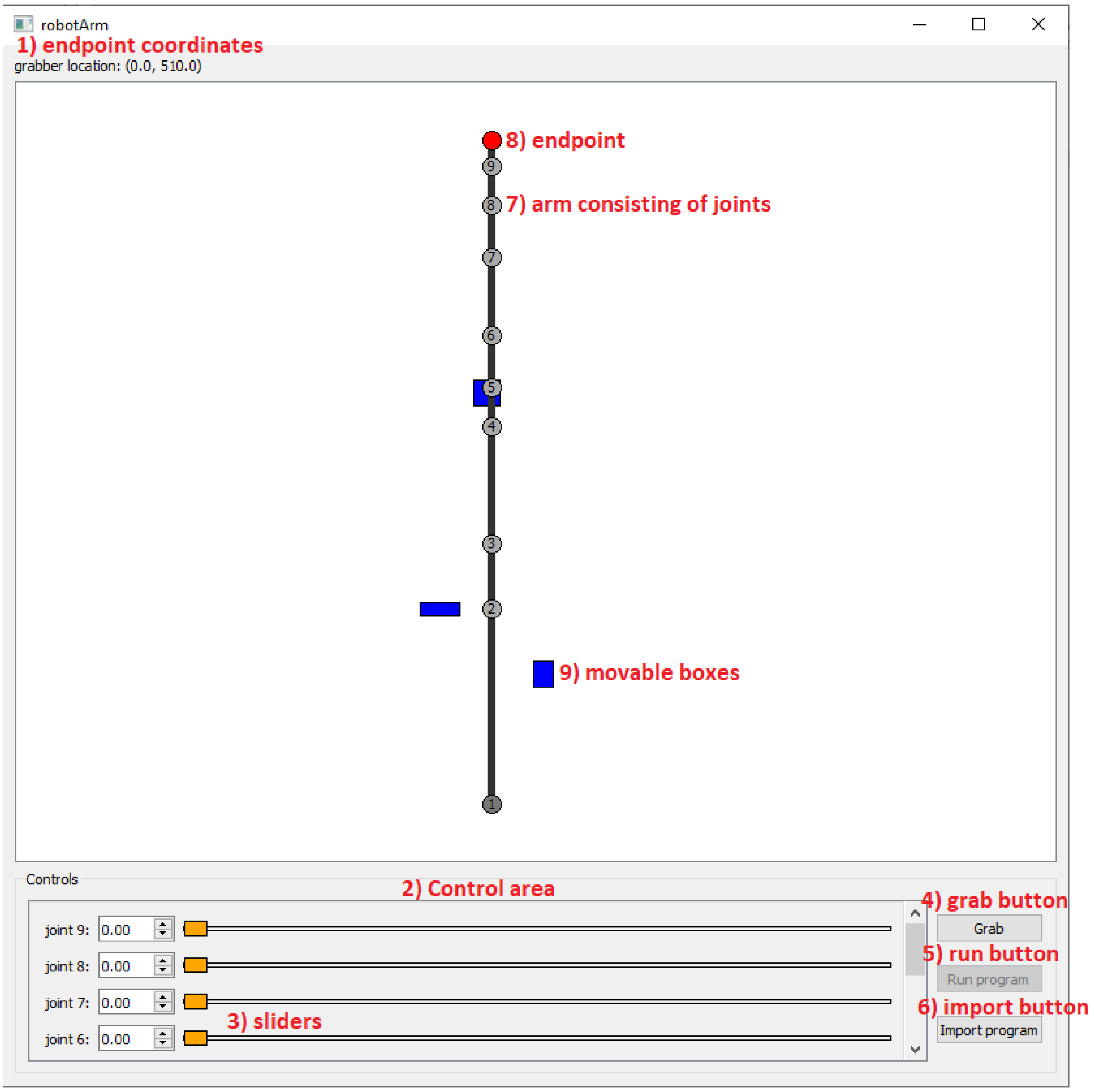
CS-A1121 Documentation

Linus Sundström, 432021, Automaatio- ja Informaatioteknologia, Dated 4.5.2020

General description

The aim of the project was to create a simulation of a 2-dimensional robot arm. This has been completed such that it satisfies all the criteria for a medium hard assignment. The software has the following features.

