Robotics Lab: Homework 4

Control a mobile robot to follow a trajectory

Pietro Falco

P38000208

Control a mobile robot to follow a trajectory

The goal of this homework is to implement an autonomous navigation software framework to control a mobile robot. The **rl_fra2mo_description** and **fra2mo_2dnav** packages must be used as a starting point for the simulation. The student is requested to address the following points and provide a detailed report of the methods employed. In addition, a personal GitHub repo with all the developed code must be shared with the instructor. The report is due in one week from the homework release.

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

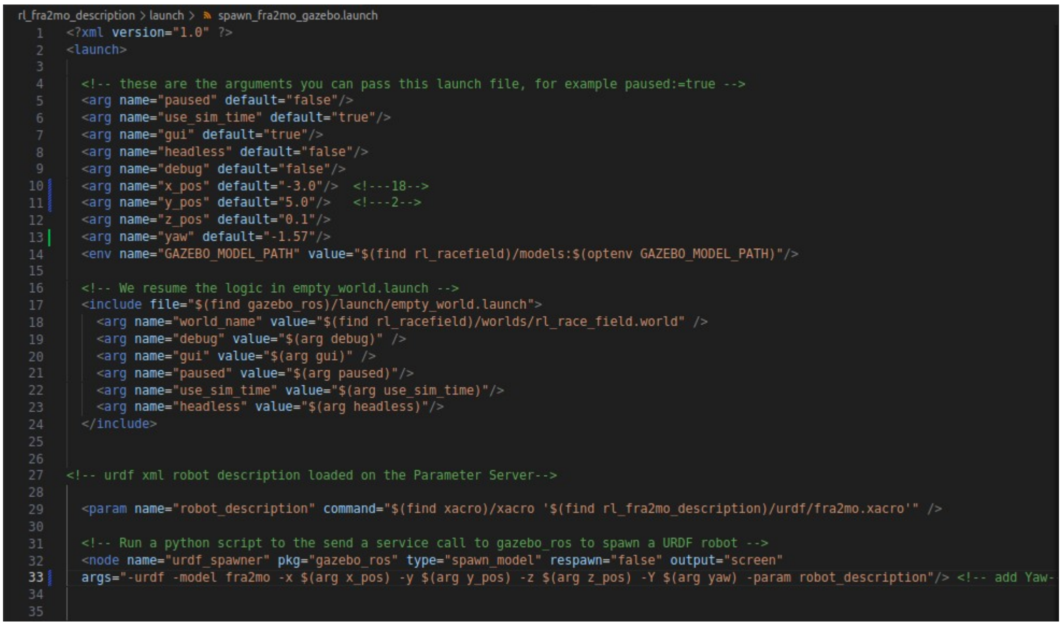
$x = -3$ $y = 5$ $yaw = -90$ deg

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

To set the spawn pose we modified the **spawn_fra2mo_gazebo** file. In that file we changed the x_pos, y_pos values and we also added a new parameter, **Y**, for changing the spawn orientation.

Then we passed also the new value adding **– Y $(arg_yaw)**

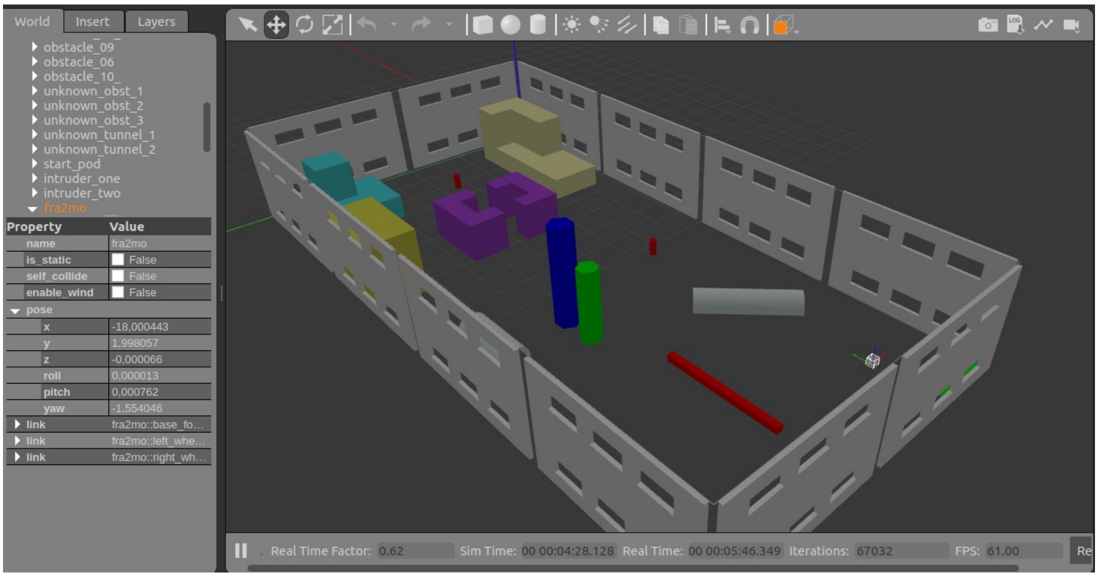
You can see the modified script in the figure below:



