Analysis

January 31, 2017

1 Analysis of timelapse data from Raspiscope

1.1 Preliminaries

Before processing the images we need to set up the tools and load the data. We need to import several packages, so before running this notebook you should create an environment (conda or virtualenv) with matplotlib, numpy, and scikit-image, and jupyter. For example:

```
conda create env -n raspiscope jupyter matplotlib numpy scikit-image
```

and then activate it, e.g.:

```
source activate raspiscope
```

or select the conda env here in Jupyter.

1.1.1 Modules

First import the standard tools, numpy and matplotlib. These are very well documented packages, more info can be found here:
http://www.matplotlib.org
http://www.numpy.org

```
In [2]: %matplotlib inline
        import matplotlib
        import matplotlib.pyplot as plt
        import numpy as np
```

We will use scikit-image to analyse and process images. There are many examples and tutorials here:
http://www.scitkit-image.org

```
In [3]: import skimage
        from skimage import io, filters
```

1.1.2 Loading images

First we specify a text string with the location and name of image files in the timelapse. We put a %d to represent the frame number, %04d means put the number with 4 digits.

You should replace the text below with the location of the images on your computer. Using the % operator on the string we can then form any file name in the series:

```
In [4]: #fname = "/Users/timrudge/RaspiScope/Timelapse060117/image_%04d.jpg"
        #fname = "/Users/timrudge/RaspiScope/timelapse/timlap_3_%04d.jpg"
        fname =  "/Users/timrudge/RaspiScope/Lab_19_1_17/image_%04d.jpg"
        fname%(0)
```

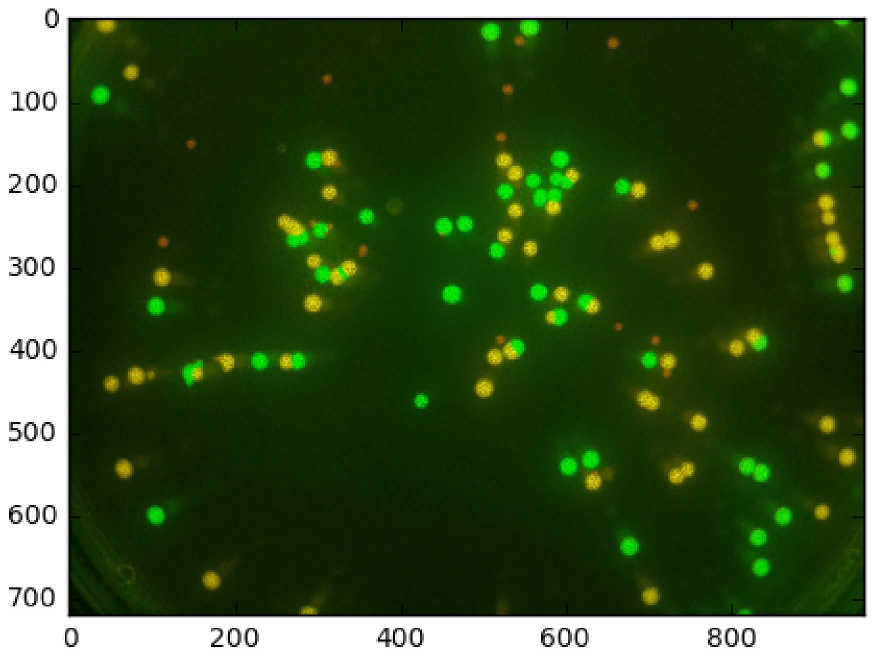
```
Out[4]: '/Users/timrudge/RaspiScope/Lab_19_1_17/image_0000.jpg'
```

Lets see what frame 500 looks like. We use PyPlot to read the file into a numpy array. This is a 3-dimensional array (a grid of values) with dimensions x,y,c, with (x,y) position and c the color channel (red, green, blue).

We can use PyPlot again to display the array, which interprets the array as a color image correctly.

```
In [5]: im = plt.imread(fname%500)
        plt.imshow(im)
```

```
Out[5]: <matplotlib.image.AxesImage at 0x1141787d0>
```



1.1.3 Loading the time series

There are 3 channels in each image. For now we will just look at the green channel (c=1). We will load the green channel of all the images into a 3-dimensional array, with dimensions x, y, time.

First get the size of the image from the numpy array shape, the 3rd dimension size is 3 for R,G,B. We dont need it so use the "_" variable.

```
In [6]: im.shape
        w,h,_ = im.shape
```

Now we create a numpy array to hold the data and loop over the files, taking only channel 1 (green). There are a lot of time points in this series, so here we take every 10th frame:

```
In [7]: nt = 55
        ims = np.zeros((w,h,nt))
        for i in range(0,nt):
            im = plt.imread(fname%(i*10))
            ims[:,:,i] = im[:,:,1]
```

