Automation and Robotics Engineering

ROBOTICS LAB

HOMEWORK 4

Control a mobile robot to follow a trajectory

Instructor:
Mario Selvaggio

Student:
Francesco Grasso
P38000046

Accademic Year 2023/2024

1. Construct a gazebo world and spawn the mobile robot in a given pose

1.a) Launch the Gazebo simulation and spawn the mobile robot in the world rl_racefield in the pose

x= -3 , y=5 , yaw = -90deg

with respect to the map frame. The argument for the yaw in the call of spawn_model is Y.

```xml
  <!-- 1.a) Pose Modified -->
  <arg name="x_pos" default="-3.0"/>
  <arg name="y_pos" default="5.0"/>
  <arg name="z_pos" default="0.1"/>
  <arg name="yaw_pos" default="-1.57"/>
  <env name="GAZEBO_MODEL_PATH" value="$(find rl_racefield)/models:$(optenv GAZEBO_MODEL_PATH)"/>

  <!-- We resume the logic in empty_world.launch -->
  <include file="$(find gazebo_ros)/launch/empty_world.launch">
    <arg name="world_name" value="$(find rl_racefield)/worlds/rl_race_field.world" />
    <arg name="debug" value="$(arg debug)" />
    <arg name="gui" value="$(arg gui)" />
    <arg name="paused" value="$(arg paused)"/>
    <arg name="use_sim_time" value="$(arg use_sim_time)"/>
    <arg name="headless" value="$(arg headless)"/>
  </include>


<!-- urdf xml robot description loaded on the Parameter Server-->

  <param name="robot_description" command="$(find xacro)/xacro '$(find rl_fra2mo_description)/urdf/fra2mo.xacro'" />

  <!-- Run a python script to the send a service call to gazebo_ros to spawn a URDF robot -->
  <!-- 1.a) Yaw added -->
  <node name="urdf_spawner" pkg="gazebo_ros" type="spawn_model" respawn="false" output="screen"
  args="-urdf -model fra2mo -x $(arg x_pos) -y $(arg y_pos) -z $(arg z_pos) -Y $(arg yaw_pos) -param robot_description"/>
```

Figura 1: Mobile robot's pose modified

The values of **x_pos, y_pos and z_pos** have been modified adding a new argument: **yaw_pos**.

Yaw position has also been added as an argument to **urdf_spawner**.

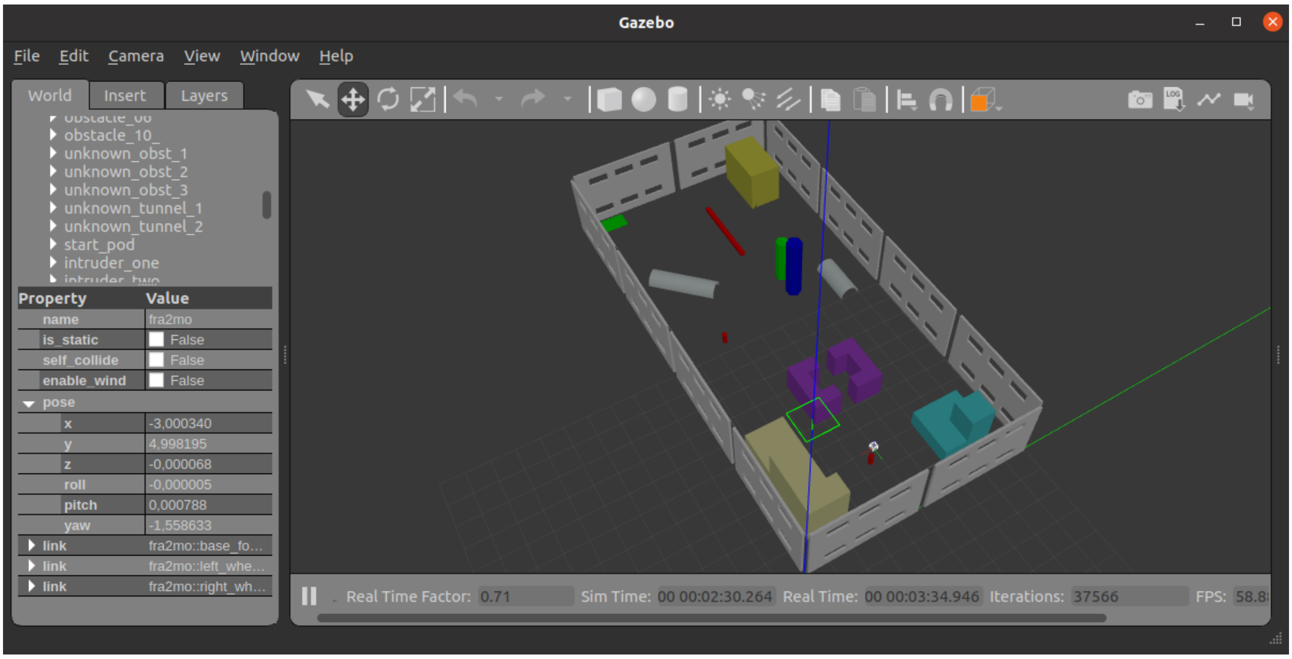

Figura 2: Mobile robot in rl_racefield world

1.b) Modify the world file of rl_racefield moving the obstacle 9 in position:

x = -17, y=9, z= 0.1, yaw=3.14

```
<!-- 1.b) position of obstacle 9 modified -->
<include>
  <name>obstacle_09</name>
  <pose> -17 9 0.1 0 0 3.14159</pose>
  <uri>model://obstacle_09</uri>
</include>
```

Figura 3: Object 9 position modified in rl_race_field.world

1.c) Place the ArUco marker number 115 on obstacle 9 in an appropriate position, such that it is visible by the mobile robot's camera when it comes in the proximity of the object.