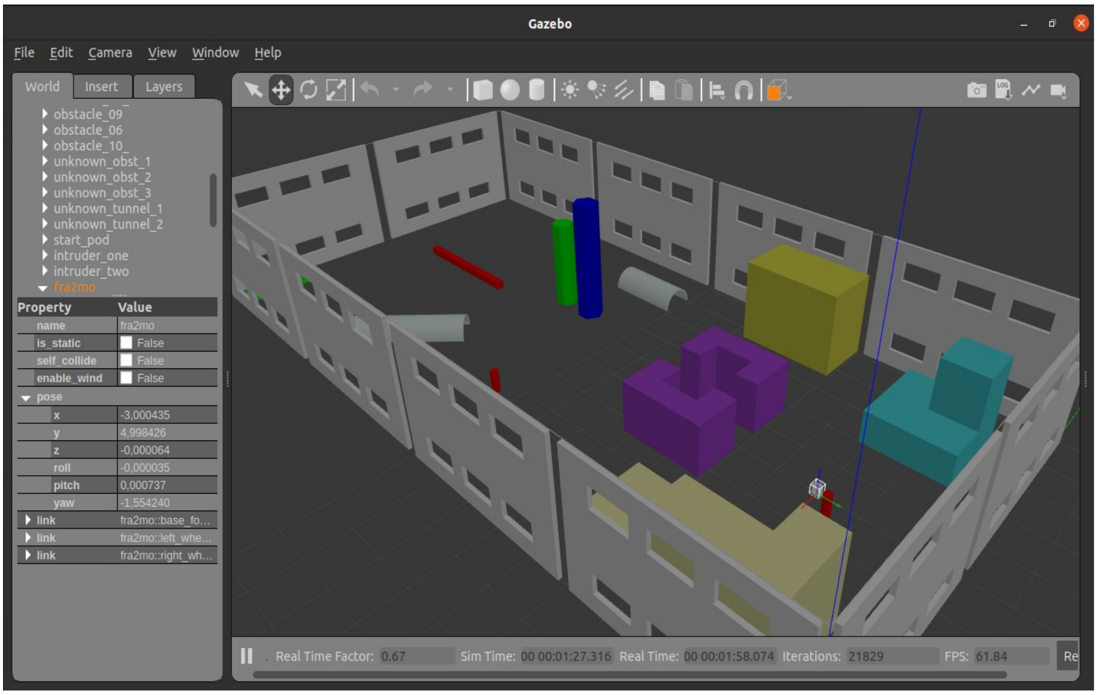
```
rl_fra2mo_description > launch > spawn_fra2mo_gazebo.launch
<?xml version="1.0" ?>
<launch>

  <!-- these are the arguments you can pass this launch file, for example paused:=true -->
  <arg name="paused" default="false"/>
  <arg name="use_sim_time" default="true"/>
  <arg name="gui" default="true"/>
  <arg name="headless" default="false"/>
  <arg name="debug" default="false"/>
  <arg name="x_pos" default="-3.0"/>  <!---18-->
  <arg name="y_pos" default="5.0"/>   <!---2-->
  <arg name="z_pos" default="0.1"/>
  <arg name="yaw" 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 -->
  <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) -param robot_description"/> <!-- add Yaw-
```

Then to launch the environment we launched the **spawn_fra2mo_gazebo.launch** file.

This is the start environment with the mobile robot in the old initial position.



Here we show the new environment with the mobile robot in the new position:



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

We moved the **obstacle_9** in the required position changing the values related to its pose modifying the **rl_race_field.world** file.

rl_racefield > worlds > rl_race_field.world

```
<!-- Static Obstacles-->
<include>
  <name>obstacle_03</name>
  <pose> -7.198 6.684 0.1 0 0 -1.57 </pose>
  <uri>model://obstacle_03</uri>
</include>

<include>
  <name>obstacle_03_second</name>
  <pose> -5 3.58 0.1 0 0 1.57 </pose>
  <uri>model://obstacle_03</uri>
</include>

<include>
  <name>obstacle_09</name>
  <pose> -17 9 0.1 0 0 3.14159</pose> <!--pose> -5 10 0.1 0 0 3.14159</pose-->
  <uri>model://obstacle_09</uri>
</include>

<include>
  <name>obstacle_06</name>
  <pose> -3.153 6.881 0.1 0 0 0</pose>
  <uri>model://obstacle_06</uri>
</include>

<include>
  <name>obstacle_10 </name>
  <pose> -3.30 1.60 0.10 0 0 3.14159</pose>
  <uri>model://obstacle_10 </uri>
</include>
```