Now we have a stack (x,y,t):

```
In [8]: ims.shape
```

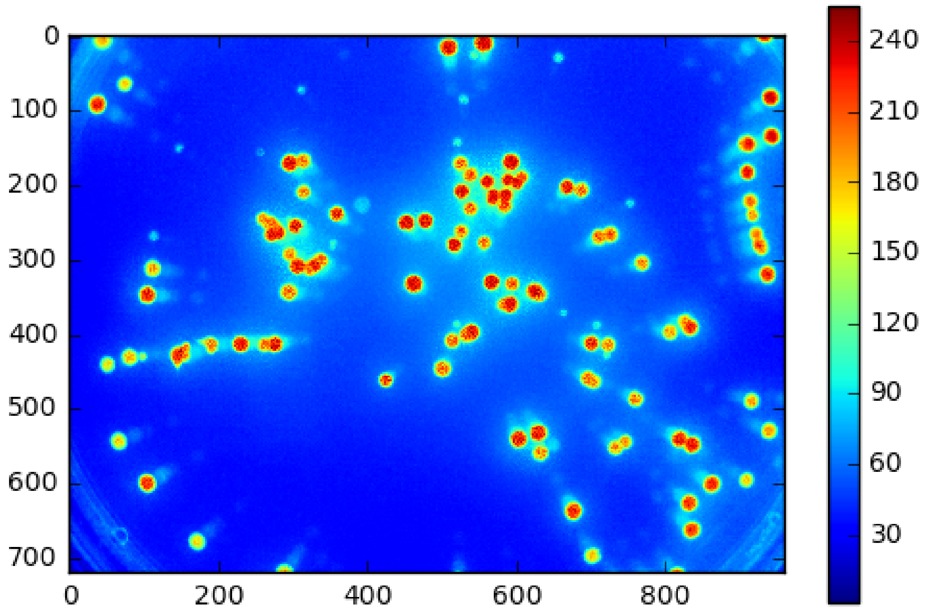
```
Out[8]: (720, 960, 55)
```

1.2 Analysis of time-lapse images

Now we have all the data loaded into a numpy array. To speed things up, take a subregion of the image to analyse. We do this by 'slicing' the array:

```
In [9]: imssub = ims #[200:600,200:600,:]
        plt.imshow(imssub[:,:,-1])
        plt.colorbar()
```

```
Out[9]: <matplotlib.colorbar.Colorbar at 0x1081d1b90>
```

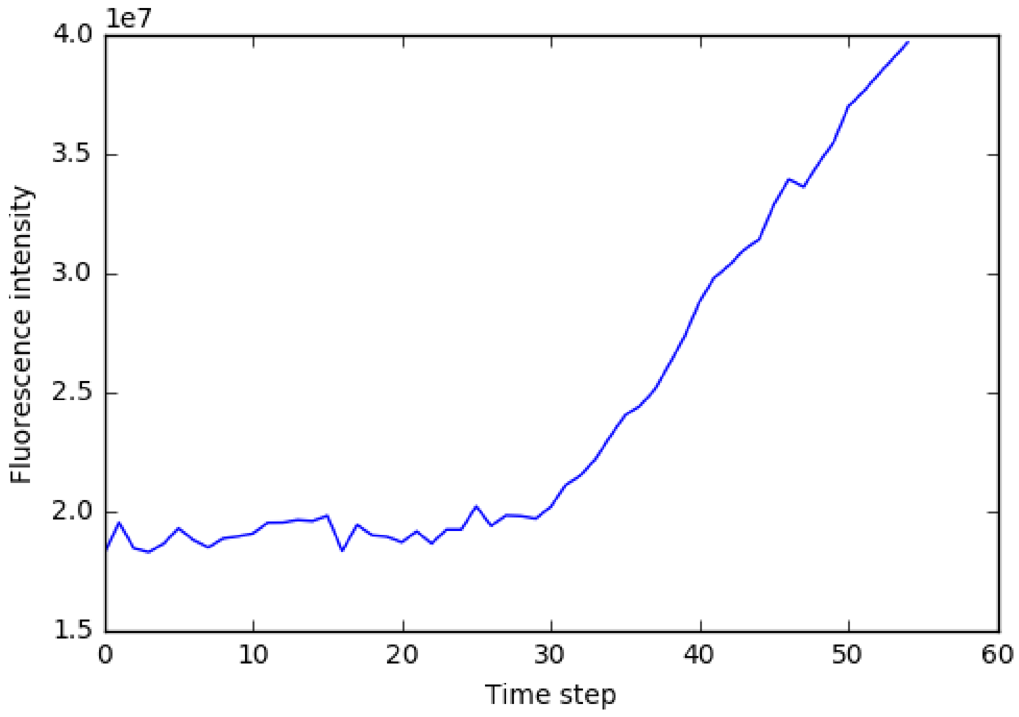


1.2.1 Mean dynamics over image domain

We can get some information by looking just at the variation in total image intensity in the time-series. Do this by taking the sum of the (x,y,t) stack over the first two dimensions (0,1):

```
In [10]: sumims = imssub.sum(axis=(0,1))
         plt.plot(sumims)
         plt.xlabel('Time step')
         plt.ylabel('Fluorescence intensity')
```

```
Out[10]: <matplotlib.text.Text at 0x11474a810>
```

