**Installation and Code Compilation Steps**

1. ***Install ROS(Robot Operating System)***

The first and foremost step is to download ROS on your system. We would prefer that you use ubuntu OS. But you can find the availability of ROS for different operating systems as well. Go to the following link for specific OS related installation

[**http://wiki.ros.org/ROS/Installation**](http://wiki.ros.org/ROS/Installation)

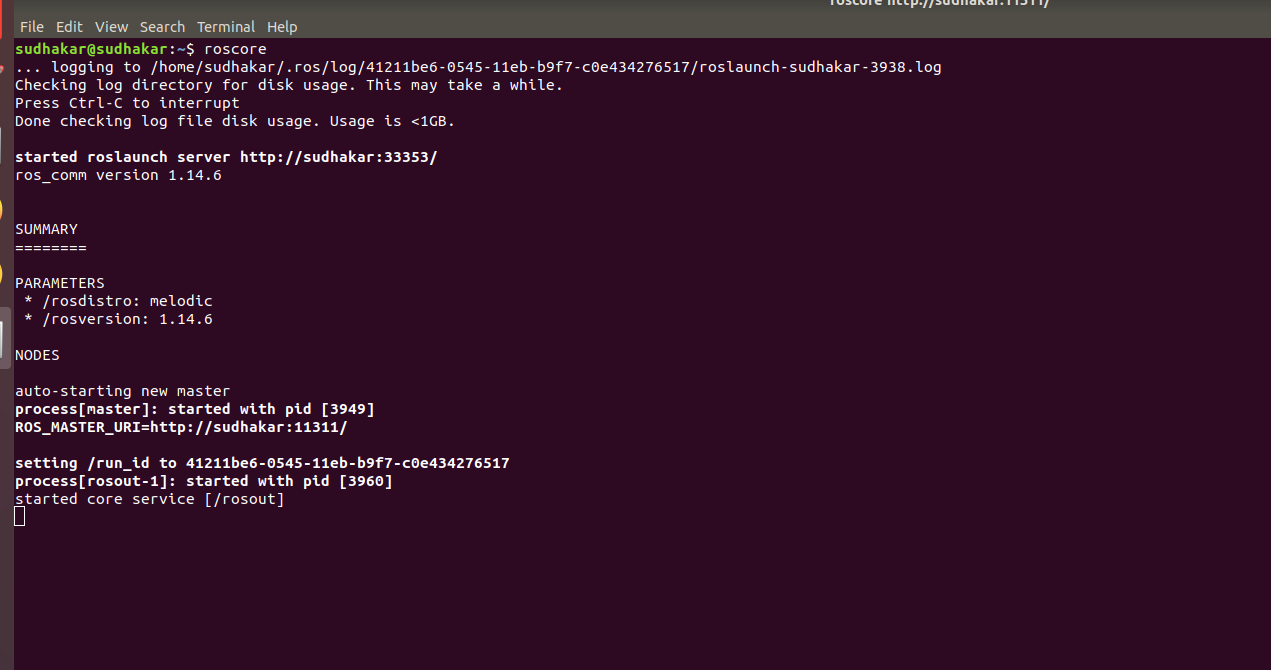
You can download any ROS version like **kinetic / melodic / noetic.** Do the **Desktop-Full Install** as you will be working on the gazebo simulator

1. ***Check for successful installation***

After the installation, you check if it is successful by writing the following command

**roscore**

You should see an output shown below.