- Graphically show a 2-dimensional robot arm.
- The arm supports an arbitrary number of joints.
- Arm movement is animated. Joint angles can be altered using sliders or number boxes.
- The robot arm is be able to grab and move objects.
- The arm can read and execute command files, saved in a csv format. Using these the user may define a set of joint angle changes or grab / release commands to occur at specified timestamps.
- The control actions are performed asynchronously, allowing changes to multiple joints at once.



grabber location: (0.0, 510.0)

joint 9: 0.00

joint 8: 0.00

joint 7: 0.00

joint 6: 0.00

Figure 1. Example UI

User Guide

Launching the program

To run the program, execute main.py and provide command line arguments corresponding to the wanted joint arm lengths. The example arm in figure 1 for example was created by running "*python main.py 150 50 90 30 40 60 40 30 20*". Note that the lengths may be floats, for example 23.788.

UI elements and operation

The UI contains the following elements, as seen in figure 1:

1) Current endpoint coordinates. Origo (0.00, 0.00) is located at joint number 1.
2) Control area containing joint sliders. It automatically expands to contain the necessary number of sliders for each joint, irrespective of the number of joints the program is launched with.
3) There is one joint slider corresponding to each joint, that may be used to control its angle. Alternatively, the angle may be inputted directly in the number box. The joints have a maximum rotation speed, and may as such take some time to reach the desired angle. The number boxes show the set angle, not the current angle.
4) The grab button is used to grab a box the endpoint is overlapping with. If no box is overlapped, nothing happens. If multiple boxes are overlapped, only one will be picked up. If a box is successfully held the endpoint will turn green and remain so until it is released using the same button.
5) A program imported using the import button may be run using this. The button is only active after a valid program has been loaded. During the execution of a program, it may be stopped by hitting it again.
6) Opens a dialog to select a csv file containing a program to be run. For details, please see the section "Files".
7) The arm. Each joint has a number in it corresponding to the slider that can be used to change its angle.
8) The arm endpoint. It is red while not holding a box and green while holding one.
9) Boxes that may be moved by the arm.

External libraries

The project uses the following libraries:

- Math – basic math operations such as sin and square root.
- Threading – certain parts (such as joint angle changes) use multithreading.
- Time – Used to limit the joint update speed.
- Sys – used for clean exit.
- PQt5 – All of the UI.
- Unittest – for unit testing

Software structure

The class structure can be perceived from the UML(ish) diagram in Figure 2. The most important functions of each class are also listed.

