

ROS Noetic Specification and using ROS to make a hypothetical robot

Fatema Mirza

Intelligent System and Robotics

Student ID: 77264327



LEEDS
BECKETT
UNIVERSITY

ROS Noetic Specification and using ROS to make a hypothetical robot

by

Fatema Mirza

Student ID: 77264327
Project duration: February 1, 2021 – May 10, 2021
Thesis committee: Dr. Abiodun Yusuf, Leeds Beckett University, Supervisor
Prof. David Love, Leeds Beckett University

This report is confidential and cannot be made public until May 10, 2021.

An electronic version of this report will be available at <https://github.com/fatimamirza94/>.



Abstract

This report will provide an explanation on how ROS Noetic works and the different components involved with ROS Noetic. Furthermore, a hypothetical robot named roomba_robot will be developed with full documentation on the source code as well as how to install and run it.

The documented source code is available at: https://github.com/fatimamirza94/roomba_robot.git

*Fatema Mirza
Leeds, May 2021*

Contents

1	Delving into the ROS Environment	1
1.1	Installing ROS Noetic.	1
1.1.1	Catkin Workspace	2
1.2	Creating ROS Package	2
1.3	Publishers and Subscribers	3
1.4	ROS Parameter Server	3
1.5	ROS Launch Files	4
1.5.1	ROS Bag files.	5
1.6	ROS Services.	6
1.7	ROS Action	8
2	Roomba_Robot Overview	11
2.1	Roomba_Robot Description & Functionalities.	11
2.2	Roomba_Robot Nodes, Services & Action	11
2.3	Roomba_Robot Topics	12
2.4	Roomba_Robot Installation & Running	12
3	Full Code Specification	18
3.1	Action files	18
3.2	CMakeLists file	18
3.3	Launch file	18
3.4	Message files	19
3.5	package.xml file	20
3.6	Nodes	20
3.6.1	control_node.py.	20
3.6.2	dirt_pub_node.py.	27
3.6.3	position_pub_node.py	28
3.6.4	speed_pub_node.py	29
3.6.5	turn_camera_client.py	30
3.6.6	turn_camera_service.py	32
3.6.7	walk_to_point_action_server_node.py	33
3.7	srv files	37

List of Figures

1.1	Roscore command	2
1.2	List of Current Nodes in the ROS project for roomba_robot	3
1.3	A node publishing and subscribing to events	3
1.4	List of ROS parameters	4
1.5	List of ROS parameters	4
1.6	Using roslaunch	5
1.7	Using rosbag to record information	5
1.8	Using rosbag to playback recorded data	6
1.9	Creating an srv file	7
1.10	CMakeList File to reflect the changes for services	7
1.11	Checking that the rosservice has been included in the project	8
1.12	Action server and client at work	8
1.13	Action file	9
1.14	CMakeList File to reflect the changes for actions	10
1.15	Checking to ensure that services has been added successfully	10
2.1	Successful catkin_make	13
2.2	Successful roslaunch	14
2.3	Successful rosrun	15
2.4	Validating input example	15
2.5	Successful service server running	16
2.6	Successful service client running	17
3.1	WalkToPoint.action code	18
3.2	roomba_robot.launch code	19
3.3	PositionAndDestination.msg code	19
3.4	RobotPosition.msg code	20
3.5	control_node.py code-1	21
3.6	control_node.py code-2	22
3.7	control_node.py code-3	23
3.8	control_node.py code-4	24
3.9	control_node.py code-5	25
3.10	control_node.py code-6	26
3.11	control_node.py code-7	27
3.12	dirt_pub_node.py code	28
3.13	position_pub_node.py code-1	29
3.14	position_pub_node.py code-2	29
3.15	speed_pub_node.py code-1	30
3.16	speed_pub_node.py code-2	30
3.17	turn_camera_client.py code-1	31
3.18	turn_camera_client.py code-2	31
3.19	turn_camera_service.py code-1	32
3.20	turn_camera_service.py code-2	33
3.21	turn_camera_service.py code-3	33
3.22	walk_to_point_action_server_node.py code-1	34
3.23	walk_to_point_action_server_node.py code-2	34
3.24	walk_to_point_action_server_node.py code-3	35
3.25	walk_to_point_action_server_node.py code-4	36
3.26	walk_to_point_action_server_node.py code-5	36

3.27	walk_to_point_action_server_node.py code-6	37
3.28	TurnCamera.srv code	37

Delving into the ROS Environment

This report will look into the ROS environment and using the knowledge gained in through that, will create a hypothetical roomba robot that is capable of sensing dirt, travelling to a position defined by the user as well as travel back to the charging station. For highly valued customers or users, the roomba robot will also allow viewing of pictures taken during its snapshot while traversing.

1.1. Installing ROS Noetic

ROS Noetic was used as the operating system. This was installed on a virtual machine of the developers computer. To install ROS-Noetic on the virtual machine the following commands were used.

Setting up the sources list

```
sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc) main" > /etc/apt/sources.list.d/ros-latest.list'
```

Setting up the keys

```
sudo apt-key adv --keyserver 'hkp://keyserver.ubuntu.com:80' --recv-key C1CF6E31E6BADE8868B172B4F42ED6FBAB17C65
```

File installation using sudo

```
sudo apt update  
sudo apt install ros-noetic-desktop-full
```

Running the installation using a bash script

```
source /opt/ros/noetic/setup.bash  
echo "source /opt/ros/noetic/setup.bash" » ~/.bashrc source ~/.bashrc
```

Installing dependencies

```
sudo apt install python3-rosdep python3-rosinstall python3-rosinstall-generator python3-wstool build-essential  
sudo apt installForis python3-rosdep  
sudo rosdep init  
rosdep update
```

Checking if ros has been installed correctly

To ensure whether ROS has been installed correctly, run roscore in the terminal to initiate the ROS Master which is the backbone responsible for communication between nodes where only one instance of it can run at a time. The command will return the following:

```

fatima@fatima-VirtualBox:~$ roscore
... logging to /home/fatima/.ros/log/110aadcc-ab90-11eb-814a-2f56a7470fc6/roslaunch-fatima-VirtualBox-27868.log
Checking log directory for disk usage. This may take a while.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://fatima-VirtualBox:33345/
ros_comm version 1.15.9

SUMMARY
=====
PARAMETERS
  * /rosdistro: noetic
  * /rosversion: 1.15.9

NODES

auto-starting new master
process[master]: started with pid [27876]
ROS_MASTER_URI=http://fatima-VirtualBox:11311/

setting /run_id to 110aadcc-ab90-11eb-814a-2f56a7470fc6
process[rosout-1]: started with pid [27886]
started core service [/rosout]

```

Figure 1.1: Roscore command

To download any additional ros-noetic packages

To download any additional ros-noetic packages, use the following command:
`sudo apt install ros-noetic-usb-cam`

1.1.1. Catkin Workspace

Catkin workspace is a directory to ensure smooth build, running and installation of the ROS projects. Run the following commands in the terminal at the level of home directory to create the catkin workspace.
`mkdir -p /catkin_ws/src`
`cd /catkin_ws/`
`mkdir src`
`catkin_make`

The command `catkin_make` will create the CMakeLists in the src directory. Additional 'build' and 'devel' folder will be created and inside the 'devel' folder it can be seen that there are now several setup.*sh files (ROS.org, 2021c).

1.2. Creating ROS Package

To create a ROS package, enter the terminal at the level of the src folder of the catkin_ws. Enter the following command in the terminal with dependencies to create the package roomba_robot .

`catkin_create_pkg roomba_robot roscpp rospy std_msgs`

A node is an executable file within a ROS package. ROS nodes use a ROS client library to communicate with other nodes to publish or subscribe to a Topic or provide or use a Service (ROS.org, 2021f). To ensure a proper structure of the ROS project, the nodes will be placed in a scripts directory (created by the developer) in the roomba_robot folder.

```
fatima@fatima-VirtualBox:~$ rosnode list
/dirt_pub_node
/position_pub_node
/rosout
/speed_pub_node
/walk_to_point_action_server_node
fatima@fatima-VirtualBox:~$
```

Figure 1.2: List of Current Nodes in the ROS project for roomba_robot

1.3. Publishers and Subscribers

As mentioned in the previous section, nodes can be publishers or subscribers. Publishers send messages with a specific topic and subscribers listen for that particular topic in the stream. For the subscriber to listen to the topic, it first has to subscribe to the topic using the call back functions.

```
fatima@fatima-VirtualBox:~$ rosnode info walk_to_point_action_server_node
-----
Node [/walk_to_point_action_server_node]
Publications:
 * /change_position [roomba_robot/RobotPosition]
 * /change_speed [std_msgs/Float32]
 * /rosout [rosgraph_msgs/Log]
 * /walk_to_point/feedback [roomba_robot/WalkToPointActionFeedback]
 * /walk_to_point/result [roomba_robot/WalkToPointActionResult]
 * /walk_to_point/status [actionlib_msgs/GoalStatusArray]

Subscriptions:
 * /walk_to_point/cancel [unknown type]
 * /walk_to_point/goal [unknown type]

Services:
 * /walk_to_point_action_server_node/get_loggers
 * /walk_to_point_action_server_node/set_logger_level

contacting node http://fatima-VirtualBox:46573/ ...
Pid: 30091
Connections:
 * topic: /rosout
   * to: /rosout
   * direction: outbound (39785 - 127.0.0.1:45192) [18]
   * transport: TCPROS
 * topic: /change_speed
   * to: /speed_pub_node
   * direction: outbound (39785 - 127.0.0.1:45182) [11]
   * transport: TCPROS
 * topic: /change_position
   * to: /position_pub_node
   * direction: outbound (39785 - 127.0.0.1:45170) [10]
   * transport: TCPROS
```

Figure 1.3: A node publishing and subscribing to events

1.4. ROS Parameter Server

A parameter server is a shared, multi-variate dictionary that is accessible via network APIs used by nodes to store and retrieve parameters at runtime. This means that parameter lists can be changed during runtime, which is useful for testing and observing the impact of those changes real time.

```
fatima@fatima-VirtualBox:~$ rosparam list
/initial_post_x
/initial_post_y
/max_cordinate
/min_cordinate
/robot_max_speed
/robot_name
/rosdistro
/roslaunch/uris/host_fatima_virtualbox__41297
/rosversion
/run_id
```

Figure 1.4: List of ROS parameters

ROS parameters can be changed and retrieved using the get set method. An example is shown below where the value was initially 0, and is now set to 2.

```
fatima@fatima-VirtualBox:~$ rosparam get /initial_post_x
0
fatima@fatima-VirtualBox:~$ rosparam set /initial_post_x 2
fatima@fatima-VirtualBox:~$ rosparam get /initial_post_x
2
```

Figure 1.5: List of ROS parameters

Alternatively, parameter server variables can be saved in a yaml file to access and change the variables from that file using the following command:
rosparam dump param.yaml

1.5. ROS Launch Files

roslaunch - which already includes the roscore - is a tool for launching multiple ROS nodes locally and remotely via SSH and set parameters on the Parameter Server (ROS.org, 2021d). roslaunch limits the number of terminals needed for the codes - as it can be done in one go rather than requiring to open a new terminal for each node. For roslaunch to work successfully, the package must be sourced through the command typed in the terminal:

```
source devel/setup.bash
```

```

fatima@fatima-VirtualBox:~/catkin_ws$ source devel/setup.bash
fatima@fatima-VirtualBox:~/catkin_ws$ roslaunch roomba_robot roomba_robot.launch
... logging to /home/fatima/.ros/log/fb0f5070-abff-11eb-814a-2f56a7470fc6/roslaun...
Checking log directory for disk usage. This may take a while.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://fatima-VirtualBox:41297/

SUMMARY
=====
PARAMETERS
  * /initial_post_x: 0.0
  * /initial_post_y: 0.0
  * /max_cordinate: 100.0
  * /min_cordinate: -100.0
  * /robot_max_speed: 0.5
  * /robot_name: roomba_robot
  * /rosdistro: noetic
  * /rosversion: 1.15.9

NODES
/
  dirt_pub_node (roomba_robot/dirt_pub_node.py)
  position_pub_node (roomba_robot/position_pub_node.py)
  speed_pub_node (roomba_robot/speed_pub_node.py)
  walk_to_point_action_server_node (roomba_robot/walk_to_point_action_server.py)

auto-starting new master
process[master]: started with pid [30071]
ROS_MASTER_URI=http://localhost:11311

setting /run_id to fb0f5070-abff-11eb-814a-2f56a7470fc6
process[rosout-1]: started with pid [30081]
started core service [/rosout]
process[dirt_pub_node-2]: started with pid [30088]
process[speed_pub_node-3]: started with pid [30089]

