Robotic Arm Mini-project
Part 2 – ROS

What is it about?

During the second part of the robotic arm mini-project, you have to add ROS support to your robot.

Aims

At the end of this lab, you should:

- Know what ROS is about
- Have written a ROS node for the Arduino
- Have created a URDF model of your robot's kinematics
- Control and visualize a 3D model of your robot

Specific Challenges

- Many new concepts in this lab. You might find it handy to have the lecture slides around.
- Much more software-oriented than the previous labs.

Note

As usual, **document in your lab journal your findings**. Add **code snippets**, **screenshots**, **pictures** and link to **videos** as needed.

And do not forget: **write your lab journal as a text file using the Markdown syntax** and **push your journal and the pictures on GitHub**.

Preliminary steps

Step 1 – Reboot on Linux

Make sure you are running Linux. Otherwise, reboot and switch to Linux. **ROS does not work on Windows**.

Part I

Servos control with ROS

Step 1 – Prepare the hardware

Plug a servo to the Arduino, make sure you can get the servo to move using the Arduino IDE and the following simple code sample:

```
#include <Servo.h>

Servo myservo;

int pos = 0;

void setup() {
  myservo.attach(9);  // attaches the servo on pin 9 to the servo object
}

void loop() {
  for (pos = 0; pos <= 180; pos += 1) {
    // in steps of 1 degree
    myservo.write(pos);
    delay(15);
  }
  for (pos = 180; pos >= 0; pos -= 1) {
    myservo.write(pos);
    delay(15);
  }
}
```

Step 2 – First steps with ROS

As discussed in the lecture, you always need to start `roscore` before being able to launch any other node.

Open a terminal and start it with the command `roscore`.

In another window, type `rostopic list` to list all the available topics. At this point, you should only see two of them. Search on the web what is the use of the `/rosout` topic. Report it in your lab journal.

Let's publish something on a new topic. Type:

```
> rostopic pub /test std_msgs/String "Hello"
```

Now type again `rostopic list`. You should see a new topic `/test`.

Open a different terminal, and type `rostopic echo /test` to display on the console the messages exchanged on the `/test` topic. Nothing should be display at this point, since our `"Hello"` message was published ***before*** we started `rostopic echo`.

Try to publish another message on the `/test` topic. This time, you should see it.

> **Note**
> Quickly enough, you will end up with many terminal windows open at the same time. You might find it usefule to use `ctrl+shift+t` to instead create tabs in the same terminal window.

As you have noticed, the ***type*** of our `/test` topic is `std_msgs/String`. ROS offers many standard datatypes. Start to familiarize yourself with the basic message types by visiting wiki.ros.org/std_msgs.

Step 3 – RViz

RViz is the main 3D visualisation tool provided with ROS. Start it now (simply type `rviz` in a different terminal). At this point, RViz does not have much to show (no ROS node is running yet, except… RViz itself), so the 3D viewport is simply an empty grid (Figure 1).

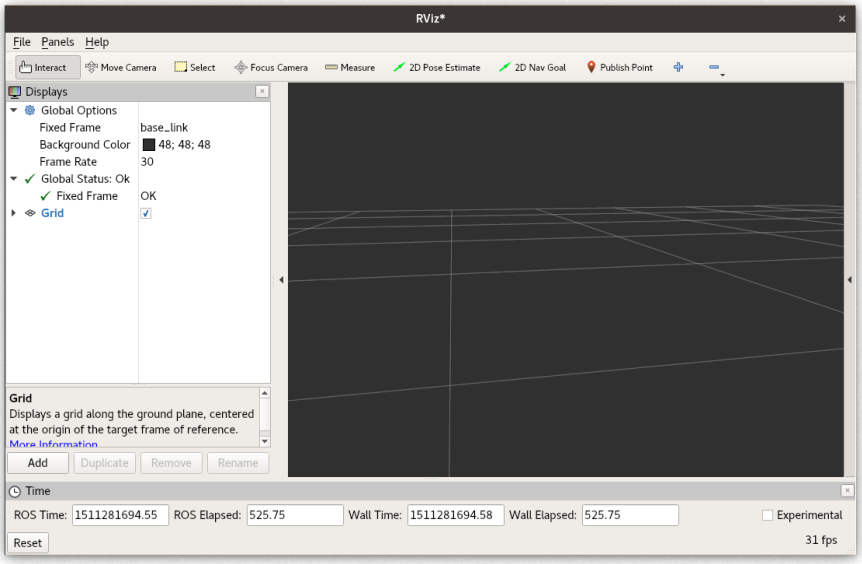

Figure 1: The default, initial RViz window.

RViz visualization rely on plugins: one plugin for every datatype we want to visualise (images, 3D models, point clouds, etc.). Figure out how to add visualisation plugins, and explore what is available.

Step 4 – Configure ROS for the Arduino

As the Arduino does not feature a network socket, we need to use a serial bridge instead. `rosserial` is such a ROS ***bridge*** that transparently transport ROS messages over a serial connection.