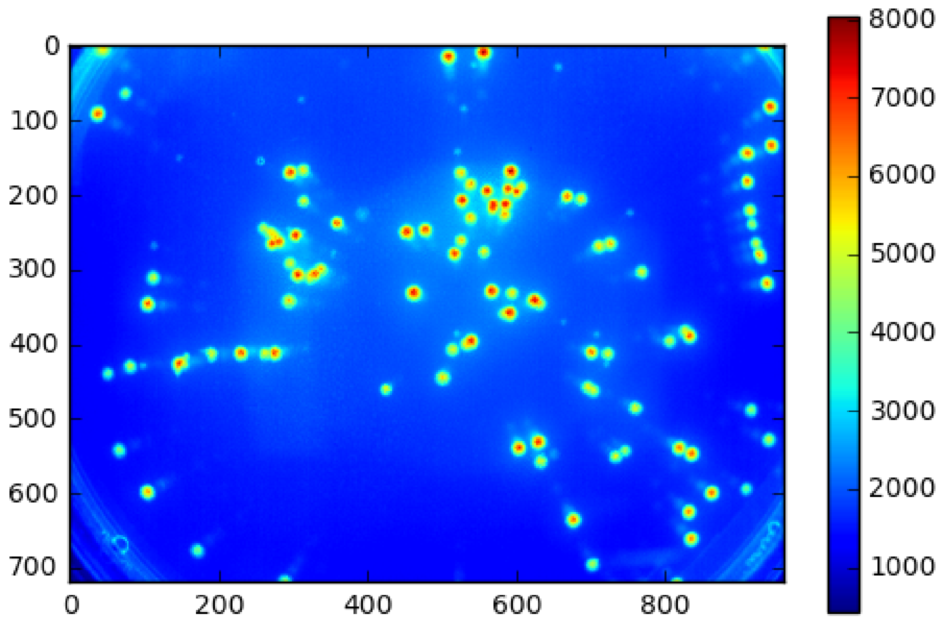


1.2.2 Identifying colonies

The colonies vary in brightness over the experiment but do not move. So we can take the sum of intensity of each pixel over time, and use it to locate colonies reliably:

```
In [11]: ims_sum = imssub.sum(axis=2)
         plt.imshow(ims_sum)
         plt.colorbar()
```

```
Out[11]: <matplotlib.colorbar.Colorbar at 0x127a7e450>
```

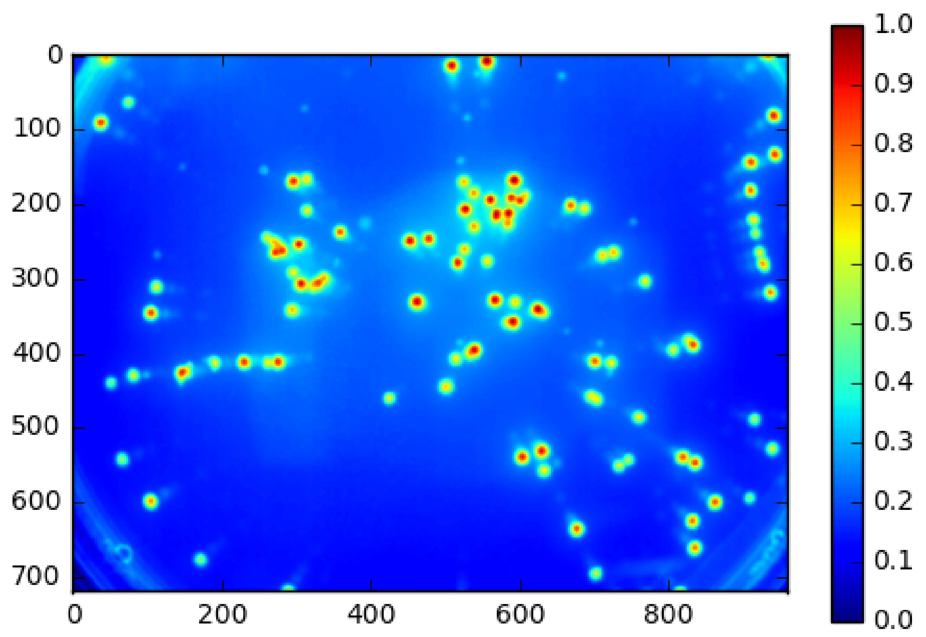


With any image its essential to remove noise before analysis, here with a Gaussian filter. Normalisation to (0,1) is also convenient:

```
In [12]: from skimage.filters import gaussian
         simsub = gaussian(ims_sum, 2)
         nsimsub = (simsub-simsub.min())/(simsub.max()-simsub.min())

In [13]: plt.imshow(nsimsub)
         plt.colorbar()

Out[13]: <matplotlib.colorbar.Colorbar at 0x12cc98850>
```



Blob detection Scikit-image provides several functions to detect Gaussian-like features (blobs). We apply one of these (blob_log) to the smoothed sum of intensity, and it returns a list of position (x,y) and width (standard deviation):