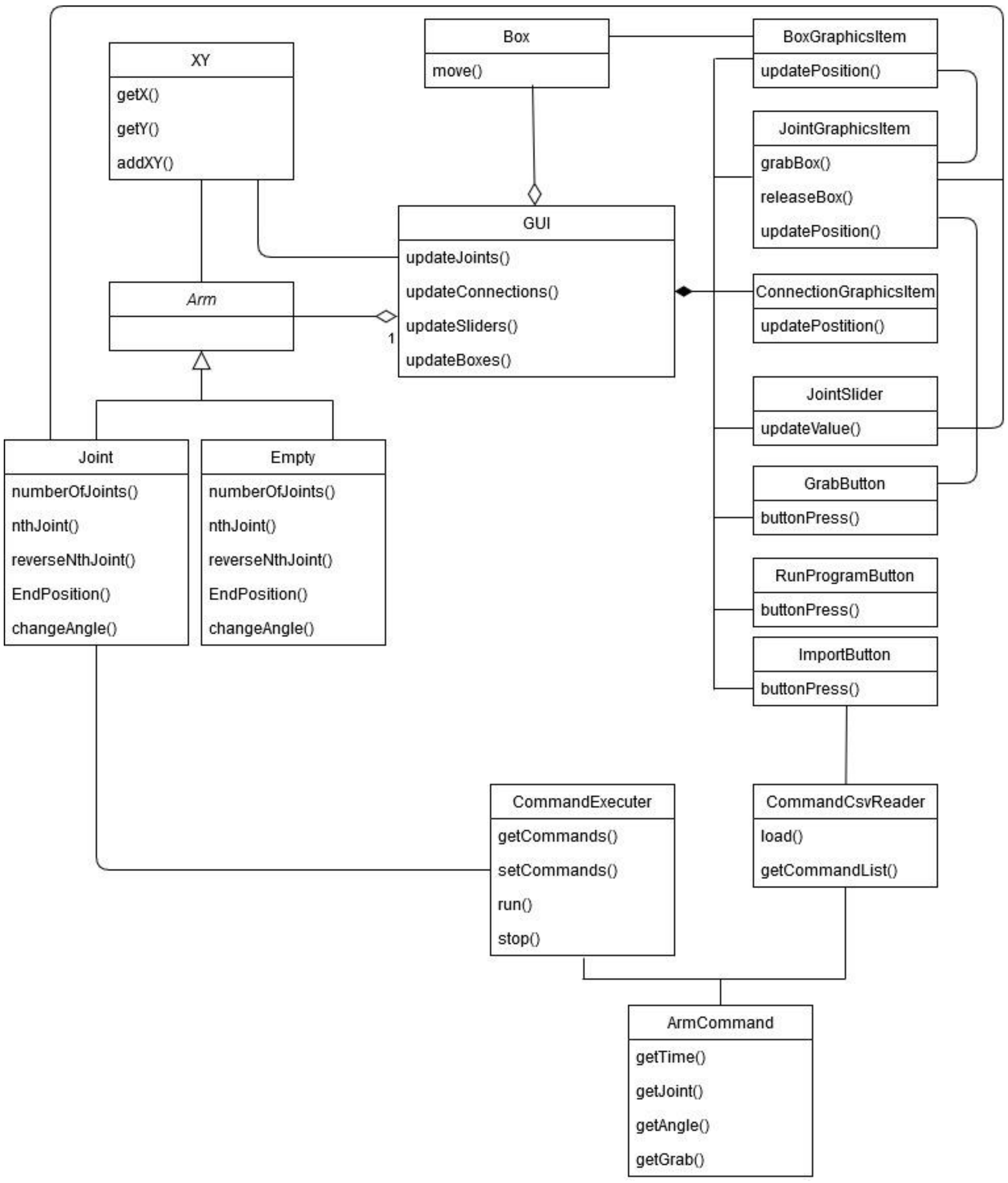


Figure 2. Diagram of class structure

The arm and boxes have matching graphical representations, BoxgraphicsItem, JointGraphicsitem and ConnectionGraphicsItem. The GUI regularly calls for updates to these to make sure that their position matches that of the underlying object. The gui itself is heavily derived from the robots assignment earlier this course.

Currently, grabbing of objects is handled purely through UI classes. I'm not quite happy with this implementation and would probably revise that if I had more time to spend on the project.

The classes CommandCsvReader, CommandExecuter and ArmCommand work together to import command files. This is perhaps the part of the project I am the most happy with. ArmCommand defines one single command with all data needed to execute it. CommandCsvReader reads a file to a list of ArmCommands, and the executer then finally executes the joint changes and grab commands at the specified timestamps. One small weakness however is that if one attempts to load an excessive amount of commands to be executed at the same time, execution may not be as expected. This is due to possibly running out of available threads. In my testing however I found this not to be a limitation for practical use, but rather a strange edge case. One improvement one could make would be to limit the number of simultaneous operations accepted in the import file. A separate possible improvement would be to add some text box giving the user information about the current program being run.

The arm itself is a linked list consisting of individual joints. Each joint runs an update loop where its actual angle approaches its setpoint. This way the angle of the joint does not just snap to its new setting, but gradually gets there. An improvement on this system would be to simulate the slowness of the system and use that for slowing the arm movement.

One clear flaw in the joint angle operations is the lack of an emergency stop. Whatever the operator does, a joint will always move towards its set angle. If this was a real world application, there would certainly need to be a feature by which one could instantly stop the arm at its current location.

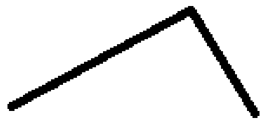
Algorithms

Linked list algorithms

The arm is modelled as a linked list. The main reason for this is that it allows for some sneaky methods when calculating arm positions and angles using recursive method calls.

The method getTrueAngle() takes the angle stored in the joint, and then recursively adds the angle of all previous joints to it. The result will be the angle in relation to the global grid.