`rosserial_arduino` is a `rosserial` ***client*** for the Arduino (i.e. the library that runs ***on the Arduino*** to deserialize the ROS messages).

You can install `rosserial` easily:

```
> sudo apt install ros-kinetic-rosserial-python ros-kinetic-rosserial-arduino
```

To make it transparently available in the Arduino IDE, you also need to install it as an Arduino library:

```
> cd $HOME/sketchbook/libraries
> rosrun rosserial_arduino make_libraries.py .
```

Restart the Arduino IDE. You should now have access to many ROS examples (Figure 2).

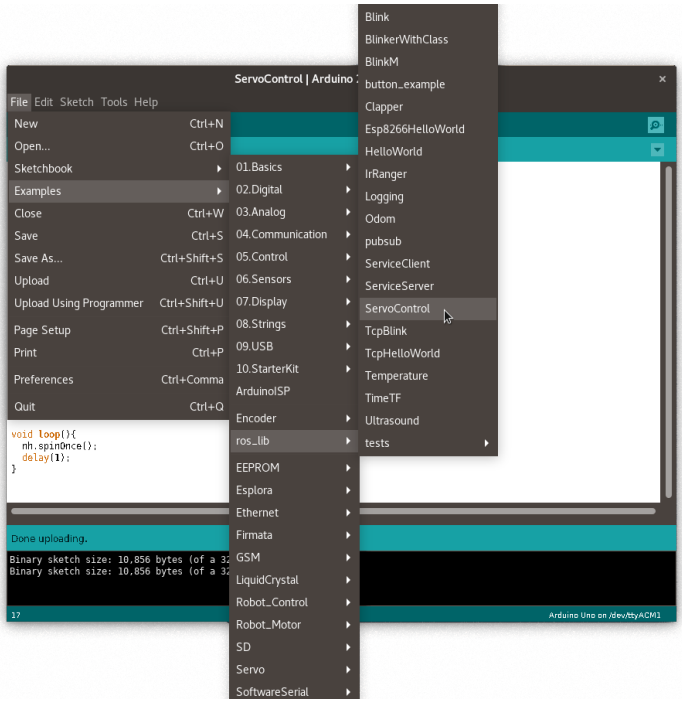

Figure 2: Examples of ROS nodes for the Arduino

Step 5 – Write a ROS node for your Arduino

Use the following code sample to control a servo by sending one integer between 0 and 180:

```
1  #include <ros.h>
2  #include <std_msgs/UInt16.h>
3  #include <Servo.h>
4
5  using namespace ros;
6
7  NodeHandle  nh;
8  Servo servo;
9
10 void cb( const std_msgs::UInt16& msg){
11   servo.write(msg.data); // 0-180
12 }
13
14 Subscriber<std_msgs::UInt16> sub("servo", cb);
15
16 void setup(){
17   nh.initNode();
18   nh.subscribe(sub);
19
20   servo.attach(9); //attach it to pin 9
21 }
22
23 void loop(){
24   nh.spinOnce();
25   delay(1);
26 }
```

Note

Analyse this code example. In particular, **explain in your lab journal what is the function defined at line 10, and the role of the object instantiated at line 14**.

Compile and upload the code to the Arduino.

In a terminal, start `rosserial` (changing the serial port of your Arduino as required):

```
> rosrun rosserial_python serial_node.py /dev/ttyACM0
```

Now, call `rostopic pub` with the adequate parameter to move your servo-motor.

Part II

3D model of your arm

We have a first ROS node, able to control a servo motor. However, to do useful things with our arm (like 3D motion planning), we need to describe its complete kinematic model.

The next step is indeed to build a 3D model of the arm. We need to create a **geometric and kinematic description of the arm** using the **URDF format**.

Step 1 – Visualise an existing URDF file in RViz

Create a new directory called `robot-project` and a sub-directory called `models`. Save the following URDF file in this subdirectory as `robot-arm.urdf`.

Note

You can also download this file from the DLE.

```
<?xml version="1.0"?>
<robot name="roco_arm">
  <link name="base_link">
    <visual>
      <geometry>
        <cylinder length="0.06" radius="0.1"/>
      </geometry>
    </visual>
  </link>

  <link name="first_segment">
    <visual>
      <geometry>
        <box size="0.6 0.05 0.1"/>
      </geometry>
      <origin rpy="0 0 0" xyz="-0.3 0 0" />
    </visual>
  </link>

  <joint name="base_to_first" type="revolute">
    <axis xyz="0 1 0" />
    <limit effort="1000" lower="0"
                   upper="3.14" velocity="0.5" />
    <parent link="base_link"/>
```

```
    <child link="first_segment"/>
    <origin xyz="0 0 0.03" />
  </joint>
</robot>
```

Load this file as the ***description*** of your robot:

```
> rosparam set robot_description -t models/robot-arm.urdf
```

Next, launch the `robot_state_publisher` and `joint_state_publisher` nodes in two terminals:

```
> rosrun robot_state_publisher robot_state_publisher
```

```
> rosrun joint_state_publisher joint_state_publisher _use_gui:=true
```

The `robot_state_publisher` node reads the robot description, and broadcasts the 6D transformations (TF frames) corresponding to each of the links described in the URDF file.

