Homework 4

Andrea Morghen, Maria Vittoria Cinquegrani, Valentina Giannotti

December 21, 2023

GitHub link: https://github.com/Andremorgh/RL_HW_04
GitHub link: https://github.com/MVCinquegrani/ROS_Homework4
GitHub link: https://github.com/ValentinaGiannotti/Homework4

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

1a Modify the starting position

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.

These lines of code in the file spawn_fra2mo_gazebo.launch have been changed:

```
  <!-- 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"/>
  <arg name="y_pos" default="5.0"/>
  <arg name="z_pos" default="0.1"/>
<arg name="yaw_pos" default="-1.57"/>

  ...

  <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_pos) -param robot_description"/>
```

1b Move the obstacle 9

Modify the world file of ***rl_racefield*** *moving the obstacle 9 in position:*
`x = -17, y = 9, z = 0.1, yaw = 3.14`

The position of obstacle 9 was changed as follows:

```
...
    <include>
      <name>obstacle_09</name>
      <pose> -17 9 0.1 0 0 3.14159</pose>
      <uri>model://obstacle_09</uri>
    </include>
...
```

1c Place the ArUco marker number 1151 on obstacle 9

Place the ArUco marker number 1151 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.

The image of the Aruco marker was downloaded from the given link. Obstacle 9 was subsequently modified by adding a material folder to its model folder, within which there are texture and script folders. The png file of the Aruco marker (ar_tag.png) was added to the texture folder, and an obstacle_09.material file was inserted into the scripts folder as follows:

```
    material obstacle_09
{
        technique
        {
                pass
                {
                        texture_unit
                        {
                                texture ar_tag.png
                        }
                }
        }
}
```

In the obstacle_09.sdf file a new visual "tag" have been added in order to visualize the aruco marker. The aruco marker is placed on the side of the obstacle and its side is 20cmx20cm.

```
<?xml version="1.0" ?>
<sdf version="1.5">
  <model name="obstacle_09">

...
      <visual name="tag">
        <pose>-0.005 0.75 0.25 0 0 1.57</pose>
        <geometry>
          <box>
            <size>0.2 0.005 0.2</size>
          </box>
        </geometry>
        <material>
          <script>
            <uri>model://obstacle_09/material/scripts</uri>
            <uri>model://obstacle_09/material/textures</uri>
            <name>obstacle_09</name>
          </script>
        </material>
      </visual>
```

```
...

    </link>
  </model>
</sdf>
```

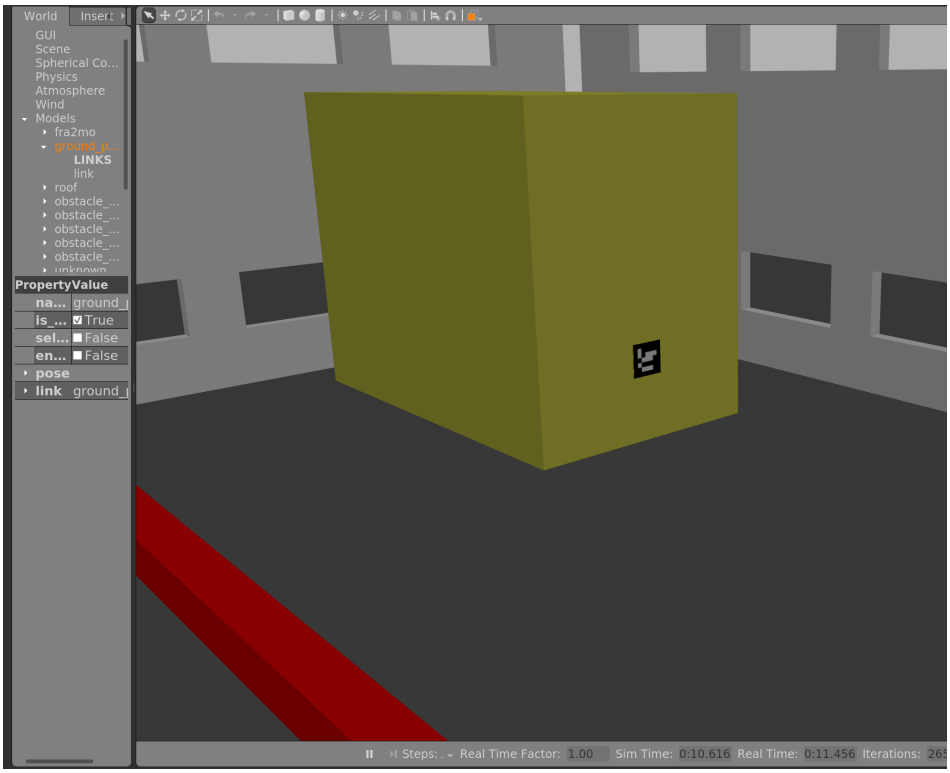

Figure 1: aruco marker

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

2a Insert 4 static tf acting as goals

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.

