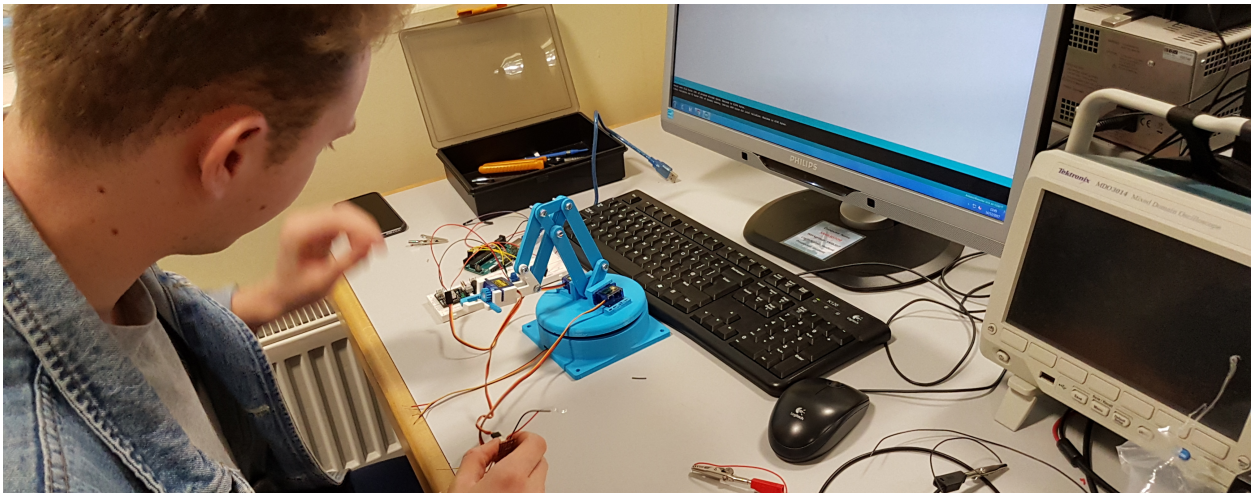
A Robotic Arm from Scratch



What is it about?

This project is about **designing your own robotic arm** (with at least 2 degrees of freedom, but you can add more), **programming it with ROS**, building a **3D visualisation** for it, and, if time permit, using a **3D motion planner** to control it automatically!

Be ready to **design & 3D print** the parts of your robot, **program an Arduino to control servo-motors** and **solder bits together**!

Part I

Getting started with the Robotic Arm mini-project

This project is about **designing your own robotic arm** (with at least 2 degrees of freedom, but you can add more), **programming it with ROS**, building a **3D visualisation** for it, and, if time permit, using a **3D motion planner** to control it automatically!

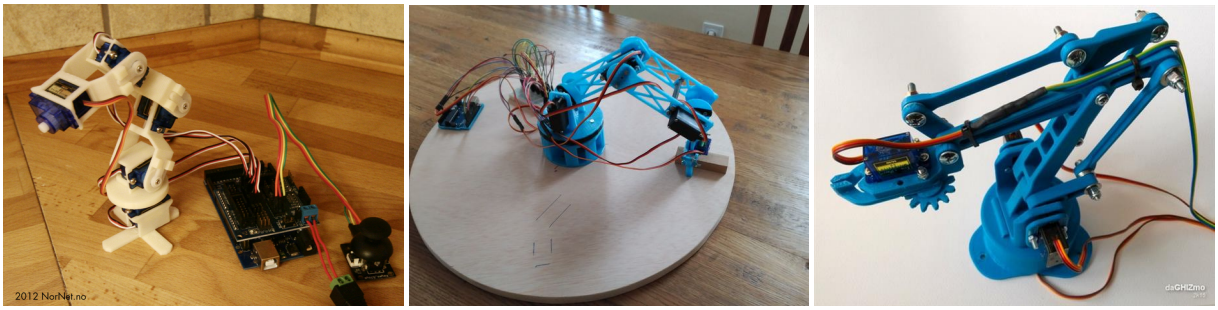


Figure 1: Three examples of 3D printed arms. The first one has 5 degrees of freedom, the second one 4 DoFs, the last one 3 DoFs.