****

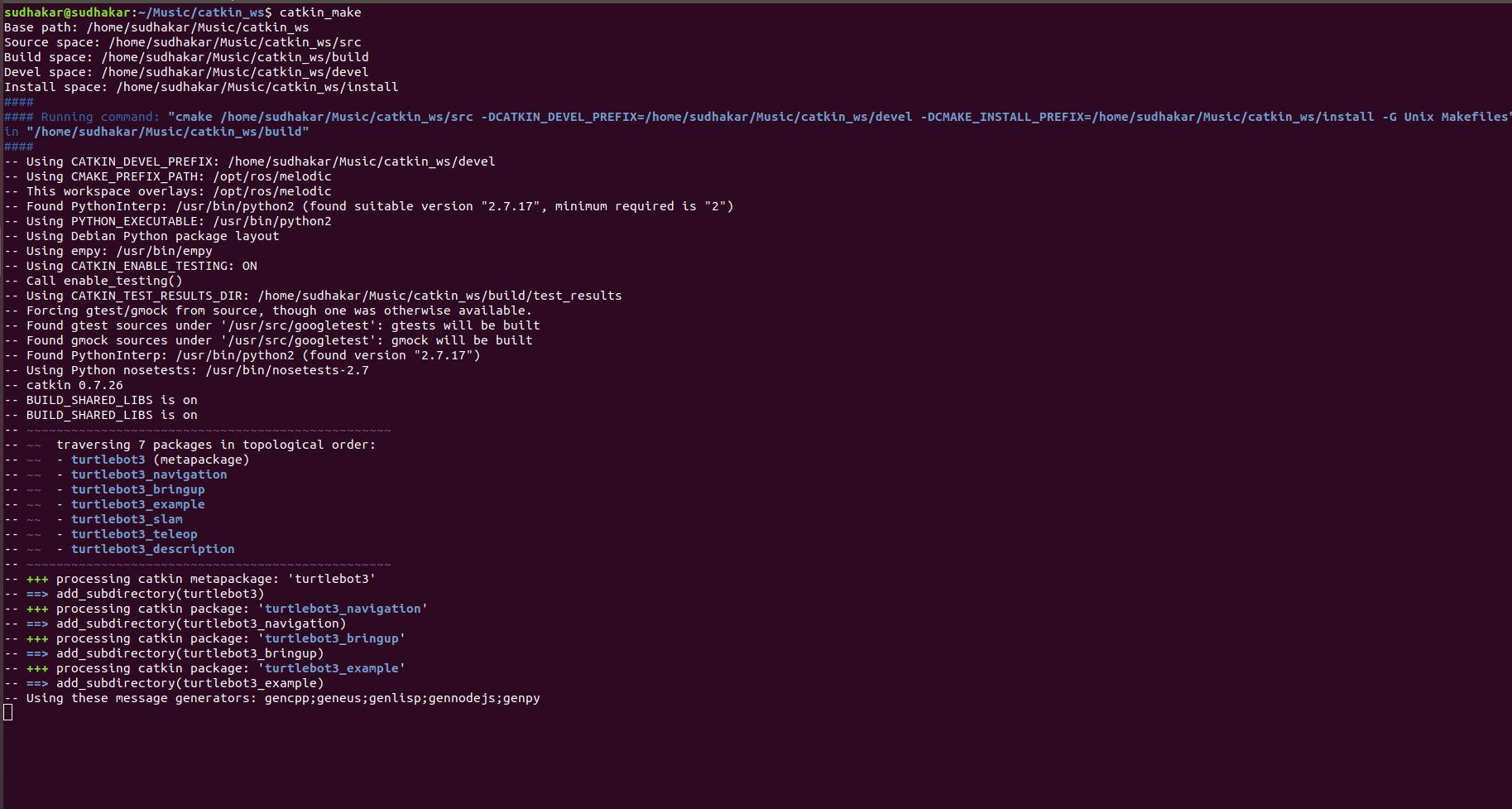
**You can use Ctrl C to stop the ROS operations**

