xArm Unity Interface

Tristan Zeller

November 2022

1 Introduction

This documentation is intended as an installation and user guide for the user interface and control software of the xArm 6. This marks the first iteration of the System in its current form which combines ufactory's xArm 6 and Unity-Robotics-Hub.

1.1 Goals of this Project

Previous test setups at the Hearing Research Laboratory HRL have used several speakers mounted on rails around a patient. Speakers were either stationary or could be moved on the rail with one degree of freedom. Such a setup is displayed in *figure 1*. Using a robotic arm is another approach to providing a moving base. A speaker mounted on a robotic arm can execute more complex movements in three dimensions. Combining this with a 3D user-interface makes the setup more user friendly, especially for people with no background in programming and/or engineering.

Figure 1: Setup which uses twelve speakers mounted on a circular rail. *Source: artorg.unibe.ch*

Figure 2: xArm 6 with the mounted speaker. *Source: Project Nautilus Documentation*

1.2 xArm 6

xArm 6 by UFACTORY is a six axys robotic arm with six degrees of freedom 6 DOF. The xArm 6 has a reach of approximately 70 cm and a payload capacity of 5 kg. It can move the end effector up to 1 m/s and has a repeatability of 0.1 mm. Ufactory supplies ROS packages and both python 3 and C++ SDK's for the xArm series. A KEF Ci130QR coaxial speaker with a 3D-printed housing is mounted to the arm. The xArm 6 with the mounted speaker is depicted in *figure 2*.

1.3 Unity Game Engine

Unity is a versatile game engine which was originally intended to aid in the creation of both 2D and 3D video games. It implements a diverse set of features to create and navigate 3D space and rigid body physics. Even though unity started as a tool to make video games, it now offers packages for AR and VR development, as well as tools for the simulation and control of robots. The Unity-Robotics-Hub comprises of packages for both unity and ROS, enabling

communication between the two.

1.4 Robot Operating System ROS

ROS (which despite the name is not an operating system) offers a standardised method to combine individual robotic components including a wide array of sensors, actors and complete robotic systems. This is achieved by serialising data and sending it in agreed upon formats, the TCP protocol. In this project, ROS is used to link the xArm to and control control it using Unity.

2 Installation Guide

This software package requires some third party installations:

- Ubuntu 20.04 on Windows Subsystem for Linux WSL
- ROS Noetic (https://jackkawell.wordpress.com/2020/06/12/ros-wsl2/ This blog post shows how to configure Ubuntu in WSL2 and install ROS. Make sure to install WSL2, not WSL1 and to also install the correct versions of ROS (noetic) and Ubuntu (20.04). If you experience difficulty getting WSL to work, https://github.com/ishkapoor2000/Install_ROS_Noetic_On_WSL might also help solving some issues. If you can not get the default WSL version to change from WSL1 to WSL2, try the command

```
wsl --set-version ubuntu-20.04 2
```

  in your PowerShell)
- Unit Hub (https://unity3d.com/get-unity/download)
- python3 (may be already pre installed when you have installed Ubuntu 20.04 on WSL2)
- the xArm-Python-SDK (https://github.com/xArm-Developer/xArm-Python-SDK) To install, run the following commands in the Ubuntu 20.04 terminal:

```
$ git clone https://github.com/xArm-Developer/
xArm-Python-SDK.git
$ cd xArm-Python-SDK
$ sudo python setup.py install
```

- UFACTORY Studio (https://www.ufactory.cc/download-xarm-robot)

To run the custom xArm 6 controller, download both the contents for the xArm6 Unity project under https://github.com/zivi80301/xArm6-Unity_controller and the catkin_ws workspace folder under https://github.com/zivi80301/xArm6-ROS. Create a new folder under: *C:\Users\⟨Your Username⟩* and extract the contents of xArm6-Unity_Controller into it. The catkin

workspace folder should be placed under *\\wsl$\Ubuntu-20.04\home\(Your Username)*. Then open up a WSL terminal and run the following commands:

```
$ cd ~/catkin_ws
$ catkin_make
$ source devel/setup.bash
```

The second command builds the ROS worskpace. This may take a few seconds. A lot of text should be output in the terminal. As long as the process ends with [100%] and no red error messages, everything should be fine. The other two should both run quickly. Navigate to your *\\wsl$\Ubuntu-20.04\home\(Your Username)* folder and open the .bashrc file. Add the following line to the very end of the text file:

```
source ~/catkin_ws/devel/setup.bash
```

