={'succeeded':'ROTATE_SEARCH','aborted':'
106 GOTO_KITCHEN','preempted':'GOTO_KITCHEN'})
107
108                 StateMachine.add('ROTATE_SEARCH', sm_rotate_search,
109 transitions={'find':'DO_STUFFS','not_find':'GOTO_KITCHEN'})
110
111                 StateMachine.add('DO_STUFFS', DoStuffs(5), transitions
112 ={'succeeded':'GOTO_KITCHEN','aborted':'GOTO_KITCHEN','
113 preempted':'GOTO_KITCHEN'})
114
115                 StateMachine.add('GOTO_KITCHEN', nav_states['kitchen'],
116 transitions={'succeeded':'','aborted':'','preempted':''})
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
259
260
261
262
263
264
265
266
267
268
269
269
270
271
272
273
274
275
276
277
278
279
279
280
281
282
283
284
285
286
287
288
289
289
290
291
292
293
294
295
296
297
298
299
299
300
301
302
303
304
305
306
307
308
309
309
310
311
312
313
314
315
316
317
318
319
319
320
321
322
323
324
325
326
327
328
329
329
330
331
332
333
334
335
336
337
338
339
339
340
341
342
343
344
345
346
347
348
349
349
350
351
352
353
354
355
356
357
358
359
359
360
361
362
363
364
365
366
367
368
369
369
370
371
372
373
374
375
376
377
378
379
379
380
381
382
383
384
385
386
387
388
389
389
390
391
392
393
394
395
396
397
398
399
399
400
401
402
403
404
405
406
407
408
409
409
410
411
412
413
414
415
416
417
418
419
419
420
421
422
423
424
425
426
427
428
429
429
430
431
432
433
434
435
436
437
438
439
439
440
441
442
443
444
445
446
447
448
449
449
450
451
452
453
454
455
456
457
458
459
459
460
461
462
463
464
465
466
467
468
469
469
470
471
472
473
474
475
476
477
478
479
479
480
481
482
483
484
485
486
487
488
489
489
490
491
492
493
494
495
496
497
498
499
499
500
501
502
503
504
505
506
507
508
509
509
510
511
512
513
514
515
516
517
518
519
519
520
521
522
523
524
525
526
527
528
529
529
530
531
532
533
534
535
536
537
538
539
539
540
541
542
543
544
545
546
547
548
549
549
550
551
552
553
554
555
556
557
558
559
559
560
561
562
563
564
565
566
567
568
569
569
570
571
572
573
574
575
576
577
578
579
579
580
581
582
583
584
585
586
587
588
589
589
590
591
592
593
594
595
596
597
598
599
599
600
601
602
603
604
605
606
607
608
609
609
610
611
612
613
614
615
616
617
618
619
619
620
621
622
623
624
625
626
627
628
629
629
630
631
632
633
634
635
636
636
637
638
639
639
640
641
642
643
644
645
645
646
647
648
649
649
650
651
652
653
654
655
656
657
658
659
659
660
661
662
663
664
665
666
667
668
669
669
670
671
672
673
674
675
676
677
678
679
679
680
681
682
683
684
685
686
687
688
689
689
690
691
692
693
694
695
696
697
698
699
699
700
701
702
703
704
705
706
707
708
709
709
710
711
712
713
714
715
716
717
718
719
719
720
721
722
723
724
725
726
727
728
729
729
730
731
732
733
734
735
736
737
738
739
739
740
741
742
743
744
745
746
747
748
749
749
750
751
752
753
754
755
756
757
758
759
759
760
761
762
763
764
765
766
767
768
769
769
770
771
772
773
774
775
776
777
778
779
779
780
781
782
783
784
785
786
787
788
789
789
790
791
792
793
794
795
796
797
797
798
799
799
800
801
802
803
804
805
806
807
808
809
809
810
811
812
813
814
815
816
817
818
819
819
820
821
822
823
824
825
826
827
828
829
829
830
831
832
833
834
835
836
837
838
839
839
840
841
842
843
844
845
845
846
847
848
849
849
850
851
852
853
854
855
856
857
858
859
859
860
861
862
863
864
865
866
867
868
869
869
870
871
872
873
874
875
876
877
878
879
879
880
881
882
883
884
885
886
887
888
889
889
890
891
892
893
894
894
895
896
897
898
899
899
900
901
902
903
904
905
906
907
908
909
909
910
911
912
913
914
915
916
917
918
919
919
920
921
922
923
924
925
926
927
928
929
929
930
931
932
933
934
935
936
937
938
939
939
940
941
942
943
944
945
945
946
947
948
949
949
950
951
952
953
954
955
956
957
958
959
959
960
961
962
963
964
965
966
967
968
969
969
970
971
972
973
974
975
976
977
978
979
979
980
981
982
983
984
985
986
987
988
989
989
990
991
992
993
994
995
996
997
998
999
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1088
1089
1090
1091
1092
1093
1094
1095
1095
1096
1097
1098
1099
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1178
1179
1180
1181
1182
1183
1184
1185
1186
1186
1187
1188
1189
1189
1190
1191
1192
1193
1194
1195
1195
1196
1197
1198
1199
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1288
1289
1290
1291
1292
1293
1294
1295
1295
1296
1297
1298
1299
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1388
1389
1390
1391
1392
1393
1394
1395
1395
1396
1397
1398
1399
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1488
1489
1490
1491
1492
1493
1494
1495
1495
1496
1497
1498
1499
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1588
1589
1590
1591
1592
1593
1594
1595
1595
1596
1597
1598
1599
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1688
1689
1690
1691
1692
1693
1694
1695
1695
1696
1697
1698
1699
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1788
1789
1790
1791
1792
1793
1794
1795
1795
1796
1797
1798
1799
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1888
1889
1890
1891
1892
1893
1894
1895
1895
1896
1897
1898
1899
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1988
1989
1990
1991
1992
1993
1994
1995
1995
1996
1997
1998
1999
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2088
2089
2090
2091
2092
2093
2094
2095
2095
2096
2097
2098
2099
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2188
2189
2190
2191
2192
2193
2194
2195
2195
2196
2197
2198
2199
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2238
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2278
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2288
2289
2290
2291
2292
2293
2294
2295
2295
2296
2297
2298
2299
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2338
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2348
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2378
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2388
2389
2390
2391
2392
2393
2394
2395
2395
2396
2397
2398
2399
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2429
2430
2431
2432
2
```