Example: an arm consists of two joints. The first (inner) joint is set to 60^o, the second is set to 90^o. The arm will then look like this:



By adding the first angle to the second angle, we end up at 150^o, which is the angle of joint 2 in relation to the grid. This works for an arbitrary number of joints.

Using similar logic, calculating the position of each joint will be done by calculating its x and y coordinates in relation to (0,0) using its angle in relation to the grid and Pythagoras theorem, and then adding this position to the position of the previous joint.

Simple recursive strategies are also used for finding the nth joint and reverse nth joint.

Data Structures

This is not an efficient way to implement a robot arm. Preferably, arm lengths and joint configurations would be stored in matrix form allowing for much more efficient calculations. Linked list is also a very inefficient data structure in python. For this project however I still chose to use the linked list approach mainly out of personal curiosity. I have never made any actual project using the data structure and I found the challenge interesting, and somewhat amusing. It also lent itself well to my goal of allowing for an arbitrary number of joints in the arm.

A linked list is a data structure where each element of the list contains a link to the previous element in the list. The data structure is not the most efficient in python, as the list objects end up nested in each other, and as such should not be used for any kind of large data. For this application I deem it acceptable however as I do not foresee anyone wanting a robotic arm with hundreds or thousands of joints.

The arm is mutable, as is evident by the ability to change joint angles. It would also be possible to change joint lengths during execution, although no UI feature allowing for this has been implemented, as this did not seem realistic from a use case point of view.

Files

The software can import a csv file containing an arbitrary number of commands. Each command should be written to a new row in the following form:

float timestamp; int jointNumber; float angle; bool grab

timestamp	Time (in seconds) after start of program execution that the change should be made
jointNumber	The joint to apply change to. The arm must be long enough to contain the joint.
angle	New angle (in degrees)
grab	If set, the arm will attempt to grab / release. It is set to false if the field is empty, or set to one of the following: 0, "False", "false", "No", "no", "FALSE". Otherwise True

```
#this is a comment
3;1;300
0;2;120
3;3;300
2;6;0;grab
#this is also a comment

3;4;60


3;5;20
```

Figure 3.Valid command file

It is also possible to insert comments into the file by prefacing the line with #. The reader will ignore empty lines.

Example of a valid command file when operating an arm with 6 joints can be seen in Figure 3. It can be copied directly to notepad for example for testing. Please note that the import feature only discovers files of type *.csv .

There is some clunkyness in that for grabbing one still needs to define a joint and angle. A better solution would be to have a separate command not in csv format. It might also be helpful if the user was able to record a sequence and store it in a file, currently the files must be made manually.

Another issue is the fact that there is no indication to the user where there is a problem in the file if file import fails. This can make fixing errors tedious. Also, if one first successfully loads a working program, but then later attempts to load an erroneous program, there is no indication as to which one is currently loaded. Simply displaying the current filename might help with this.

For future features, I would consider adding a command to set the arm to some baseline state. A command to wait for the arm to reach its currently set configuration would also be helpful. Some sort of if else logic depending on whether a box was picked up or not might also prove useful.

Testing

I started this project with a plan of doing test driven development. In the early stages this worked well. It quickly became apparent however that unit testing the UI was not going to be possible. As such a large part of the project lacks unit tests. I also didn't keep testing in mind when creating the file importer, which made unit testing it close to impossible. As such test coverage is nowhere near close to where I would like it to be.

