UNIVERSITÉ DE BOURGOGNE

BACHELOR'S IN COMPUTER VISION AND ROBOTICS

Robotics Engineering Project

Technical report on a cooperative task between Turtlebot 2 and a PhantomX Pincher Robotic Arm.

Written by:

Deogratias LUKAMBA
Macaulay SADIQ
Antwi Kwaku EBENEZER

Supervisors:

Nathan CROMBEZ
nathan.crombez@u-bourgogne.fr

Yifei ZHANG
yifei.zhang.fr@gmail.com

Raphael DUVERNE
raphael.duverne@u-bourgogne.fr

Prof. Ralph SEULIN
ralph.seulin@u-bourgogne.fr

Centre Universitaire Condorcet – uB South Campus, Le Creusot
May 2018

Acknowledgements

The achievement of this project required a lot of guidance and assistance from many people and we are extremely privileged to have got this all along till the completion of our project. We would like to especially thank our Robotics Lab supervisors, Dr. Nathan CROMBEZ and Ms. Yifei ZHANG who gave us continuous guidance, assistance, and inspiration to continue efficiently working on our project and obtain expecting results. We would like to thank Mr. Raphael DUVERNE for his advice, help, and aid in troubleshooting the robot. Also, we would like to extend our sincere appreciation to the Master student Marc BLANCHON for his timely support. Finally, we would like to present a special thanks to our families, for their love, understanding, encouragement, and confidence in us.

Table of contents

I. PRESENTATION

The present report is part of our semester project in the robotics engineering module. Working in ROS environment has provided us a new approach to deal with large number of robots such as the Turtlebot 2 and the PhantomX Pincher arm. Those current autonomous mobile service robots are custom manufactured for research environments and limiting their availability. The objective was to design and implement a cooperative task for a low- cost service robot based on TurtleBot 2 platform and the Pincher-arm. This project covered a wide range of topics from how to setup ROS and the turtlebot and some of their basic packages. Also, building and setting up the PhantomX Pincher robotic arm to perform the pick and place operation for a complete project scenario.

II. PROJECT OVERVIEW

Workspace environment

Before starting the project, we completed the ROS Installation as described in the technical survey. Then we created our ROS Workspace:

$ mkdir -p ~/catkin_ws/src
$ cd ~/catkin_ws/
$ catkin_make

then we must source the setup after performing some settings on Workstation and TurtleBot2 Netbook. We have noticed that all workstations and netbooks in our robotics lab are already configured with ROS Indigo. Catkin here is the only build system used for new development, it is located at:

$ ~/ros/indigo/catkin_ws

we can edit the bash file by:
$ gedit ~/. Bashrc

To control our turtlebot which is the netbook (Asus) via the workstation (Dell).
$ ssh *turtlebot@192.168.0.100*

Topics and Nodes

Nodes use a **ROS** client library like **rospy** or **roscpp** to communicate with other nodes. They can publish or subscribe to buses over which messages are exchanged called "Topics".

Packages

In our project work we created some packages ourselves, modified existing packages or install new packages to perform specific task with our robots and Pincher arm. However, some major packages that may be very essential for our project work are briefly described as follows:

- Creating packages:
  - *turtlebot_nav*
  - *turtlebot_pose_commands*

        - *Joy2twist*
- Modified existing packages:
    - *turtlebot_navigation*
- New installed packages:
    - *rplidar-turtlebot2*
    - *arbotix_ros*
    - *turtlebot_arm*

Launch Files

Launch files were the only convenient way to start up multiple nodes and master, as well as other initialization requirements such as setting parameters for the initial pose, destination of our turtlebot within the map.

Scripts

They are created to run some specific operations controlling the turtlebot using the joystick with joy2twist_listener.py inside the joy2twist package, also the **pick_and_place.py** inside the **turtlebot_pose_commands** package was used to control the robotic arm operation.

