The STANLEy User Manual

Stabilization and Transducer Alignment for Nearby Laser Elastography

Ben Anderson

November 20, 2022

Contents

1 Introduction

The focus of my work for the better part of the last year has been studying how to use a UR3e Cobot for the purposes of focusing and stabilizing a sensor at its focal length away from a target surface. This has involved a thorough examination of the API and programming tools that accompany the Universal Robots ecosystem, as well as creating systems for synchronizing the robotic control system with signal processing and sensor systems.

In doing so a number of systems are being used in ways for which they were not designed. The consequence of this is that my codebase resembles a kludgy pile of half-fixes and temporary-solutions that, technically, have achieved the objective, at the expense of stability and ease of use. In recent months more efforts have been made to improve the readability of the codebase and allow for easier changes later on, but the work is ongoing and the project cannot wait for me to complete this work.

The hope is with this documentation, others in the OCE project can gain enough of an understanding of how the system works to use it in my absence as well as modify or extend the platform I've built.

Due to the kludgy nature of the platform, it may not be suitable to read this guide linearly. The introduction section should hopefully give you an overview of how the components of this section communicate and interface with each other, and a more thorough description of the workings of each module are in the following chapters.

If your only goal is to get started running the robot, strictly speaking you need only read the startup section (section 2), though I recommend also reading subsection 4.1, which covers use of the teach pendant.

If you need to

2 Startup guide

Pending the rest of the summary, the following procedure should be all you need to start running this control system on your machine.

2.1 First installation

These steps only need to be ran once when initially setting up your Cobot, and don't need to be repeated on subsequent use.

1. Install all software dependencies on the computer.
    - Python 3.8+ on target machine
    - Packages required for all of the python code can be installed by running
      `pip install -r requirements.txt`
      from the main directory of this repository
    - MATLAB/LabView/whatever other signal processing code you need for dealing with sensor input
2. Connect the UR3e robot to your machine via an ethernet cable.
3. Assign the ethernet connection to your robot a static IP address with the following criteria:
    - **IPv4**
    - **IP address:** `192.168.0.5` (Corresponds to addresses hard-coded into the running programs)
    - **Subnet prefix length:** `255.255.255.0`
    - **Gateway:** `192.168.0.1`
    - **Preferred DNS:** `10.1.2.1`
    - **Alternate DNS:** `8.8.8.8` (Less critical, any valid DNS will do)
4. Do the same on the robot, instead assigning the following values:

- **IPv4**
- **IP address:** `192.168.0.10`
- **Subnet prefix length:** `255.255.255.0`
- **Gateway:** `192.168.0.1`
- Set the DNS settings to any valid DNS addresses.
- Details on modifying network settings an be found on page 108 of the Polyscope manual

5. Write/save/open a program on the UR3e that does the following:

```
BeforeStart:
    script(before_start.script)
Robot Program:
    script(robot_program.script)
```

   - The corresponding scripts are found in the `\Code_for_the_robot` folder in this repository.
   - For more information on node programming, check out the chapter on programming in the Poylscope manual

6. Make sure the robot has the correct installation selected. This includes making sure the dimensions of the tool match the current setup of the sensor on the end effector of the robot. For more information on configuring the tool, read the section on TCP configuration in the Polyscole manual

   - This will require some knowledge of the shape/orientation of the sensor you are currently working with.
   - When working with sensors that have a focal point offset from the end of the tool, the TCP offset should be defined at the location of the focal point, not the edge of the tool.

This setup will only need to be run when a part of your installation changes. Steps one through three need to be repeated whenever you run the installation on a new computer; step four only needs to be run once for each robot (or whenever the robot goes through a hard-reset). Steps five and six shouldn't *need* to be repeated regularly, but sometimes need to be rerun if the robot isn't shut down correctly. (This is very easy to do).

2.2 Starting up the robot

Once the above configuration steps have be completed, you should be ready to run experiments with the robot after following this procedure:

1. Power on the robot by following the boot-up procedure described in subsubsection 4.1.1 (also power on your computer, obviously)

   Warning: The robot will move slightly when it boots up as it unlocks its brakes. Keep sensitive equipment clear from the robot, and do not power off the robot in a position that will put nearby objects at risk when it is turned on.