```

85     sm_table2 = StateMachine(outcomes=['succeeded', 'aborted', 'preempted'])
86
87     with sm_table2:
88
88        StateMachine.add('GOTO_TABLE2', nav_states['table2'],
89             transitions={'succeeded':'ROTATE_SEARCH','aborted':'GOTO_KITCHEN',
90                 'preempted':'GOTO_KITCHEN'})
91
92        StateMachine.add('ROTATE_SEARCH', sm_rotate_search,
93             transitions={'find':'DO_STUFFS','not_find':'GOTO_KITCHEN'})
94
95        StateMachine.add('DO_STUFFS', DoStuffs(5), transitions
96             ={'succeeded':'GOTO_KITCHEN','aborted':'GOTO_KITCHEN',
97                 'preempted':'GOTO_KITCHEN'})
98
99        StateMachine.add('GOTO_KITCHEN', nav_states['kitchen'],
100             transitions={'succeeded':'','aborted':'','preempted':''})
101
102     sm_table3 = StateMachine(outcomes=['succeeded', 'aborted', 'preempted'])
103
104     with sm_table3:
105
105        StateMachine.add('GOTO_TABLE3', nav_states['table3'],
106             transitions={'succeeded':'ROTATE_SEARCH',
107
108                 'aborted':'GOTO_CHECKPOINT_GO_2',
109
110                 'preempted':'GOTO_CHECKPOINT_GO_2'})
111
112        StateMachine.add('ROTATE_SEARCH', sm_rotate_search,
113             transitions={'find':'DO_STUFFS',
114
115                 'not_find':'GOTO_KITCHEN'})
116
117        StateMachine.add('DO_STUFFS', DoStuffs(5), transitions
118             ={'succeeded':'GOTO_KITCHEN',
119
120                 'aborted':'GOTO_KITCHEN',
121
122                 'preempted':'GOTO_KITCHEN'})
123
124        StateMachine.add('GOTO_KITCHEN', nav_states['kitchen'],
125             transitions={'succeeded':'',
126
127                 'aborted':'GOTO_CHECKPOINT_BACK_1'})
128
129
130 # if something wrong when we tried to go to the table

```

```

107         StateMachine.add('GOTO_CHECKPOINT_GO_2', nav_states['
checkpoint2'], transitions={'succeeded':'GOTO_TABLE3_2',
                                         'aborted'
:'GOTO_CHECKPOINT_GO_1' } )

109

110         StateMachine.add('GOTO_CHECKPOINT_GO_1', nav_states['
checkpoint1'], transitions={'succeeded':'GOTO_CHECKPOINT_GO_2a',
                                         'aborted'
:'GOTO_KITCHEN' } )

112

113         StateMachine.add('GOTO_CHECKPOINT_GO_2a', nav_states['
checkpoint1'], transitions={'succeeded':'GOTO_TABLE3_2',
                                         'aborted'
:'GOTO_KITCHEN' } )

115

116         StateMachine.add('GOTO_TABLE3_2', nav_states['table3'],
], transitions={'succeeded':'ROTATE_SEARCH',
                                         'aborted'
:'CLEARING_NOISE_GO' })

118

119         StateMachine.add('CLEARING_NOISE_GO', Rotate360(0.8,
2*pi), transitions={'full_rotate':'GOTO_TABLE3_3',
                                         'not_full_rotate'
:'GOTO_KITCHEN' })

121

122         StateMachine.add('GOTO_TABLE3_3', nav_states['table3'],
], transitions={'succeeded':'ROTATE_SEARCH',
                                         'aborted'
:'GOTO_KITCHEN' })

124

125         # if something wrong when we tried to go back to
kitchen

126

127         StateMachine.add('GOTO_CHECKPOINT_BACK_1', nav_states['
checkpoint1'], transitions={'succeeded':'GOTO_KITCHEN_2',
                                         'aborted'
:'GOTO_CHECKPOINT_BACK_2' })

```



```

151
    'aborted': 'GOTO_CHECKPOINT_GO_2',
152
    'preempted': 'GOTO_CHECKPOINT_GO_2'))
153
    StateMachine.add('ROTATE_SEARCH', sm_rotate_search,
154
        transitions={'find': 'DO_STUFFS',
155
            'not_find': 'GOTO_KITCHEN'})
156
    StateMachine.add('DO_STUFFS', DoStuffs(5), transitions
157
        ={'succeeded': 'GOTO_KITCHEN',
158
            'aborted': 'GOTO_KITCHEN',
159
            'preempted': 'GOTO_KITCHEN'})
160
    StateMachine.add('GOTO_KITCHEN', nav_states['kitchen'],
161
        transitions={'succeeded': '',
162
            'aborted': 'GOTO_CHECKPOINT_BACK_1',
163
            'preempted': 'GOTO_CHECKPOINT_BACK_1'})
164
    # if something wrong when we tried to go to the table
165
    StateMachine.add('GOTO_CHECKPOINT_GO_2', nav_states['
166
        checkpoint2'], transitions={'succeeded': 'GOTO_TABLE4_2',
167
            'aborted':
168
            : 'GOTO_CHECKPOINT_GO_1' } )
169
    StateMachine.add('GOTO_CHECKPOINT_GO_1', nav_states['
170
        checkpoint1'], transitions={'succeeded': 'GOTO_CHECKPOINT_GO_2a'
171
            ,
172
            'aborted':
173
            : 'GOTO_KITCHEN' } )
174
    StateMachine.add('GOTO_CHECKPOINT_GO_2a', nav_states['
175
        checkpoint1'], transitions={'succeeded': 'GOTO_TABLE4_2',
176
            'aborted':
177
            : 'GOTO_KITCHEN' } )
178

```

```

173         StateMachine.add('GOTO_TABLE4_2', nav_states['table4'],
174     ], transitions={'succeeded':'ROTATE_SEARCH',
175                         'aborted':'
176             'CLEARING_NOISE_GO' })
177
178         StateMachine.add('CLEARING_NOISE_GO', Rotate360(0.8,
179     2*pi), transitions={'full_rotate':'GOTO_TABLE4_3',
180                         'not_full_rotate':
181             'GOTO_KITCHEN' })
182
183         StateMachine.add('GOTO_TABLE4_3', nav_states['table4'],
184     ], transitions={'succeeded':'ROTATE_SEARCH',
185                         'aborted':'
186             'GOTO_KITCHEN' })
187
188         # if something wrong when we tried to go back to
189         kitchen
190
191         StateMachine.add('GOTO_CHECKPOINT_BACK_1', nav_states[
192     'checkpoint1'], transitions={'succeeded':'GOTO_KITCHEN_2',
193                         'aborted':
194             'GOTO_CHECKPOINT_BACK_2' })
195
196         StateMachine.add('GOTO_CHECKPOINT_BACK_2', nav_states[
197     'checkpoint2'], transitions={'succeeded':'
198             'GOTO_CHECKPOINT_BACK_1a',
199                         'aborted':
200             'CLEARING_NOISE_BACK_a' })
201
202         StateMachine.add('CLEARING_NOISE_BACK_a', Rotate360
203     (0.8, 2*pi), transitions={'full_rotate':'GOTO_CHECKPOINT_BACK_2',
204     ',',
205                         'not_full_rotate':'
206             'GOTO_CHECKPOINT_BACK_2' })
207
208         StateMachine.add('GOTO_CHECKPOINT_BACK_1a', nav_states[
209     'checkpoint1'], transitions={'succeeded':'GOTO_KITCHEN_2',
210                         'aborted':
211             'CLEARING_NOISE_BACK_b' })

```