```

Figure 1.6: Using roslaunch

1.5.1. ROS Bag files

A bag is a file format in ROS for storing ROS message data which can subscribe to one or more ROS topics, and store the serialized message data in a file as it is received and playback at a later time (ROS.org, 2021b).

```

fatima@fatima-VirtualBox:~$ rosbag record -a -o test.bag
[ INFO] [1620042004.871548753]: Subscribing to /rosout_agg
[ INFO] [1620042004.879317053]: Subscribing to /rosout
[ INFO] [1620042004.879483686]: Recording to 'test_2021-05-03-13-40-04.bag'.
[ WARN] [1620042004.879567097]: Less than 5 x 1G of space free on disk with 'test_2021-05-03-13-40-04.bag.active'.
[ INFO] [1620042004.886500974]: Subscribing to /dirt
[ INFO] [1620042004.894942250]: Subscribing to /position
[ INFO] [1620042004.905506203]: Subscribing to /change_speed
[ INFO] [1620042004.912621644]: Subscribing to /change_position
[ INFO] [1620042004.919841065]: Subscribing to /walk_to_point/status
[ INFO] [1620042004.922893978]: Subscribing to /walk_to_point/result
[ INFO] [1620042004.925496217]: Subscribing to /walk_to_point/feedback
[ INFO] [1620042004.928089893]: Subscribing to /speed

```

Figure 1.7: Using rosbag to record information

```
fatima@fatima-VirtualBox:~$ rosbag play test.bag
[ INFO] [1620042543.228833356]: Opening test.bag

Waiting 0.2 seconds after advertising topics... done.

Hit space to toggle paused, or 's' to step.
[RUNNING] Bag Time: 1614798199.919378 Duration: 10.081344 / 61.000444
[RUNNING] Bag Time: 1614798199.922469 Duration: 10.084436 / 61.000444
[RUNNING] Bag Time: 1614798200.023503 Duration: 10.185469 / 61.000444
[RUNNING] Bag Time: 1614798200.120236 Duration: 10.282203 / 61.000444
[RUNNING] Bag Time: 1614798200.124647 Duration: 10.286614 / 61.000444
[RUNNING] Bag Time: 1614798200.224796 Duration: 10.386763 / 61.000444
[RUNNING] Bag Time: 1614798200.322380 Duration: 10.484347 / 61.000444
[RUNNING] Bag Time: 1614798200.324592 Duration: 10.486558 / 61.000444
[RUNNING] Bag Time: 1614798200.424961 Duration: 10.586927 / 61.000444
[RUNNING] Bag Time: 1614798200.522283 Duration: 10.684250 / 61.000444
[RUNNING] Bag Time: 1614798200.525929 Duration: 10.687895 / 61.000444
[RUNNING] Bag Time: 1614798200.626778 Duration: 10.788745 / 61.000444
[RUNNING] Bag Time: 1614798200.719633 Duration: 10.881600 / 61.000444
```

Figure 1.8: Using rosbag to playback recorded data

1.6. ROS Services

The publish / subscribe model is useful communication model, but it is not suitable for RPC request / reply interactions. Request / reply is done via a Service, which is defined by a pair of custom messages built as per the requirement of the project: one for the request and one for the reply. A providing ROS node offers a service under a string name, and a client calls the service by sending the request message and awaiting the reply. Services are defined using .srv files, which are compiled into source code by a ROS client library (ROS.org, 2021e). The .srv file will contain a request, and the corresponding response. In the case of this roomba, turn_degrees is a angle requested for the file name, whereas the image is the response message. A directory named srv is created and the following dependencies are added in the package.xml file.

```
<build_depend>message_generation</build_depend>
<exec_depend>message_runtime</exec_depend>
```

A .srv file is created and the service and dependencies are added in the CMakeList file.

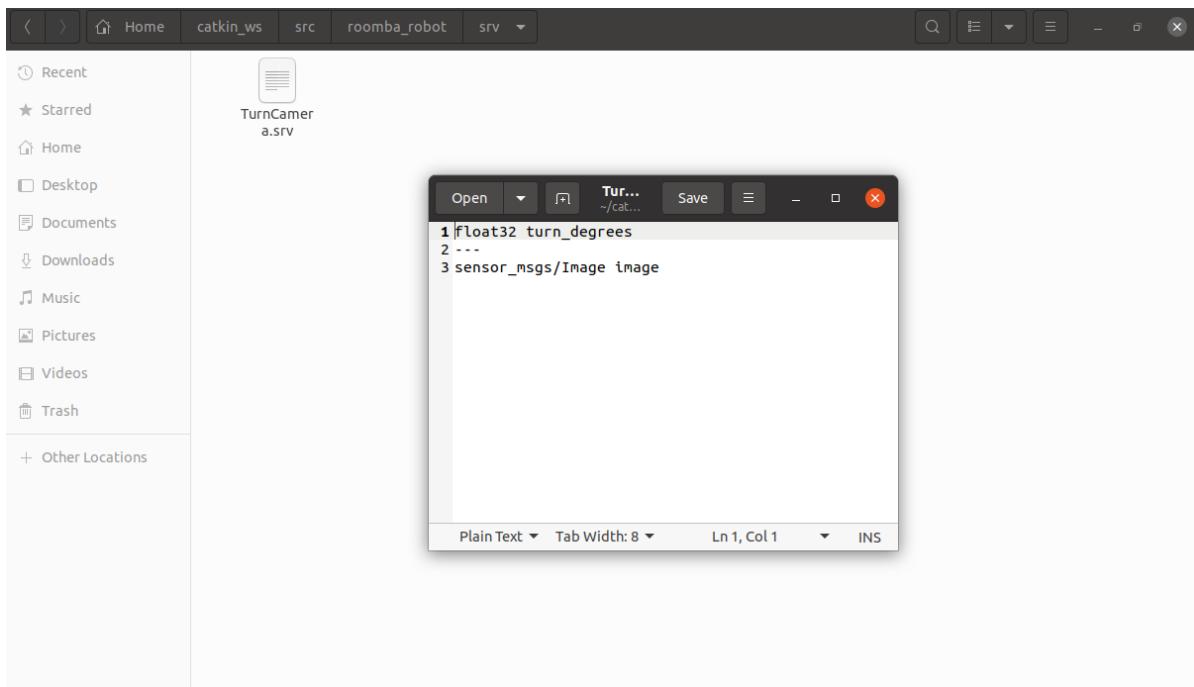


Figure 1.9: Creating an srv file

```
--  
63 ## Generate services in the 'srv' folder  
64 add_service_files(  
65   FILES  
66   TurnCamera.srv  
67 )  
68  
69 ## Generate actions in the 'action' folder  
70 add_action_files(  
71   FILES  
72   WalkToPoint.action  
73 )  
74  
75 ## Generate added messages and services with any dependencies listed here  
76 generate_messages(  
77   DEPENDENCIES  
78   std_msgs  
79   actionlib_msgs  
80   sensor_msgs  
81   geometry_msgs  
82 )  
--
```

Figure 1.10: CMakeList File to reflect the changes for services

```
fatima@fatima-VirtualBox:~/catkin_ws$ rosservice list
/control_node/get_loggers
/control_node/set_logger_level
/dirt_pub_node/get_loggers
/dirt_pub_node/set_logger_level
/position_pub_node/get_loggers
/position_pub_node/set_logger_level
/rosout/get_loggers
/rosout/set_logger_level
/speed_pub_node/get_loggers
/speed_pub_node/set_logger_level
/turn_camera
/turn_camera_service_node/get_loggers
/turn_camera_service_node/set_logger_level
/walk_to_point_action_server_node/get_loggers
/walk_to_point_action_server_node/set_logger_level
```

Figure 1.11: Checking that the rosservice has been included in the project

1.7. ROS Action

ROS Action refers to a standardized platform for interfacing with preemptable tasks such as moving the roomba to a particular place or returning to base ROS.org, 2021a. From the original documentation, the following image is copied to best describe the process of ROS Action.

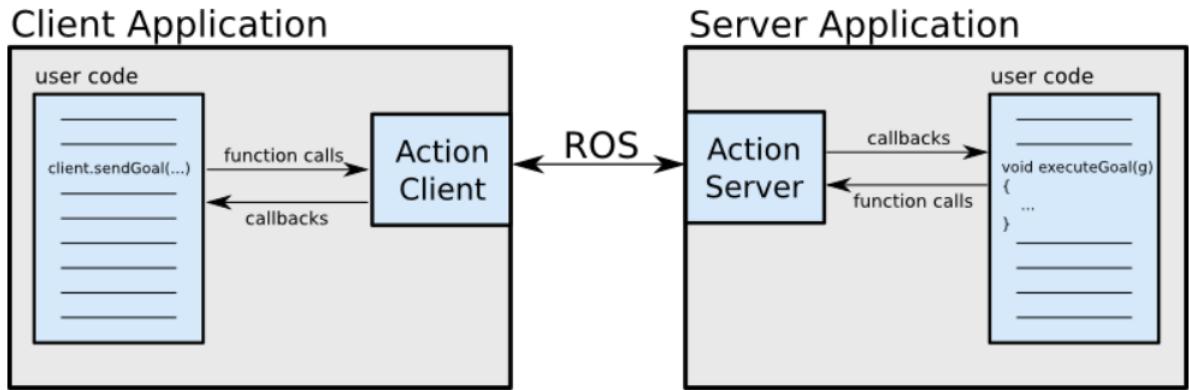


Figure 1.12: Action server and client at work

For the client and server to communicate, custom messages are defined - action specification. This defines the Goal, Feedback, and Result messages with which clients and servers communicate in the .action file:

Goal Goal aims to complete tasks using actions, the goal is a message that is sent to an ActionServer by an ActionClient and in the case, it is the destination to reach by the roomba.

Feedback Feedback returns the incremental progress of the goal to the ActionClient- that is how much distance has been covered till now to reach the destination.

Result A result is generated from the ActionServer to inform the ActionClient that the goal has been completed - that is how long the roomba took to traverse from the previous current position to the specified final destination.

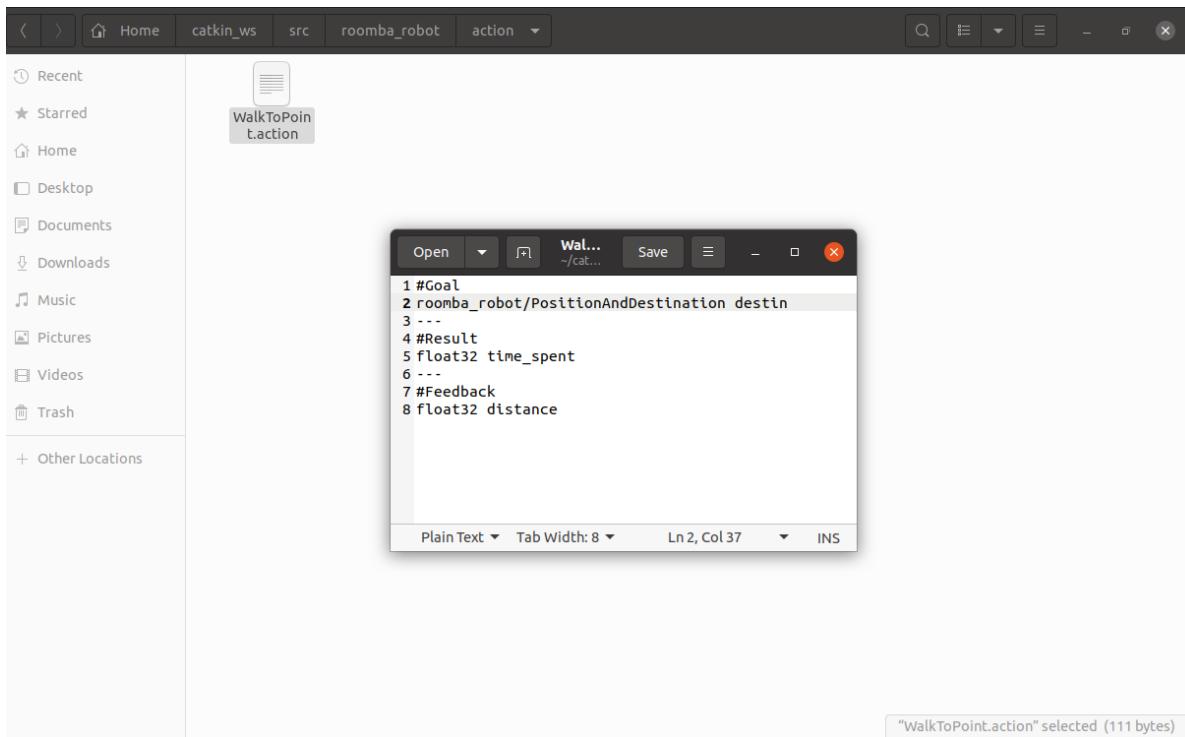


Figure 1.13: Action file

To initiate the messages of action, add the following dependencies in the package.xml similar to what was done for the service files. file.

```
<build_depend>actionlib</build_depend>
<build_depend>actionlib_msgs</build_depend>
<exec_depend>actionlib</exec_depend>
<exec_depend>actionlib_msgs</exec_depend>
```