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.***

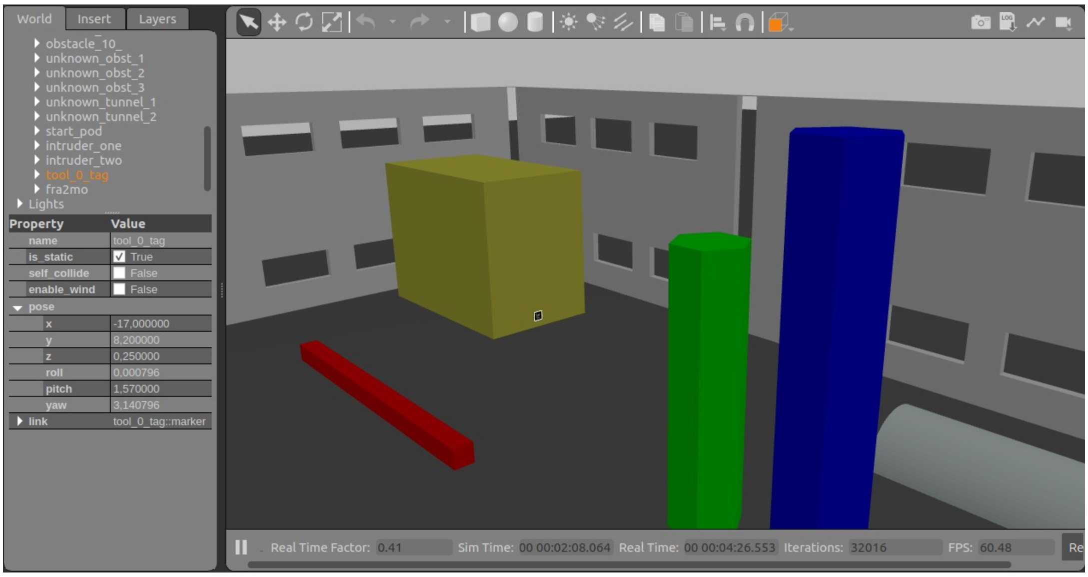
First of all we generated a new Aruco representing the number 115 with the online Aruco generator.
We put it inside a new folder for Aruco textures, named **marker_new**. Then we copied the Aruco texture, naming it **ar_tag_115**, inside the right folder.
In the scripts folder we also modified the **marker_new.material** like in the figure above:

```
material marker_new
{
    technique
    {
        pass
        {
            texture_unit
            {
                texture ar_tag_115.png
            }
        }
    }
}
```

In order to spawn the marker on the **obstacal_9** we changed its position in the **rl_race_field.world** file like shown in the figure above:

```
<!-- AR markers -->
<include>
  <name>tool_0_tag</name>
  <uri>model://marker_new</uri>
  <pose>-17 8.2 0.25 0 1.57 3.14</pose>
</include>
```

How it appears:



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.

We added a publisher for each goal in the **spawn_fra2mo_gazebo.launch**.

Those are 4 **static_transform_publisher** nodes from the tf package to that publish the rigid transformation from a starting coordinate system **map** to a target coordinate system (**goal1, goal2, goal3, goal4**).

These transformations are combined representations of translations and rotations. The rotation is expressed in Quaternion.

```
<!--Static tf publisher for 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 0.25881904510252074 0.9659258262890683 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 0.6087614 0.7933533 map goal4 100"/>
```

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

In order to do that we implemented four **tf::StampedTransform** in the **tf_nav.cpp**. One for each goal.

The function **waitForTransform()** waits for a transform to arrive and the **lookupTransform()** get the transform between the two frames passing it to the transform variable.

```
tf::TransformListener listener;
tf::StampedTransform transform1, transform2, transform3, transform4;
```

```cpp
while ( ros::ok() )
{
    try
    {
        listener.waitForTransform("map", "goal1", ros::Time(0), ros::Duration(10.0));
        listener.lookupTransform("map", "goal1", ros::Time(0), transform1);

        listener.waitForTransform("map", "goal2", ros::Time(0), ros::Duration(10.0));
        listener.lookupTransform("map", "goal2", ros::Time(0), transform2);

        listener.waitForTransform("map", "goal3", ros::Time(0), ros::Duration(10.0));
        listener.lookupTransform("map", "goal3", ros::Time(0), transform3);

        listener.waitForTransform("map", "goal4", ros::Time(0), ros::Duration(10.0));
        listener.lookupTransform("map", "goal4", ros::Time(0), transform4);
```