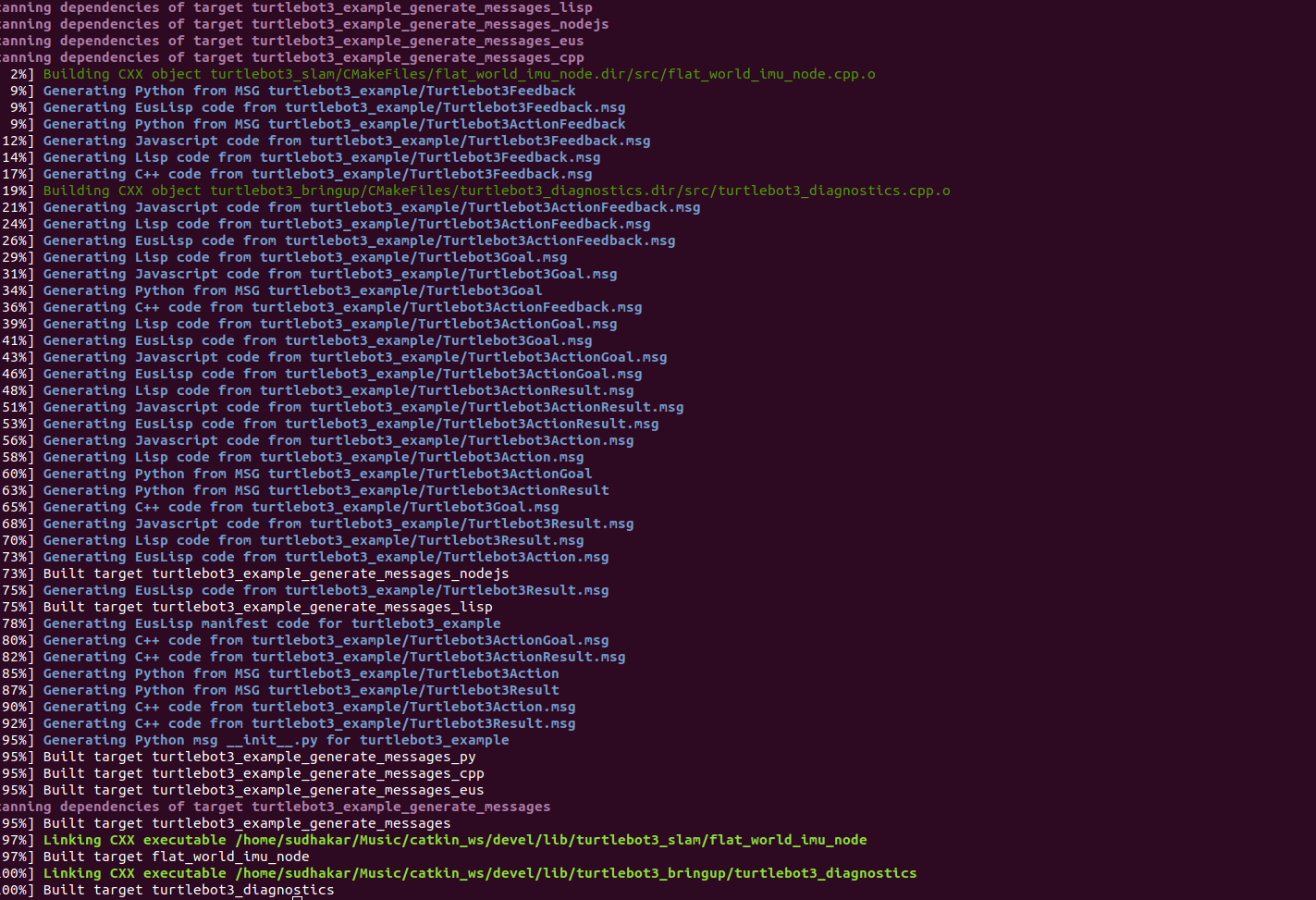
1. ***Unzip the package***

We have provided the whole workspace as a zip file*.* You need to unzip the package and go to the directory. You just need to build the workspace by running the following command in the terminal

**catkin\_make**

**You will see the following output**

****

****

If the workspace is 100% built, this means all the packages in the src folder are built successfully without any errors and you can carry out further operations.

In case there are any errors, you will be required to install specific dependencies that might be missing on your system. ROS related dependencies can be downloaded by using the following commands

**sudo apt-get install ros-rosversion-ros\_package**

Example: *if you are using ROS kinetic version , you can use the following command sudo apt-get install ros-kinetic-slam-gmapping*

This will download the slam-gmapping package, You will have to check from the error which package is missing and you can perform the installation of the same.

