Automation Tutorial 1: Algorithms and Formulas

Marco Selvi

29th August 2013

Aims

The main aim of the *Automation* set of tutorials is to explain how to implement automation in the **microscope project** of the **OpenLabTools** initiative.
In particular, here in *Tutorial 1* we will present the concepts, algorithms and formulas used in the auto-focusing program available in the OpenLabTools GitHub repository. We will then give a brief overview of other possibilities within this project.

Concepts

We now make a brief overview of the fundamental concepts necessary for the correct operation of the microscope. Here we will maintain a rather abstract approach. For a more detailed source code description please refer to the *Automation Tutorial 2.*

Serial Ports

One of the fundamental things to know in order to implement full automation through C code is **serial communication** with the Arduino board used (for the Arduino tutorials check here). A simple USB connection between the Raspberry Pi and the Arduino board is sufficient to provide serial connection. The arduino is generally associated with the device **ttyUSB0**, unless another device is already associated with it. To check, simply type

```
ls /dev
```

in a terminal window. Once you know what device to connect to, you can test your serial communication with Python by simply using

```
import serial
ser = serial.Serial('name_of_device')
```

Now you can freely communicate to the Arduino by writing into the serial port, and reading outputs from it. Many ways to do so exist, depending on the libraries and the programming languages used. We will be programming in C++ and using the *Boost Asio* library whose description can be found here. Details of how this is done are in the *Automation Tutorial 2* document.

Image Processing

Another rather obviously important part of our automation routines is image processing. Again many options exist, depending on the language chosen. The Python PIL (Python Imaging Library) for example is a very versatile if a little slow imaging library that would serve anyone's purpose. Here however we will be using the CImg Library, optimized for C++ and very portable.

Image processing is the core of any program that exploits the full functionality of our microscope. Depending on what we wish to do, it can be a very complex process and of non-trivial implementation. It can also be highly CPU-demanding, especially with high-quality images.
The Raspberry Pi may be a fantastic computer for real-time control and development of projects, and it does indeed provide the perfect platform for the electronics of our microscope. However, due to its limited processing power, it is not the ideal machine for heavy-duty tasks like image processing of large pictures. For these reasons we aim to use a reasonably efficient library, and we also tried to keep the resolution of the images rather low wherever possible.

We will go into more details later, but it should suffice to know for now that the library is used only for the most basic tasks of opening a *jpg* file and displaying images. All other image processing routines are custom built to save on computational time, with the aim of speeding up the program.

Motor Control and Serial Commands

Motor control is what concerns the mechanics of the microscope's stage. The job of actually controlling the motors and interpreting the commands sent through the serial port is done by the Arduino board, for which we suggest reading through the Arduino tutorials. For the mechanical structure of the microscope itself, read the Mechanics Tutorials. What we are interested in at the *automation* stage is what the mechanics can actually do: namely, for auto-focusing, the fact that we can control the vertical movement of the stage with commands sent to the Arduino through the serial port.
Other programs might need to make use of the horizontal stage control, also available through the serial commands. Other commands exist to control lighting, read vertical positions, and many more can be added to cover any functionality that future development may make necessary.

Algorithms and Formulas

In this section we will give a theoretical description of all the algorithms used in writing the various programs. Of particular interest are the algorithm for auto-focusing and the algorithm for edge detection used in a sample program on image processing.

Canny Edge Detection

The Canny Edge Detection is a rather advanced and functional edge detection algorithm for pictures. It is a perfect example of image processing implementation, and we will briefly explain here the algorithm on which it is based. For detailed reference to the code please see here.

This edge detection works in the following way:

Algorithm 1: Canny Edge Detection

```
Data: Image to be analysed, in our case a .jpg file
Result: Edges of said image
begin
    if Image not in grayscale then
        Turn image to grayscale

    Convolve image with smoothing matrix (to eliminate artifacts)
    Calculate gradient of intensity associated with each pixel
    Select edges:
    if gradient $<$ lower threshold then
        Turn pixel black
    if lower threshold $<$ gradient $<$ higher threshold then
        Turn pixel grey $\rightarrow$ minor edges
    if gradient $>$ higher threshold then
        Turn pixel white $\rightarrow$ major edges

    Connect edges:
    if minor edge is connected to major edges then
        Turn this minor edge into major edge
    else
        Discard pixel

end
```

The algorithm is rather self explanatory, but I would suggest to read quickly through the commented code if you are interested in seeing how this algorithm is implemented, or refer to the *Automation Tutorial 2*.

Auto-Focusing

The auto-focusing routine is rather more complex, and separated in subroutines that implement the various parts of the program needed for the whole to work. We will here describe only the full and automanted focusing program, while the various subroutines are described in *Automation Tutorial 2*.

To understand the following algorithm, one must keep in mind that we are using a stepper motor for the vertical movement. Thus all references to numbers of steps refer to steps associated with the stepper. The initial and minimum number of steps has to be established depending on the objective in use, as with focal distances and focal depths decreasing we require an ever finer adjustment. Also the computation of the focusing value for a picture is done through this formula:

$$F = \frac{1}{WH\mu} \sum (I(x,y) - \mu)^2$$

Here μ is the average intensity of the picture calculated over all the pixels, and I is the intensity of pixel at position (x, y) in the image. W and H are width and height of the image (in pixels) respectively. This formula simply represents a standard deviation weighted by the mean, and it is the most effective focusing parameter according to this paper.

Algorithm 2: Focusing

```
begin
    Set number of steps to start with

    Execute sweep:
      Take ten images at equal spacings from very close to the objective
      Compute focusing value for each image
      Move to position of image with max value

    Execute tuning:
    while number of steps > minimum number of steps do
        Take picture
        Compute focusing value for picture (F_st)
        if Last movement was up then
            Move up
        else
            Move down

        Take picture
        Compute focusing value for picture (F_1)
        if F_1 > F_st then
            Stay there
            Repeat cycle
        else
            Move in opposite direction (by twice number of steps)
            Take picture
            Compute focusing value for picture(F_2)

            if F_2 > F_st then
                Stay there
                Halve number of steps
                Repeat cycle
            else
                Go back to central position
                Halve number of steps
                Repeat cycle

    Focus position reached within minimum number of steps limitation
end
```