UNIVERSITA' DEGLI STUDI DI NAPOLI FEDERICO II

Scuola Politecnica e delle Scienze di Base
Corso di Laurea Magistrale in Ingegneria dell'Automazione e Robotica

Assignment Report: **Robotics Lab**

HOMEWORK

Anno Accademico 2023 - 2024

Studente
Luca Borriello P38/48
Docente
Prof. Selvaggio

Homework 1

Building Your Robot Manipulator

1 Create the description of your robot and visualize it in Rviz

It's possible to fetch all the implemented code to the following git repository:

```
https://github.com/RobLab23/Homework_1
```

1.1 Dowloading the *arm_description* into the workspace

In order to download the *arm_description* package into the *catkin_ws* from terminal:

```
$ roscd
$ cd ..
$ cd src
$ git clone https://github.com/RoboticsLab2023/arm_description.git
```

1.2 Robot visualization

In order to create the launch file from terminal:

```
$ roscd arm_description
$ mkdir launch
$ cd launch
$ touch display.launch
```

One fills the *display.launch* file as follows in vsCode:

```
<launch>

<!-- ROS param -->
<param name="robot_description" textfile="$(find
↪  arm_description)/urdf/arm.urdf" />
<!-- robot state publisher -->
<node name="robot_state_publisher" pkg="robot_state_publisher"
↪  type="robot_state_publisher" />
<!-- joint state publisher -->
<node name="joint_state_publisher_gui" pkg="joint_state_publisher_gui"
↪  type="joint_state_publisher_gui" />
<!-- rviz -->
<node name="rviz" pkg="rviz" type="rviz" args="-d $(find
↪  arm_description)/config/arm.rviz" required="true" />

</launch>
```

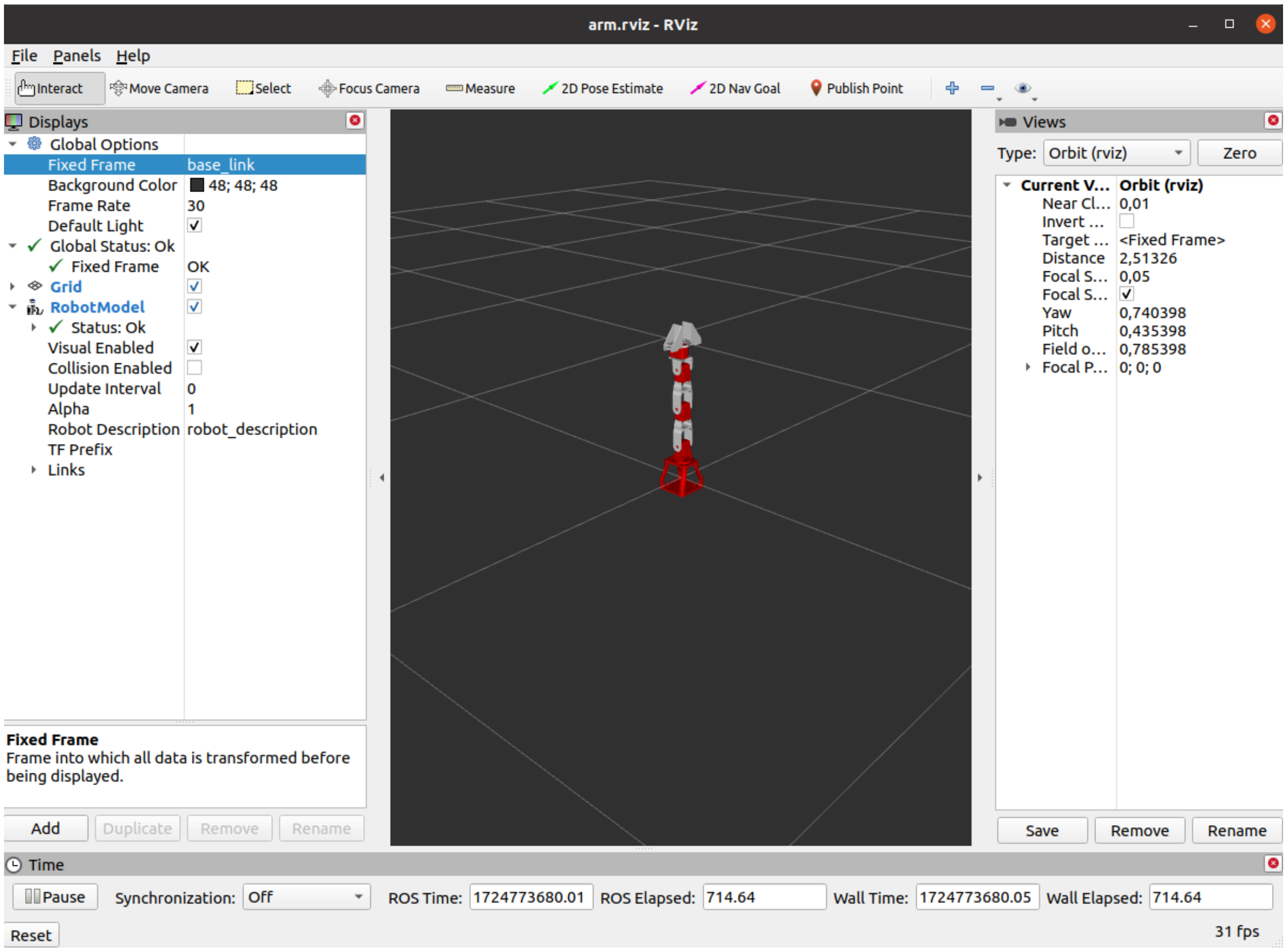
Then one can launch the file from terminal:

```
$ roslaunch arm_description display.launch
```

Once RViz opened [Fig.1], it's necessary to change the Fixed Frame checking "base_link" and to add the Robot Model plugin interface. Then, one saves the Rviz configuration into a file *arm.rviz* inside the *config*:

```
$ roscd arm_description
$ mkdir config
```

The configuration file is then passed as argument into the launch file.



```
Global Options
  Fixed Frame          base_link
  Background Color     48; 48; 48
  Frame Rate           30
  Default Light        ✓
✓ Global Status: Ok
  ✓ Fixed Frame        OK
Grid                   ✓
RobotModel             ✓
  ✓ Status: Ok
  Visual Enabled       ✓
  Collision Enabled
  Update Interval      0
  Alpha                1
  Robot Description    robot_description
  TF Prefix
  Links
```

Fixed Frame
Frame into which all data is transformed before being displayed.

```
Type: Orbit (rviz)
Current V...   Orbit (rviz)
  Near Cl...   0,01
  Invert ...
  Target ...   <Fixed Frame>
  Distance     2,51326
  Focal S...   0,05
  Focal S...   ✓
  Yaw          0,740398
  Pitch        0,435398
  Field o...   0,785398
  Focal P...   0; 0; 0
```

Synchronization: Off   ROS Time: 1724773680.01   ROS Elapsed: 714.64   Wall Time: 1724773680.05   Wall Elapsed: 714.64

31 fps

Figure 1: Arm visualization in RViz.

1.3 Collision meshes substitution

In order to substitute all the collision meshes with primitive shapes inside the *arm.urdf* file, the <box> tag has been used by trial and error adjusting the collision meshes size with geometries of reasonable size:

```
<geometry>
    <!-- <mesh filename="package://arm_description/meshes/base_link.stl"
     ↪ scale="0.001 0.001 0.001"/> -->
    <box size="0.1 0.1 0.1"/>
</geometry>
```

Similar code has been used for replacing all collision meshes. [Fig.2]

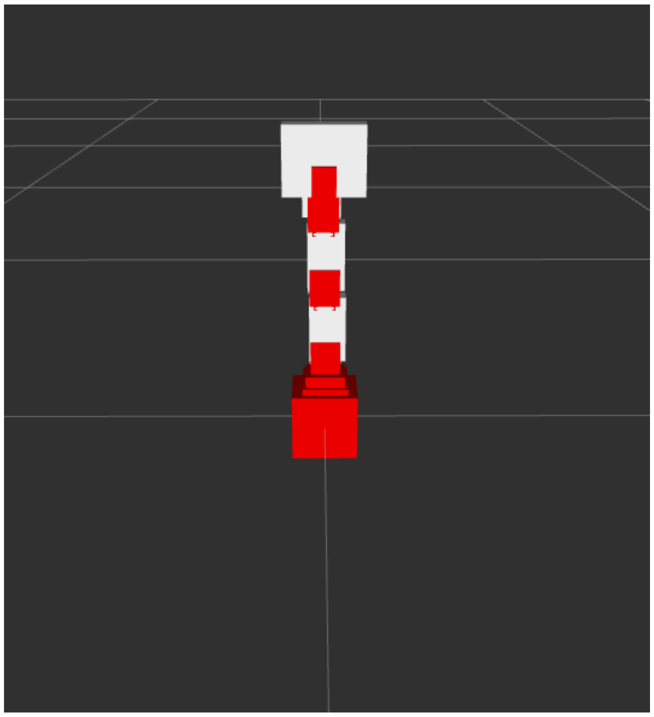


Figure 2: Replaced collision meshes.

1.4 Xacros definitions

In order to create the *arm.gazebo.xacro* file:

```
$ roscd arm_description/urdf
$ touch arm.gazebo.xacro
```

Then, the file has been filled with all the gazebo tags previously located inside the *arm.urdf* file:

```
<robot xmlns:xacro="http://www.ros.org/wiki/xacro">

    <xacro:macro name="arm_gazebo">

    <gazebo reference="f4">
        <material>Gazebo/Red</material>
    </gazebo>

    <gazebo reference="f5">
        <material>Gazebo/Red</material>
    </gazebo>

    <gazebo reference="wrist">
        <material>Gazebo/Red</material>
    </gazebo>

    <gazebo reference="crawer_base">
        <material>Gazebo/Red</material>
    </gazebo>

    <gazebo reference="base_link">
        <material>Gazebo/Red</material>
    </gazebo>

    <gazebo reference="base_turn">
        <material>Gazebo/Red</material>
    </gazebo>

    <gazebo reference="base_turn_rot">
        <material>Gazebo/Red</material>
    </gazebo>

    <gazebo reference="dyn2">
        <material>Gazebo/Black</material>
    </gazebo>

    <gazebo reference="dyn3">
        <material>Gazebo/Black</material>
    </gazebo>

    <gazebo reference="dyn4">
        <material>Gazebo/Black</material>
    </gazebo>

    <gazebo reference="dyn5">
        <material>Gazebo/Black</material>
    </gazebo>

    <gazebo reference="crawer_left">
```

```
        <material>Gazebo/Red</material>
    </gazebo>

    <gazebo reference="crawer_right">
        <material>Gazebo/Red</material>
    </gazebo>

    </xacro:macro>

</robot>
```

Then the *arm.urdf* file has been renamed into *arm.urdf.xacro* and the *arm.gazebo.xacro* file has been included inside it:

```
<robot name="arm" xmlns:xacro="http://www.ros.org/wiki/xacro"> <!-- Added
↪  xmlsn:xacro-->

  <!--Inclusions-->
  <xacro:include filename="$(find arm_description)/urdf/arm.gazebo.xacro" />
    .
    .
    .
</robot>
```

Finally, the *display.launch* file has been modified in order to match the new implementations:

```
<!-- <param name="robot_description" textfile="$(find
↪  arm_description)/urdf/arm.urdf" /> -->
<param name="robot_description" command="$(find xacro)/xacro $(find
↪  arm_description)/urdf/arm.urdf.xacro" />
```

2 Adding transmission and controllers to the robot and spawning it in Gazebo

2.1 Gazebo package creation

In order to create the *arm_gazebo* package:

```
$ roscd
$ cd ..
$ cd src
$ catkin_creat_pkg arm_gazebo
```

2.2 Launch creation

In order to create the *arm_world.launch* file within the proper directory:

```
$ roscd arm_gazebo
$ mkdir launch
$ cd launch
$ touch arm_world.launch
```

2.3 Spawning the robot in Gazebo

In order to fill the launch file:

```
<launch>

    <!-- similar to https://github.com/IFL-CAMP/iiwa_stack/blob/
    master/iiwa_gazebo/launch/iiwa_world.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"/>

    <!-- We resume the logic in empty_world.launch, changing only the name of
    ↪ the world to be launched -->
    <include file="$(find gazebo_ros)/launch/empty_world.launch">
        <arg name="debug" value="$(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>

    <!-- Load the URDF with the given hardware interface into the ROS
    ↪ Parameter Server -->
    <param name="robot_description" command="$(find xacro)/xacro $(find
    ↪ arm_description)/urdf/arm.urdf.xacro" />


    <!-- Run a python script to 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 arm -param
    ↪ robot_description"/>

</launch>
```

This following code line has been added to *arm.urdf.xacro* file in order to call the gazebo macro:

```
  <xacro:arm_gazebo/>
```

Then it's possible to launch gazebo [Fig.3]:

```
$ roslaunch arm_gazebo arm_world.launch
```

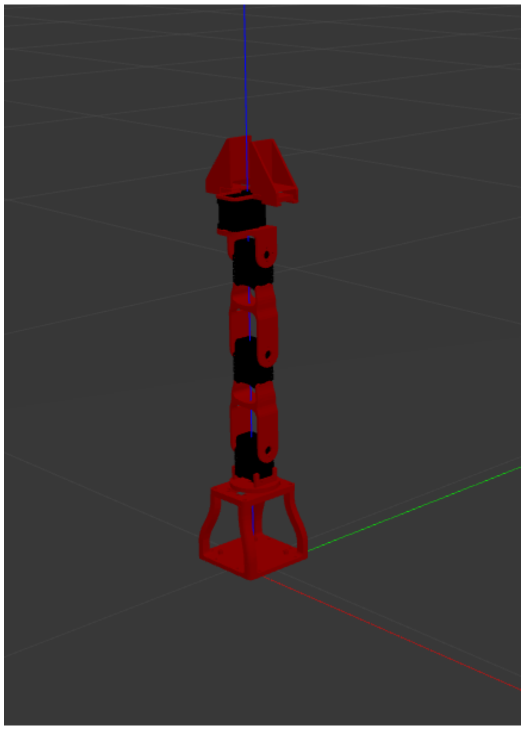


Figure 3: Robot visualization in Gazebo.

2.4 Adding PositionJointInterface

In order to create the *arm.transmission.xacro* file:

```
$ roscd arm_description/urdf
$ touch arm.transmission.xacro
```

The file has been filled in order to implement mechanical transmissions:

```
  <robot xmlns:xacro="http://www.ros.org/wiki/xacro">

    <xacro:macro name="transmission_block" params="joint_name">
    <transmission name="${joint_name}_tran">
        <type> transmission_interface/SimpleTransmission</type>
        <joint name="${joint_name}">
            <hardwareInterface> hardware_interface/PositionJointInterface
             ↪  </hardwareInterface>
        </joint>
        <actuator name="${joint_name}_motor">
```

```
            <hardwareInterface> hardware_interface/PositionJointInterface
             ↪  </hardwareInterface>
            <mechanicalReduction>1</mechanicalReduction>
        </actuator>
    </transmission>
    </xacro:macro>

</robot>
```

and called into the *arm.urdf.xacro* file as:

```
    <xacro:transmission_block joint_name="j0"/>
    <xacro:transmission_block joint_name="j1"/>
    <xacro:transmission_block joint_name="j2"/>
    <xacro:transmission_block joint_name="j3"/>
```

and it has been included inside the *arm.urdf.xacro* file:

```
  <xacro:include filename="$(find
  ↪  arm_description)/urdf/arm.transmission.xacro" />
```

2.5 Adding controls

In order to add controllers to the robot:

```
$ roscd
$ cd ..
$ cd src
$ catkin_create_pkg arm_control
$ cd arm_control
$ mkdir launch
$ mkdir config
$ cd launch
$ touch arm_control.launch
$ cd ..
$ cd config
$ touch arm_control.yaml
```

2.6 Control launch implementation

In order to fill the *arm_control.launch* file:

```
<launch>

    <!-- similar to https://github.com/IFL-CAMP/iiwa_stack/blob/master
    /iiwa_control/launch/iiwa_control.launch -->

    <!-- Launches the controllers according to the hardware interface
    ↪ selected -->
    <!-- Everythings is spawned under a namespace with the same name as the
    ↪ robot's. -->

    <arg name="hardware_interface" default="PositionJointInterface"/>
    <arg name="controllers" default="joint_state_controller
    ↪ PositionJointInterface_J0_controller
        PositionJointInterface_J1_controller
        PositionJointInterface_J2_controller
        PositionJointInterface_J3_controller"/>
    <arg name="robot_name" default="arm" />
    <arg name="joint_state_frequency" default="100" />
    <arg name="robot_state_frequency" default="100" />

    <!-- Loads joint controller configurations from YAML file to parameter
    ↪ server -->
    <rosparam file="$(find arm_control)/config/arm_control.yaml"
    ↪ command="load" />
    <param name="/$(arg robot_name)/joint_state_controller/publish_rate"
    ↪ value="$(arg joint_state_frequency)" />

    <!-- Loads the controllers -->
    <node name="controller_spawner" pkg="controller_manager" type="spawner"
    ↪ respawn="false" output="screen" args="$(arg controllers)" />

     <!-- Converts joint states to TF transforms for rviz, etc -->
    <node name="robot_state_publisher" pkg="robot_state_publisher"
    ↪ type="robot_state_publisher" respawn="false" output="screen">
        <remap from="joint_states" to="/$(arg robot_name)/joint_states" />
        <param name="publish_frequency" value="$(arg robot_state_frequency)"
        ↪ />
    </node>
</launch>
```

2.7 Control parameters configuration

In order to fill the *arm_control.yaml* file:

```
#arm:

  #similar to https://github.com/IFL-CAMP/iiwa_stack/blob/master
  /iiwa_control/config/iiwa_control.yaml ------

  # Publish all joint states ------------------------------------
  joint_state_controller:
    type: joint_state_controller/JointStateController
    publish_rate: 50


  # Controllers for singular joint ------------------------------------

  # Forward Position Controllers ---------------------------------------
  PositionJointInterface_J0_controller:
    type: position_controllers/JointPositionController
    joint: j0
  # Forward Position Controllers ---------------------------------------
  PositionJointInterface_J1_controller:
    type: position_controllers/JointPositionController
    joint: j1

  # Forward Position Controllers ---------------------------------------
  PositionJointInterface_J2_controller:
    type: position_controllers/JointPositionController
    joint: j2

  # Forward Position Controllers ---------------------------------------
  PositionJointInterface_J3_controller:
    type: position_controllers/JointPositionController
    joint: j3
```

2.8 Gazebo launch creation

In order to launch the simulation:

```
$ roscd arm_gazebo/launch
$ touch arm_gazebo.launch
```

The *arm_gazebo.launch* has been filled as follows:

```
<launch>

    <!-- similar to https://github.com/IFL-CAMP/iiwa_stack/blob/master
    /iiwa_gazebo/launch/iiwa_gazebo.launch -->

    <arg name="hardware_interface" default="PositionJointInterface" />
    <arg name="robot_name" default="arm" />

    <!-- Loads the Gazebo world. -->
    <include file="$(find arm_gazebo)/launch/arm_world.launch"> </include>

    <group ns="$(arg robot_name)">
        <!-- Spawn controllers - it uses a JPositionJointInterface -->
        <include file="$(find arm_control)/launch/arm_control.launch">
            <arg name="hardware_interface" value="$(arg hardware_interface)"
            ↪  />
            <arg name="controllers" value="joint_state_controller
                                   PositionJointInterface_J0_controller
                                   PositionJointInterface_J1_controller
                                   PositionJointInterface_J2_controller
                                   PositionJointInterface_J3_controller" />
            <arg name="robot_name" value="$(arg robot_name)" />
        </include>
    </group>

</launch>
```

Then, the *gazebo_ros_control* plugin has been added to the *arm.gazebo.xacro* file:

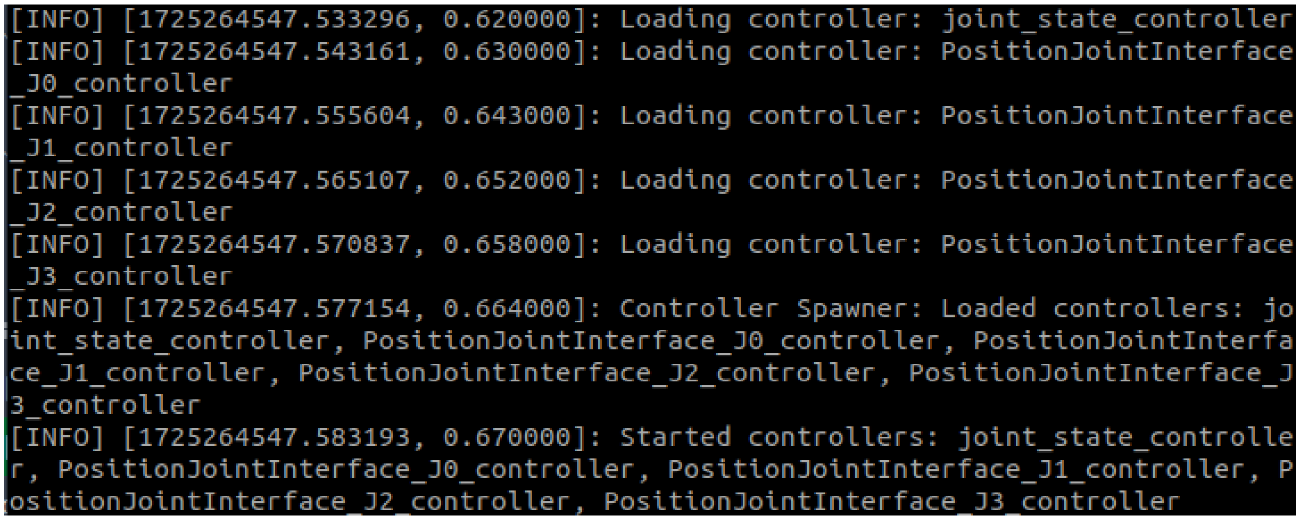
```
   <!-- Load Gazebo lib -->
    <gazebo>
        <plugin name="gazebo_ros_control"
         ↪  filename="libgazebo_ros_control.so">
```

```
            <robotNamespace>/arm</robotNamespace>
        </plugin>
        <robotSimType>gazebo_ros_control/DefaultRobotHWSim</robotSimType>
        <legacyModeNS>true</legacyModeNS>
    </gazebo>
```

Finally:

```
$ roslaunch arm_gazebo arm_gazebo.launch
```

One can check the log to verify the correct functioning of the controllers [Fig.4]:



```
[INFO] [1725264547.533296, 0.620000]: Loading controller: joint_state_controller
[INFO] [1725264547.543161, 0.630000]: Loading controller: PositionJointInterface
_J0_controller
[INFO] [1725264547.555604, 0.643000]: Loading controller: PositionJointInterface
_J1_controller
[INFO] [1725264547.565107, 0.652000]: Loading controller: PositionJointInterface
_J2_controller
[INFO] [1725264547.570837, 0.658000]: Loading controller: PositionJointInterface
_J3_controller
[INFO] [1725264547.577154, 0.664000]: Controller Spawner: Loaded controllers: jo
int_state_controller, PositionJointInterface_J0_controller, PositionJointInterfa
ce_J1_controller, PositionJointInterface_J2_controller, PositionJointInterface_J
3_controller
[INFO] [1725264547.583193, 0.670000]: Started controllers: joint_state_controlle
r, PositionJointInterface_J0_controller, PositionJointInterface_J1_controller, P
ositionJointInterface_J2_controller, PositionJointInterface_J3_controller
```

Figure 4: Terminal Log

3 Adding camera sensor

In order to define all the requirements for adding the camera, a *camera.xacro* file has been created:

```
$ roscd arm_description/urdf
$ touch camera.xacro
```

In order to check the controllers are correctly loaded:

3.1 Camera definition

In order to set the camera, the *camera.xacro* file has been filled as follows:

```
<?xml version="1.0"?>
<robot xmlns:xacro="http://www.ros.org/wiki/xacro" name="camera">

    <!-- similar to
    ↪  https://github.com/CentroEPiaggio/irobotcreate2ros/blob/master
    /model/camera.urdf.xacro -->

    <xacro:macro name="camera_sensor">

        <joint name="camera_joint" type="fixed">
            <axis xyz="0 1 0" />
            <origin xyz="0 0 0" rpy="0 0 0"/>
            <parent link="base_link"/>
            <child link="camera_link"/>
        </joint>

        <link name="camera_link">
            <collision>
                <origin xyz="0 0 0" rpy="0 0 0"/>
                <geometry>
                    <box size="0.02 0.08 0.05"/>
                </geometry>
            </collision>
            <visual>
                <origin xyz="0 0 0" rpy="0 0 0"/>
                <geometry>
                    <box size="0.02 0.08 0.05"/>
                </geometry>
            </visual>
            <inertial>
                <mass value="0.0001" />
                <origin xyz="0 0 0" rpy="0 0 ${pi}"/>
                <inertia ixx="0.0000001" ixy="0" ixz="0" iyy="0.0000001"
                 ↪  iyz="0" izz="0.0000001" />
            </inertial>
        </link>

    </xacro:macro>

</robot>
```

The file has been included and macro called inside the *arm.urdf.xacro* file:

```
<xacro:include filename="$(find arm_description)/urdf/camera.xacro" />
.
.
.
<xacro:camera_sensor/>
```

3.2 Camera sensor and plugin

In order to enable the camera the *arm.gazebo.xacro* has been filled with:

```
 <!-- camera -->
    <gazebo reference="camera_link">
        <sensor type="camera" name="camera">
            <update_rate>30.0</update_rate>
            <camera name="head">
                <horizontal_fov>1.3962634</horizontal_fov>
                <image>
                    <width>800</width>
                    <height>600</height>
                    <format>R8G8B8</format>
                </image>
                <clip>
                    <near>0.02</near>
                    <far>300</far>
                </clip>
                <noise>
                    <type>gaussian</type>
                    <mean>0.0</mean>
                    <stddev>0.007</stddev>
                </noise>
            </camera>

            <plugin name="camera_controller"
            ↪  filename="libgazebo_ros_camera.so">
                <alwaysOn>true</alwaysOn>
                <updateRate>0.0</updateRate>
                <cameraName>camera</cameraName>
                <imageTopicName>image_raw</imageTopicName>
                <cameraInfoTopicName>camera_info</cameraInfoTopicName>
                <frameName>camera_link_optical</frameName>
                <hackBaseline>0.0</hackBaseline>
                <distortionK1>0.0</distortionK1>
                <distortionK2>0.0</distortionK2>
                <distortionK3>0.0</distortionK3>
                <distortionT1>0.0</distortionT1>
                <distortionT2>0.0</distortionT2>
                <CxPrime>0</CxPrime>
                <Cx>0.0</Cx>
                <Cy>0.0</Cy>
                <focalLength>0.0</focalLength>
            </plugin>

        </sensor>
    </gazebo>
```

3.3 Camera simulation

In order to simulate the camera:

```
$ roslaunch arm_gazebo arm_gazebo.launch
```

and in another terminal:

```
$ rqt_image_view
```

and one chooses *\camera\image_raw* as topic to visualize. In order to verify the proper functioning of the camera, a green object has been added to the scene [Fig.5]. In Fig.6 is possible to check what the camera is observing.

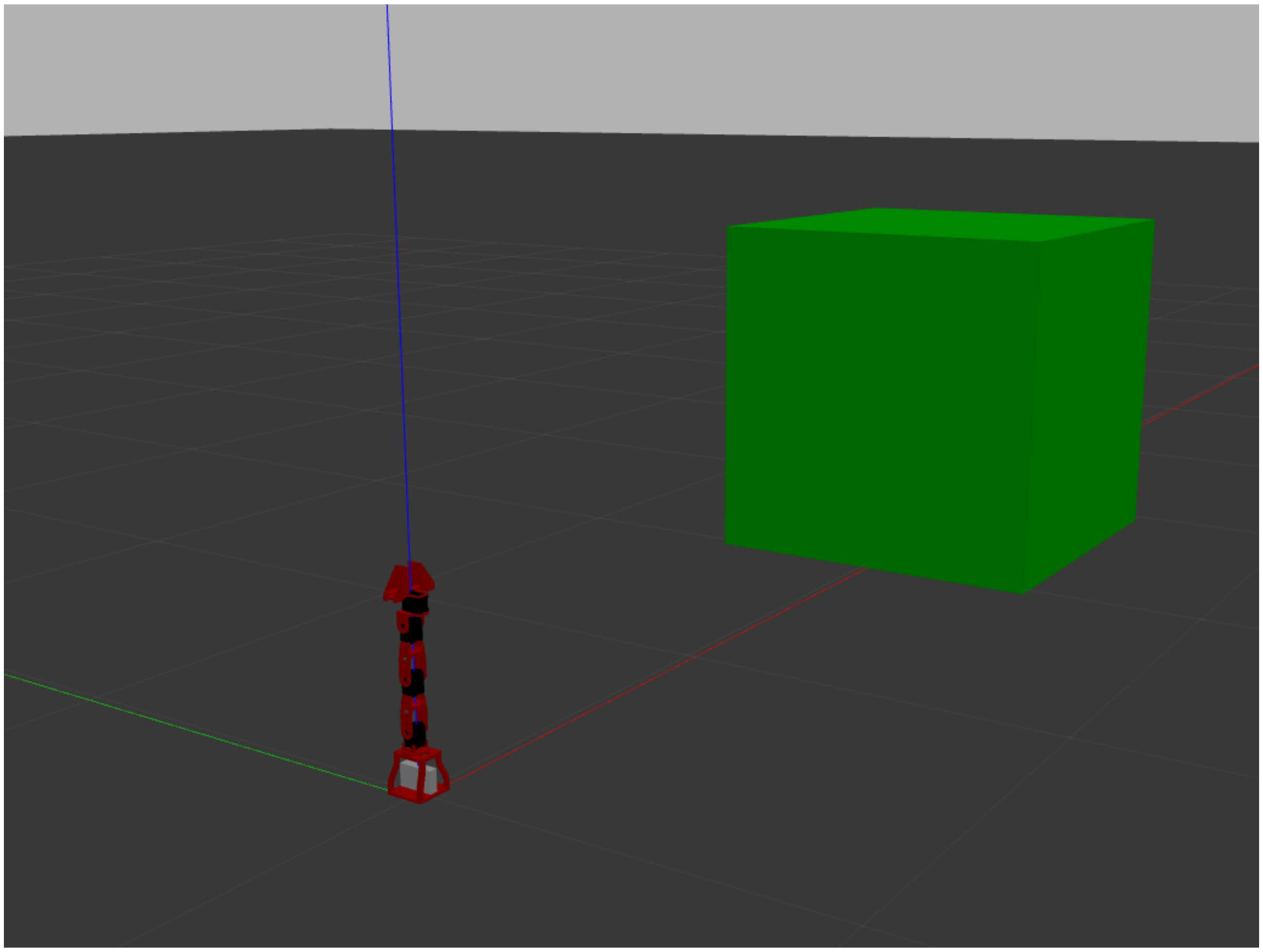


Figure 5: Gazebo scene.

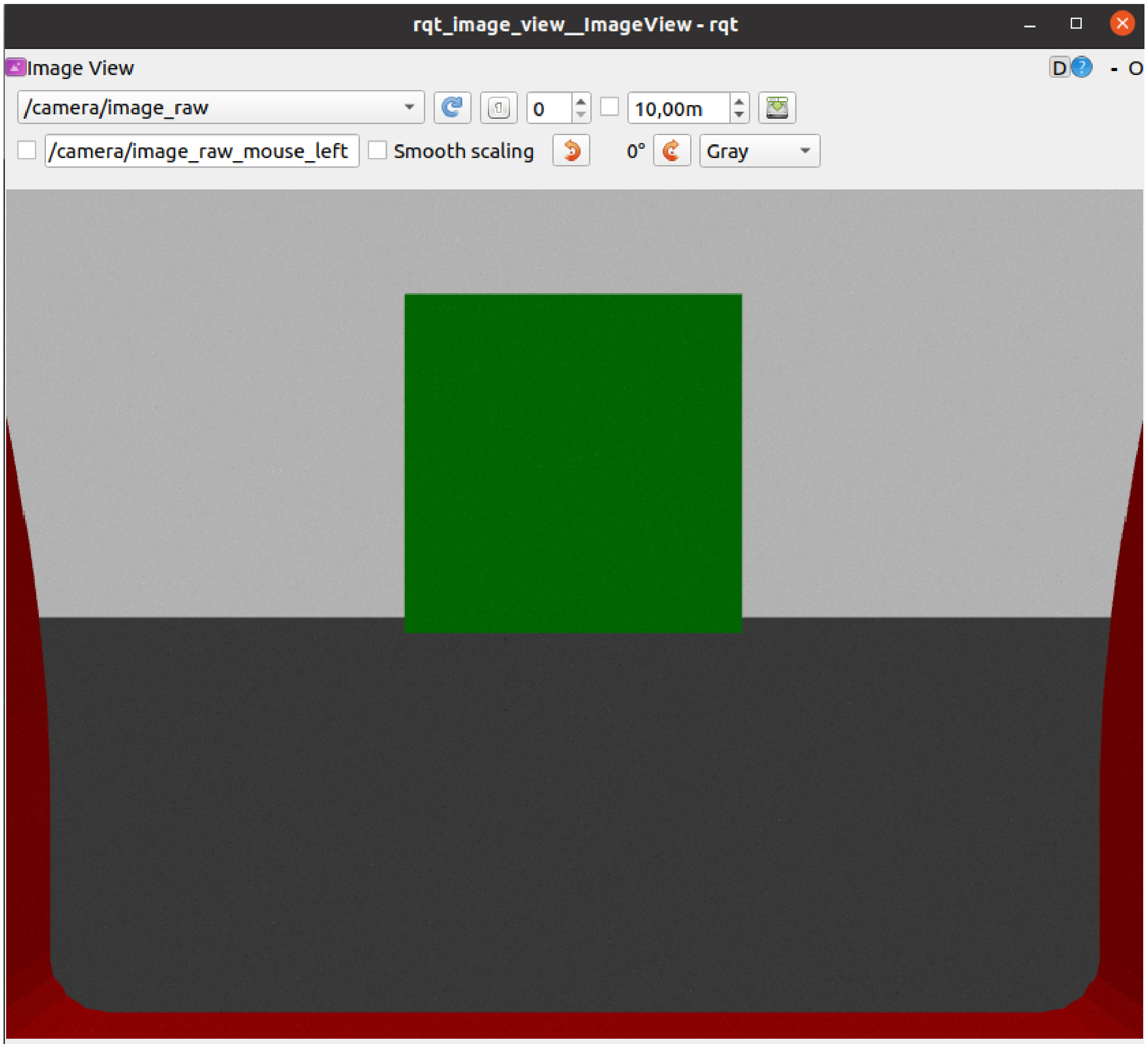


Figure 6: Camera.

4 ROS Publisher

4.1 Arm Controller Node

In order to create the *arm_controller* package with a ROS C++ node named *arm_controller_node*:

```
$ roscd
$ cd ..
$ catkin_create_pkg arm_controller roscpp sensor_msgs std_msgs
$ cd arm_controller/src
$ touch arm_controller_node.cpp
```

Then the *CMakeLists.txt* file has been properly modified in order to compile the node:

```
.
.
.
 add_executable(${PROJECT_NAME}_node src/arm_controller_node.cpp)
.
.
.
target_link_libraries(${PROJECT_NAME}_node
   ${catkin_LIBRARIES}
 )
  .
  .
  .
```

these lines code have been uncommented.
The *arm_controller_node.cpp* file has been initially set up to be completed in the next steps:

```
#include <ros/ros.h>
.
.
.
int main(int argc, char* argv[])
{
ros::init(argc, argv, "arm_controller_node");
ros::NodeHandle nodeHandle;
ros::Rate loopRate(10);
.
.
.
return 0;
}
```

4.2 Subscriber creation

In order to define the subscriber, the *arm_controller_node.cpp* file has been properly filled:

```
#include "sensor_msgs/JointState.h" //for Subscriber

void printJointPosition_CB(const sensor_msgs::JointState& msg)
{
    for(int i=0; i<4; i++){
        ROS_INFO("Joint_Position_[%i] : [%f]", i, msg.position[i]);
    }
}

int main(int argc, char* argv[])
{
.
.
.
ros::Subscriber subscriber =
↪   nodeHandle.subscribe("/arm/joint_states",1,printJointPosition_CB);
.
.
.
return 0;
}
```

4.3 Publishers creation

In order to define the publishers, the *arm_controller_node.cpp* file has been properly filled:

```
#include "std_msgs/Float64.h" //for Publishers

int main(int argc, char* argv[])
```

```
{

ros::Publisher jointPublisher0 = nodeHandle.advertise<std_msgs::Float64>
("/arm/PositionJointInterface_J0_controller/command", 1);
ros::Publisher jointPublisher1 = nodeHandle.advertise<std_msgs::Float64>
("/arm/PositionJointInterface_J1_controller/command", 1);
ros::Publisher jointPublisher2 = nodeHandle.advertise<std_msgs::Float64>
("/arm/PositionJointInterface_J2_controller/command", 1);
ros::Publisher jointPublisher3 = nodeHandle.advertise<std_msgs::Float64>
("/arm/PositionJointInterface_J3_controller/command", 1);

while(ros::ok()){
    std_msgs::Float64 pos0, pos1, pos2, pos3;
    pos0.data=0.1;
    pos1.data=0.2;
    pos2.data=0.3;
    pos3.data=0.4;

    jointPublisher0.publish(pos0);
    jointPublisher1.publish(pos1);
    jointPublisher2.publish(pos2);
    jointPublisher3.publish(pos3);

    ros::spinOnce();
    loopRate.sleep();
}

return 0;
}
```

4.4 Results

In order to compile the node:

```
$ roscd
$ cd ..
$ catkin_make
```

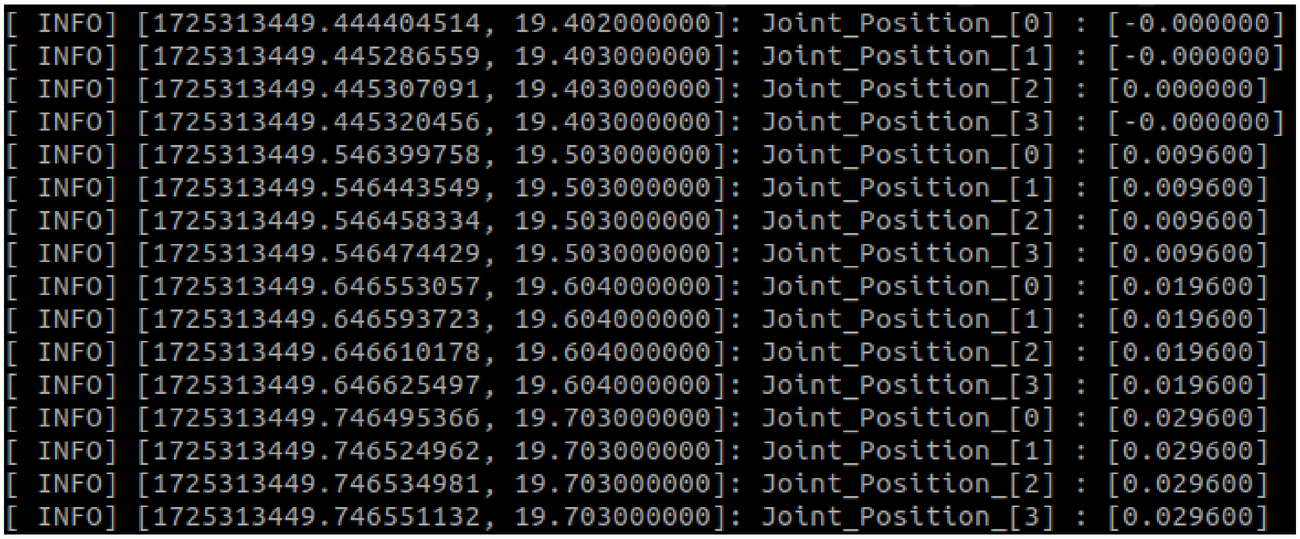
Then in a terminal:

```
$ roslaunch arm_gazebo arm_gazebo.launch
```

and in another terminal:

```
$ rosrun arm_controller arm_controller_node
```

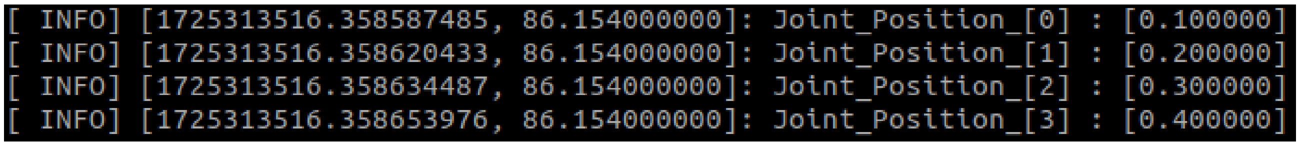
Once the node is running, the log will show the positions of the joints [Fig.7],



```
[ INFO] [1725313449.444404514, 19.402000000]: Joint_Position_[0] : [-0.000000]
[ INFO] [1725313449.445286559, 19.403000000]: Joint_Position_[1] : [-0.000000]
[ INFO] [1725313449.445307091, 19.403000000]: Joint_Position_[2] : [0.000000]
[ INFO] [1725313449.445320456, 19.403000000]: Joint_Position_[3] : [-0.000000]
[ INFO] [1725313449.546399758, 19.503000000]: Joint_Position_[0] : [0.009600]
[ INFO] [1725313449.546443549, 19.503000000]: Joint_Position_[1] : [0.009600]
[ INFO] [1725313449.546458334, 19.503000000]: Joint_Position_[2] : [0.009600]
[ INFO] [1725313449.546474429, 19.503000000]: Joint_Position_[3] : [0.009600]
[ INFO] [1725313449.646553057, 19.604000000]: Joint_Position_[0] : [0.019600]
[ INFO] [1725313449.646593723, 19.604000000]: Joint_Position_[1] : [0.019600]
[ INFO] [1725313449.646610178, 19.604000000]: Joint_Position_[2] : [0.019600]
[ INFO] [1725313449.646625497, 19.604000000]: Joint_Position_[3] : [0.019600]
[ INFO] [1725313449.746495366, 19.703000000]: Joint_Position_[0] : [0.029600]
[ INFO] [1725313449.746524962, 19.703000000]: Joint_Position_[1] : [0.029600]
[ INFO] [1725313449.746534981, 19.703000000]: Joint_Position_[2] : [0.029600]
[ INFO] [1725313449.746551132, 19.703000000]: Joint_Position_[3] : [0.029600]
```

Figure 7: Joint positions.

until the arm has reached the desired final position [Fig.8-9].



```
[ INFO] [1725313516.358587485, 86.154000000]: Joint_Position_[0] : [0.100000]
[ INFO] [1725313516.358620433, 86.154000000]: Joint_Position_[1] : [0.200000]
[ INFO] [1725313516.358634487, 86.154000000]: Joint_Position_[2] : [0.300000]
[ INFO] [1725313516.358653976, 86.154000000]: Joint_Position_[3] : [0.400000]
```

Figure 8: Final joint positions.

