Robotics

Technical Report

Athanasiadis Stefanos

Melidoni Despoina

Contents

Introduction

The goal of our project is to implement the part of localization and mapping of the course project. For our part, we had to move the turtlebot from the starting point (outside the lab) and bring it to the center of the classroom. Then, by using tag recognition algorithms, locate the loading and unloading areas. After that, the turtlebot should go to the loading area (Table A), where the visual servoing and the robotic arm procedures will take place. When they are competed, the turtlebot moves on to the unloading area (Table B), where the above-mentioned procedures are executed again.

Challenges

For the tag recognition part, we decided to use the AR_Track_Alvar package. We searched and found a few others, but most of them where incompatible with ros indigo and the rest had pour documentation, which made their use very difficult. Ar_Track_Alvar is really well structured, documented and extensively tested on ros indigo.

After creating the map of the lab, we tried to test the ar_track_alvar package along with the navigation part. That's where we found a conflict. We could not run the amcl launch file along with the alvar launch file. Whenever we tried the first one crushed. We had a working navigation part and a working tag recognition part, but we could not combine them. We spent three weeks trying to figure out the conflict, but then, as it was also suggested to us by Professor Crombez, we decided to move on from that problem and change the plan. Our goal now is to move the turtlebot from the staring point to the center of the lab, then go to table A, wait for the other two procedures to finish and then move to table B.

On the last week before the break, a solution to our problem was found, but unfortunately by then we did not had the time to implement the tag localization on the map that was requested from us, so we added only the tag recognition part.

Implementation

Mapping

In order for the turtlebot to be able to autonomously navigate in the lab area we had to create a 2D map of the lab. For that purpose, we used the turtlebot_navigation package.

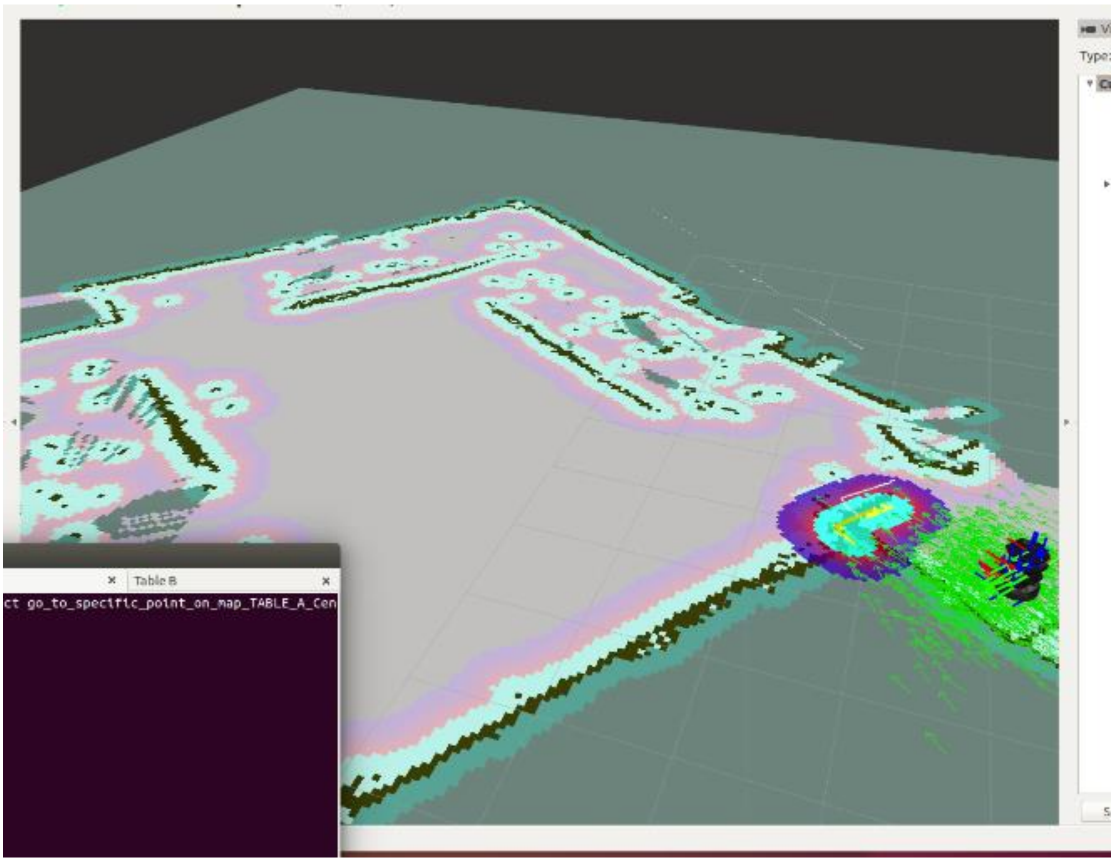
So, at first we bring up our turtlebot: roslaunch turtlebot_bringup minimal.launch