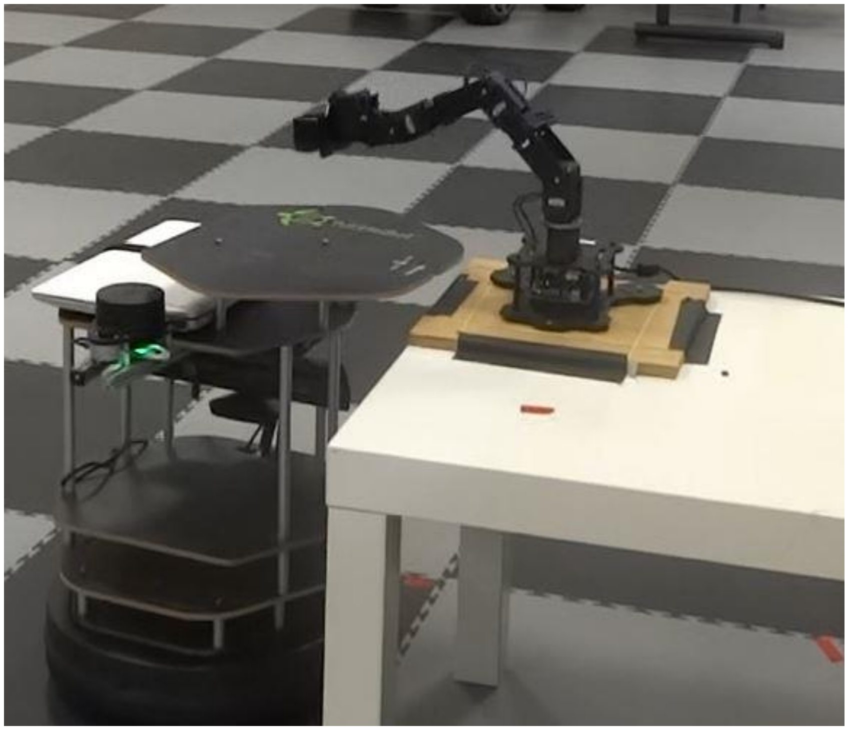
Bag File

This bag file was to record all topics which was used to create the map. Further information on how to create a map will be introduced in next section on Map creation.

III. PROJECT DESCRIPTION

The project is about the implementation of the complete scenario of the turtlebot and the robotic PhantomX pincher-arm where the turtlebot is made to navigate to the position of the arm from its initial position, on arrival at the arm position, the arm will have to pick a cube that will be positioned on a specific location on the table and place it on the turtlebot then the turtlebot will take the cube to another location which in our case is the initial position of the turtlebot.

The project implementation is primarily based on the navigation of the turtlebot on a created map by giving it a goal position command on the map and the turtlebot could plan its path with reference to its initial position on the map and navigate to the goal position executing its trajectory using the laser scan sensor and as well avoid dynamically obstacles along its path.

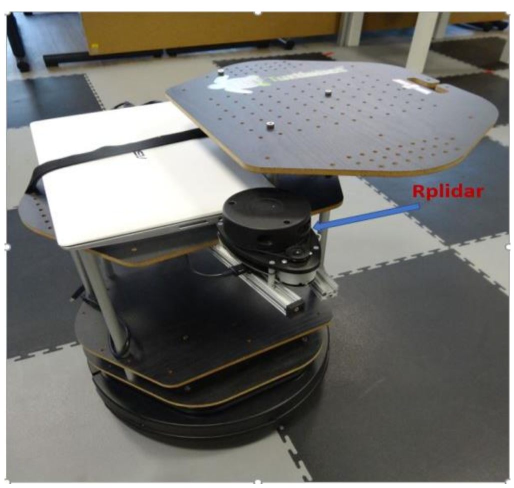


Picture showing the turtlebot at the position of the robotic pincher arm to receive the cube

IV. PROJECT SETUP

Adding the 2D Laser sensor to the turtlebot2

The type of the laser scan sensor attached to our turtlebot2 is the RPLIDAR 2D laser sensor.



Picture showing the turtlebot with the RPLIDAR sensor attached to it

The rplidar-turtlebot2 is the Ros package designed to add the sensor to the turlebot2. This package can be cloned and used from the repository; https://github.com/roboticslab-fr/rplidar-turtlebot2.git. Kindly refer to the README on the repository and follow the guides on its installation.