A new aruco marker (115) was generated using the online aruco generator available at **https://chev.me/arucogen/**.

The following steps have been followed:

1) A new folder **marker_new** has been created in:

catkin_ws/src/rl_racefield/models.

2) The files **marker_new.sdf**, **model.config** and **marker_new.material** have been created taking the markers already created as an example and have been placed in the folder **marker_new** and in **marker_new/material/scripts** folder respectively.

3) The marker in png format has been added in:

catkin_ws/src/rl_racefield/models/marker_new/material/textures.

In order to spawn the marker on the obstacol 9, its position in the rl_race_field.world file was modified like shown in the figure:

```
<!-- AR markers -->
<include>
  <name>tool_0_tag</name>
  <uri>model://marker_new</uri>
  <pose>-17 7.7 0.21 0 1.57 3.14</pose>
</include>
```

Figura 4: Aruco Marker position

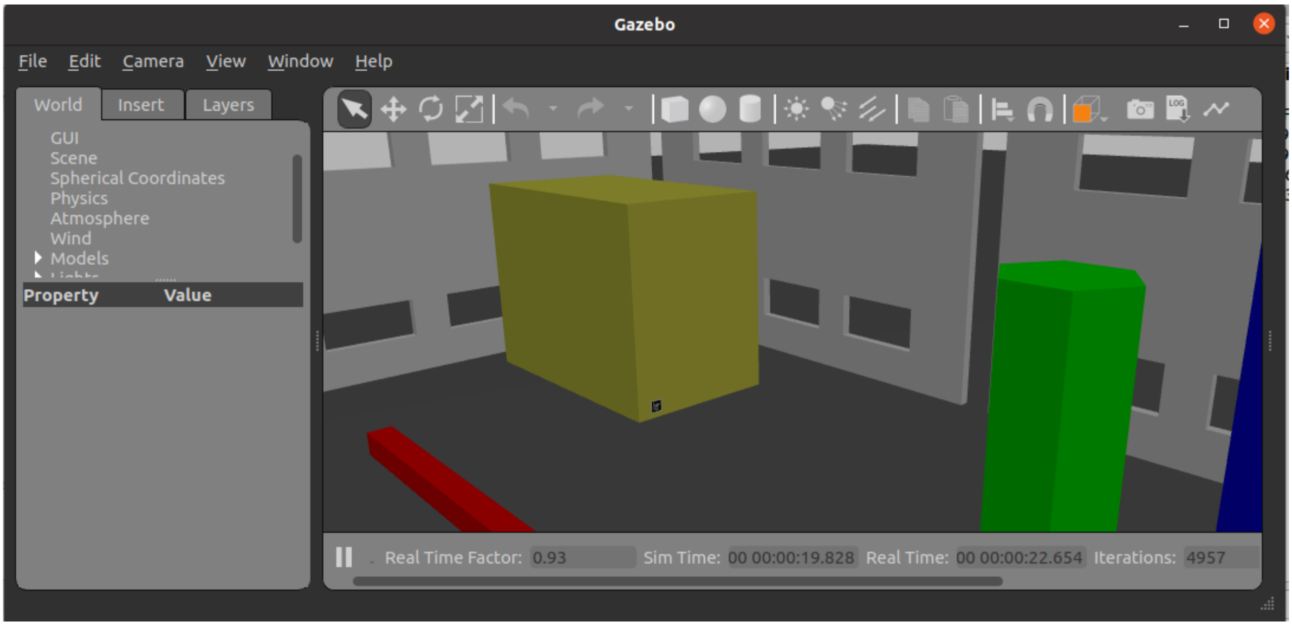

Figura 5: Aruco Marker in the Gazebo environment

2. **Place static tf acting as goals and get their pose to enable an autonomous navigation task**

2.a) Insert 4 static tf acting as goals in the following poses with respect to the map frame:

- **Goal 1: x = -10 , y = 3 , yaw = 0 deg**
- **Goal 2: x = -15, y = 7 ,yaw = 30 deg**
- **Goal 3: x = 6, y = 8 ,yaw = 180 deg**
- **Goal 4: x = 17.5, y = 3 ,yaw = 75 deg**