To be able to execute pyton scripts in ROS, the corresponding .py files need to be set as executable. To do this, run the following commands in your Ubuntu 20.04 terminal:

```
$ cd ~/catkin_ws/src/ROS-TCP-Endpoint/src/
ros_tcp_endpoint
$ chmod +x default_server_endpoint.py
$ cd ~/catkin_ws/src/xarm_bringup/scripts
$ chmod +x setpoint_sub.py
$ chmod +x joint_state_publisher.py
$ chmod +x reset.py
$ chmod +x clear_err.py
```

Open Unity Hub, click the add button and select the xArm6 folder. When you try to open the xArm6 project, Unity will ask you to download the apropriate version. Do so without selecting any additional downloads. Make sure to read the xArm user manual (http://download.ufactory.cc/xarm/en/xArm%20User%20Manual.pdf?v=1578910898247) before trying to start the robot. Refer to the user manual to set up the connection between the robot and your computer. Before starting the controller, make sure to adjust the end effector self-collision prevention and initial position according to the dimensions of the speaker which is also explained in the manual. If you fail to do this, the robot may hit itself or the table it is mounted on with the speaker.

3 User Guide

Before Starting the control software, start the robotic arm in accordance with the xArm user manual. The proper sequence is as follows:

1. Securely plug in all the required cables, make sure the emergency stop (E-stop) button is pressed.

2. Turn on the power supply

3. If the controller does not start on its own, press the power button on the front side of the controller box

4. Twist the E-stop in the direction of the indicator arrows

5. You can check if everything works correctly using UFACTORY Studio

A few individual pieces are necessary to run the application used to control the robotic arm. First open the Unity project called xArm6 and the Ubuntu 20.04 terminal. Then run the following command in the Terminal:

```
$ roslaunch ros_tcp_endpoint endpoint.launch
```

If this throws an error, check whether the robot is connected the pc. This connects ROS to the Unity project and starts all the necessary nodes to receive data from Unity. Make sure NOT to close this window! If you encounter problems during operation, try stopping this process by using the keyboard shortcut ctrl + c while in the terminal and then relaunching ROS. This concludes all the external steps. From here on out, everything can be done inside unity.

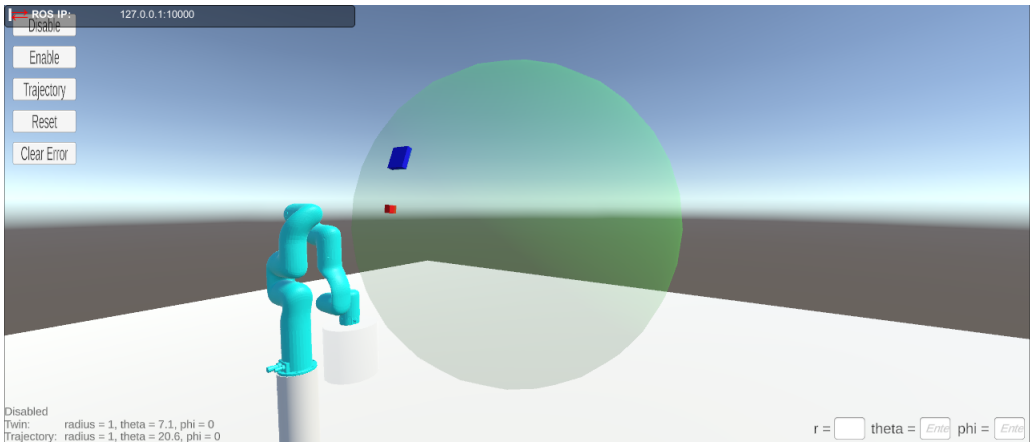

Figure 3: User interface inside the Unity application

In unity, click the play button to start the application. *Figure 3* shows the user interface inside the Unity application. The color of the two arrows in the top left indicate, whether Unity and ROS connected successfully. If they are red as in the figure, the connection was not successful. If they are blue, you are good to go. You can now use the arrow keys or w,a,s,d to move the camera around the scene. Use mouse scroll or two finger scrolling on a track pad to zoom in or out. You can drag the red and blue boxes using your mouse. The red box is a digital twin of the robot arm, meaning the robot will mirror the position of the red box in the virtual space. The blue box is the trajectory target. An array of buttons is visible on the left side of the screen, as well as