The file spawn_fra2mo_gazebo.launch has been modified. Four static transformation nodes (static_transform_publisher) have been created to represent targets positioned at specific coordinates relative to the map frame. The first three coordinates indicate the position, while the second three represent the orientation of the object along the roll, pitch, and yaw axes.

```
    <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 0.5235 1 map goal2 100" />
  <node pkg="tf" type="static_transform_publisher"
  name="goal_3_pub" args="-6    8 0 0 0 3.1415 1 map goal3 100" />
  <node pkg="tf" type="static_transform_publisher"
  name="goal_4_pub" args="-17.5 3 0 0 0 1.3090 1 map goal4 100" />
```

2b implement tf listeners to get target poses

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.

A multiple_goal.cpp file has been implemented and in the first lines of code a TF listener has been created using the tf::TransformListener class provided by the tf package.The listener is queried for a specific transformation using the lookupTransform method. This method waits for the specified transformation to become available and then retrieves it.

```
#include <ros/ros.h>
#include "../include/tf_nav.h"
#include <vector>
#include <cmath>
#include <tf2/LinearMath/Quaternion.h>
#include <move_base_msgs/MoveBaseAction.h>
#include <actionlib/client/simple_action_client.h>

typedef actionlib::SimpleActionClient<move_base_msgs::MoveBaseAction>
    MoveBaseClient;
// the order of the goal is decided here
std::vector<std::string> traj = {"goal3", "goal4", "goal2", "goal1"};

const float toRadians = M_PI/180.0;

int main(int argc, char** argv){
  ros::init(argc, argv, "multiple_navigation_goals");

  tf::TransformListener listener;
  tf::StampedTransform transform;
  ros::Rate r(1);

  move_base_msgs::MoveBaseGoal goal;

  //tell the action client that we want to spin a thread by default
  MoveBaseClient ac("move_base", true);

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

  for(int i = 0; i < traj.size(); i++){
    try{
      listener.waitForTransform( "map", traj[i], ros::Time(0), ros::Duration
          (10.0));
      listener.lookupTransform( "map", traj[i], ros::Time(0), transform);

      // print current position and orientation
      ROS_INFO("Current fra2mo position:\t\t %f, %f, %f", transform.getOrigin
          ().x(), transform.getOrigin().y(), transform.getOrigin().z());
      ROS_INFO("Current fra2mo orientation (quaterinion) [x,y,z,w]:\t %f, %f,
          %f, %f", transform.getRotation().x(), transform.getRotation().y(),
          transform.getRotation().z(), transform.getRotation().w());
    }
    catch( tf::TransformException &ex ) {
      ROS_ERROR("%s", ex.what());
      r.sleep();
      continue;
    }
...
```

2c multiple goals

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 the multiple_goal.cpp file we also implemented the Action Client communication protocol as it is shown here:

```
...
// the order of the goal is decided here
std::vector<std::string> traj = {"goal3", "goal4", "goal2", "goal1"};
...

    // upload the goal
    goal.target_pose.header.frame_id = "map";
    goal.target_pose.header.stamp = ros::Time::now();
    goal.target_pose.pose.position.x = transform.getOrigin().x();
    goal.target_pose.pose.position.y = transform.getOrigin().y();
    goal.target_pose.pose.orientation.x = transform.getRotation().x();
    goal.target_pose.pose.orientation.y = transform.getRotation().y();
    goal.target_pose.pose.orientation.z = transform.getRotation().z();
    goal.target_pose.pose.orientation.w = transform.getRotation().w();

    // send the goal and waiting for the result
    ROS_INFO("Sending %i goal", i+1);
    ac.sendGoal(goal);
    ac.waitForResult();

    if(ac.getState() == actionlib::SimpleClientGoalState::SUCCEEDED){
      ROS_INFO("Hooray, the base moved\n");
      ros::Duration(2.0).sleep();
    }
    else{
      ROS_INFO("The base failed to move for some reason");
      i = 5;
    }
  }
   return 0;
 }
```