Follow the example provided in the launch file **rl_fra2mo_description/launch /spawn_fra2mo_gazebo.launch** of the simulation.

```
<!-- 2.a) static tf as goals -->
 <node pkg="tf" type="static_transform_publisher" name="goal_1_pub" args="-10   3 0 0 0 0 1 map goal1 100" />
 <node pkg="tf" type="static_transform_publisher" name="goal_2_pub" args="-15   7 0 0 0 0.2588 0.9659 map goal2 100" />
 <node pkg="tf" type="static_transform_publisher" name="goal_3_pub" args="-6    8 0 0 0 1 0 map goal3 100" />
 <node pkg="tf" type="static_transform_publisher" name="goal_4_pub" args="-17.5 3 0 0 0 0.6087 0.7933 map goal4 100" />
```

Figura 6: static_transform_publishers (rotation expressed in quaternion)

2.b) **Following the example code in fra2mo_2dnav/src/tf_nav.cpp, implement tf listeners to get target poses and print them to the terminal as debug.**

```cpp
//2.b) TF listeners ////////////////////////


void TF_NAV::goal_listener_1() {
    ros::Rate r( 1 );
    tf::TransformListener listener;
    tf::StampedTransform transform;

    while ( ros::ok() )
    {
        try
        {
            listener.waitForTransform( "map", "goal1", ros::Time( 0 ), ros::Duration( 10.0 ) );
            listener.lookupTransform( "map", "goal1", ros::Time( 0 ), transform );
        }
        catch( tf::TransformException &ex )
        {
            ROS_ERROR("%s", ex.what());
            r.sleep();
            continue;
        }

        _goal1_pos << transform.getOrigin().x(), transform.getOrigin().y(), transform.getOrigin().z();
        _goal1_or << transform.getRotation().w(),  transform.getRotation().x(), transform.getRotation().y(), transform.getRotation().z();

        //ROS_INFO("Goal Position: %f %f %f", _goal1_pos[0], _goal1_pos[1], _goal1_pos[2]);
        //ROS_INFO("Goal Orientation: %f %f %f %f", _goal1_or[0], _goal1_or[1], _goal1_or[2], _goal1_or[3]);

        r.sleep();
    }
}
```

Figura 7: example of tf_listener relative to the goal1

```
transforms:
  -
    header:
      seq: 0
      stamp:
        secs: 54
        nsecs:  36000000
      frame_id: "map"
    child_frame_id: "goal1"
    transform:
      translation:
        x: -10.0
        y: 3.0
        z: 0.0
      rotation:
        x: 0.0
        y: 0.0
        z: 0.0
        w: 1.0
```

Figura 8: goal1 tf example with the command rostopic echo /tf

2.c) **Using move_base, send goals to the mobile platform in a given order. Go to the next one once the robot has arrived at the current goal. The order of the explored goals must be Goal 3 → Goal 4 → Goal 2 → Goal 1. Use the Action Client communication protocol to get the feedback from move_base. Record a bagfile of the executed robot trajectory and plot it as a result.**

```cpp
void TF_NAV::Goal( Eigen::Vector3d& goal_pos,  Eigen::Vector4d& goal_or, int goal_number) {

    move_base_msgs::MoveBaseGoal goal;

    MoveBaseClient ac("move_base", true);
    while(!ac.waitForServer(ros::Duration(5.0))){
        ROS_INFO("Waiting for the move_base action server to come up");
    }

    goal.target_pose.header.frame_id = "map";
    goal.target_pose.header.stamp = ros::Time::now();

    goal.target_pose.pose.position.x = goal_pos[0];
    goal.target_pose.pose.position.y = goal_pos[1];
    goal.target_pose.pose.position.z = goal_pos[2];
    goal.target_pose.pose.orientation.w = goal_or[0];
    goal.target_pose.pose.orientation.x = goal_or[1];
    goal.target_pose.pose.orientation.y = goal_or[2];
    goal.target_pose.pose.orientation.z = goal_or[3];

    ROS_INFO("Sending Goal %d", goal_number);
    ac.sendGoal(goal);
    ac.waitForResult();

    if(ac.getState() == actionlib::SimpleClientGoalState::SUCCEEDED) {
        ROS_INFO("The mobile robot arrived at the Goal %d", goal_number);
         }

    else{
        ROS_INFO("The base failed to move for some reason");
    }
    }
```

Figura 9: Goal function

In the figure above, a **Goal Function** is implemented.

It takes the position and orientation of a goal with its number.

The function creates a **MoveBaseClient object** and a

move_base_msgs::MoveBaseGoal object.

Next, it enters a while loop that waits for the move_base action server to be ready.

A goal is sent to the move_base server using the ac client via the **sendGoal(goal)** method. Next, the program waits for the result of executing the action with **ac.waitForResult()**.

If the action was successful, a message is printed indicating that the mobile robot has arrived at the specified goal.

```
void TF_NAV::send_goal() {
    ros::Rate r( 5 );
    int cmd;
    int count;
    move_base_msgs::MoveBaseGoal goal;

    while ( ros::ok() )
    {

        std::cout<<"\nInsert 1 to set the order: Goal3 ----> Goal4 ----> Goal2 -----> Goal1 "<<std::endl;
        std::cout<<"\nInsert 2 to set the order: Goal5 ----> Goal6 ----> Goal7 "<<std::endl;
        std::cout<<"\nInsert 3 to send the robot in the proximity of obstacle 9 (Goal8) "<<std::endl;


        std::cin>>cmd;

        if ( cmd == 1) {

        //2.c)
        TF_NAV::Goal(_goal3_pos,_goal3_or,3);
        TF_NAV::Goal(_goal4_pos,_goal4_or,4);
        TF_NAV::Goal(_goal2_pos,_goal2_or,2);
        TF_NAV::Goal(_goal1_pos,_goal1_or,1);

        }
        else if (cmd ==2){

        //3.a)
        TF_NAV::Goal(_goal5_pos,_goal5_or,5);
        TF_NAV::Goal(_goal6_pos,_goal6_or,6);
        TF_NAV::Goal(_goal7_pos,_goal7_or,7);

        }
```

Figura 10: send_goal function

In the function above,the first choice allows to send goals to the mobile platform in the order specified, calling the **Goal function** for each goal.