> **Note**
>
> For this project, it is best to **work in pairs**, but feel free to start alone or to form a bigger group if you prefer.

Indicative working plan

- **Week 1**: control of two servo-motors with the Arduino; use of potentiometers to control them
- **Week 2**: Mock-up arm with cardboard; refresher on 3D design with SolidWorks with Julian; start of the 3D printing
- **Week 3**: Design of the arm continued; arm assembly
- **Week 4**: ROS on Arduino; control of the servo using ROS
- **Week 5**: 3D model of the arm, visualisation of the arm in RViz
- **Week 6**: Finalisation of arms; if time permit, 3D motion planning using ROS

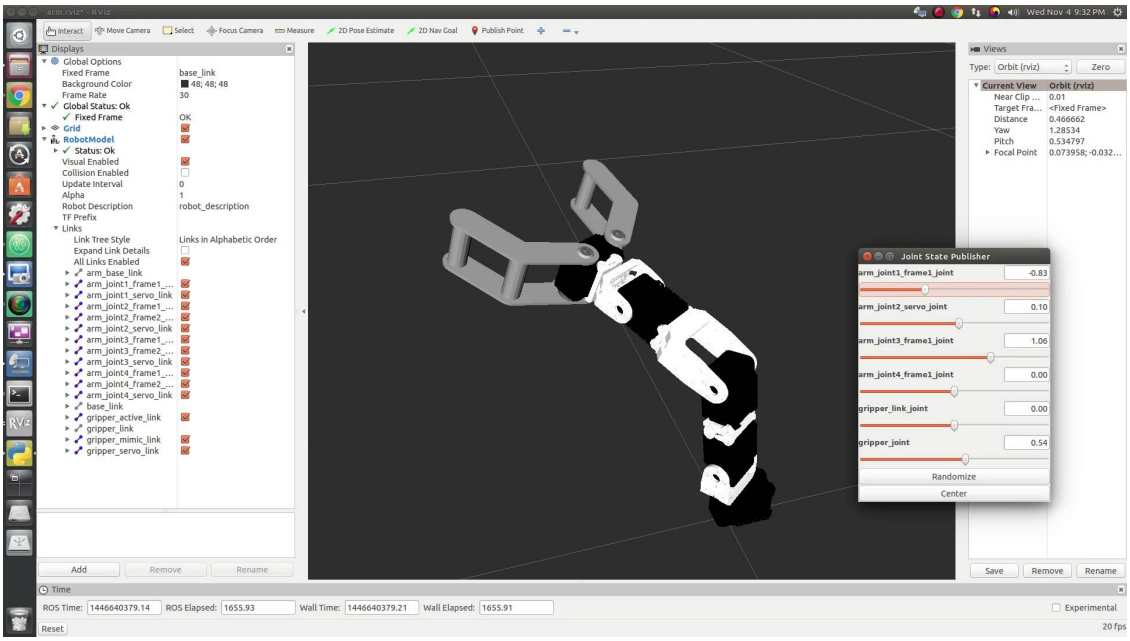


Figure 2: Later during the project, you will create a 3D interface to control your arm, using the standard ROS tools like RViz, pictured here.

Part II

Servo-motor control with an Arduino

Step 1 – Get ready

First, get from the workshop 2 RC mini-servos as well as an Arduino board and two potentiometers.

> **Note**
>
> Take a plastic bin as well, that you can label with you names. The bins can be left in the lockers when you are not working on your project.

> **Taking it further**
>
> If you wish, you can create a design with additional degrees of freedom. You can borrow more servos from the workshop. You may as well want to source additional actuators by yourselves: small servo-motors or stepper motors can be found online for less than 5 pounds.

The RC servomotor (Fig. 3) is a unit consisting of a small electric motor driving a gear train. It often uses a potentiometer to measure angular position measurement. Often such a unit is small and cheap although more expensive designs using metal gears and digital control circuitry are also available.

The RC servo generally incorporates circuitry that implements position feedback control (Fig. 4). The output position is compared to the commanded target position. This gives rise to an error signal. The error drives the electric motor in appropriate direction. When it becomes zero the servo stops moving and reaches equilibrium. A position servomotor usually sets the output target angle through the width of a control signal (pulse width) as show in Fig. 5.