Then we assign the listened values to the goal variables that we have defined:

```cpp
TF_NAV::TF_NAV()
{
    _position_pub = _nh.advertise<geometry_msgs::PoseStamped>( "/fra2mo/pose", 1 );
    _cur_pos << 0.0, 0.0, 0.0;
    _cur_or << 0.0, 0.0, 0.0, 1.0;
    _home_pos << -18.0, 2.0, 0.0;

    //Point 2
    _goal1_pos << 0.0, 0.0, 0.0;
    _goal1_or << 0.0, 0.0, 0.0, 1.0;
    _goal2_pos << 0.0, 0.0, 0.0;
    _goal2_or << 0.0, 0.0, 0.0, 1.0;
    _goal3_pos << 0.0, 0.0, 0.0;
    _goal3_or << 0.0, 0.0, 0.0, 1.0;
    _goal4_pos << 0.0, 0.0, 0.0;
    _goal4_or << 0.0, 0.0, 0.0, 1.0;
```

```cpp
    _goal1_pos << transform1.getOrigin().x(), transform1.getOrigin().y(), transform1.getOrigin().z();
    _goal1_or << transform1.getRotation().w(),  transform1.getRotation().x(), transform1.getRotation().y(), transform1.getRotation().z();

    _goal2_pos << transform2.getOrigin().x(), transform2.getOrigin().y(), transform2.getOrigin().z();
    _goal2_or << transform2.getRotation().w(),  transform2.getRotation().x(), transform2.getRotation().y(), transform2.getRotation().z();

    _goal3_pos << transform3.getOrigin().x(), transform3.getOrigin().y(), transform3.getOrigin().z();
    _goal3_or << transform3.getRotation().w(),  transform3.getRotation().x(), transform3.getRotation().y(), transform3.getRotation().z();

    _goal4_pos << transform4.getOrigin().x(), transform4.getOrigin().y(), transform4.getOrigin().z();
    _goal4_or << transform4.getRotation().w(),  transform4.getRotation().x(), transform4.getRotation().y(), transform4.getRotation().z();
```

This is the print of the all the pose of the target goals. We did it using the command **rostopic echo /tf**

```
transforms:
  -
    header:
      seq: 0
      stamp:
        secs: 69
        nsecs: 696000000
      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
```

```
transforms:
  -
    header:
      seq: 0
      stamp:
        secs: 69
        nsecs: 696000000
      frame_id: "map"
    child_frame_id: "goal2"
    transform:
      translation:
        x: -15.0
        y: 7.0
        z: 0.0
      rotation:
        x: 0.0
        y: 0.0
        z: 0.2588190451025207
        w: 0.9659258262890683
```

```
transforms:
  -
    header:
      seq: 0
      stamp:
        secs: 69
        nsecs: 696000000
      frame_id: "map"
    child_frame_id: "goal3"
    transform:
      translation:
        x: -6.0
        y: 8.0
        z: 0.0
      rotation:
        x: 0.0
        y: 0.0
        z: 1.0
        w: 0.0
```

```
transforms:
  -
    header:
      seq: 0
      stamp:
        secs: 69
        nsecs: 676000000
      frame_id: "map"
    child_frame_id: "goal4"
    transform:
      translation:
        x: -17.5
        y: 3.0
        z: 0.0
      rotation:
        x: 0.0
        y: 0.0
        z: 0.608761430209528
        w: 0.7933533393698233
```

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.***

In order to do that we sent the goals by using the **send_goal()** function implemented in the **MoveBaseClient**.

After sending the goal, it waits for the robot to get there thanks to the **waitForResult()** function.

```
if (cmd == 1)
{
    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 3
    goal3.target_pose.header.stamp = ros::Time::now();

    goal3.target_pose.pose.position.x = _goal3_pos[0];
    goal3.target_pose.pose.position.y = _goal3_pos[1];
    goal3.target_pose.pose.position.z = _goal3_pos[2];

    goal3.target_pose.pose.orientation.w = _goal3_or[0];
    goal3.target_pose.pose.orientation.x = _goal3_or[1];
    goal3.target_pose.pose.orientation.y = _goal3_or[2];
    goal3.target_pose.pose.orientation.z = _goal3_or[3];

    ROS_INFO("Sending goal3");
    std::cout<<"Goal 3 pose:  x: "<<_goal3_pos[0]<<", y: "<<_goal3_pos[1]<<", z: "<<_goal3_pos[2]<<std::endl;
    ac.sendGoal(goal3);

    ac.waitForResult();

    if(ac.getState() == actionlib::SimpleClientGoalState::SUCCEEDED)
    ROS_INFO("The mobile robot arrived in the TF goal");
    else
        ROS_INFO("The base failed to move for some reason");
```

In order to plot the trajectory, we saved the pose of the robot in a bag file using this command:

```
davide@davide:~/catkin_ws$ rosbag record -O /home/davide/Documents/BAG/totTraj.bag /fra2mo/pose
```

Then we had to plot it doing the **rostopic echo** of the **/fra2mo/pose**.

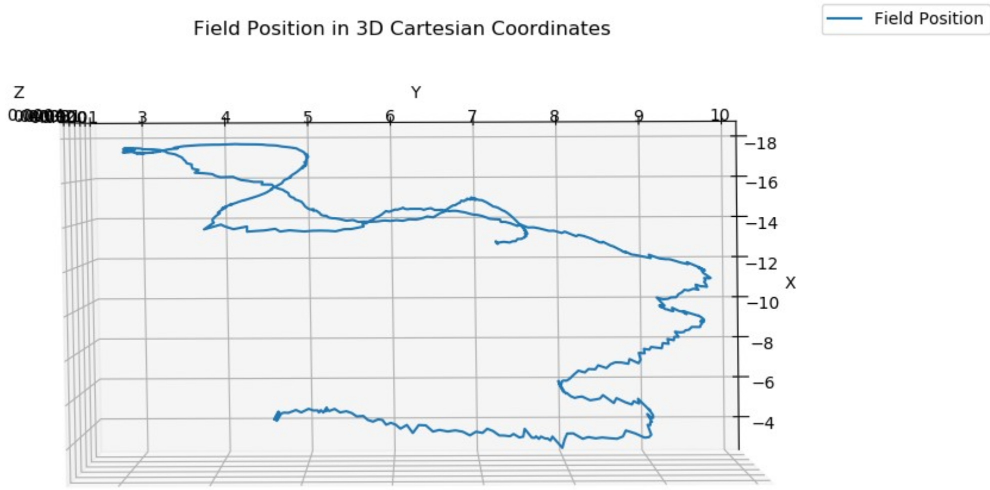
We just need the robot position to plot the trajectory, so we made a python script to take only the position values and then plot them in a 3D view.