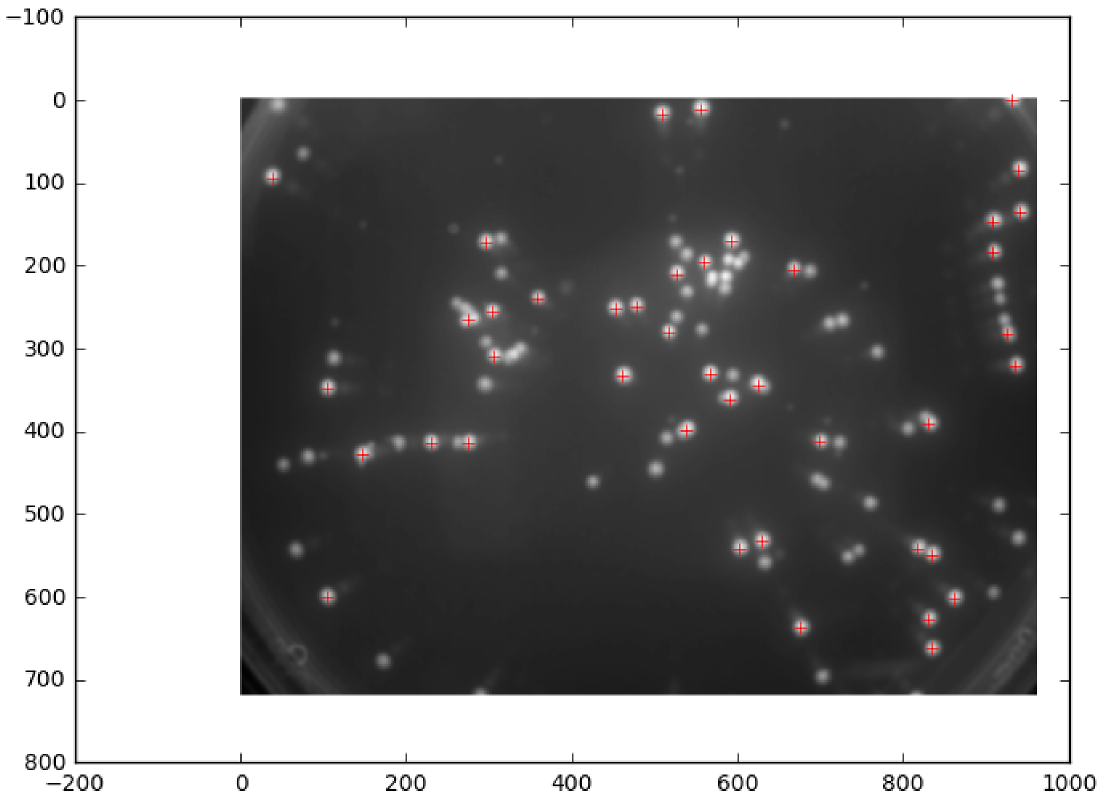
```
In [14]: import skimage.feature as skfeat
         A = skfeat.blob_log(nsimsub, min_sigma=1.0, max_sigma=10.0, num_sigma=100,

In [15]: print A.shape
         print A[0:4,:]
```

```
(42, 3)
[[   0.          931.            6.72727273]
 [  12.          555.            6.09090909]
 [  17.          508.            6.36363636]
 [  84.          939.            6.09090909]]
```

The array A now contains the location of probable colonies. We can check by showing the image and plotting markers on top at each location contained in A:

```
In [16]: plt.figure(figsize=(8,8))
         plt.imshow(nsimsub, cmap='gray')
         plt.hold(True)
         for i in range(len(A)):
             plt.plot(A[i,1],A[i,0],'r+')
```