2. (Optional) Use the teach pendant to manuever the robot into a position convenient for the following test. This is optional as the control loop allows you to toggle freedirve mode, but it may be more convenient to do so here.

3. Make sure the correcet program and installation are selected on the teach pendant; To read more about navigating the program/installation menus, read the section on this subject in the Polyscope Manual

4. Do any setup necessary for your sensor now; the robot can be left idle while you do so.

5. Once your setup is ready, open the `COBOT_Transducer_Control_Code` directory on your PC, and double-click `server.py`

    - A blank terminal window should pop up with a blinking cursor; (You do not need to interact with the terminal window, it is only there for status messages)
    - This activates the socket server that the system uses to communicate between modules.

6. Tap the 'play' button at the bottom of the teach pendant to run the program; The terminal window from the previous step shold show a new connection has been established.

7. Start your signal processing script; make sure your signal processing program follows the appropriate communication protocol (TODO elaborate)

8. Finally, run `TransducerHoming.py`. This can be done by double-clicking the file in file explorer, but it is better to run it from an IDE such as Visual Studio; the software is not stable and being able to see an error message when it breaks is very helpful.

    - This should open a terminal window that looks like the following:

```
TCP position in relation to its initial position:
        ([[-1000.00 -1000.00 -1000.00]](mm),[[-57.30 -57.30 -57.30]](deg))
=========================
TCP position in base: ([[ 0.00  0.00  0.00]], [[ 0.00  0.00  0.00]]
Current joint position in degrees: ([[ 0.00  0.00  0.00  0.00  0.00  0.00]])
Recent refresh rate: 0.1465371191501617415
------------------------
Latest mag:48
|||||                                                                    |
Freedrive active: False
Press (f) to toggle (f)reedrive mode :)

Press (d)emo to demonstrate the basic pathrouting module
Press (k) to trigger a full scan with hard-coded resolution.
Press (g) to trigger a amplitude max-finding pathrouter.
Press (t) to trigger an alternate amplitude max-finding pathrouter.
Press (q) to quit

Sun Nov 20 18:23:36 2022
----------------
```

From here you are ready to run your test. Follow the directions on screen to enable/disable freedrive and activate the appropriate pathfinding module. The two menu options I suggest using are '(k) for fullscan' and '(g) for maxfinding'. The fullscan module runs a full scan of a search volume of a hard-coded shape and resolution, and the maxfinding module runs the current-best module for focusing a transducer.

2.3 Common debugging steps

Because the sole author of 90% of this code is such a genius, occasionally this software breaks in ways that may be frustrating. Here are common problems and the best-known steps to address them (short of writing better sofware)

- The robot collides with something during a test and locks itself up
    1. If this happens the experiment is ruined, you will need to restart
    2. Close the `TransducerHoming.py` script, and stop the socket server.
    3. The robot will need to be unlocked through the main menu, after you have done so tap the 'stop' button to cancel the running program (the default behavior is to pause the program, you don't want that)
- `TransducerHoming.py` crashes
    - This is a frequent occurance whenever major changes are made to the program or installation
    - It is worth noting though, that `TransducerHoming.py` is not a dependency for the other components; they will simply sto pmoving. You are able to restart the script without resetting any of the other systems.
    - If you ran the script through an IDE, take note of the error message and where it occured, and alert Ben Anderson to the problem (bpa13@uw.edu) or submit a pull request on github

3 System Overview

The system is designed to keep as much modularity as possible and to isolate responsibilities for particular tasks.

The sensor is a generic device that requires focusing and produces an output that is read by the signal processing package within the computer. The Stabilization and Transducer Alignment for Nearby Laser Elastography system (hencforth referred to as STANLEy) consists of a UR3e cobot from Unviersal Robots, and a script written in using the Universal Robots proprietary scripting language. (Attempts are being made to reverse-engineer some of this languages functions to cut this box out of the control diagram, but the process will be very involved and is being put off in order to meet deadlines).

Physically, this system includes a robotic arm manipulator, a control box, and a teaching pendant. STANLEy interfaces with the sensor through a physical linkage at the end of the robotic manipulator, and with the PC through an ethernet cable.