On the turtlebot2:

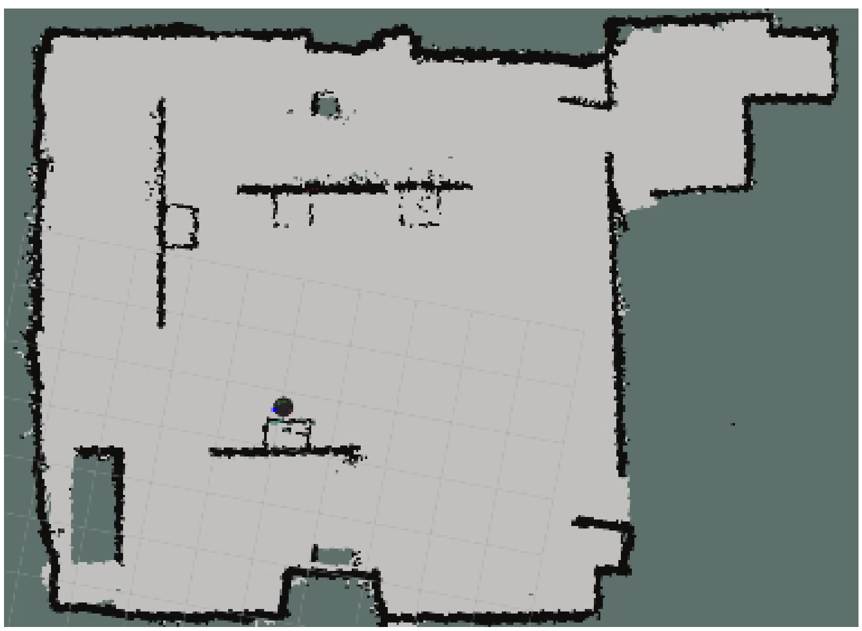
$ cd ~/ros/indigo/catkin_ws/src
$ git clone https://github.com/roboticslab-fr/rplidar-turtlebot2.git
$ cd .. && catkin_make

Map Creation

We have created a map from the bag file that was recorded while the turtlebot was manually controlled with a joystick through the space where all the tasks were to be carried out. The data required for the creation of the map is the scan topic recorded in the bag file which was published from the RPLIDAR 2D scan sensor we previously added to the turtlebot. By following the command lines given below we were able to successfully create the map called ClassMap.pgm and its corresponding *. yaml **file**: ClassMap.yaml.

$ rosmake gmapping
$ rosparam set use_sim_time true
$ rosrun gmapping slam_gmapping scan:=scan
$ rosbag play --clock ros/indigo/catkin_ws/Bags/ClassBag.bag

After the bags has finished playing then the map is obtained from the map server with the command;
$ rosrun map_server map_saver -f ClassMap



Picture of the map created for the turtlebot navigation

Installation of packages

To run the complete scenario for the project, several packages were put together including our own created packages with some modifications of script files and parameters.

The following packages will need to be installed in the Catkin workspace for the execution of the scenario.

On the turtlebot:

- To launch the RPLIDAR sensor together with the turtlebot minimal launch and the ***AMCL*** launch in the ***turtlebot_navigation*** stack package.

$ cd ~/ros/indigo/catkin_ws/src
$ git clone https://github.com/Macaulay123/BSCVRoboticsProject/tree/master/turtlebot_nav
$ cd .. && catkin_make

On the work station:

- The Arbotix ros driver to interface with the Arbotix board of the PhantomX pincher-arm.

$ cd ~/ros/indigo/catkin_ws/src
$ git clone https://github.com/Macaulay123/arbotix_ros.git
$ cd .. && catkin_make

- The turlebot_arm package to run the PhantomX pincher-arm bring-up and other packages

$ cd ~/ros/indigo/catkin_ws/src
$ git clone https://github.com/Macaulay123/turtlebot_arm.git
$ cd .. && catkin_make

- The turtlebot_pose_commands package puts the different launch for complete scenario of the Project from the work station

$ cd ~/ros/indigo/catkin_ws/src
$ git clone https://github.com/Macaulay123/BSCVRoboticsProject/tree/master/turtlebot_pose_commands
$ cd .. && catkin_make

haven completed all the set up on the workstation and on the turtlebot netbook as earlier suggested, the next step is required to launch each process of the scenario which we have simplified by creating launch files that groups other launch files from different packages that is require and sets the different parameters to them in order to achieve a particular processes of the scenario.

