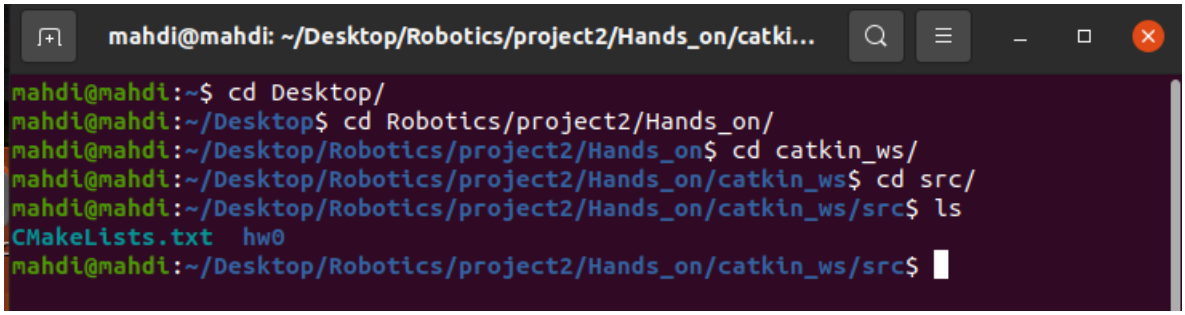
هندزان اول) آشنایی با Gazebo

در اینجا میخواهیم ببینیم که چطور یک ربات را میتوان به گزبو اضافه کرد و کنترل آن را از طریق کیبورد به دست گرفت. همچنین چگونه میتوان یک map به empty world اضافه کرد.

برای شروع طبق هندزان قبلی یک ورک اسپیس به اسم catkin work space در ویدیوی قبل درست کردیم. (انو توی فولدر project2 کپی کردم)

Cd /Desktop/Robotics/project2/Hands_on/catkin_ws/src

توی این سورس باید ۳ تا ریپازیتوری از یک ادرس گیت هابی clone کنیم. هرکدوم از اینا رو که کلون کنیم قراره یک پکیج جدید در کنار پکیج hw0 ایجاد کنند:



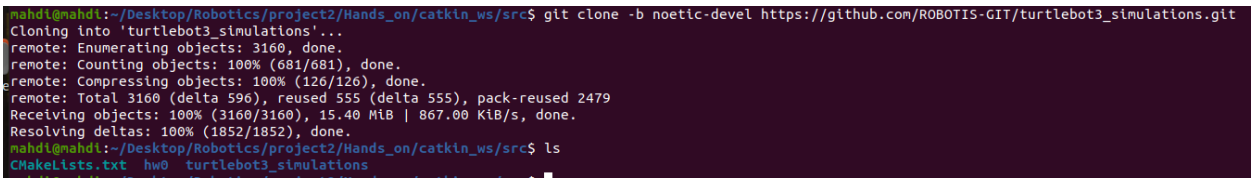
آدرس ریپوها به ترتیب در زیر آمده اند:

git clone -b noetic-devel https://github.com/ROBOTIS-GIT/turtlebot3_simulations.git

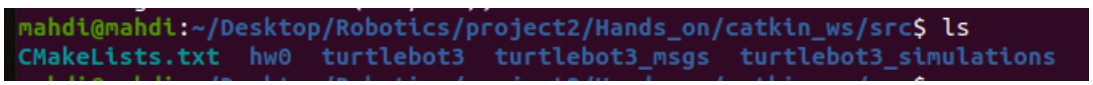
git clone -b noetic-devel https://github.com/ROBOTIS-GIT/turtlebot3.git

git clone -b noetic-devel https://github.com/ROBOTIS-GIT/turtlebot3_msgs.git

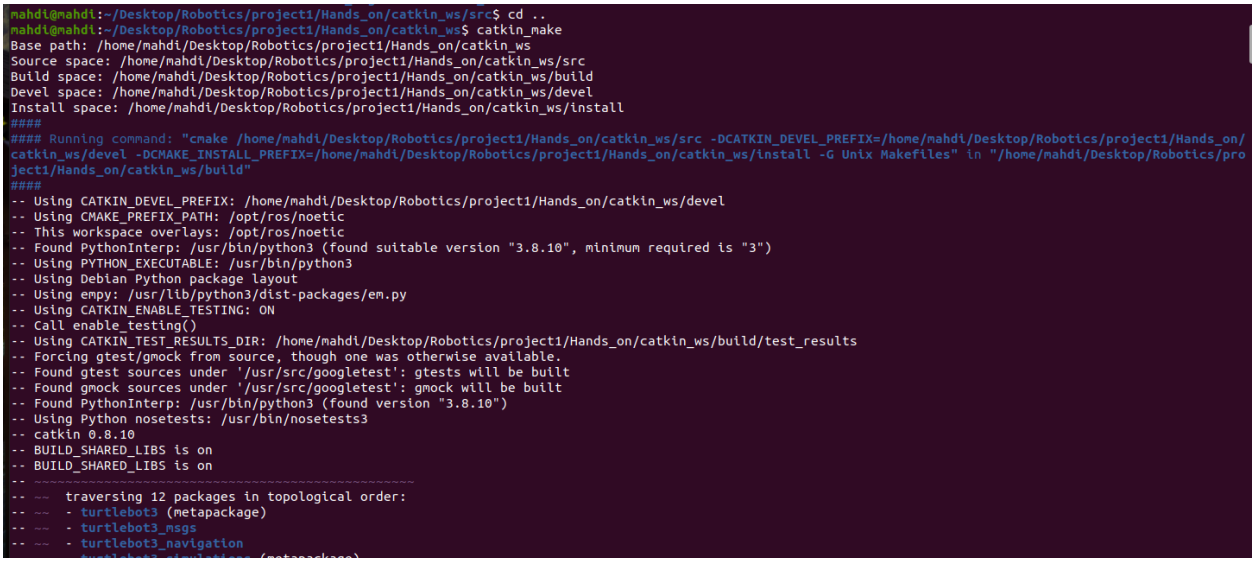
بعد از نصب اولی اگر ls بزنیم خواهیم دید:



دستورات دوم و سوم را هم اجرا میکنیم. سپس اگر ls بزنیم میتوان ۳ تا پکیج جدید را مشاهده کرد:

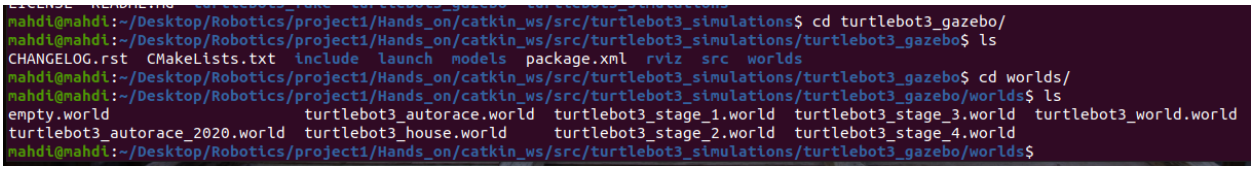


سپس به catkin_ws برگشته و چون پکیج جدید اضافه کردیم باید catkin_make بزنیم که توی پوشه‌های devel و build هم اضافه شود.



حال اگر یک سری به پکیج turtlebot3_simulations بزنیم خواهیم دید:

ما با turtlebot3_gazebo کار داریم. اگر داخل آن برویم چند فولدر داخلشه که مهم هاش یکی launch و دیگری worlds میباشد. چیزی که داخل worlds هست mapهاییی است که میتوانیم از آن‌ها استفاده کنیم. و مثلا اگر داخل آن برویم خواهیم داشت:



یه سری world. هست و یکیشون هم empty.world هست که وقتی gazebo را باز میکنیم به صورت دیفالت میبینیم. میتوان به این فولدر world هم اضافه کرد که در ادامه خواهیم دید.

یک فولدر مهم دیگر launch هست که داخل آن یک سری لانچ فایل داریم که به کمک آن‌ها میایم gazebo را ران میکنیم. فرض کنید توی تمرین ۴ تا نود سنسور، کنترلر و دوتا موتور داریم و برای ران کردن میومدیم یه سری ترمینال جدا باز میکردیم و پکیج و درواقع خود نود را بهش میدهیم و اینا باید به ترتیب پشت هم اجرا شوند که کل سیستم کارش را درست انجام دهد. پس باید به تعداد نودها ترمینال باز کرد و ران کنیم ولی به کمک لانچ فایل همه اینا را میتوان به کمک یک فایل انجام داد و شما دستور roslaunch مینویسید و توی یک فایل لانچ نودها را به همون ترتیبی که میخواید صدا میکنید.

خب برگردیم سر کارمون. ما میخوایم گزبو را بالا اوریم و یک ربات هم در اون محیط مورد نظر قرار دهیم. اولا سورس میکنیم اینجایی که هستیم رو.

```
mahdi@mahdi:~/Desktop/Robotics/project1/Hands_on/catkin_ws$ . devel/setup.bash
mahdi@mahdi:~/Desktop/Robotics/project1/Hands_on/catkin_ws$
```

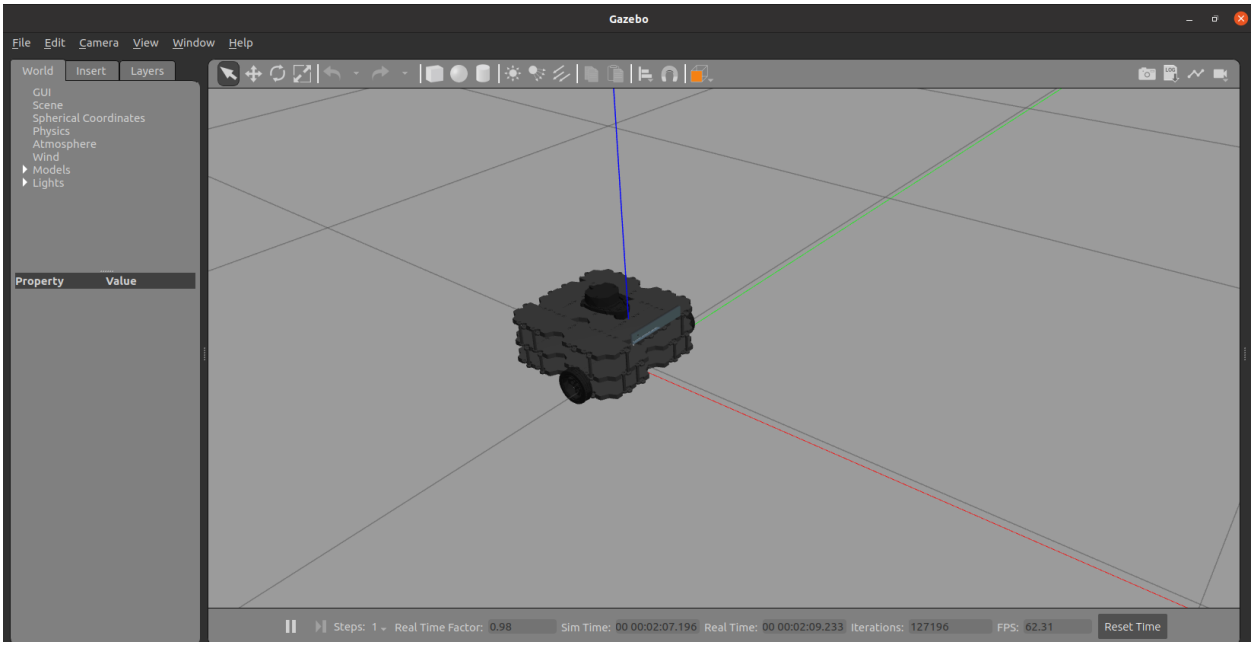
سپس باید بیایم مدل ربات را به gazebo معرفی کنیم. مدل‌هایی که واسه turtlebot3 داریم یا waffle یا burger میباشد که ما فعلا از همین waffle استفاده میکنیم.

```
mahdi@mahdi:~/Desktop/Robotics/project1/Hands_on/catkin_ws$ export TURTLEBOT3_MODEL=waffle
mahdi@mahdi:~/Desktop/Robotics/project1/Hands_on/catkin_ws$
```