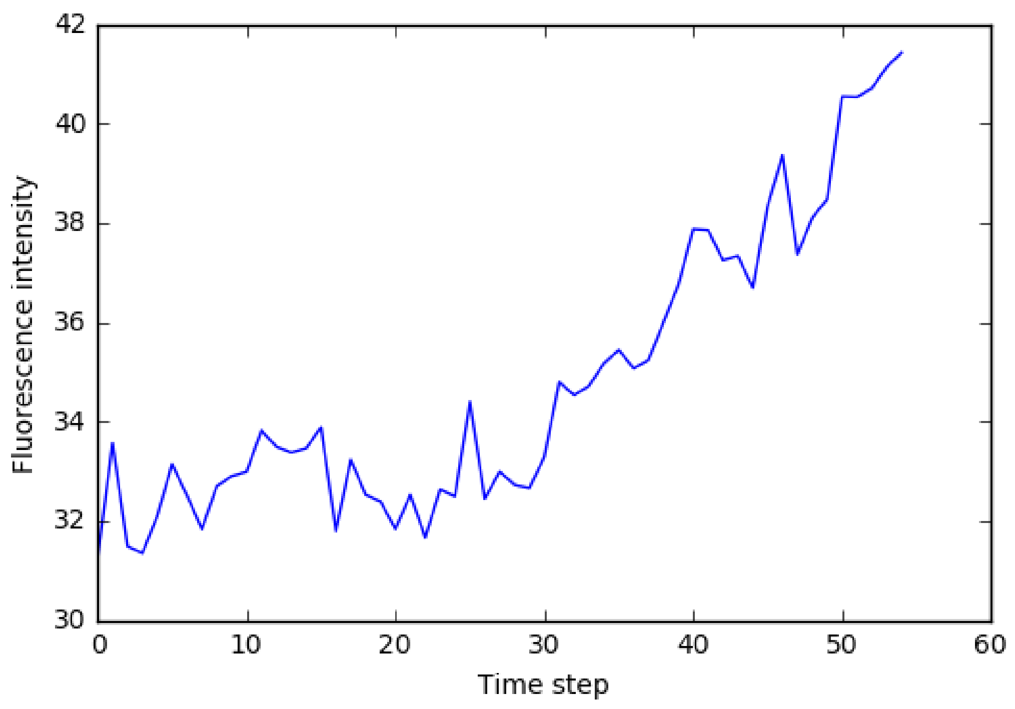


1.2.3 Time dynamics of colonies

Now we have estimated colony location (and size) we can analyse the dynamics of fluorescence and growth. First to distinguish cell fluorescence from media and background light, we select an empty region of the plate (by eye) and take the mean value in each frame as the background:

```
In [17]: #bg = imssub[300:400,0:100,:].mean(axis=(0,1))
         bg = imssub[0:100,200:300,:].mean(axis=(0,1))
         plt.plot(bg)
         plt.xlabel('Time step')
         plt.ylabel('Fluorescence intensity')

Out[17]: <matplotlib.text.Text at 0x12a99f790>
```



To restrict image analysis for each colony, we can extract a region at each location, with size given by the Gaussian width (x2 = 2 standard deviations). Here we put the image series into a dict indexed by colony. We also subtract the background estimate from each time series.

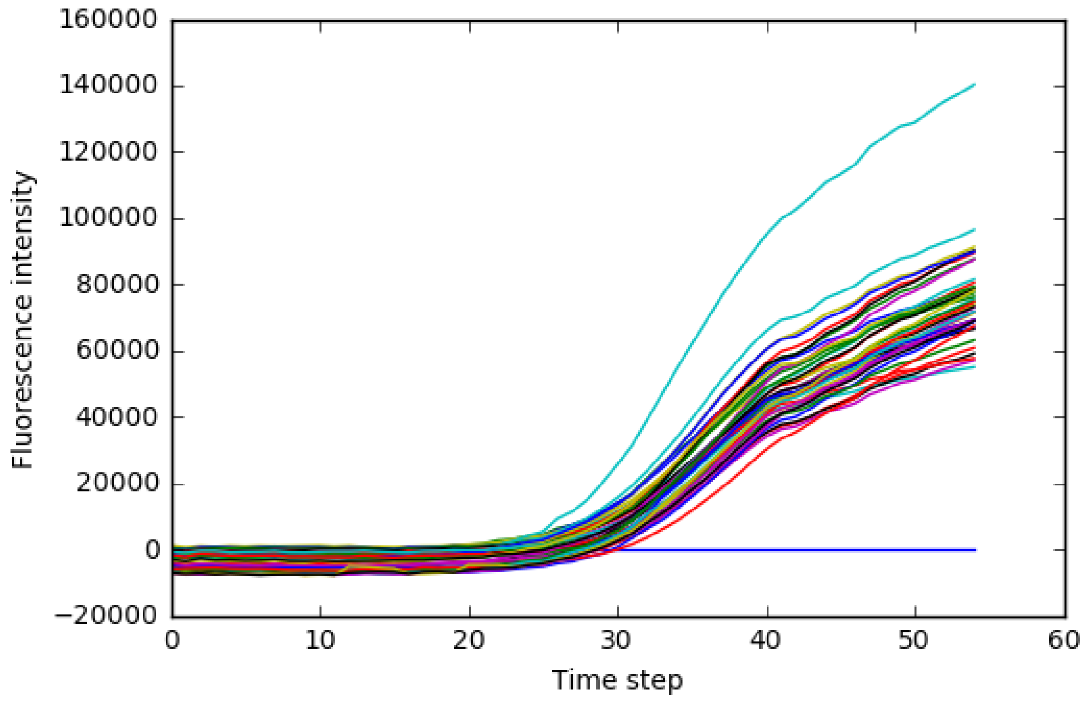
```
In [18]: rois = {}
         for i in range(len(A)):
             x = A[i,0]
             y = A[i,1]
             r = 2*A[i,2]
             rois[i] = imssub[x-r:x+r,y-r:y+r,:]-bg
```

```
/Users/timrudge/miniconda/envs/raspiscope/lib/python2.7/site-packages/ipykernel/__m
```

Now we can plot the total fluoresence in the region of each colony:

```
In [19]: for i in range(len(A)):
             plt.plot(rois[i].sum(axis=(0,1)))
             plt.hold(True)
         plt.xlabel('Time step')
         plt.ylabel('Fluorescence intensity')
         #plt.legend(['Colony %d'%i for i in range(len(A))])
```

```
Out[19]: <matplotlib.text.Text at 0x12ab6c810>
```



Colony radius growth We have the location and some kind of size estimate based on the total fluorescence of each colony over the time-lapse. We would like to know the size of the colony in each frame; from this we can compute the growth rate.