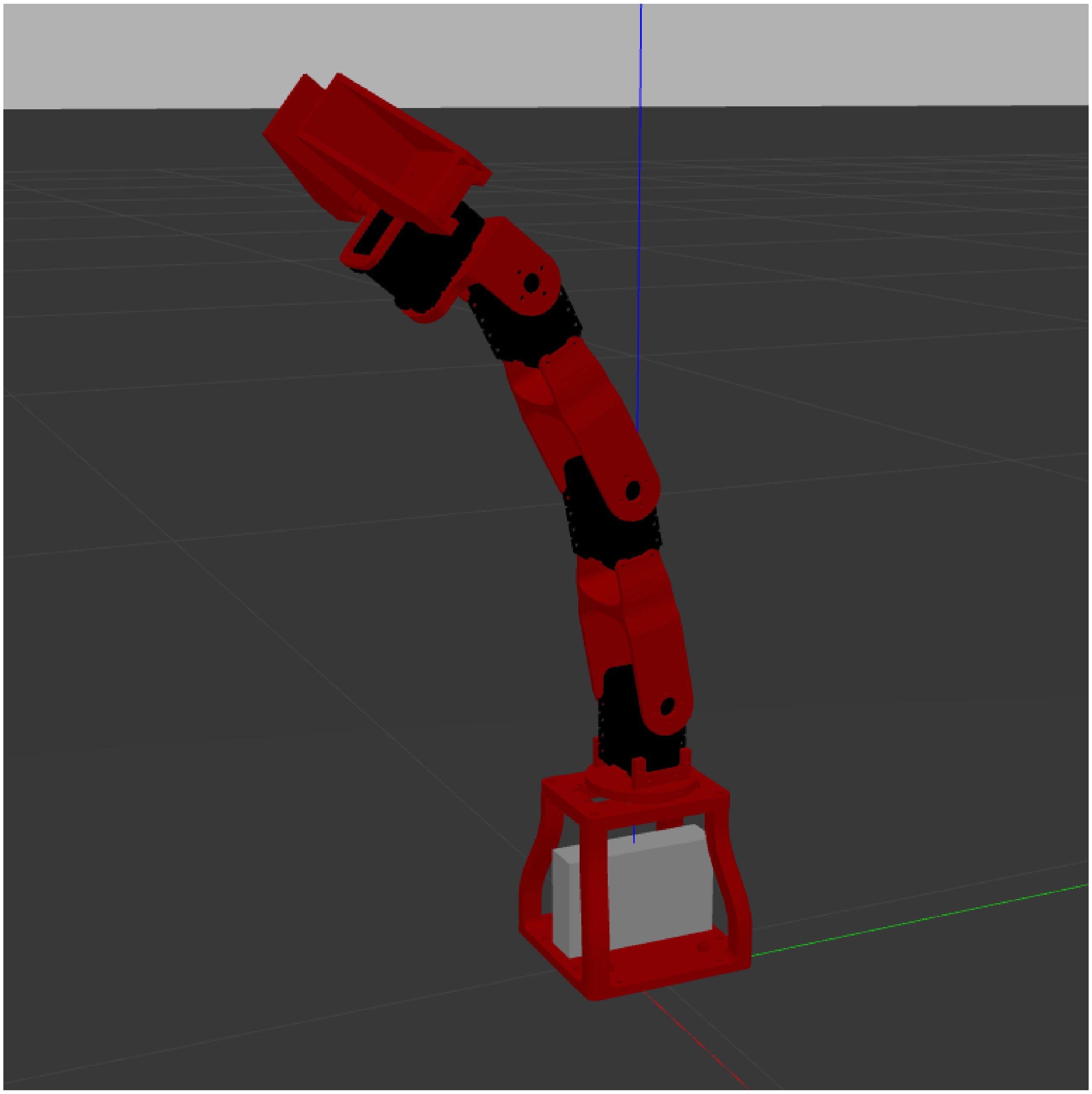


Figure 9: Final arm positions.

Homework 2

Control a Manipulator to Follow a Trajectory

1 Curvilinear abscissa definition

It's possible to fetch all the implemented code to the following git repository:

```
https://github.com/RobLab23/Homework_2
```

1.1 Trapezoidal velocity profile

In order to compute a trajectory with trapezoidal velocity profile the following lines of code have been implemented:

```
    //HomeWork 1.a -------------- kdl_planner.h ------------------------
    void trapezoidal_vel(double time, double accDuration_, double &s, double
     ↪  &ds, double &dds);
```

```
    //HomeWork 1.a -------------- kdl_planner.cpp ------------------------
void KDLPlanner::trapezoidal_vel(double time, double accDuration_, double &s,
 ↪  double &ds, double &dds)
{
  /*
    trapezoidal velocity profile with accDuration_ acceleration time period
     ↪  and trajDuration_ total duration.

     time = current time
     trajDuration_  = final time
     accDuration_   = acceleration time

    s,ds,dds = curvilinear abscissa  [0,1] and its derivatives
  */

  double ddsc =
  ↪  -1.0/(std::pow(accDuration_,2)-trajDuration_*accDuration_);//FR pag.166
  ↪  formula 4.5 with qi=0, qf=1, accDuration_<=trajDuration_/2

  if(time <= accDuration_)
  {
    s = 0.5*ddsc*std::pow(time,2);
    ds = ddsc*time;
    dds= ddsc;
  }
  else if(time <= trajDuration_-accDuration_)
  {
    s = ddsc*accDuration_*(time-accDuration_/2);
    ds = ddsc*accDuration_;
    dds = 0.0;
  }
  else
  {
    s = 1.0 - 0.5*ddsc*std::pow(trajDuration_-time,2);
    ds = ddsc*(trajDuration_-time);
    dds = -ddsc;
  }
}
//-------------------------------------------------------------------------------
```

1.2 Cubic polynomial profile

In order to compute a trajectory with cubic polynomial profile the following lines of code have been implemented:

```
    //HomeWork 1.b -------------- kdl_planner.h ------------------------
    void cubic_polinomial(double time, double &s, double &ds, double &dds);
    //HomeWork 1.b -------------- kdl_planner.cpp ------------------------
void KDLPlanner::cubic_polinomial(double time, double &s, double &ds, double
 ↪  &dds)
{
  /*
    cubic polinomial profile curvilinear abscissa:
    s(t) = a3*t^3 + a2*t^2 + a1*t + a0
    ds(t) = 3*a3*t^2 + 2*a2*t + a1
    dds(t) = 6*a3*t + 2*a2

     time = current time
     trajDuration_  = final time
     s,ds,dds = curvilinear abscissa  [0,1] and its derivatives
  */

  static Eigen::Matrix4d matCoeffs;
  static Eigen::Vector4d boundaries;
  static Eigen::Vector4d coeffs;

  /*
                        A*x == b

    s(0)  = a0 +   0    +   0      + 0          == 0
    ds(0) =   0 + a1    +   0      + 0          == 0
    s(tf) = a0 + a1*tf + a2*tf^2 + a3*tf^3    == 1
    ds(tf)=   0 + a1    + 2*a2*tf + 3*a3*tf^2 == 0
  */
  matCoeffs << 1,0,0,0,
               0,1,0,0,
               1,trajDuration_,pow(trajDuration_,2),pow(trajDuration_,3),
               0,1,2*trajDuration_,3*pow(trajDuration_,2),
  boundaries << 0,0,1,0;
  coeffs = matCoeffs.colPivHouseholderQr().solve(boundaries);

  s = coeffs(3)*pow(time,3) + coeffs(2)*pow(time,2) + coeffs(1)*time +
   ↪  coeffs(0);
  ds = 3*coeffs(3)*pow(time,2) + 2*coeffs(2)*time + coeffs(1);
  dds = 6*coeffs(3)*time + 2*coeffs(2);
}
```

```
//--------------------------------------------------------------------------
```

2 Trajectory definition

2.1 Trajectory Constructors

In order to properly define the trajectory, four constructors have been define, one per each kind of trajectory: Linear trajectory with trapezoidal velocity profile, Linear trajectory with cubic polynomial profile, Circular trajectory with trapezoidal velocity profile and Circular trajectory with cubic polynomial profile.

```
    //HomeWork 2.a -------------- kdl_planner.h ------------------------

    //Linear trajectory with trapezoidal velocity profile
    KDLPlanner(double _trajDuration, double _accDuration, Eigen::Vector3d
    ↪  _trajInit, Eigen::Vector3d _trajEnd);

    //Linear trajectory with cubic polinomial profile
    KDLPlanner(double _trajDuration, Eigen::Vector3d _trajInit,
    ↪  Eigen::Vector3d _trajEnd);

    //Circular trajectory with trapezoidal velocity profile
    KDLPlanner(double _trajDuration, double _accDuration, Eigen::Vector3d
    ↪  _trajInit, double _trajRadius);

    //Circular trajectory with cubic polinomial profile
    KDLPlanner(double _trajDuration, Eigen::Vector3d _trajInit, double
    ↪  _trajRadius);

//--------------------------------------------------------------------------

    double trajDuration_, accDuration_, trajRadius_; //HomeWork 2.a : radius
```

```
    //HomeWork 2.a -------------- kdl_planner.cpp ------------------------

//Linear trajectory with trapezoidal velocity profile
```

```cpp
KDLPlanner::KDLPlanner(double _trajDuration, double _accDuration,
 ↪  Eigen::Vector3d _trajInit, Eigen::Vector3d _trajEnd)
{
  trajDuration_ = _trajDuration;
  accDuration_  = _accDuration;
  trajInit_     = _trajInit;
  trajEnd_      = _trajEnd;
  trajRadius_   = -1;
}

//Linear trajectory with cubic polinomial profile
KDLPlanner::KDLPlanner(double _trajDuration, Eigen::Vector3d _trajInit,
 ↪  Eigen::Vector3d _trajEnd)
{
  trajDuration_ = _trajDuration;
  accDuration_  = -1;
  trajInit_     = _trajInit;
  trajEnd_      = _trajEnd;
  trajRadius_   = -1;
}

//Circular trajectory with trapezoidal velocity profile
KDLPlanner::KDLPlanner(double _trajDuration, double _accDuration,
 ↪  Eigen::Vector3d _trajInit, double _trajRadius)
{
  trajDuration_ = _trajDuration;
  accDuration_  = _accDuration;
  trajInit_     = _trajInit;
  trajEnd_      = _trajInit;
  trajRadius_   = _trajRadius;
}

//Circular trajectory with cubic polinomial profile
KDLPlanner::KDLPlanner(double _trajDuration, Eigen::Vector3d _trajInit,
 ↪  double _trajRadius)
{
  trajDuration_ = _trajDuration;
  accDuration_  = -1;
  trajInit_     = _trajInit;
  trajEnd_      = _trajInit;
  trajRadius_   = _trajRadius;
}
//---------------------------------------------------------------------------
```

2.2 Circular and linear trajectories

In order to define circular or linear trajectories the *compute_trajectory* function has been properly modified:

```
    //HomeWork 2.b-c -------------- kdl_planner.cpp ------------------------

trajectory_point KDLPlanner::compute_trajectory(double time)
{

  double s, ds, dds;
  //Profile selection
  if(accDuration_ < 0)
    cubic_polinomial(time, s, ds, dds);
  else
    trapezoidal_vel(time, accDuration_, s, ds, dds);

  trajectory_point traj;
  //HomeWork 2.c
  //Curve selection
  if(trajRadius_ < 0)
  {
    traj.pos = trajInit_ + s*(trajEnd_ - trajInit_);
    traj.vel = ds*(trajEnd_ - trajInit_);
    traj.acc = dds*(trajEnd_ - trajInit_);
  }
  //HomeWork 2.b
  else
  {

    traj.pos(0) = trajInit_(0);
    traj.pos(1) = trajInit_(1) - trajRadius_*std::cos(2*M_PI*s);
    traj.pos(2) = trajInit_(2) - trajRadius_*std::sin(2*M_PI*s);

    traj.vel(0) = 0;
    traj.vel(1) = trajRadius_*2*M_PI*std::sin(2*M_PI*s)*ds;
    traj.vel(2) = -trajRadius_*2*M_PI*std::cos(2*M_PI*s)*ds;

    traj.acc(0) = 0;


    traj.acc(1) =
    ↪   trajRadius_*std::pow(2*M_PI,2)*std::cos(2*M_PI*s)*std::pow(ds,2) +
    ↪   trajRadius_*2*M_PI*std::sin(2*M_PI*s)*dds;
    traj.acc(2) =
    ↪   trajRadius_*std::pow(2*M_PI,2)*std::sin(2*M_PI*s)*std::pow(ds,2) -
    ↪   trajRadius_*2*M_PI*std::cos(2*M_PI*s)*dds;
  }

  return traj;
}
//----------------------------------------------------------------------------
```

3 Testing

In order to test the four trajectories, first the inverse kinematics problem has been solved:

```
    //HomeWork 3.a-b -------------- kdl_robot.h ------------------------
void getInverseKinematics(KDL::Frame &f,
                              KDL::Twist &twist,
                              KDL::Twist &acc,
                              KDL::JntArray &q,
                              KDL::JntArray &dq,
                              KDL::JntArray &ddq);

    Eigen::VectorXd getFriction();
//----------------------------------------------------------------------------
```

```
    //HomeWork 3.a-b -------------- kdl_robot.cpp ------------------------
void KDLRobot::getInverseKinematics(KDL::Frame &f,
                                KDL::Twist &twist,
                                KDL::Twist &acc,
                                KDL::JntArray &q,
                                KDL::JntArray &dq,
                                KDL::JntArray &ddq)
{
    q = getInvKin(q,f);
    ikVelSol_->CartToJnt(q,twist,dq);

    Eigen::Matrix<double,6,7> J = toEigen(getEEJacobian());
    Eigen::VectorXd x_ddot = toEigen(acc);
    Eigen::VectorXd Jdot_qdot = getEEJacDotqDot();
    Eigen::Matrix<double,7,6> Jpinv = pseudoinverse(J);

    ddq.data = Jpinv*(x_ddot - Jdot_qdot); //pag. 141 formula 3.99 FR
}
//-----------------------------------------------------------------------------
```

Then the user interface has been modified in order to simplify the choice of the trajectory:

```
    //HomeWork 3.a -------------- kdl_robot_test.cpp ------------------------
KDLPlanner chooseTrajectory(int trajFlag, Eigen::Vector3d init_position,
 ↪  Eigen::Vector3d end_position, double traj_duration, double acc_duration,
 ↪  double t, double init_time_slot, double traj_radius)
{
    switch(trajFlag) {
        case 1:
            return KDLPlanner(traj_duration, acc_duration, init_position,
             ↪  end_position);
            break;
        case 2:
            return KDLPlanner(traj_duration, init_position, end_position);
            break;
        case 3:
            return KDLPlanner(traj_duration, acc_duration, init_position,
             ↪  traj_radius);
            break;
        case 4:
            return KDLPlanner(traj_duration, init_position, traj_radius);
            break;
        default:
            return KDLPlanner(traj_duration, acc_duration, init_position,
             ↪  end_position);
    }
}

int main ()
{
    .
    .
    .
    // Choosing Trajectory
    int trajFlag;
    std::cout << "Choose desired trajectory:" << std::endl
                << "1. Linear with trapezoidal profile\n" << "2. Linear with
                 ↪  cubic profile\n" <<
                "3. Circular with trapezoidal profile\n" << "4. Circular with
                 ↪  cubic profile\n";
    std::cout << "Insert number and press Enter: ";
    std::cin >> trajFlag;
    .
    .
    .
    double traj_radius;
    KDLPlanner planner = chooseTrajectory(trajFlag, init_position,
     ↪  end_position, traj_duration, acc_duration, t, init_time_slot,
     ↪  traj_radius);

    // Gains
    double Kp = 58, Kd = 10; //HomeWork 3.b
    .
    .
    .
    while()
    {
    .
    .
    .
    robot.getInverseKinematics(des_pose, des_cart_vel,
     ↪  des_cart_acc,qd,dqd,ddqd);
    .
    .
    .
    }
}
//---------------------------------------------------------------------------
```

Some videos of the four trajectory have been saved and stored inside the github repository.