A bagfile has been recorded with the command **rosbag record /fra2mo/pose**.

The tool **rqt_bag** was used to plot it:

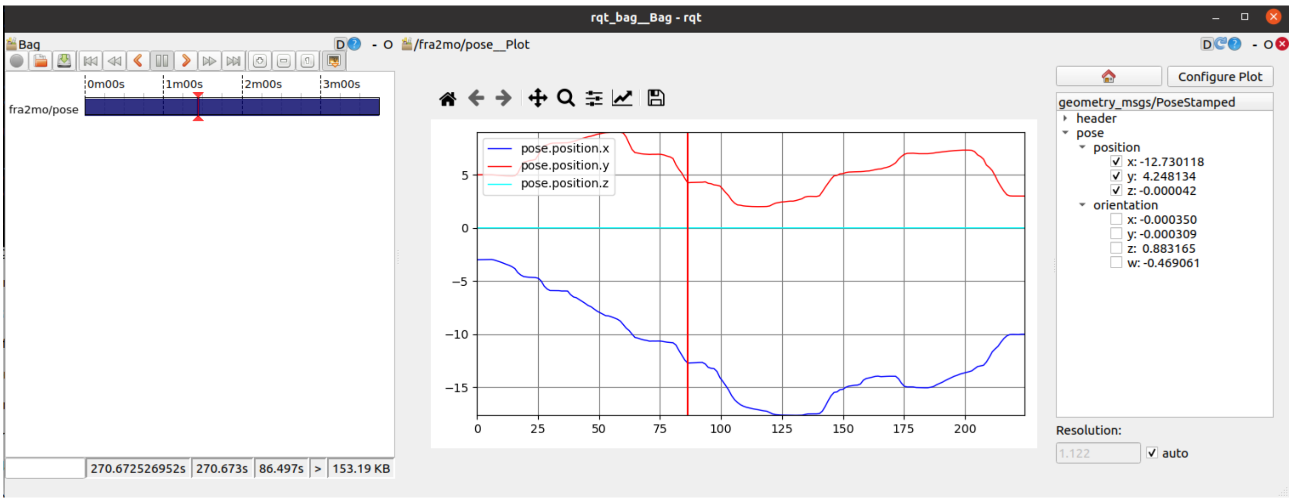

Figura 11: position of the robot on rqt_bag

3. . Map the environment tuning the navigation stack's parameters

3.a) Modify, add, remove, or change pose, the previous goals to get a complete map of the environment

With the same method shown in the point 2.a) three goals were added:

```
<!-- 3.a) static tf as goals to get a complete map of the environment -->
<node pkg="tf" type="static_transform_publisher" name="goal_5_pub" args="-0.6 9 0 0 0 0 1 map goal5 100" />
<node pkg="tf" type="static_transform_publisher" name="goal_6_pub" args="-0.52 0.53 0 0 0 0 1 map goal6 100" />
<node pkg="tf" type="static_transform_publisher" name="goal_7_pub" args="-18.73 9.6 0 0 0 0 1 map goal7 100" />
```

Figura 12: Goal 5, Goal 6 and Goal 7

By implementing listeners also for these nodes and selecting the second choice after the first, as shown in figure 10, it is possible to obtain a complete map of the environment:

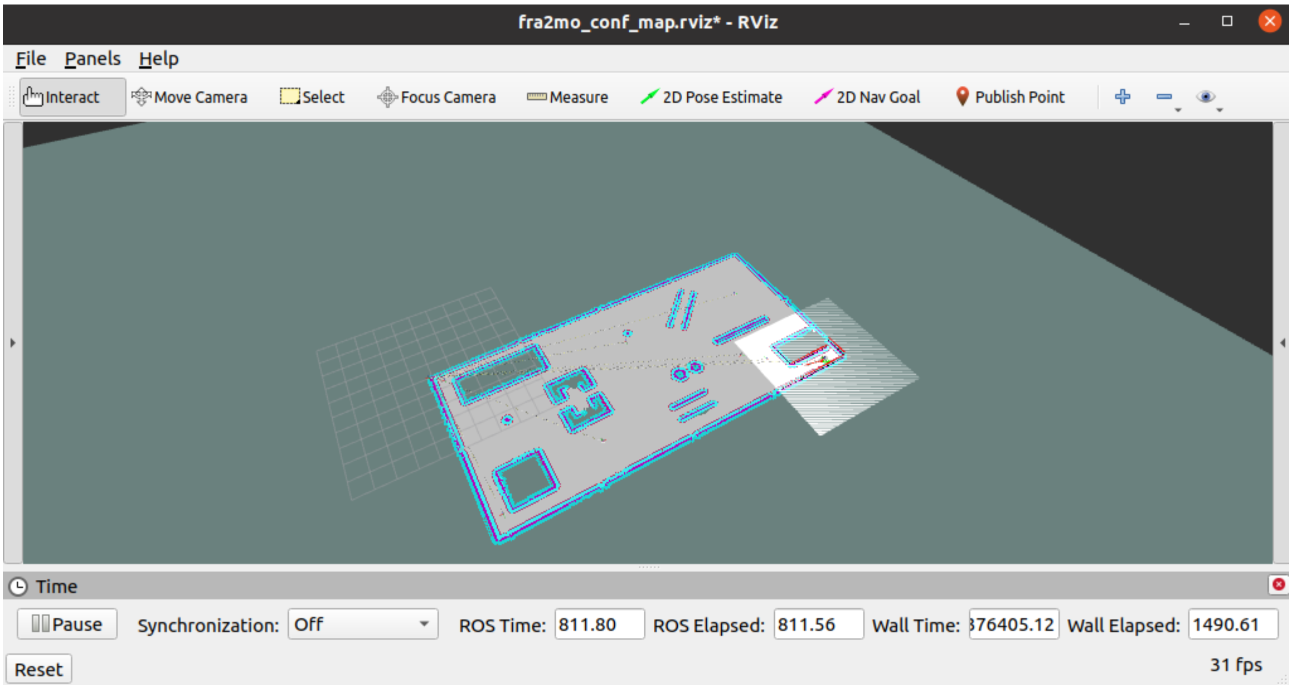

Figura 13: complete map on Rviz

3.b) Change the parameters of the planner and move_base (try at least 4 different configurations) and comment on the results you get in terms of robot trajectories. The parameters that need to be changed are:

- In file teb_locl_planner_params.yaml: tune parameters related to the section about trajectory, robot, and obstacles.

```yaml
# Trajectory

teb_autosize: True
dt_ref: 0.5          #Desired temporal resolution of the trajectory
dt_hysteresis: 0.2  #Hysteresis for automatic resizing depending on the current temporal resolution, usually approx.
global_plan_overwrite_orientation: True #Overwrite orientation of local subgoals provided by the global planner
max_global_plan_lookahead_dist: 8.0      #1)5.0    2)5.0   3)3.0   4)8.0   5)8.0  #Specify the maximum length (cumul
feasibility_check_no_poses: 5

publish_feedback: true

# Robot

max_vel_x: 0.75            #1)0.6    2)0.6   3)0.2   4)0.9   5)0.75
max_vel_x_backwards: 0.3
max_vel_theta: 0.6        #1)0.6    2)0.6   3)0.8   4)0.3   5)0.6
acc_lim_x: 0.7            #1)0.6    2)0.6   3)0.3   4)0.8   5)0.7
acc_lim_theta: 0.6        #1)0.6    2)0.6   3)0.8   4)0.3   5)0.6
min_turning_radius: 0.0
footprint_model:
 type: "polygon"


 vertices: [[0.1, -0.1], #1)[0.4, -0.4]    2)[0.1, -0.1]     3)[0.1, -0.1]     4)[0.1, -0.1]     5)[0.1, -0.1]
           [-0.1, -0.1], #1)[-0.4, -0.4]   2)[-0.1, -0.1]    3)[-0.1, -0.1]    4)[-0.1, -0.1]    5)[-0.1, -0.1]
           [-0.1, 0.1],  #1)[-0.4, 0.4]    2)[-0.1, 0.1]     3)[-0.1, 0.1]     4)[-0.1, 0.1]     5)[-0.1, 0.1]
           [0.1, 0.1]]   #1)[0.4, 0.4]     2)[0.1, 0.1]      3)[0.1, 0.1]      4)[0.1, 0.1]      5)[0.1, 0.1]

# GoalTolerance

xy_goal_tolerance: 0.1  #1)0.15   2)0.15    3)0.18    4)0.18    4)0.1
yaw_goal_tolerance: 0.1 #1)0.15   2)0.15    3)0.18    4)0.18    4)0.1
free_goal_vel: False
```

Figura 14: teb_local_planner_params.yaml

- In file local_costmap_params.yaml and global_costmap_params.yaml: change dimensions' values and update costmaps' frequency.

```
# example2 ---------------------------------------------------------------
# local_costmap:
#   global_frame: map
#   robot_base_frame: base_footprint
#   update_frequency: 15.0
#   publish_frequency: 30.0
#   static_map: false
#   rolling_window: true
#   width: 15.0
#   height: 15.0
#   resolution: 0.05

# example3 ---------------------------------------------------------------
# local_costmap:
#   global_frame: map
#   robot_base_frame: base_footprint
#   update_frequency: 5.0
#   publish_frequency: 10.0
#   static_map: false
#   rolling_window: true
#   width: 15.0
#   height: 15.0
#   resolution: 0.03
```