a text box. The text indicates the current mode of operation and shows the position of the red an blue boxes in spherical coordinates. Three input fields on the right allow you to precisely change the position of both boxes using spherical coordinates. By default, the robot is disabled. Click enable to start live tracking. You can now move the red box around the sphere to control the robot. In this mode, the input fields control the red box. If you wish to move the speaker in a defined arc, move the red box to the desired initial position and the blue box to the desired target position. The arm will performed this movement once you click the trajectory button. In this mode, the input fields control the trajectory target. If at any point you wish to return the robot to its home position, click the reset button. This disables the unity controller and resets the robot position. If you have navigated the robot into a singularity, collision or end up with another error but do not wish to reset the machine, use the clear error button. Then change the trajectory and re-enable the robot. Some joint configurations may lock the robot in a state where it can not be reset using these means. If such a configuration occurs, use the manual mode in UFACTORY Studio to maneuver the robot into a different position before resetting. Unity offers animation tools which allow you to choreograph more complex movements which can be used to create repeatable tests.

4 Technical Details

This section explains important segments of code and justifies some decisions that went into the design process.

4.1 ROS

ROS is used to combine the user interface and control inputs from Unity with the xArm SDK. The launch file ros_tcp_endpoint.launch is used to start the relatively simple ROS network consisting of the following nodes:

- rosout
- unity_endpoint
- pos_rot_sub
- joint_state_sub
- reset_server
- clear_error_server

4.1.1 rosout

The rosout node is part of the ROS master. It is used publish, subscribe to and log messages .

4.1.2 unity_endpoint

This node communicates with the Unity-Robotics-Hub and forms the ROS counter part to the ROS TCP Connector Unity packege. Everything that happens inside of unity and connects to ROS does so through the unity_endpoint node. This enables unity to publish positional and rotational data to the robot, receive joint angles from the robot and call ROS services.

4.1.3 pos_rot_sub

This node subscribes to the pos_rot topic which contains positional data in the form of a three-dimensional vector and rotational data in the form of a quaternion. This is achieved with a python script.

```
from tf.transformations import euler_from_quaternion
(r,p,y) = euler_from_quaternion(x,y,z,w)
xArm6.set_position(
    pos_x,pos_y,pos_z,r,p,y,radius=1000)
```

tf.transformations is necessary, since the xArm SDK requires rotation commands in the roll pitch yaw format. The code displayed above is set in the subscriber callback and runs whenever a new PosRot message is published to the pos_rot topic.

4.1.4 joint_state_sub

This is the node used to visualise the robot state inside the Unity application. On the ROS side,

```
xArm6.arm.get_joint_states(is_radian=False)[1][0]
```

is used to get the joint angles of the robotic arm.

4.1.5 reset_server

The reset_srv service server runs on this node.This service uses the std_srvs/Empty service type. On service call, the following python script clears any errors, sets mode and state to zero and resets the robot to its initial position:

```
arm.motion_enable(True,8)
arm.set_mode(mode=0)
arm.set_state(state=0)
arm.reset(wait=True)
```

4.1.6 clear_error_server

The clear_error_srv service runs on this node. It behaves very similarly to the reset_srv. Contrary to the reset service, the clear error service does not reset the robot position, making it possible to clear errors in a less disruptive manner. The code works very similarly as well:

```
arm.motion_enable(True,8)
arm.set_mode(mode=0)
arm.set_state(state=0)
```

4.2 Unity

The core idea of the Unity control interface is to implement a digital twin of the robot arm which can be manipulated in the Unity environment. This manipulation should be mirrored by the real robot. Choreographing different poses, constraining the robots movement to keep the speaker membrane perpendicular to the patient and real time tracking are all implemented inside unity. Furthermore, some visual feedback is provided by the user interface which also offers some functionality to clear errors. The application consists of three major components.

