Code_to_find_and_measure_lasers

August 2, 2021

0.1 Image Processing: Measuring properties of laser spots

Author: Mark Reeves

- **Date:** 7/16/2020
- **Affiliation:** George Washington University

In this demo, we will measure the properties of laser dots from photos on gorillas. We'll use image processing concepts from the image prcoessing framework in Python, **scikit-image**.

This code can be used as a step 2 after the 'Gorilla Mask Code' (see github: https://github.com/ejlevy/Photogrammetry_Coding_InterLaser_Distance) or alternatively can be used alone without the gorilla masking as the first step. Using it alone is advisable if less familiar with python coding and implementing machine learning techniques.

We recommend running this script in Jupyter notebooks, we use the Juptyer notebooks plugin using anaconda navigator.

After downloading the ipynb file from github (https://github.com/ejlevy/Photogrammetry_Coding_InterLaser_D create a sub folder in the same folder that you place the code named 'Images_to_process'. This folder can be renamed, however you must changed the root directory line to match the organisation of the folder, in the 'Setting working directory and files to save outputs in.' code block line: 'root_dir="Images_to_process"' - thes name and the folder name must match exactly.

0.1.1 Defining Functions

Run the below code blocks to define the functions necessary for the coding.

[ ]:
```
def mask_image(imag,mask,min_cut,max_cut):
#
# Function takes in an array, its mask, 0's and 1's that define particles, and␣
 ↪a cutoff for particle size.
# It masks the array and returns the number of particles, the B/W mask of␣
 ↪labeled particles and resulting
# masked image.
#
    s = ndi.generate_binary_structure(2,2)
    part_array,n_part=ndi.label(mask, structure=s)
    size_part=np.bincount(part_array.ravel())
    mask_sizes = (size_part > min_cut)&(size_part < max_cut)  # Remove noisy␣
 ↪pixels.
```

```
    size_part=mask_sizes
    mask_sizes[0] = 0
    mask_c = mask_sizes[part_array] # Redimension array back to image aspect␣
 ↪ratio.
    part_array,n_part=ndi.label(mask_c, structure=s)
    im_out = imag
    im_out[:,:,0] = imag[:,:,0]*mask_c
    im_out[:,:,1] = imag[:,:,1]*mask_c
    im_out[:,:,2] = imag[:,:,2]*mask_c
    im_outi=Image.fromarray(im_out) # If cut = 1, then size_part=n_part, none␣
 ↪of the mask was cut out.
    return n_part,mask_c,im_outi,size_part;
```