1. ***Running the packages***

Every time you run a package using the terminal, you need to use the command ***source devel/setup.bash*** on every terminal screen where you run the package

Now to work with the packages, these are the following steps

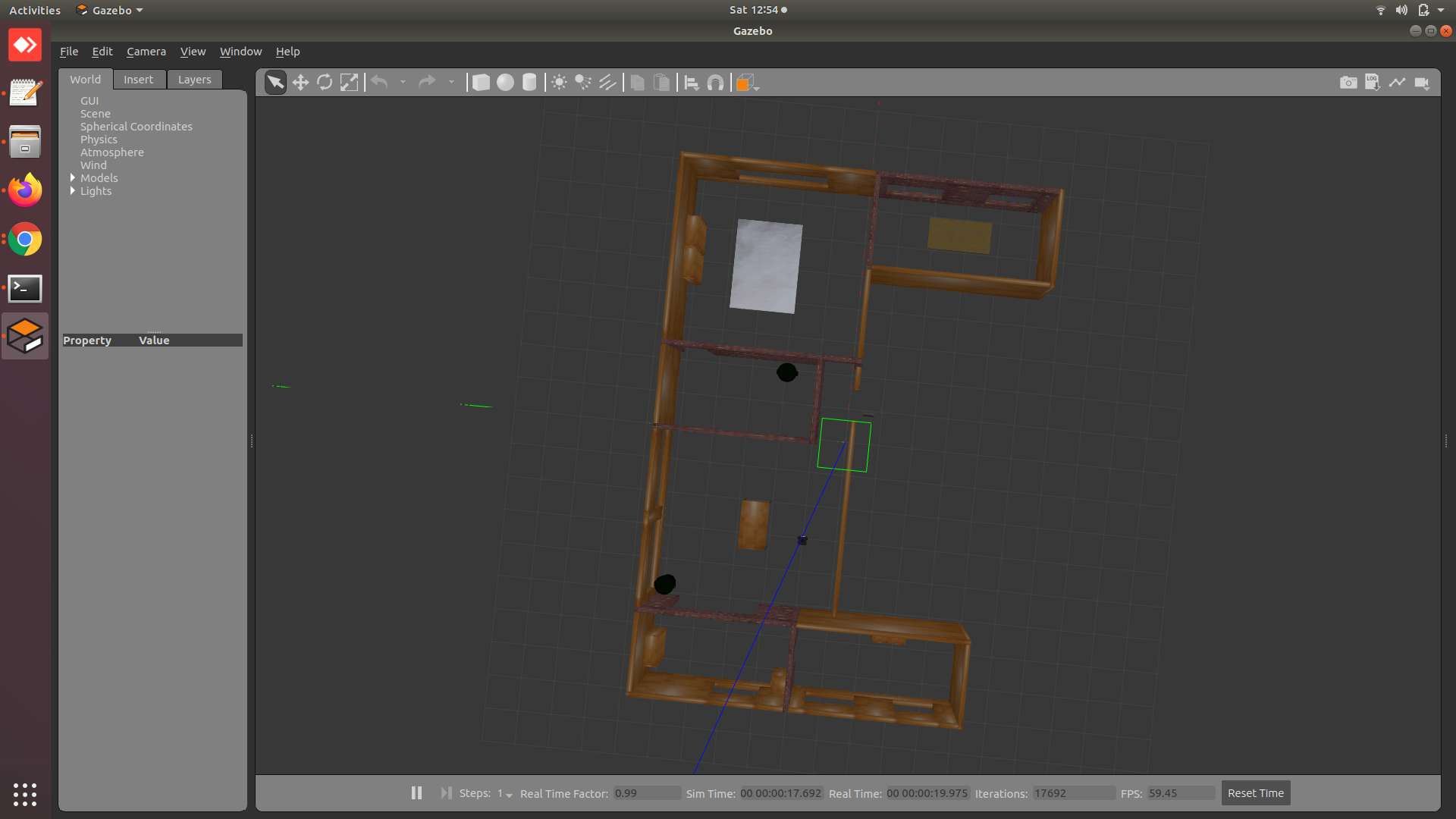
**a) Open the gazebo simulator**

Open the gazebo simulator using the following command

***roslaunch turtlebot3\_gazebo turtlebot3\_house.launch***

****

**The output will be as follows**

****

**You will see a simulated house with a robot. You can zoom in and out using your mouse and also check out every room in the simulator**