Generate the action file dependencies in the CMakeLists.

```
--  
63 ## Generate services in the 'srv' folder  
64 add_service_files(  
65   FILES  
66   TurnCamera.srv  
67 )  
68  
69 ## Generate actions in the 'action' folder  
70 add_action_files(  
71   FILES  
72   WalkToPoint.action  
73 )  
74  
75 ## Generate added messages and services with any dependencies listed here  
76 generate_messages(  
77   DEPENDENCIES  
78   std_msgs  
79   actionlib_msgs  
80   sensor_msgs  
81   geometry_msgs  
82 )  
-- .
```

Figure 1.14: CMakeList File to reflect the changes for actions

Ensure that the service has been correctly defined, check whether the goal, feedback and results have been added to the rostopic.

```
fatima@fatima-VirtualBox:~$ rostopic list  
/change_position  
/change_speed  
/dirt  
/position  
/rosout  
/rosout_agg  
/speed  
/walk_to_point/cancel  
/walk_to_point/feedback  
/walk_to_point/goal  
/walk_to_point/result  
/walk_to_point/status
```

Figure 1.15: Checking to ensure that services has been added successfully

2

Roomba_Robot Overview

The previous chapter introduced all the terminologies related to ROS programming and provided a description for them. Utilizing those core ideologies a new robot named Roomba_Robot has been developed which has been briefly touched upon in the previous chapter.

2.1. Roomba_Robot Description & Functionalities

Roomba_Robot is a robot which has a hypothetical dirt sensor that can detect the amount of dirt at the current position and inform the user about the dirt amount as well as where it is situation in a 2D plane at the moment. It is assumed that if the sensor has detected dirt it picks it up. Furthermore, the user can direct the robot to walk to a particular position (specified in a 2D coordinates, as for now the roomba can only traverse on the ground and has no concept of height) and also direct it to walk up to the charging base. Furthermore, an additional feature is present for special customers - that is those users can view the pictures taken by the roomba (it is assumed that the roomba has a hypothetical pictures using a hypothetical camera and stores it in the images folder). It has to be noted for user discrepancy that the camera is not really an actual camera and is just assumed to be present and has not been coded. However, the portion where the user can view the pictures stored in the roomba has been coded.

To simplify the functionalities the roomba can perform:

1. Detect dirt amount and report to the user.
2. Detect current position in a 2D plane and report to the user.
3. Traverse to position in a 2D plane and report to the user to the distance and time taken to reach there.
4. Return back to charging station and report to the user to the distance and time taken to reach there.
5. Can retrieve images stored in the directory based on the file specified by the user which is a secret functionality.

2.2. Roomba_Robot Nodes, Services & Action

The nodes are the files which are in the scripts folder.

1. control_node - a subscriber node that controls using user input from the terminal. It subscribes to dirt, position and speed to display to the user the dirt amount, the current position and the distance and time needed to traverse a requested location.
2. dirt_pub_node - only a publisher node that publishes the amount of dirt randomly.
3. position_pub_node - simultaneously a publisher and subscriber node that stores and deals with the position data. This node publishes the topic position and subscribes to the topic of change_position.

4. speed_pub_node - simultaneously a publisher and subscriber node that stores and deals with the speed data. This node publishes the topic speed and subscribes to the topic of change_speed.
5. turn_camera_client - this is a service node that assists the user in asking the roomba to retrieve which image by asking for the image name in the format of angles. This node sends this information to the server to request this image to be shown.
6. turn_camera_service - this is a service node that accepts the requests send forward by the client for viewing a particular image and responds by displaying the image on the terminal screen.
7. walk_to_point_action_server_node - is an action server nodes that assists the roomba in walking to a point specified by the user as well as returning to charging station. It operates by publishing change_position, change_speed, walk_to_point/feedback, walk_to_point/result and walk_to_point/status and subscribing to walk_to_point/cancel and walk_to_point/goal.

2.3. Roomba_Robot Topics

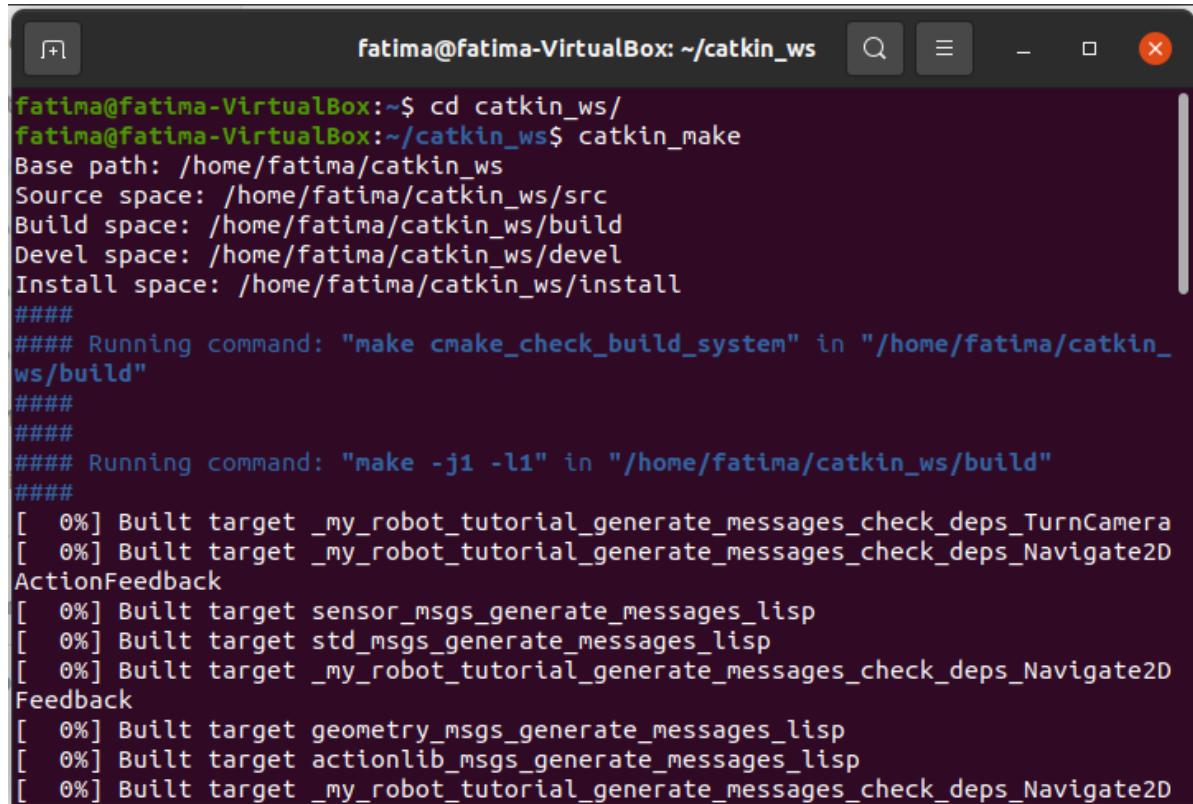
1. change_position
2. change_speed
3. dirt
4. position
5. speed
6. walk_to_point/cancel
7. walk_to_point/feedback
8. walk_to_point/goal
9. walk_to_point/result
10. walk_to_point/status

2.4. Roomba_Robot Installation & Running

1. Place the package roomba_robot in the src folder of catkin_ws.
2. Make the files inside the scripts folder of the roomba_robot executable by running the following command in the terminal: chmod +x *.py so that they can be launched by rosrun.
3. Check if the install_dependencies.sh file exists in the root folder of the package roomba_robot. If it does not exist, it has to be installed using the following commands typed in the terminal:
 chmod +x *.sh
 ./install_dependencies.sh

This will cause the virtual machine to restart - added as a note so that the user does not panic and think the program has crashed.

4. Source the project by using the following command at catkin_ws level:
 source devel/setup.bash
5. Build dependencies for the package at the same catkin_ws level by using following command:
 catkin_make



```
fatima@fatima-VirtualBox:~$ cd catkin_ws/
fatima@fatima-VirtualBox:~/catkin_ws$ catkin_make
Base path: /home/fatima/catkin_ws
Source space: /home/fatima/catkin_ws/src
Build space: /home/fatima/catkin_ws/build
Devel space: /home/fatima/catkin_ws/devel
Install space: /home/fatima/catkin_ws/install
#####
##### Running command: "make cmake_check_build_system" in "/home/fatima/catkin_ws/build"
#####
#####
##### Running command: "make -j1 -l1" in "/home/fatima/catkin_ws/build"
#####
[ 0%] Built target _my_robot_tutorial_generate_messages_check_deps_TurnCamera
[ 0%] Built target _my_robot_tutorial_generate_messages_check_deps_Navigate2D
ActionFeedback
[ 0%] Built target sensor_msgs_generate_messages_lisp
[ 0%] Built target std_msgs_generate_messages_lisp
[ 0%] Built target _my_robot_tutorial_generate_messages_check_deps_Navigate2D
Feedback
[ 0%] Built target geometry_msgs_generate_messages_lisp
[ 0%] Built target actionlib_msgs_generate_messages_lisp
[ 0%] Built target _my_robot_tutorial_generate_messages_check_deps_Navigate2D
```

Figure 2.1: Successful catkin_make

6. Once `catkin_make` has been successfully executed, at the same level `catkin_ws` source the project again using the following command:
`source devel/setup.bash`
7. Launch `roomba_robot` using the launch file named `roomba_robot.launch` by using the following command:
`roslaunch roomba_robot roomba_robot.launch`

```
fatima@fatima-VirtualBox:~/catkin_ws$ roslaunch roomba_robot roomba_robot.launch
... logging to /home/fatima/.ros/log/9ab9ce08-ac27-11eb-814a-2f56a7470fc6/roslaunch-fatima-VirtualBox-33645.log
Checking log directory for disk usage. This may take a while.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

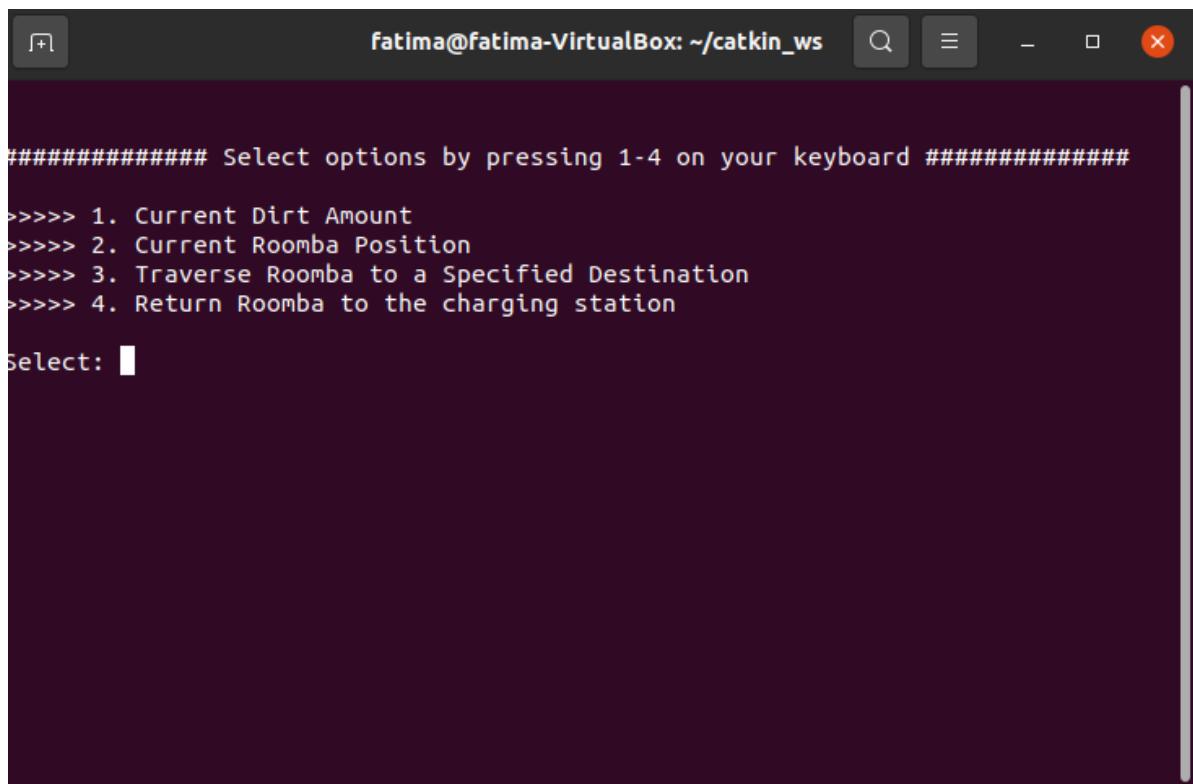
started roslaunch server http://fatima-VirtualBox:40167/

SUMMARY
=====

PARAMETERS
* /initial_post_x: 0.0
* /initial_post_y: 0.0
* /max_cordinate: 100.0
* /min_cordinate: -100.0
* /robot_max_speed: 0.5
* /robot_name: roomba_robot
* /rosdistro: noetic
* /rosversion: 1.15.9
```

Figure 2.2: Successful roslaunch

8. Once the project has successfully launched, open a new terminal at the level catkin_ws and source the project again using the following command:
source devel/setup.bash
9. To control the roomba_robot and view the menu of options, type in the following command: rosrun roomba_robot control_node.py
The user inputs have all been validated to ensure that wrong inputs are not accepted and the user will be prompted for the correct inputs.