Navigation

In this part of the project we want to be able to assign our map to the turtlebot to navigate on, and as well, assume a default initial position of the turtlebot on the map in other to be able to give the turtlebot specific goal position command to reach. To achieve this, we considered using the **turtlebot_navigation** stack package which uses the *AMCL* package. We have created our own package called turtlebot_nav it basically contains the launch files that is used to organize the processes involved for the turtlebot navigation. The content of this launch file is displayed below.

```
<launch>
        <!--Launch the LaserScan minimal-->
    <include file="$(find turtlebot_le2i)/launch/remap_rplidar_minimal.launch" />
        <!--Launch the AMCL with the initial position on the Map -->
    <include file="$(find turtlebot_navigation)/launch/amcl_demo.launch" >
     <arg name="map_file" value="/home/turtlebot/ros/indigo/catkin_ws/Bags/ClassMap.yaml" />
     <arg name="initial_pose_x" default="9.541"/>
     <arg name="initial_pose_y" default="-6.338"/>
     <arg name="initial_pose_a" default="1.803"/>
    </include>
</launch>
```

Explanation:
The first part of the code simply launches the **remap_rplidar_minimal** launch file from the ***turtlebot_le2i*** package. This launch file does the following: *remap the 'cmd_vel_mux/input/navi' to 'cmd_vel', launches the turtlebot2 minimal launch*, sets the environment variable for the 2D rplaidar scan sensor rather than the default which is the 3D-sensor, and bring-up the 2D rplidar scan sensor. This part put together basically enables us to be able to navigate our turtlebot using the scan message published by the rplidar 2D scan sensor on the **cmd_vel** topic

The second part of our launch file code launches the **amcl_demo.launch** from the **turtlebot_navigation** package, and sets our map as an argument to the **mapserver** for ***amcl*** and then set the assumed initial position of the turtlebot on the map (its linear position and the orientation). With this process we can now conveniently place our turtlebot on the marked initial position and give it a goal command on the map for it to navigate to by simply running the following commands:

On the turtlebot2:
$ roslaunch turtlebot_nav RP_nav_turt.launch
On the work station:
$ roslaunch turtlebot_pose_commands tb_goal_pose.launch

The tb_goal_pose.launch is a launch file that simply contains a command to a target goal pose on the map which can take any value of the position and orientation of the turtlebot on the map depending on the user target goal position.
This command can as well be run directly on the workstation by publishing on the topic /move_base_simple/goal, however for convenience and organization of our work, we have chosen to rather have this launch file.

Obstacle avoidance

In this case we want the turtlebot to be able to avoid dynamical obstacles along its path to it goal position. The **turtlebot_navigation** package performs this operation by default however, the package was created to work with the 3D sensor environment variable which seem to be responsible for the response to the velocity commands received on the *cmd_vel* topic for the turtlebot base for obstacle avoidance. The rplidar laser scan environment variable seems to publish its velocity commands to the *cmd_vel_mux/input/navi**and*** as a result the turtlebot base receives two different velocity commands on different topics. The topic to be executed is the one with the highest priority and then continues with the other with a lesser priority when the first stops. Because of this, the turtlebot will continue to move in its regular path even when there is an obstacle. Also, by default, the amcl package sets the parameters for its obstacle avoidance to be responsive to the command velocity received on the *cmd_vel*. This problem was corrected eventually by first remapping the *cmd_vel_mux/input/navi* to *cmd_vel* as shown earlier in the **remap_rplidar_minimal.launch**, and it was also very important to change the parameter settings of the turtlebot_navigation by following the procedure highlighted below on the turtlebot which also requires administrative permission.