readBag.py totTraj.bag

The python script:

```python
import rosbag
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

bag_file_path = '/home/davide/Documents/BAG/totTraj.bag'

with rosbag.Bag(bag_file_path, 'r') as bag:
    positions_x = []
    positions_y = []
    positions_z = []

    for topic, msg, t in bag.read_messages(topics=['/fra2mo/pose']):
        x = msg.pose.position.x
        y = msg.pose.position.y
        z = msg.pose.position.z

        positions_x.append(x)
        positions_y.append(y)
        positions_z.append(z)

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.plot(positions_x, positions_y, positions_z, label='Field Position')
ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_zlabel('Z')
ax.set_title('Field Position in 3D Cartesian Coordinates')
plt.legend()
plt.show()
```

The represented trajectory:



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.

In order to do that we defined an entire new set of goals as shown:

- In **spawn_fra2mo_gazebo.launch**:

```xml
<node pkg="tf" type="static_transform_publisher" name="goal_3_1_pub" args="-0.5 9.5 0 0 0 1 0 map goal3_1 100"/>
<node pkg="tf" type="static_transform_publisher" name="goal_3_2_pub" args="-4.2 8.3 0 0 0 0 1 map goal3_2 100"/>
<node pkg="tf" type="static_transform_publisher" name="goal_3_3_pub" args="-6.3 5.14 0 0 0 -0.7071067812 0.7071067812 map goal3_3 100"/>
<node pkg="tf" type="static_transform_publisher" name="goal_3_4_pub" args="-9 7.3 0 0 0.7071067812 0.7071067812 map goal3_4 100"/>
<node pkg="tf" type="static_transform_publisher" name="goal_3_5_pub" args="-18 9.6 0 0 0 1 0 map goal3_5 100"/>
<node pkg="tf" type="static_transform_publisher" name="goal_3_6_pub" args="-16 1.2 0 0 0 0 1 map goal3_6 100"/>
<node pkg="tf" type="static_transform_publisher" name="goal_3_7_pub" args="-6 8 0 0 0 1 0 map goal3_7 100"/>
<node pkg="tf" type="static_transform_publisher" name="goal_3_8_pub" args="-2.1 0.41 0 0 0 0.7071067812 0.7071067812 map goal3_8 100"/>
```

- In **tf_nav**: first of all we defined a transform for each goal and we passed to them the values listened.

```cpp
listener.waitForTransform("map", "goal3_1", ros::Time(0), ros::Duration(10.0));
listener.lookupTransform("map", "goal3_1", ros::Time(0), transform3_1);

listener.waitForTransform("map", "goal3_2", ros::Time(0), ros::Duration(10.0));
listener.lookupTransform("map", "goal3_2", ros::Time(0), transform3_2);

listener.waitForTransform("map", "goal3_3", ros::Time(0), ros::Duration(10.0));
listener.lookupTransform("map", "goal3_3", ros::Time(0), transform3_3);

listener.waitForTransform("map", "goal3_4", ros::Time(0), ros::Duration(10.0));
listener.lookupTransform("map", "goal3_4", ros::Time(0), transform3_4);

listener.waitForTransform("map", "goal3_5", ros::Time(0), ros::Duration(10.0));
listener.lookupTransform("map", "goal3_5", ros::Time(0), transform3_5);

listener.waitForTransform("map", "goal3_6", ros::Time(0), ros::Duration(10.0));
listener.lookupTransform("map", "goal3_6", ros::Time(0), transform3_6);

listener.waitForTransform("map", "goal3_7", ros::Time(0), ros::Duration(10.0));
listener.lookupTransform("map", "goal3_7", ros::Time(0), transform3_7);

listener.waitForTransform("map", "goal3_8", ros::Time(0), ros::Duration(10.0));
listener.lookupTransform("map", "goal3_8", ros::Time(0), transform3_8);
```