Known flaws

1) The file import issue detailed under files
2) If very large joint lengths are defined, seeing the arm endpoint becomes very tricky. This could possibly be alleviated by making the view scalable. While there are scrollbars these are tricky to use with a moving arm. There should perhaps also be an option to centre the view on the endpoint.
3) Performance suffers greatly with a very large number of joints (50+). I should probably have implemented some sort of cap, but then the theoretical maximum will depend a lot on the running environment.
4) When picking up boxes, they snap to the centre of the endpoint. I considered fixing this by adding the offset to the boxes position while it is grabbed, but finally decided against it as a real world grabber also probably would end up pulling whatever it is its picking up towards the middle. Some animation of the box being pulled in might be nice though.
5) If the arm has a lot of joints, the control box does not display enough sliders at a time to not feel clunky. This could be fixed by having the controls at the side of the view instead of below.
6) There are no tick marks on the sliders, making accurate use very hard. I investigated this a bit but getting tick marks to a custom stylesheet is apparently quite a big undertaking. As such I decided not to invest the time.
7) There is currently no way for the user to add or remove boxes outside of modifying main.py. One could add an interface for it but it didn't see it as that necessary.
8) If multiple boxes overlap with the endpoint, the arm will pick up the one whose graphics object was created later. Adding collision detection and not allowing boxes to overlap would solve this.
9) Boxes currently do not rotate. The algorithm for solving this would not be trivial, however. I am not quite sure what that algorithm would look like, but it would be recursive.
10) Currently when running an imported program, there is no way to lock the controls, meaning the user may mess up the execution willingly or by accident. Having the option to disable controls in a program file would solve this.

3 strong points

1) The file reader and command execution
    a. Loading of programs work well, and the implementation feels clear (at least to me)
2) The linked list arm at the core of the program
    a. I'm very happy with how well a linked list worked for implementing the arm, and had fun writing the recursive functions making it possible.
3) The expandable controls
    a. A necessity for the variable arm length, I am very happy with how the list of sliders expands automatically and still makes it clear what part of the arm each slider operates.

3 weak points

1) The implementation of box grabbing in guiControls. I really do not like how it's all done in the UI, I would prefer to have more clear separation between UI and backend.
2) The main UI class (GUI) is hard to understand and tricky to expand. Its maybe the one part of the software I still do not feel I fully grasp. It could do with some heavy refactoring.
3) The angleUpdater method in Joint. While it seems to work well, splitting each joint to individual threads does seem to be the bottleneck when moving above 50 joints. It should probably be reworked so that a single thread updates more than one joint.

Deviations from the original plan

I planned to develop this project using test driven development. That did not happen in the end due to such a large part of the project being about PyQt. Apart from that however The project turned out surprisingly close to as planned.

Realized Timetable

I planned to complete the project in 4-5 iterations as outlined on page 3 in the project plan. I also set up the following timetable:

- Increment 1: 17.3
- Increment 2: 31.3
- Increment 3: 14.4
- Increment 4: 28.4
- Increment 5: any remaining time

In the end I stuck close to that. I ended up combining increment 1 and 2 due to the fact that separating them did not really prove any benefit or tangible simplification. I finished Increment 2 pretty close to schedule. I created a separate branch in GitLab for that state which has its last update on 3.4, which was some final bugfixes.

I then quickly finished increment 3 (i.e. movable boxes) on 6.4, and then moved on to finalizing the UI. The import functionality and final bug fixes was implemented 20.4. Bar some minor cleanup, that is also the state the project is in now, as I had new and rather time consuming courses starting.

So in summary, I finished Increment 4 with a week to spare, but then unfortunately didn't find the extra time to implement any additional features. For a full progress rundown I suggest checking the GitLab commits, I have done my best to make the commit messages at least somewhat helpful.

Final review

In the end, I feel the end result may not have that many features, but what it does it does rather well. As such I am happy with the project. Adding classes and functionality shouldn't be to difficult either. The one stop to that would probably be the GUI class, which would need some major refactoring. I would however once again like to reiterate that a linked list should not be used as a basis for a project like this, there are much more efficient solutions available.

The csv file import is not really pure csv, which may make things tricky for certain editors. The file format should probably be changed to something more readable.

I think the biggest flaw in this project is the lack of testing. It could and should certainly be more extensive. Certain methods such as the csv importer should also be reworked to make testing easier. Thankfully however, there are not that many strange states in the project.