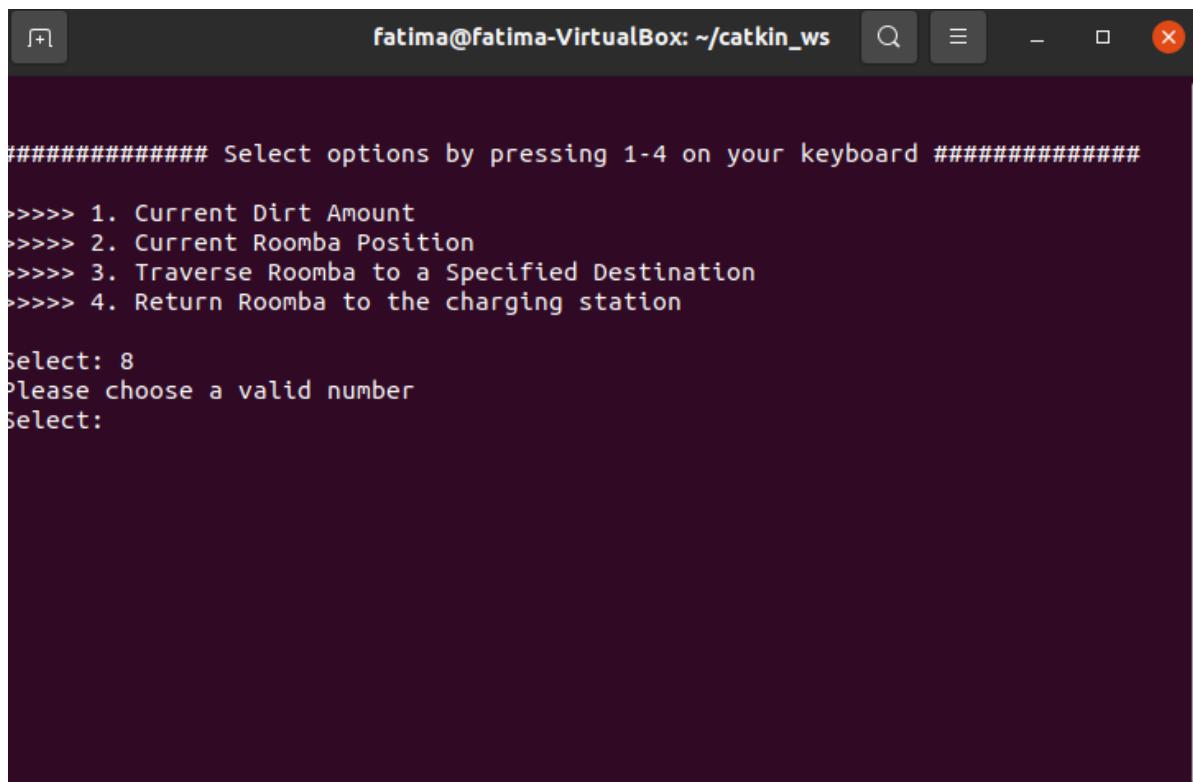


```
fatima@fatima-VirtualBox: ~/catkin_ws
```

```
##### Select options by pressing 1-4 on your keyboard #####
>>>> 1. Current Dirt Amount
>>>> 2. Current Roomba Position
>>>> 3. Traverse Roomba to a Specified Destination
>>>> 4. Return Roomba to the charging station

Select: 
```

Figure 2.3: Successful rosrun



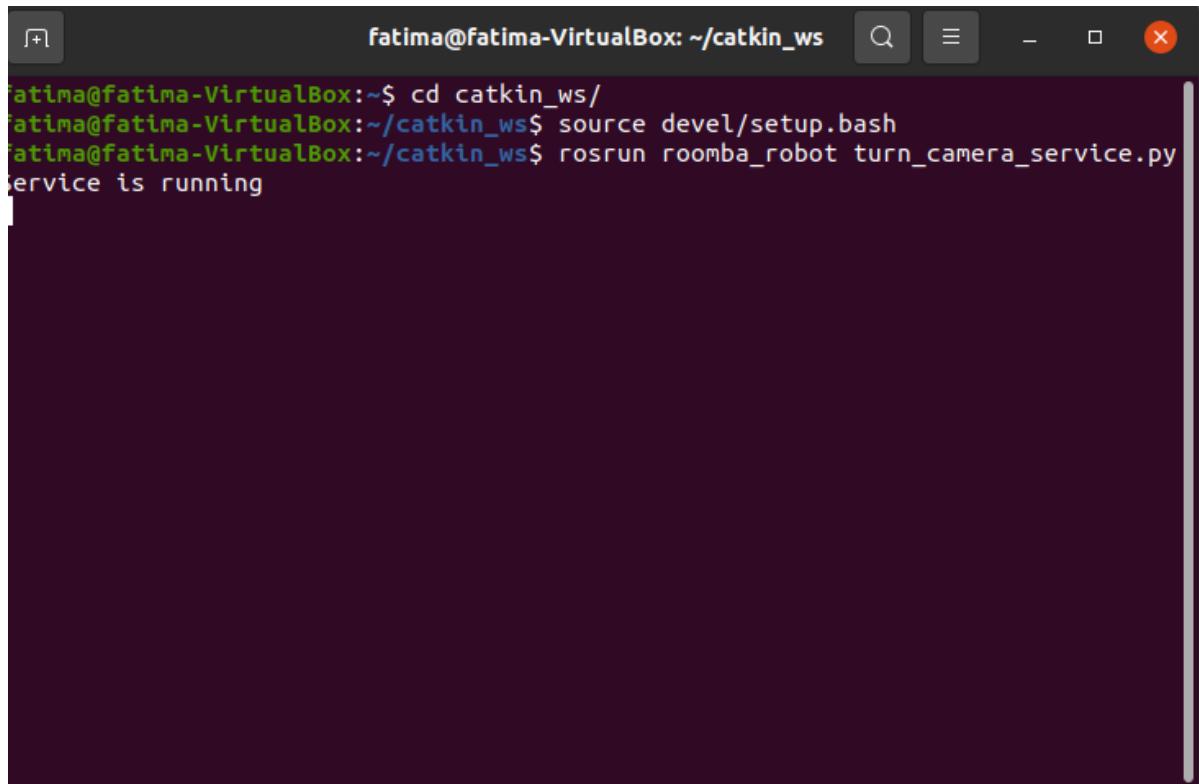
```
fatima@fatima-VirtualBox: ~/catkin_ws
```

```
##### Select options by pressing 1-4 on your keyboard #####
>>>> 1. Current Dirt Amount
>>>> 2. Current Roomba Position
>>>> 3. Traverse Roomba to a Specified Destination
>>>> 4. Return Roomba to the charging station

Select: 8
Please choose a valid number
Select: 
```

Figure 2.4: Validating input example

10. User can now play with the roomba as the user prefers.
11. To access the secret service for retrieving images stored in the roomba_robot, do the following:
 - (a) Open a new terminal at the level catkin_ws and source the project again using the following command:
source devel/setup.bash
 - (b) Run the Server service using the following command:
rosrun roomba_robot turn_camera_service.py

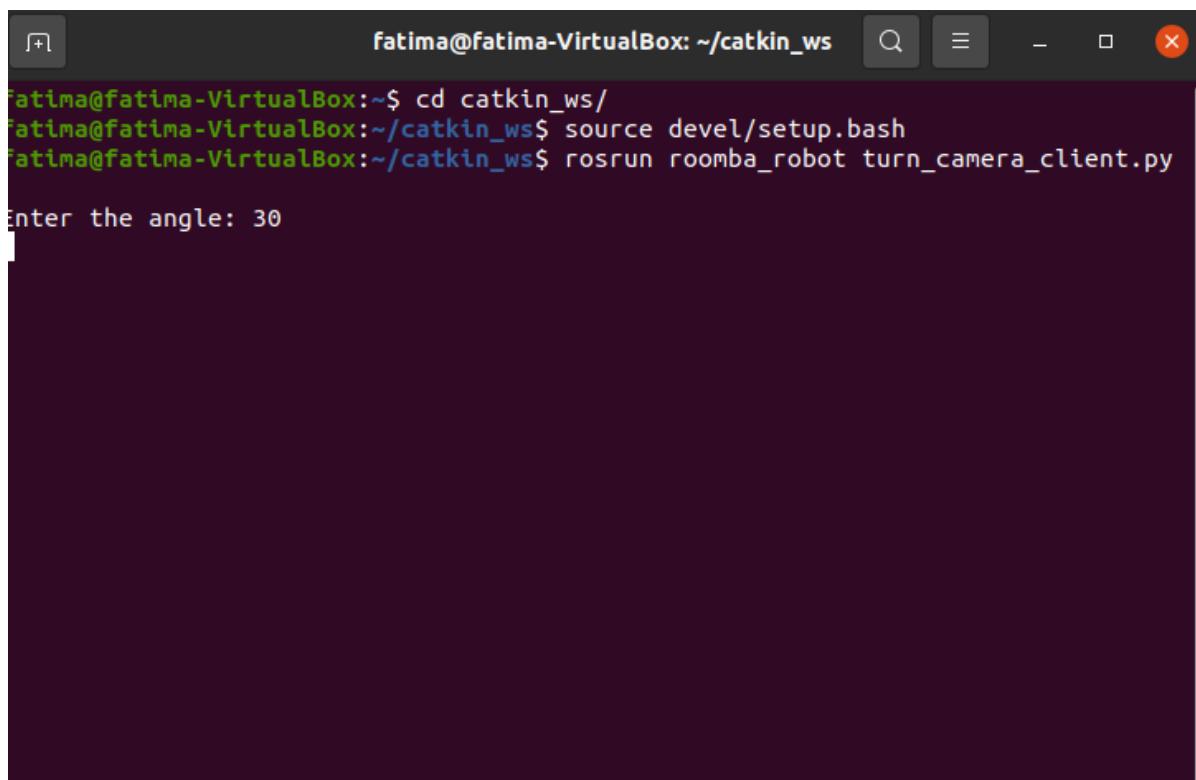


The screenshot shows a terminal window titled "fatima@fatima-VirtualBox: ~/catkin_ws". The terminal displays the following command-line session:

```
fatima@fatima-VirtualBox:~$ cd catkin_ws/
fatima@fatima-VirtualBox:~/catkin_ws$ source devel/setup.bash
fatima@fatima-VirtualBox:~/catkin_ws$ rosrun roomba_robot turn_camera_service.py
Service is running
```

Figure 2.5: Successful service server running

- (c) Open a new terminal at the level catkin_ws and source the project again using the following command:
source devel/setup.bash
- (d) Run the Client service using the following command:
rosrun roomba_robot turn_camera_client.py



The screenshot shows a terminal window titled "fatima@fatima-VirtualBox: ~/catkin_ws". The terminal contains the following command-line session:

```
fatima@fatima-VirtualBox:~$ cd catkin_ws/
fatima@fatima-VirtualBox:~/catkin_ws$ source devel/setup.bash
fatima@fatima-VirtualBox:~/catkin_ws$ rosrun roomba_robot turn_camera_client.py

Enter the angle: 30
```