```
195
196         StateMachine.add('CLEARING_NOISE_BACK_b', Rotate360
197             (0.8, 2*pi), transitions={'full_rotate':''
198                 'GOTO_CHECKPOINT_BACK_1a',
199
200                 ,
201
202                 'not_full_rotate':'GOTO_CHECKPOINT_BACK_1a'})
203
204
205
206         StateMachine.add('GOTO_KITCHEN_2', nav_states['kitchen']
207             ], transitions={'succeeded':'',
208
209                 'aborted':''
210                 'CLEARING_NOISE_BACK_c'})
211
212
213         StateMachine.add('CLEARING_NOISE_BACK_c', Rotate360
214             (0.8, 2*pi), transitions={'full_rotate':'GOTO_KITCHEN_2',
215
216                 ,
217
218                 'not_full_rotate':'GOTO_KITCHEN_2'})
219
220
221
222         sm_table5 = StateMachine(outcomes=['succeeded','aborted',''
223             'preempted'])
224
225         with sm_table5:
226
227             StateMachine.add('GOTO_TABLE5', nav_states['table5'],
228                 transitions={'succeeded':'ROTATE_SEARCH','aborted':''
229                     'GOTO_KITCHEN','preempted':'GOTO_KITCHEN'})
230
231             StateMachine.add('ROTATE_SEARCH', sm_rotate_search,
232                 transitions={'find':'DO_STUFFS','not_find':'GOTO_KITCHEN'})
233
234             StateMachine.add('DO_STUFFS', DoStuffs(5), transitions
235                 ={'succeeded':'GOTO_KITCHEN','aborted':'GOTO_KITCHEN',''
236                     'preempted':'GOTO_KITCHEN'})
237
238             StateMachine.add('GOTO_KITCHEN', nav_states['kitchen']
239                 ], transitions={'succeeded':'','aborted':'','preempted':''})
240
241
242         sm_table6 = StateMachine(outcomes=['succeeded','aborted',''
243             'preempted'])
244
245         with sm_table6:
246
247             StateMachine.add('GOTO_TABLE6', nav_states['table6'],
248                 transitions={'succeeded':'ROTATE_SEARCH',
249
250                     'aborted':'GOTO_CHECKPOINT_GO_2',
```

```
217         'preempted':'GOTO_CHECKPOINT_GO_2' } )  
218     StateMachine.add('ROTATE_SEARCH', sm_rotate_search,  
219     transitions={'find':'DO_STUFFS',  
220  
221         'not_find':'GOTO_KITCHEN' } )  
222     StateMachine.add('DO_STUFFS', DoStuffs(5), transitions  
223 ={'succeeded':'GOTO_KITCHEN',  
224  
225         'aborted':'GOTO_KITCHEN',  
226  
227         'preempted':'GOTO_KITCHEN' } )  
228     StateMachine.add('GOTO_KITCHEN', nav_states['kitchen'],  
229     transitions={'succeeded':'',  
230  
231         'aborted':'GOTO_CHECKPOINT_BACK_1',  
232  
233         'preempted':'GOTO_CHECKPOINT_BACK_1' } )  
234  
235     # if something wrong when we tried to go to the table  
236  
237     StateMachine.add('GOTO_CHECKPOINT_GO_2', nav_states['  
238     checkpoint2'], transitions={'succeeded':'GOTO_TABLE6_2',  
239         'aborted':  
240         :'GOTO_CHECKPOINT_GO_1' } )  
241  
242     StateMachine.add('GOTO_CHECKPOINT_GO_1', nav_states['  
243     checkpoint1'], transitions={'succeeded':'GOTO_CHECKPOINT_GO_2a',  
244         ,  
245         'aborted':  
246         :'GOTO_KITCHEN' } )  
247  
248     StateMachine.add('GOTO_CHECKPOINT_GO_2a', nav_states['  
249     checkpoint1'], transitions={'succeeded':'GOTO_TABLE6_2',  
250         'aborted':  
251         'GOTO_KITCHEN' } )  
252  
253     StateMachine.add('GOTO_TABLE6_2', nav_states['table6'],  
254     transitions={'succeeded':'ROTATE_SEARCH',  
255         ,  
256         'aborted':  
257         'GOTO_KITCHEN' } )
```

```
239             'aborted' :'
240
241             StateMachine.add('CLEARING_NOISE_GO', Rotate360(0.8,
242             2*pi), transitions={'full_rotate':'GOTO_TABLE6_3',
243             'not_full_rotate':
244             {'GOTO_KITCHEN'}})
245
246
247             StateMachine.add('GOTO_TABLE6_3', nav_states['table6'],
248             transitions={'succeeded':'ROTATE_SEARCH',
249             'aborted' :'
250             GOTO_KITCHEN'})
251
252
253             # if something wrong when we tried to go back to
254             kitchen
255
256
257             StateMachine.add('GOTO_CHECKPOINT_BACK_1', nav_states[
258             'checkpoint1'], transitions={'succeeded':'GOTO_KITCHEN_2',
259             'aborted' :
260             {'GOTO_CHECKPOINT_BACK_2'}})
261
262
263             StateMachine.add('GOTO_CHECKPOINT_BACK_2', nav_states[
264             'checkpoint2'], transitions={'succeeded':'
265             GOTO_CHECKPOINT_BACK_1a',
266             'aborted' :
267             {'CLEARING_NOISE_BACK_a'}})
268
269
270             StateMachine.add('CLEARING_NOISE_BACK_a', Rotate360
271             (0.8, 2*pi), transitions={'full_rotate':'GOTO_CHECKPOINT_BACK_2',
272             ',',
273             'not_full_rotate':'GOTO_CHECKPOINT_BACK_2'})
274
275
276             StateMachine.add('GOTO_CHECKPOINT_BACK_1a', nav_states
277             ['checkpoint1'], transitions={'succeeded':'GOTO_KITCHEN_2',
278             'aborted' :
279             {'CLEARING_NOISE_BACK_b'}})
```