Sources

Course material in A+:
https://plus.cs.aalto.fi/y2/2020/

Python.org pyQT tutorials
https://wiki.python.org/moin/PyQt/Tutorials

Python standard library documentation
https://docs.python.org/3/library/

All the heroes on Stack Exchange
https://stackexchange.com/

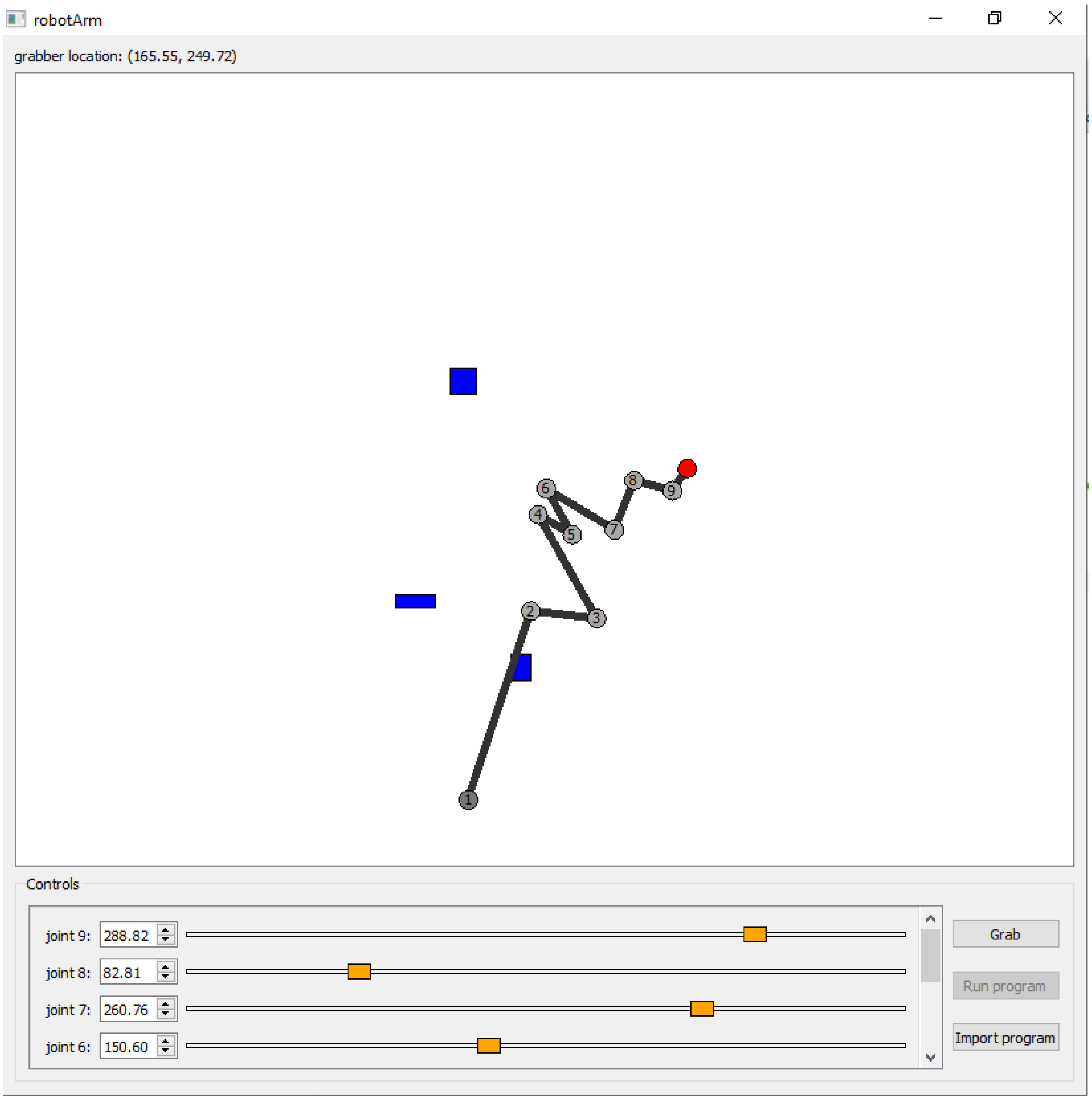
TutorialsPoint
https://www.tutorialspoint.com/pyqt/index.htm

Qt Documentation
https://doc.qt.io/qt-5/layout.html

Demo import files

There is an example import project in /src called armCommandsDemo.csv that may be imported and tested. There is also a file called armCommandsError.csv that demonstrates the handling off erroneous files. The arm layout used for the demo can be found in the file as a comment.