سپس به کمک roslaunch میخوایم turtlebot3_empty_world را در gazebo بالا بیاریم:

```
mahdi@mahdi:~/Desktop/Robotics/project1/Hands_on/catkin_ws$ roslaunch turtlebot3_gazebo turtlebot3_empty_world.launch
... logging to /home/mahdi/.ros/log/7c4e7b74-dad5-11ed-b91f-adfb35253529/roslaunch-mahdi-7502.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.

xacro: in-order processing became default in ROS Melodic. You can drop the option.
started roslaunch server http://mahdi:46443/

SUMMARY
========

PARAMETERS
 * /gazebo/enable_ros_network: True
 * /robot_description: <?xml version="1....
 * /rosdistro: noetic
 * /rosversion: 1.15.15
 * /use_sim_time: True

NODES
  /
    gazebo (gazebo_ros/gzserver)
    gazebo_gui (gazebo_ros/gzclient)
    spawn_urdf (gazebo_ros/spawn_model)

auto-starting new master
process[master]: started with pid [7520]
ROS_MASTER_URI=http://localhost:11311

setting /run_id to 7c4e7b74-dad5-11ed-b91f-adfb35253529
process[rosout-1]: started with pid [7530]
started core service [/rosout]
process[gazebo-2]: started with pid [7533]
process[gazebo_gui-3]: started with pid [7535]
```

سپس محیط gazebo باز شده و تصویر زیر را خواهیم دید:



خب پس الان هم مپ لود شد و هم رباتمون. فرض کنید نمیخوایم ربات در نقطه ۰ و ۰ و ۰ که الان لود شده، لود شود. برای مثال توی 2 و 2 و 0 میخوایم لود شود. پس فعلا در ترمینال از gazebo بیرون میایم. ابتدا به مسیر زیر میرویم از درون خود فولدرهای لپتاپ(نه ترمینال):

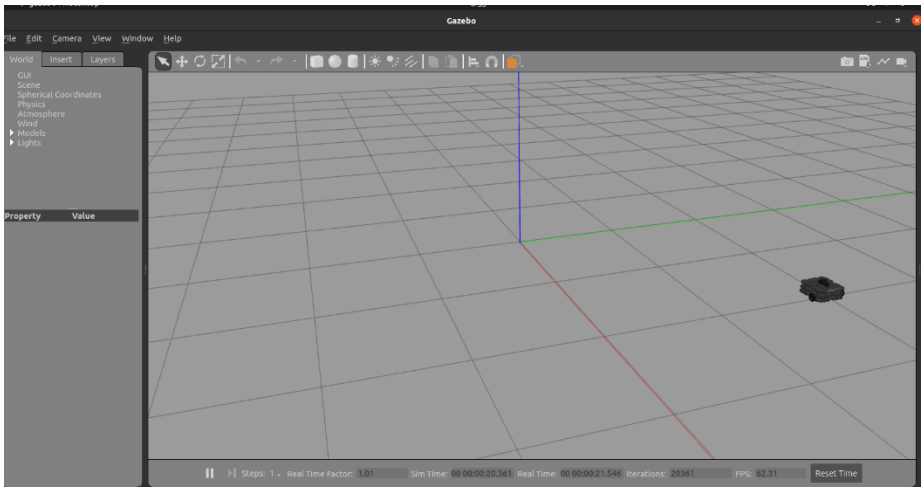
/catkin_ws/src/turtlebot3_simulations/turtlebot3_gazebo/launch

سپس فایل turtlebot3_empty_world را باز میکنیم. حالا پوزیشن ها را به صورت زیر تغییر میدهیم:

Open turtlebot3_empty_world.launch ~/Desktop/Robotics/project1/Hands_o...imulations/turtlebot3_gazebo/launch Save

```
1 <launch>
2   <arg name="model" default="$(env TURTLEBOT3_MODEL)" doc="model type [burger, waffle,
  waffle_pi]"/>
3   <arg name="x_pos" default="2.0"/>
4   <arg name="y_pos" default="2.0"/>
5   <arg name="z_pos" default="0.0"/>
6
7   <include file="$(find gazebo_ros)/launch/empty_world.launch">
8     <arg name="world_name" value="$(find turtlebot3_gazebo)/worlds/empty.world"/>
9     <arg name="paused" value="false"/>
10    <arg name="use_sim_time" value="true"/>
11    <arg name="gui" value="true"/>
12    <arg name="headless" value="false"/>
13    <arg name="debug" value="false"/>
14  </include>
15
16  <param name="robot_description" command="$(find xacro)/xacro --inorder $(find
  turtlebot3_description)/urdf/turtlebot3_$(arg model).urdf.xacro" />
17
18  <node pkg="gazebo_ros" type="spawn_model" name="spawn_urdf" args="-urdf -model turtlebot3_$-
  (arg model) -x $(arg x_pos) -y $(arg y_pos) -z $(arg z_pos) -param robot_description" />
19 </launch>
```

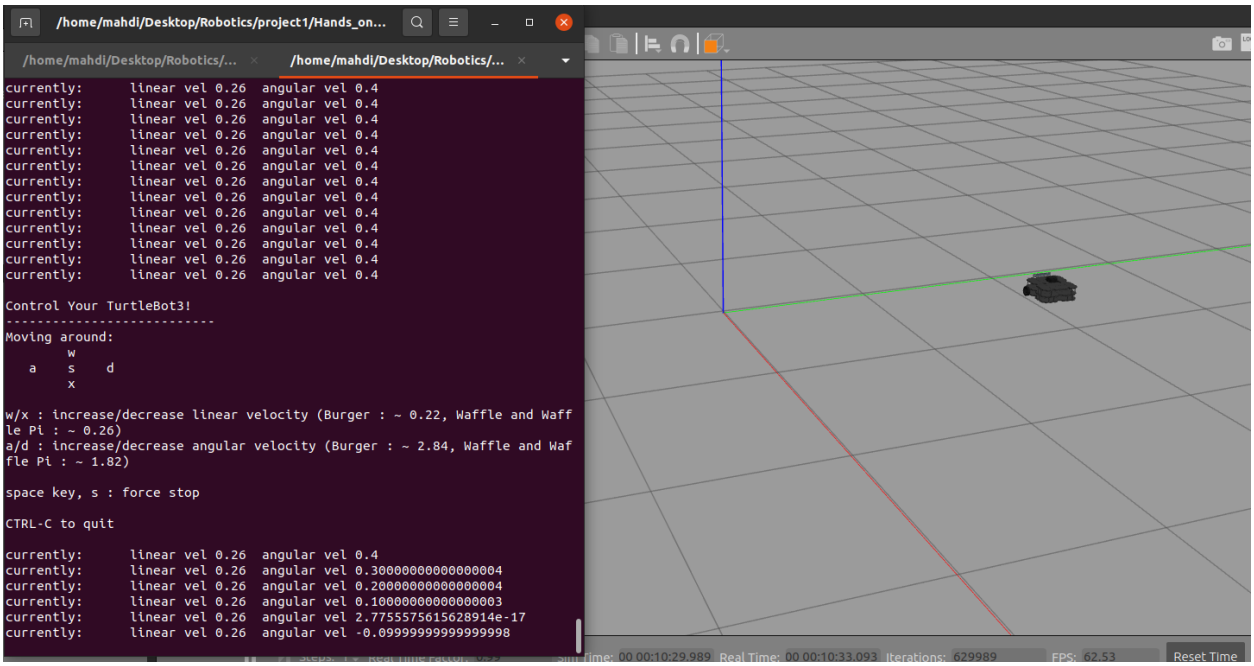
سپس بار دیگر از طریق ترمینال ران میکنیم. همانطور که دیده میشود مکان آن تغییر کرد.



حال یک ترمینال جدید باز کرد و آن را با devel/setup.bash . سورس میکنیم. میخواهیم در اینجا کنترل ربات را به کمک کیبورد به دست بگیریم. باید قبل از آن در این ترمینال جدید نیز حتما مدل ربات را مشخص کنیم. (با دستور export)

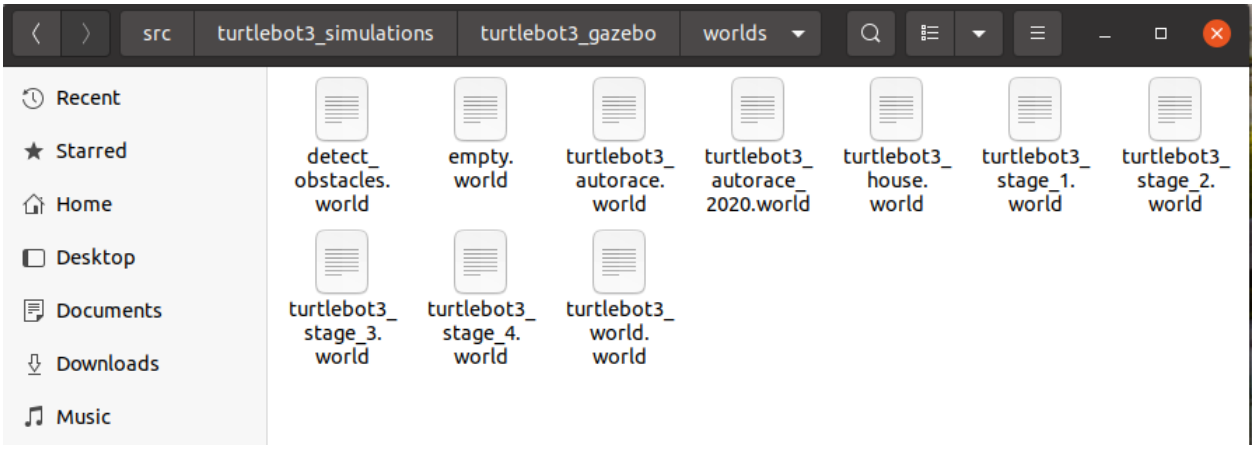
```
mahdi@mahdi:~/Desktop/Robotics/project1/Hands_on/catkin_ws$ . devel/setup.bash
mahdi@mahdi:~/Desktop/Robotics/project1/Hands_on/catkin_ws$ export TURTLEBOT3_MODEL=waffle
mahdi@mahdi:~/Desktop/Robotics/project1/Hands_on/catkin_ws$ roslaunch turtlebot3_teleop turtlebot3_teleop_key.launch
... logging to /home/mahdi/.ros/log/1d298b6e-dad7-11ed-b91f-adfb35253529/roslaunch-mahdi-8408.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://mahdi:41869/

SUMMARY
========

PARAMETERS
 * /model: waffle
 * /rosdistro: noetic
 * /rosversion: 1.15.15

NODES
  /
    turtlebot3_teleop_keyboard (turtlebot3_teleop/turtlebot3_teleop_key)

ROS_MASTER_URI=http://localhost:11311

process[turtlebot3_teleop_keyboard-1]: started with pid [8422]

Control Your TurtleBot3!
---------------------------
Moving around:
```

حال به کمک کلیدهای کیبورد میتوان آن را جا به جا کرد:

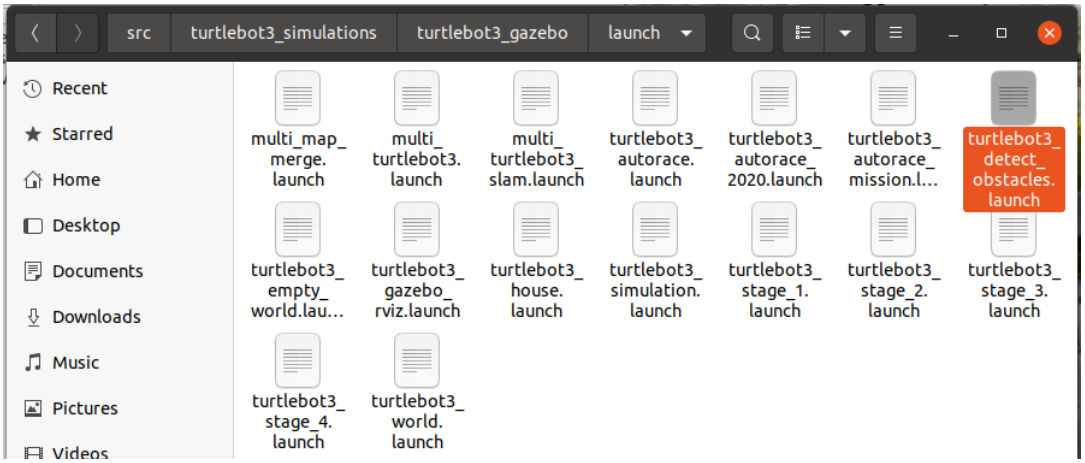


در بخش بعدی هدف ما این است که فرض کنیم یک زمین جدید داریم و میخواهیم رباتمان را در این زمین جدید قرار دهیم. زمین جدیدی که داریم detect_obstacles.world میباشد. کافی است این فایل را در فولدر worlds در مسیر زیر کپی کنید:

/catkin_ws/src/turtlebot3_simulations/turtlebot3_gazebo/worlds



حال به سراغ launch میرویم. همون فایل turtlebot3_empty_world.launch را کپی کرده در همونجا و سپس اسم آن را turtlebot3_detect_obstacles.launch میگذاریم.