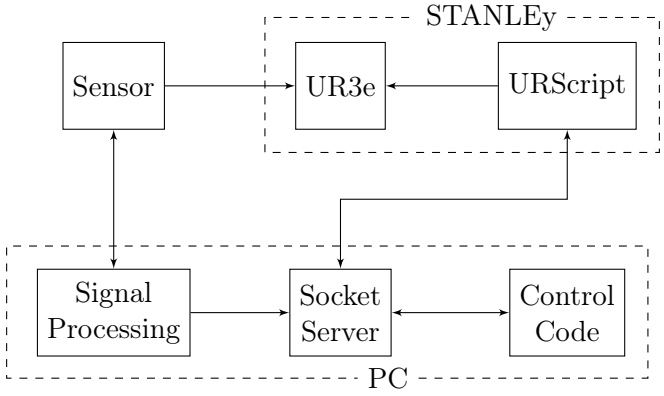


Figure 1: Component overview of the STANLEy system

The remaining control blocks occur within a nearby computer. The signal processing block refers to a program (typically in MATLAB or LabView) which recieves the data from the sensor and processes it for further use.

For the purposes of the control loop that focuses the sensor, the signal processing block must reduce the sensor input, no matter how complex, to a simple magnitude that should be maximized to bring the sensor into alignment. More on this to be added later.

An internal socket server is spun up in a simple python program. This is a program that listens on a hard-coded IP-address and port within the computer for input sent from the signal processing block, the control code, and to the robot through the control box. More on the specific IP addresses in the initial setup section of the startup section. (section 2)

The control code is run from a script named `TransducerHoming.py`, and associated libraries. For now, directly editing this file is the main interface with which we can modify how the homing sequence runs. In future I hope to expand the user interface as well as the command-line options to allow more on-the-fly modification of program parameters, but the duration off tests and the speed at which changes are made have made this an inefficient goal for some time.

Overall the system utilizes three discrete devices and at least three programming languages. I (Ben Anderson) am most directly involved with the programming of the control code and URScript. The top-down overview of each components' functions are pretty simple, but debugging/modifying them will require a rough understanding of the URScript API and a strong understanding of Python.

4 STANLEy and the URScript API

The body of the novelty surrounding this project centers around the use of a UR3e robotic arm to position and hold the OCE sensor.

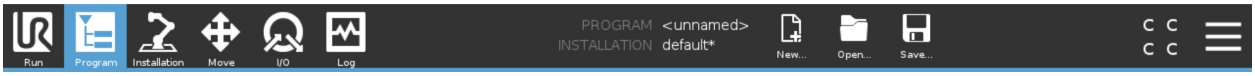
The UR3e is manufactured by Universal Robots and is primarily designed to automate human labor. Shipped with the robotic arm is a control box and a teach pendant, a hand-held tablet for manually controlling the robot. The control box allows one to write and execute scripts using the proprietary URScript API, a set of functions built into the robot that handle common tasks related to positioning and animating the arm.

Additionally, the control box has an ethernet port supporting the secondary and Real Time Data Exchange interfaces. Both of these have their uses and can be read about in the remote control interface guide, however they weren't adequate for the method ultimately used.

4.1 The Teach Pendant and the PolyScope interface

The teach pendant is the default accesory for interacting with the internals of the robot. It is shipped with the robot along with the control box.

The pendant, when active, is running an interface known as PolyScope, the GUI used for operation of all UR robots. Along the top of the page, there are six different menus and a file explorer interface that we will refer to later:



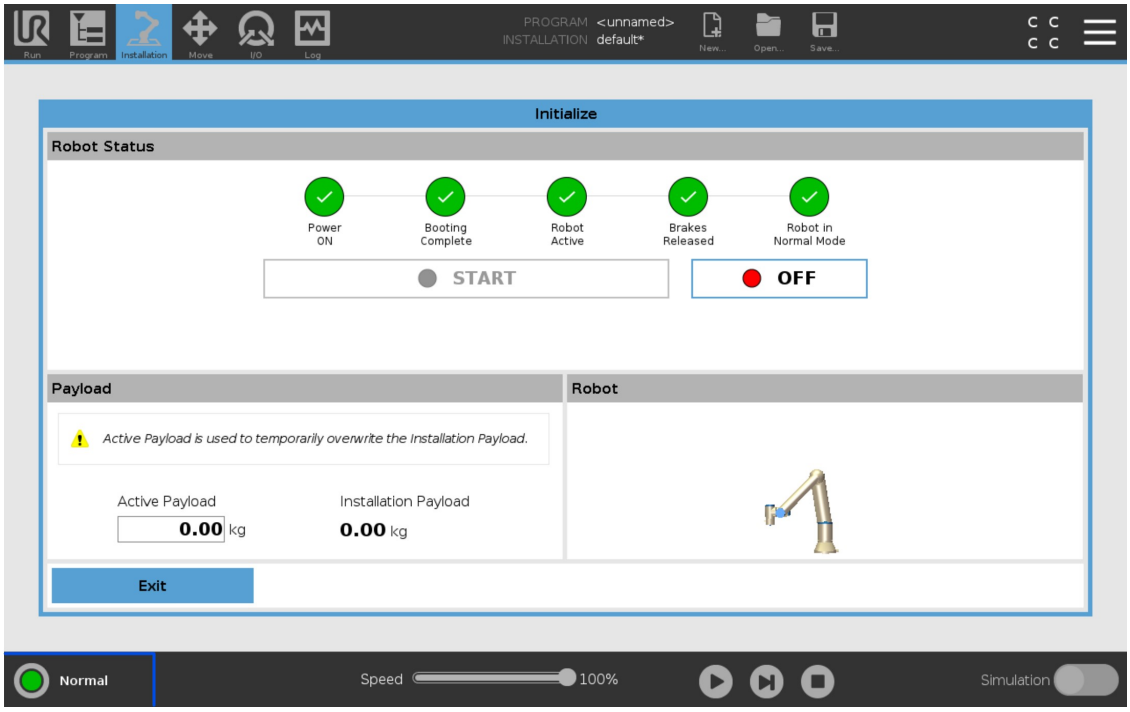
The full manual for PolyScope can be read here, but this guide will cover the elements necessary for operation of STANLEy.

4.1.1 Initialization and setup

In order to use the robot, power on the tool box by pressing the physical power button above the screen and next to the e-stop on the teach pendant. (It can be finicky how long a press is necessary, so if you suspect you're holding the button down too long listen for a click from the control box to indicate it has been powered on. You should release the button immediately after hearing this click).

The power button should light up green and the screen should show a loading screen as the robot takes a minute to boot up.

Upon fully booting up, the robot itself still remains dormant. In the bottom-left of the screen is the robot status indicator, shown here:



The color inside the circle indicates the state of the robot:

- *Red*: The robot is powered off.
- *Blue*: The robot is in freedrive mode.
- *Yellow*: The robot is idle; powered on but not ready for operation.
- *Green*: Powered on and ready for normal operation.

After booting up the control box you enter the Initialization screen (shown in the image above). When powering up the robot from full-shutdown the central button will be active and you will press it twice to enable the robot; first it will say 'ON' and you press it to deliver power to the robot, then it will say 'START' and you will press it to unlock the brakes and enable the robot completely.

Warning: The robot will move slightly as the brakes are unlocked, do not leave the robot in a stat where this motion will put nearby equipment at risk.

4.1.2 Program tab

The program tab is one of the options in the menu tab. Here is where you write/modify programs for controling the robot. Polyscope has a node-based programming langauge similar to scratch, in addition to a powerful scripting language that can be baked into a robot program. When you press

the 'play' button at the bottom of the tablet, the program it runs will be in this form (if one is loaded).

Stanely (The UR3e currently used in our lab) has a program saved called `test.urp` that works with our control code. It can be opened by tapping the 'open' button in the top banner and tapping 'Program' from the subsequent dropdown. Our program relies heavily on the scripting language, so the only nodes in our program are a 'BeforeStart' and 'Run Program' node, each containing a script that does everything we need.