```

261         StateMachine.add('CLEARING_NOISE_BACK_b', Rotate360
262             (0.8, 2*pi), transitions={'full_rotate':'
263                 GOTO_CHECKPOINT_BACK_1a',
264
265                 ,
266                 not_full_rotate:'GOTO_CHECKPOINT_BACK_1a' })
267
268             StateMachine.add('GOTO_KITCHEN_2', nav_states['kitchen
269             '], transitions={'succeeded':'',
270
271                 'aborted':''
272                 CLEARING_NOISE_BACK_c' })
273
274             StateMachine.add('CLEARING_NOISE_BACK_c', Rotate360
275             (0.8, 2*pi), transitions={'full_rotate':'GOTO_KITCHEN_2',
276
277                 ,
278                 not_full_rotate:'GOTO_KITCHEN_2' })
279
280             # let's initialize the overall state machine
281             sm_deliverfood = StateMachine(outcomes=['succeeded','
282                 aborted','preempted'])
283
284             with sm_deliverfood:
285
286                 StateMachine.add('STARTING_TASK', Welcome(),
287                     transitions={'succeeded':'GOTO_KITCHEN'})
288
289                 StateMachine.add('COMPUTER_VISION_TASK',
290                     ComputerVision(), transitions={'detect1':'TABLE_ONE_TASK',
291
292                     ,
293                     'detect2':'TABLE_TWO_TASK',
294
295                     ,
296                     'detect3':'TABLE_THREE_TASK',
297
298                     ,
299                     'detect4':'TABLE_FOUR_TASK',
300
301                     ,
302                     'detect5':'TABLE_FIVE_TASK',
303
304                     ,
305                     'detect6':'TABLE_SIX_TASK',
306
307                     ,
308                     'preempted':'' })
309
310             StateMachine.add('TABLE_ONE_TASK', sm_table1,
311                 transitions={'succeeded':'COMPUTER_VISION_TASK','aborted':'
312                 GOTO_KITCHEN','preempted':'GOTO_KITCHEN' })

```

```
282         StateMachine.add('TABLE_TWO_TASK', sm_table2,
283                         transitions={'succeeded':'COMPUTER_VISION_TASK', 'aborted':''
284                                     'GOTO_KITCHEN', 'preempted':'GOTO_KITCHEN'})
285
286         StateMachine.add('TABLE_THREE_TASK', sm_table3,
287                         transitions={'succeeded':'COMPUTER_VISION_TASK', 'aborted':''
288                                     'GOTO_KITCHEN', 'preempted':'GOTO_KITCHEN'})
289
290         StateMachine.add('TABLE_FOUR_TASK', sm_table4,
291                         transitions={'succeeded':'COMPUTER_VISION_TASK', 'aborted':''
292                                     'GOTO_KITCHEN', 'preempted':'GOTO_KITCHEN'})
293
294         StateMachine.add('TABLE_FIVE_TASK', sm_table5,
295                         transitions={'succeeded':'COMPUTER_VISION_TASK', 'aborted':''
296                                     'GOTO_KITCHEN', 'preempted':'GOTO_KITCHEN'})
297
298         StateMachine.add('TABLE_SIX_TASK', sm_table6,
299                         transitions={'succeeded':'COMPUTER_VISION_TASK', 'aborted':''
300                                     'GOTO_KITCHEN', 'preempted':'GOTO_KITCHEN'})
301
302
303     # Create and start the SMACH introspection server
304     intro_server = IntrospectionServer('deliver_food',
305                                         sm_deliverfood, '/SM_ROOT')
306
307     intro_server.start()
308
309
310     # Execute the state machine
311
312     sm_outcome = sm_deliverfood.execute()
313
314     rospy.on_shutdown(self.shutdown)
315
316
317     def move_base_result_cb(self, userdata, status, result):
318
319         if status == actionlib.GoalStatus.SUCCEEDED:
320
321             self.n_succeeded += 1
322
323         elif status == actionlib.GoalStatus.ABORTED:
324
325             self.n_aborted += 1
326
327         elif status == actionlib.GoalStatus.PREEMPTED:
328
329             self.n_preempted += 1
330
331
332     def concurrence_child_termination_callback(self, outcome_map):
```

```
308     if outcome_map['SEARCH'] == 'detect':
309         return True
310
311     elif outcome_map['ROTATE'] == 'full_rotate':
312         return True
313
314     else:
315         return False
316
317 def concurrence_outcome_callback(self, outcome_map):
318     if outcome_map['SEARCH'] == 'detect':
319         return 'find'
320
321     elif outcome_map['ROTATE'] == 'full_rotate':
322         return 'not_find'
323
324     # lazy to think, just put this
325     else:
326         return 'not_find'
327
328 def initialize_destination(self):
329     self.move_base_timeout = rospy.get_param("~move_base_timeout", 10) # seconds
330
331     # Subscribe to the move_base action server
332     self.move_base = actionlib.SimpleActionClient("move_base",
333                                                 MoveBaseAction)
334
335     rospy.loginfo("Waiting for move_base action server...")
336     self.move_base.wait_for_server(rospy.Duration(60))
337     rospy.loginfo("Connected to move_base action server")
338
339     # Orientation list
340     quaternions = list()
341
342     euler_angles = (0, -pi/2, pi, 0, pi/2, -pi/2, -3*pi/4, pi/2, pi/2)
343
344     for angle in euler_angles:
```