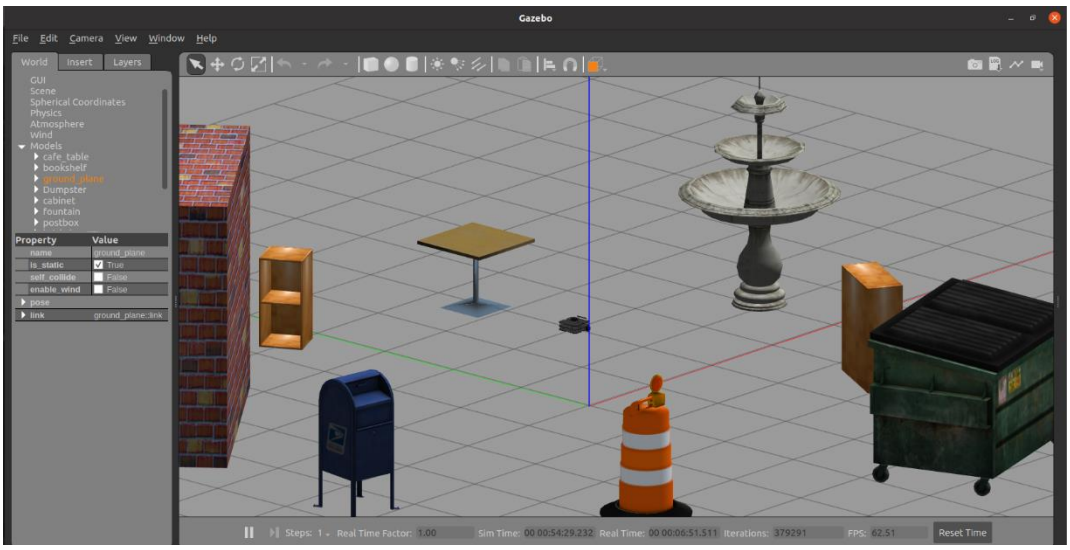
سپس آن را باز کرده و در خط ۸ باید اسم world ای که کپی کردیم را وارد کنیم:

```
1 <launch>
2   <arg name="model" default="$(env TURTLEBOT3_MODEL)" doc="model type [burger, waffle,
  waffle_pi]"/>
3   <arg name="x_pos" default="2.0"/>
4   <arg name="y_pos" default="2.0"/>
5   <arg name="z_pos" default="0.0"/>
6
7   <include file="$(find gazebo_ros)/launch/empty_world.launch">
8     <arg name="world_name" value="$(find turtlebot3_gazebo)/worlds/detect_obstacles.world"/>
9     <arg name="paused" value="false"/>
10    <arg name="use_sim_time" value="true"/>
11    <arg name="gui" value="true"/>
12    <arg name="headless" value="false"/>
13    <arg name="debug" value="false"/>
14  </include>
15
16  <param name="robot_description" command="$(find xacro)/xacro --inorder $(find
  turtlebot3_description)/urdf/turtlebot3_$(arg model).urdf.xacro" />
17
18  <node pkg="gazebo_ros" type="spawn_model" name="spawn_urdf" args="-urdf -model turtlebot3_$-
  (arg model) -x $(arg x_pos) -y $(arg y_pos) -z $(arg z_pos) -param robot_description" />
19 </launch>
```

حال میتوانیم ران بگیریم. دستورات زیر را اجرا کرده:

```
mahdi@mahdi:~/Desktop/Robotics/project1/Hands_on/catkin_ws$ . devel/setup.bash
mahdi@mahdi:~/Desktop/Robotics/project1/Hands_on/catkin_ws$ export TURTLEBOT3_MODEL=waffle
mahdi@mahdi:~/Desktop/Robotics/project1/Hands_on/catkin_ws$ roslaunch turtlebot3_gazebo turtlebot3_detect_obstacles.launch
... logging to /home/mahdi/.ros/log/fc3b486a-dadc-11ed-b91f-adfb35253529/roslaunch-mahdi-8856.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.

xacro: in-order processing became default in ROS Melodic. You can drop the option.
started roslaunch server http://mahdi:36601/

SUMMARY
========

PARAMETERS
 * /gazebo/enable_ros_network: True
 * /robot_description: <?xml version="1....
 * /rosdistro: noetic
 * /rosversion: 1.15.15
 * /use_sim_time: True
```

سپس خروجی به صورت زیر خواهد بود:



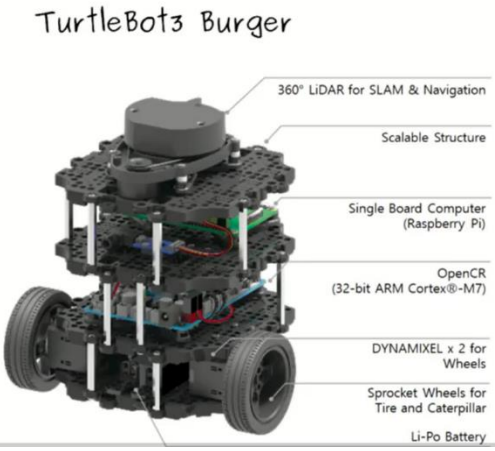
هندزان دوم) Mobile Robot Sensors and Control

ابتدا در رابطه با اینکه چگونه از سنسورهای ربات اطلاعات دریافت کنیم و اینکه چگونه ربات را به حرکت درآوریم و دستورات کنترلی دهیم صحبت میشود. سپس در رابطه با زوایای اویلر و کواترنیون صحبت میشود.

رباتی که کار میکنیم tutlebot3 میباشد و این ربات از بخش‌های مختلفی تشکیل شده.

Turtlebot3

- DYNAMIXEL: Moving wheels - odometry data.
- OpenCR: special arduino board designed for robotics. We can control out actuators here.
- Raspberry Pi : We put our codes here! requires ROS and an OS to be installed.
- LIDAR: For obstacle detection.



ربات از قسمت‌های مختلفی از قبیل لینک‌ها و جوینت‌ها و سنسورها تشکیل میشود و هرکدام یک فریم مخصوص دارند و اطلاعاتی هم که از هر بخش گرفته میشود در همان فریم بیان میشود. برای آنکه آن ها را تفسیر کنیم لازمه که به یک فریم مرکزی و اصلی برای ربات برده شوند.

Frames

- **base_link** is rigidly attached to the mobile robot base. It can be attached to the base in any arbitrary position or orientation.
- **odom** is a world-fixed frame. The pose of a mobile platform in the odom frame can drift over time, without any bounds. This drift makes the odom frame useless as a long-term global reference.
- The pose of a robot in the odom frame is guaranteed to be continuous, meaning that the pose of a mobile platform in the odom frame always evolves in a smooth way, without discrete jumps.
- In a typical setup the odom frame is computed based on an odometry source, such as wheel odometry, visual odometry or an IMU.

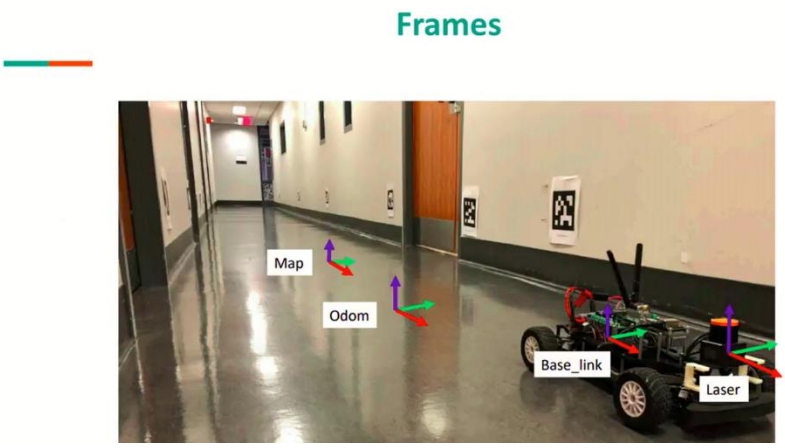
Base_link به مرکز ربات وصل میشود و درواقع اگر بخواهیم ربات را یک نقطه در نظر بگیریم این فریم بیانگر آن است. این فریم با ربات حرکت میکند و معمولا محور x آن بیانگر هدینگ ربات میباشد. فریم Odom یک فریم فیکس هست و حرکت نمیکند و یک نقطه به عنوان مرکز آن تعریف میشود. در این فریم در حقیقت pose (متشکل از position و orientation) ربات تعریف میشود. مثلا به کمک سنسورهای چرخ ها اطلاعات را میخوانیم و با محاسباتی میگوییم ربات ما الان کجا قرار دارد. ایراد آن هم دریفت و خطاش میباشد و برای فاصله های زیاد به علت انباشت خطا، اسفاده از این فریم گزینه مناسبی نمیباشد.

Frames

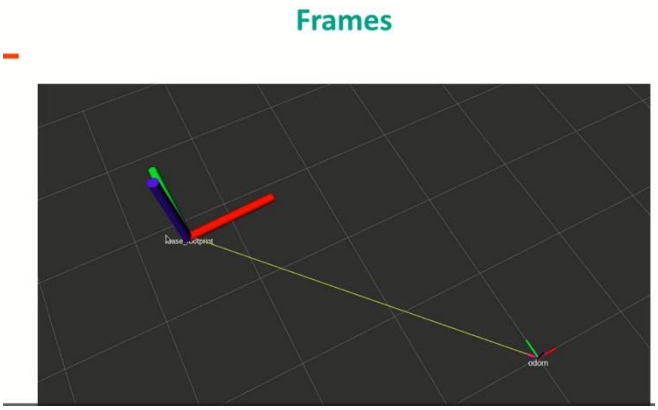
- **map** is a world fixed frame, with its Z-axis pointing upwards. The pose of a mobile platform, relative to the map frame, should not significantly drift over time. The map frame is not continuous, meaning the pose of a mobile platform in the map frame can change in discrete jumps at any time.
- **earth** is the origin of ECEF. This frame is designed to allow the interaction of multiple robots in different map frames. If the application only needs one map the earth coordinate frame is not expected to be present. REP 105

فریم دیگه فریم map میباشد. مثل Odom یک فریم فیکس است و حرکت ندارد. یه نقطه ای به عنوان مرکزش تعریف میکنیم و توی این فریم هم pose ربات تعریف میشود ولی فرقش با odom در اینه که pose رابت را دقیقتر به ما میدهد و منبعش هم یک الگوریتم localization است که میاد داده‌های چندین سنسور را باهم ترکیب میکند و موقعیت ربات را به ما میدهد. اطلاعاتی که میدهد هزینه بر است و برخلاف فریم Odom پرش دارد و برای مکان یابی در فواصل زیاد مناسبه نه فواصل کوتاه.