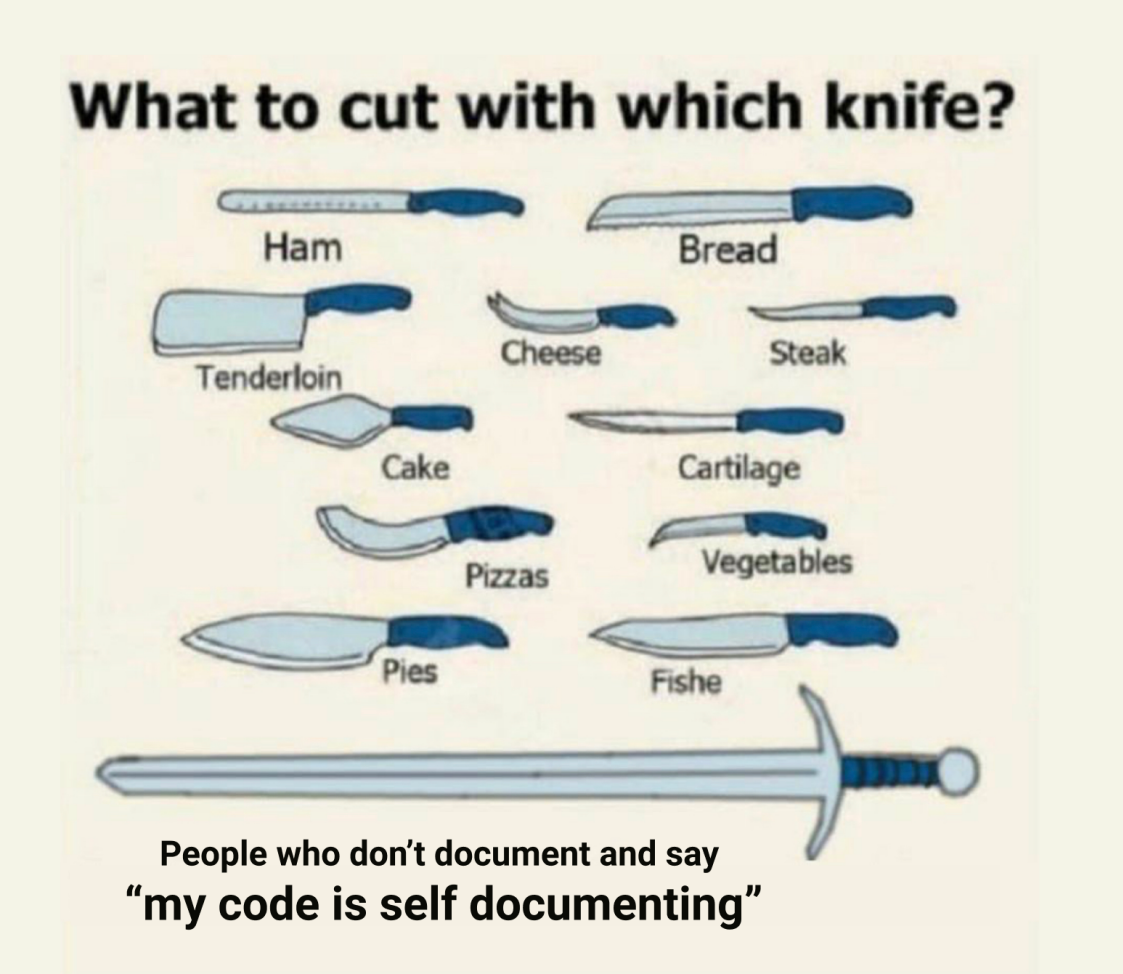
> **Note**: If you want to edit either of these programs, *use a USB stick and a computer to transfer/edit the program.* The text input for the teach pendant is a finicky touch screen and I've had trouble getting a keyboard with the right language settings.

4.2 URScript API

URScript is the propreitary scripting language created by Universal Robots and runs natively on all of their machines. The functions are tailored to work for their machines specifically, and I haven't found a way to run URScript outside of the control box.

5 Control code

Working on it



5.1 `TransducerHoming.py`

476 lines of nonsense and yes I wrote the whole thing myself. No I am not fine. More extensive documentation is pending but for now we need only refer to the data formatting standard used to pass data between the sensor and the control code: subsubsection 5.1.1