One way to do this is the find Gaussian blobs in each of the image regions of the colony, for each frame of the time-lapse. Then assume that the first (only?) blob detected is the colony we are interested in. This is very slow, there may be a faster way since we can assume a single Gaussian in each image region. Anyway for now it works ok.

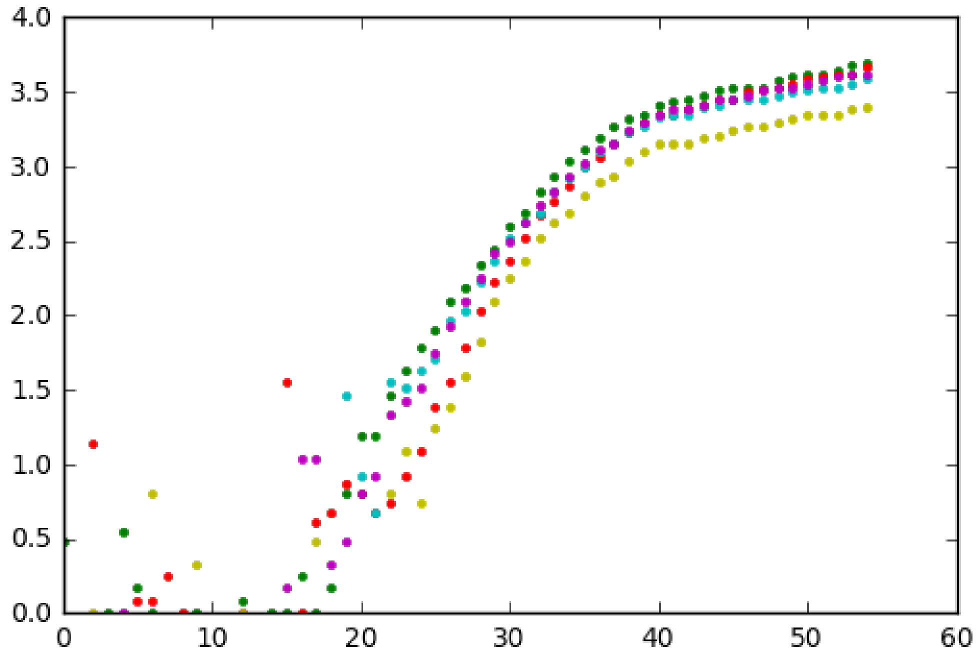
The following loop finds blobs and stores the width in each frame. The result is a dict containing the size at each time point for every colony.

```
In [20]: R = {}
         for k in range(len(A)):
             R[k] = np.zeros((nt,))
             for i in range(nt):
                 troi = rois[k][:,:,i].astype(np.float32)
                 if len(troi):
                     ntroi = (troi-troi.min())/(troi.max()-troi.min())
                     AA = skfeat.blob_log(ntroi, min_sigma=1.0, max_sigma=10.0, num
                     if len(AA)>0:
                         R[k][i] = AA[0,2]
```

Since the colony is roughly circular and flat, an estimate of volume is R^2. Lets plot this for some colonies. Plotting the log is useful because the slope is the relative growth rate. We can see the colonies slow their growth during the experiment.

```
In [21]: idx = [0,1,2,3,4,9]
         for i in idx:
             r = R[i]
             plt.plot(np.log(r*r), '.')
             plt.hold(True)
```