```

343     q_angle = quaternion_from_euler(0, 0, angle, axes='
344                                         sxyz')
345
346     q = Quaternion(*q_angle)
347     quaternions.append(q)
348
349     # Position list
350
351     points = list()
352
353     points.append(Point(0, 0, 0))
354     points.append(Point(0, -1.55497133732, 0))
355     points.append(Point(-2.21499538422, -1.55497133732, 0))
356     points.append(Point(-1.3093624115, 0.198241680861, 0))
357     points.append(Point(-3.51539182663, 0, 0))
358     points.append(Point(-0.926210045815, -2.36771917343, 0))
359     points.append(Point(-3.55022335052, -1.96715021133, 0))
360     points.append(Point(-1.13520205021, -1.3924216032, 0))
361     points.append(Point(-2.56331586838, -0.289407044649, 0))
362
363     # Create a list to hold the waypoint poses
364     self.waypoints = list()
365
366     # Append each of the four waypoints to the list. Each
367     # waypoint
368
369     # is a pose consisting of a position and orientation in
370     # the map frame.
371
372     self.waypoints.append( Pose(points[0], quaternions[0]) )
373     self.waypoints.append( Pose(points[1], quaternions[1]) )
374     self.waypoints.append( Pose(points[2], quaternions[2]) )
375     self.waypoints.append( Pose(points[3], quaternions[3]) )
376     self.waypoints.append( Pose(points[4], quaternions[4]) )
377     self.waypoints.append( Pose(points[5], quaternions[5]) )
378     self.waypoints.append( Pose(points[6], quaternions[6]) )
379     self.waypoints.append( Pose(points[7], quaternions[7]) )
380     self.waypoints.append( Pose(points[8], quaternions[8]) )
381
382     # Create a mapping of room names to waypoint locations
383
384     room_locations = (('kitchen', self.waypoints[0]),
385                           ('table1', self.waypoints[1]),
386                           ('table2', self.waypoints[2]),
387                           ('table3', self.waypoints[3])),
```

```
379         ('table4', self.waypoints[4]),
380         ('table5', self.waypoints[5]),
381         ('table6', self.waypoints[6]),
382         ('checkpoint1', self.waypoints[7]),
383         ('checkpoint2', self.waypoints[8]))
384
385     # Store the mapping as an ordered dictionary so we can
386     # visit the rooms in sequence
387
388     self.room_locations = OrderedDict(room_locations)
389
390     #initialize position
391
392     initial_publisher = rospy.Publisher('/initialpose',
393                                         PoseWithCovarianceStamped, queue_size=20)
394
395     rospy.sleep(3)
396
397     posewithcovariance_msg = PoseWithCovarianceStamped()
398
399     posewithcovariance_msg.pose.pose= self.room_locations['
400     checkpoint1']
401
402     initial_publisher.publish(posewithcovariance_msg)
403
404
405     # Initialize a marker for the docking station for RViz
406
407     self.init_waypoint_markers()
408
409     self.init_docking_station_marker()
410
411
412     self.init_waypoint_markers()
413
414     for waypoint in self.waypoints:
415
416         p = Point()
417
418         p = waypoint.position
419
420         self.waypoint_markers.points.append(p)
421
422
423     # Publisher to manually control the robot (e.g. to stop it
424
425     )
426
427     self.cmd_vel_pub = rospy.Publisher('cmd_vel', Twist)
428
429     rospy.loginfo("Starting Tasks")
430
431
432     # Publish the waypoint markers
433
434     self.marker_pub.publish(self.waypoint_markers)
435
436     rospy.sleep(1)
437
438     self.marker_pub.publish(self.waypoint_markers) # <- this
439
440     is weird
```

```
413
414      # Publish the docking station marker
415      self.docking_station_marker_pub.publish(self.
416          docking_station_marker)
417
418  def init_waypoint_markers(self):# Set up our waypoint markers
419      marker_scale = 0.2
420      marker_lifetime = 0 # 0 is forever
421      marker_ns = 'waypoints'
422      marker_id = 0
423      marker_color = {'r': 1.0, 'g': 0.7, 'b': 1.0, 'a': 1.0}
424
425      # Define a marker publisher.
426      self.marker_pub = rospy.Publisher('waypoint_markers',
427          Marker)
428
429      # Initialize the marker points list.
430      self.waypoint_markers = Marker()
431      self.waypoint_markers.ns = marker_ns
432      self.waypoint_markers.id = marker_id
433      self.waypoint_markers.type = Marker.CUBE_LIST
434      self.waypoint_markers.action = Marker.ADD
435      self.waypoint_markers.lifetime = rospy.Duration(
436          marker_lifetime)
437
438      self.waypoint_markers.scale.x = marker_scale
439      self.waypoint_markers.scale.y = marker_scale
440      self.waypoint_markers.color.r = marker_color['r']
441      self.waypoint_markers.color.g = marker_color['g']
442      self.waypoint_markers.color.b = marker_color['b']
443      self.waypoint_markers.color.a = marker_color['a']
444
445
446  def init_docking_station_marker(self):
447      # Define a marker for the charging station
448      marker_scale = 0.3
```

```

449     marker_lifetime = 0 # 0 is forever
450
451     marker_ns = 'waypoints'
452
453     marker_id = 0
454
455     marker_color = {'r': 0.7, 'g': 0.7, 'b': 0.0, 'a': 1.0}
456
457
458     self.docking_station_marker_pub = rospy.Publisher('
459         docking_station_marker', Marker)
460
461
462     self.docking_station_marker = Marker()
463
464     self.docking_station_marker.ns = marker_ns
465
466     self.docking_station_marker.id = marker_id
467
468     self.docking_station_marker.type = Marker.CYLINDER
469
470     self.docking_station_marker.action = Marker.ADD
471
472     self.docking_station_marker.lifetime = rospy.Duration(
473         marker_lifetime)
474
475     self.docking_station_marker.scale.x = marker_scale
476
477     self.docking_station_marker.scale.y = marker_scale
478
479     self.docking_station_marker.scale.z = 0.02
480
481     self.docking_station_marker.color.r = marker_color['r']
482
483     self.docking_station_marker.color.g = marker_color['g']
484
485     self.docking_station_marker.color.b = marker_color['b']
486
487     self.docking_station_marker.color.a = marker_color['a']
488
489
490     self.docking_station_marker.header.frame_id = 'odom'
491
492     self.docking_station_marker.header.stamp = rospy.Time.now
493
494     ()
495
496     self.docking_station_marker.pose = self.waypoints[0]
497
498
499     def shutdown(self):
500
501         rospy.loginfo("Stopping the robot...")
502
503         self.sm_deliverfood.request_preempt()
504
505         self.cmd_vel_pub.publish(Twist())
506
507         rospy.sleep(1)
508
509
510     if __name__ == '__main__':
511
512         try:
513
514             MyMainClass()
515
516         except rospy.ROSInterruptException:
517
518             rospy.loginfo("SMACH test finished.")

```

## B.2 Waiting Order State (computer\_vision\_beep.py)