To create the bag file, the nodes of interest and tasks were executed, after which the rosbag record command was used to record the topic /fra2mo/pose. The resulting bag file was named 'trajectory.bag' and saved in the 'bagfile' folder.

```
rosbag record -O trajectory.bag  /fra2mo/pose
```

Next, to view the contents of the bag file, the

```
rqt_bag trajectory.bag
```

command was executed. This opened the graphical interface of rqt_bag, showing position graphs representing the trajectory. Visual analysis of these graphs provided a dynamic representation of the robot's trajectory over time.

The recording of the bag file is this:

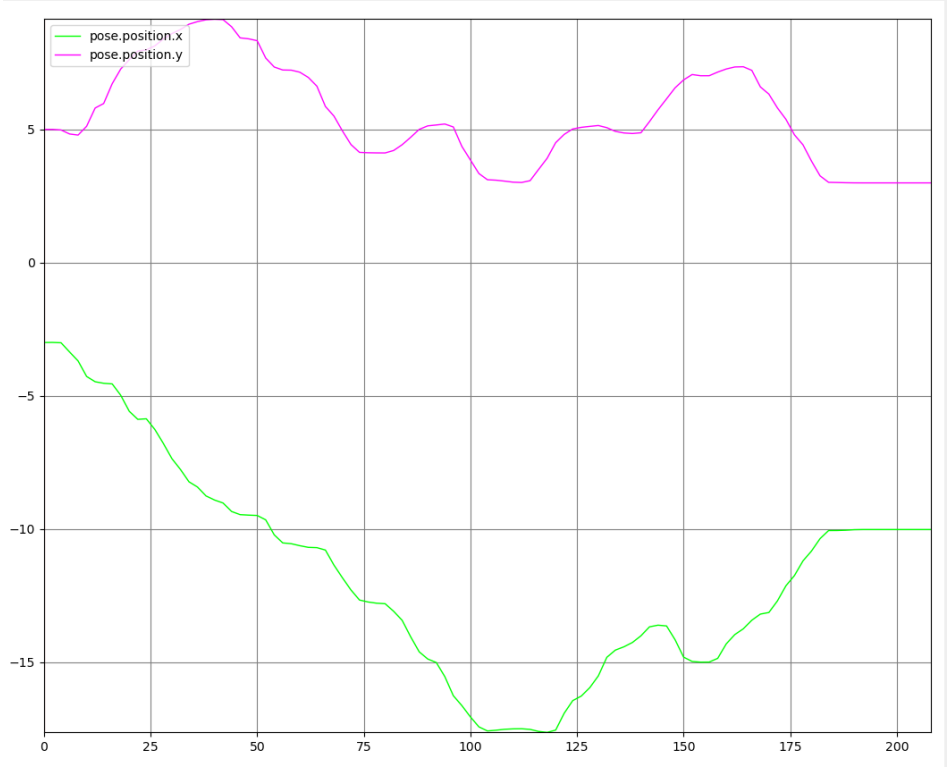

Figure 2: position

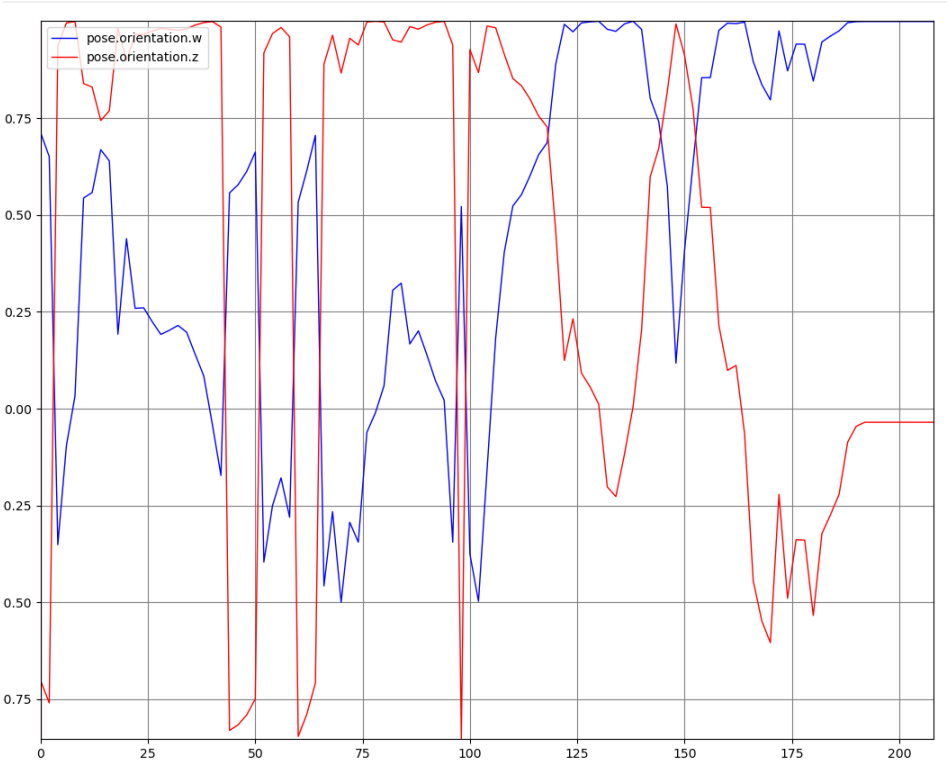

Figure 3: orientation

3 Map the Environment Tuning the Navigation Stack's Parameters

3a Modify, Add, Remove, or Change Pose:

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

In order to complete the entire environment map, two targets were added and the previous ones in the file spawn_fra2mo.launch modified.

```
<?xml version="1.0" ?>
<launch>
  <!-- these are the arguments you can pass this launch file, for example
      paused:=true -->

  ...

  <!--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
        9.5 0 0 0 0.5235 1 map goal2 100" />