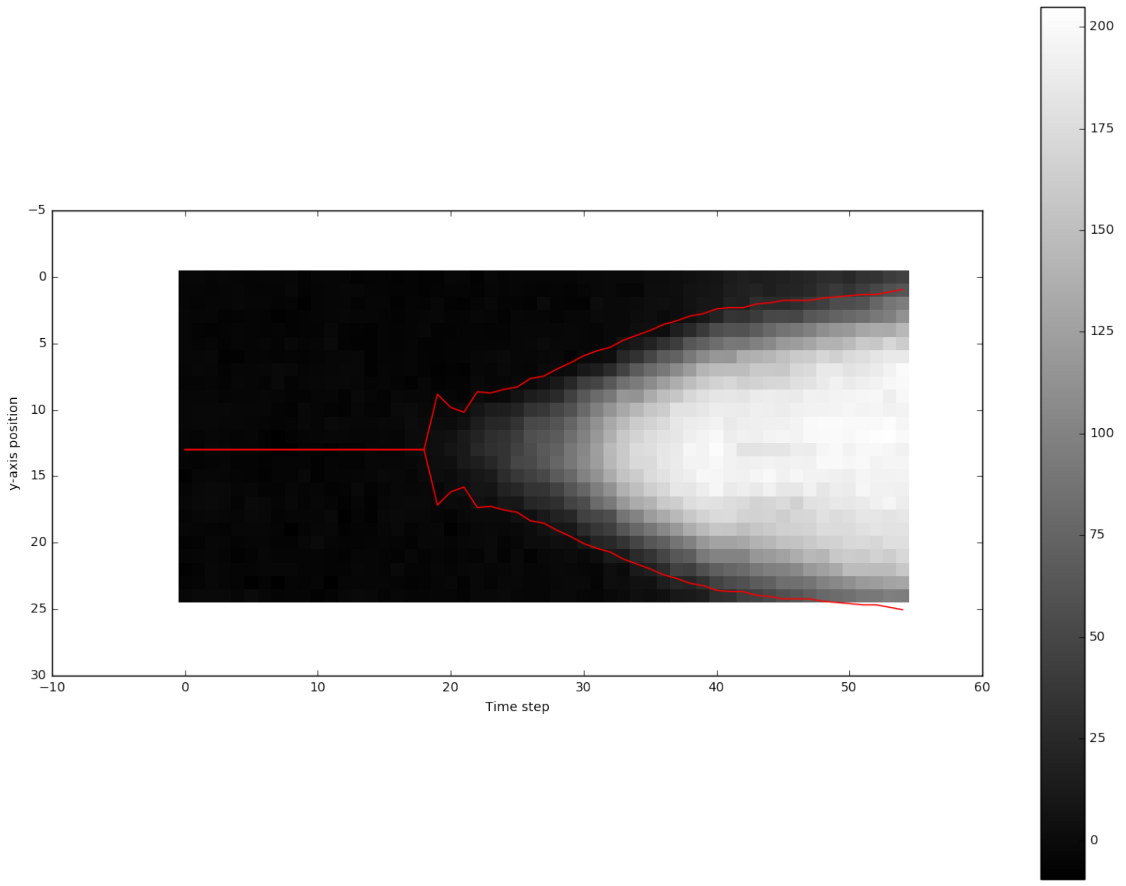
```
/Users/timrudge/miniconda/envs/raspiscope/lib/python2.7/site-packages/ipykernel/__m
```



Check the radius estimate To see how well the colony size is estimated, we can compare to the profile of a colony in a slice of its image region. The radius (red line) should give the border of the colony.

```
In [22]: idx = 3
         plt.figure(figsize=(16,12))
         w,h,d = rois[idx].shape
         plt.imshow(rois[idx][w/2+1,:,:], interpolation='none', cmap='gray')
         plt.colorbar()
         plt.hold(True)
         plt.plot(-R[idx]*2+h/2+1,'r')
         plt.plot(R[idx]*2+h/2+1,'r')
         plt.xlabel('Time step')
         plt.ylabel('y-axis position')
```

```
Out[22]: <matplotlib.text.Text at 0x1302fc710>
```



1.2.4 Dynamics plotting

A simple way to view dynamics is a movie, plotting each frame successively. To do this we have to turn off inline plotting so that a new window will open. Also create the directory to place output images. These can then be converted into a video (e.g. using Fiji/ImageJ).

```
In [24]: %matplotlib auto
         plt.figure(figsize=(12,10))
         for i in range(nt):
             for idx in range(20):
                 if len(rois[idx]):
                     mx = np.max(rois[idx+20])
                     plt.subplot(4,5,idx+1)
                     #plt.imshow(rois[idx+20][:,:,-1], interpolation='none', vmin=(
                     #plt.colorbar()
                     roi = rois[idx+20][:,:,i]
                     plt.imshow(roi, interpolation='none', vmin=0, vmax=255)
             # Uncomment here to save figures as images
             #plt.savefig('AnalysisOutput/rois2/rois_step%d.png'%(i))
             #plt.title('Colonies 1-20')
```

```
            plt.pause(0.1)
        %matplotlib inline
```

```
Using matplotlib backend: MacOSX
```

```
/Users/timrudge/miniconda/envs/raspiscope/lib/python2.7/site-packages/matplotlib/im
  "mode." % renderer.__class__)
```

1.2.5 Parameter estimation from data

We now have information about sub-populations of bacteria (colonies) on the same plate. The fluorescence in each region of interest estimates the time varying fluorescent protein level in each colony. The radius (or R^2) approximates the size of each sub-population (colony) and so is similar to optical density (OD) in a bulk plate-reader experiment.

Lets write a simple model of fluorescent protein expression from a single cell:

$$\frac{dF}{dt} = k(t) - (\mu(t) + \delta(t))F \tag{1}$$

where F is the cell's fluorescent protein concentration, which is produced at time-varying rate $k(t)$. The cell grows at relative rate $\mu(t) = (1/V)dV/dt$ (V =volume) causing dilution. The protein is degraded at rate $\delta(t)$.