```
1 #!/usr/bin/env python
2
3 import rospy
4 import smach
5 import actionlib
6 from smach import State, StateMachine
7 from smach_ros import SimpleActionState, IntrospectionServer
8 from geometry_msgs.msg import Twist
9 from move_base_msgs.msg import MoveBaseAction, MoveBaseGoal
10 from geometry_msgs.msg import Pose, PoseWithCovarianceStamped,
11     Point, Quaternion, Twist
12 from visualization_msgs.msg import Marker
13 from ar_track_alvar.msg import AlvarMarkers
14
15 import random
16 from sound_play.msg import SoundRequest
17 from sound_play.libsoundplay import SoundClient
18 from std_msgs.msg import String
19
20 class ComputerVision(State):
21     def __init__(self):
22         State.__init__(self, outcomes=['detect1','detect2',
23             'detect3','detect4','detect5','detect6', 'preempted'])
24         self.taskname="doing the computer vision task"
25
26         # initialize a subscriber of ar_pose_marker
27         rospy.Subscriber("ar_pose_marker", AlvarMarkers, self.
28             my_listener)
29
30         self.tag_ids = [1,2,3,4,5,6]
31         self.execute_flag = False
32
33         self.marker_table_dictionary = {1:'detect1', 2:'detect2',
34             3:'detect3', 4:'detect4', 5:'detect5', 6:'detect6'}
35         self.my_outcome = self.marker_table_dictionary[1]
36         self.n_markers = 0
37         self.my_tagid = 0
```

```
34         self.soundstring_publisher = rospy.Publisher('
35             sound_message', String, queue_size=10)
36
37     def execute(self, userdata):
38         self.execute_flag = True
39         self.soundstring_publisher.publish("waiting_order")
40         rospy.sleep(3);
41
42         # while there is no marker, hold the execution of execute
43         # function
44         while(self.n_markers==0):
45
46             if self.preempt_requested():
47                 rospy.loginfo("State ComputerVision is being
48                 preempted!!!")
49                 self.service_preempt()
50
51                 return 'preempted'
52
53             rospy.loginfo("waiting for delivery command...")
54             rospy.sleep(1)
55
56             # right after there is marker return an outcome
57             rospy.loginfo("number detected! it is: " + str(self.
58             my_tagid) )
59
60             if (self.my_tagid==1):
61                 self.soundstring_publisher.publish("O1")
62                 rospy.sleep(3)
63                 self.soundstring_publisher.publish("M1")
64                 rospy.sleep(3)
65             elif (self.my_tagid==2):
66                 self.soundstring_publisher.publish("O2")
67                 rospy.sleep(3)
68                 self.soundstring_publisher.publish("M2")
69                 rospy.sleep(3)
70             elif (self.my_tagid==3):
71                 self.soundstring_publisher.publish("O3")
72                 rospy.sleep(3)
73                 self.soundstring_publisher.publish("M3")
```

```
69         rospy.sleep(3)
70
71     elif (self.my_tagid==4):
72
73         self.soundstring_publisher.publish("O4")
74
75         rospy.sleep(3)
76
77         self.soundstring_publisher.publish("M4")
78
79         rospy.sleep(3)
80
81     elif (self.my_tagid==5):
82
83         self.soundstring_publisher.publish("O5")
84
85         rospy.sleep(3)
86
87         self.soundstring_publisher.publish("M5")
88
89         rospy.sleep(3)
90
91
92     else:
93
94         self.soundstring_publisher.publish("O6")
95
96         rospy.sleep(3)
97
98         self.soundstring_publisher.publish("M6")
99
100        rospy.sleep(3)
101
102
103    def my_listener(self, msg):
104
105
106        if self.execute_flag:
107
108            # get the number of markers
109            self.n_markers = len(msg.markers)
110
111
112            # if there is no marker stop the listener
113            # at least we already have the number of markers
114            if self.n_markers == 0:
115
116                return
117
118
119            # assuming there are only 1 tag at once
120            for tag in msg.markers:
121
122
123                # Skip any tags that are not in our list
124                if self.tag_ids is not None and not tag.id in self
125
126                    .tag_ids:
```

```

107         continue
108
109         self.my_tagid    = tag.id
110
111         self.my_outcome = self.marker_table_dictionary[tag
112             .id]
113
114     # this function will reset every attribute of this class
115     def reset_class_attribute(self):
116
117         self.execute_flag = False
118
119         self.n_markers = 0

```

### B.3 Rotate State (rotate360.py)

```

1  #!/usr/bin/env python
2
3  import rospy
4  import smach
5  import actionlib
6
7  from smach import State, StateMachine
8
9  from geometry_msgs.msg import Twist
10
11 from geometry_msgs.msg import Point, Quaternion, Twist
12 from ar_track_alvar.msg import AlvarMarkers
13
14
15 from math import radians, pi
16
17 from rbx1_nav.transform_utils import quat_to_angle,
18     normalize_angle
19
20 import tf
21
22
23 from std_msgs.msg import String
24
25
26 class DoStuffs(State):
27
28     def __init__(self, timer):
29
30         State.__init__(self, outcomes=['succeeded', 'aborted',
31             'preempted'])
32
33         self.timer=timer
34
35         self.soundstring_publisher = rospy.Publisher('
36             sound_message', String, queue_size=10)
37
38
39     def execute(self, userdata):

```

```
24         self.soundstring_publisher.publish("sound_option5")
25         rospy.sleep(2)
26
27         rospy.loginfo("Waiting for customer to take the food")
28         counter = 0
29
30         while (counter<self.timer):
31             counter+=1
32             rospy.loginfo(str(counter))
33             rospy.sleep(1)
34
35             self.soundstring_publisher.publish("thanks")
36             rospy.loginfo("Thank you for the order, have a nice meal..
")
37             rospy.sleep(4)
38             rospy.loginfo("Bye! :D")
39             rospy.sleep(1)
40
41         return 'succeeded'
```

## B.4 Search State (*search\_table\_beep.py*)

```
1 #!/usr/bin/env python
2
3 import rospy
4 import smach
5 import actionlib
6 from smach import State, StateMachine
7 from geometry_msgs.msg import Twist
8 from geometry_msgs.msg import Point, Quaternion, Twist
9 from ar_track_alvar.msg import AlvarMarkers
10
11 from math import radians, pi
12 from rbx1_nav.transform_utils import quat_to_angle,
13     normalize_angle
14
15 class Rotate360(State):
16     def __init__(self, speed, goal):
```

```
17     State.__init__(self, outcomes=['full_rotate', 'not_full_rotate'])

18
19     # Initialize the tf listener
20     self.tf_listener = tf.TransformListener()

21
22     # Give tf some time to fill its buffer
23     rospy.sleep(2)

24
25     # Set the odom frame
26     self.odom_frame = '/odom'