فریم دیگه فریم earth هست که مرکزش هسته کره زمینه. فریم هایی که تا الان تعریف شده دنیای یک ربات را داره تشکیل میده درحالی که ممکنه نیاز باشه چندین ربات داشته باشیم که باهم بخواهند interaction داشته باشند. همانطور که در زیر میبینید فریم map هرجایی در نقشه میتونه باشد ولی برای odom معمولا نقطه شروع ربات را به عنوان مرکز فریم در نظر میگیرند.

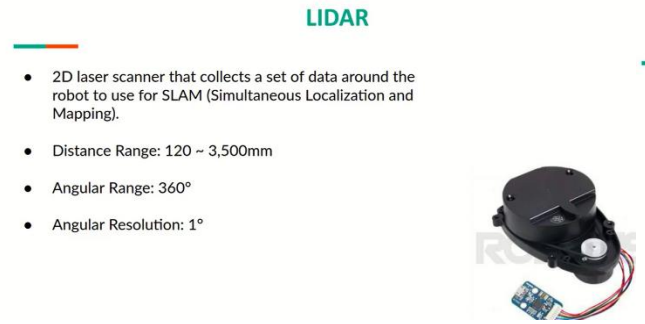


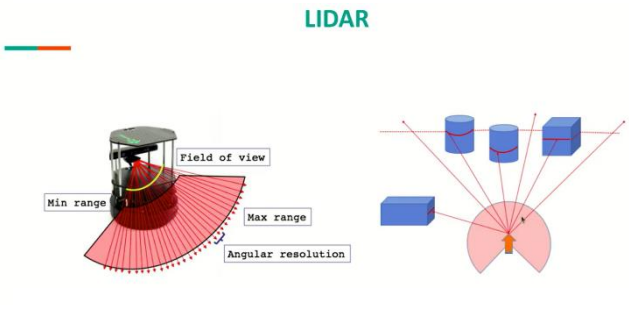
در تصویر زیر که از شبیه ساز rviz میباشد ما یک فریم به اسم base_footprint داریم که این هم به ربات چسبیده و با آن حرکت میکند. این فریم درواقع map شده‌ی فریم base link روی صفحه x و y میباشد درحالی که فریم base_link یه مقدار با صفحه xy فاصله دارد پس این z حذف میشود که برای راحتی کار و مثلا تبدیل به فریم odom راحت تر باشیم.



سنسور لایدار

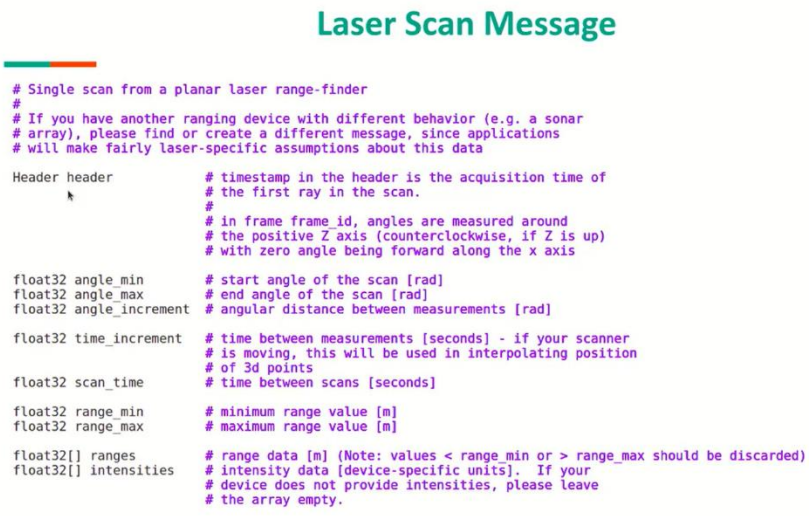
با پرتاب یک سری پرتوها در راستاهای مختلف و اندازه گیری زمان رفت و برگشت آن‌ها به ما میگوید در چه فاصله‌ای جسم از آن قرار دارد. ازش برای تشخیص مانع و همچنین نقشه برداری استفاده میشود.





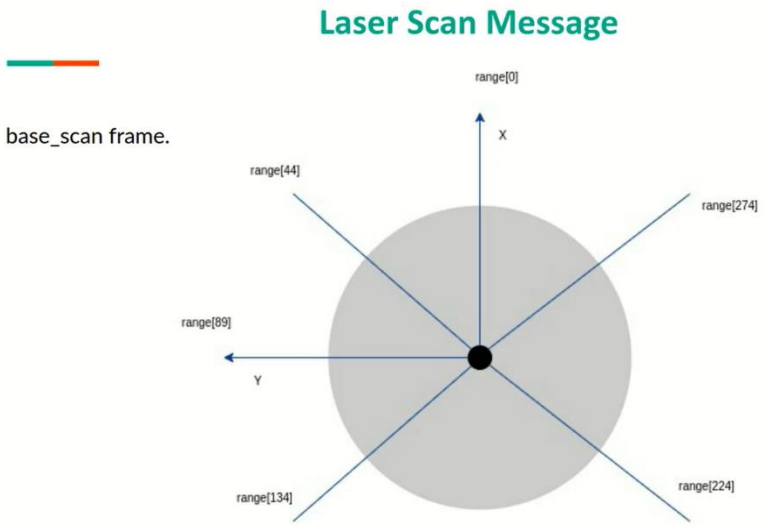
Laser Scan message

در تمرین قبلی با messageهای ROS آشنا شدیم و اگر بخواهیم اطلاعات را در شبکه نودهامون انتقال بدیم با نودها و تاپیک‌هاس و یاد گرفتیم چگونه یک مسیج کاستوم را تعریف کنیم.



بخش اول هدر هست که یک سری اطلاعات از اینکه در چه فریمی هستیم و درواقع توی چه فریم این داده میاد ازش و اینکه time stampش کِیه و کی این اسکن گرفته شده. دیگه angle_min که مینیم زاویه ای که داده اسکن میشه ازش و برحسب رادیانه و ماکسیممش هم که خب ماکسه. Angle_increment هم رزولوشن ما میباشد. range_min و range_max مقدار مینیمم و ماکسیمم فاصله ایست که میتوان موانع را تشخیص داد است. یک آرایه ranges داریم که فواصلی که به دست آمده را داخل آن قرار میدهیم.

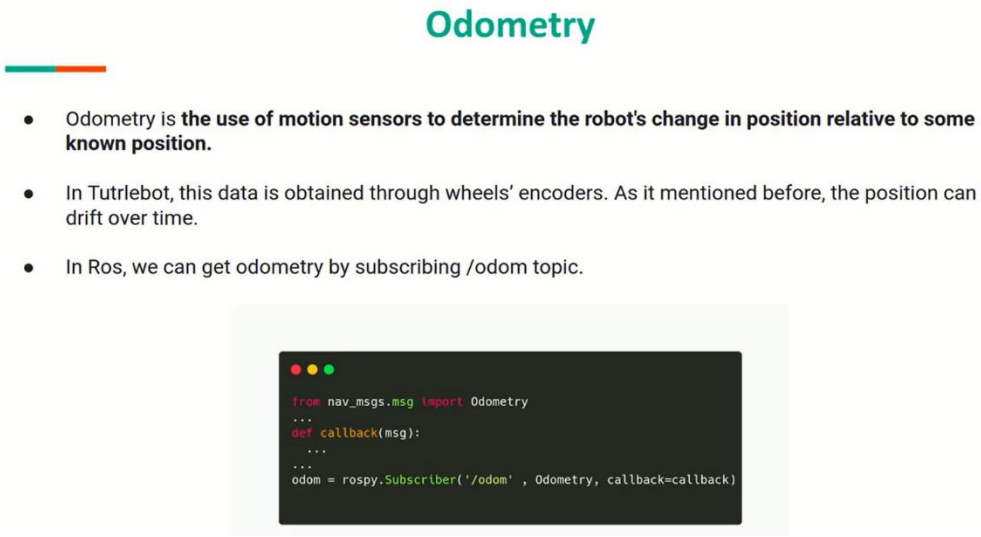
اطلاعات داخل فریم base scan که دقیقا بالای فریم base_link است ذخیره میشود و حالا چون رباتم دایره ای شکل هست لازم نیست دیگه اطلاعات رو به فریم base_link ببریم. در شکل زیر میتوانید چگونگی ذخیره داده ها در ایندکس‌های مختلف range را ببینید.



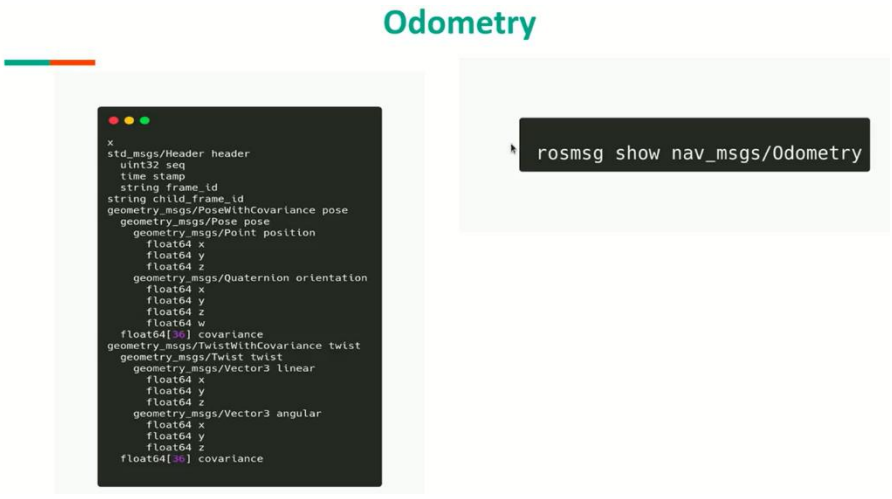
Odometry

ما میخوایم بیایم موقعیت ربات را بخونیم و بدونیم تو هر لحظه در مکانی قرار داریم. ما میایم داده‌های یک سری سنسورها را جمع آوری کرده و ازشون استفاده میکنیم تا موقعیت ربات را نسبت به یک نقطه‌ای به دست آوریم.

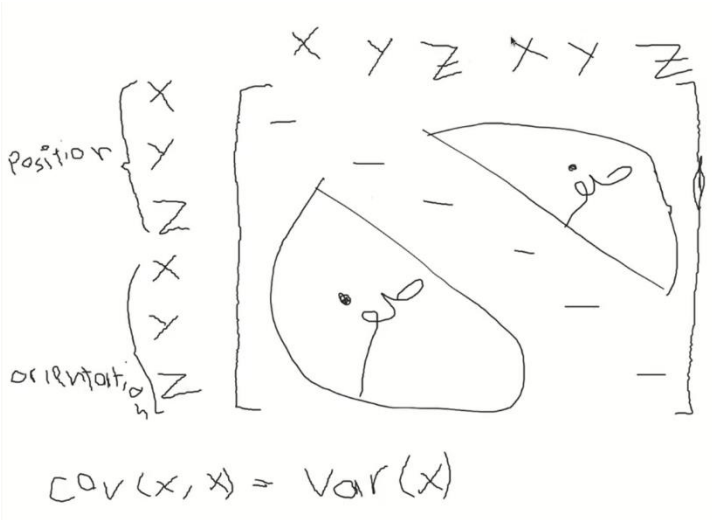
در ربات گفتیم میتوان از انکدر چرخ‌ها اطلاعات لازم را به دست آورد. در RoS میتوان اطلاعات را از تاپیک Odom به دست آورد. مسیج Odometry توی تاپیک odom پابلیش میشه و ما با سابسکرایب کردن میتونیم اونا رو به دست بیاریم.