  <node pkg="tf" type="static_transform_publisher" name="goal_3_pub" args="-6
        8 0 0 0 3.1415 1 map goal3 100" />
  <node pkg="tf" type="static_transform_publisher" name="goal_4_pub" args="
      -17.5 3 0 0 0 1.3090 1 map goal4 100" />
  <node pkg="tf" type="static_transform_publisher" name="goal_5_pub" args="-1
      0.3 0 0 0 1.3090 1 map goal5 100" />
  <node pkg="tf" type="static_transform_publisher" name="goal_6_pub" args="
      -0.5  9.3 0 0 0 3.1415 1 map goal6 100" />

</launch>
```

In addition, the file multiple_goal.cpp was modified by adding another goal as follows:

```
  std::vector<std::string> traj = {"goal6", "goal3", "goal4", "goal2", "goal1
      ", "goal5"};
```

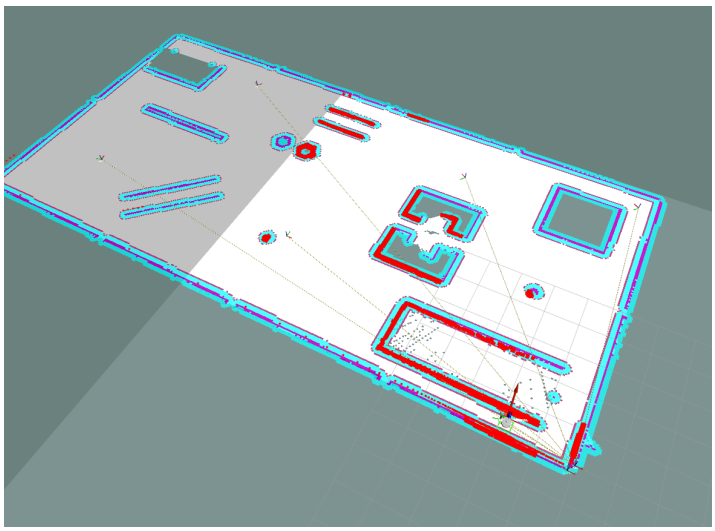

Figure 4: map

3b Change Planner and Move_Base Parameters

1. Try at least 4 different configurations by changing parameters in the following files:
   - In `teb_local_planner_params.yaml`, tune parameters related to the trajectory, robot, and obstacles. Generate it here.
   - In `local_costmap_params.yaml` and `global_costmap_params.yaml`, change dimensions' values and update costmaps' frequency.
   - In `costmap_common_params.yaml`, tune parameters related to the obstacle and raytrace ranges and footprint coherently as done in the planner parameters.
2. Comment on the results you obtain in terms of robot trajectories.