3.1 Rosbag

In order to have a better visualization of the resulting torques, the data has been passed to MatLab and plotted there, for doing this the rosbag-record functionality has been used:

```
    #HomeWork 3.c -------------- rosbag.sh -----------------------

#!/bin/bash

# Verifica se è stato passato un parametro e se è un numero tra 1 e 4
if [ "$#" -ne 1 ] || ! [[ "$1" =~ ^[1-4]$ ]]; then
    echo "Errore: Devi passare un numero tra 1 e 4 come parametro."
    exit 1
fi

# Assegna il parametro a una variabile
NUMERO="$1"

# Specifica il nome del file finale
FINAL_BAGFILE=""

# Decidi il nome finale del file in base al numero
case "$NUMERO" in
    "1")
        FINAL_BAGFILE="/home/lucabor/Scrivania/RoboticsLab/src/
        kdl_robot/rosbag/LinTrap.bag"
        ;;
    "2")
        FINAL_BAGFILE="/home/lucabor/Scrivania/RoboticsLab/src/
        kdl_robot/rosbag/LinCub.bag"
        ;;
    "3")
        FINAL_BAGFILE="/home/lucabor/Scrivania/RoboticsLab/src/
        kdl_robot/rosbag/CircTrap.bag"
        ;;
    "4")
        FINAL_BAGFILE="/home/lucabor/Scrivania/RoboticsLab/src/
        kdl_robot/rosbag/CircCub.bag"
        ;;
esac

# Avvia la registrazione
rosbag record -O "$FINAL_BAGFILE"
 ↪  /iiwa/iiwa_joint_1_effort_controller/command
 ↪  /iiwa/iiwa_joint_2_effort_controller/command
 ↪  /iiwa/iiwa_joint_3_effort_controller/command
 ↪  /iiwa/iiwa_joint_4_effort_controller/command
 ↪  /iiwa/iiwa_joint_5_effort_controller/command
 ↪  /iiwa/iiwa_joint_6_effort_controller/command
 ↪  /iiwa/iiwa_joint_7_effort_controller/command
#---------------------------------------------------------------------------
```

This bash file must be run in another terminal once launched the gazebo simulation.
Using the following MatLab script, the data has been plotted:

```
    %HomeWork 3.c -------------- PlotBag.m -----------------------

%%Bag Selection

bag =
rosbag('/home/lucabor/Scrivania/RoboticsLab/src/kdl_robot/rosbag/LinTrap.bag');

%bag =
rosbag('/home/lucabor/Scrivania/RoboticsLab/src/kdl_robot/rosbag/LinCub.bag');

%bag =
rosbag('/home/lucabor/Scrivania/RoboticsLab/src/kdl_robot/rosbag/CircTrap.bag');

%bag =
rosbag('/home/lucabor/Scrivania/RoboticsLab/src/kdl_robot/rosbag/CircCub.bag');
```

```

jnt1 =
timeseries(bag.select('Topic','/iiwa/iiwa_joint_1_effort_controller/command'));

jnt2 =
timeseries(bag.select('Topic','/iiwa/iiwa_joint_2_effort_controller/command'));

jnt3 =
timeseries(bag.select('Topic','/iiwa/iiwa_joint_3_effort_controller/command'));

jnt4 =
timeseries(bag.select('Topic','/iiwa/iiwa_joint_4_effort_controller/command'));

jnt5 =
timeseries(bag.select('Topic','/iiwa/iiwa_joint_5_effort_controller/command'));

jnt6 =
timeseries(bag.select('Topic','/iiwa/iiwa_joint_6_effort_controller/command'));

jnt7 =
timeseries(bag.select('Topic','/iiwa/iiwa_joint_7_effort_controller/command'));

figure;
hold on;
plot(jnt1.Data)
plot(jnt2.Data)
plot(jnt3.Data)
plot(jnt4.Data)
plot(jnt5.Data)
plot(jnt6.Data)
plot(jnt7.Data)
grid on;
legend show;
xlabel('Time [s]');
ylabel('Torque [Nm]');

%-----------------------------------------------------------------------
```

3.2 Results

It's possible to launch the simulation:

```
#---------------- Terminal 1 ------------------------
$ roslaunch iiwa_gazebo iiwa_gazebo_effort.launch
```

```
#---------------- Terminal 2 ------------------------
$ rosrun kdl_ros_control kdl_robot_test /home/lucabor/Scrivania/
    RoboticsLab/src/iiwa_stack/iiwa_description/urdf/iiwa14.urdf
```

```
#---------------- Terminal 3 ------------------------
$ roscd kdl_ros_control/bash
$ ./rosbag.sh
```

In Fig.1-4 some results are reported for four different trajectories:

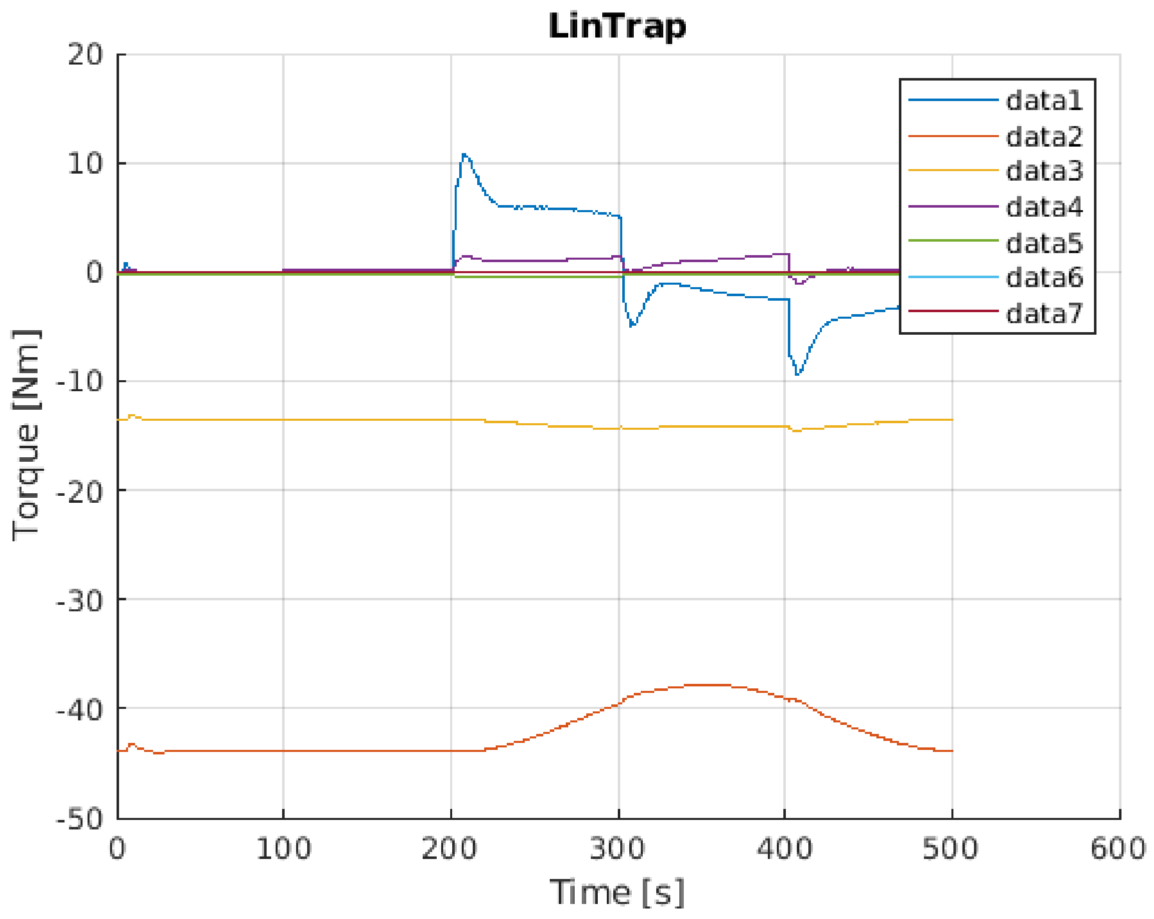


Figure 1: Linear trajectory with trapezoidal velocity profile (JSC).

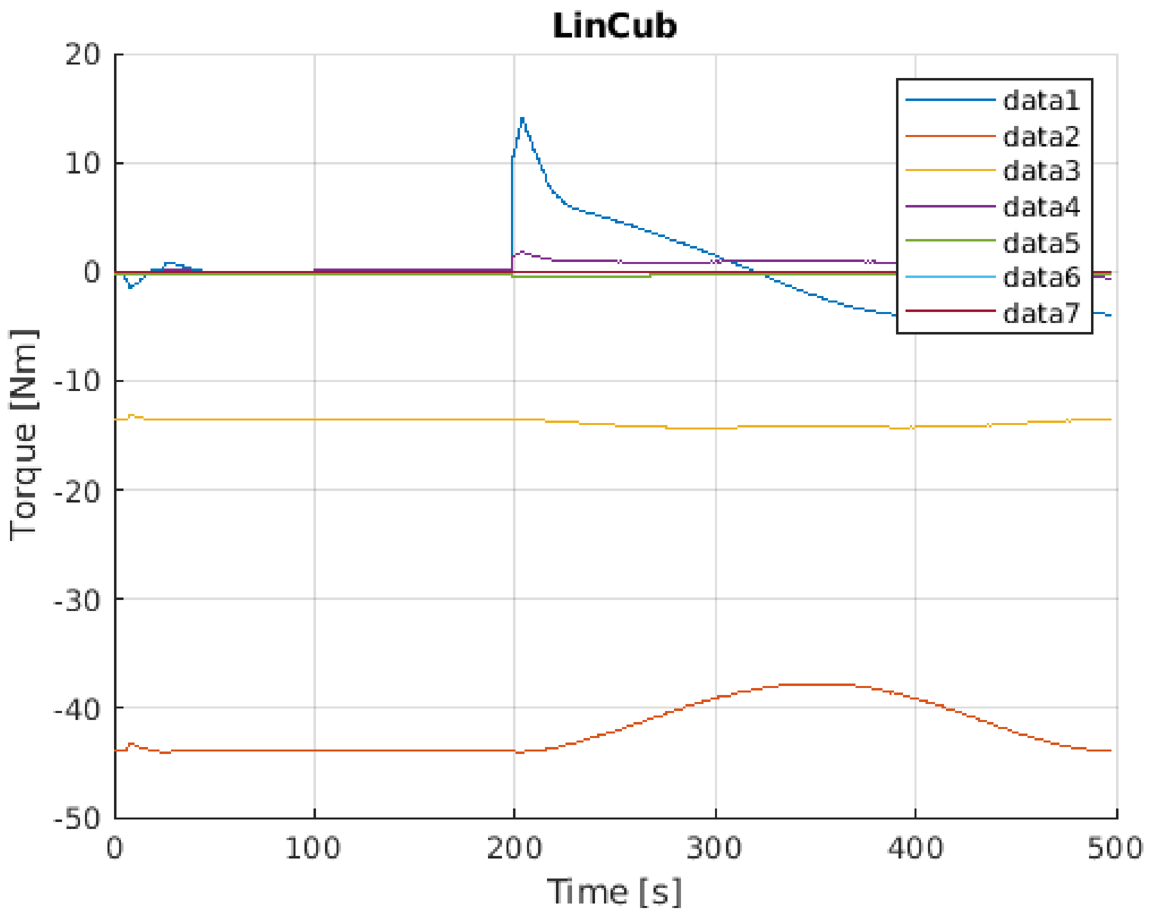


Figure 2: Linear trajectory with cubic polynomial profile (JSC).

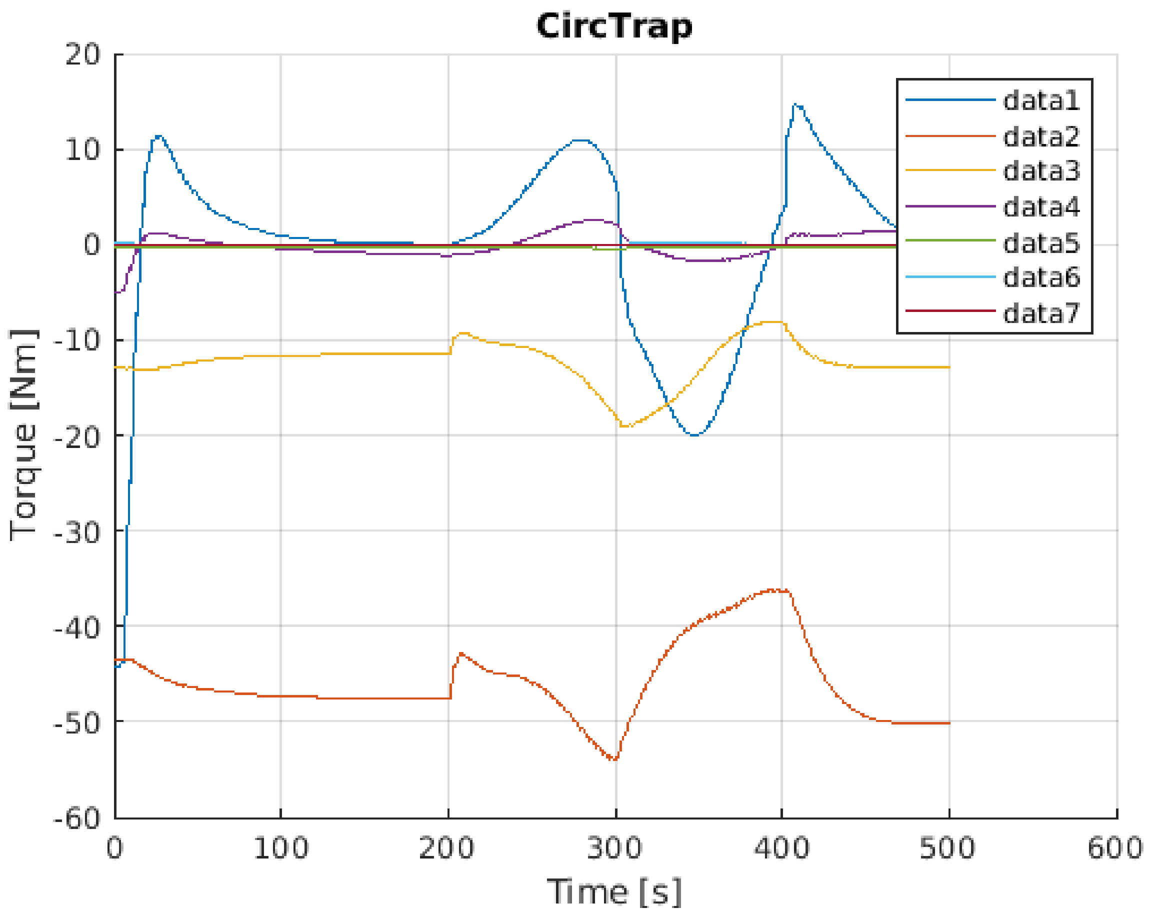


Figure 3: Circular trajectory with trapezoidal velocity profile (JSC).

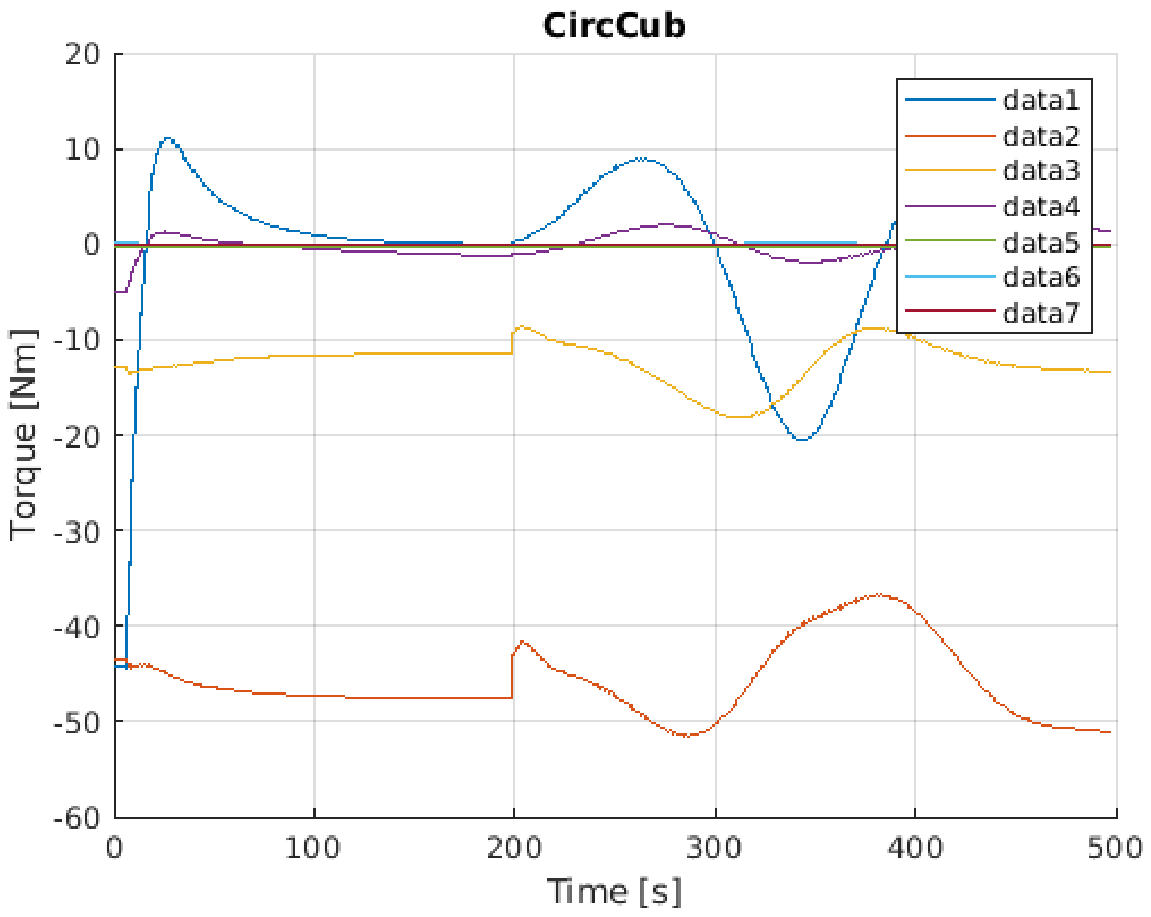


Figure 4: Circular trajectory with cubic polynomial profile (JSC).

4 Inverse dynamics operational space controller

4.1 Development

The development of the controller has followed the path suggested by the professor:

```
#HomeWork 4.a-b -------------- kdl_control.cpp -----------------------

Eigen::VectorXd KDLController::idCntr(KDL::Frame &_desPos,
                                      KDL::Twist &_desVel,
                                      KDL::Twist &_desAcc,
                                      double _Kpp, double _Kpo,
                                              double _Kdp, double _Kdo, double
                                               ↪  &e_norm)
{
    // calculate gain matrices
    Eigen::Matrix<double,6,6> Kp, Kd;
    Kp = Eigen::MatrixXd::Zero(6,6);
    Kd = Eigen::MatrixXd::Zero(6,6);
    Kp.block(0,0,3,3) = _Kpp*Eigen::Matrix3d::Identity();
    Kp.block(3,3,3,3) = _Kpo*Eigen::Matrix3d::Identity();
    Kd.block(0,0,3,3) = _Kdp*Eigen::Matrix3d::Identity();
    Kd.block(3,3,3,3) = _Kdo*Eigen::Matrix3d::Identity();

    // read current state
    Eigen::Matrix<double,6,7> J = toEigen(robot_->getEEJacobian());
    Eigen::Matrix<double,7,7> I = Eigen::Matrix<double,7,7>::Identity();
    Eigen::Matrix<double,7,7> M = robot_->getJsim();
    Eigen::Matrix<double,7,6> Jpinv = weightedPseudoInverse(M,J);

    // position
    Eigen::Vector3d p_d(_desPos.p.data);
    Eigen::Vector3d p_e(robot_->getEEFrame().p.data);
    Eigen::Matrix<double,3,3,Eigen::RowMajor> R_d(_desPos.M.data);
    Eigen::Matrix<double,3,3,Eigen::RowMajor>
     ↪  R_e(robot_->getEEFrame().M.data);
    R_d = matrixOrthonormalization(R_d);
    R_e = matrixOrthonormalization(R_e);

    // velocity
    Eigen::Vector3d dot_p_d(_desVel.vel.data);
    Eigen::Vector3d dot_p_e(robot_->getEEVelocity().vel.data);
    Eigen::Vector3d omega_d(_desVel.rot.data);
    Eigen::Vector3d omega_e(robot_->getEEVelocity().rot.data);

    // acceleration
    Eigen::Matrix<double,6,1> dot_dot_x_d;
    Eigen::Matrix<double,3,1> dot_dot_p_d(_desAcc.vel.data);
    Eigen::Matrix<double,3,1> dot_dot_r_d(_desAcc.rot.data);

    // compute linear errors
    Eigen::Matrix<double,3,1> e_p = computeLinearError(p_d,p_e);
    Eigen::Matrix<double,3,1> dot_e_p = computeLinearError(dot_p_d,dot_p_e);

    // compute orientation errors
    Eigen::Matrix<double,3,1> e_o = computeOrientationError(R_d,R_e);
    Eigen::Matrix<double,3,1> dot_e_o =
    ↪  computeOrientationVelocityError(omega_d, omega_e, R_d, R_e);

    Eigen::Matrix<double,6,1> x_tilde;
    Eigen::Matrix<double,6,1> dot_x_tilde;
    x_tilde << e_p, e_o;
    //e_norm = x_tilde.norm();
    dot_x_tilde << dot_e_p, dot_e_o;    //dot_e_o;
    dot_dot_x_d << dot_dot_p_d, dot_dot_r_d;

    // null space control
    double cost;
    Eigen::VectorXd grad =
    ↪  gradientJointLimits(robot_->getJntValues(),robot_->getJntLimits(),cost);

    // inverse dynamics
    Eigen::Matrix<double,6,1> y;
    y << dot_dot_x_d - robot_->getEEJacDotqDot() + Kd*dot_x_tilde +
    ↪  Kp*x_tilde;

    return M * (Jpinv*y + (I-Jpinv*J)*(/*- 10*grad */-
    ↪  1*robot_->getJntVelocities()))
            + robot_->getGravity() + robot_->getCoriolis();
}

//--------------------------------------------------------------------
```