To see the charging docking station, you will have to make changes to the path of the docking station model in the following file ***catkin\_ws/src/turtlebot3\_simulations/turtlebot3\_gazebo/worlds/turtlebot3\_house.world***

You will find the following uri

***<uri>/home/user/catkin\_ws/src/turtlebot3\_simulations/turtlebot3\_gazebo/models/docking\_station</uri>***

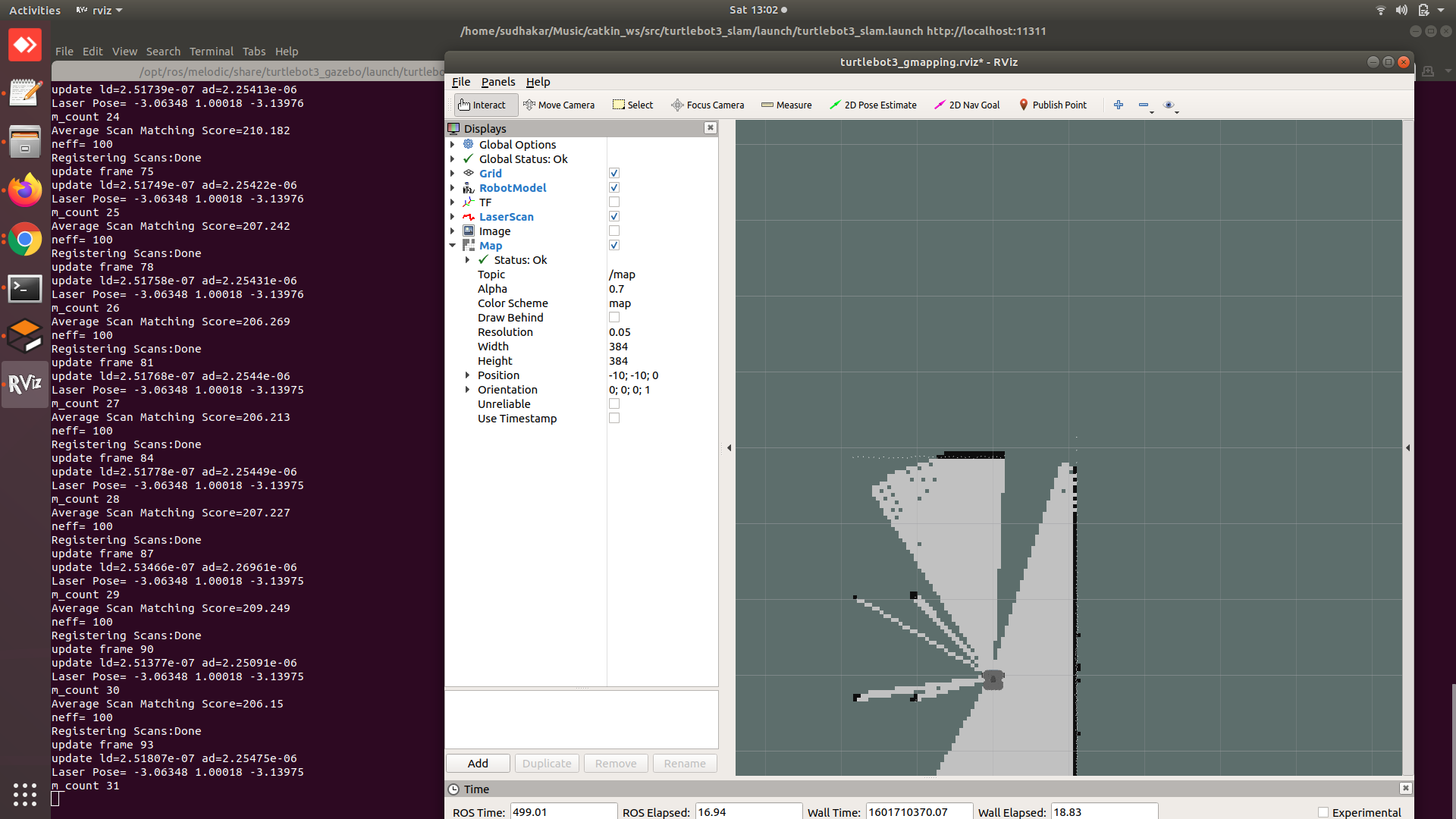
You need to change the path according to the path on your system and you will see a charging station for docking as a box attached to a wall in the hall