A RC servomotor typically has three connections:

- The black wire is usually the ground, the red wire is connected to V_{ref} (5V) and the white (or yellow) wire is the control pulse input.
- If this is not the case on your servo, check its datasheet
- You will need pins to connect the female plug to the female Arduino outputs

Figure 3: Micro servo motor

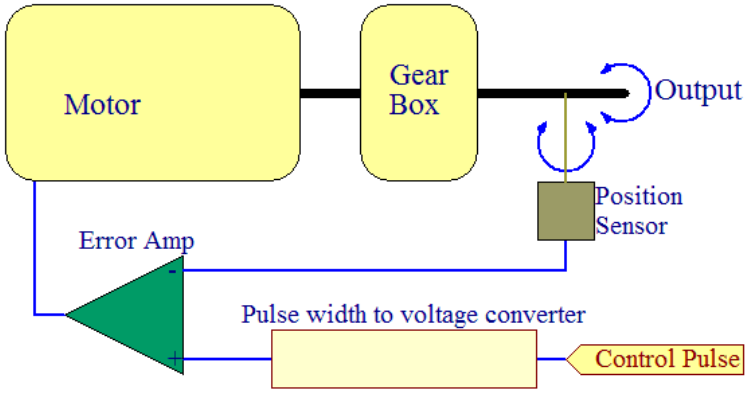
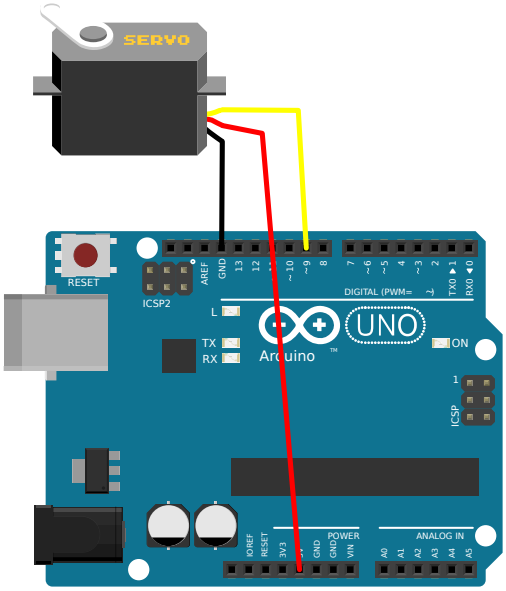


Figure 4: RC servo control showing feedback pathways

Step 2 – Control an RC servo



1. Attach the RC servo power connections to the Arduino power output pins
2. Attach the control input to a PWM output pin on the Arduino (e.g. port 9)
3. Write a simple program that runs the servo to generate an output movement back and forward over its entire range so its movement follows a low frequency sine wave of frequency around 0.2Hz. Use the code below as a starting point.

```
#include <Servo.h>

Servo myservo;  // create servo object

int val;    // variable to read analog value

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

void loop() {
  val = 10;
  myservo.write(val); // sets the servo position
  delay(15); // waits for the servo to get there
}
```