Several attempts have been made, in the `teb_local_planner_params.yaml` file this change have been done:

```
TebLocalPlannerROS:

 ...

 # Trajectory
 ...
 dt_ref: 1.0 #0.5 old parameter
 ...
 max_global_plan_lookahead_dist: 4.0 #2.0 old parameter
 ...

 # Robot

 max_vel_x: 0.4 #0.6 old parameter
 max_vel_x_backwards: 0.2 #0.3 old parameter
 max_vel_theta: 0.9 #0.3 old parameter
 ...

 # GoalTolerance

 ...

 # Obstacles

 min_obstacle_dist: 0.20 #0.10 old parameter
 ...
 inflation_dist: 1.0 #0.1 old parameter
 ...

 # Optimization
 ...

 # Homotopy Class Planner
 ...
```

In this file the changes are the most impactful ones, in particular increasing the max_vel_tetha makes the robot more dexterous and the inflation_dist and the min_obstacle_dist make sure the robot doesn't stuck or waste time in proximity of obstacles.

In the `local_costmap_params.yaml` file this change have been done:

```
local_costmap:
  ...
  update_frequency: 25.0 #5.0 old parameter
  publish_frequency: 50.0 #10.0 old parameter
  ...
```

These changes don't have much effect on the path planned by the planner but they made the simulation plan slightly more accurate.

In the `global_costmap_params.yaml` file this change have been done:

```
global_costmap:
  ...
  update_frequency: 25.0 #5.0 old parameter
  publish_frequency: 10.0 #2.0 old parameter
  ...
```

These changes follow the ones done in the previous file.

In the `costmap_common_params.yaml` file this change have been done:

```
global_costmap:
...
obstacle_range: 5.0 #7.0 old parameter
raytrace_range: 6.0 #8.0 old parameter
footprint: [[0.15, -0.15],
            [-0.15, -0.15],
            [-0.15, 0.15], #0.2 old parameter],
            [0.15, 0.15]]
```

These changes improve some choices made by the planner in some situation but they are not that impactful. The last change is made in order to have cooherence with the parameters declaration in the `teb_local_planner_params.yaml` file.

4 Vision-based navigation of the mobile platform

4a Run ArUco ROS node using the robot camera

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.

The ros aruco node was run by uncommenting it in the file fra2mo.xacro of the mobile robot rl_fra2mo_description and install the camera description pkg: sudo apt-get install ros-noetic-realsense2-description.

4b Implement a 2D navigation task

Implement a 2D navigation task following this logic:

1. Send the robot in the proximity of obstacle 9. the file aruco_search.cpp was created in which a trajectory consisting of 3 goals was implemented to bring the robot close to the obstacle 9

   We created a new aruco_search.cpp file, in the first part of this file the robot is guided, using some goals, near the obstacle 9:

```
#include <ros/ros.h>
#include "../include/tf_nav.h"
#include <vector>
#include <cmath>
#include <tf2/LinearMath/Quaternion.h>
#include <move_base_msgs/MoveBaseAction.h>
#include <tf/transform_broadcaster.h>
#include <actionlib/client/simple_action_client.h>

typedef actionlib::SimpleActionClient<move_base_msgs::MoveBaseAction>
    MoveBaseClient;
const float toRadians = M_PI/180.0;

std::vector<std::vector<double>> goals(4);
std::vector<double> fra2mo_pose(7,0.0), aruco_pose(7,0.0);
bool aruco_pose_available = false, find_des_pose = false, task_ended =
    false;

void arucoPoseCallback(const geometry_msgs::PoseStamped & msg)
{
    ...
}

int main(int argc, char** argv){
  ros::init(argc, argv, "aruco_search");
  ros::NodeHandle nh;
  tf::TransformListener listener;
  tf::TransformBroadcaster broadcaster;
  tf::StampedTransform base_footprint_tf, aruco_pose_tf;

  // Rate
  ros::Rate loop_rate(10);

  // Subscribers to node
  ros::Subscriber aruco_pose_sub = nh.subscribe("/aruco_single/pose", 1,
      arucoPoseCallback);
  ros::Publisher aruco_pose_pub = nh.advertise<geometry_msgs::PoseStamped
      >( "/aruco/pose", 1);

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

```
  MoveBaseClient ac("move_base", true);