For the proteins we are using we will assume that $\delta(t) = 0$ for all times. We measure the total fluorescence intensity, which is the concentration F multiplied by the total volume of the colony $V_{tot} \approx R^2$. Hence,

$$I(t) = F(t)V(t) \approx F(t)R^2(t) \tag{2}$$

From these equations it is possible to derive:

$$k(t) = \frac{1}{V(t)}\frac{dI}{dt} \tag{3}$$

Lets get V and I from our data:

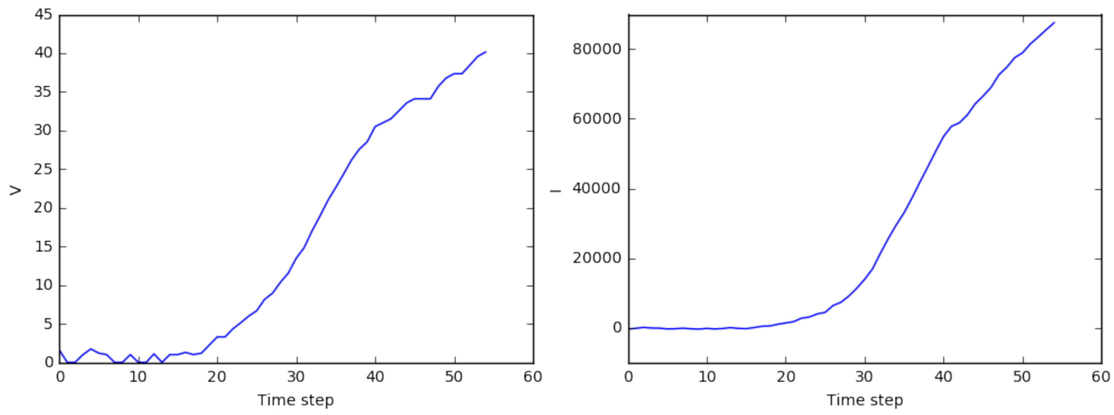
```
In [25]: # Approximate volume as R^2
         V = {}
         for idx,r in R.iteritems():
             V[idx] = r*r

         # Get total fluorescence intensity for each colony, summing over all pixel
         I = {}
         for idx,roi in rois.iteritems():
             I[idx] = roi.sum(axis=(0,1))
```

Plot an example colony:

```
In [43]: plt.figure(figsize=(12,4))
         plt.subplot(1,2,1)
         plt.plot(V[1])
         plt.xlabel('Time step')
         plt.ylabel('V')
         plt.subplot(1,2,2)
         plt.plot(I[1])
         plt.xlabel('Time step')
         plt.ylabel('I')
```

```
Out[43]: <matplotlib.text.Text at 0x145b1fa10>
```

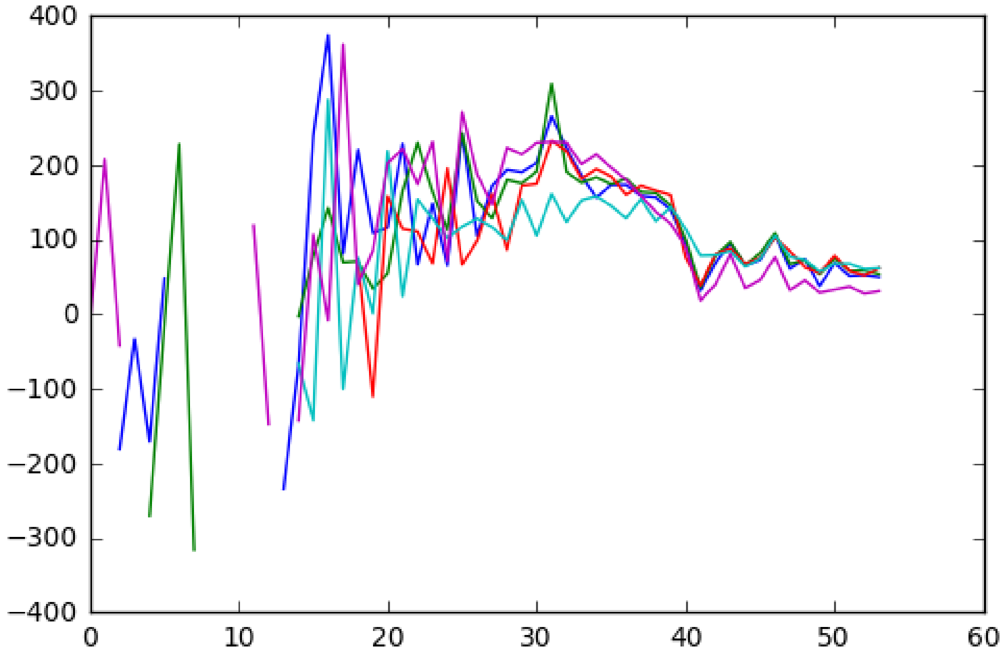


Now we can use a simple approach to estimate $k(t)$. Since $dI/dt \approx \Delta I/\Delta t$, we can compute:

$$k(t) \propto \frac{I(t) - I(t-1)}{V(t)} \tag{4}$$

```
In [48]: colonies = [1,2,3,4,10]
         for idx in colonies:
             intensity = I[idx]
             volume = V[idx][1:]
             delta_I = np.diff(intensity)
             k = delta_I/volume
             plt.plot(k)
             plt.hold(True)
```

```
/Users/timrudge/miniconda/envs/raspiscope/lib/python2.7/site-packages/ipykernel/__m
```


400
300
200
100
0
−100
−200
−300
−400
0
10
20
30
40
50
60