Then, to start creating the map: roslaunch turtlebot_navigation gmapping_demo.launch

We used the joystick to move the turtlebot around the lab area: roslaunch turtlebot_teleop logitech.alunch

To view the creation procedure we used rviz: roslaunch turtlebot_rviz_launchers view_navigation.launch

Finally, we save our map: rosrun map_server map_saver -f /… /my_map



Navigation & Localization

We begin our procedure by launching minimal on turtlebot. After that we launch our map. Both of these commands are launched through our launch file:

roslaunch project turtlebot.launch

```
<launch>
    <include file="$(find turtlebot_bringup)/launch/minimal.launch"></include>
    <include file="$(find turtlebot_navigation)/launch/amcl_demo.launch"/>

</launch>
```

Then we launch the rviz (as before) and finally ar_tag_alvar launch file: roslaunch project main.launch

From our starting point (outside the lab room), we use python code to give our turtlebot the coordinates to move to a specific location on the map. At first, we run the python code to make the turtlebot move to the center of the room and then to table A. After that, we run another python code to move it from table A to table B.

rosrun project go_to_specific_point_on_map_TABLE_A_Center.py rosrun project go_to_specific_point_on_map_TABLE_B.py

Figure 1: Turtlebot at starting point

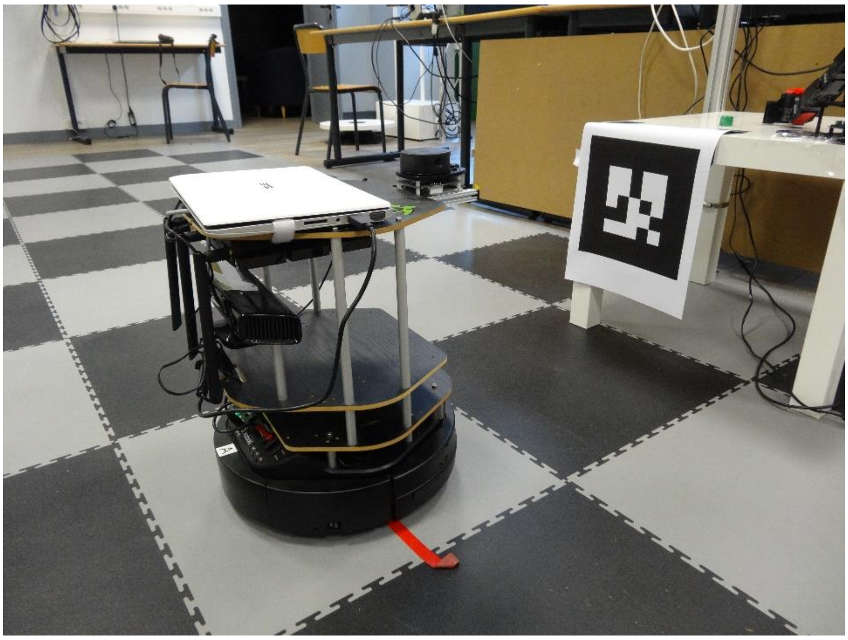


Figure 2Turtlebot at finishing point

```python
while i <2 :
    navigator = GoToPose()
    #navdonePublisher = rospy.Publisher("/relay/nav_status", std_msgs.msg.String, queue_size = 10)
    if i == 0: # center
        position = {'x': 6.53, 'y' : 1.03} #{'x': 7.22, 'y' : 0.192}
        quaternion = {'r1' : 0.000, 'r2' : 0.000, 'r3' : -2.000, 'r4' : 1.000}
        center = True
        rospy.loginfo("Go to Center (%s, %s) pose", position['x'], position['y'])
        i += 1
    elif i== 1: #table 1
        # table 1
        position = {'x': 5.77, 'y' : 0.121}
        quaternion = {'r1' : 0.000, 'r2' : 0.000, 'r3' : -2.000, 'r4' : 1.000}
        tableA = True
        rospy.loginfo("Go to Table A (%s, %s) pose", position['x'], position['y'])
        i += 1

    success = navigator.goto(position, quaternion)
```