حالا مسیج Odometry که برنامه نویسای این ربات تعریف کردن، به کمک دستور سمت راست در شکل پایین میتوان دید که مسیج ما چه شکلی دارد. دوتا آبجکت داریم که یکی PoseWithCovariance و دیگری TwistWithCovariance میباشد. که داخل همین اولی مثلا دوتا آبجکت Pose و Covariance را داریم. Pose که در حقیقت شامل موقعیت و Orientation ربات میباشد.

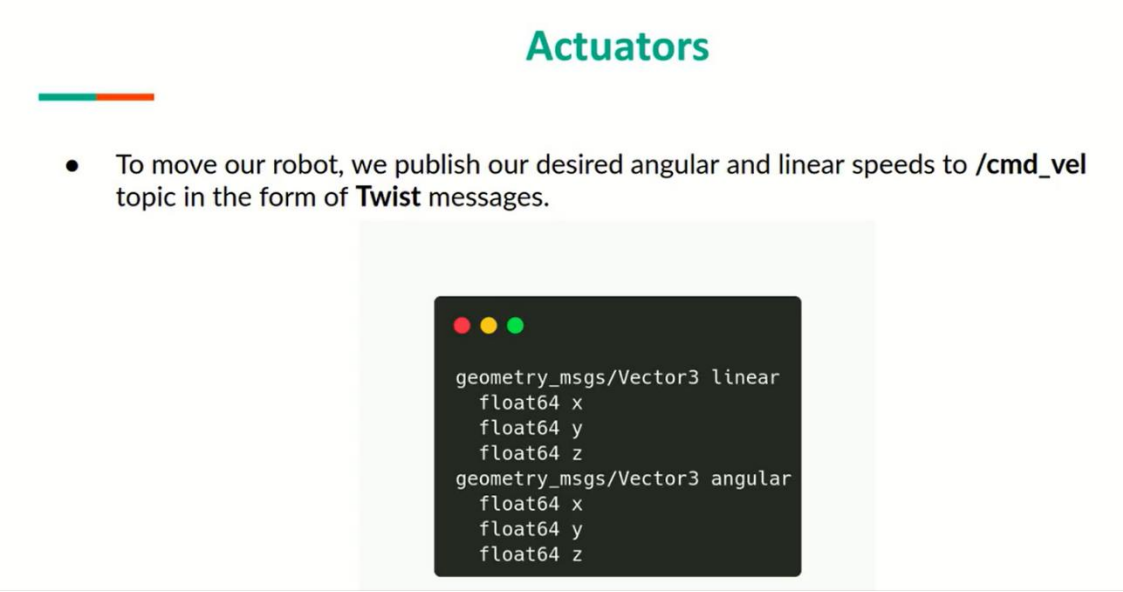


برای Covarince یک آرایه ۳۶ تایی داریم که یک ماتریس ۶ در ۶ قطری میباشد. این ماتریس هر درایه ش نشان دهده کواریانس بین هر دو پارامتر مثلا x و z میباشد. میگوید ما چه قدر به این سنسور اعتماد داریم. اگر مثلا برای درایه x و x مقدار ۰.۱ باشد یعنی اگر ۱ متر در جهت x جابه جا بشم، مقداری که نشان میدهد بین ۹۰ تا ۱۱۰ سانتی متر میباشد. و بعد برای turtle bot چون در جهت z نمیتوانیم حرکت کنیم در ماتریس براش یک مقدار زیاد قرار داده میشود.



آبجکت دیگه Twist بود که به کمک آن سرعت زاویه‌ای و خطی ربات را در هر لحظه داریم و کواریانس هم مثل قبل واسه اینجا هم تعریف میشود.

حال به سراغ actuatorها و اینکه چگونه ربات را به حرکت دربیاریم میرویم. برای فرستادن فرمان های کنترلی باید بیایم یه تاپیک درست کنیم به اسم cmd_vel و بیایم سرعت‌هایی که میخوایم را در قالب مسیج‌های twist و بیایم سرعت‌هایی که میخوایم را در قالب مسیج‌های twist پابلیش کنیم. اونطرف هم یه نودی هست که این رو سابسکرایب میکنه و این دستورات کنترلی را به لایه‌های پایین تر انتقال میدهد و باعث میشود ربات در جهت مورد نظر با سرعت مورد نظر حرکت کند.



Euler angles and Quaternions

هر روتیشنی در فضا را میتوان با چرخش‌هایی حول محورهای x و y و z انجام بدیم.

Euler Angles

The great eighteenth-century mathematician Leonhard Euler proved that an arbitrary three-dimensional rotation can be obtained by three individual rotations around the axes. In his honor, the angles of the rotations are called Euler angles.

Rotation around X axis

$$R_x = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & -sin\alpha \\ 0 & \sin\alpha & \cos\alpha \end{pmatrix}$$

Rotation around Y axis

$$R_y = \begin{pmatrix} \cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & \cos\beta \end{pmatrix}$$

Rotation around Z axis

$$R_z = \begin{pmatrix} \cos\gamma & -\sin\gamma & 0 \\ \sin\gamma & \cos\gamma & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

حال میخواهیم یک بازویی را کنترل کنیم که میتواند در ۳ جهت دوران داشته باشد. بعد اینا رو به یه ترتیبی در هم ضرب میکنیم که یک ماتریس واحد شود و ۳ تا دوران را درجا برای ما انجام دهد. خب حال اگر بتا را ۹۰ بگیریم و جاگذاری کنیم:

Gimbal Lock

We first want to rotate to $\beta = \frac{\pi}{2}$. Let $R = R_x R_y R_z$, which is:

$$R = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha \\ 0 & \sin\alpha & \cos\alpha \end{pmatrix} \begin{pmatrix} \cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & \cos\beta \end{pmatrix} \begin{pmatrix} \cos\gamma & -\sin\gamma & 0 \\ \sin\gamma & \cos\gamma & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Since $\cos\frac{\pi}{2} = 0$, and $\sin\frac{\pi}{2} = 1$, the above becomes:

$$R = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha \\ 0 & \sin\alpha & \cos\alpha \end{pmatrix} \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ -1 & 0 & 0 \end{pmatrix} \begin{pmatrix} \cos\gamma & -\sin\gamma & 0 \\ \sin\gamma & \cos\gamma & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Which equals:

$$R = \begin{pmatrix} 0 & 0 & 1 \\ \sin\alpha\cos\gamma + \sin\gamma\cos\alpha & \cos\gamma\cos\alpha - \sin\gamma\sin\alpha & 0 \\ -\cos\gamma\cos\alpha + \sin\gamma\sin\alpha & \sin\alpha\cos\gamma + \sin\gamma\cos\alpha & 0 \end{pmatrix}$$

خب بعد باتوجه به خواص تبدیل ضرب به جمع سینوس کسینوس ها و نوشتن روابط و ساده کردن ماتریس فوق به ماتریس زیر میرسیم و در این حالت اگر دقت شود تغییر آلفا یا گاما تاثیر یکسانی دارند و انگار ما یک درجه آزادی را از دست دادیم. به این پدیده Gimbal lock میگویند و باعث میشه بازوی ربات به حالت عجیبی بچرخه و روتیشن موردنظر را ندهد. در حقیقت اگر مثلا ترتیب xyz باشد و اون وسطی یعنی y به اندازه ۹۰ بچرخد دیگه محور x و z روی هم میوفته.

Gimbal Lock

Using the fact that:

$$\sin(\alpha \pm \gamma) = \sin\alpha\cos\gamma \pm \sin\gamma\cos\alpha$$
$$\cos(\alpha \pm \gamma) = \cos\gamma\cos\alpha \mp \sin\gamma\sin\alpha$$

We obtain:

$$R = \begin{pmatrix} 0 & 0 & 1 \\ \sin(\alpha+\gamma) & \cos(\alpha+\gamma) & 0 \\ -\cos(\alpha+\gamma) & \sin(\alpha+\gamma) & 0 \end{pmatrix}$$

As you can see, changing α or γ has the same effect!!! R is a rotation matrix around the z axis and modifying both parameters lead to the same change. As a result, we lose one degree of freedom. In order to gain more intuition watch this video.

راه حلی که برای این مشکل قرار داده شد، این بود که بیان نوع نمایش رو عوض کنند و از کواترنیون استفاده کنند.

Quaternions

As an alternative, we can use Quaternions which is another form of representation of rotations. A quaternion is a 4-tuple written formally as $q_0 + q_1 i + q_2 j + q_3 k$, where q_i are real numbers and the symbols i, j, k satisfy the following identities:

$$i^2 = j^2 = k^2 = -1$$
$$ij = k, \quad ji = -k$$
$$jk = i, \quad kj = -i$$
$$ki = j, \quad ik = -j$$

The conjugate and norm of a quaternion

Given $q = q_0 + q_1 i + q_2 j + q_3 k$, its conjugate is:

$$q_* = q_0 - q_1 i - q_2 j - q_3 k$$

and its norm is:

$$|q| = \sqrt{q_0^2 + q_1^2 + q_2^2 + q_3^2}$$

یک کواترنیون در حالتی میتواند یک روتیشن باشد که qR نرمش یک باشد که qR یک روتیشن را نمایش میدهد. کارش اینه که یک نقطه در فریم A را با روتیشن ببره روی فریم B . برای این کار از رابطه ای که در زیر آمده استفاده میشود.

Quaternions for rotations

A rotation is expressed by a quaternion q_R with the additional requirement that its norm $|q_R|$ be equal to 1. A rotation from one coordinate frame A to another B is given by the conjugation operation:

$$q_B = q_R q_A q_R^*$$

$$\begin{aligned} q_R\, q_A\, q_R^* &= (q_0 + q_1 i + q_2 j + q_3 k)(xi + yj + zk)(q_0 - q_1 i - q_2 j - q_3 k) \\ &= (x(q_0^2 + q_1^2 - q_2^2 - q_3^2) + 2y(q_1 q_2 - q_0 q_3) + 2z(q_0 q_2 + q_1 q_3))i + \\ &\quad (2x(q_0 q_3 + q_1 q_2) + y(q_0^2 - q_1^2 + q_2^2 - q_3^2) + 2z(q_2 q_3 - q_0 q_1))j + \\ &\quad (2x(q_1 q_3 - q_0 q_2) + 2y(q_0 q_1 + q_2 q_3) + z(q_0^2 - q_1^2 - q_2^2 + q_3^2))k. \end{aligned}$$

For better understanding of the functionality of each component of a quaternion, watch this video

حال برمیگردیم به بحث Odometry خودمون. باتوجه به مشکلات گفته شده تصمیم بر این شد که از کواترنیون استفاده شود. همانطور که در زیر در قسمت Pose و بعد در قسمت Orientation دیده میشود، به ترتیب x و y و z همان q1 تا q3 هستند و w همان q0 میباشد.

Representation Of Orientation In ROS

```
x
std_msgs/Header header
  uint32 seq
  time stamp
  string frame_id
string child_frame_id
geometry_msgs/PoseWithCovariance pose
  geometry_msgs/Pose pose
    geometry_msgs/Point position
      float64 x
      float64 y
      float64 z
    geometry_msgs/Quaternion orientation
      float64 x
      float64 y
      float64 z
      float64 w
  float64[36] covariance
geometry_msgs/TwistWithCovariance twist
  geometry_msgs/Twist twist
    geometry_msgs/Vector3 linear
      float64 x
      float64 y
      float64 z
    geometry_msgs/Vector3 angular
      float64 x
      float64 y
      float64 z
  float64[36] covariance
```

More information: nav_msgs/Odometry

الگوریتم های ما همشون با زوایای اویلرند. بیایم همونا رو باز با کواترنیون تعریف کنیم؟ خب خبر خوبه اینه که ما داریم روی فضای دو بعدی حرکت میکنیم و تنها روتیشنی که داریم حول محور z میباشد پس دیگه به مشکل gimbal lock نمیخوریم و لازم نیست الگوریتم ها را به کواترنیون ببریم ولی فقط کافیه اون کواتنیون هایی که توی Odometry میخونیم رو تبدیل کنیم به roll و pitch و yaw که با یه خط کد به صورت زیر هندل میشود و فقط هم yaw مهمه چون حول z فقط دوران داریم.