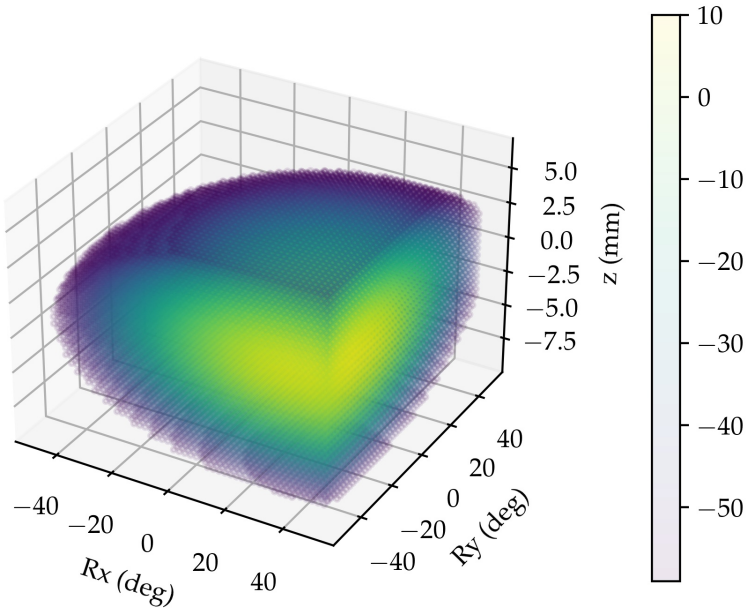
STANLEy -> Stabilization and Transducer Alignment for Nearby Laser Elastography

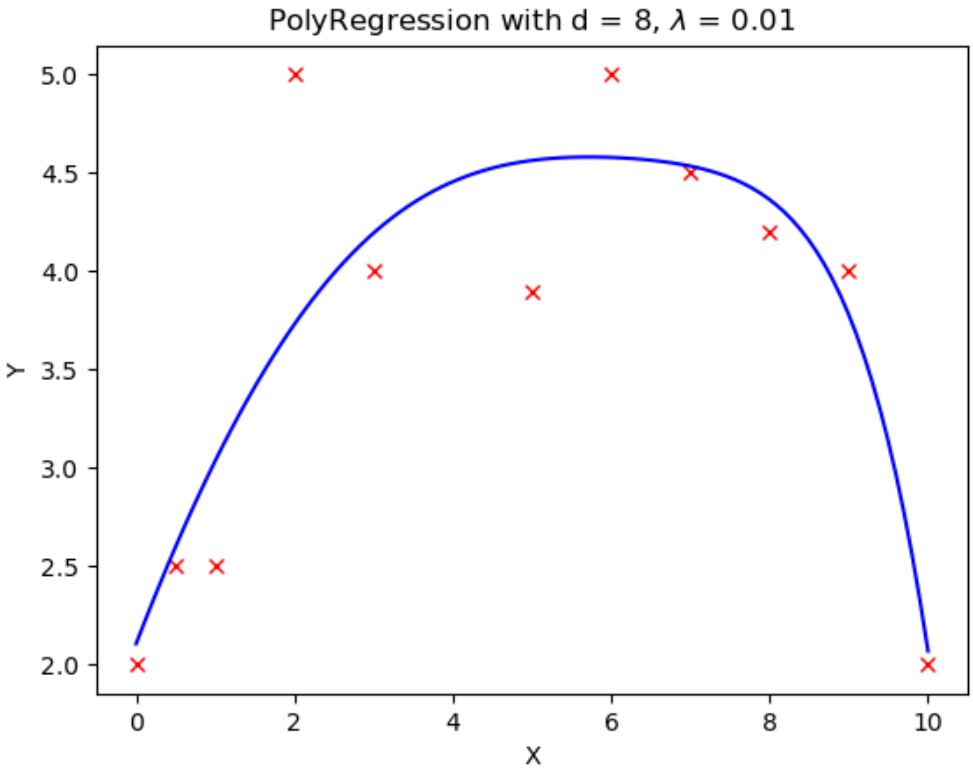
5.1.1 Interfacing with Sensor output

T	1	3	Z	0	0	1	6	4	5	1	0	2

Timing block Data block

Yeah looks like that.


5.0
2.5
0.0
−2.5
−5.0
−7.5
z (mm)
10
0
−10
−20
−30
−40
−50
−40
−20
0
20
40
Rx (deg)
40
20
0
−20
−40
Ry (deg)


PolyRegression with d = 8, λ = 0.01
5.0
4.5
4.0
3.5
3.0
2.5
2.0
Y
0
2
4
6
8
10
X