- Then we assigned the goal values to the variables that we defined for each goal:

```cpp
_goal3_1_pos << transform3_1.getOrigin().x(), transform3_1.getOrigin().y(), transform3_1.getOrigin().z();
_goal3_1_or << transform3_1.getRotation().w(),  transform3_1.getRotation().x(), transform3_1.getRotation().y(), transform3.getRotation().z();

_goal3_2_pos << transform3_2.getOrigin().x(), transform3_2.getOrigin().y(), transform3_2.getOrigin().z();
_goal3_2_or << transform3_2.getRotation().w(),  transform3_2.getRotation().x(), transform3_2.getRotation().y(), transform3.getRotation().z();

_goal3_3_pos << transform3_3.getOrigin().x(), transform3_3.getOrigin().y(), transform3_3.getOrigin().z();
_goal3_3_or << transform3_3.getRotation().w(),  transform3_3.getRotation().x(), transform3_3.getRotation().y(), transform3.getRotation().z();

_goal3_4_pos << transform3_4.getOrigin().x(), transform3_4.getOrigin().y(), transform3_4.getOrigin().z();
_goal3_4_or << transform3_4.getRotation().w(),  transform3_4.getRotation().x(), transform3_4.getRotation().y(), transform3.getRotation().z();

_goal3_5_pos << transform3_5.getOrigin().x(), transform3_5.getOrigin().y(), transform3_5.getOrigin().z();
_goal3_5_or << transform3_5.getRotation().w(),  transform3_5.getRotation().x(), transform3_5.getRotation().y(), transform3.getRotation().z();

_goal3_6_pos << transform3_6.getOrigin().x(), transform3_6.getOrigin().y(), transform3_6.getOrigin().z();
_goal3_6_or << transform3_6.getRotation().w(),  transform3_6.getRotation().x(), transform3_6.getRotation().y(), transform3.getRotation().z();

_goal3_7_pos << transform3_7.getOrigin().x(), transform3_7.getOrigin().y(), transform3_7.getOrigin().z();
_goal3_7_or << transform3_7.getRotation().w(),  transform3_7.getRotation().x(), transform3_7.getRotation().y(), transform3.getRotation().z();

_goal3_8_pos << transform3_8.getOrigin().x(), transform3_8.getOrigin().y(), transform3_8.getOrigin().z();
_goal3_8_or << transform3_8.getRotation().w(),  transform3_8.getRotation().x(), transform3_8.getRotation().y(), transform3.getRotation().z();
```

- Then in the **send_goal()** function we did this for each goal:

```
else if (cmd == 3)
{
    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 3_1
    goal3_1.target_pose.header.stamp = ros::Time::now();

    goal3_1.target_pose.pose.position.x = _goal3_1_pos[0];
    goal3_1.target_pose.pose.position.y = _goal3_1_pos[1];
    goal3_1.target_pose.pose.position.z = _goal3_1_pos[2];

    goal3_1.target_pose.pose.orientation.w = _goal3_1_or[0];
    goal3_1.target_pose.pose.orientation.x = _goal3_1_or[1];
    goal3_1.target_pose.pose.orientation.y = _goal3_1_or[2];
    goal3_1.target_pose.pose.orientation.z = _goal3_1_or[3];

    ROS_INFO("Sending goal3_1");
    std::cout<<"Goal 3_1 pose:  x: "<<_goal3_1_pos[0]<<", y: "<<_goal3_1_pos[1]<<", z: "<<_goal3_1_pos[2]<<std::endl;
    ac.sendGoal(goal3_1);

    ac.waitForResult();

    if(ac.getState() == actionlib::SimpleClientGoalState::SUCCEEDED)
    ROS_INFO("The mobile robot arrived in the TF goal");
    else
        ROS_INFO("The base failed to move for some reason");
```

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.

```
# Trajectory

teb_autosize: True
dt_ref: 0.5
dt_hysteresis: 0.2
global_plan_overwrite_orientation: True
max_global_plan_lookahead_dist: 8.0      #1)5.0    2)5.0   3)3.0   4)8.0   5)8.0
feasibility_check_no_poses: 5

publish_feedback: true

# Robot

max_vel_x: 0.8            #1)0.6    2)0.6   3)0.2   4)0.9   5)0.8
max_vel_x_backwards: 0.3
max_vel_theta: 0.4        #1)0.6    2)0.6   3)0.8   4)0.3   5)0.4
acc_lim_x: 0.7            #1)0.6    2)0.6   3)0.3   4)0.8   5)0.7
acc_lim_theta: 0.35        #1)0.6    2)0.6   3)0.8   4)0.3   5)0.35
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

# Obstacles

min_obstacle_dist: 0.20                      #1)0.10   2)0.10    3)0.15    4)0.35    5)0.20
include_costmap_obstacles: True
costmap_obstacles_behind_robot_dist: 1.0     #1)1.0    2)1.0     3)1.0     4)1.0     5)1.0
obstacle_poses_affected: 20                  #1)20     2)20      3)20      4)20      5)20
costmap_converter_plugin: ""
costmap_converter_spin_thread: True
costmap_converter_rate: 5                    #1)5.0    2)5       3)5       4)5       5)5
inflation_dist: 0.1                          #1)0.1    2)0.1     3)0.4     4)0.1     5)0.1
include_dynamic_obstacles: false
```