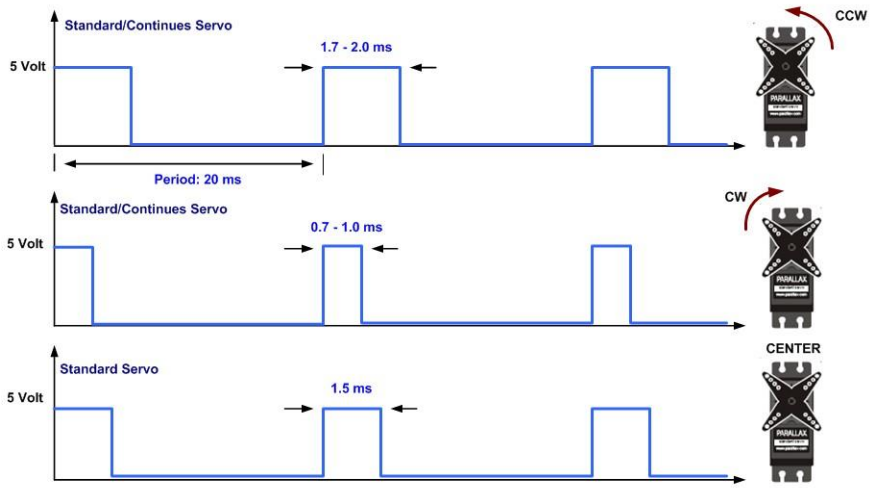
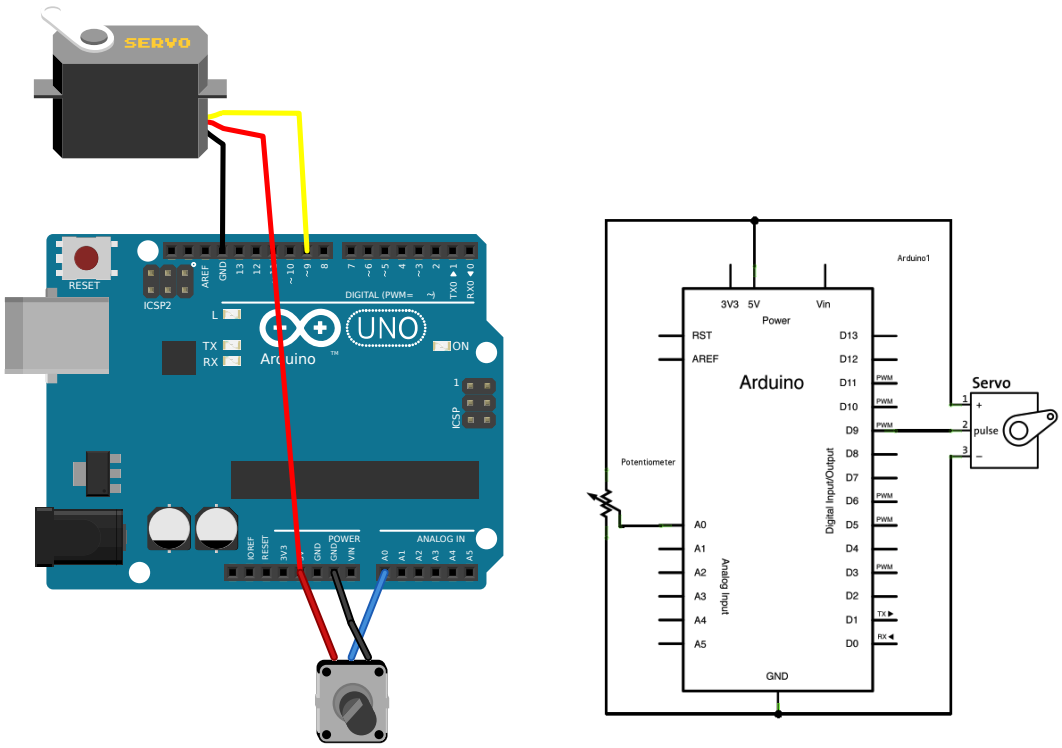


Figure 5: RC servo operation by pulse width control

Step 3 – Control the servo with a potentiometer



1. Wire the provided potentiometer to one of the Arduino analog port. Write a program that reads the value of the potentiometer and writes it onto the serial port.
2. Use the potentiometer to rotate the servo-motor.

The code below reads the potentiometer. Look online for way to display back this value using the `SerialPort`.

```
int potpin = 0;  // analog pin used for potentiometer
int val;    // variable to read analog value

void loop() {
  val = analogRead(potpin); // reads potentiometer
  val = map(val, 0, 1023, 0, 180); // remap values from 0 - 1023 to 0 - 180
}
```

Step 4 – A robot arm mock-up

1. Using pieces of cardboard, create an arm and attach it to the motor shaft (you can draw inspiration from the picture below, or come up with your own ideas)

2. Estimate the torque of the servo-motor by hanging a fixed mass to the arm, at increasing distance of the motor shaft. Document the process, and check your estimate using the servo-motor datasheet (you can find it easily online)
3. Modify your arm to insert the second servo motor
4. Write a program that control both servos with the two provided potentiometers.