4.2 Testing and results

Then it has been tested:

```
#HomeWork 4.c ----------- kdl_robot_test.cpp ------------------------
double Kp = 100;
            double Ko = 10;
            // Cartesian space inverse dynamics control
            tau = controller_.idCntr(des_pose, des_cart_vel, des_cart_acc,
            ↪  Kp, Ko, 2*sqrt(Kp), 2*sqrt(Ko), error);
//-----------------------------------------------------------------------
```

In Fig.5-8 are reported the commanded torques:

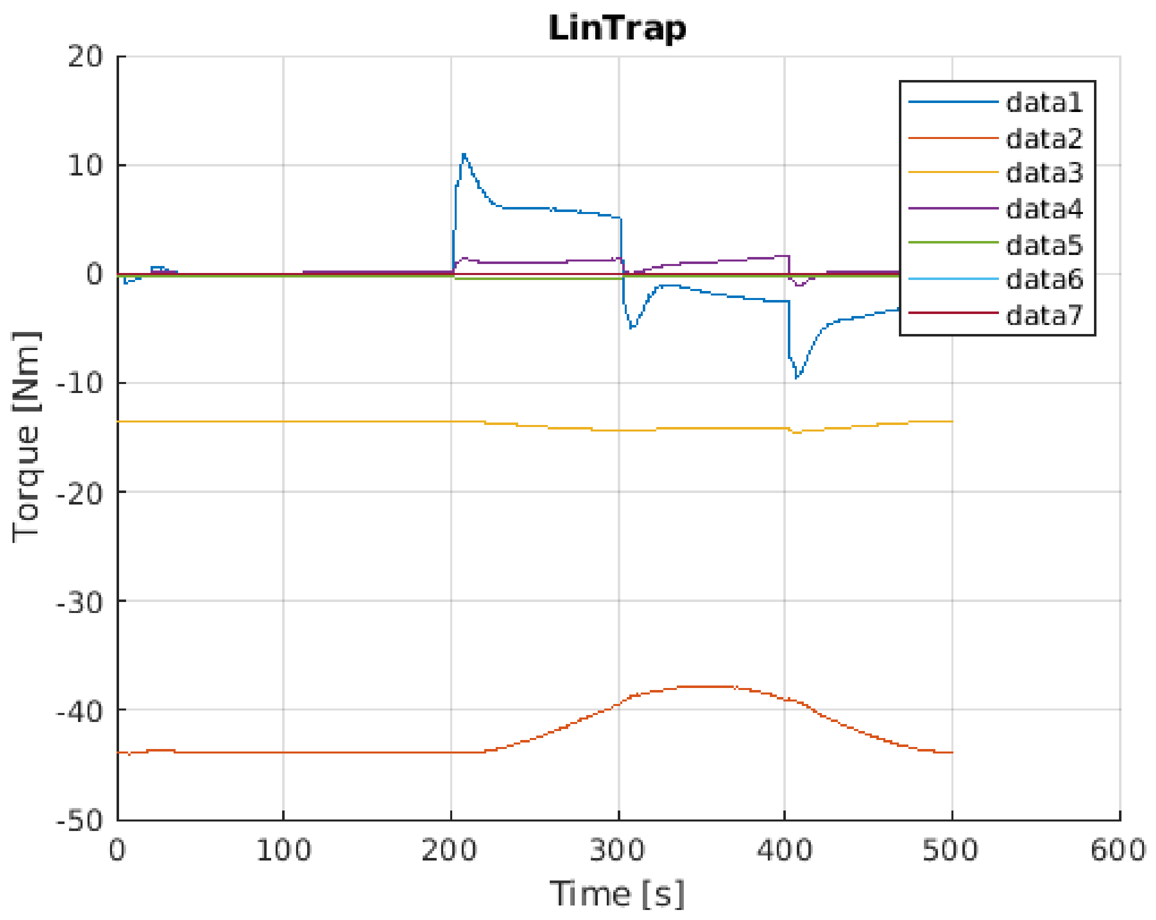


Figure 5: Linear trajectory with trapezoidal velocity profile (OSC).

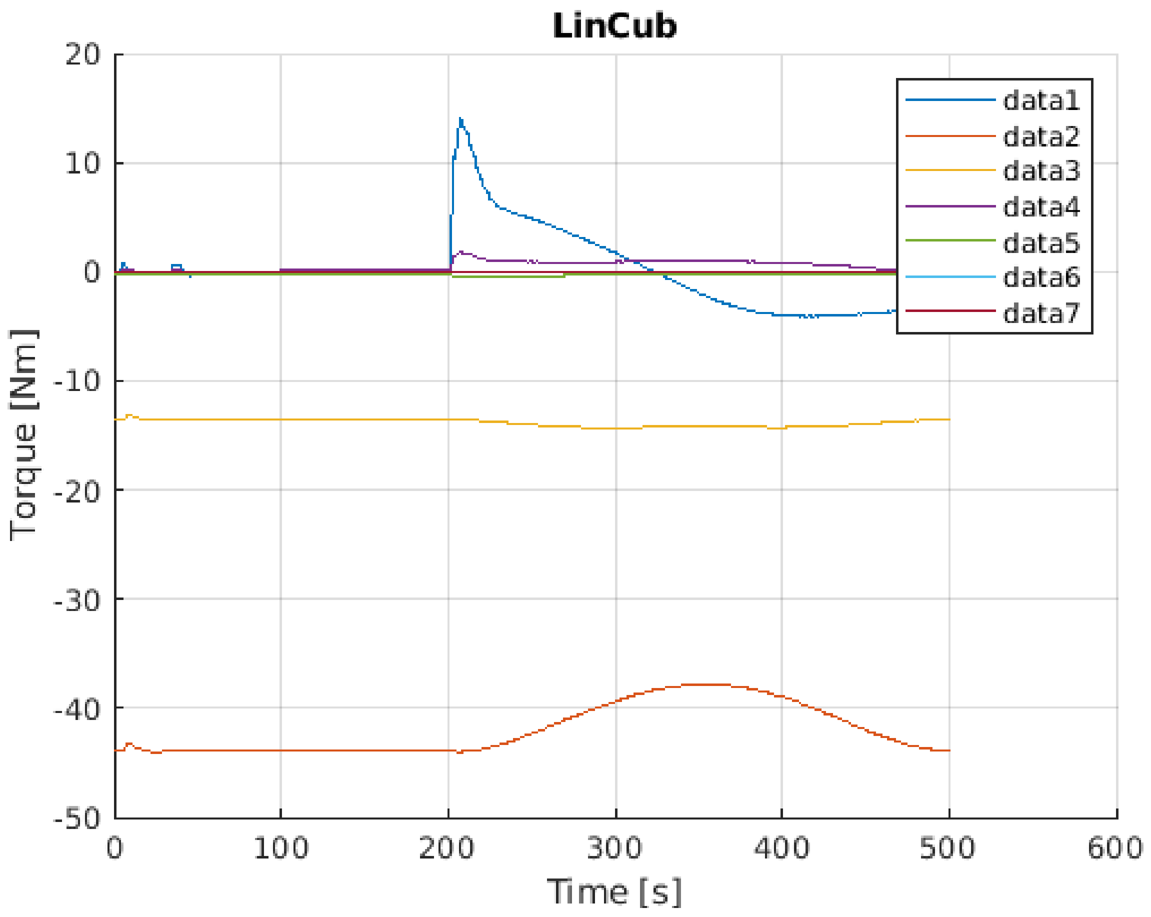


Figure 6: Linear trajectory with cubic polynomial profile (OSC).

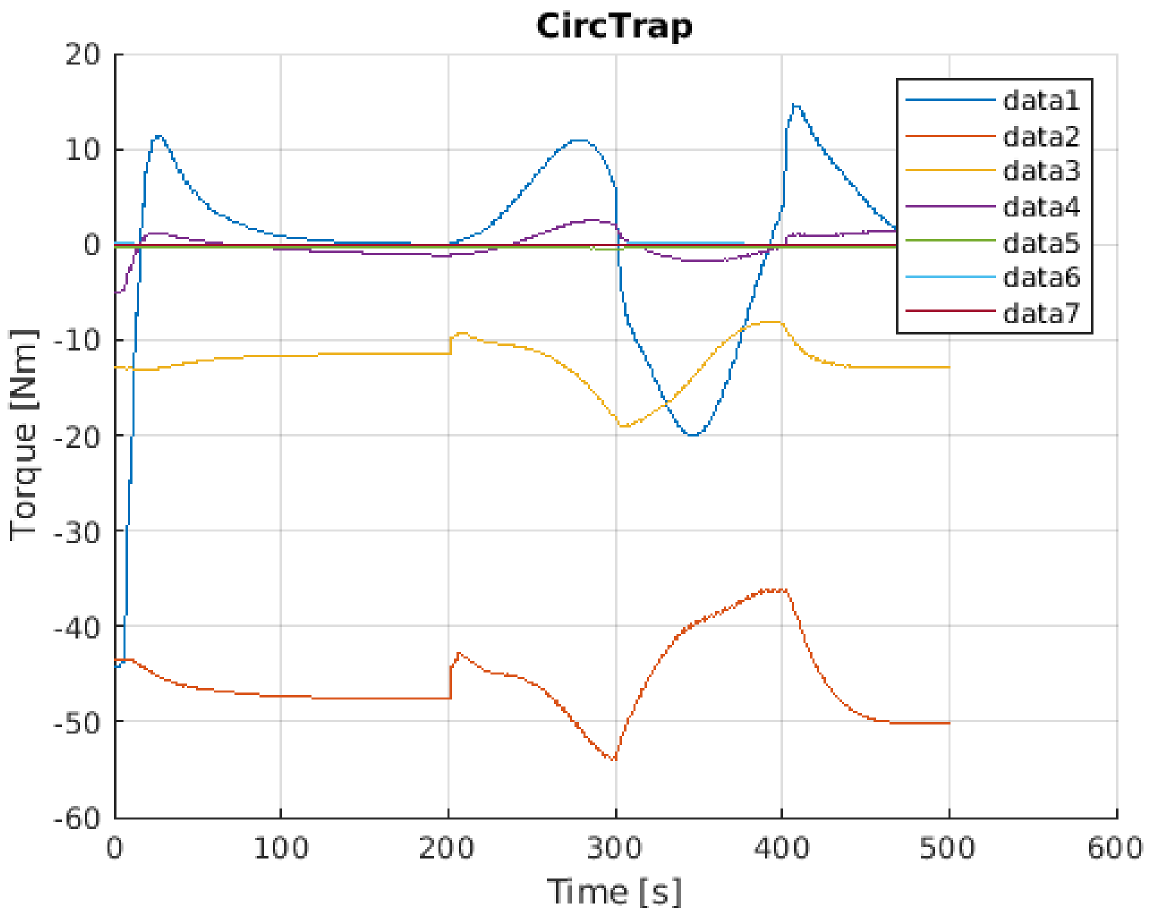


Figure 7: Circular trajectory with trapezoidal velocity profile (OSC).

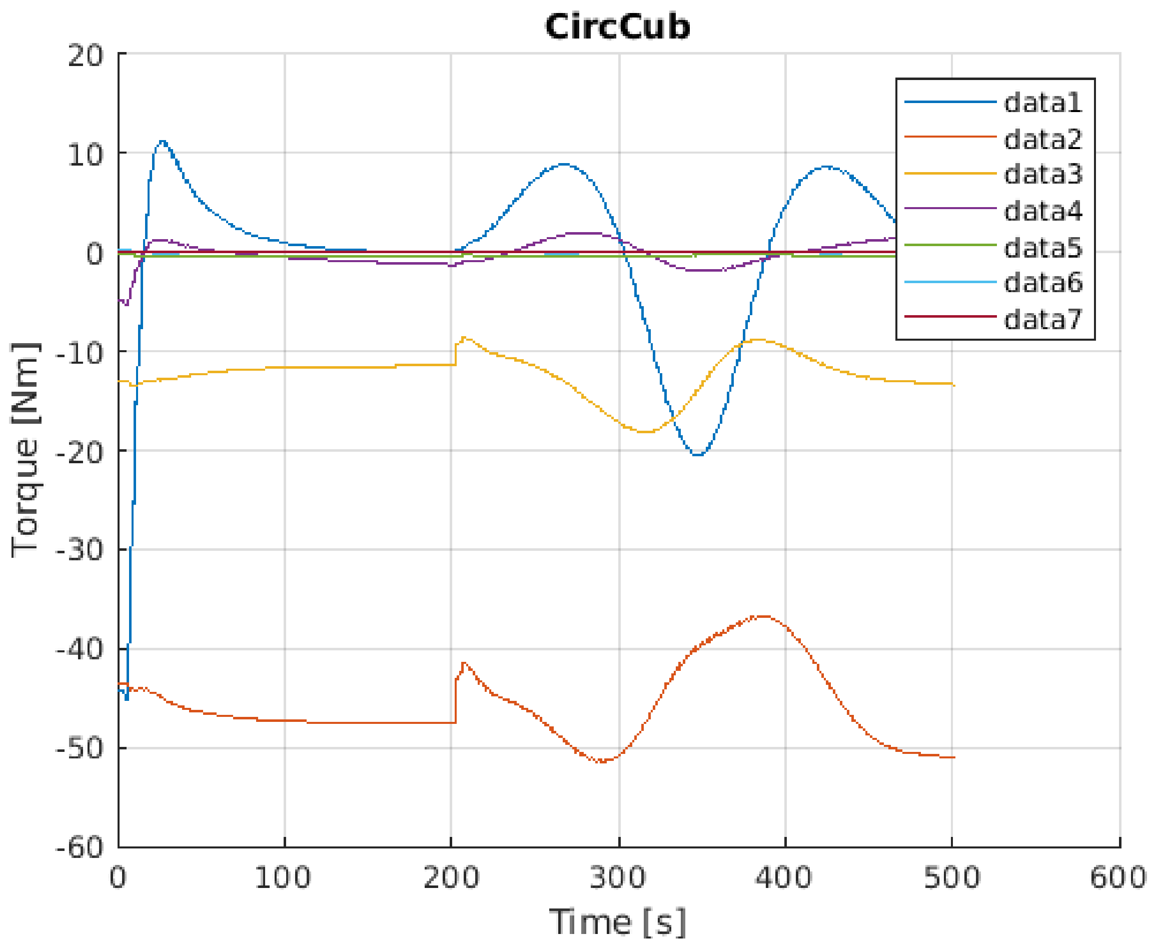


Figure 8: Circular trajectory with cubic polynomial profile (OSC).

Compared with the joint space controller, the operational space controller operates quite similar but seems to be smoother avoiding too many overshoots.

Homework 3

Implement a Vision-Based Task

1 *opencv_ros* package

It's possible to fetch all the implemented code to the following git repository:

```
https://github.com/RobLab23/Homework_3
```

1.1 New world with circular object creation

In order to create a circular object, a procedure similar to the creation of aruco marker has been followed:

```
$ roscd iiwa_gazebo/models
```

within this directory, the *aruco_marker* folder is contained. This folder has been copied and renamed as *circular_object*. Of this new folder, only two files were maintained that were suitably modified to accomplish the task:

```
<!-- -------------------------- model.config ---------------------------- -->
<?xml version="1.0"?>
<model>
  <name>circular_object</name>
  <version>1.0</version>
  <sdf version="1.6">model.sdf</sdf>

  <author>
    <name>Luca Borriello</name>
    <email>lucaborriello97@gmail.com</email>
```

```xml
  </author>

  <description>
    <p>Circular Object</p>
  </description>
</model>
```

```xml
<!-- -------------------------- model.sdf ---------------------------- -->
<?xml version="1.0"?>
<!-- example: https://classic.gazebosim.org/tutorials?tut=color_model&cat=
↪  -->

<sdf version="1.6">
  <model name="circular_object">
    <link name="link">
      <inertial>
        <pose>0 0 0 0 0 0</pose>
        <mass>0.001</mass>
        <inertia>
          <ixx>3.7499999999999997e-06</ixx>
          <ixy>0.0</ixy>
          <ixz>0.0</ixz>
          <iyy>1.8750008333333333e-06</iyy>
          <iyz>0.0</iyz>
          <izz>1.8750008333333333e-06</izz>
        </inertia>
      </inertial>
      <visual name="front_visual">
        <pose>1 -0.5 0.6 0 1.57 0</pose>
        <geometry>
          <cylinder>
           <radius>0.15</radius>
           <length>0.05</length>
           </cylinder>
        </geometry>
        <material>
          <ambient>0 0 1 1</ambient>
        </material>
      </visual>
      <collision name="collision">
        <pose>0 0 0 0 0 0</pose>
        <geometry>
          <cylinder>
           <radius>0.15</radius>
           <length>0.05</length>
          </cylinder>
        </geometry>
      </collision>
    </link>
  </model>
</sdf>
```

Then the circular object has been spawned inside the gazebo scene and saved as:
iiwa_circular_object.world

1.2 Scene preparation

In order to prepare the simulation:

```xml
<!-- -------------- iiwa_gazebo_circular_object.launch -------------- -->
<?xml version="1.0"?>
<launch>

    <arg name="hardware_interface" default="PositionJointInterface" />
    <arg name="robot_name" default="iiwa" />
    <arg name="model" default="iiwa14"/>
    <arg name="trajectory" default="true"/>

    <env name="GAZEBO_MODEL_PATH" value="$(find iiwa_gazebo)/models:$(optenv
    ↪  GAZEBO_MODEL_PATH)" />

    <!-- Loads the Gazebo world. -->
    <include file="$(find
    ↪  iiwa_gazebo)/launch/iiwa_world_circular_object.launch">
        <arg name="hardware_interface" value="$(arg hardware_interface)" />
        <arg name="robot_name" value="$(arg robot_name)" />
        <arg name="model" value="$(arg model)" />
    </include>

    <!-- Spawn controllers - it uses a JointTrajectoryController -->
    <group  ns="$(arg robot_name)" if="$(arg trajectory)">

        <include file="$(find iiwa_control)/launch/iiwa_control.launch">
            <arg name="hardware_interface" value="$(arg hardware_interface)"
             ↪  />
            <arg name="controllers" value="joint_state_controller $(arg
             ↪  hardware_interface)_trajectory_controller" />
            <arg name="robot_name" value="$(arg robot_name)" />
            <arg name="model" value="$(arg model)" />
        </include>

    </group>

    <!-- Spawn controllers - it uses an Effort Controller for each joint -->
    <group ns="$(arg robot_name)" unless="$(arg trajectory)">

        <include file="$(find iiwa_control)/launch/iiwa_control.launch">
            <arg name="hardware_interface" value="$(arg hardware_interface)"
             ↪  />
            <arg name="controllers" value="joint_state_controller
                 $(arg hardware_interface)_J1_controller
                 $(arg hardware_interface)_J2_controller
                 $(arg hardware_interface)_J3_controller
                 $(arg hardware_interface)_J4_controller
                 $(arg hardware_interface)_J5_controller
                 $(arg hardware_interface)_J6_controller
                 $(arg hardware_interface)_J7_controller"/>
            <arg name="robot_name" value="$(arg robot_name)" />
            <arg name="model" value="$(arg model)" />
        </include>

    </group>


</launch>
```

```xml
<!-- -------------- iiwa_world_circular_object.launch -------------- -->
<?xml version="1.0"?>
<launch>

    <!-- Loads thee iiwa.world environment in Gazebo. -->

    <!-- These are the arguments you can pass this launch file, for example
     ↪  paused:=true -->
    <arg name="paused" default="true"/>
    <arg name="use_sim_time" default="true"/>
    <arg name="gui" default="true"/>
    <arg name="headless" default="false"/>
    <arg name="debug" default="false"/>
    <arg name="hardware_interface" default="PositionJointInterface"/>
    <arg name="robot_name" default="iiwa" />
    <arg name="model" default="iiwa7"/>

    <!-- We resume the logic in empty_world.launch, changing only the name of
     ↪  the world to be launched -->
    <include file="$(find gazebo_ros)/launch/empty_world.launch">
        <arg name="world_name" value="$(find
         ↪  iiwa_gazebo)/worlds/iiwa_circular_object.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>

    <!-- Load the URDF with the given hardware interface into the ROS
     ↪  Parameter Server -->
    <include file="$(find iiwa_description)/launch/$(arg
     ↪  model)_upload.launch">
        <arg name="hardware_interface" value="$(arg hardware_interface)"/>
        <arg name="robot_name" value="$(arg robot_name)" />
    </include>

    <!-- Run a python script to 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 iiwa -param robot_description"/>

</launch>
```

In order to better position the object, *rqt_image_view* has been used.