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

```
#config1 --------------------------------
# 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

# config2 -------------------------------
# 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
```

```
# config3 -------------------------------
# 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

# config4 -------------------------------
# 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: 10.0
#   height: 10.0
#   resolution: 0.03
```

```
# config5 ----------------------------------------------------
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: 10.0
  height: 10.0
  resolution: 0.03
```

```
#config1 --------------------------------------------------
# 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

#config2 --------------------------------------------------
# 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
```

```
#config3 --------------------------------------------------
# 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

#config4 --------------------------------------------------
# global_costmap:
#   global_frame: map
#   robot_base_frame: base_footprint
#   update_frequency: 5.0
#   publish_frequency: 10.0
#   #always_send_full_costmap: false #default is false
#   rolling_window: false
#   resolution: 0.05
#   width: 10
#   height: 10
#   origin_x: -10
#   origin_y: -10
```

```
#config5 --------------------------------------------------
global_costmap:
  global_frame: map
  robot_base_frame: base_footprint
  update_frequency: 5.0
  publish_frequency: 10.0
  #always_send_full_costmap: false #default is false
  rolling_window: false
  resolution: 0.05
  width: 10
  height: 10
  origin_x: -10
  origin_y: -10
```

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

```
#config1 -----------------------
# publish_voxel_map: false
# transform_tolerance: 0.5
# meter_scoring: true
# obstacle_range: 5.0 # maximum
# raytrace_range: 6.0 # range to
# footprint: [[0.4, -0.4],
#             [-0.4, -0.4],
#             [-0.4, 0.4],
#             [0.4, 0.4]]

#config2 -----------------------
# publish_voxel_map: false
# transform_tolerance: 0.7
# meter_scoring: true
# obstacle_range: 5.0 # maximum
# raytrace_range: 6.0 # range to
# footprint: [[0.1, -0.1],
#             [-0.1, -0.1],
#             [-0.1, 0.1],
#             [0.1, 0.1]]
```

```
#config3 -----------------------
# publish_voxel_map: false
# transform_tolerance: 0.5
# meter_scoring: true
# obstacle_range: 5.0 # maximum
# raytrace_range: 6.0 # range to
# footprint: [[0.1, -0.1],
#             [-0.1, -0.1],
#             [-0.1, 0.1],
#             [0.1, 0.1]]

#config4 -----------------------
# publish_voxel_map: false
# transform_tolerance: 0.5
# meter_scoring: true
# obstacle_range: 7.0 # maximum
# raytrace_range: 7.0 # range to
# footprint: [[0.1, -0.1],
#             [-0.1, -0.1],
#             [-0.1, 0.1],
#             [0.1, 0.1]]
```

```
#config5 -----------------------------------------------
publish_voxel_map: false
transform_tolerance: 0.5
meter_scoring: true
obstacle_range: 7.0 # maximum range sensor reading that will
raytrace_range: 7.0 # range to which we will raytrace freespa
footprint: [[0.1, -0.1],
            [-0.1, -0.1],
            [-0.1, 0.1],
            [0.1, 0.1]]
```

After all these configurations we noticed that:

- By modifing **footprint** and **vertices** changes the trjectory planning.
- 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.
- By changing the **frequencies** of **publish** and **update** we noticed that it happens that in the terminal appears a warning "Lost update frequency".
- 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** we uncommented the lines about the D435 camera.

```
<robot name="fra2mo" xmlns:xacro="http://ros.org/wiki/xacro">
  <xacro:include filename="$(find rl_fra2mo_description)/urdf/fra2mo_base_macro.xacro" />
  <xacro:include filename="$(find rl_fra2mo_description)/urdf/lidar_gazebo_macro.xacro" />
  <xacro:include filename="$(find rl_fra2mo_description)/urdf/d435_gazebo_macro.xacro" />


  <xacro:property name="LIDAR" value="True"/>
  <xacro:property name="DEPTH" value="True"/>

  <!-- Diff Drive Robot Base  -->
  <xacro:fra2mo_base />

  <!-- LIDAR Sensor -->
  <xacro:if value="${LIDAR}" >
    <xacro:lidar_gazebo_sensor parent="lidar_link" />
  </xacro:if>
  <!-- Uncomment if you want to add also a D435 camera -->
  <!-- RGBD Sensor -->
  <xacro:if value="${DEPTH}" >
    <xacro:d435_gazebo_sensor parent="d435_link" />
  </xacro:if>

</robot>
```

To see the camera view we launched the **usb_cam_aruco.launch** file.

We modified the .launch file **usb_cam_aruco** changing: **markerID**, **camera**, **ref_frame** (in order to obtain the aruco pose with respect to the map frame) and also modified the default value related to **camera_frame**.

```
<launch>

   <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>
```

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***

$$x = x_m + 1,\ y = y_m,$$

where x_m, y_m are the marker coordinates.