Euler From Quaternion

Fortunately, as you know Turtlebot rotates just around the z-axis. Therefore, we can use Euler angles for implementing our algorithms without worrying about gimbal lock. In ROS, we can convert Qunaterion to Euler by just a single line of code:

```
import tf
 ...
# returns (roll, pitch, yaw)
tf.transformations.euler_from_quaternion(my_quaternion)
```

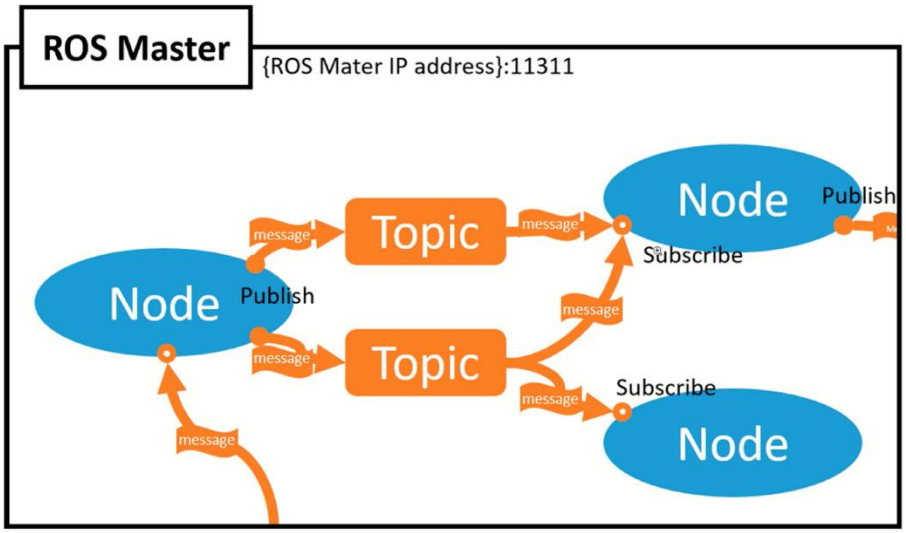
قسمت عملی

ربات با یه سرعتی جلو حرکت میکند و هرجا مانع جلوش ببیند، ۹۰ درجه به سمت راست میچرخد. میتوانید فیلم مربوطه را در ویدیوی ۴ مشاهده کنید. به لینک زیر نیز میتوانید مراجعه کنید:

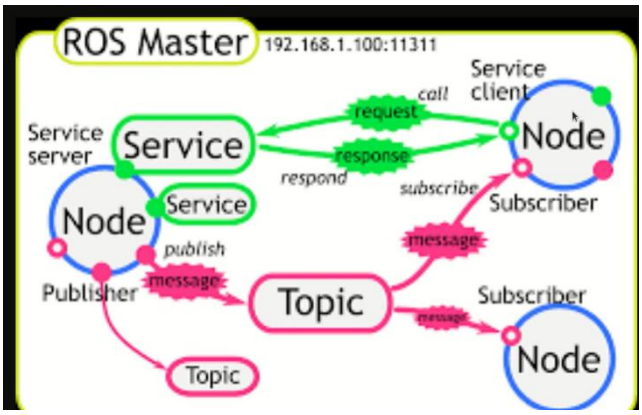
https://github.com/AlirezaAK2000/ros-tutorial

هندزان سوم) سرویس و launch file

در ویدیوی پنجم این مفاهیم را مشاهده خواهید کرد. در این هندزان در رابطه با سرویس و تفاوتش با تاپیک و اکشن صحبت میشود و در رابطه با لانچ فایل هم نکاتی گفته میشود. داخل یک پکیج راس تعدادی نود هستند که با رد وبدل کردن اطلاعات با هم دیگه کارشان را جلو میبرند. دقت شود که هر نود باید یک کار را هندل کند و نباید اینطور باشد که یک نود داخلش بیست تا کار انجام شود و از زیاد شدن نودها نترسید و در راس برای رابطه بین نودها فکر شده است. با تاپیک اشنا شدیم که روی آن یک سری مسیج ها منتشر میشد و یادگرفتیم چگونه یک سری مسیج ها را بسازیم. اما خب کجاها تاپیک و کجاها سرویس باید استفاده کرد؟ تاپیک از مدل پابلیش سابسکرایب استفاده میکنه. تعدادی نود میتوانند روی نود پابلیش کنند(؟) و تعدادی هم سابسکرایب کنند و درواقع یک ارتباط many to many ایجاد میکنه ولی سرویس اینطور نیست و اون اطلاعات یک به یک ایجاد میکند.



سرویس اولا توسط یه نود ارائه میشه که بهش نود سرور گویند. (چون سرویس ارائه میده بهش Service server گویند) نودهای دیگه میتونند این سرویس رو کال کنند و ازش استفاده کنند و به این صورته که نودی که کال میکنه بهش کلاینت گویند. (رابطه کلاینت سرور) در اینجا رابطه ها یک به یک اند.



نکته دیگر آنکه سرویس سنکرون است به این معنی که وقتی کلاینت درخواست میدهد منتظر میمونه تا ریسپانس رو بگیره. پس نباید پردازش زمانبری در سرویس باشد و نتونه جواب نود کلاینت را بدهد یا فرایندهای با ماهیت آسنکرون نباید در سرویس استفاده شود. مثلا حرکت ربات نباید داخل سرویس باشد حتی حرکت بازوی ربات هم باشه نیم ثانیه هم باشد توی کلاک کامپیوتر زمان زیادیه. برای این دست از کارا راس یه چیزی به نام اکشن داره که نحوه ساختنش مثل سرویسه و چیزای پیچیده ای ندارد و اینجا دیگه به صورت اسنکرون میباشد یعنی نود کلاینت وقتی اون اکشن رو کال میکنه متوقف نمیشه و به کارش ادامه میده و هروقت ریسپانسش اومد درواقع یک کال بک فانکشن داره و اون اجرا میشود. همچنین اکشن ها به صورت یک به یک اند. پس در سرویس نباید عملیات سنگین و زمانبر انجام شود تا سرویس همیشه در دسترس باشد و نودهای کلاینت متوقف نشوند.

در ادامه به پیاده سازی سرویس میپردازیم. در این لینک میتوانید آموزش خود راس را ببینید.

خب یدونه ورک اسپیس میسازیم و ربات را در یک مکانی که دور و برش دیواره قرار میدهیم و توی ورودی سرویس میگیم که سمت جلوی ربات و خروجی سرویس بگوید نزدیکترین مانع در چه فاصله ای میباشد. مثلا بگیم راست ربات چی بگه دو متر بگیم چپ ربات چی بگه ... . به ترتیب دستورات زیر را وارد کنید.

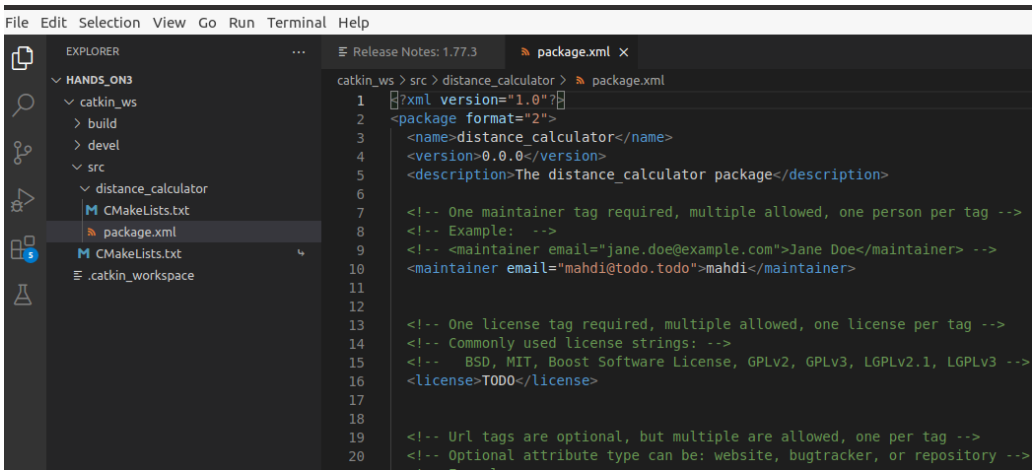
```
mkdir -p catkin_ws/src
cd catkin_ws/src/
catkin_init_workspace
cd ..
catkin_make
cd src/
catkin_create_pkg distance_calculator std_msgs sensor_msgs nav_msgs
```

```
mahdi@mahdi:~/Desktop/Robotics/project2/Hands_on3/catkin_ws$ cd src/
mahdi@mahdi:~/Desktop/Robotics/project2/Hands_on3/catkin_ws/src$ catkin_create_pkg distance_calculator std_msgs sensor_msgs nav_msgs
Created file distance_calculator/package.xml
Created file distance_calculator/CMakeLists.txt
Successfully created files in /home/mahdi/Desktop/Robotics/project2/Hands_on3/catkin_ws/src/distance_calculator. Please adjust the values in package.xml.
mahdi@mahdi:~/Desktop/Robotics/project2/Hands_on3/catkin_ws/src$
```

سپس وارد distance_calculator میشویم. اگر داخل آن را نگاه کنیم:

```
mahdi@mahdi:~/Desktop/Robotics/project2/Hands_on3/catkin_ws/src$ cd distance_calculator/
mahdi@mahdi:~/Desktop/Robotics/project2/Hands_on3/catkin_ws/src/distance_calculator$ ll
total 20
drwxrwxr-x 2 mahdi mahdi 4096 19:21 19   آوریل ./
drwxrwxr-x 3 mahdi mahdi 4096 19:21 19   آوریل ../
-rw-rw-r-- 1 mahdi mahdi 7155 19:21 19   آوریل CMakeLists.txt
-rw-rw-r-- 1 mahdi mahdi 2922 19:21 19   آوریل package.xml
mahdi@mahdi:~/Desktop/Robotics/project2/Hands_on3/catkin_ws/src/distance_calculator$
```

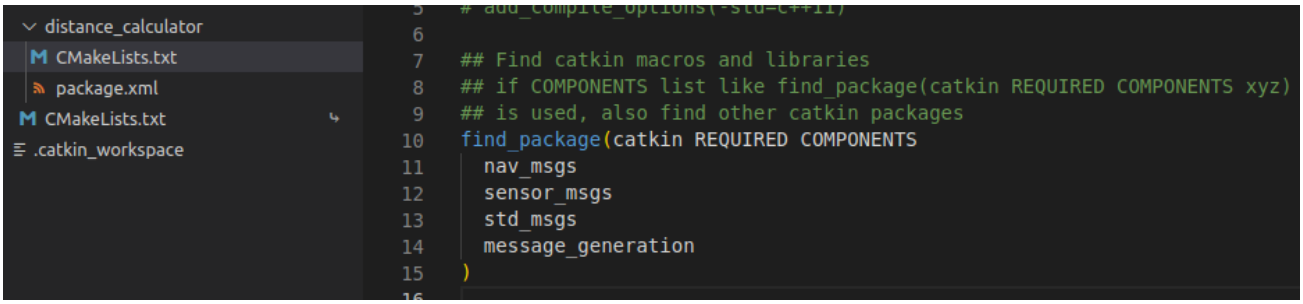
خب اینجا کاری که باید بکنیم خیلی شبیه کاریست که برای ساخت message جدید میکردیم. ابتدا این ورک اسپیس را درون vs code باز میکنیم:



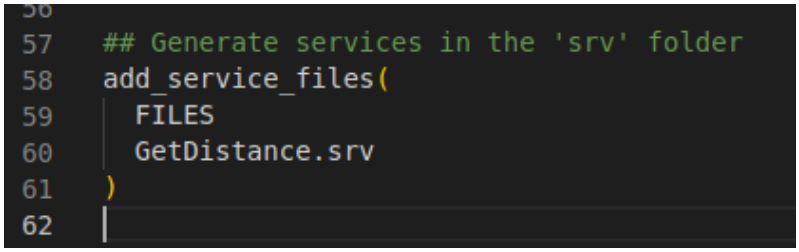
۱- در خط ۵۴ اینتر زده و بک build_depend جدید ایجاد کنیم و در خط ۶۲ هم یک exec_depend ایجاد میکنیم:

```
<!--    <doc_depend>doxygen</doc_depend> -->
<buildtool_depend>catkin</buildtool_depend>
<build_depend>nav_msgs</build_depend>
<build_depend>sensor_msgs</build_depend>
<build_depend>std_msgs</build_depend>
<build_depend>message_generation</build_depend>
<build_export_depend>nav_msgs</build_export_depend>
<build_export_depend>sensor_msgs</build_export_depend>
<build_export_depend>std_msgs</build_export_depend>
<exec_depend>nav_msgs</exec_depend>
<exec_depend>sensor_msgs</exec_depend>
<exec_depend>std_msgs</exec_depend>
<exec_depend>message_runtime</exec_depend>
```

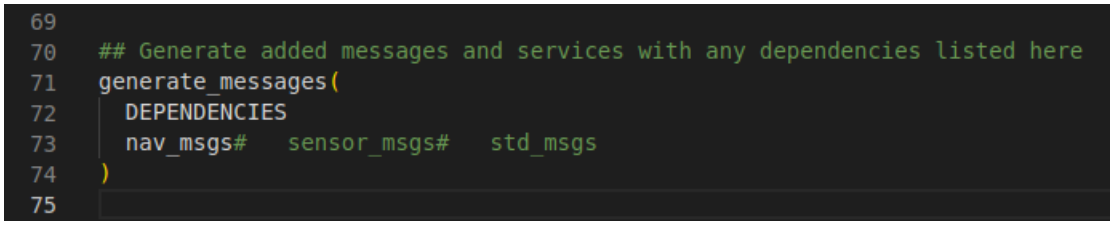
۲- سپس در CMakeLists هم در خط ۱۴ message_generation را میگذاریم:



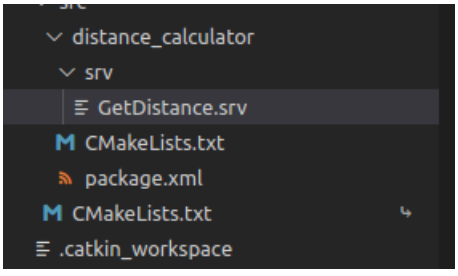
۳- خطوط ۵۸ تا ۶۲ که مربوط به سرویس هست را uncomment میکنیم و تغییرات را اعمال میکنیم:



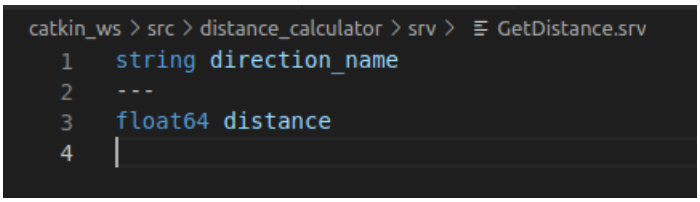
۴- خطوط ۷۱ تا ۷۴ هم uncomment شوند:



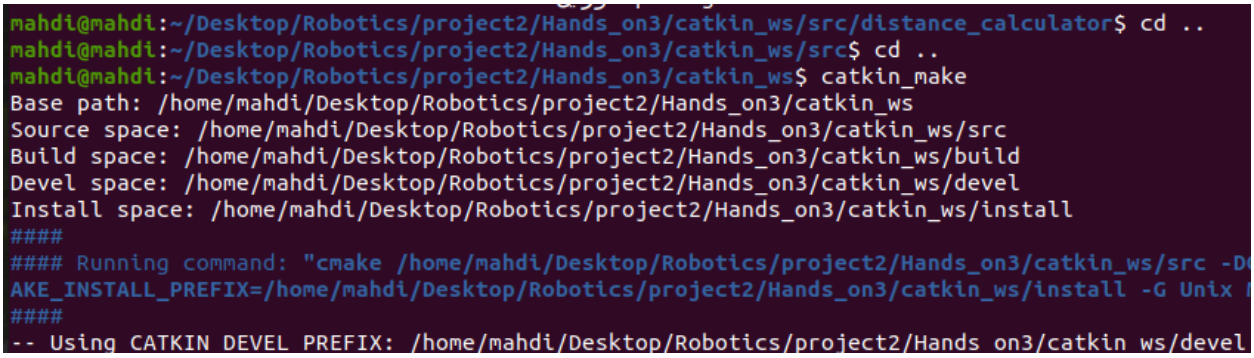
۵- برای آنکه کارمان تکمیل شود، در همان پوشه distance_calculator یک پوشه به نام srv درست میکنیم و بعد داخل آن یک فایل با نام اونی که در استپ ۳ مشخص کردیم میذاریم.



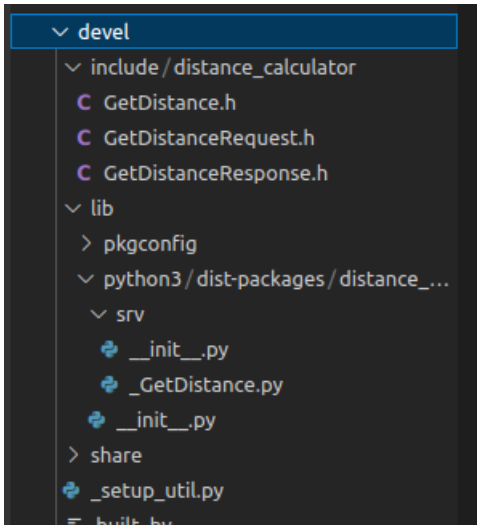
۶- توی فایل فوق لازمه که ورودی ها و خروجی های سرویس را بگوییم. بین ورودی ها و خروجی ها با --- جدا میشود. اگرم ورودی نداره خط اول همون --- میشود.



۷- حال به پوشه catkin_ws در ترمینال برگشته و catkin_make میکنیم.



دقت شود زمانی که فایل‌های package.xml و CMakeLists.txt تغییر میکنند باید catkin_make اجرا شود و فایل های مورد نیاز برای اجرا را در build و حتی devel تولید میکند. اگه c++ بودیم باید با هر تغییری یک بار catkin_make میکردیم که کامپایل بشه منتها برا پایتون نیاز نیست. اگر نگاه کنیم میبینیم فایلای زیر اضافه شدند.



حال در همان فولدر distance_calculator میبایم فولدر src میسازیم. بعد داخل آن فایل distance_calculator_node.py را اضافه میکنیم. بهتره اخره فایلاتون node_ بذارید که اگر نذارید ممکنه یه وقت با اسم پکیج قاطی بشه و موقع ایمپورت کردن داخل خود فایل پایتون ممکنه گیج بشه که میگیم از مثلا distance_calculator ایپورت کن خود فایل پایتونه منظور یا فولدر کلی.

در کد زیر برای فانکشن متعلق به سرویس دقت شود در سرویس اگر None برگردونیم راس به منزله‌ی Error میدونه اون رو و به کلاینتی که None گرفته میفهمونه که ارور دریافت کرده. همچنین در این تابع سعی شده است که پردازشی صورت نگیرد.

```python
#!/usr/bin/env python3
import rospy
from sensor_msgs.msg import LaserScan
from distance_calculator.srv import GetDistance, GetDistanceResponse

class Distance_calculator():
    def __init__(self) -> None:
        self.front_dis = -1
        self.left_dis = -1
        self.behind_dis = -1
        self.right_dis = -1

    def read_distance(self, data):
        ranges = data.ranges
        front_ranges = ranges[-5:0] + ranges[1:4]
        left_ranges = ranges[85:94]
        behind_ranges = ranges[175:184]
        right_ranges = ranges[265:274]

        self.front_dis = min(front_ranges)
        self.right_dis = min(right_ranges)
        self.behind_dis = min(behind_ranges)
        self.left_dis = min(left_ranges)

    def get_distance(self, req):
        direction_name = req.direction_name
        rospy.loginfo(f"NEW CALL: {direction_name}")
        distance = -1
        if direction_name == 'front':
            distance = self.front_dis
        elif direction_name == 'left':
            distance = self.left_dis
        elif direction_name == 'behind':
```

```
            distance = self.behind_dis
        elif direction_name == 'right':
            distance = self.right_dis
        else:
            rospy.logerr(f'direction_name is not valid: {direction_name}')
            return None
        res = GetDistanceResponse()
        res.distance = distance

        return res

def listener():

    rospy.init_node('distance_calculator_node', anonymous=True)
    dc = Distance_calculator()
    rospy.Subscriber("/scan", LaserScan, dc.read_distance)
    s = rospy.Service('get_distance', GetDistance, dc.get_distance)
    rospy.spin()

if __name__ == '__main__':
    listener()
```

سپس در تزمینال به صورت زیر عمل میکنیم:

```
mahdi@mahdi:~/Desktop/Robotics/project2/Hands_on3/catkin_ws$ cd src/distance_calculator/
mahdi@mahdi:~/Desktop/Robotics/project2/Hands_on3/catkin_ws/src/distance_calculator$ cd src/
mahdi@mahdi:~/Desktop/Robotics/project2/Hands_on3/catkin_ws/src/distance_calculator/src$ chmod +x distance_calculator_node.py
mahdi@mahdi:~/Desktop/Robotics/project2/Hands_on3/catkin_ws/src/distance_calculator/src$ cd ..
mahdi@mahdi:~/Desktop/Robotics/project2/Hands_on3/catkin_ws/src/distance_calculator$ cd ..
mahdi@mahdi:~/Desktop/Robotics/project2/Hands_on3/catkin_ws/src$ cd ..
mahdi@mahdi:~/Desktop/Robotics/project2/Hands_on3/catkin_ws$ . devel/setup.bash
mahdi@mahdi:~/Desktop/Robotics/project2/Hands_on3/catkin_ws$ rosrun distance_calculator distance_calculator_node.py
^Cmahdi@mahdi:~/Desktop/Robotics/project2/Hands_on3/catkin_ws$
```

اگر دستور rosrun بدون ارور اجرا شد پس اوکیه. بعد در حالی که اون ران شده در ترمینال دیگری میتوانید به صورت زیر عمل کنید تا از عملکرد و ساختار سرویس مطمئن شوید.

```
mahdi@mahdi:~/Desktop/Robotics/project2/Hands_on3/catkin_ws$ . devel/setup.bash
mahdi@mahdi:~/Desktop/Robotics/project2/Hands_on3/catkin_ws$ rosservice list
/distance_calculator_node_8182_1681925413671/get_loggers
/distance_calculator_node_8182_1681925413671/set_logger_level
/get_distance
/rosout/get_loggers
/rosout/set_logger_level
mahdi@mahdi:~/Desktop/Robotics/project2/Hands_on3/catkin_ws$ rossrv show distance_calculator/GetDistance
string direction_name
---
float64 distance

mahdi@mahdi:~/Desktop/Robotics/project2/Hands_on3/catkin_ws$
```

حالا یک دستور دیگه هست که میتوان خود سرویس را کال کرد و خروجی ازش گرفت. دقت شود در اینجا چون رباتی اضافه نکردیم و gazebo ای ران نشده همون مقدار دیفالت ۱- را برمیگرداند: (تب بزنیم خودش فیلد رو میاره پر کنیم) و ورودی اشتباه هم بدیم میفهمه که None گرفتیم.

```
mahdi@mahdi:~/Desktop/Robotics/project2/Hands_on3/catkin_ws$ rosservice call /get_distance "direction_name: 'front'"
distance: -1.0
mahdi@mahdi:~/Desktop/Robotics/project2/Hands_on3/catkin_ws$ rosservice call /get_distance "direction_name: 'frontt'"
ERROR: service [/get_distance] responded with an error: b'service cannot process request: service handler returned None'
mahdi@mahdi:~/Desktop/Robotics/project2/Hands_on3/catkin_ws$
```

همچنین در ترمینال دیگر هم به واسطه لاگی که نوشته بودیم ارور چاپ شد:

```
^Cmahdi@mahdi:~/Desktop/Robotics/project2/Hands_on3/catkin_ws$ rosrun distance_calculator distance_calculator_node.py
[INFO] [1681925987.642054]: NEW CALL: front
[INFO] [1681926097.799441]: NEW CALL: frontt
[ERROR] [1681926097.801215]: direction_name is not valid: frontt
```

حال میرویم سراغ اضافه کردن ربات:

ابتدا یک پوشه لانچ درون فولدر distance_calculator درست کرده و داخل آن distance_calculator.launch را اضافه میکنیم. محتویات آن به صورت زیر خواهد بود: (فایل custom_world.launch جلوتر توضیح داده شده است.)