- move to the *turtlebot_navigation* directory,
- back up the param folder with a different file name as you wish,

download the files base_local_planner_params.yaml, costmap_common_params.yaml, global_costmap_params.yaml, global_planner_params.yaml, and local_costmap_params.yaml, move_base_params.yaml or simply copy the parameter settings in the each of the files with their corresponding files name in the *turtlebot_navigation* **package**.

Now by running the commands given below on the terminals

On the turtlebot2:
$ roslaunch turtlbot_nav RP_nav_turt.launch

On the work station:
$ roslaunch turtlebot_pose_commands tb_goal_pose.launch

you will have the turtlebot navigate to the given goal pose on the map and will avoid any obstacle along its path.

PhantomX pincher arm pick and place Operation

The robotic arm operation was to pick a cube from the table on which it was mounted and place the cube on the turtlebot when the turtlebot arrives to the position of the table. Since the idea was to manually control the robotic arm to pick the cube and place it on the turtlebot, a script named pick_and_place.py in ***turtlebot_pose_commands/bin*** directory was used together with the turtlebot_arm_moveit_config package to control the arm to go to a marked point on the table to pick the cube. By running the command below, we were able to achieve the robotic arm pick and place operation.

On the turtlebot2:
$ roslaunch turtlebot_nav RP_nav_turt.launch

On the work station:
$ roslaunch turtlebot_pose_commands arm_pick_and_place.launch

V. PROJECT EXECUTION

The complete scenario of our project can be executed in a stream by following these steps bellow.

On the turtlebot2:
$ roslaunch turtlebot_nav RP_nav_turt.launch

On the work station:
To view the turtlebot on the map with rviz GUI
$ roslaunch turtlebot_rviz_launchers view_navigation.launch –screen

first pose command (closer to the position of the robotic arm)
$ roslaunch turtlebot_pose_commands tb_goal_pose.launch

second pose command (to the exact position of the robotic arm)
$ roslaunch turtlebot_pose_commands tb_goal_pose.launch

robotic arm pick and place operation
$ roslaunch turtlebot_pose_commands arm_pick_and_place.launch

after the completion of the arm pick and place operation, one will have to kill the operation, and also the *turtlbot_nav RP_nav_turt.launch* operation on the turtlebot, then you can continue with the processes below.
On the turtlebot2:

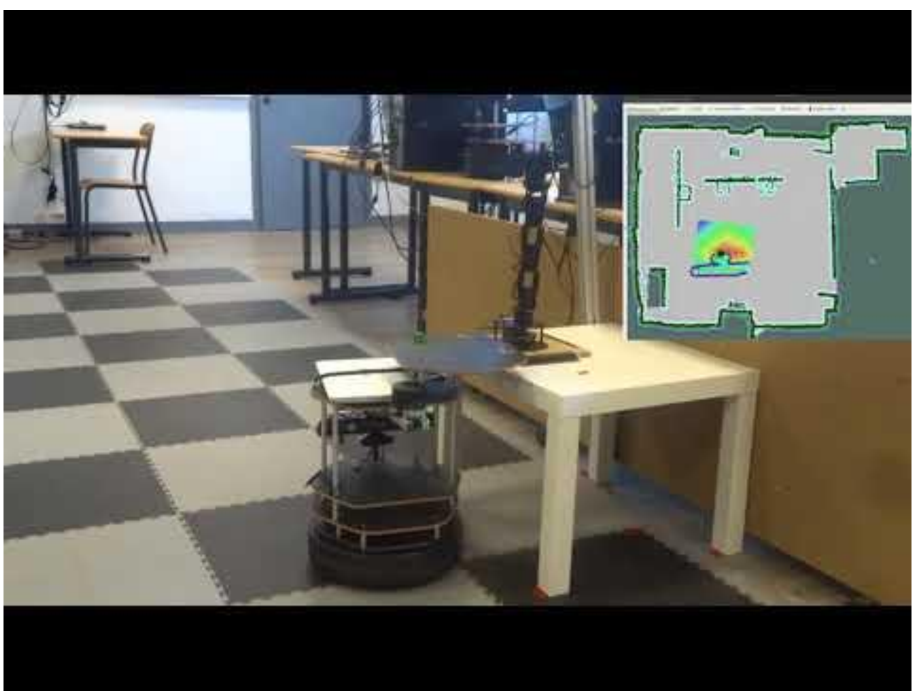
To launch the turtlebot with its current position on the map
$ roslaunch turtlebot_nav RP_nav_turt2.launch