1.3 Detection

In order to detect the object:

```
// -------------------- opencv_ros_node.cpp --------------------

#include <ros/ros.h>
#include <image_transport/image_transport.h>
#include <cv_bridge/cv_bridge.h>
#include <sensor_msgs/image_encodings.h>
#include <opencv2/imgproc/imgproc.hpp>
#include <opencv2/highgui/highgui.hpp>

#include "opencv2/opencv.hpp"

static const std::string OPENCV_WINDOW = "Image window";

//https://learnopencv.com/blob-detection-using-opencv-python-c/
//https://github.com/spmallick/learnopencv/blob/master/BlobDetector/blob.cpp

class ImageConverter
{
  ros::NodeHandle nh_;
  image_transport::ImageTransport it_;
  image_transport::Subscriber image_sub_;
  image_transport::Publisher image_pub_;

public:
  ImageConverter()
    : it_(nh_)
  {
    // Subscribe to input video feed and publish output video feed
    image_sub_ = it_.subscribe("//iiwa/camera1/image_raw", 1,
      &ImageConverter::imageCb, this);
    image_pub_ = it_.advertise("/image_converter/output_video", 1);

    cv::namedWindow(OPENCV_WINDOW);
  }

  ~ImageConverter()
  {
    cv::destroyWindow(OPENCV_WINDOW);
  }

  void imageCb(const sensor_msgs::ImageConstPtr& msg)
  {
    cv_bridge::CvImagePtr cv_ptr;
    try
    {
      cv_ptr = cv_bridge::toCvCopy(msg, sensor_msgs::image_encodings::BGR8);
    }
    catch (cv_bridge::Exception& e)
    {
      ROS_ERROR("cv_bridge exception: %s", e.what());
      return;
    }

    //HomeWork 1.c ----------------------------------------------------
    //Read image and Convert to grayscale
    cv::Mat im;
    cv::cvtColor(cv_ptr->image, im, cv::COLOR_BGR2GRAY);

    // Setup SimpleBlobDetector parameters.
          cv::SimpleBlobDetector::Params params;

          // Change thresholds
          params.minThreshold = 1;
    params.maxThreshold = 250;
    params.thresholdStep = 1;


```

```
        // Filter by Area.
        params.filterByArea = true;
        params.minArea = 100;
    params.maxArea = 10000000;

        // Filter by Circularity
        params.filterByCircularity = true;
        params.minCircularity = 0.8;

        // Filter by Convexity
        params.filterByConvexity = false;
        params.minConvexity = 0.87;

        // Filter by Inertia
        params.filterByInertia = false;
        params.minInertiaRatio = 0.01;


        // Storage for blobs
        std::vector<cv::KeyPoint> keypoints;

        // Set up detector with params
        cv::Ptr<cv::SimpleBlobDetector> detector =
        ↪  cv::SimpleBlobDetector::create(params);

        // Detect blobs
    detector->detect( im, keypoints);

    // Draw detected blobs as red circles.
        // DrawMatchesFlags::DRAW_RICH_KEYPOINTS flag ensures
        // the size of the circle corresponds to the size of blob

        cv::Mat im_with_keypoints;
        cv::drawKeypoints( cv_ptr->image, keypoints, im_with_keypoints,
        ↪  cv::Scalar(0,0,255), cv::DrawMatchesFlags::DRAW_RICH_KEYPOINTS
        ↪  );

    // Update GUI Window
    cv::imshow(OPENCV_WINDOW, im_with_keypoints);
```

```cpp
    cv::waitKey(1);

    // Output modified video stream
    sensor_msgs::ImagePtr output_image_msg =
    ↪  cv_bridge::CvImage(std_msgs::Header(), "bgr8",
    ↪  im_with_keypoints).toImageMsg();
    image_pub_.publish(cv_ptr->toImageMsg());

    //---------------------------------------------------------------
  }
};

int main(int argc, char** argv)
{
  ros::init(argc, argv, "image_converter");
  ImageConverter ic;
  ros::spin();
  return 0;
}
```

In order to start the simulation:

```
# --------------- Terminal 1 ---------------------
$ roslaunch iiwa_gazebo iiwa_gazebo_circular_object.launch
```

```
# --------------- Terminal 2 ---------------------
$ rosrun opencv_ros opencv_ros_node
```

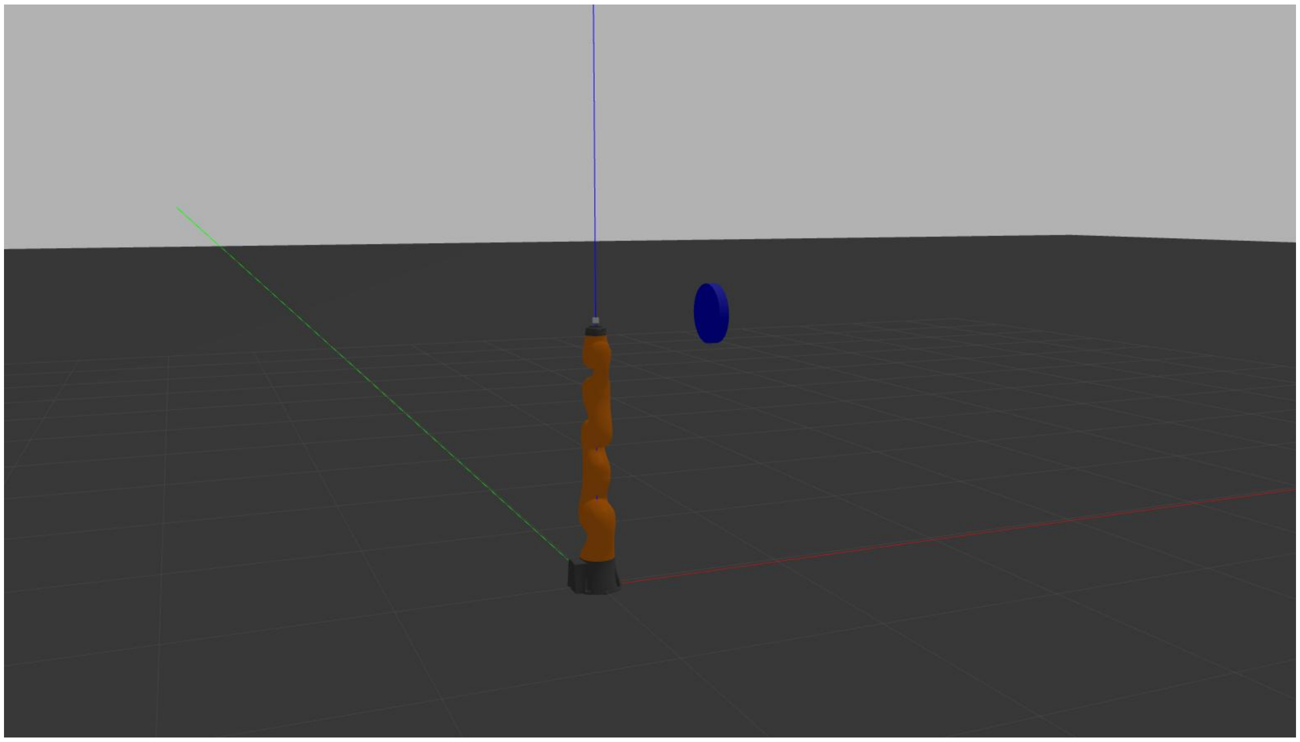


Figure 1: Circular object inside the scene.

In Fig.1-2 the result is shown.

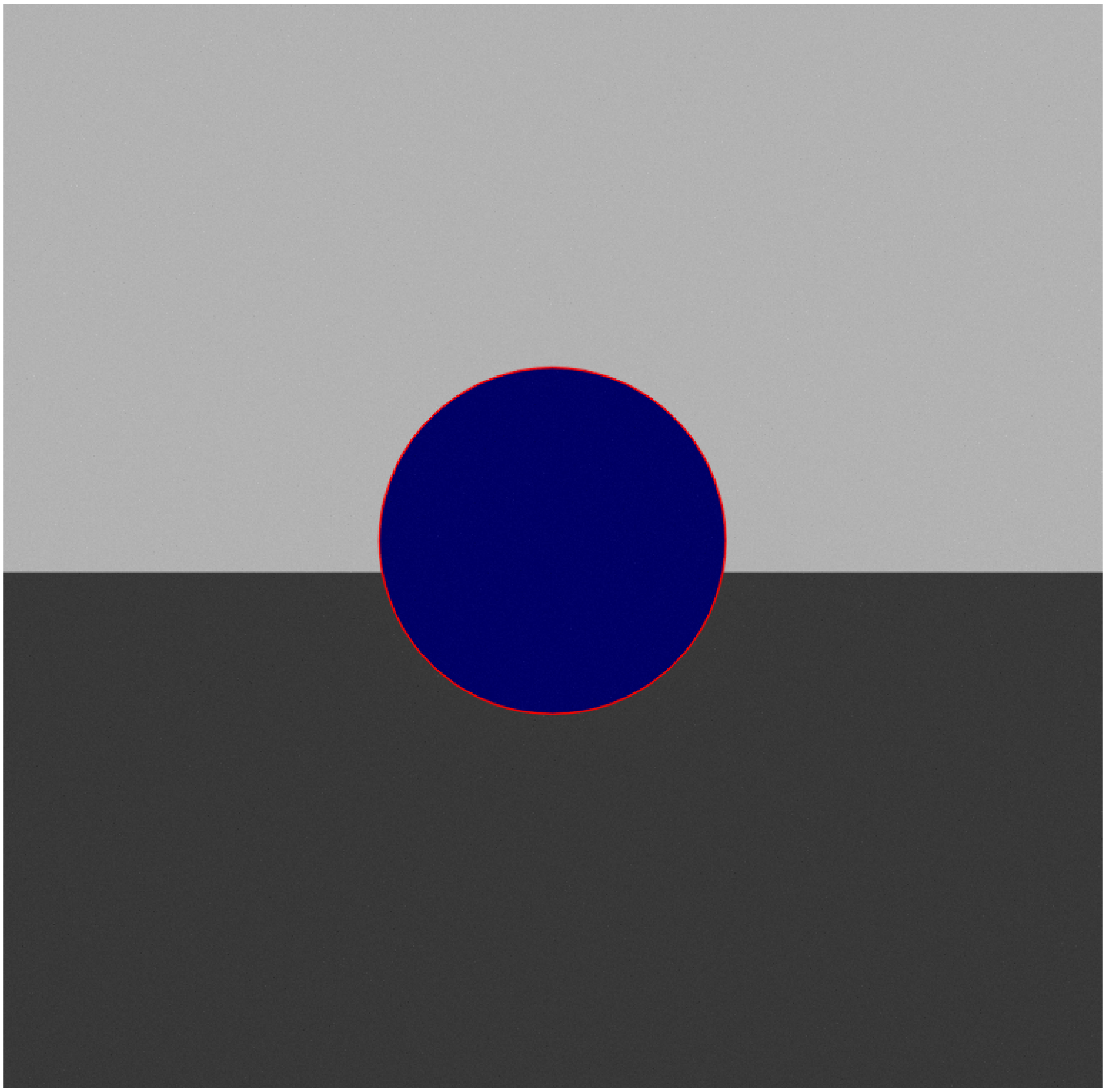


Figure 2: Circular object correctly detected as shown by the red circle.

2 Vision Based Control

2.1 Look-at-point task

In order to implement the look-at-point task:

```cpp
// --------------- kdl_robot_vision_control.cpp ---------------------
//HomeWork 2.a----------------------------------------
//pag 445 FR formula 10.68
KDL::Frame base_T_object = robot.getEEFrame() * cam_T_object; //To
KDL::Rotation R_off = KDL::Rotation::RotX(M_PI/36);
KDL::Vector P_off(0, 0, 0.2);
KDL::Frame T_offset(R_off, P_off);
KDL::Frame T_desired = base_T_object * T_offset; //Tdo

Eigen::Matrix<double, 3, 1> aruco_pos_n = toEigen(cam_T_object.p);
aruco_pos_n.normalize();

Eigen::Matrix<double, 3, 1> e_o_n =
 ↪  computeOrientationError(toEigen(T_desired.M),
 ↪  toEigen(robot.getEEFrame().M));
Eigen::Matrix<double, 3, 1> e_p = computeLinearError(toEigen(T_desired.p),
 ↪  toEigen(robot.getEEFrame().p));
Eigen::Matrix<double, 6, 1> x_tilde;
x_tilde << e_p, e_o_n;

// resolved velocity control low
Eigen::MatrixXd J_pinv =
 ↪  J_cam.data.completeOrthogonalDecomposition().pseudoInverse();
dqd.data = lambda*J_pinv*x_tilde + 10*(Eigen::Matrix<double,7,7>::Identity()
 ↪   - J_pinv*J_cam.data)*(qdi - toEigen(jnt_pos));
//-------------------------------------------------------------
```

In order to start the simulation:

```
# --------------- Terminal 1 ---------------------
$ roslaunch iiwa_gazebo iiwa_gazebo_aruco.launch
```

```
# --------------- Terminal 2 ---------------------
$ roslaunch aruco_ros usb_cam_aruco.launch camera:=/iiwa/camera1/
```

```
# --------------- Terminal 3 ---------------------
$ rosrun kdl_ros_control kdl_robot_vision_control
 ↪  ./src/iiwa_stack/iiwa_description/urdf/iiwa14.urdf
```

In Fig.3 the behaviour of the joint velocity commands is reported:

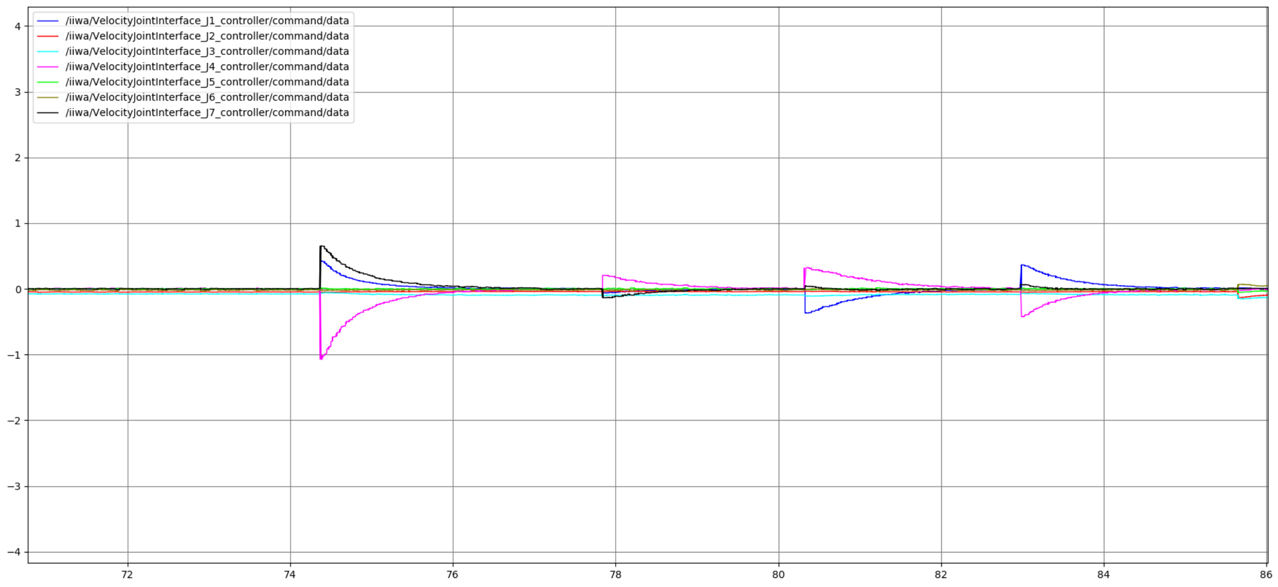


Figure 3: Joint velocity commands.

2.2 Improved Look-at-point algorithm

In order to develop an improved look-at-point algorithm:

```cpp
// --------------- kdl_robot_vision_control.cpp ---------------------
//HomeWork 2.b---------------------------------------
Eigen::Matrix<double, 3, 1> aruco_pos_n = toEigen(cam_T_object.p);
aruco_pos_n.normalize();

Eigen::Matrix<double, 3, 3> R_c = toEigen(robot.getEEFrame().M);

Eigen::Matrix<double, 3, 6> L;
L.block(0, 0, 3, 3) = (-1 / cam_T_object.p.Norm()) * (Eigen::Matrix<double, 3, 3>::Identity() - (aruco_pos_n * aruco_pos_n.transpose())) * R_c.transpose();
L.block(0, 3, 3, 3) = skew(aruco_pos_n) * R_c.transpose();
Eigen::MatrixXd LJ_pinv = (L * J_cam.data).completeOrthogonalDecomposition().pseudoInverse();

Eigen::Vector3d sd;
sd << 0, 0, 1;

Eigen::MatrixXd N = Eigen::Matrix<double, 7, 7>::Identity() - (LJ_pinv * (L * J_cam.data));

double k =2;
dqd.data = k * LJ_pinv * sd + N * (qdi - toEigen(jnt_pos));

s_msg.x = aruco_pos_n(0, 0);
s_msg.y = aruco_pos_n(1, 0);
s_msg.z = aruco_pos_n(2, 0);
//-------------------------------------------------------------
```

In Fig.4 joint velocities and s components are reported:

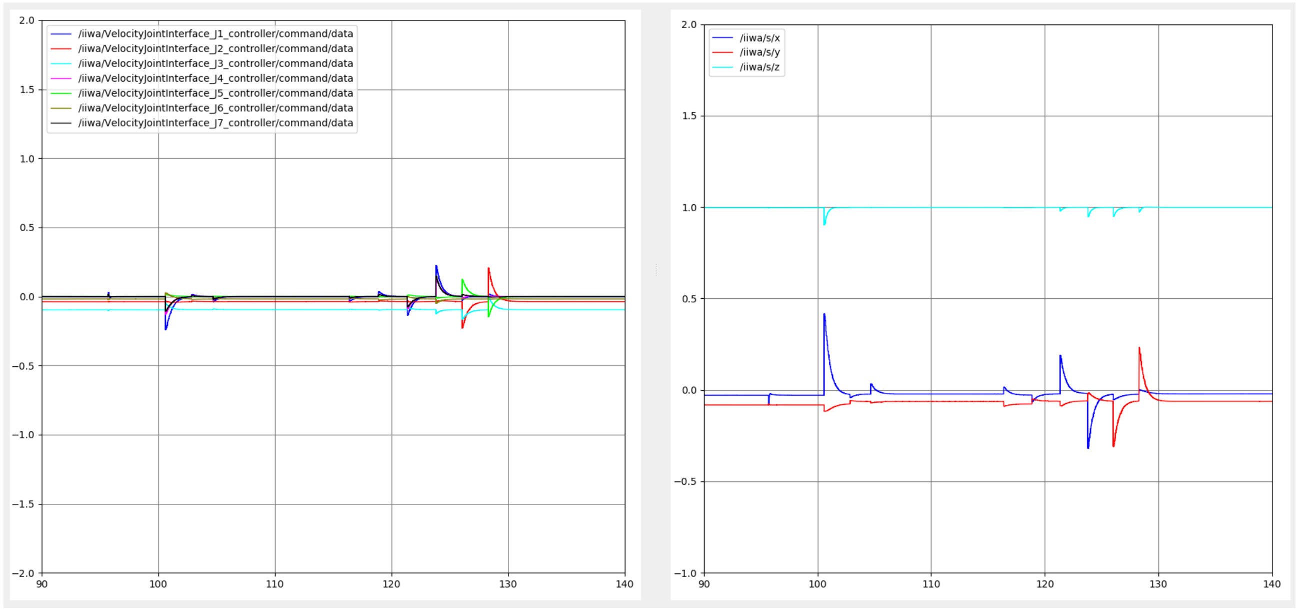


Figure 4: Joint velocity commands on the left, s components on the right.