4.2.1 Controller Driver

This part constrains the controller object to the surface of a sphere centered around the head of a patient. Converting the position of the mouse cursor to 3d coordinates makes it possible to drag the controller object around the scene:

```
Vector3 GetMouseAsWorldPoint()
    {
        Vector3 mousePoint = Input.mousePosition;
        mousePoint.z = mZCoord;
        return Camera.main.ScreenToWorldPoint(
            mousePoint);
    }

private void OnMouseDrag()
    {
        controller.position =
        GetMouseAsWorldPoint() + mOffset;
    }
```

Using some geometric manipulation offered by the Unity scripting API enables the possibility to restrict the controller to the surface of a sphere with a defined radius:

```
controller.position = Vector3.MoveTowards(
    transform.position,
    target.position,
    Vector3.Distance(
        transform.position, target.position)
    - radius);

controller.LookAt(target, Vector3.back);
```

Trajectory planning between two points **A to B** is also a useful feature. This is implemented by adding a trajectory target to the scene and adding a trigger event to move the controller object to the trajectory target:

```
controller.position = Vector3.MoveTowards(
    controller.position,
    trajectoryTarget.position,
    step);
```

Lastly, text input fields can be used to offer a more accurate way of moving the controller.Spherical coordinates are used to achieve this. The following method achieves a rotation in phi direction and adjust the axis of rotation of theta as this axis is not static:

```
void SetPhi(float phi)
    {
        controller.RotateAround(
            target.position, Vector3.down, phi);
        thetaAxis = Quaternion.AngleAxis(
            phi, Vector3.down)
        * thetaAxis;
    }
```

The rotation along theta works much in the same way:

```
void SetTheta(float theta)
    {
        controller.RotateAround(
            target.position, thetaAxis, theta);
    }
```

4.2.2 User Interface

The user interface comprises two main parts. Input options in the form of buttons and input fields and feedback in the form of text boxes displaying information about the scene and the animated robot. The positions of the target and controller object are used to calculate the displayed coordinates First cartesian coordinates with the target object as the origin point are calculated, then these coordinates are used to calculate radius, phi and theta values to display spherical coordinates:

```
x = controller.position.x - target.position.x;
y = controller.position.y - target.position.y;
z = controller.position.z - target.position.z;

r = Mathf.Sqrt(Mathf.Pow(X, 2.0f)
    + Mathf.Pow(y, 2.0f) + Mathf.Pow(z, 2.0f));
t = -Mathf.Acos(y / r);
p = Mathf.Acos(X / Mathf.Sqrt(Mathf.Pow(x, 2.0f)
```

```
    + Mathf.Pow(z, 2.0f))) * Mathf.Sign(z);
```

The joint states provided by the ROS publisher are read and used to manipulate the rig of the robot 3d model in unity. This is achieved by using quaternions to rotate the bones. This makes it possible to restrict the joints to only one axis of rotation and to cascade the rotations to mimic the movement of the real arm:

```
joint1.rotation = Quaternion.Euler(0, angle1, 0);
joint2.rotation = joint1.rotation
    * Quaternion.Euler(0, 0, angle2);
joint3.rotation = joint2.rotation
    * Quaternion.Euler(0, 0, angle3);
joint4.rotation = joint3.rotation
    * Quaternion.Euler(0, angle4, 0);
joint5.rotation = joint4.rotation
    * Quaternion.Euler(0, 0, angle5);
```

The UI buttons are much more straight forward. They mainly toggle certain functions in the controller driver on or off.

5 Discussion

In its current states, the Unity control application is more of a framework to build experiments on rather than a finished product. Anything more complex than a simple A to B movement requires the usage of the Unity animation tool. Furthermore, it is currently not possible to adaptively change the speed at which the speaker is moved. Real-time control is also rather choppy and has some delay, which could be fixed by using the robot in velocity control mode or online trajectory planing mode, instead of position control mode. In its current state, the entire package still requires a few steps to run and necessitates a number of other installations to work. While the clear error and reset functions fix many problem that would require a user to switch to UFACTORY Studio, it is still possible to lock the robot in an unfortunate pose and require a switch to manual mode. In the future, a hearing test parser which automatically generates animations and incorporates audio may be a worthwhile endeavor. Concerning audio, there currently is no way of controlling the speaker when connected to the arm. While Unity offers easy ways of triggering and outputting audio, the xArm end effector connector does not support audio signals. This may be fixed with a Bluetooth receiver which connects to the pc. This way, the standard Unity audio engine could be used and no additional cables would be required.