Discussion and detail of system design and choices made

System Design

I wrote this script using Python 3.5.2, and OpenCv 3.1.0, to match the versions available on the DUDE PCs in the School. Operation of the script is done via key presses, and output of the program is both visual in the main window, as well as various images being saved to disk, in the img_out/ directory. The main window displays one file ID at a time, with the top row being the w1 channel, and the bottom being w2. These are labelled, and labelling can be toggled by pressing the l key. Full list of commands:

- n - next image
- b - previous image
- l - toggle labels
- s - save the currently displayed w1 and w2 processed images
- x - exit

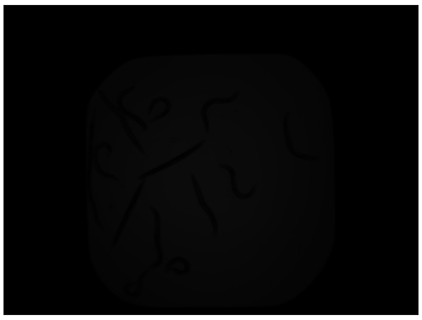
There are also boolean settings toward the top of the script, to allow for automatic saving of various images.
Processing of images is done in steps, with each step being a standalone method. All methods are documented and commented. The processing steps are as follows:

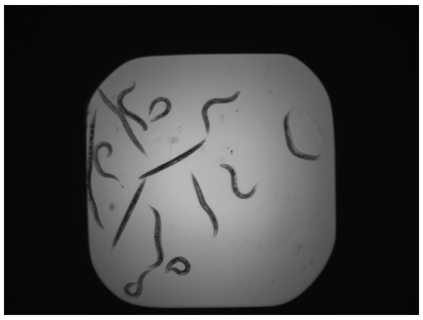
1. Isolate and Compare
   (a) **Isolate** the worms from the background and border.
   (b) **Compare** the isolated worms against the provided ground truth.
2. Individualise and Save each worm
   (a) Run the **watershed** algorithm to mark worms with unique colours.
   (b) Save each individual worm to disk, under img_out/separated/ directory.
3. Label each worm either dead or alive
   (a) Find the rotated bound box of each contour, and classify dead or alive depending on the width/height ratio of said box.

Examples of what the above steps produce can be found under later sections. All images shown here are of A01 w2, to clearly show the changes.

Choices Made

As the images are saved as 16 bit but only really using 12 bits, when reading as 8 bit images the 4 most significant bits were lost, and the image was very dark (Figure a), and so I read in the images as 16 bit, right shifted by 4 bits, and then converted to 8 bit. This ignores the 4 least significant bits, in favour of the 4 most significant bits. This results in a much more usable and bright image (Figure b).


(a) Loaded as 8 bit


(b) Loaded as 16 bit and shifted

I then had to ensure the image was of reasonable brightness, and I found OpenCV's adaptive threshold to be much more effective than it's histogram equalisation, and the possibility to go straight to a binary image proved useful.

The next choice I made was border removal for w2. My initial attempt found all contours, and assumed the largest was the border, filling this in black to remove it. This worked for most images, although big worm clusters were occassionally larger than the border in area, and so were removed instead. I instead settled on dilating a flood fill that started from the point (0, 0). This meant that it would surround the border, and slowly grow over it. Experimenting with the values given to this function resulted in very good border removal.

When comparing my binary images to the provided binary images, I decided to simply subtract one from the other, so that the resulting image's white pixels were representative of the error of my conversion. While this always resulted in high percentage values due to large empty spaces, and a big border surrounding the plate itself, other methods seemed to always provide no measure of how big or small their result actually was, they were just numbers without context.

To detect which worms were dead, and which were alive, I initially tried using contours, which could be checked by a method called isContourConvex, as this would indiciate whether the worm was curved or not. However, this method was inconsistent, and unreliable in actually providing a boolean return, so I changed to finding the bounding box of the contour, and checking whether it is long and thin, or approximately square.

Evidence of the success of system in performing the specified task

Processing Steps

The loaded image, before any of the major processing steps, is shown as Figure 1.

Step 1, which isolated the worms from the background and border, produced Figure 2.

Step 2, which isolated individual worms, produced Figure 3 where worms are all outlined in red, and Figure 4 which has each worm coloured a different shade of grey.

Step 3, which drew boxes around worms, and coloured them green if considered alive, red if considered dead, produced Figure 5, the final image which is shown on screen during runtime.

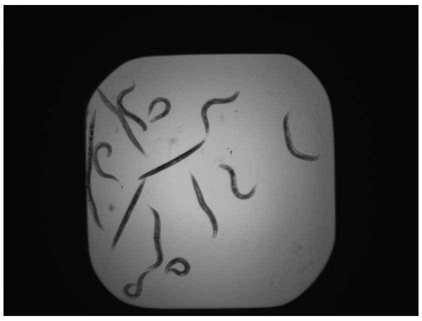


Figure 1

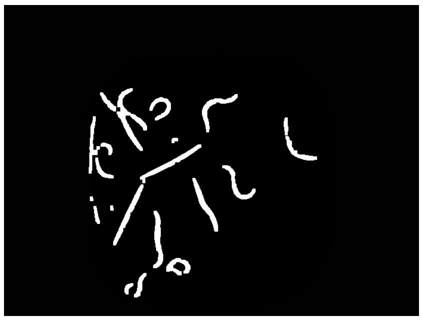


Figure 2

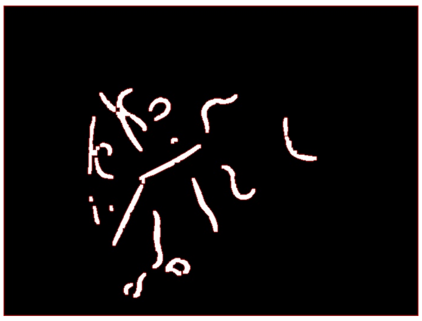


Figure 3

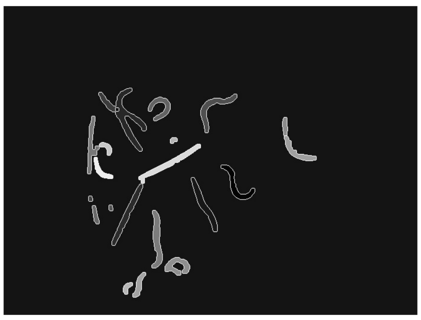


Figure 4

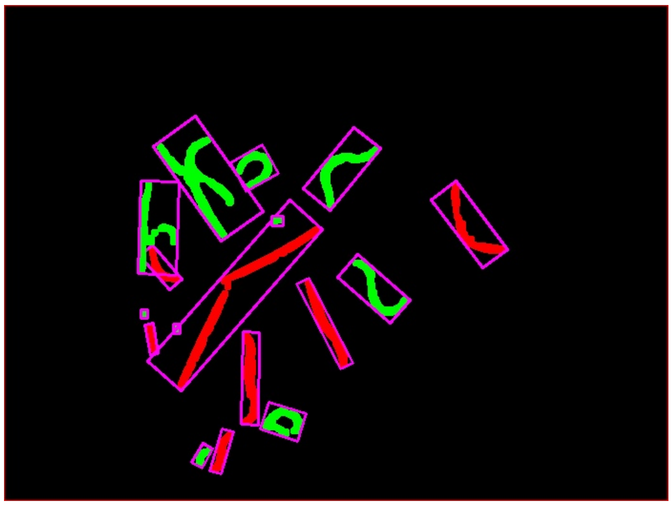


Figure 5