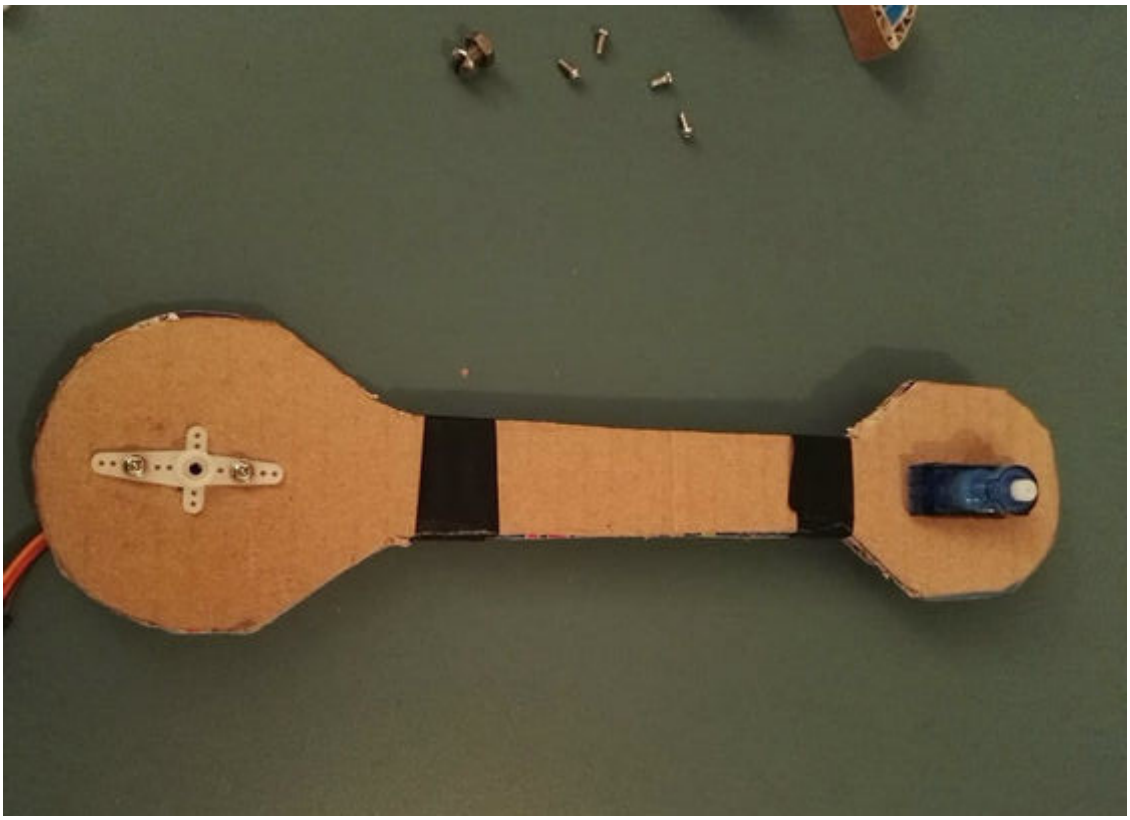


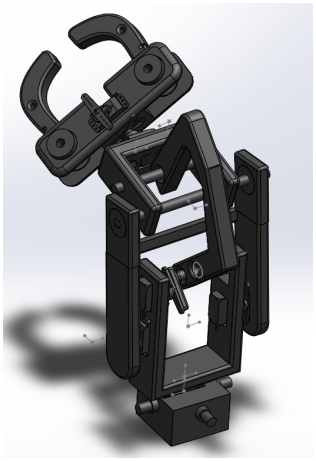
Figure 6: A possible assembly of 2 servos onto a cardboard arm. Taken from `http://www.instructables.com/id/CARDBIRD-the-Cardboard-Robotic-Arm/`

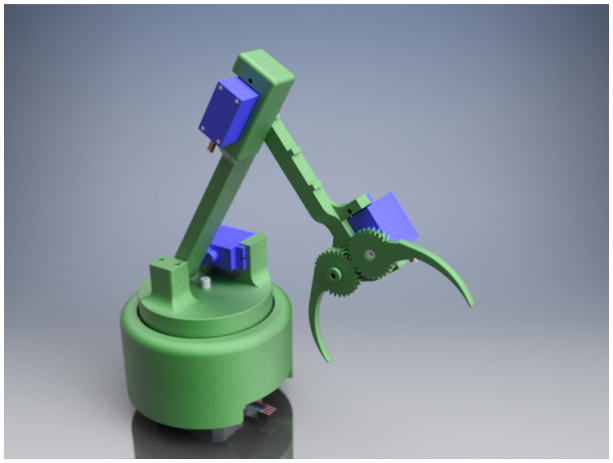
Part III

Design and 3D print your arm

Using the cardboard mock-up as an initial reference, design your arm in SolidWorks.