Figura 15: local_costmap_params.yaml

```
#example2 ------------------------------------------------------------
# global_costmap:
#   global_frame: map
#   robot_base_frame: base_footprint
#   update_frequency: 7.0
#   publish_frequency: 3.5
#   #always_send_full_costmap: false #default is false
#   rolling_window: false
#   resolution: 0.05
#   width: 15
#   height: 15
#   origin_x: -15
#   origin_y: -15

#example3 ------------------------------------------------------------
# global_costmap:
#   global_frame: map
#   robot_base_frame: base_footprint
#   update_frequency: 5.0
#   publish_frequency: 2.0
#   #always_send_full_costmap: false #default is false
#   rolling_window: false
#   resolution: 0.05
#   width: 15
#   height: 15
#   origin_x: -15
#   origin_y: -15
```

Figura 16: global_costmap_params.yaml

• **In file costmap_common_params.yaml: tune parameters related to the obstacle and raytrace ranges and footprint coherently as done in planner parameters.**

```
#example1 --------------------------------------------------
# publish_voxel_map: false
# transform_tolerance: 0.5
# meter_scoring: true
# obstacle_range: 5.0 # maximum range sensor reading that will result in an obstacle being put into the costmap
# raytrace_range: 6.0 # range to which we will raytrace freespace given a sensor reading
# footprint: [[0.4, -0.4],
#             [-0.4, -0.4],
#             [-0.4, 0.4],
#             [0.4, 0.4]]

#example2  --------------------------------------------------
# publish_voxel_map: false
# transform_tolerance: 0.7
# meter_scoring: true
# obstacle_range: 5.0 # maximum range sensor reading that will result in an obstacle being put into the costmap
# raytrace_range: 6.0 # range to which we will raytrace freespace given a sensor reading
# footprint: [[0.1, -0.1],
#             [-0.1, -0.1],
#             [-0.1, 0.1],
#             [0.1, 0.1]]
```

Figura 17: costmap_common_params.yaml

It is possibile to see that:

• The trajectory planning changes by modifing footprint and vertices.

• By modifing the **max_vel_x** and **max_acc_x** changes the linear velocity but usually it also has more difficulties doing curves.

• By modifing the **max_vel_theta** and **max_acc_theta** changes the angular velocity. If are both too high the robot has difficulties doing a linear movements.

• By changing **goal_tolerance** the robot gets to a more or less precise pose.

• For higher values of the **min_obstacle_dist** the robot doesn't go through tight spaces.

• For higher values of **max_global_plan_lookahead_dist** the trajectories are planned considering also the more distant obstacles .

4. **Vision-based navigation of the mobile platform**

4.a) Run ArUco ROS node using the robot camera: bring up the camera model and uncomment it in that fra2mo.xacro file of the mobile robot description rl_fra2mo_description. Remember to install the camera description pkg: sudo apt-get install ros-<DISTRO>-realsense2-description

In the file **d435_gazebo_macro** the lines of code about the D435 camera have been uncommented.

```
<launch>
    <!-- 4.b) file changed in order to let robot to look for the Aruco Marker -->
    <arg name="markerId"        default="115"/>
    <arg name="markerSize"      default="0.1"/>    <!-- in m -->
    <arg name="camera"          default="depth_camera/depth_camera"/>
    <arg name="marker_frame"    default="aruco_marker_frame"/>
    <arg name="ref_frame"       default="map"/>
    <arg name="corner_refinement" default="LINES" />

    <node pkg="aruco_ros" type="single" name="aruco_single">
        <remap from="/camera_info" to="/$(arg camera)/camera_info" />
        <remap from="/image" to="/$(arg camera)/image_raw" />
        <param name="image_is_rectified" value="True"/>
        <param name="marker_size"        value="$(arg markerSize)"/>
        <param name="marker_id"          value="$(arg markerId)"/>
        <param name="reference_frame"    value="$(arg ref_frame)"/>   <!-- frame in which the marker pose will be refered -->
        <param name="camera_frame"       value="camera_depth_optical_frame"/>
        <param name="marker_frame"       value="$(arg marker_frame)" />
        <param name="corner_refinement"  value="$(arg corner_refinement)" />
    </node>

</launch>
```

Figura 18: usb_cam_aruco.launch

1)As shown in the figure above, the **markerId** was changed to **115**.

2)The argument **camera** was changed in **depth_camera/depth_camera**.

3)The **ref_frame** was changed in **map**, to express the pose of the marker in the map frame.

4)The **camera_frame** was changed in **camera_depth_optical_frame**.

5)The **usb_cam_aruco.launch** file was included in the

fra2mo_nav_bringup.launch

4.b) **Implement a 2D navigation task following this logic**

- **Send the robot in the proximity of obstacle 9.**
- **Make the robot look for the ArUco marker. Once detected, retrieve its pose with respect to the map frame.**
- **Set the following pose (relative to the ArUco marker pose) as next goal for the robot**

  x = xm + 1, y = ym

  where xm, ym are the marker coordinates.

- A new goal (**goal8**) has been set at position x = -16 and y = 8 near the obstacle 9. Selecting the third choice of the **send_goal** function the robot will reach position 8 and then it will move to the position relative to the aruco marker.