Figure 2.6: Successful service client running

3

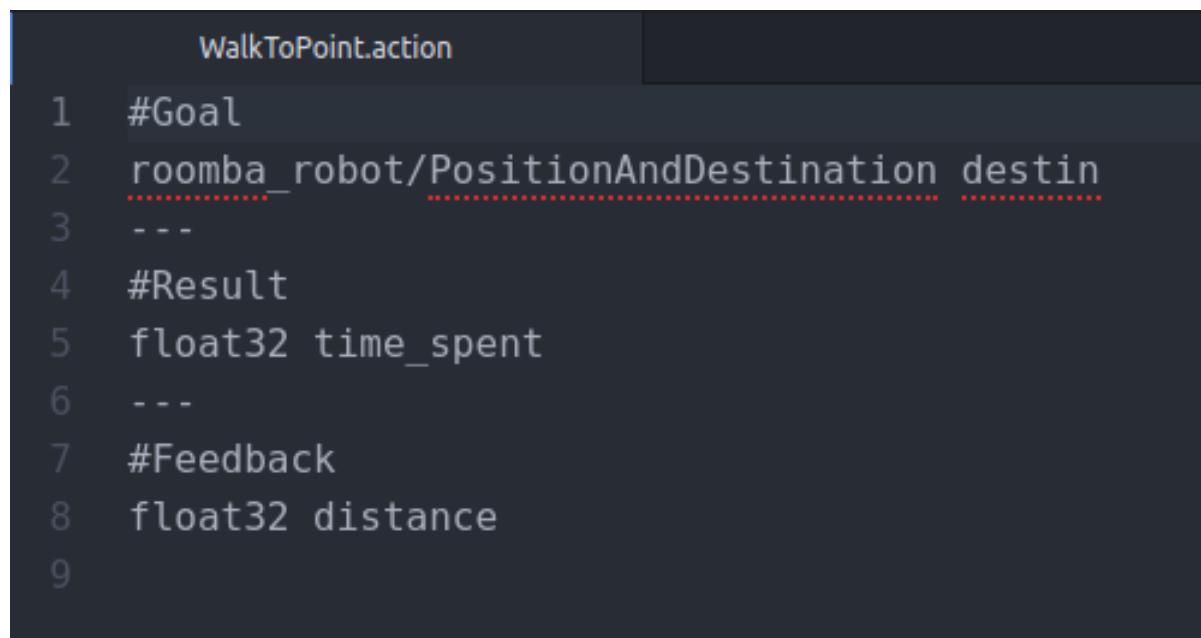
Full Code Specification

The codes have been arranged in alphabetical orders as they appear in the folders. The codes have been fully commented, documented and explained with reasoning for the logic behind it and is available at - https://github.com/fatimamirza94/roomba_robot.git.

Note: Check out the repository for a clearer explanation as the screenshots of the code might not be clear.

3.1. Action files

Action files are placed in the action folder and there is only 1 action file named: WalkToPoint.action



A screenshot of a code editor showing the content of the WalkToPoint.action file. The file is a ROS action definition. It starts with a header section containing '#Goal' and 'roomba_robot/PositionAndDestination' followed by a destination field. This is followed by a separator line '---', then a '#Result' section with a 'time_spent' field. Another separator line follows, then a '#Feedback' section with a 'distance' field. The file ends with a final separator line.

```
WalkToPoint.action
1 #Goal
2 roomba_robot/PositionAndDestination destin
3 ---
4 #Result
5 float32 time_spent
6 ---
7 #Feedback
8 float32 distance
9
```

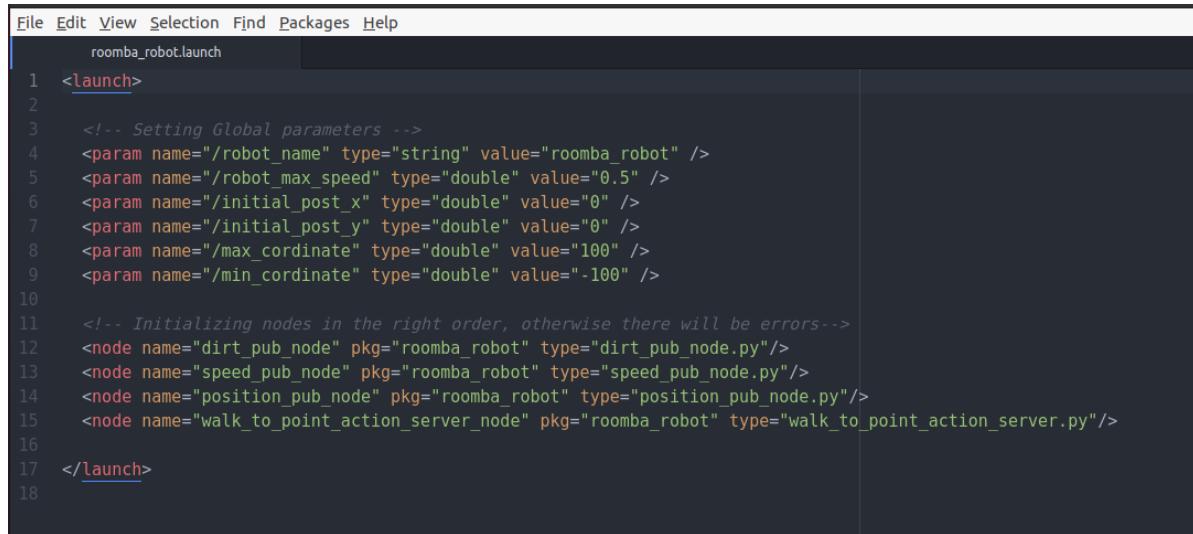
Figure 3.1: WalkToPoint.action code

3.2. CMakeLists file

Everything that has been added to the new file have already being discussed in chapter 1.

3.3. Launch file

Launch files are placed in the launch folder and there is only 1 launch file named: roomba_robot.launch



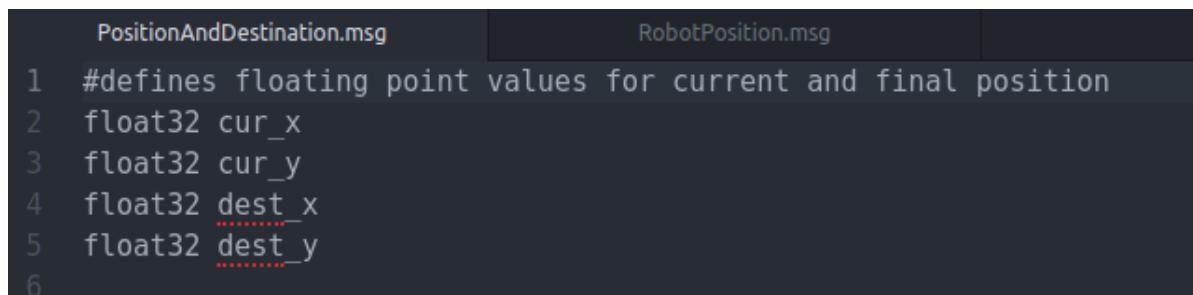
The screenshot shows a code editor window with the title bar "File Edit View Selection Find Packages Help". The file name in the title bar is "roomba_robot.launch". The code content is as follows:

```
1 <launch>
2
3   <!-- Setting Global parameters -->
4   <param name="/robot_name" type="string" value="roomba_robot" />
5   <param name="/robot_max_speed" type="double" value="0.5" />
6   <param name="/initial_pos_x" type="double" value="0" />
7   <param name="/initial_pos_y" type="double" value="0" />
8   <param name="/max_cordinate" type="double" value="100" />
9   <param name="/min_cordinate" type="double" value="-100" />
10
11  <!-- Initializing nodes in the right order, otherwise there will be errors-->
12  <node name="dirt_pub_node" pkg="roomba_robot" type="dirt_pub_node.py"/>
13  <node name="speed_pub_node" pkg="roomba_robot" type="speed_pub_node.py"/>
14  <node name="position_pub_node" pkg="roomba_robot" type="position_pub_node.py"/>
15  <node name="walk_to_point_action_server_node" pkg="roomba_robot" type="walk_to_point_action_server.py"/>
16
17 </launch>
18
```

Figure 3.2: roomba_robot.launch code

3.4. Message files

Message files are placed in the msg folder and there are 2 custom message files named: PositionAndDestination.msg and RobotPosition.msg



The screenshot shows a code editor window with two tabs: "PositionAndDestination.msg" and "RobotPosition.msg". The "PositionAndDestination.msg" tab is active and contains the following code:

```
1 #defines floating point values for current and final position
2 float32 cur_x
3 float32 cur_y
4 float32 dest_x
5 float32 dest_y
6
```

Figure 3.3: PositionAndDestination.msg code

```
PositionAndDestination.msg | RobotPosition.msg
1 #defines floating point for the position variables
2 float32 pos_x
3 float32 pos_y
4
```

Figure 3.4: RobotPosition.msg code

3.5. package.xml file

Everything that has been added to the new file have already being discussed in chapter 1.

3.6. Nodes

The node files are placed in the script files. There are many files in this and contain the most important aspects of the codes. Hence, each node's code will be displayed in the subsections.

3.6.1. control_node.py

```
control_node.py
7 import actionlib
8
9 #Float32 msg type is imported from standard messages
10 from std_msgs.msg import Float32
11
12
13 #msg types are needed to enable for Action client and
14 #subsequent Subscriber to position topic
15 #They are imported from the messages that have been created before by the author
16 from roomba_robot.msg import RobotPosition
17 from roomba_robot.msg import PositionAndDestination
18
19 #msg types required to communicate with Action Server are imported from msgs folder
20 from roomba_robot.msg import WalkToPointAction
21 from roomba_robot.msg import WalkToPointResult
22 from roomba_robot.msg import WalkToPointFeedback
23 from roomba_robot.msg import WalkToPointGoal
24
25 #importing built-in packages and modules
26 import os
27 import math
28
29
30 #creating menu for selection of task for roomba
31 menu = """
32 ##### Select options by pressing 1-4 on your keyboard #####
33
34 >>> 1. Current Dirt Amount
35 >>> 2. Current Roomba Position
36 >>> 3. Traverse Roomba to a Specified Destination
37 >>> 4. Return Roomba to the charging station
38 """
39
40
41 #Initializing Global variables with default and rosparam server values
42 CUR_DIRT = 0
43 CUR_ROB_POS = [rospy.get_param('initial_pos_x'), rospy.get_param('initial_pos_y')]
44 ROB_BASE_POINT = [rospy.get_param('initial_pos_x'), rospy.get_param('initial_pos_y')]
45 CUR_ROB_SPEED = 0
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
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
scripts/control_node.py 17:52
```

Figure 3.5: control_node.py code-1

```
control_node.py
45 CUR_ROB_SPEED = 0
46 ROB_MAX_SPEED = rospy.get_param("robot_max_speed")
47
48 #getting range values for the roomba
49 MAX_COORDINATE = rospy.get_param("max_coordinate")
50 MIN_COORDINATE = rospy.get_param("min_coordinate")
51 # -----
52
53
54 #function to set up the environment and initialize values
55 def setup():
56     #setting up the menu as global
57     global menu
58     #clear previous things on the screen
59     clear()
60
61
62
63 #main function of Control Node
64 def main():
65
66
67     #main loop of the control_node
68     while True:
69         #printing menu variable to the screen to show options to the user
70         print(menu)
71
72         #Validity check for input
73         check = True
74         while check:
75             try:
76                 #asking the user for input
77                 choice = int(input("Select: "))
78                 #cheking if the input in this list, otherwise AssertionError will be raised
79                 assert choice in [1, 2, 3, 4, 5]
80                 check = False
81             except:
82                 print("Please choose a valid number")
83
84             #if choice is 1 let the user know about the amount of dirt in the position
scripts/control_node.py 81:9
```