2.3 Dynamic Vision Based Control

In order to develop a dynamic version of the vision based control, the *kdl_robot_test.cpp* file from HomeWork_2 has been modified and appropriately rearranged for the purpose:

```cpp
// ------------------- kdl_robot_test.cpp ---------------------
//HomeWork 2.c---------------------------------------
 // look at point: compute rotation error from angle/axis
KDL::Frame cam_T_object(KDL::Rotation::Quaternion(aruco_pose[3],
↪  aruco_pose[4], aruco_pose[5], aruco_pose[6]), KDL::Vector(aruco_pose[0],
↪  aruco_pose[1], aruco_pose[2]));
Eigen::Matrix<double, 3, 1> aruco_pos_n = toEigen(cam_T_object.p);
aruco_pos_n.normalize();
Eigen::Vector3d r_o = skew(Eigen::Vector3d(0, 0, 1)) * aruco_pos_n;
double aruco_angle = std::acos(Eigen::Vector3d(0, 0, 1).dot(aruco_pos_n));
KDL::Rotation Re = KDL::Rotation::Rot(KDL::Vector(r_o[0], r_o[1], r_o[2]),
↪  aruco_angle);

des_pose.p = KDL::Vector(p.pos[0], p.pos[1], p.pos[2]);
des_pose.M = robot.getEEFrame().M * Re;

// inverse kinematics
qd.data << jnt_pos[0], jnt_pos[1], jnt_pos[2], jnt_pos[3], jnt_pos[4],
↪  jnt_pos[5], jnt_pos[6];
KDL::Frame flangePose = des_pose * (robot.getFlangeEE().Inverse());
robot.getInverseKinematics(flangePose, des_cart_vel, des_cart_acc, qd, dqd,
↪  ddqd);
//---------------------------------------------------------------
```

An appropriate launch file has been developed for the simlation:

```xml
<!-- ------------- iiwa_gazebo_aruco_effort.launch ----------- -->
<?xml version="1.0"?>
<launch>

    <arg name="hardware_interface" default="EffortJointInterface" />
    <arg name="robot_name" default="iiwa" />
    <arg name="model" default="iiwa14"/>

    <env name="GAZEBO_MODEL_PATH" value="$(find iiwa_gazebo)/models:$(optenv
    ↪  GAZEBO_MODEL_PATH)" />

    <!-- Loads the Gazebo world. -->
    <include file="$(find iiwa_gazebo)/launch/iiwa_world_aruco.launch">
        <arg name="hardware_interface" value="$(arg hardware_interface)" />
        <arg name="robot_name" value="$(arg robot_name)" />
        <arg name="model" value="$(arg model)" />
    </include>

    <!-- Spawn controllers - it uses an Effort Controller for each joint -->
    <group ns="$(arg robot_name)">

        <!-- Spawn controllers - it uses an Effort Controller for each joint
         ↪  -->
        <include file="$(find iiwa_control)/launch/iiwa_control.launch">
            <arg name="hardware_interface" value="$(arg hardware_interface)"
             ↪  />
            <arg name="controllers" value="joint_state_controller
                   iiwa_joint_1_effort_controller
                   iiwa_joint_2_effort_controller
                   iiwa_joint_3_effort_controller
                   iiwa_joint_4_effort_controller
                   iiwa_joint_5_effort_controller
                   iiwa_joint_6_effort_controller
                   iiwa_joint_7_effort_controller"/>
            <arg name="robot_name" value="$(arg robot_name)" />
            <arg name="model" value="$(arg model)" />
        </include>

    </group>

</launch>
```

In order to start the simulation:

```
# --------------- Terminal 1 ---------------------
$ roslaunch iiwa_gazebo iiwa_gazebo_aruco_effort.launch
```

```
# --------------- Terminal 2 ---------------------
$ roslaunch aruco_ros usb_cam_aruco.launch camera:=/iiwa/camera1/
```

```
# --------------- Terminal 3 ---------------------
$ rosrun kdl_ros_control kdl_robot_test
 ↪  ./src/iiwa_stack/iiwa_description/urdf/iiwa14.urdf
```

The simulation has been tested [Fig.5-6] for the same linear trajectory with trapezoidal velocity profile and with two different controllers (JS and OS):

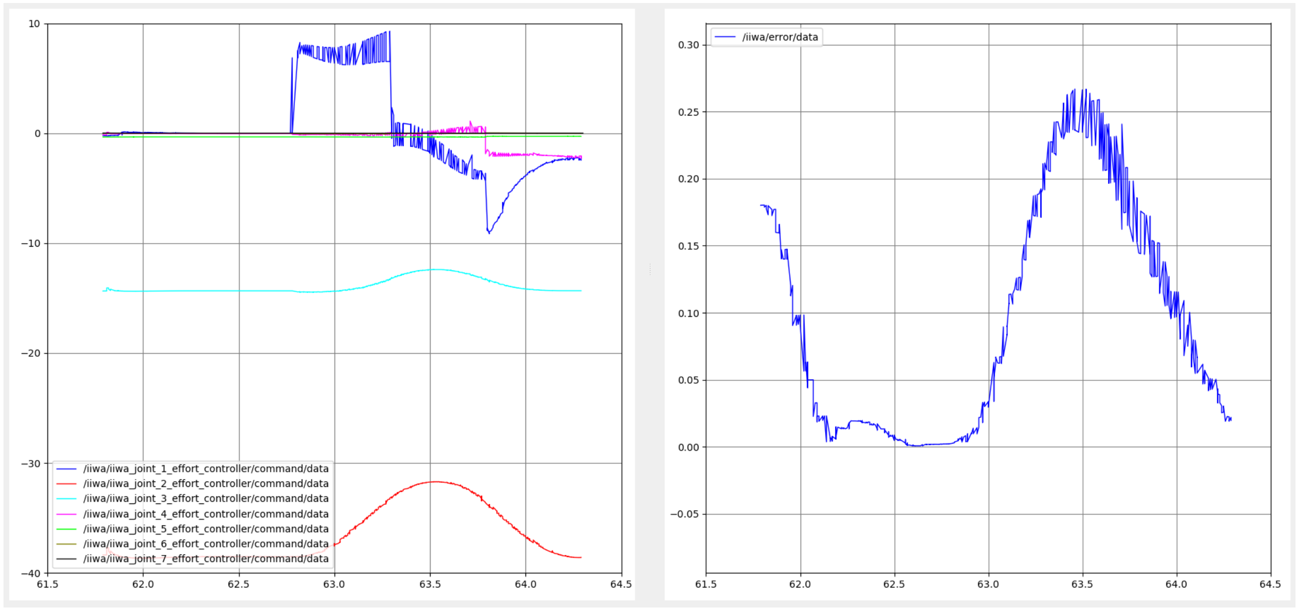


Figure 5: Linear Trajectory with Joint Space controller: joint torques on the left, error norm on the right.

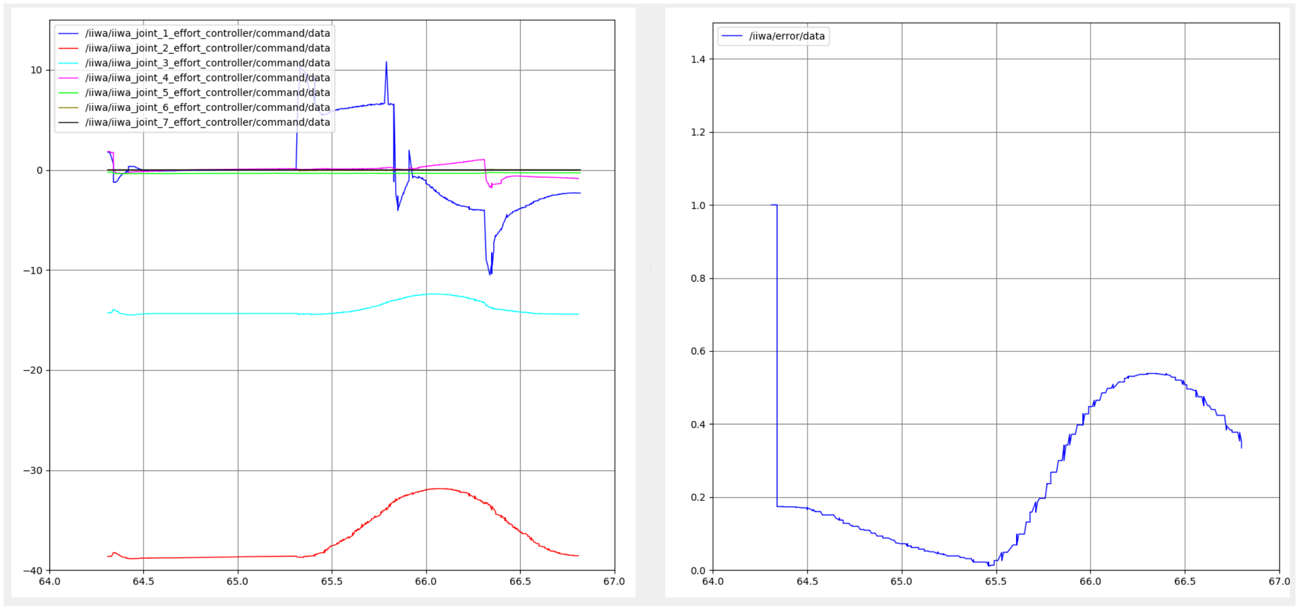


Figure 6: Linear Trajectory with Operational Space controller: joint torques on the left, error norm on the right.

Homework 4

Control a Mobile Robot to Follow a Trajectory

1 World construction

It's possible to fetch all the implemented code to the following git repository:

```
https://github.com/RobLab23/Homework_4
```

1.1 Spawning the robot in a given pose

In order to spawn the robot in a given pose:

```
<!-- rl_fra2mo_description/launch/spawn_fra2mo_gazebo.launch -->

<!-- HomeWork 1.a -->
  <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"/>
  <!-- ______________ -->

  <!-- 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"/> <!--HomeWork
         ↪  1.a -->

```

1.2 Racefield modification

In order to move the obstacle 9 in a given position:

```
<!-- rl_racefields/world/rl_race_fiel.world -->

<include>
      <name>obstacle_09</name>
      <pose> -17 9 0.1 0 0 3.14159</pose> <!-- HomeWork 1.b -->
      <uri>model://obstacle_09</uri>
    </include>

```

1.3 Aruco marker creation

As first, the aruco marker number 115 has been generated by the link-site and saved in .png format. Then the aruco_marker directory of the homework 3 has been copied in rl_racefield/models. Then the texture has been modified with the image of the marker 115, and the model.config and model.sdf files have been properly modified in order to make the marker static.
Then using:

```
$ roslaunch rl_racefield rl_racefield_gazebo.launch
```

the marker has been properly positioned and once read the position, the .world file has been changed.

```
<!-- rl_racefields/world/rl_race_fiel.world -->

<!-- HomeWork 1.c -->
    <!-- AR markers -->
    <include>
      <name>aruco_marker</name>
      <uri>model://aruco_marker</uri>
       <pose>-17 8.25 0.4 0 0 0</pose>
    </include>
    <!-- ____________ -->
```

2 Autonomous navigation task

2.1 Static goals

In order to insert 4 tf static goals:

```
<!-- rl_fra2mo_description/launch/spawn_fra2mo_gazebo.launch -->

<arg name="sendGoals" default="false"/>

<group if="$(arg sendGoals)">
  <param name="goalNumber" value="4"/>
  <node pkg="tf" type="static_transform_publisher" name="goal_1_pub"
  ↪  args="-10   3 0  0     0 0 map goal1 100" /> <!--YPR-->
  <node pkg="tf" type="static_transform_publisher" name="goal_2_pub"
  ↪  args="-15   7 0  0.523 0 0 map goal2 100" />
  <node pkg="tf" type="static_transform_publisher" name="goal_3_pub" args="-6
  ↪  8 0  3.141 0 0 map goal3 100" />
  <node pkg="tf" type="static_transform_publisher" name="goal_4_pub"
  ↪  args="-17.5 3 0  1.309 0 0 map goal4 100" />
</group>

```

2.2 TF listener

In order to implement a tf listener the goal_listener() function has been modified:

```
// fra2mo_2dnav/src/tf_nav.cpp

void TF_NAV::goal_listener() { //HomeWork 2.b -------------------
    ros::Rate r( 1 );
    tf::TransformListener listener;
    tf::StampedTransform transform;

    int goalNumber;
    _nh.getParam("goalNumber", goalNumber);
    static std::vector<bool> logged(goalNumber, false);

    while ( ros::ok() )
    {
        for (int i = 0; i < goalNumber; i++) {
            std::string goal_frame = "goal" + std::to_string(i + 1);

            try
            {
                listener.waitForTransform( "map", goal_frame, ros::Time( 0 ),
                ↪   ros::Duration( 10.0 ) );
                listener.lookupTransform( "map", goal_frame, ros::Time( 0 ),
                ↪   transform );
                if (!logged[i]) {
                    ROS_INFO("Goal_[%d]: \n pos (x:%f, y:%f, z:%f) \n rot
                    ↪   (x:%f, y:%f, z:%f, w:%f)\n", (i+1),
                                                transform.getOrigin().x(),
                                                ↪   transform.getOrigin().y(),
                                                ↪   transform.getOrigin().z(),
                                                transform.getRotation().w(),
                                                ↪   transform.getRotation().x(),
                                                ↪   transform.getRotation().y(),
                                                ↪   transform.getRotation().z());
                    logged[i]=true;
                }
            }
            catch( tf::TransformException &ex )
            {
                ROS_ERROR("%s", ex.what());
                r.sleep();
                continue;
            }

            _goal_pos.at(i) << transform.getOrigin().x(),
            ↪   transform.getOrigin().y(), transform.getOrigin().z();
            _goal_or.at(i) << transform.getRotation().w(),
            ↪   transform.getRotation().x(), transform.getRotation().y(),
            ↪   transform.getRotation().z();

        }
        r.sleep();
    }
```

```
}

```

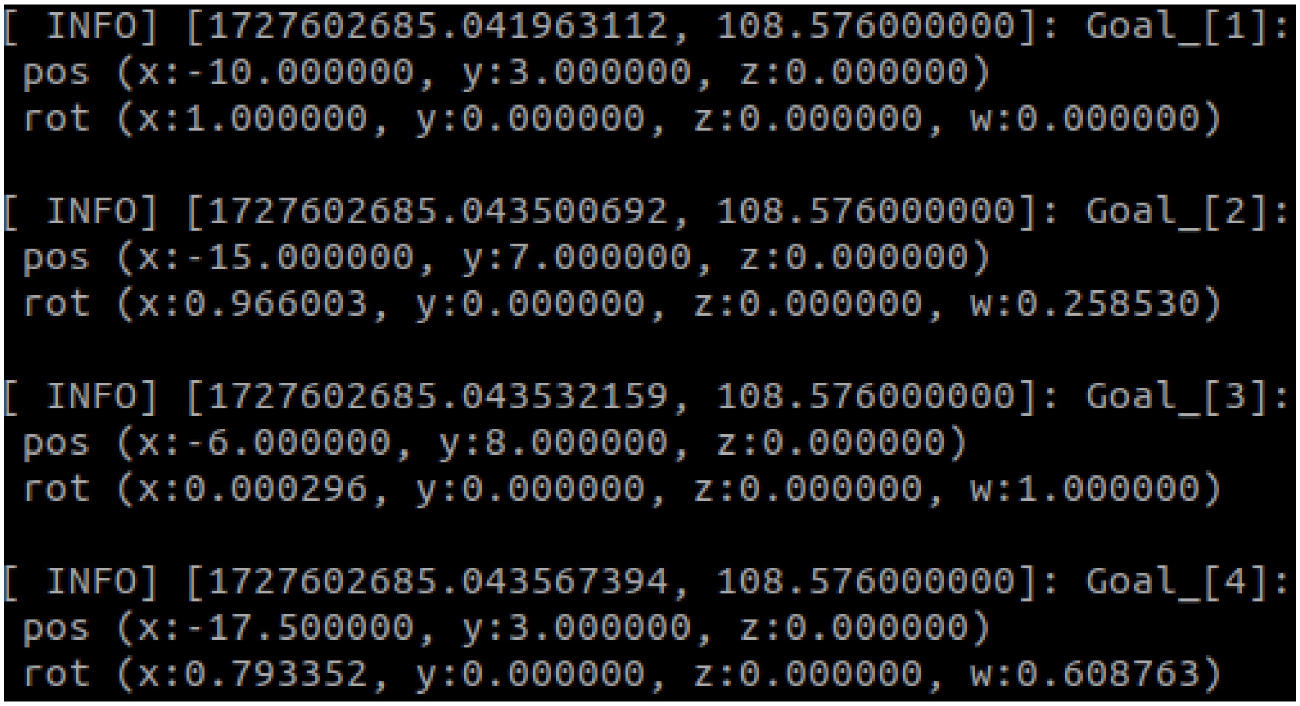
In order to check [Fig.1]:

```
# Terminal 1

$ roslaunch fra2mo_2dnav fra2mo_nav_bringup.launch sendGoals:=true
```

```
# Terminal 2

$ rosrun fra2mo_2dnav tf_nav
```



```
[ INFO] [1727602685.041963112, 108.576000000]: Goal_[1]:
 pos (x:-10.000000, y:3.000000, z:0.000000)
 rot (x:1.000000, y:0.000000, z:0.000000, w:0.000000)

[ INFO] [1727602685.043500692, 108.576000000]: Goal_[2]:
 pos (x:-15.000000, y:7.000000, z:0.000000)
 rot (x:0.966003, y:0.000000, z:0.000000, w:0.258530)

[ INFO] [1727602685.043532159, 108.576000000]: Goal_[3]:
 pos (x:-6.000000, y:8.000000, z:0.000000)
 rot (x:0.000296, y:0.000000, z:0.000000, w:1.000000)

[ INFO] [1727602685.043567394, 108.576000000]: Goal_[4]:
 pos (x:-17.500000, y:3.000000, z:0.000000)
 rot (x:0.793352, y:0.000000, z:0.000000, w:0.608763)
```

Figure 1: Log of the goals.