Figure 3: Part of Python code for Center and Table A

```python
while i <2 :
    navigator = GoToPose()
    #navdonePublisher = rospy.Publisher("/relay/nav_status", std_msgs.msg.String, queue_size = 10)
    if i == 0: # pro_table2
        position = {'x': 5.77, 'y' : 0.121}
        quaternion = {'r1' : 0.000, 'r2' : 0.000, 'r3' : 0.6, 'r4' : 1.000}
        rospy.loginfo("Almost Go to Table B (%s, %s) pose", position['x'], position['y'])
        turn = True
    elif i== 1: #table 1
        # table 2
        #position = {'x': 8.57, 'y' : 0.423} # correct
        position = {'x': 8.57, 'y' : 0.423}
        quaternion = {'r1' : 0.000, 'r2' : 0.000, 'r3' : 0.6, 'r4' : 1.000}
        rospy.loginfo("Go to Table B (%s, %s) pose", position['x'], position['y'])
        tableB = True

    success = navigator.goto(position, quaternion)
```

Figure 4: Part of Python code for Table B

Tag Recognition

As mentioned before, for this part we use ar_track_alvar package and we launch the “main.launch” file in order to start the recognition procedure.

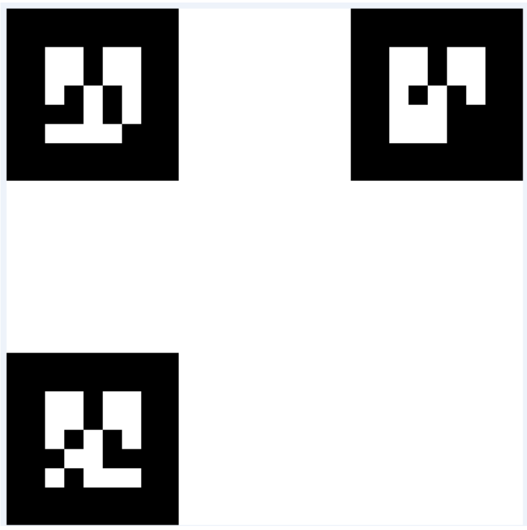


Figure 5: Possible Tags to track

```python
rospy.Subscriber('ar_pose_marker', AlvarMarkers, callback)
```

Figure 6: Subscription to topic for recognition

```python
def callback(msg):

    msg.markers
    for tag in (msg.markers):
        rospy.loginfo("Tag's Pose : %f, %f, %f",tag.pose.pose.position.x, tag.pose.pose.position.y, tag.pose.pose.position.z)
```