Examples of designs from the previous years:





Part IV

Servos control with ROS

Step 1 – Reboot on Linux

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

Step 2 – 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 3 – First steps with ROS

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.

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 4 – 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 7).

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 5 – 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).

It should alrady be installed on the computer. However, if this is not the case, you can install `rosserial` easily:

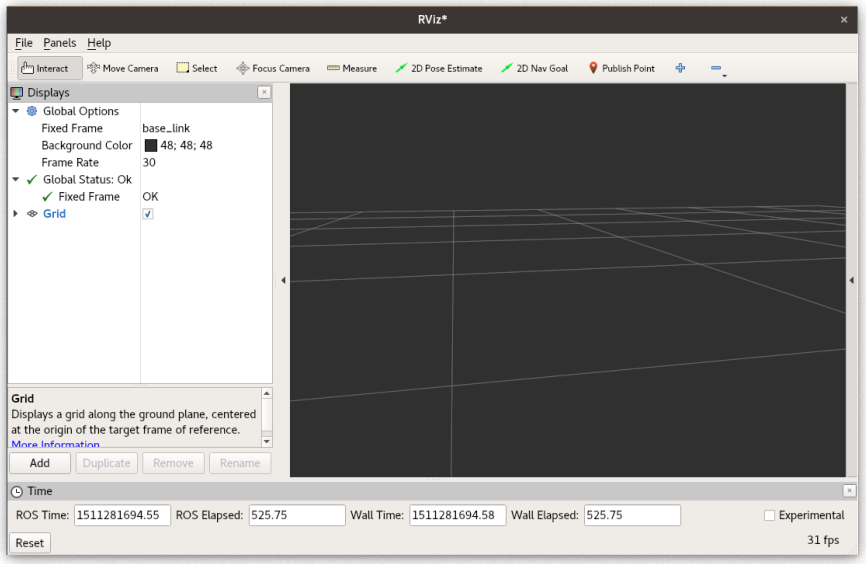


Figure 7: The default, initial RViz window.

```
> sudo apt install ros-melodic-rosserial-python ros-melodic-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 8).

Step 6 – 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;
```

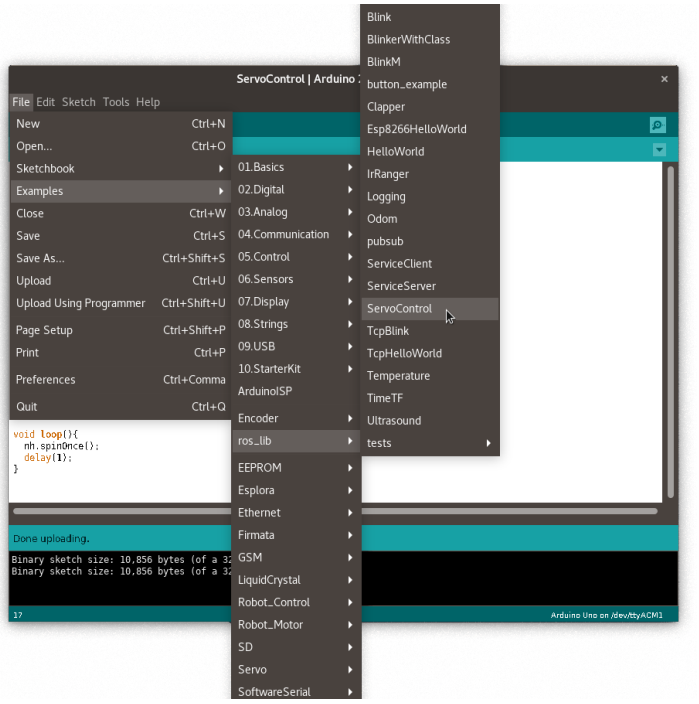


Figure 8: Examples of ROS nodes for the Arduino

```
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, 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 V

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 GitHub repo.