The `joint_state_publisher` node reads as well the robot description and creates a GUI with one slider per joint, making it easy to manipulate the pose of our robot.

Finally, add the `Robot model` plugin to RViz and set the `Fixed frame` to `/base_link` (Figure 3).

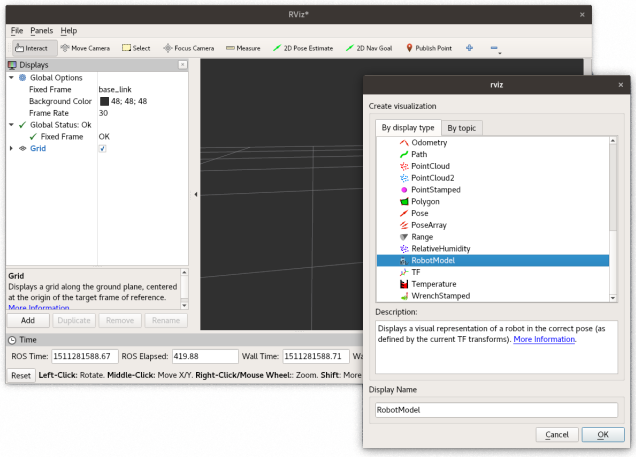

Figure 3: Adding the `Robot model` visualisation plugin to RViz

You should see the following model in the 3D viewport (Figure 4):

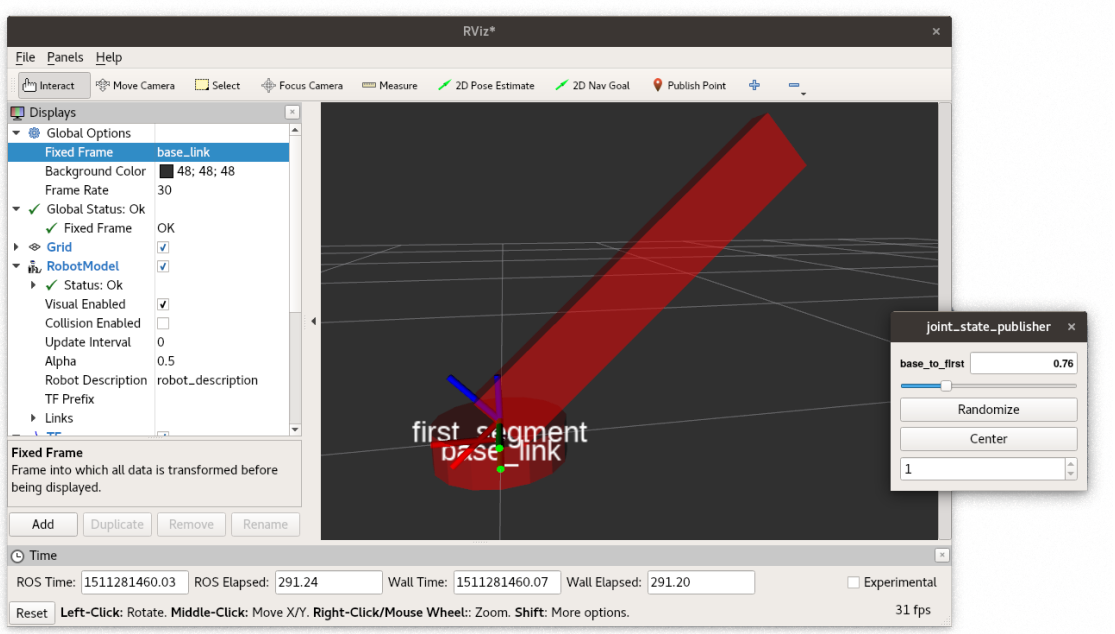

Figure 4: A first, simple, URDF model, visualised in RViz. On the right, the `joint_state_publisher` interface.

Step 2 – Create the URDF file of your robot

Using `robot-arm.urdf` as a starting point, complete the URDF to accurately model your arm. Measure precisely the dimension of each segment, and position accurately the joints.

> **Note**
>
> You might find it useful to use your CAD software to quickly measure the segments and the position of the joint axes.

> **Taking it further**
>
> Some CAD tools (like Solidworks) have extensions to export directly to URDF. Check online!

If you wish to check your model, reload the robot description, and restart the `robot_state_publisher` and `joint_state_publisher` nodes. You do not need to restart RViz, but you need to desactivate and re-activate the `robot model` plugin to update the rendering.

Taking it further

Instead of simple geometric primitive, you can use STL meshes (like the ones you 3D printed) for the visuals of your arm. Check the URDF documentation to learn how to do that.

Note

Do not forget to include screenshots of your URDF model in your lab journal.

Part III

Control the servo-motors from the robot's joint state

Lastly, modify the Arduino code to directly read the robot joint state and to control the servo-motors accordingly.

Note

As short video-clip of your arm moving alongside the 3D model in RViz is certainly appropriate for your lab journal!