- A new subscriber to the topic /**aruco_single**/**pose** was defined in the file **tf_nav** and the function **aruco_call** was called to retrive pose of the aruco marker with respect to map frame.

```cpp
        else if (cmd ==3){

        //4.b)
        TF_NAV::Goal(_goal3_pos,_goal3_or,3);
        TF_NAV::Goal(_goal8_pos,_goal8_or,8);
        std::cout<<"\n new goal with x=x_m +1 , y=ym "<<std::endl;

         move_base_msgs::MoveBaseGoal goal;

        MoveBaseClient ac("move_base", true);
        while(!ac.waitForServer(ros::Duration(5.0))){
            ROS_INFO("Waiting for the move_base action server to come up");
        }

        goal.target_pose.header.frame_id = "map";
        goal.target_pose.header.stamp = ros::Time::now();

        goal.target_pose.pose.position.x = aruco_pose[0]+1;
        goal.target_pose.pose.position.y = aruco_pose[1];
        goal.target_pose.pose.position.z = _goal8_pos[2];

        goal.target_pose.pose.orientation.w = _goal8_or[0];
        goal.target_pose.pose.orientation.x = _goal8_or[1];
        goal.target_pose.pose.orientation.y = _goal8_or[2];
        goal.target_pose.pose.orientation.z = _goal8_or[3];

        ROS_INFO("Sending the new Goal!");
        ac.sendGoal(goal);
        ac.waitForResult();

        if(ac.getState() == actionlib::SimpleClientGoalState::SUCCEEDED) {
        ROS_INFO("The mobile robot has reached the new goal!");
```

Figura 19: send_goal function

```
//4.b)////
std::vector<double> aruco_pose(7,0.0);

void aruco_call(const geometry_msgs::PoseStamped & msg)
{
    aruco_pose.clear();
    aruco_pose.push_back(msg.pose.position.x);
    aruco_pose.push_back(msg.pose.position.y);
    aruco_pose.push_back(msg.pose.position.z);
    aruco_pose.push_back(msg.pose.orientation.x);
    aruco_pose.push_back(msg.pose.orientation.y);
    aruco_pose.push_back(msg.pose.orientation.z);
    aruco_pose.push_back(msg.pose.orientation.w);

}
```