Figure 7: Callback function to show tags' position relatively to the turtlebot

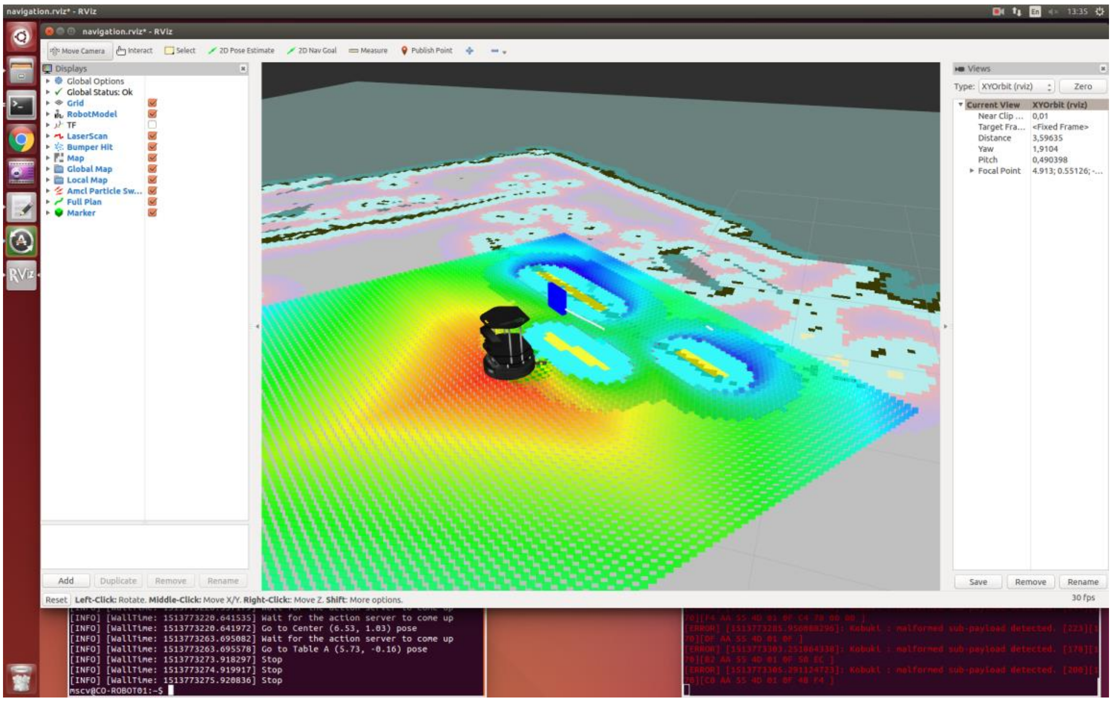


Figure 8: Turtlebot recognizes tag at Table A

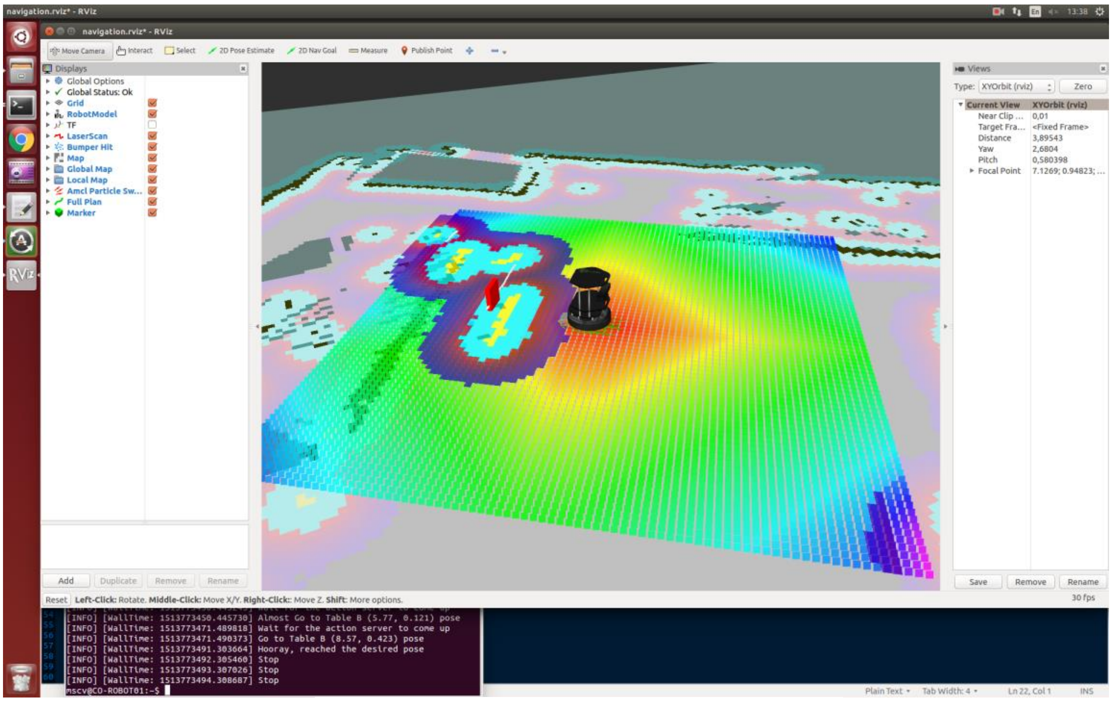


Figure 9: Turtlebot recognizes tag at Table B

Turtlebot adjustments

A few modifications were made, so that our turtlebot could respond to our code in a better way.

- <u>Solution to conflict</u>

  The solution to our conflict that was mentioned earlier between the alvar package and the amcl was the change in some values inside "turtlebot_navigation/launch/amcl_demo.launch", where these initially false values were changed to true.

```
<arg name="rgb_processing" value="false" />
<arg name="depth_registration" value="false" />
<arg name="depth_processing" value="false" />
```

Figure 10: Default file's parameters

- <u>Path planning error</u>

  At some points our turtlebot would get stuck trying to stop at the exact coordinates we give him. That was solved by decreasing its speed, so it would be more accurate at its movement. Moreover, we increased its goal tolerance, so it would be more flexible with its goal target position.

  - Speed decrease
    At "turtlebot_bringup/param/defaults/smoother.yaml" file

```
# Mandatory parameters
speed_lim_v: 0.8
speed_lim_w: 5.4

accel_lim_v: 1.0 # maximum is actually 2.0, but we push it down to be smooth
accel_lim_w: 2.0
```

Figure 11: Default file's parameters

  - Tolerance increase
    At "turtlebot_navigation/param/dwa_local_planner_params.yaml" file

```
# Goal Tolerance Parameters
  yaw_goal_tolerance: 0.3  # 0.05
  xy_goal_tolerance: 0.15  # 0.10
```

Figure 12: Default file's parameters

Results

Along with this technical report and the code, we also provide videos showing the turtlebot's course in the lab ("turtlebot.mts") and also screen captures from the computer showing our commands execution and the tags' recognition in rviz ("Screen_Capture.mp4").

Links to online videos:
Screen Capture: https://www.youtube.com/watch?v=YftMoYck1sQ&feature=youtu.be
Turtlebot
Demonstration: https://www.youtube.com/watch?v=Tj5jQ8Jz5ds&feature=youtu.be