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

  // define base variable for client-server comunication
  move_base_msgs::MoveBaseGoal goal;
  Eigen::Vector3d des_pose;

  // define the trajectory to make the robot go near by the obstacle 9
  tf2::Quaternion traj_orient, oriz_orient;
  traj_orient.setRPY( 0, 0, 180.0*toRadians);
  Eigen::Vector3d P_1 = {-6, 8.0, 1};
  Eigen::Vector3d P_2 = {-11.0, 4.0, 1};
  Eigen::Vector3d P_3 = {-13.5, 8.0, 1};
  std::vector<Eigen::Vector3d> traj = {P_1, P_2, P_3};

  // execute the trajectory
  for (int i=0; i<3; i++){

    // upload the goal
    goal.target_pose.header.frame_id = "map";
    goal.target_pose.header.stamp = ros::Time::now();
    goal.target_pose.pose.position.x = traj[i][0];
    goal.target_pose.pose.position.y = traj[i][1];

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

    // send the goal and waiting for the result
    ROS_INFO("Sending %i goal", i+1);
    ac.sendGoal(goal);
    ac.waitForResult();

    if(ac.getState() == actionlib::SimpleClientGoalState::SUCCEEDED){
      ROS_INFO("Hooray, the base moved\n");
      ros::Duration(2.0).sleep();
    }
    else{
      ROS_INFO("The base failed to move for some reason");
    }
  }

    ...

  return 0;
}
```

2. Make the robot look for the ArUco marker. Once detected, retrieve its pose with respect to the map frame.

   In the fra2mo_nav_bringup.launch file we implemented a callback to the usb_cam_aruco.launch file with same parameters changed in order to make the robot looking through the camera installed (depth_camera/depth_camera_image_row) for the right aruco code(115) and compute the right distance from the marker(markerSize changed):

```
        <launch>

        <include file="$(find rl_fra2mo_description)
        /launch/spawn_fra2mo_gazebo.launch"/>
        <include file="$(find rl_fra2mo_description)/launch
        /gmapping.launch"/>
        <include file="$(find fra2mo_2dnav)/launch/move_base.launch"/>
        <include file="$(find aruco_ros)/launch/usb_cam_aruco.launch">
            <arg name="markerId"    value="115"/>
            <arg name="markerSize"  value="0.2"/>
            <arg name="camera"      value="depth_camera/depth_camera"/>
        </include>

    </launch>
```

In the same way it was implemented in the Homework 3 we managed to compute the position of the marker:

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

...

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

  // variable for the research
  int j=1;
  Eigen::Vector3d p_map_to_object;
  Eigen::Matrix3d rot_map_to_object;
  while(ros::ok()){

    if(aruco_pose_available){

      //// compute the position of the marker in the map frame ////

      // position and orientation of the object in the camera frame
      Eigen::Vector3d p_cam_to_object  = {aruco_pose[0], aruco_pose[1],
          aruco_pose[2]};
      Eigen::Quaterniond quaternion_cam(aruco_pose[6], aruco_pose[3],
          aruco_pose[4], aruco_pose[5]);
      Eigen::Matrix3d rot_cam_to_object = quaternion_cam.toRotationMatrix
          ();

      // position and orientation of the camera in the footprint frame
      Eigen::Vector3d p_base_to_cam = {0.0975, 0.0, 0.065};
      Eigen::Matrix3d rot_base_to_cam =
        (Eigen::AngleAxisd(-90.0 * toRadians, Eigen::Vector3d::UnitZ()) *
         Eigen::AngleAxisd(0.0, Eigen::Vector3d::UnitY()) *
         Eigen::AngleAxisd(-90.0 * toRadians, Eigen::Vector3d::UnitX())).
             toRotationMatrix();


      // position and orientation of the footprint in the map frame
      try{
        listener.waitForTransform( "map", "base_footprint", ros::Time(0),
            ros::Duration(10.0));
        listener.lookupTransform( "map", "base_footprint", ros::Time(0),
            base_footprint_tf);
      }
      catch( tf::TransformException &ex ) {
        ROS_ERROR("%s", ex.what());
```

```
        loop_rate.sleep();
        continue;
      }
      Eigen::Vector3d p_map_to_base = {base_footprint_tf.getOrigin().x(),
          base_footprint_tf.getOrigin().y(), base_footprint_tf.getOrigin
          ().z()};
      Eigen::Quaterniond quaternion_base(base_footprint_tf.getRotation().
          w(), base_footprint_tf.getRotation().x(), base_footprint_tf.
          getRotation().y(), base_footprint_tf.getRotation().z());
      Eigen::Matrix3d rot_map_to_base = quaternion_base.toRotationMatrix
          ();

      // final computation
      Eigen::Vector3d p_base_to_object = p_base_to_cam + rot_base_to_cam*
          p_cam_to_object;
      p_map_to_object = p_map_to_base + rot_map_to_base*p_base_to_object;
      rot_map_to_object = rot_map_to_base*rot_base_to_cam*
          rot_cam_to_object;

      Eigen::Vector3d offset = {1.0, 0.0, 0.0};
      des_pose = p_map_to_object + offset;
      find_des_pose = true;

    }

    ...

    ros::spinOnce();
    loop_rate.sleep();
  }
  return 0;
}
```