On the work station:

Third pose command to prepare the turtlebot for its destination on the map
$ roslaunch turtlebot_pose_commands tb_return2.launch

Fourth pose command (takes the turtlebot to its destination on the map)
$ roslaunch turtlebot_pose_commands tb_return_pose.launch.

Video



Youtube video of complete the scenario of the project

VI. Bugs

Running the PhantomX pincher arm

There seems to be no package that was more efficient than the turtlebot_arm package which is supposed to run the robotic arm and the turtlebot2 attached together. A modified version of that package was used for the project, which was cloned from the repository https://github.com/NathanCrombez/turtlebot_arm which allows us to able to run the robotic arm independent of the turtlebot successfully, however, with some warnings on the terminal. Those warnings prevent the turtlebot for further operation like giving it a new goal pose command. This was the reason we must kill all operation on the terminal and launch back the *RP_nav_turt2.launch* before giving the turtlebot its consequent goal pose command as indicated above in the project execution.

Planning of the Turtlebot Trajectory

With the turtlebot_navigation package it was quite difficult to give certain goal commands to the turtlebot depending on its initial orientation. The turtlebot would take more time to plan its trajectory to a target goal position if its initial orientation were about 180 degrees and so for a more effective planning of the turtlebot trajectory, it was required that we give the turtlebot goal position commands relative to its initial orientation. This was the reason we had to split the goal position command in the project scenario from two to four different positions for a faster and more effective execution of the project. This problem is because of the limitation in tuning the parameters of the **AMCL** package which was really complicated since we had to tune different parameters yet the function of one parameter always seem contradictory to another, making the operation of the robot erratic.

Problem with Obstacle Avoidance

This also had a lot to do with the parameter settings of the local and global costmap parameters of the **AMCL** package. This involves the turtlebot perceiving the table at the position of the robot arm as an obstacle and so finding it difficult to move closer to the table. By adjusting the tolerance of the distance between the turtlebot and the obstacle will result in the turtlebot getting too close to the obstacle in other cases and as a result colliding with the obstacle while trying to avoid it. We were able to resolve this to minimal by allowing the turtlebot reach the position from its side by taking its trajectory off the direction of the table and providing an extension to the turtlebot that makes it able to reach the robot arm at a reasonable distance from the table

VII. CONCLUSION

The project described the design and implementation of a turtlebot cooperative task with a robotic arm. We were able to build the map, integrate the Rplidar laser sensor and test the avoidance obstacle on the turtlebot as well as the pick and place with the PhantomX pincher arm. In a nutshell, we can say that the overall scenario was executed successfully. Further interest will be on how to remove the bugs and finding an adequate package that could run the pincher arm separately. As future work, we will be glad to implement, at latter stage, the ***ROS Smach*** to run the sequence of the operations.

VIII. REFERENCES

- *R. Patrick Goebel, ROS by Example: A do-it-yourself guide to the robot Operating System, pi robot production,jan 2015, vol1.*
- *R. Patrick Goebel, ROS by Example: A do-it-yourself guide to the robot Operating System, pi robot production,jan 2015, vol2.*
- http://wiki.ros.org/
- https://www.turtlebot.com/turtlebot2/
- https://github.com/roboticslab-fr/rplidar-turtlebot2
- https://github.com/roboticslab-fr/rplidar-turtlebot2
- https://github.com/roboticslab-fr/rplidar-turtlebot2
- *https://edu.gaitech.hk/turtlebot/turtlebot-tutorials.html*
- *http://learn.turtlebot.com/*

Project repositories:

- https://github.com/Macaulay123/BSCVRoboticsProject
- https://github.com/nsadiasluk/Robotics-Engineering-Project.git
- https://github.com/ebenezer11/BSCVRoboticsProject

IX. APPENDIX

RP_nav_turt.launch

```
<launch>
        <!--Launch the LaserScan minimal-->
    <include file="$(find turtlebot_le2i)/launch/remap_rplidar_minimal.launch" />

        <!--Launch the AMCL with the initial position on the Map -->
    <include file="$(find turtlebot_navigation)/launch/amcl_demo.launch" >
     <arg name="map_file" value="/home/turtlebot/ros/indigo/catkin_ws/Bags/ClassMap.yaml" />
     <arg name="initial_pose_x" default="9.541"/>
     <arg name="initial_pose_y" default="-6.338"/>
     <arg name="initial_pose_a" default="1.803"/>
    </include>

</launch>
```

RP_nav_turt2.launch

```
<launch>
       <!--Launch the LaserScan Minimal launch-->
    <include file="$(find turtlebot_le2i)/launch/remap_rplidar_minimal.launch" />

              <!--Launch the AMCL with the Pincher arm as initial pose on the map-->
    <include file="$(find turtlebot_navigation)/launch/amcl_demo.launch" >
     <arg name="map_file" value="/home/turtlebot/ros/indigo/catkin_ws/Bags/ClassMap.yaml" />
        <arg name="initial_pose_x" default="2.729"/>
     <arg name="initial_pose_y" default="-0.058"/>
     <arg name="initial_pose_a" default="1.987"/>
    </include>


</launch>
```

arm_pick_and_place.launch

```
<launch>
        <!-- Launching the arm_bringup-->
  <include file="$(find turtlebot_arm_bringup)/launch/arm.launch" />

       <!-- Launching the moveit package-->
  <include file="$(find turtlebot_arm_moveit_config)/launch/turtlebot_arm_moveit.launch" />

       <!-- Launching the pick and place script-->
       <node pkg="turtlebot_arm_moveit_demos" name="arm_pick_and_place" type="pick_and_place "
/>
</launch>
```

tb_goal_pose.launch

```
<launch>
        <!--Here we give a second goal position command to the turtlebot -->
      <node name="goal_pose" pkg="rostopic" type="rostopic" args="pub /move_base_simple/goal
geometry_msgs/PoseStamped '{ header: {stamp: now, frame_id: 'map'}, pose: { position: {x: 3.288, y: -
1.763, z: 0.000}, orientation: {x: 0.000, y: 0.000, z: 0.756, w: 0.654}}}'"/>
</launch>
```

tb_goal_pose2.launch

```
<launch>
        <!--Here we give a second goal position command to the turtlebot -->
      <node name="goal_pose" pkg="rostopic" type="rostopic" args="pub /move_base_simple/goal
geometry_msgs/PoseStamped '{ header: {stamp: now, frame_id: 'map'}, pose: { position: {x: 2.729, y: -
0.058, z: 0.0}, orientation: {x: 0.000, y: 0.000, z: 0.657, w: 0.754}}}'"/>
</launch>
```

tb_return_pose.launch

```
<launch>
        <!--Here we give a final return position command to the turtlebot -->
      <node name="goal_pose" pkg="rostopic" type="rostopic" args="pub /move_base_simple/goal
geometry_msgs/PoseStamped '{ header: {stamp: now, frame_id: 'map'}, pose: { position: {x: 9.541, y: -
6.338, z: 0.0}, orientation: {x: 0.000, y: 0.000, z: 0.784, w: 0.620}}}'"/>
</launch>
```

tb_return2.launch

```
<launch>
        <!--Here we give a destination return position command to the turtlebot -->
      <node name="goal_pose" pkg="rostopic" type="rostopic" args="pub /move_base_simple/goal
geometry_msgs/PoseStamped '{ header: {stamp: now, frame_id: 'map'}, pose: { position: {x: 4.088, y:
1.134, z: 0.0}, orientation: {x: 0.000, y: 0.000, z: -0.106, w: 0.994}}}'"/>

</launch>
```