Figure 3.6: control_node.py code-2

```
control_node.py .  
83  
84     #if choice is 1, let the user know about the amount of dirt in the position  
85     if choice == 1:  
86         print("Current DIRT: {} Amount".format(CUR_DIRT))  
87  
88  
89     #if choice is 2, let the user know about the current position of robot  
90     elif choice == 2:  
91         print("Current Position: X = {} Y = {}".format(CUR_ROB_POS[0], CUR_ROB_POS[1]))  
92  
93  
94     #if choice is 3, action client is used to go to the desired position  
95     elif choice == 3:  
96         #from positionanddestination get the goal  
97         goal = PositionAndDestination()  
98         print("Cordinations of Destination")  
99  
100  
101    #to ensure that the value is within the range  
102    x = check_input(" X is equal to ")  
103    y = check_input(" Y is qual to ")  
104  
105    #setting goal msg data  
106    goal.dest_x = x  
107    goal.dest_y = y  
108    goal.cur_x = CUR_ROB_POS[0]  
109    goal.cur_y = CUR_ROB_POS[1]  
110  
111    #calling the walk_to_point to create an Action Client  
112    walk_to_point(goal)  
113  
114    #if choice is 5, use action client to go to the charging station of robot  
115    else:  
116        #creating the goal message  
117        goal = PositionAndDestination()  
118        #setting goal msg data  
119        goal.dest_x = 0  
120        goal.dest_y = 0  
121        goal.cur_x = CUR_ROB_POS[0]  
122        goal.cur_y = CUR_ROB_POS[1]  
scripts/control_node.py 119:2
```

Figure 3.7: control_node.py code-3

```

control_node.py
120     goal.dest_y = 0
121     goal.cur_x = CUR_ROB_POS[0]
122     goal.cur_y = CUR_ROB_POS[1]
123
124     #calling the walk_to_point to create an Action Client
125     walk_to_point(goal)
126
127
128 #Walk To Point Action client function
129 def walk_to_point(message):
130     #defining global variables
131     global ROB_MAX_SPEED
132     global CUR_ROB_POS
133
134     #equation to determining is the robot is in the final destination
135     if message.dest_x == message.cur_x and message.dest_y == message.cur_y:
136         print("Roomba is in the final destination")
137     #otherwise create an action client to walk to the final destination
138     else:
139         #creating action client; action name is 'walk_to_point'
140         #msg type is WalkToPointAction
141         walk_client = actionlib.SimpleActionClient('walk_to_point', WalkToPointAction)
142         #waiting for Action server to respond
143         walk_client.wait_for_server()
144         #create a goal message for walking
145         destination = WalkToPointGoal(message)
146
147         #Let the user know about distance and time to reach the final destination
148         distance = round(math.dist([message.cur_x, message.cur_y], [message.dest_x, message.dest_y]), 1)
149         time_to_reach = int(distance/ROB_MAX_SPEED)
150         print("Distance: {} meters and the Time Taken: {} s".format(distance, time_to_reach))
151
152         #sending the goal to the Action server, feedback callback function is process_feedback
153         #Feedback callback function creates a thread for itself
154         walk_client.send_goal(destination, feedback_cb = process_feedback)
155         #wait for the Action server to finish the task to get the result
156         walk_client.wait_for_result()
157
158         rospy.sleep(0.7)
159
scripts/control_node.py 156:32

```

Figure 3.8: control_node.py code-4

```
control_node.py
159
160     #clear the terminal screen
161     clear()
162
163     #printing the current position of roomba
164     print("Current Cordinations position: X = {} Y = {}".format(CUR_ROB_POS[0], CUR_ROB_POS[1]))
165
166
167 #action client feedback callback funtion
168 def process_feedback(feedback):
169     #accessing gloabl variable
170     global CUR_ROB_SPEED
171     #Let user know abot the distance to reach final destination
172     print("Current Distance: {} m moving at Speed: {} m/s".format(round(feedback.distance, 1), CUR_ROB_SPEED))
173
174
175
176
177
178 #callback function for the subscriber of '/dirt' topic
179 def dirt_sub(val):
180     #defining the global variable
181     global CUR_DIRT
182     #updating the global variable value to a the float number using built in round function
183     CUR_DIRT = round(val.data, 1)
184
185
186
187
188
189 #callback function for the subscriber of '/position' topic
190 def rob_pos_sub(val):
191     #defining the global variable
192     global CUR_ROB_POS
193     #updating the global variable value to a the float number using built in round function
194     CUR_ROB_POS[0] = round(val.pos_x)
195     CUR_ROB_POS[1] = round(val.pos_y)
196
197
198 #callback function for the subscriber of '/speed' topic
scripts/control_node.py 195:35
```

Figure 3.9: control_node.py code-5

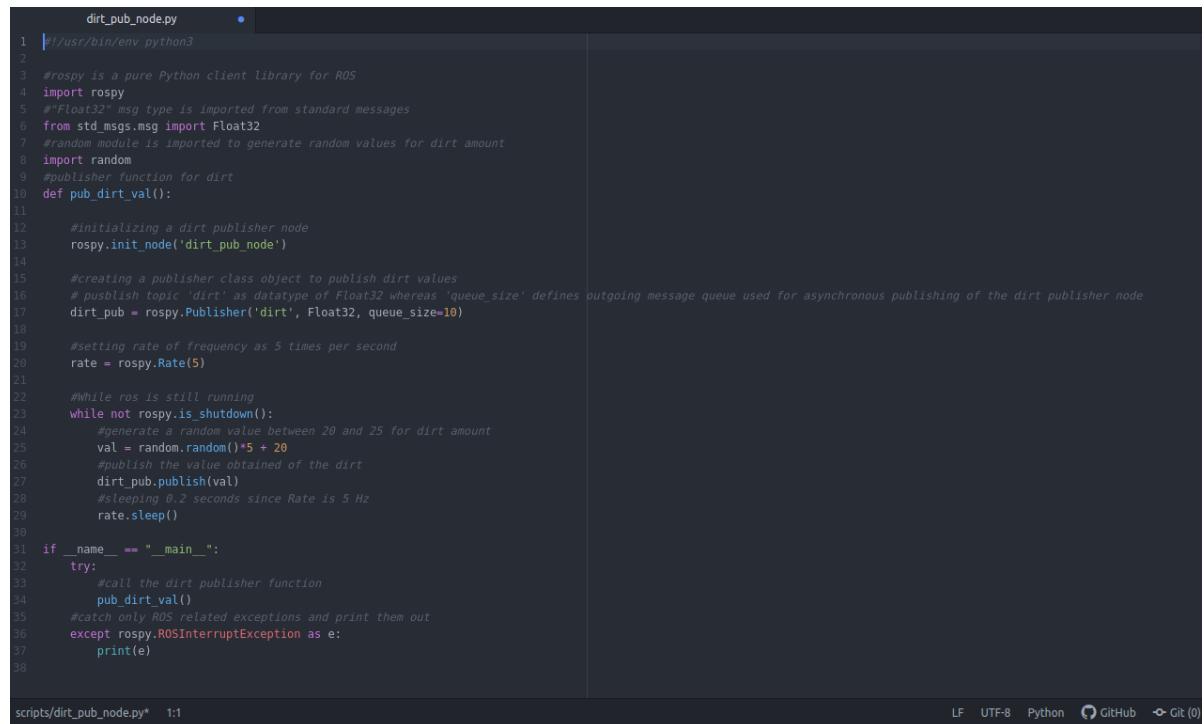
```
control_node.py
197
198 #callback function for the subscriber of '/speed' topic
199 def rob_speed_sub(val):
200     #defining the global variable
201     global CUR_ROB_SPEED
202     #updating the global variable value to a the float number using built in round function
203     CUR_ROB_SPEED = round(val.data, 1)
204
205
206 #clearing the user terminal  function
207 def clear():
208     # for mac and linux (os.name is 'posix')
209     if os.name == 'posix':
210         _ = os.system('clear')
211         print()
212     else:
213         # for windows platform
214         _ = os.system('cls')
215         print()
216
217
218 #auto-validate user input values
219 def check_input(asdked):
220     #defining the global variable
221     #the range of roomba operation
222     global MAX_CORDINATE
223     global MIN_CORDINATE
224     #Flag variable to keep track of whether the user input is valid
225     flag = True
226
227     while flag:
228         try:
229             #User input for valid position
230             number = float(input(asdked))
231             #checking the input value is within the acceptable range
232             assert MIN_CORDINATE <= number and number <= MAX_CORDINATE, "Select from {} to {}"
233             #if there is no AssertionError,the loop is stopped
234             flag = False
235
236             #if AssertionError occurs raise an exception and print an error has occurred
scripts/control_node.py  233:18
```

Figure 3.10: control_node.py code-6

```
control_node.py •  
235  
236     #if AssertionError occurs, raise an exception and print an error has occurred  
237     except AssertionError as msg:  
238         print("Error")  
239  
240     #if other exception arises, ask user to reenter the input  
241     except:  
242         print("Enter a valid number")  
243  
244     #return the valid user input  
245     return number  
246 if __name__ == "__main__":  
247     try:  
248         #initializing a control node  
249         rospy.init_node("control_node")  
250  
251         #creating a subscriber for the topic "dirt"  
252         #received dirt as data type of Float32 from the callback func named dirt_sub  
253         rospy.Subscriber('dirt', Float32, dirt_sub)  
254         #creating a subscriber for the topic "position"  
255         #received position as data type of RobotPosition from the callback func named rob_pos_sub  
256         rospy.Subscriber('position', RobotPosition, rob_pos_sub)  
257         #creating a subscriber for the topic "speed"  
258         #received speed as data type of Float32 from the callback func named rob_speed_sub  
259         rospy.Subscriber('speed', Float32, rob_speed_sub)  
260  
261         #launching setup function to set up the environment  
262         setup()  
263         #main function of Control Node - this is from where all the execution begins  
264         main()  
265  
266         #to continuously run this control_node  
267         rospy.spin()  
268  
269         #catch only ROS related exceptions and print them out  
270     except rospy.ROSInterruptException as e:  
271         print(e)  
272  
273  
274  
scripts/control_node.py* 245:15
```

Figure 3.11: control_node.py code-7

3.6.2. dirt_pub_node.py



```
dirt_pub_node.py
1 #!/usr/bin/env python3
2
3 #rospy is a pure Python client library for ROS
4 import rospy
5 #"Float32" msg type is imported from standard messages
6 from std_msgs.msg import Float32
7 #random module is imported to generate random values for dirt amount
8 import random
9 #publisher function for dirt
10 def pub_dirt_val():
11
12     #initializing a dirt publisher node
13     rospy.init_node('dirt_pub_node')
14
15     #creating a publisher class object to publish dirt values
16     # publish topic 'dirt' as datatype of Float32 whereas 'queue_size' defines outgoing message queue used for asynchronous publishing of the dirt publisher node
17     dirt_pub = rospy.Publisher('dirt', Float32, queue_size=10)
18
19     #setting rate of frequency as 5 times per second
20     rate = rospy.Rate(5)
21
22     #while ros is still running
23     while not rospy.is_shutdown():
24         #generate a random value between 20 and 25 for dirt amount
25         val = random.random()*5 + 20
26         #publish the value obtained of the dirt
27         dirt_pub.publish(val)
28         #sleeping 0.2 seconds since Rate is 5 Hz
29         rate.sleep()
30
31 if __name__ == "__main__":
32     try:
33         #call the dirt publisher function
34         pub_dirt_val()
35     #catch only ROS related exceptions and print them out
36     except rospy.ROSInterruptException as e:
37         print(e)
38
```

LF UTF-8 Python GitHub Git (0)

Figure 3.12: dirt_pub_node.py code

3.6.3. position_pub_node.py

```

position_pub_node.py
1 #!/usr/bin/env python3
2
3 #rospy is a pure Python client library for ROS
4 import rospy
5
6 #Import RobotPosition to access pos_x and pos_y for Roomba
7 from roomba_robot.msg import RobotPosition
8
9 #Define and initialize global variables
10 POSITION = RobotPosition()
11 #Access and assign initial values from rosparam server
12 POSITION.pos_x = rospy.get_param('initial_pos_x')
13 POSITION.pos_y = rospy.get_param('initial_pos_y')
14
15
16 #callback function for the subscriber of '/change_position' topic
17 def update_position(data):
18     #defining the global variable
19     global POSITION
20     #updating the global variable value with current position value to always keep the last position
21     POSITION = data
22
23
24 #publisher function
25 def pub_position_val():
26     #defining the global variable
27     global POSITION
28
29     #creating a publisher class object to publish position values
30     #publish topic 'position' as msgtype of RobotPosition where 'queue_size' is defined as outgoing message queue used for asynchronous publishing
31     position_pub = rospy.Publisher('position', RobotPosition, queue_size=10)
32
33     #setting rate of frequency at 10 times per second
34     rate = rospy.Rate(10)
35