**b) Create a map of the house**

Next you need to create a map of the environment i.e house. To start the mapping process, write the following command in the next terminal

***roslaunch turtlebot3\_slam turtlebot3\_slam.launch slam\_methods:=gmapping***

A new window, Rviz, will open which will show the map being created. The output will be something like this



You need to move the robot (shown in next step) so as to create the whole map. When you are done creating the map, you can save the map by writing the following command

***rosrun map\_server map\_saver -f map\_file\_name***

**(map\_file\_name -** file name of the map created)

This will create 2 files

* **yaml file**
* **pgm file**

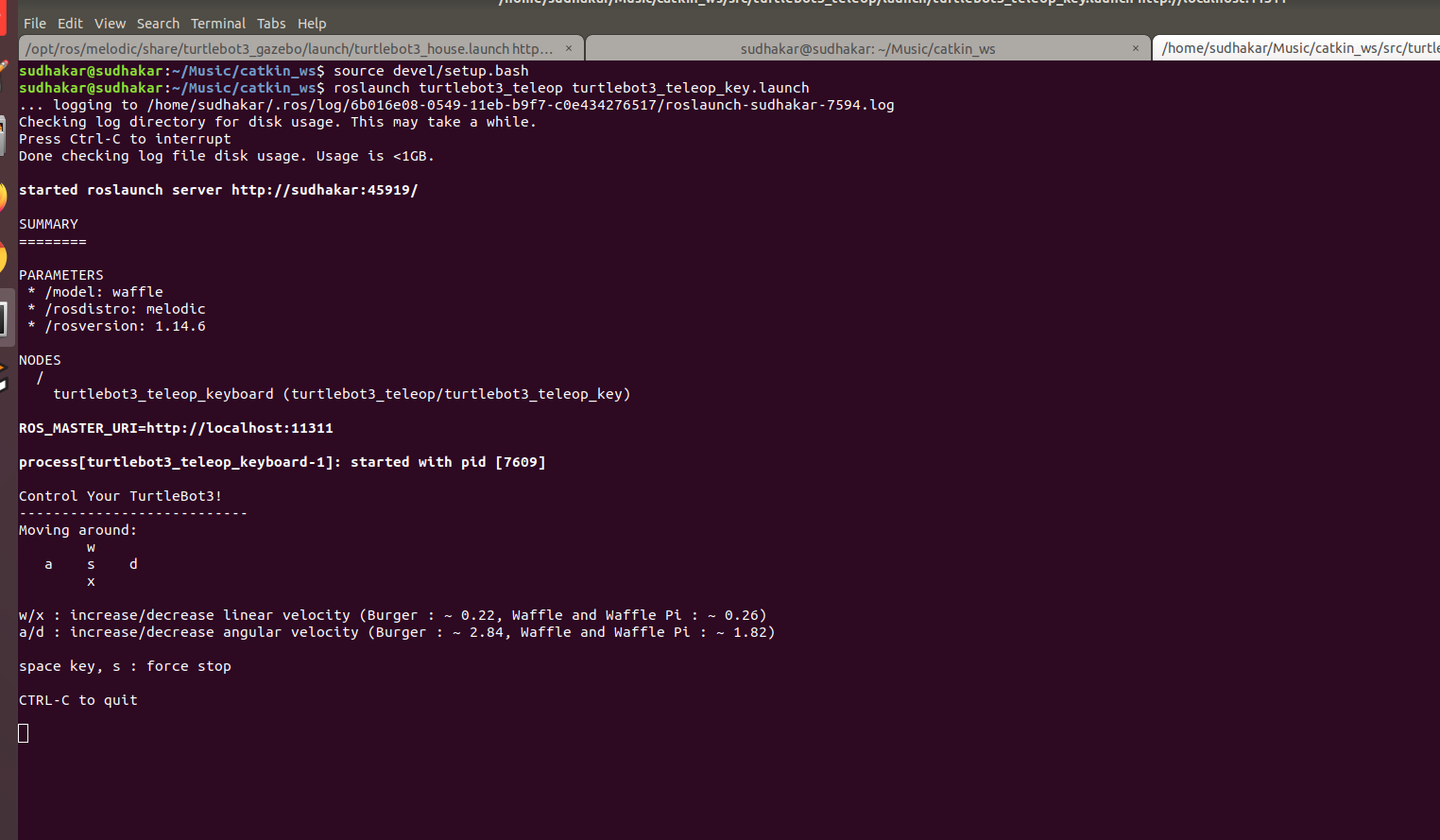
pgm file is an image that shows you the map and yaml file contains information related to the same.