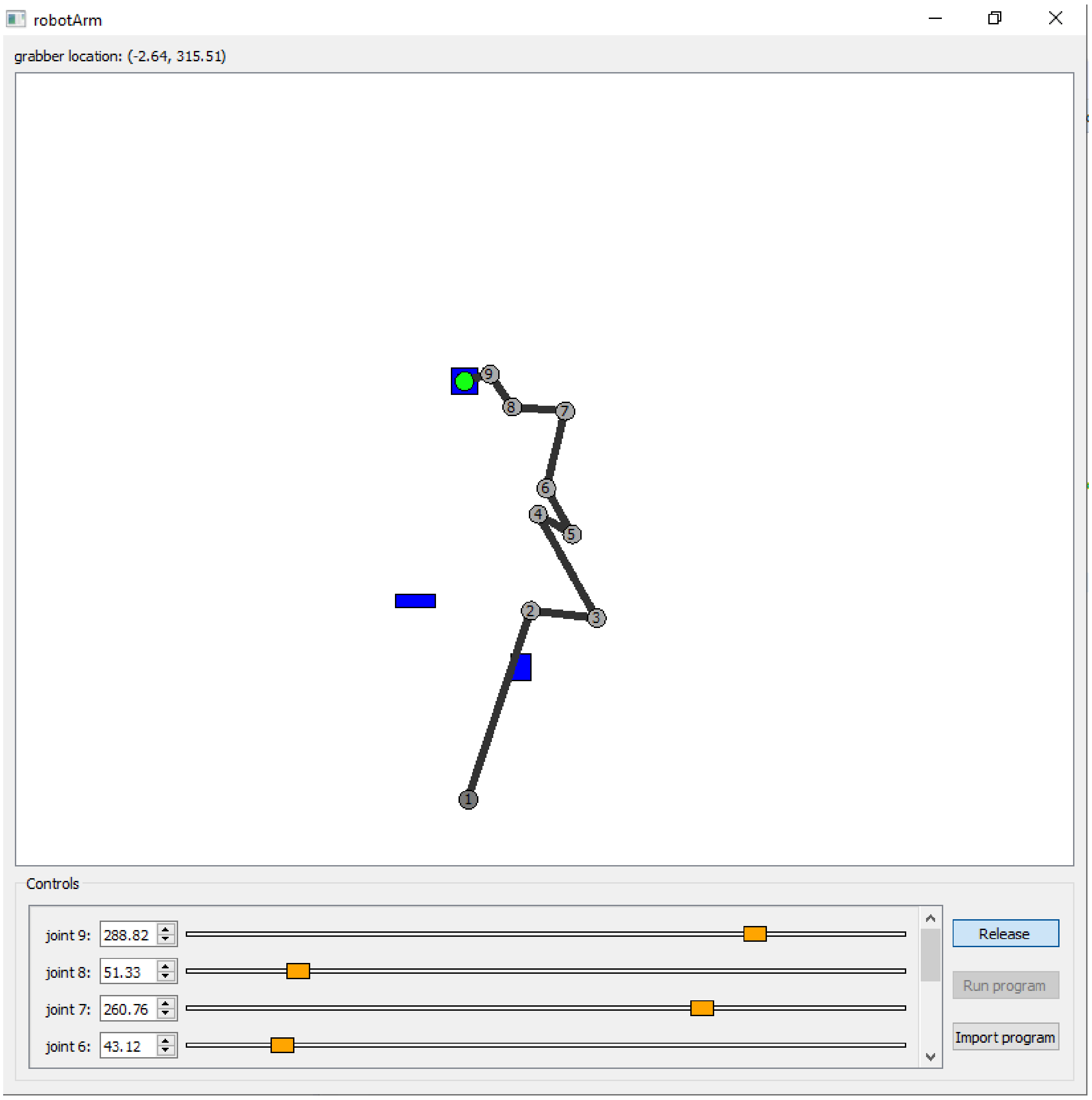
Attachments



grabber location: (165.55, 249.72)

joint 9: 288.82
joint 8: 82.81
joint 7: 260.76
joint 6: 150.60

Figure 4. Arm with a arbitrary joint angle values



grabber location: (-2.64, 315.51)

joint 9: 288.82
joint 8: 51.33
joint 7: 260.76
joint 6: 43.12

Figure 5. Arm holding box