In order to achieve these set of points, we created a new goal named **goal_aruco** which bring the robot in the proximity of **obstacle_9**. We added a **static_tf publisher** as done previously with the following pose:

```
<node pkg="tf" type="static_transform_publisher" name="goal_aruco_pub" args="-15 8.5 0 0 0 1 0 map goal_aruco 100"/>
```

Moreover, in **tf_nav.cpp** file, we added the following lines:

```
//Point 4
_goal_aruco_pos << 0.0, 0.0, 0.0;
_goal_aruco_or << 0.0, 0.0, 0.0, 1.0;
```

```
tf::StampedTransform transform_aruco;
```

```
listener.waitForTransform("map", "goal_aruco", ros::Time(0), ros::Duration(10.0));
listener.lookupTransform("map", "goal_aruco", ros::Time(0), transform_aruco);
```

```
_goal_aruco_pos << transform_aruco.getOrigin().x(), transform_aruco.getOrigin().y(), transform_aruco.getOrigin().z();
_goal_aruco_or << transform_aruco.getRotation().w(),  transform_aruco.getRotation().x(), transform_aruco.getRotation().y(), transform_aruco.getRotation().z();
```

```
//Goal Aruco
goal_aruco.target_pose.header.stamp = ros::Time::now();

goal_aruco.target_pose.pose.position.x = _goal_aruco_pos[0];
goal_aruco.target_pose.pose.position.y = _goal_aruco_pos[1];
goal_aruco.target_pose.pose.position.z = _goal_aruco_pos[2];

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

//vedi se c'è aruco e prendine la posizione
ros::Subscriber aruco_pose_sub = n.subscribe("/aruco_single/pose", 1, arucoPoseCallback);

ROS_INFO("Sending goal_aruco");
std::cout<<"Goal aruco pose:  x: "<<_goal_aruco_pos[0]<<", y: "<<_goal_aruco_pos[1]<<", z: "<<_goal_aruco_pos[2]<<std::endl;
ac.sendGoal(goal_aruco);

ac.waitForResult();
bool arrived = false;
if(ac.getState() == actionlib::SimpleClientGoalState::SUCCEEDED){
    arrived = true;
    ROS_INFO("The mobile robot arrived in the TF goal");
}
else
    ROS_INFO("The base failed to move for some reason");
```

The bool variable **arrived** will be useful in the next step.
In the **tf_nav.cpp** file we added the **arucoPoseCallback()** function and the **aruco_pose_sub** subscriber, that we used in the last homework, in order to detect the aruco_marker and retrieve its pose with respect to the map frame:

```
void arucoPoseCallback(const geometry_msgs::PoseStamped & msg)
{
    aruco_pose_available = true;
    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);
    //ROS_INFO("aruco Position: %f %f %f", aruco_pose[0], aruco_pose[1], aruco_pose[2]);


}
```

```
ros::Subscriber aruco_pose_sub = n.subscribe("/aruco_single/pose", 1, arucoPoseCallback);
```

We noticed that the aruco_marker frame didn't correspond to the real position of the Aruco so we changed the dimension of the Aruco to fix this error.

```
<box>
  <size>0.10 0.10 0.002</size>
</box>
```

Once the robot reaches the **obstacle_9** and detects the **aruco_marker** it retrieves its pose. Then it goes to a new goal with the following coordinates: x = x_m+1 and y = y_m.

In order to obtain these results we added these lines:

```
if(aruco_pose_available && arrived){
    goal_aruco.target_pose.pose.position.x = aruco_pose[0]+1;
    goal_aruco.target_pose.pose.position.y = aruco_pose[1];
    goal_aruco.target_pose.pose.position.z = aruco_pose[2];

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


    ROS_INFO("Positioning 1 meter from Aruco");
    ac.sendGoal(goal_aruco);

    ac.waitForResult();

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

    ROS_INFO("The mobile robot is in front of the Aruco");
```

4.c Publish the ArUco pose as TF following the example at this link.

To publish the ArUco pose as TF we defined a new **tf::Transform**.
We defined a static object **tf::TransformBroadcaster** to send the Aruco transform. This class publishes the coordinate frame transform information. It handles the messaging and stuffing of messages.

```
#include <tf/transform_broadcaster.h>
```

```
tf::Transform transform;
transform.setOrigin(tf::Vector3(aruco_pose[0],aruco_pose[1],aruco_pose[2]));
tf::Quaternion q(aruco_pose[3],aruco_pose[4],aruco_pose[5],aruco_pose[6]);
transform.setRotation(q);

// Pubblica la trasformazione TF
static tf::TransformBroadcaster br;
br.sendTransform(tf::StampedTransform(transform, ros::Time::now(), "map", "aruco_marker_frame"));
```

To check if it works, we used the command **rosrun tf tf_echo /map /aruco_marker_frame**

```
At time 55.704
- Translation: [-17.020, 8.209, 0.268]
- Rotation: in Quaternion [0.512, 0.498, 0.490, 0.501]
            in RPY (radian) [1.589, -0.003, 1.546]
            in RPY (degree) [91.063, -0.182, 88.603]
```

You can find all the files at the following GitHub url: https://github.com/FalcoPietro/Homework4

Group members:
Davide Busco
Pietro Falco
Davide Rubinacci
Giuseppe Saggese