**c) Move the robot around**

Use the following command to move the robot around

***roslaunch turtlebot3\_teleop turtlebot3\_teleop\_key.launch***

You will see the below screen

****

**w -> move forward**

**a -> turn left**

**d -> turn right**

**x -> move backwards**

**s -> stop**

**d) Localization**

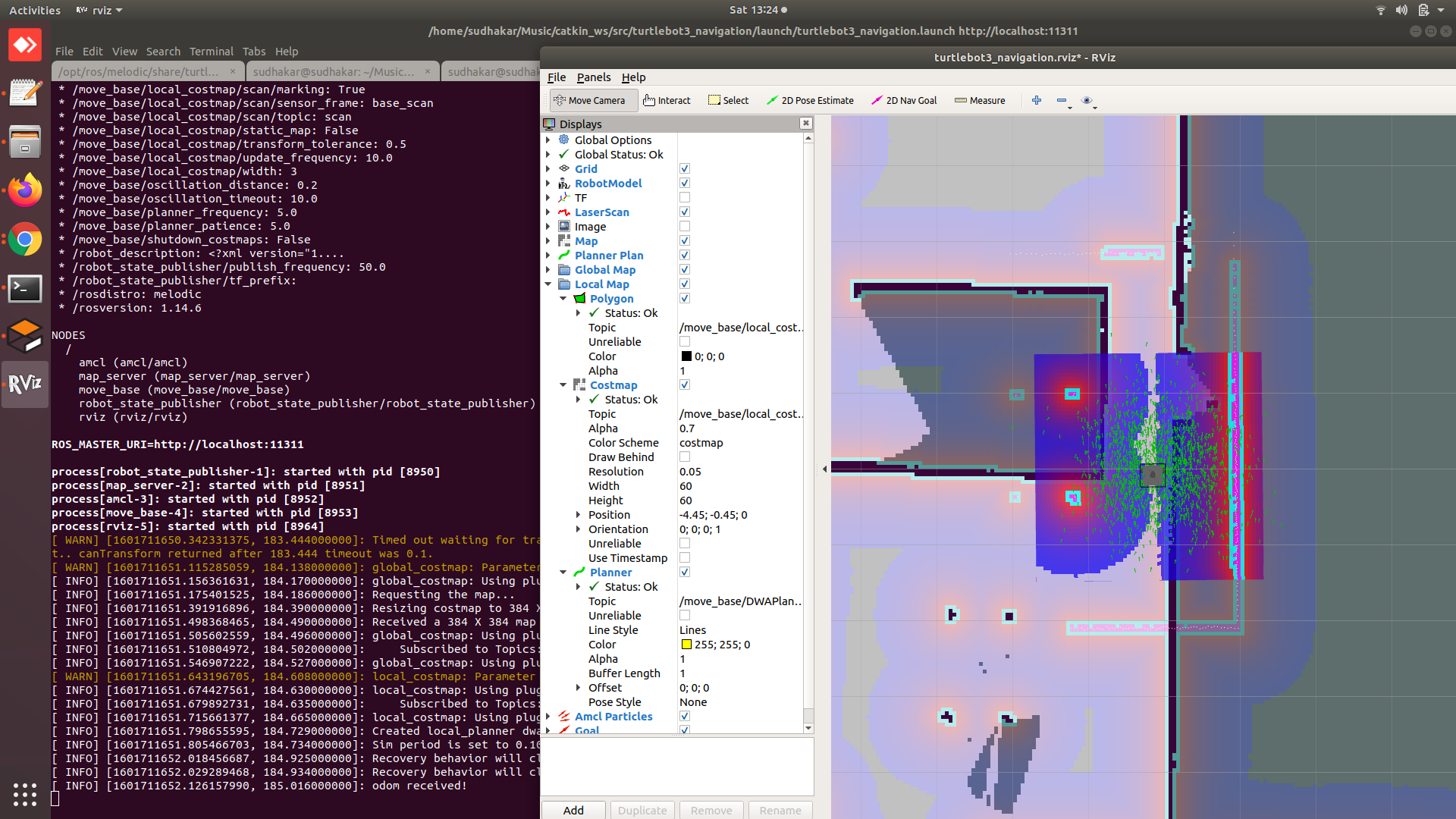
Once the map is created and saved, close every terminal. Relaunch the gazebo model by using command