The solution we provided doesn't make use of the other tf, we have chosen to compute the aruco pose in the same way of the honmework 3 (with the variable in the main plus the aruco relative pose) in order to optimize the delay (possibly caused by the listeners of the tf)

3. Set the following pose (relative to the ArUco marker pose) as the next goal for the robot:

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

where x_m and y_m are the marker coordinates.

The ArUco position was calculated, modified by appropriate offsets and set as a new goal as follows:

```
  while(ros::ok()){

    if(aruco_pose_available){

     ...

      Eigen::Vector3d p_map_to_base = {base_footprint_tf.getOrigin().x(),
          base_footprint_tf.getOrigin().y(), base_footprint_tf.getOrigin
          ().z()};
      Eigen::Quaterniond quaternion_base(base_footprint_tf.getRotation().
          w(), base_footprint_tf.getRotation().x(), base_footprint_tf.
          getRotation().y(), base_footprint_tf.getRotation().z());
      Eigen::Matrix3d rot_map_to_base = quaternion_base.toRotationMatrix
          ();
```

```
      // final computation
      Eigen::Vector3d p_base_to_object = p_base_to_cam + rot_base_to_cam*
          p_cam_to_object;
      Eigen::Vector3d p_map_to_object = p_map_to_base + rot_map_to_base*
          p_base_to_object;
      Eigen::Matrix3d rot_map_to_object = rot_map_to_base*rot_base_to_cam
          *rot_cam_to_object;

      Eigen::Vector3d offset = {1.0, 0.0, 0.0};
      //desire position of the next goal for the robot
      des_pose = p_map_to_object + offset;
      find_des_pose = true;

    }

    //Desired position set as new goal
    if(find_des_pose && !task_ended){

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

      goal.target_pose.pose.position.x = des_pose[0];
      goal.target_pose.pose.position.y = des_pose[1];

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

      ROS_INFO("Sending final pose");
      ac.sendGoal(goal);

      ac.waitForResult();

      if(ac.getState() == actionlib::SimpleClientGoalState::SUCCEEDED){
        ROS_INFO("Hooray, the base reached the desired position");
        ros::Duration(2.0).sleep();
      }
      else{
        ROS_INFO("The base failed to move for some reason");
      }

      task_ended = true;
    }

   ...

    ros::spinOnce();
    loop_rate.sleep();
  }
  return 0;
}
```

4c Publish the ArUco pose as TF

A tf as a publisher was used to publish the position 'aruco_pose' relative to the 'map' frame.

```
  while(ros::ok()){

    ...

    // Homework 4.c tf for the aruco pose
    aruco_pose_tf.stamp_ = ros::Time::now();
    aruco_pose_tf.frame_id_ = "map";
    aruco_pose_tf.child_frame_id_ = "aruco_pose";
    Eigen::Quaterniond quat_map_to_object(rot_map_to_object);
    tf::Quaternion quat_map_to_object_tf(quat_map_to_object.x(),
        quat_map_to_object.y(), quat_map_to_object.z(), quat_map_to_object.w()
        );

    aruco_pose_tf.setOrigin({p_map_to_object[0], p_map_to_object[1],
        p_map_to_object[2]});
    aruco_pose_tf.setRotation(quat_map_to_object_tf);

    broadcaster.sendTransform(aruco_pose_tf);

    ...
  }
  return 0;
}
```

In order to visualize the tf publication in another terminal run

```
rosrun tf tf_echo /map /aruco_pose
```