```
<?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 9).

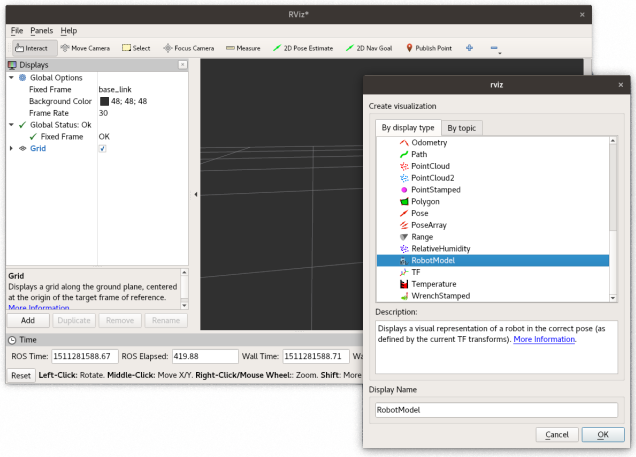


Figure 9: Adding the `Robot model` visualisation plugin to RViz

You should see the following model in the 3D viewport (Figure 10):

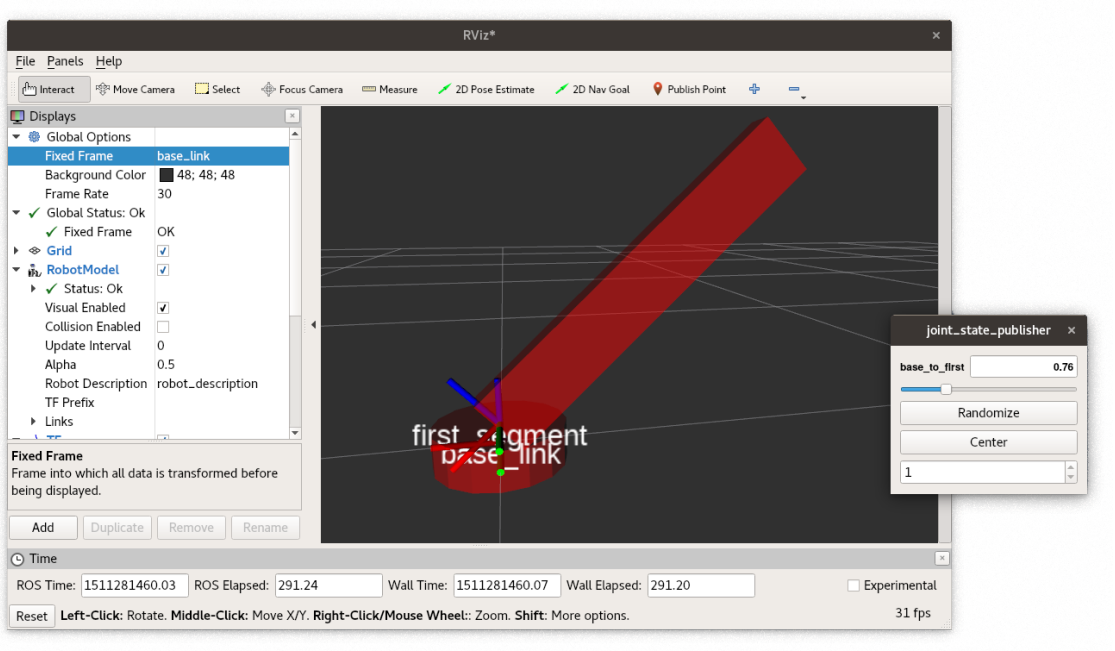


Figure 10: A first, simple, URDF model, visualised in RViz. On the right, the `joint_state_publisher` interface.

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.

Part VI

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 Twitter/Instagram account!

Part VII

Automatic motion planning

The last step is to use ROS to plan a full arm motion. Check the MoveIt! tutorials, and get your robot to move!