***roslaunch turtlebot3\_gazebo turtlebot3\_house.launch***

To start the localization process, use the following command

***roslaunch turtlebot3\_navigation turtlebot3\_navigation.launch map\_file:=map\_file\_name.yaml***

The map\_file\_name is the name of file with which you have saved the map

The output will look like 

You can give an approximate correct position of the robot if the robot is not at the correct position by clicking the **2d Pose Estimate** button in the top bar and then clicking on the correct position in RVIZ

You can also ask the robot to navigate to a certain position by clicking on **2D nav goal** in the top bar and clicking on the desired position on the RVIZ screen



If you are able to complete all the steps mentioned above, the mapping and localization pipeline will be in place. Once your robot is able to understand the user’s voice commands, you need to generate navigation commands. The starter code for generating navigation commands is given below.

**Participants are encouraged to look for ROS commands online if they get stuck somewhere and figure out a way to solve code / dependency related issues.**

**Navigation starter code**

**#include <ros/ros.h>**

**#include <move\_base\_msgs/MoveBaseAction.h>**

**#include <actionlib/client/simple\_action\_client.h>**

**typedef actionlib::SimpleActionClient<move\_base\_msgs::MoveBaseAction> MoveBaseClient;**

**int main(int argc, char\*\* argv){**

**ros::init(argc, argv, "simple\_navigation\_goals");**

**//tell the action client that we want to spin a thread by default**

**MoveBaseClient ac("move\_base", true);**

**//wait for the action server to come up**

**while(!ac.waitForServer(ros::Duration(5.0))){**

**ROS\_INFO("Waiting for the move\_base action server to come up");**

**}**

**move\_base\_msgs::MoveBaseGoal goal;**

**//we'll send a goal to the robot to move 1 meter forward**

**goal.target\_pose.header.frame\_id = "base\_link";**

**goal.target\_pose.header.stamp = ros::Time::now();**

**goal.target\_pose.pose.position.x = 1.0;**

**goal.target\_pose.pose.orientation.w = 1.0;**

**ROS\_INFO("Sending goal");**

**ac.sendGoal(goal);**

**ac.waitForResult();**

**if(ac.getState() == actionlib::SimpleClientGoalState::SUCCEEDED)**

**ROS\_INFO("Hooray, the base moved 1 meter forward");**

**else**

**ROS\_INFO("The base failed to move forward 1 meter for some reason");**

**return 0;**

**}**

**The teams will be required to change the frame\_id from “base\_link” to “map” as the robot needs to navigate with respect to the map and not with respect to itself(base\_link). The teams are required to check it out themselves how they will achieve the same.**