27
28     # Find out if the robot uses /base_link or /base_footprint
29     try:
30         self.tf_listener.waitForTransform(self.odom_frame, '/base_footprint', rospy.Time(), rospy.Duration(1.0))
31         self.base_frame = '/base_footprint'
32     except (tf.Exception, tf.ConnectivityException, tf.LookupException):
33         try:
34             self.tf_listener.waitForTransform(self.odom_frame, '/base_link', rospy.Time(), rospy.Duration(1.0))
35             self.base_frame = '/base_link'
36         except (tf.Exception, tf.ConnectivityException, tf.LookupException):
37             rospy.loginfo("Cannot find transform between /odom and /base_link or /base_footprint")
38             rospy.signal_shutdown("tf Exception")

39
40     self.cmd_vel = rospy.Publisher('cmd_vel', Twist)

41
42     self.r = rospy.Rate(20)
43     self.angular_speed = speed
44     self.angular_tolerance = radians(2.5)
45     self.goal_angle = goal

46
47     def execute(self, userdata):
48         rospy.loginfo("Rotating...")
```

```
50     # Get the starting position values
51
52     (position, rotation) = self.get_odom()
53
54     last_angle = rotation
55
56     turn_angle = 0
57
58
59     rotate_command = Twist()
60
61     rotate_command.angular.z = self.angular_speed
62
63
64     while abs(turn_angle + self.angular_tolerance) < abs(self.
65         goal_angle) and not rospy.is_shutdown():
66
67         if self.preempt_requested():
68
69             rospy.loginfo("State Rotate360 is being preempted
70             !!!")
71
72             self.service_preempt()
73
74             return 'not_full_rotate'
75
76
77
78         # Publish the Twist message and sleep 1 cycle
79
80         self.cmd_vel.publish(rotate_command)
81
82         self.r.sleep()
83
84
85         # Get the current rotation
86
87         (position, rotation) = self.get_odom()
88
89
90         # Compute the amount of rotation since the last loop
91
92         delta_angle = normalize_angle(rotation - last_angle)
93
94
95         # Add to the running total
96
97         turn_angle += delta_angle
98
99         last_angle = rotation
100
101
102         # Stop the robot before the next leg
103
104         rotate_command = Twist()
105
106         self.cmd_vel.publish(rotate_command)
107
108         rospy.sleep(1)
109
110
111         return 'full_rotate'
112
113
114     def get_odom(self):
```

```

86     # Get the current transform between the odom and base
87     frames
88
89     try:
90         (trans, rot) = self.tf_listener.lookupTransform(self.
91             odom_frame, self.base_frame, rospy.Time(0))
92
93     except (tf.Exception, tf.ConnectivityException, tf.
94         LookupException):
95
96         rospy.loginfo("TF Exception")
97
98     return
99
100
101
102
103     return (Point(*trans), quat_to_angleQuaternion(*rot)))

```

## B.5 Waiting State (do\_stuffs\_beep.py)

```

1 #!/usr/bin/env python
2
3 import rospy
4 import smach
5 import actionlib
6 from smach import State, StateMachine
7 from smach_ros import SimpleActionState, IntrospectionServer
8 from geometry_msgs.msg import Twist
9 from move_base_msgs.msg import MoveBaseAction, MoveBaseGoal
10 from geometry_msgs.msg import Pose, PoseWithCovarianceStamped,
11     Point, Quaternion, Twist
12 from visualization_msgs.msg import Marker
13 from ar_track_alvar.msg import AlvarMarkers
14
15
16 from sound_play.msg import SoundRequest
17 from sound_play.libsoundplay import SoundClient
18 from std_msgs.msg import String
19
20
21 class SearchTable(State):
22     def __init__(self):
23         State.__init__(self, outcomes=['detect', 'not_detect'])
24
25         rospy.Subscriber("ar_pose_marker", AlvarMarkers, self.
26                         my_listener)
27
28         self.n_markers = 0

```

```
24         self.table_flag = False
25
26         self.execute_flag = False
27
28         self.sum_x = 0
29
30         self.counter = 0
31
32         self.mean_distance = 0
33
34
35         self.soundstring_publisher = rospy.Publisher('
36             sound_message', String, queue_size=10)
37
38
39         def execute(self, userdata):
40
41             self.execute_flag = True
42
43             rospy.loginfo("execute SearchTable class")
44
45             self.soundstring_publisher.publish("table_search")
46
47             rospy.sleep(4)
48
49
50             # while( self.n_markers==0 or not(self.my_tagposition_y >
51             -0.2 and self.my_tagposition_y < 0.2)) :
52
53                 while( not(self.table_flag) or (self.mean_distance > 1) ) :
54
55
56                 if self.preempt_requested():
57
58                     rospy.loginfo("State SearchTable is being
59                         preempted!!!")
60
61                     self.service_preempt()
62
63                     self.reset_class_attribute()
64
65
66                     message_str = "sound_option4" # mario die sound
67
68                     self.soundstring_publisher.publish(message_str)
69
70                     rospy.sleep(3)
71
72                     return 'not_detect'
73
74
75
76                     rospy.loginfo("Searching for table...")
77
78                     rospy.sleep(1)
79
80
81                     rospy.loginfo("Table detected!")
82
83                     self.reset_class_attribute()
84
85                     return 'detect'
86
87
88
89
```

```
60     def my_listener(self, msg):
61
62         if self.execute_flag:
63             # get the number of markers
64             self.n_markers = len(msg.markers)
65
66             # if there is no marker stop the listener
67             # at least we already have the number of markers
68             if self.n_markers == 0:
69                 return
70
71             # assuming there are only 1 tag at once
72             for tag in msg.markers:
73
74                 if tag.id is 7:
75                     self.table_flag = True
76
77                     self.counter=self.counter+1
78                     self.sum_x = self.sum_x + tag.pose.pose.
position.x
79                     self.mean_distance = self.sum_x/self.counter
80
81                     rospy.loginfo("detected: " +str(tag.id) + ", "
it is: "+str(tag.pose.pose.position.x) + "m")
82                     rospy.loginfo("the mean is: "+str(self.
mean_distance) + "m")
83
84             # this function will reset every attribute of this class
85             def reset_class_attribute(self):
86                 self.table_flag = False
87                 self.execute_flag = False
88                 self.n_markers = 0
89
90                 self.sum_x = 0
91                 self.counter = 0
92                 self.mean_distance = 0
93
94             #done
```

## B.6 Workstation PC Launch File

```

1 <launch>
2   <node pkg="image_view" type="image_view" name="my_camera_node">
3     <remap from="image" to="/camera/rgb/image_color" />
4   </node>
5
6   <!--
7   <node pkg="rbx2_tasks" type="delivering_food.py" name=""
8     deliver_food_node" />
9   <node pkg="rbx2_tasks" type="something.py" name="something_node"
10    />
11   <node name="soundplay_node" pkg="sound_play" type="
12     soundplay_node.py"/>
13   <include file="$(find rbx2_ar_tags)/launch/ar_indiv_kinect.
14    launch" />
15 -->
16
17   <include file="$(find turtlebot_rviz_launchers)/launch/
18     view_navigation.launch" />
19
20 </launch>

```

LISTING B.1: Workspace Launch File

## B.7 Turtlebot Notebook Launch File