```

Figure 3.13: position_pub_node.py code-1

```

55
56     #while ros is still running
57     while not rospy.is_shutdown():
58         #publishing the position whose msg type is of the type RobotPosition
59         position_pub.publish(POSITION)
60         rate.sleep()
61
62
63     if __name__ == "__main__":
64         try:
65             #initializing this position publisher node
66             rospy.init_node('position_pub_node')
67
68             #creating a subscriber for the topic "change_position"
69             #received change_position as data type of RobotPosition from callback func of update_position
70             rospy.Subscriber("change_position", RobotPosition, update_position)
71
72             #call publisher for position
73             pub_position_val()
74
75             #to continuously keep this node running
76             rospy.spin()
77
78         #catch only ROS related exceptions and print them out
79         except rospy.ROSInterruptException as e:
80             print(e)
81
scripts/position_pub_node.py 61:1
LF  UTF-8  F

```

Figure 3.14: position_pub_node.py code-2

3.6.4. speed_pub_node.py

```

1 #!/usr/bin/env python3
2
3 #rospy is a pure Python client library for ROS
4 import rospy
5
6 #"Float32" msg type is imported from standard messages
7 from std_msgs.msg import Float32
8
9 # initializing the speed to start off with 0
10 SPEED = 0
11
12
13 #callback function for the subscriber of '/change_speed' topic
14 def update_speed(val):
15     #defining the global variable
16     global SPEED
17     #updating the speed to reflect the current speed value with the datatype of float32 as set by round function
18     SPEED = round(val.data, 1)
19
20
21 #publisher function
22 def pub_speed_val():
23     #defining the global variable
24     global SPEED
25
26     #creating a publisher class object to publish speed values
27     #publish the topic 'speed' as datatype of Float32 where the 'queue_size' is defined outgoing message queue used for asynchronous publishing
28     speed_pub = rospy.Publisher('speed', Float32, queue_size=10)
29
30     #setting the frequency of 10 times per second
31     rate = rospy.Rate(10)
32
33     #while the ros is still running
34     while not rospy.is_shutdown():
35         #publishing the speed as datatype of Float32
36         speed_pub.publish(SPEED)
37         rate.sleep()
38
39

```

Figure 3.15: speed_pub_node.py code-1

```

39
40 if __name__ == "__main__":
41     try:
42         #initializing this speed publisher node
43         rospy.init_node('speed_pub_node')
44
45         #creating a subscriber for the topic "change_speed"
46         #received the topic change speed as data type of Float32 where the callback func is defined as update_speed
47         rospy.Subscriber("change_speed", Float32, update_speed)
48
49         #call publisher for speed
50         pub_speed_val()
51
52         #to continuously run this node
53         rospy.spin()
54
55     #catch only ROS related exceptions and print them out
56     except rospy.ROSInterruptException as e:
57         print(e)
58

```

scripts/speed_pub_node.py 58:1

LF UTF-8 Python GitHub Git (0)

Figure 3.16: speed_pub_node.py code-2

3.6.5. turn_camera_client.py

```

turn_camera_client.py
1 #!/usr/bin/env python3
2
3 #rospy is a pure Python client library for ROS
4 import rospy
5
6 #import TurnCamera and TurnCameraResponse message types from the srv folder
7 from roomba_robot.srv import TurnCamera, TurnCameraResponse
8
9 #import opencv modules and packages for python for displaying the images
10 import cv2
11
12 # import CvBridge module which converts between ROS Image messages and OpenCV images.
13 from cv_bridge import CvBridge
14
15 #import built-in modules and packages
16 import os
17 import numpy as np
18
19
20 #define the function for angle to turn image at
21 def config_request(angle):
22
23     #wait for the service to connect
24     rospy.wait_for_service('/turn_camera')
25
26     try:
27         #Service definitions container for the request and response type for turning the camera at a particular angle
28         service_proxy = rospy.ServiceProxy('/turn_camera', TurnCamera)
29         resp_msg = service_proxy(angle)
30
31
32         #send response of the image as message
33         image_msg = resp_msg.image
34
35         #convert a ROS image message into an cv
36         #Since the default value of "passthrough" is given, the destination image encoding will be the same as the image message encoding
37         image = CvBridge().imgmsg_to_cv2(image_msg, desired_encoding='passthrough')
38
39
40

```

Figure 3.17: turn_camera_client.py code-1

```

turn_camera_client.py
37     image = CvBridge().imgmsg_to_cv2(image_msg, desired_encoding='passthrough')
38
39
40     #After successfully converted images to OpenCV format, see a HighGui window with the desired image will be displayed.
41     cv2.imshow('Turn Camera Image', image)
42
43     #handling keyboard interrupt
44     cv2.waitKey(0)
45     cv2.destroyAllWindows()
46
47     #catch only ROS Service related exceptions and print them out
48     except rospy.ServiceException as e:
49         print("Service Request Failed")
50         print(e)
51
52
53     if __name__ == "__main__":
54         try:
55             #initialize this camera client
56             rospy.init_node("turn_camera_client_node")
57
58             #prompt user for angle input
59             user_input = input("\nEnter the angle: ")
60
61             #continue trying to process angle request until user presses q for quitting
62             while user_input != 'q':
63                 try:
64                     #unless error occurs continue asking user for angles which will be in the float32 datatype
65                     config_request(float(user_input))
66                     user_input = input("\nEnter the angle: ")
67
68             #if any error while trying to process the request, an exception will be thrown
69             except Exception as e:
70                 print("Error trying to process request")
71                 print(e)
72
73     #catch only ROS related exceptions and pass
74     except rospy.ROSInterruptException:
75         pass

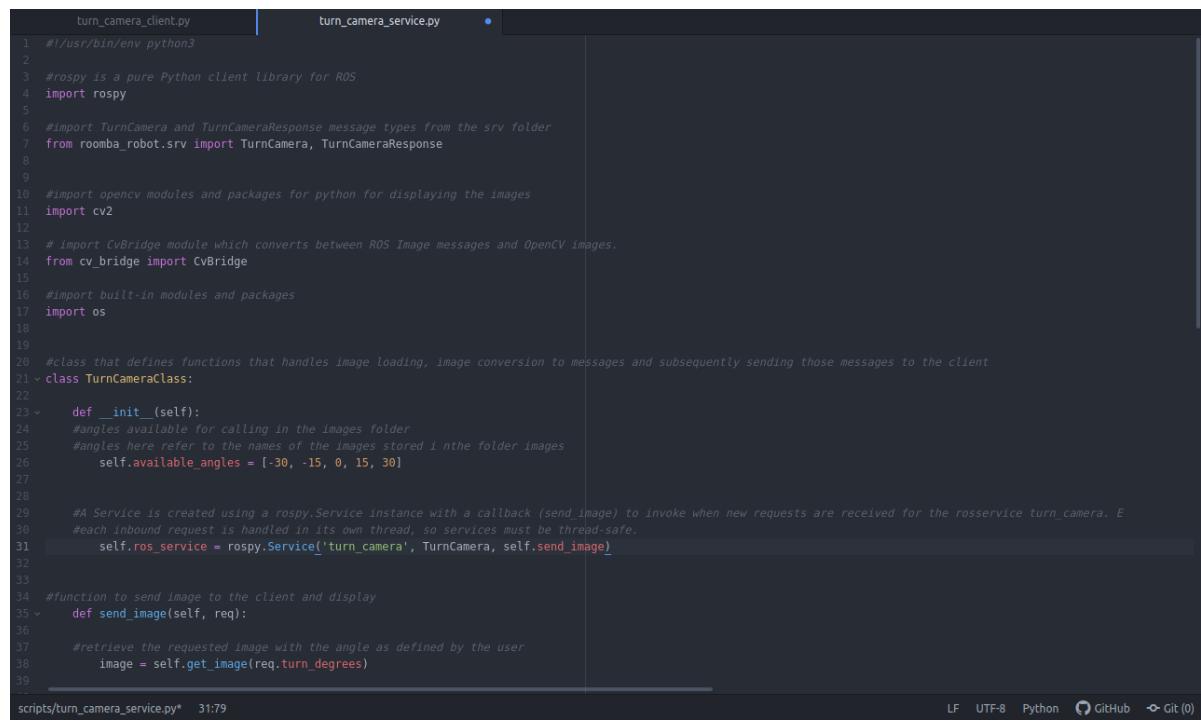
```

scripts/turn_camera_client.py 66:46

LF UTF-8 Python GitHub Git (0)

Figure 3.18: turn_camera_client.py code-2

3.6.6. turn_camera_service.py



```
turn_camera_client.py          turn_camera_service.py •
1 #!/usr/bin/env python3
2
3 #rospy is a pure Python client library for ROS
4 import rospy
5
6 #import TurnCamera and TurnCameraResponse message types from the srv folder
7 from roomba_robot.srv import TurnCamera, TurnCameraResponse
8
9
10 #import opencv modules and packages for python for displaying the images
11 import cv2
12
13 # import CvBridge module which converts between ROS Image messages and OpenCV images.
14 from cv_bridge import CvBridge
15
16 #import built-in modules and packages
17 import os
18
19
20 #class that defines functions that handles image loading, image conversion to messages and subsequently sending those messages to the client
21 class TurnCameraClass:
22
23     def __init__(self):
24         #angles available for calling in the images folder
25         #angles here refer to the names of the images stored in the folder images
26         self.available_angles = [-30, -15, 0, 15, 30]
27
28
29     #A Service is created using a rospy.Service instance with a callback (send_image) to invoke when new requests are received for the rosservice turn_camera. Each inbound request is handled in its own thread, so services must be thread-safe.
30     self.ros_service = rospy.Service('turn_camera', TurnCamera, self.send_image)
31
32
33
34     #function to send image to the client and display
35     def send_image(self, req):
36
37         #retrieve the requested image with the angle as defined by the user
38         image = self.get_image(req.turn_degrees)
39
scripts/turn_camera_service.py* 31:79
LF  UTF-8  Python  GitHub  Git (0)
```

Figure 3.19: turn_camera_service.py code-1

```

turn_camera_client.py          turn_camera_service.py •
56
57 #retrieve the requested image with the angle as defined by the user
58 image = self.get_image(req.turn_degrees)
59
60 #using the cvbridge module, convert the retrieved image to a message and make it in a format ready to be sent to the client
61 image_msg = CvBridge().cv2_to_imgmsg(image)
62 return TurnCameraResponse(image_msg)
63
64
65 #function to get the correct image
66 def get_image(self, angle):
67
68     #as mentioned above, the angle is the file name, however, if the user inputs a value which is not in the listed available_angles, an equation is developed to match to
69     closest_angle = min(self.available_angles, key=lambda x: abs(x-angle))
70
71     return self.read_image_file(str(closest_angle) + '.png')
72
73 #function to read the image from the image folder
74 def read_image_file(self, file_name):
75
76     #sets the directory path
77     dir_name = os.path.dirname(__file__)
78
79     #set the file location directory, which is the images folder
80     file_location = dir_name + "/Images/" + file_name
81
82     #use the python module cv2 function imread to read the image from the file location specified
83     image = cv2.imread(file_location)
84
85     return image
86
87

```

Figure 3.20: turn_camera_service.py code-2

```

88 if __name__ == '__main__':
89     try:
90
91         #initialize this node
92         rospy.init_node('turn_camera_service_node')
93         #call the service class to access all the functions defined by this class
94         TurnCameraClass()
95         print("Service is running")
96
97         #continue running this node
98         rospy.spin()
99
100    #catch only ROS related exceptions and print the exception
101    except rospy.ROSInterruptException as e:
102        print(e)
103

```

scripts/turn_camera_service.py 83:1

LF UTF-8 Python GitHub Git (0)