Figura 20: Aruco Callback

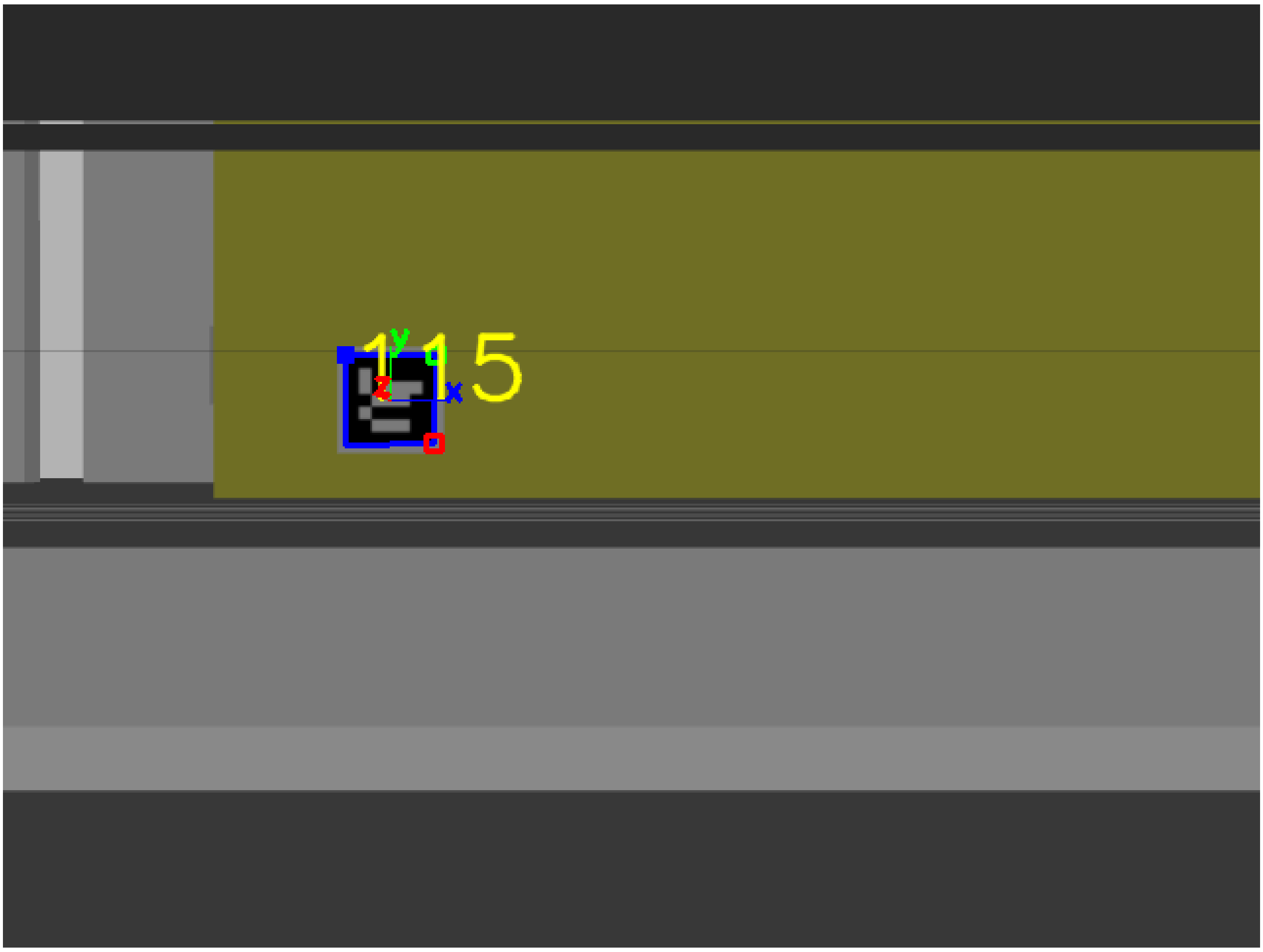

Figura 21: Aruco marker detected by mobile robot in the position of goal 8.

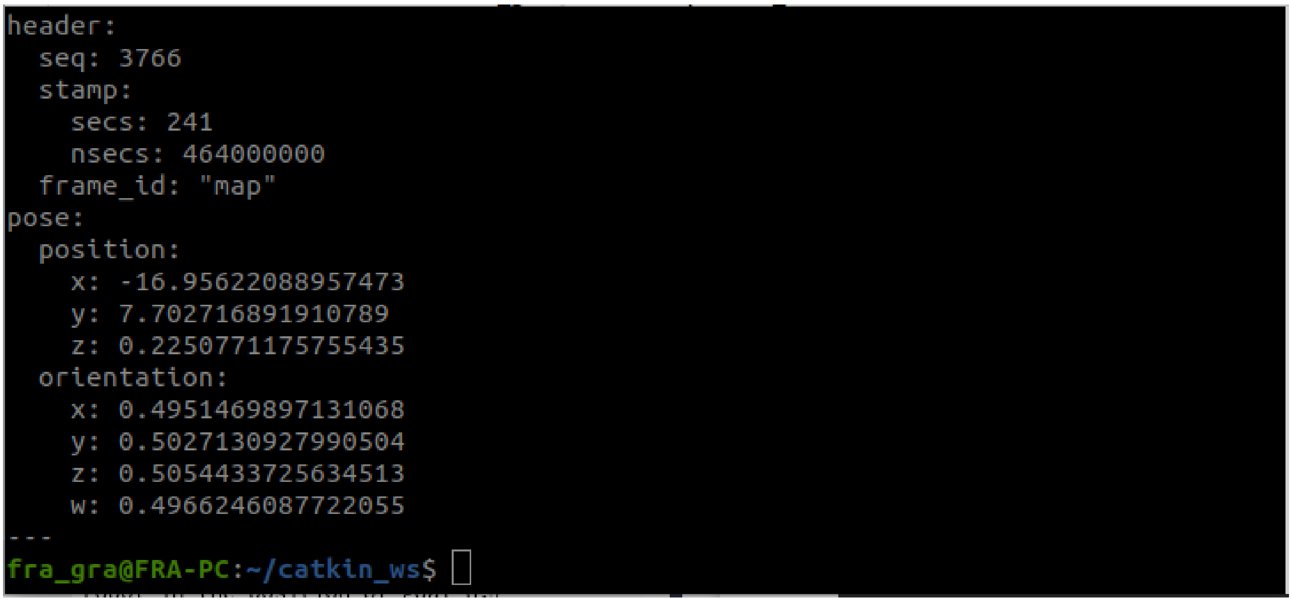

Figura 22: Aruco marker pose with respect to the map frame. (Obtain with rostopic echo -c /aruco_single/pose)

Figura 23: Final pose of the robot with respect to map frame

4.c) Publish the ArUco pose as TF.

- A new subscriber to the topic **/aruco_single/pose** was defined in the file **tf_nav** and the function **poseCallback** was called , defining a static object

  tf::TransformBroadcaster to send the Aruco transform.

```cpp
//4.c)//// Publish Aruco Pose as TF
void poseCallback(const geometry_msgs::PoseStamped & msg)
{
    static tf::TransformBroadcaster br;
    tf::Transform transform;
    transform.setOrigin( tf::Vector3(msg.pose.position.x, msg.pose.position.y, msg.pose.position.z));
    tf::Quaternion q;
    q.setX(msg.pose.orientation.x);
    q.setY(msg.pose.orientation.y);
    q.setZ(msg.pose.orientation.z);
    q.setW(msg.pose.orientation.w);
    transform.setRotation(q);
    br.sendTransform(tf::StampedTransform(transform, ros::Time::now(), "map", "aruco_frame"));

    }
```

Figura 24: poseCallback function

```
At time 913.536
- Translation: [-16.844, 7.707, 0.225]
- Rotation: in Quaternion [0.497, 0.503, 0.504, 0.496]
            in RPY (radian) [1.571, -0.002, 1.584]
            in RPY (degree) [89.986, -0.120, 90.778]
```

Figura 25: Aruco pose as TF obtained with **rosrun tf tf_echo /map /aruco_frame**