```

1 <launch>
2   <include file="$(find turtlebot_le2i)/launch/
3     remap_rplidar_minimal.launch" />
4   <include file="$(find rbx1_nav)/launch/tb_demo_amcl.launch" >
5     <arg name="map" value="my_arena.yaml" />
6   </include>
7
8   <!--
9   <include file="$(find rbx2_vision)/launch/openni_node.launch" />
10  -->
11  <include file="$(find rbx2_ar_tags)/launch/ar_indiv_kinect.
12    launch" />

```

```
11 <include file="$(find turtlebot_bringup)/launch/3dsensor.launch"
12   />
13
13 <node name="soundplay_node" pkg="sound_play" type="
14   soundplay_node.py" />
14 <node name="soundlistener_node" pkg="rbx2_tasks" type="
15   sound_listener.py" />
15
16 </launch>
```

LISTING B.2: Turtlebot Launch File

## B.8 Full Smach Graph

The next page shows the full image of our State Machine that provided by Smach Viewer. Details already explained in Chapter 5.

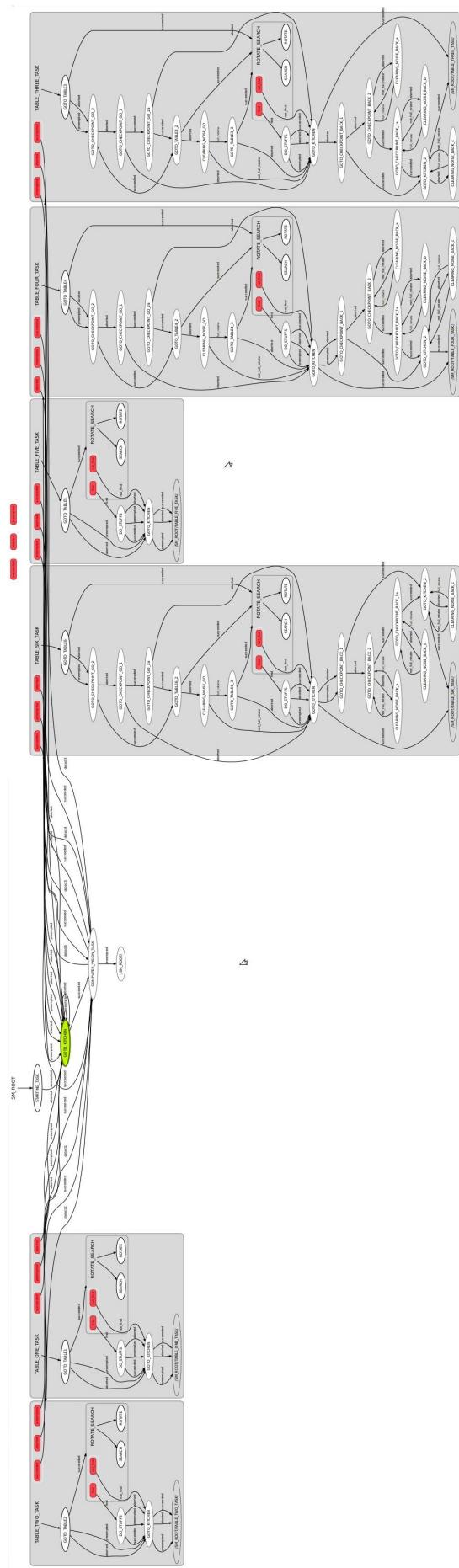


FIGURE B.1: Full Smach Graph from Smach Viewer



# Bibliography

- [1] *About ROS*. <http://www.ros.org/about-ros/>. Accessed: December 2015.
- [2] *Amcl*. <http://wiki.ros.org/amcl>. Accessed: December 2015.
- [3] *Ar Track Alvar*. [http://wiki.ros.org/ar\\_track\\_alvar](http://wiki.ros.org/ar_track_alvar). Accessed: December 2015.
- [4] *gmapping*. <http://wiki.ros.org/gmapping>. Accessed: December 2015.
- [5] Patrick Goebel. *ROS By Example*. Ed. by Lulu. Lulu, 2013. URL: \url{http://www.lulu.com/shop/r-patrick-goebel/ros-by-example-hydro-volume-1/ebook/product-21393108.html}.
- [6] Patrick Goebel. *ROS By Example*. Ed. by Lulu. Lulu, 2014. URL: \url{http://www.lulu.com/shop/r-patrick-goebel/ros-by-example-volume-2-hydro/ebook/product-21756214.html}.
- [7] *Intro to ROS*. [http://www.clearpathrobotics.com/guides/ros/Intro\\_to\\_the\\_Robot\\_Operating\\_System.html](http://www.clearpathrobotics.com/guides/ros/Intro_to_the_Robot_Operating_System.html). Accessed: December 2015.
- [8] *MoveBaseActionGoal Message*. [http://docs.ros.org/fuerte/api/move\\_base\\_msgs/html/msg/MoveBaseActionGoal.html](http://docs.ros.org/fuerte/api/move_base_msgs/html/msg/MoveBaseActionGoal.html). Accessed: December 2015.
- [9] *OpenSlam*. <http://openslam.org/gmapping.html>. Accessed: December 2015.
- [10] *Point*. [http://docs.ros.org/fuerte/api/geometry\\_msgs/html/msg/Point.html](http://docs.ros.org/fuerte/api/geometry_msgs/html/msg/Point.html). Accessed: December 2015.

- [11] *Point Message*. [http://wiki.ros.org/sound\\_play](http://wiki.ros.org/sound_play). Accessed: December 2015.
- [12] Ralph Seulin. *ROS Tutorial Module 4*. 2015.
- [13] *Smach*. <http://wiki.ros.org/smach>. Accessed: December 2015.
- [14] *Smach Class*. [http://docs.ros.org/diamondback/api/smach/html/python/smach.state\\_machine.StateMachine-class.html](http://docs.ros.org/diamondback/api/smach/html/python/smach.state_machine.StateMachine-class.html). Accessed: December 2015.
- [15] *Turtlebot Navigation*. [http://wiki.ros.org/turtlebot\\_navigation](http://wiki.ros.org/turtlebot_navigation). Accessed: December 2015.