Figure 3.21: turn_camera_service.py code-3

3.6.7. walk_to_point_action_server_node.py

```

turn_camera_client.py      walk_to_point_action_server.py
1 #!/usr/bin/env python3
2
3 #rospy is a pure Python client library for ROS
4 import rospy
5
6 #import built-in modules
7 import math
8 import threading
9
10 #import actionlib module to help in creating an Action server
11 import actionlib
12
13
14 #"Float32" msg type is imported from standard messages
15 from std_msgs.msg import Float32
16
17 #import and create action msg types for action server
18 from roomba_robot.msg import WalkToPointAction
19 from roomba_robot.msg import WalkToPointResult
20 from roomba_robot.msg import WalkToPointFeedback
21
22 #import and create msg types for updating positional data
23 from roomba_robot.msg import PositionAndDestination
24 from roomba_robot.msg import RobotPosition
25
26
27 #create a class for Action Server
28 class Walking:
29
30     def __init__(self, pub1, pub2):
31         #create an Action server with name: 'walk to point'
32         #received topic walk_to_point from callback func of reach destination
33         self.action_server = actionlib.SimpleActionServer('walk_to_point', WalkToPointAction, self.reach_destination)
34
35         #assign 'speed_change'&'position_change' to the created publisher
36         self.pub_speed_change = pub1
37         self.pub_pos_change = pub2
38         #create current speed variable
39         self.cur_speed = 0
40
41         self.robot_max_speed = rospy.get_param("robot_max_speed")
42         #setting rate of frequency as 20 times per second
43         self.publisher_rate = rospy.Rate(20)
44
45         #creating current and final position objects
46         self.robot_cur_point = RobotPosition()
47         self.robot_goal_point = RobotPosition()
48
49
50     #callback function of Action server
51     def reach_destination(self, goal):
52
53         #threads to publish changes in speed and position while performing the action
54         #target is the publisher function
55         self.thread_speed = threading.Thread(target=self.pub_change_speed)
56         self.thread_position = threading.Thread(target=self.pub_change_position)
57
58         #flag to continue running the thread
59         self.thread_flag = True
60
61         #Extract current position of robot
62         self.robot_cur_point.pos_x = goal.destin.cur_x
63         self.robot_cur_point.pos_y = goal.destin.cur_y
64
65         #Extract final destination position values
66         self.robot_goal_point.pos_x = goal.destin.dest_x
67         self.robot_goal_point.pos_y = goal.destin.dest_y
68
69         #calculate the distance to final destination using math.dist
70         #time = distance/speed from physics formula
71         distance = round(math.dist([self.robot_cur_point.pos_x, self.robot_cur_point.pos_y], [self.robot_goal_point.pos_x, self.robot_goal_point.pos_y]), 1)
72         time_to_reach = int(distance/self.robot_max_speed)
73
74         #let the user know about distance and time to reach
75         print("Distance: {} meters travelled in the Time: {} s\n".format(distance, time_to_reach))
76
scripts/walk_to_point_action_server.py  1:1
LF  UTF-8  Python  GitHub  Git (0)

```

Figure 3.22: walk_to_point_action_server_node.py code-1

```

turn_camera_client.py      walk_to_point_action_server.py
37         self.thread_flag = True
38
39         #Extract current position of robot
40         self.robot_cur_point.pos_x = goal.destin.cur_x
41         self.robot_cur_point.pos_y = goal.destin.cur_y
42
43         #Extract final destination position values
44         self.robot_goal_point.pos_x = goal.destin.dest_x
45         self.robot_goal_point.pos_y = goal.destin.dest_y
46
47         #calculate the distance to final destination using math.dist
48         #time = distance/speed from physics formula
49         distance = round(math.dist([self.robot_cur_point.pos_x, self.robot_cur_point.pos_y], [self.robot_goal_point.pos_x, self.robot_goal_point.pos_y]), 1)
50         time_to_reach = int(distance/self.robot_max_speed)
51
52         #let the user know about distance and time to reach
53         print("Distance: {} meters travelled in the Time: {} s\n".format(distance, time_to_reach))
54
scripts/walk_to_point_action_server.py  73:3
LF  UTF-8  Python  GitHub  Git (0)

```

Figure 3.23: walk_to_point_action_server_node.py code-2

```
turn_camera_client.py           walk_to_point_action_server.py
75   print("Distance: {} meters travelled in the time: {} s\n".format(distance, time_to_reach))
76
77 #Roomba traverses at 45 angle to the final position
78 #Defining x step for each second where roomba walks x step distance in x coordinate
79 if self.robot_cur_point.pos_x < self.robot_goal_point.pos_x:
80     step_x = (self.robot_goal_point.pos_x - self.robot_cur_point.pos_x)/time_to_reach
81 else:
82     step_x = (self.robot_cur_point.pos_x - self.robot_goal_point.pos_x)/time_to_reach
83     step_x *= -1
84
85 #Defining y step for each second where roomba walks y_step distance in y coordinate
86 if self.robot_cur_point.pos_y < self.robot_goal_point.pos_y:
87     step_y = (self.robot_goal_point.pos_y - self.robot_cur_point.pos_y)/time_to_reach
88 else:
89     step_y = (self.robot_cur_point.pos_y - self.robot_goal_point.pos_y)/time_to_reach
90     step_y *= -1
91
92 #assign current speed to max speed
93 self.cur_speed = self.robot_max_speed
94
95 #initiate threads to publish changes in speed and position independently
96 self.thread_speed.start()
97 self.thread_position.start()
98
99 #Let the users know about the remaining distance
100 if time_to_reach < 10:
101     interval = 2
102 elif time_to_reach < 20:
103     interval = 3
104 else:
105     interval = 4
106
107 #initiate walking
108 for i in range(time_to_reach):
109     if i%interval == 0:
110         #calculating the distance remaining and sending it to Action client as feedback
111         distance = math.dist([self.robot_cur_point.pos_x, self.robot_cur_point.pos_y], [self.robot_goal_point.pos_x, self.robot_goal_point.pos_y])
112         self.action_server.publish_feedback(WalkToPointFeedback(distance))
113
114 #clean for the 1 sec and
```

scripts/walk_to_point_action_server.py 111:87

LF UTF-8 Python GitHub Git (0)

Figure 3.24: walk_to_point_action_server_node.py code-3

```

turn_camera_client.py          walk_to_point_action_server.py
112                         self.action_server.publish_feedback(WalkToPointFeedback(distance))
113
114     #sleep for the 1 second
115     rospy.sleep(1)
116
117     #add or subtract the position coordinates with step_x and step_y values
118     self.robot_cur_point.pos_x += step_x
119     self.robot_cur_point.pos_y += step_y
120
121     #after Roomba reaches destination, update current position values with final destination position values
122     self.robot_cur_point.pos_x = self.robot_goal_point.pos_x
123     self.robot_cur_point.pos_y = self.robot_goal_point.pos_y
124
125     #setting the current speed value to 0
126     self.cur_speed = 0
127
128     #letting threads wait to publish the last changed values in speed and position
129     rospy.sleep(1)
130
131     #setting thread_flag False to end threads
132     self.thread_flag = False
133
134     #ensure threads stopped successfully
135     rospy.sleep(0.5)
136
137     #join additional threads to the main thread
138     self.thread_speed.join()
139     self.thread_position.join()
140
141     #From walktopoint results create Action result object
142     result = WalkToPointResult()
143     #save value of time spent to reach final destination
144     result.time_spent = time_to_reach
145     #set success answer to client with Action Result msg
146     self.action_server.set_succeeded(result)
147
148
149     #thread function to publish changes in current speed of roomba
150     def pub_change_speed(self):
151

```

LF UTF-8 Python GitHub Git (0)

Figure 3.25: walk_to_point_action_server_node.py code-4

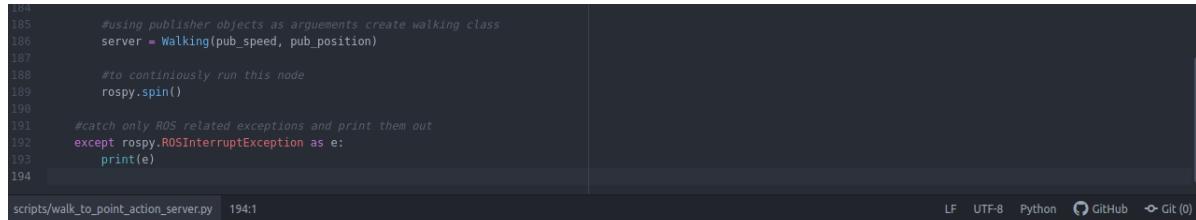
```

turn_camera_client.py          walk_to_point_action_server.py
148
149     #thread function to publish changes in current speed of roomba
150     def pub_change_speed(self):
151
152         #checking thread_flag value
153         while self.thread_flag:
154             #publishing changes
155             self.pub_speed_change.publish(self.cur_speed)
156             #wait 0.05 second (20 Hz)
157             self.publisher_rate.sleep()
158
159
160     #thread function to publish changes in current position of roomba
161     def pub_change_position(self):
162
163         #checking thread_flag value
164         while self.thread_flag:
165             #publishing changes
166             self.pub_pos_change.publish(self.robot_cur_point)
167             #wait 0.05 second (20 Hz)
168             self.publisher_rate.sleep()
169
170
171 if __name__ == '__main__':
172
173     try:
174         #initializing this node
175         rospy.init_node('walk_to_point_action_server_node')
176
177         #create a publisher class object to publish speed changes
178         #publish topic 'change_speed' as datatype of Float32 where 'queue_size' is defined as outgoing message queue used for asynchronous publishing
179         pub_speed = rospy.Publisher('change_speed', Float32, queue_size=10)
180
181         #create a publisher class object to publish position changes
182         #publish topic 'change_position' as datatype RobotPosition where 'queue_size' -> outgoing message queue used for asynchronous publishing
183         pub_position = rospy.Publisher('change_position', RobotPosition, queue_size=10)
184
185         #using publisher objects as arguments create walking class
186         server = Walking(pub_speed, pub_position)
187

```

LF UTF-8 Python GitHub Git (0)

Figure 3.26: walk_to_point_action_server_node.py code-5



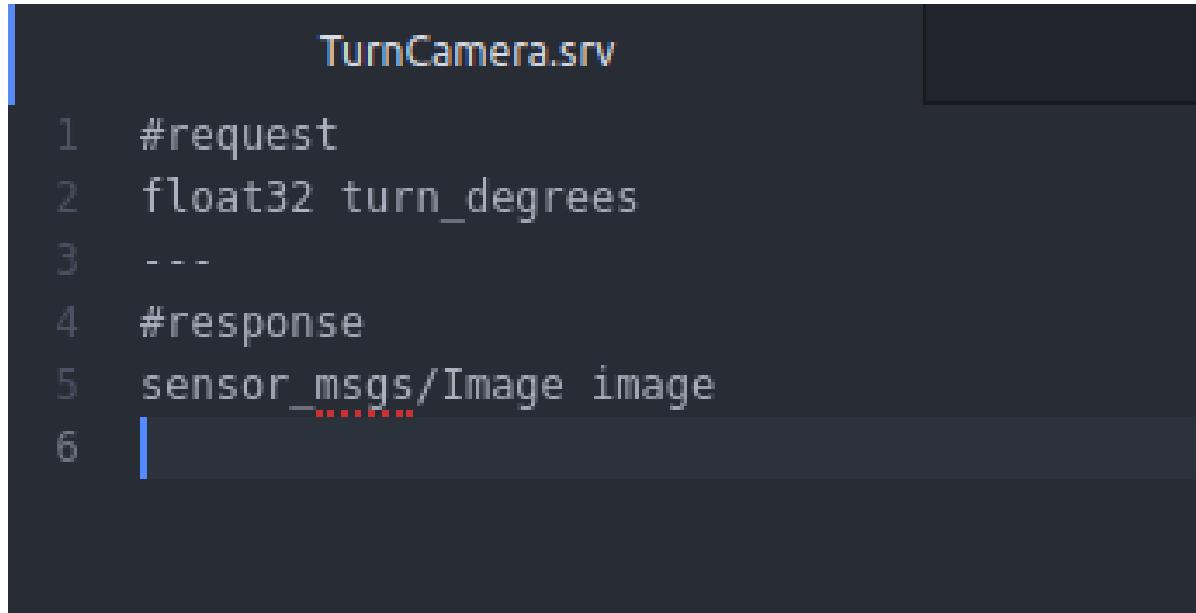
```
84     #using publisher objects as arguments create walking class
85     server = Walking(pub_speed, pub_position)
86
87     #to continuously run this node
88     rospy.spin()
89
90
91     #catch only ROS related exceptions and print them out
92     except rospy.ROSInterruptException as e:
93         print(e)
94
95
scripts/walk_to_point_action_server.py 194:1
```

LF UTF-8 Python GitHub Git (0)

Figure 3.27: walk_to_point_action_server_node.py code-6

3.7. srv files

srv files are placed in the srv folder and there is only 1 srv file named: TurnCamera.srv



```
TurnCamera.srv
```

```
1 #request
2 float32 turn_degrees
3 ---
4 #response
5 sensor_msgs/Image image
6 |
```

Figure 3.28: TurnCamera.srv code

Bibliography

- ROS.org. (2021a). *Actionlib*. <http://wiki.ros.org/actionlib>
- ROS.org. (2021b). *Bags*. <http://wiki.ros.org/Bags>
- ROS.org. (2021c). *Creating a workspace for catkin*. http://wiki.ros.org/catkin/Tutorials/create_a_workspace
- ROS.org. (2021d). *Roslaunch*. <http://wiki.ros.org/roslaunch>
- ROS.org. (2021e). *Services*. <http://wiki.ros.org/Services>
- ROS.org. (2021f). *Understanding ros nodes*. <http://wiki.ros.org/ROS/Tutorials/UnderstandingNodes>