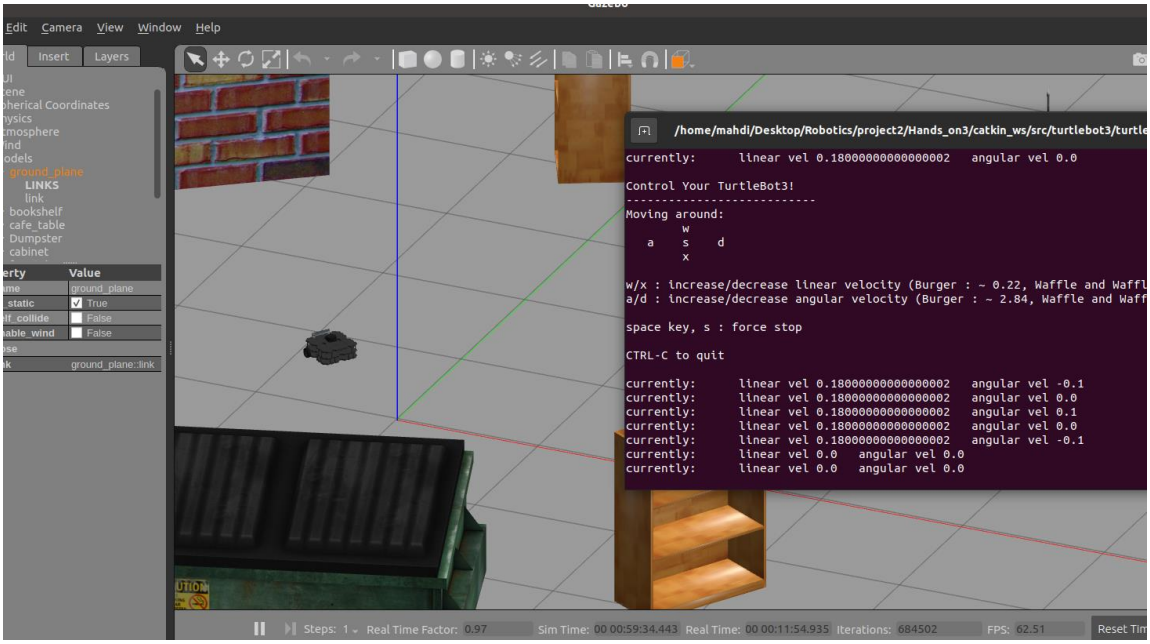
```
<launch>

    <node pkg="distance_calculator" type="distance_calculator_node.py"
name="distance_calculator_node" output="screen"></node>

    <include file="$(find turtlebot3_gazebo)/launch/custom_world.launch">
        <arg name="model" value="waffle"/>
        <arg name="world_name_file" value=" /home/mahdi/Desktop/Robotics/project2/detect_obstacles.world"/>
        <arg name="x_pos" value="0.0"/>
        <arg name="y_pos" value="0.0"/>
        <arg name="z_pos" value="0.0"/>
        <arg name="yaw" value="1.57"/>
    </include>

    <!-- <include file="$(find turtlebot_rviz_launchers)/launch/view_robot.launch/>"-->

</launch>
```

سپس دستورهای زیر را در یک ترمینال اجرا میکنیم:

```
mahdi@mahdi:~/Desktop/Robotics/project2$ . Hands_on3/catkin_ws/devel/setup.bash
mahdi@mahdi:~/Desktop/Robotics/project2$ export TURTLEBOT3_MODEL="waffle"
mahdi@mahdi:~/Desktop/Robotics/project2$ export TURTLEBOT3_GAZEBO_WORLD_FILE=detect_obstacles.world
mahdi@mahdi:~/Desktop/Robotics/project2$ roslaunch distance_calculator distance_calculator.launch
... logging to /home/mahdi/.ros/log/c7d92972-ded6-11ed-b0f7-f1b2141c868c/roslaunch-mahdi-32665.log
Checking log directory for disk usage. This may take a while.
Press Ctrl-C to interrupt
```

حال در یک ترمینال دیگر لازم است تا teleop را بالا بیاوریم تا ربات را کمی جا به جا کرده و مقادیر را بخوانیم.

```
mahdi@mahdi:~$ cd Desktop/Robotics/project2/Hands_on3/catkin_ws/
mahdi@mahdi:~/Desktop/Robotics/project2/Hands_on3/catkin_ws$ . devel/setup.bash
mahdi@mahdi:~/Desktop/Robotics/project2/Hands_on3/catkin_ws$ export TURTLEBOT3_MODEL=waffle
mahdi@mahdi:~/Desktop/Robotics/project2/Hands_on3/catkin_ws$ roslaunch turtlebot3_teleop turtlebot3_teleop_key.launch
```



میتوان به کمک rosservice call فاصله را بخوانیم. برای این منظور در ترمینالی دیگر دستورهای زیر را وارد میکنیم. در حالت اول چون جلوی ربات مانعی نبوده inf داده ولی با جابه جا کردن آن و گرفتن مجدد خروجی مقدار عددی برگرداند.

```
mahdi@mahdi:~/Desktop/Robotics/project2/Hands_on3/catkin_ws$ . devel/setup.bash
mahdi@mahdi:~/Desktop/Robotics/project2/Hands_on3/catkin_ws$ cd src/distance_calculator/
mahdi@mahdi:~/Desktop/Robotics/project2/Hands_on3/catkin_ws/src/distance_calculator$ rosservice call /get_distance "direction_name: 'front'"
distance: inf
mahdi@mahdi:~/Desktop/Robotics/project2/Hands_on3/catkin_ws/src/distance_calculator$ rosservice call /get_distance "direction_name: 'front'"
distance: 2.993802785873413
mahdi@mahdi:~/Desktop/Robotics/project2/Hands_on3/catkin_ws/src/distance_calculator$
```

حال به سراغ توضیح فایلای لانچ برویم:

فایلای لانچ درواقع به اجرای پکیجمون کمک میکند یه سری گایدلایناس برای اینکه پکیج اجرا شود و ... . داخل work space یک پکیج به اسم turtlebot3_simulations داریم که داخلش turtlebot3_gazebo را داریم که معمولا زمانی که بخواهیم turtlebot را داخل gazebo بذاریم، با اون اجرا میکنیم و دوباره توی اون یک فولدر لانچ داره که اگه به لانچ فایلای درونش نگاه کنیم، تعداد زیادی اند و میتوان همشون رو هم تست کرد، اما base آن همون empty_world است، نگاه کنیم یک ساختار XML دارد و نکات مد نظر را داخلش میگه.

اولا این لانچ فایل ها درون خودشون میتوانند لانچ فایلای دیگه رو صدا کنند، همچنین یک سری argument میتوانند داشته باشند. حالا مثلا در فایل empty اگر خط ۲ تا ۵ داره یک سری ارگومان تعریف میشه که یه سری مقدار دیفالت هم براشون مشخص شده است. در خطوط ۸ تا ۱۳ که دوباره مقدار یک سری آرگومان ست شده است در حقیقت مقادیر آرگومان های لانچ فایلی است که در خط ۷ گفته و اگه به اون لانچ فایل بریم مثل خطوط ۲ تا ۵ همین فایل یک سری آرگومان ها تعریف شده که حالا از اینجا در خطوط ۸ تا ۱۳ داریم مقادیرشون رو ست میکنیم.

توی خط ۱۸ هم درواقع اومده گفته فلان نود را ران کند و یه خط کد کامندلاینی هم تعریف کرده که اگه در CLI مقادیرش ست شود جای همون آرگومان های خط ۲ تا ۵ مینشینند.

```
1 <launch>
2   <arg name="model" default="$(env TURTLEBOT3_MODEL)" doc="model type [burger, waffle, waffle_pi]"/>
3   <arg name="x_pos" default="0.0"/>
4   <arg name="y_pos" default="0.0"/>
5   <arg name="z_pos" default="0.0"/>
6
7   <include file="$(find gazebo_ros)/launch/empty_world.launch">
8     <arg name="world_name" value="$(find turtlebot3_gazebo)/worlds/empty.world"/>
9     <arg name="paused" value="false"/>
10    <arg name="use_sim_time" value="true"/>
11    <arg name="gui" value="true"/>
12    <arg name="headless" value="false"/>
13    <arg name="debug" value="false"/>
14  </include>
15
16  <param name="robot_description" command="$(find xacro)/xacro --inorder $(find turtlebot3_description)/urdf/turtlebot3_$(arg model).urdf.xacro" />
17
18  <node pkg="gazebo_ros" type="spawn_model" name="spawn_urdf" args="-urdf -model turtlebot3_$(arg model) -x $(arg x_pos) -y $(arg y_pos) -z $(arg z_pos) -param
  robot_description" />
19 </launch>
```

حال در اینجا فقط x و y و z ربات تعیین میشه ولی در مثالی که قبلتر اوردیم لازم بود زاویه اولیه yaw ربات هم تعیین شود. برای این منظور در همین خط ۱۸ باید مثلا یه Y- (که با y- فرق داره) اضافه کنیم. خب حالا بنابراین یک کپی از همین میگیریم و در همان فولدر launch که گفتیم فایل custom_world.launch را اضافه کنید.

```
<launch>
  <arg name="model" default="$(env TURTLEBOT3_MODEL)" doc="model type [burger,
waffle, waffle_pi]"/>
  <arg name="world_name_file" default="$(env TURTLEBOT3_GAZEBO_WORLD_FILE)"/>
  <arg name="x_pos" default="0.0"/>
  <arg name="y_pos" default="0.0"/>
  <arg name="z_pos" default="0.0"/>
  <arg name="yaw" default="0.0"/>


  <include file="$(find gazebo_ros)/launch/empty_world.launch">
    <arg name="world_name" value="$(arg world_name_file)"/>
    <arg name="paused" value="false"/>
    <arg name="use_sim_time" value="true"/>
    <arg name="gui" value="true"/>
    <arg name="headless" value="false"/>
    <arg name="debug" value="false"/>
  </include>

  <param name="robot_description" command="$(find xacro)/xacro --inorder $(find
turtlebot3_description)/urdf/turtlebot3_$(arg model).urdf.xacro" />

  <node pkg="gazebo_ros" type="spawn_model" name="spawn_urdf" args="-urdf -model
turtlebot3_$(arg model) -x $(arg x_pos) -y $(arg y_pos) -z $(arg z_pos) -Y $(arg
yaw) -param robot_description" />
</launch>
```