2.3 Exploring the goals

In order to explore the goals the goal_send() function has been modified:

```
// fra2mo_2dnav/src/tf_nav.cpp

if ( (cmd == 1) && (goalNumber == 4))  { //HomeWork 2.c ------------------
            MoveBaseClient ac("move_base", true);
            while(!ac.waitForServer(ros::Duration(5.0))){
            ROS_INFO("Waiting for the move_base action server to come up");
            }

            Eigen::Vector4d goalOrder;
            goalOrder << 3, 4, 2, 1;

                for (int goal_index = 0; goal_index < goalNumber;
                 ↪  goal_index++) {
                    goal.target_pose.header.frame_id = "map";
                    goal.target_pose.header.stamp = ros::Time::now();

                    goal.target_pose.pose.position.x =
                     ↪  _goal_pos.at(goalOrder(goal_index) - 1)[0];
                    goal.target_pose.pose.position.y =
                     ↪  _goal_pos.at(goalOrder(goal_index) - 1)[1];
                    goal.target_pose.pose.position.z =
                     ↪  _goal_pos.at(goalOrder(goal_index) - 1)[2];

                    goal.target_pose.pose.orientation.w =
                     ↪  _goal_or.at(goalOrder(goal_index) - 1)[0];
                    goal.target_pose.pose.orientation.x =
                     ↪  _goal_or.at(goalOrder(goal_index) - 1)[1];
                    goal.target_pose.pose.orientation.y =
                     ↪  _goal_or.at(goalOrder(goal_index) - 1)[2];
                    goal.target_pose.pose.orientation.z =
                     ↪  _goal_or.at(goalOrder(goal_index) - 1)[3];

                    ROS_INFO("Sending goal %f", goalOrder(goal_index));

                    ros::Duration(0.1).sleep();
                    ac.sendGoal(goal);
                    ac.waitForResult();

                    if (ac.getState() ==
                     ↪  actionlib::SimpleClientGoalState::SUCCEEDED)
```

```
                        ROS_INFO("The mobile robot arrived in the TF goal
                         ↪  number %f",  goalOrder(goal_index));
                    else
                        ROS_INFO("The base failed to move for some reason");
                }
        }
```

Trajectory has been recorded using rosbag:

```
# Terminal 3
$ roscd fra2mo_2dnav/bash
$ ./rosbag.sh
```

then the trajectory has been plotted using the matlab scrip:

```
%PlotPath.m

bag =
 ↪  rosbag('/home/lucabor/Scrivania/RoboticsLab/src/fra2mo_2dnav/rosbag/Path.bag');


    pose = bag.select("Topic", "/fra2mo/pose");
    poseTs = timeseries(pose);
    t = poseTs.Time - poseTs.Time(1);
    x = poseTs.Data(:, 4);
    y = poseTs.Data(:, 5);
    % z = poseTs.Data(:, 6);
    yaw = quat2eul([poseTs.Data(:, 10), poseTs.Data(:, 7:9)]);
    yaw = yaw(:, 1);


    % trajectory
    idxs = 1:round(length(x)/50):length(x);
    ql = 0.5;
    qx = x(idxs);
    qy = y(idxs);
    qu = cos(yaw(idxs));
    qv = sin(yaw(idxs));
    figure(Name="Trajectory")
    plot(x, y)
    hold on
    quiver(qx, qy, qu, qv, ql)
    scatter(-3, 5, 50, 'filled', 'MarkerFaceColor', 'r');
    scatter(-6, 8, 50, 'filled', 'MarkerFaceColor', 'g');
    scatter(-17.5, 3, 50, 'filled', 'MarkerFaceColor', 'b');
    scatter(-15, 7, 50, 'filled', 'MarkerFaceColor', 'c');
    scatter(-10, 3, 50, 'filled', 'MarkerFaceColor', 'k');

    grid on
    title("Trajectory")
    axis equal
    xlabel("x [m]")
    ylabel("y [m]")

```

In Fig.2 the followed trajectory is showed.

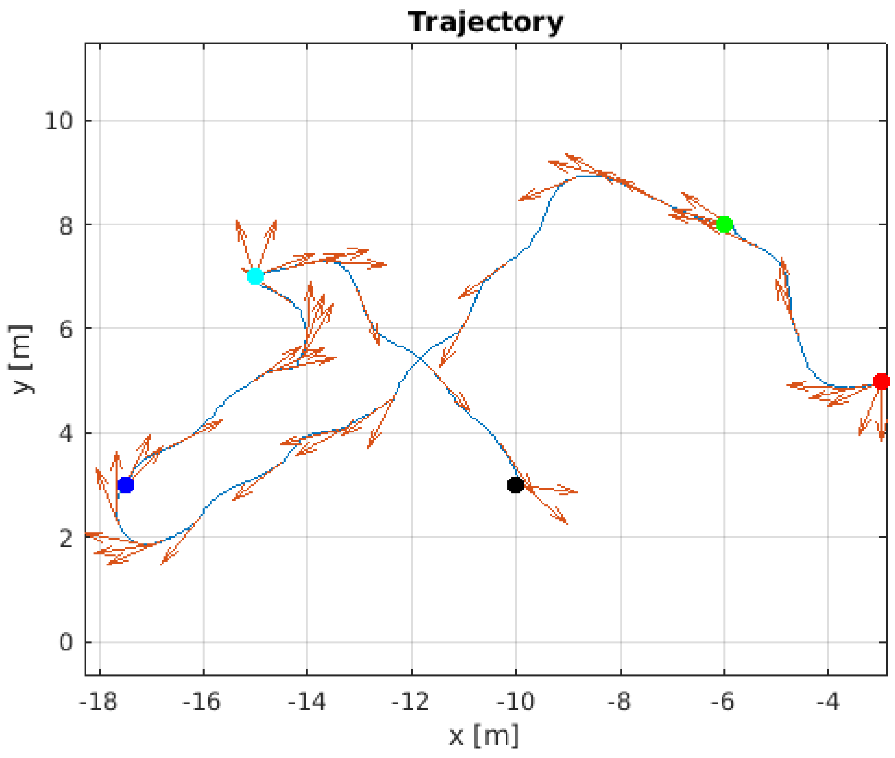


Figure 2: Trajectory: ’rgbck’ order.

3 Map exploration

3.1 Goals setting

In order to explore the map, the following goals have been chosen through the rviz interface publishing on the /clicked_point topic and then set:

```
<!-- rl_fra2mo_description/launch/spawm_fra2mo_gazebo.launch -->

<arg name="enableExploration" default="false"/>
<group if="$(arg enableExploration)">
  <param name="goalNumber" value="6"/>
  <node pkg="tf" type="static_transform_publisher" name="goal_1_pub"
   ↪  args="-6.05 5.2  0  0 0 -1 0 map goal1 100" />
  <node pkg="tf" type="static_transform_publisher" name="goal_2_pub"
   ↪  args="-16.5 1.5  0  0 0 0.91 0.41 map goal2 100" />
  <node pkg="tf" type="static_transform_publisher" name="goal_3_pub"
   ↪  args="-19   6.5  0  0 0 0.72 0.70 map goal3 100" />
  <node pkg="tf" type="static_transform_publisher" name="goal_4_pub"
   ↪  args="-17.5   9.5  0  0 0 -0.1 0.99 map goal4 100" />
  <node pkg="tf" type="static_transform_publisher" name="goal_5_pub"
   ↪  args="-3.37 9.51 0  0 0 0.07 1 map goal5 100" />
  <node pkg="tf" type="static_transform_publisher" name="goal_6_pub"
   ↪  args="-1.5 0.49 0  0 0 -1 0.03 map goal6 100" />
</group>

```

```
// fra2mo_2dnav/src/tf_nav.cpp

else if ( (cmd == 3) && (goalNumber == 6) ) { //HomeWork 3.a
 ↪  --------------------
            MoveBaseClient ac("move_base", true);
            while(!ac.waitForServer(ros::Duration(5.0))){
            ROS_INFO("Waiting for the move_base action server to come up");
            }
            for (int goal_index = 0; goal_index < goalNumber; goal_index++) {
                goal.target_pose.header.frame_id = "map";
                goal.target_pose.header.stamp = ros::Time::now();

                goal.target_pose.pose.position.x =
                 ↪   _goal_pos.at(goal_index)[0];
                goal.target_pose.pose.position.y =
                 ↪   _goal_pos.at(goal_index)[1];
                goal.target_pose.pose.position.z =
                 ↪   _goal_pos.at(goal_index)[2];

                goal.target_pose.pose.orientation.w =
                 ↪   _goal_or.at(goal_index)[0];
                goal.target_pose.pose.orientation.x =
                 ↪   _goal_or.at(goal_index)[1];
                goal.target_pose.pose.orientation.y =
                 ↪   _goal_or.at(goal_index)[2];
                goal.target_pose.pose.orientation.z =
                 ↪   _goal_or.at(goal_index)[3];

                ROS_INFO("Sending goal %i", goal_index);

                ros::Duration(0.1).sleep();
                ac.sendGoal(goal);
                ac.waitForResult();

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

In order to start the exploration:

```
# Terminal 1

$ roslaunch fra2mo_2dnav fra2mo_nav_bringup.launch enableExploration:=true
```

```
# Terminal 2

$ rosrun fra2mo_2dnav tf_nav
```

In Fig.3 a screen of a fully explored map has been reported.

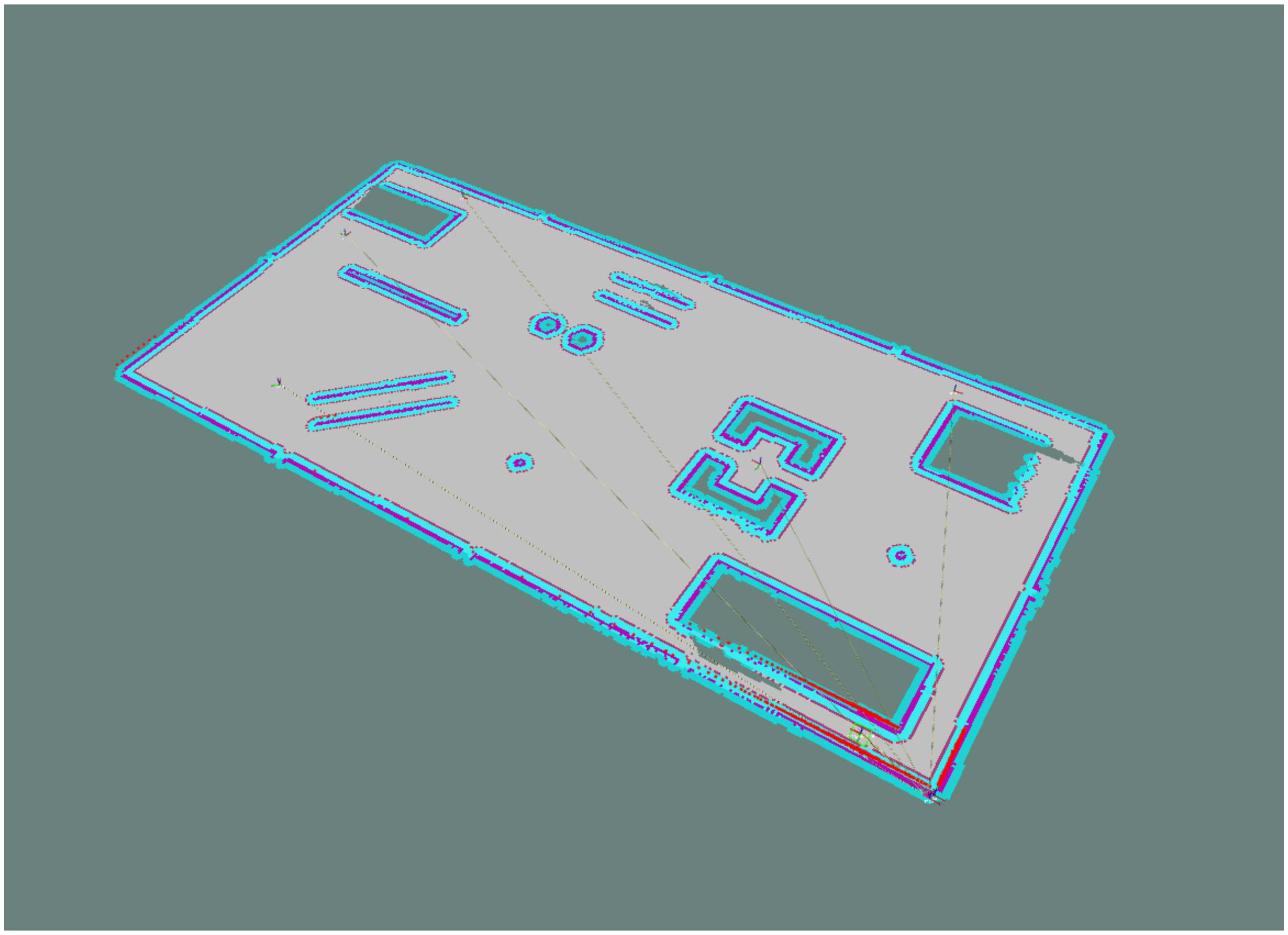
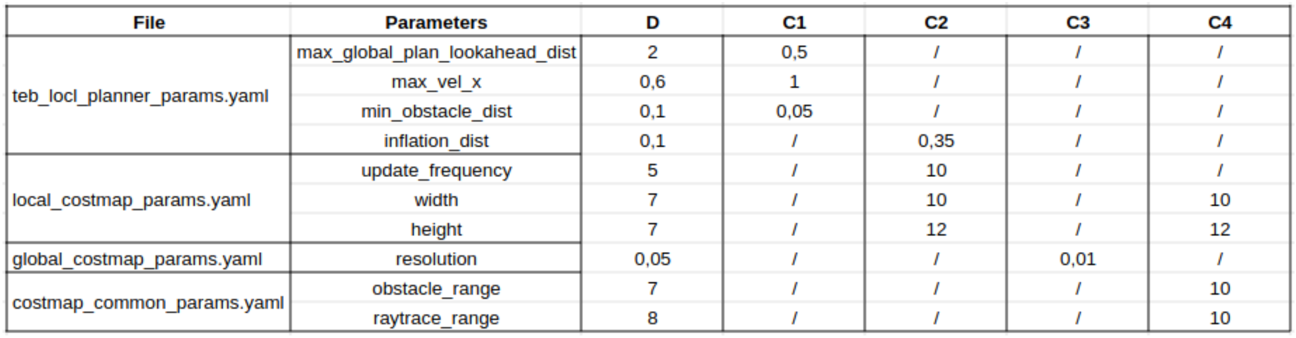


Figure 3: Explored map.

3.2 Navigation parameters tuning

Some tests have been made in order to experiment the navigation with different parameter settings. In particular these parameters (fra2mo_2dnav/config) have been changed and tested in four different configurations [Fig.4]:



File	Parameters	D	C1	C2	C3	C4
teb_locl_planner_params.yaml	max_global_plan_lookahead_dist	2	0,5	/	/	/
	max_vel_x	0,6	1	/	/	/
	min_obstacle_dist	0,1	0,05	/	/	/
	inflation_dist	0,1	/	0,35	/	/
local_costmap_params.yaml	update_frequency	5	/	10	/	/
	width	7	/	10	/	10
	height	7	/	12	/	12
global_costmap_params.yaml	resolution	0,05	/	/	0,01	/
costmap_common_params.yaml	obstacle_range	7	/	/	/	10
	raytrace_range	8	/	/	/	10

Figure 4: Table of parameter configurations.

- **Configuration 1**: the robot results to be slightly faster but having reduced the minimum distance from the obstacles and decreased the scanning horizon, this may lead to wrong path where the robot might get stuck or accidentally hit obstacles.
- **Configuration 2**: width and height of the local costmap window has been enlarged to a rectangular shape in order to have a greater view of the scene and its updating frequency has been increased, unfortunately having increased the inflation distance of the obstacles, the robot is no longer able to cross narrow passages.
- **Configuration 3**: the resolution of the global costmap has been improved in order to have better details and accuracy about the environment. This implies a more computational demanding simulation but enhanced path planning.
- **Configuration 4**: the ability of the robot to discover the environment and evaluate the costmap has been significantly improved.

4 Vision-based navigation

4.1 Camera configuration

In order to enable the camera the following lines have been uncommented:

```xml
<!-- rl_fra2mo_description/urdf/fra2mo.xacro -->

  <xacro:include filename="$(find
   ↪  rl_fra2mo_description)/urdf/d435_gazebo_macro.xacro" />

<xacro:if value="${DEPTH}" >
    <xacro:d435_gazebo_sensor parent="d435_link" />
  </xacro:if>
```

Then in order to correctly launch the aruco_ros package:

```xml
<!-- aruco_ros/launch/usb_cam_aruco.launch -->

 <arg name="camera"          default="/depth_camera/depth_camera"/>
  ↪  <!--HomeWork 4-->
    <arg name="marker_frame"    default="aruco_marker_frame"/>

```

and then:

```xml
<!-- fra2mo_2dnav/launch/fra2mo_nav_bringup.launch -->

        <include file="$(find aruco_ros)/launch/usb_cam_aruco.launch"/>
```

4.2 Navigation task

In order to send the robot in proximity of the aruco marker:

```xml
<!-- rl_fra2mo_description/launch/spawn_fra2mo_gazebo.launch -->

<arg name="findAruco" default="false"/>

<group if="$(arg findAruco)">
  <param name="goalNumber" value="1"/>
  <node pkg="tf" type="static_transform_publisher" name="goal_1_pub"
   ↪  args="-14 8.25 0 3.141 0 0 map goal1 100" />
</group>

```

In order to localize the aruco marker and move to the desired position:

```
//fra2mo_2dnav/tf_nav.cpp

else if ( (cmd == 4) && (goalNumber == 1) ) { //HomeWork 4.b
 ↪  ------------------

            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.at(0)[0];
            goal.target_pose.pose.position.y = _goal_pos.at(0)[1];
            goal.target_pose.pose.position.z = _goal_pos.at(0)[2];

            goal.target_pose.pose.orientation.w = _goal_or.at(0)[0];
            goal.target_pose.pose.orientation.x = _goal_or.at(0)[1];
            goal.target_pose.pose.orientation.y = _goal_or.at(0)[2];
            goal.target_pose.pose.orientation.z = _goal_or.at(0)[3];

            ROS_INFO("Sending Goal to find Aruco Marker");
            ac.sendGoal(goal);

            ac.waitForResult();

            if(ac.getState() == actionlib::SimpleClientGoalState::SUCCEEDED)
                ROS_INFO("The mobile robot has found the Aruco Marker");
            else
                ROS_INFO("The base failed to move for some reason");

            //Vision Task
            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 = _tfAruco.getOrigin().x()+1;
            goal.target_pose.pose.position.y = _tfAruco.getOrigin().y();
            goal.target_pose.pose.position.z = 0;

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

            ROS_INFO("Sending Goal to move away from Aruco Marker");
            ac.sendGoal(goal);

            ac.waitForResult();

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

        }

```

where the aruco marker pose has been retrieved:

```
//fra2mo_2dnav/tf_nav.cpp

TF_NAV::TF_NAV() {

    if(goalNumber == 1){
        ros::Rate r( 5 );
        tf::TransformListener listener;
        tf::StampedTransform tfBaseCamera;
        _aruco_pose_sub = _nh.subscribe("/aruco_single/pose", 1,
         ↪  &TF_NAV::arucoPoseCallback, this);


        try {
            listener.waitForTransform( "base_footprint", "camera_depth_optical_frame", ros::Time(0), ros::Duration(10.0) );
            listener.lookupTransform( "base_footprint", "camera_depth_optical_frame", ros::Time(0), tfBaseCamera );
        } catch ( tf::TransformException &ex ) {
            ROS_ERROR("%s", ex.what());
            r.sleep();
            return;
        }
        _tfBaseCamera = tfBaseCamera;
    }
}


void TF_NAV::arucoPoseCallback(const geometry_msgs::PoseStamped & msg){ //HomeWork 4

    tf::Vector3 arucoPosition(msg.pose.position.x, msg.pose.position.y, msg.pose.position.z);
    tf::Quaternion arucoOrientation(msg.pose.orientation.x, msg.pose.orientation.y, msg.pose.orientation.z, msg.pose.orientation.w);
    tf::Transform tfCameraAruco(arucoOrientation, arucoPosition);
    // aruco wrt world frame
    _tfAruco = _tfBase * _tfBaseCamera * tfCameraAruco;
    //Broadcast TF ---- HomeWork 4.c
    static tf::TransformBroadcaster br;
    br.sendTransform(tf::StampedTransform(_tfAruco, ros::Time::now(), "map", "aruco_frame"));

}
```

In order to test the simulation:

```
# Terminal 1

$ roslaunch fra2mo_2dnav fra2mo_nav_bringup.launch findAruco:=true
```

```
# Terminal 2

$ rosrun fra2mo_2dnav tf_nav
```

```
# Terminal 3

$ rqt_image_view
```

4.3 Broadcast TF aruco marker

In order to publish the aruco marker pose as tf:

```
//fra2mo_2dnav/tf_nav.cpp

void TF_NAV::arucoPoseCallback(const geometry_msgs::PoseStamped & msg){
↪  //HomeWork 4

    tf::Vector3 arucoPosition(msg.pose.position.x, msg.pose.position.y,
    ↪  msg.pose.position.z);
    tf::Quaternion arucoOrientation(msg.pose.orientation.x,
    ↪  msg.pose.orientation.y, msg.pose.orientation.z,
    ↪  msg.pose.orientation.w);
    tf::Transform tfCameraAruco(arucoOrientation, arucoPosition);
    // aruco wrt world frame
    _tfAruco = _tfBase * _tfBaseCamera * tfCameraAruco;
    //Broadcast TF ---- HomeWork 4.c
    static tf::TransformBroadcaster br;
    br.sendTransform(tf::StampedTransform(_tfAruco, ros::Time::now(), "map",
    ↪  "aruco_frame"));

}
```

It’s possible to check the publishing:

```
# Terminal 4

$ rostopic echo /tf
```

and looking for "aruco_marker_frame".