[ ]:
```
def prop_gen(mask_c,img,code):
    m_img=img*mask_c

#    Input variables:
#    Implicit variables that need to be added to the function call are the:␣
 ↪file name, the directory name,
#    the elapsed time.
#
#    Here is the list of variables to pass directly:
#    root_dir, img_name, t0

#    Invoke the measure package to measure the properties of each labeled␣
 ↪object in the image, the code is an
#    image descriptor and includes information about which step of the␣
 ↪iteration on the background variation is evoked.
#
    obj,num_obj=ndi.label(mask_c)
    masks=np.bincount(obj.ravel())
    masks[0] = 0
    label_arr = measure.label(mask_c, connectivity = 1)

    # Get dots green color properties.
    Props = measure.regionprops(label_arr, m_img)
    PropsTab = measure.regionprops_table(label_arr, m_img,␣
 ↪properties=('label','bbox_area','centroid','weighted_moments_hu',\
   ␣
 ↪'weighted_centroid','mean_intensity','max_intensity','eccentricity','area','perimeter'))␣
 ↪   # Get dataframe with dot properties.
    print('number of objects =',num_obj)

    # Prepare to write object properties to a file, first based in green␣
 ↪intensity.
```

```
    listdir = [code+root_dir] * (num_obj)    #Create list with file name of␣
↪image.
    dd={'0image_dir':listdir}
    PropsTab.update(dd)  # PropsTab is a dictionary with lists of object␣
↪properties extracted by the measure package.
    listfile = [img_name] * (num_obj)    # Create list with file name of image.
    dd={'0image_file':listfile}
    PropsTab.update(dd)
    listT =[str(time.time()-t0)] * (num_obj)    # Create list with elapsed time,␣
↪but leave space at the end.
    dd={'secs elapsed':listT}  # Create elapsed time entry.
    PropsTab.update(dd)  # Add list of time to dictionary.
    listout = [root_dir+'/'+ 'laserdotfind/ '+img_name+" grn-Inten"] *␣
↪(num_obj)    # Create list with file name.
    dd={'the out files':listout}  # Create list of results files.
    PropsTab.update(dd)  # Add list of time to dictionary.
    PTab=pd.DataFrame(PropsTab) # Convert PropsTab to a data frame for easier␣
↪manipulation and storage.
    PTab.insert(0, "Obj_size", masks[1:], True) # Calculate composite␣
↪properties for the objects and addthem to the data frame.
    PTab['circularity']=4*np.pi*PTab['area']/PTab['perimeter']/
↪PTab['perimeter'] # Standard circularity, ratio of bounding box
    # to object area. A circle has a value of 4/pi, a value of 1 is a perfect␣
↪circle, a long strand will have a value much larger than 1.
    PTab['circsqr']=.25*np.pi*PTab['bbox_area']/PTab['area']
    PTab['total_intensity']=PTab['area']*PTab['mean_intensity'] # How bright␣
↪does an object appear? Number of pixels times the brightness of each one.
    # How peaked is the brightness of the object? A uniform object has value of␣
↪1. A 2-d, uniform, circularly
    # symmetric Gaussian distribution would have a value of bbox/2*pi*sig^2,␣
↪which would be pi/2 for a FWHM = half the
    # bounding box.
    PTab['intensity_ratio']=PTab['max_intensity']/PTab['mean_intensity']
    PTab['GBR_ratio']=PTab['intensity_ratio']/PTab['intensity_ratio']
    return num_obj,label_arr,Props,PTab
```

[ ]:
```
def I2_contour(huf):

# Function calculates the I2 contour from the sum of the 7 hu-moments.

    n,m=huf.shape
    si=np.copysign(np.ones([n,m]),huf)
    huf=np.abs(huf)
    huf=si*np.log10(huf)
    hu_sum=np.sum(huf, axis=1)
    return hu_sum
```

[ ]:
```
def dot_check(tem):
#
# Function to check dots are in a right angle, if you want to use this one␣
 ↪change name to `dot_check`.
# (This function works for the GW-gorilla dataset)

# Looks for 3 spots in an "L"-shaped configuration. The input array, tem, is␣
 ↪three elements
# long and is taken from rows of the particle properties array (PTab or␣
 ↪PTabgrn) Two element functionality can
# be added later.
#
    nr,nc=shape(tem)
    Flag=False
    E_code='NA'
    box=array([[0,0],[1,1]])
    box=box.astype(int)
    coords= [[1,2,3,4,5,6,7,8]]

    # Create the pandas DataFrame.
    coords=pd.DataFrame(coords, columns = ['hor dist', 'vert dist', 'diag',␣
 ↪'calc hypot', 'hor wt',\
                                           'vert wt','diag wt', 'calc hypot␣
 ↪wt'])
    d01=0
    d12=0
    d02=0
    dh=0
    d01w=0
    d12w=0
    d02w=0
    dhw=0
    if(nr>=2):
        tem=tem.sort_values(by='centroid-1', ascending=0)
        tem.reset_index(drop=True, inplace=True)

    # Sort the three laser spots into the standard order.
        if(nr==3):
            if(np.abs(tem['centroid-0'][1]-tem['centroid-0'][0])>(.2*np.
 ↪abs(tem['centroid-0'][1]-tem['centroid-0'][2]))):
                tem=tem.reindex([0,2,1])
                tem.reset_index(drop=True, inplace=True)
        d=np.
 ↪sqrt((tem['centroid-1'][1]-tem['centroid-1'][0])**2+(tem['centroid-0'][1]-tem['centroid-0']
        dw=np.
 ↪sqrt((tem['weighted_centroid-1'][1]-tem['weighted_centroid-1'][0])**2\
```

```
                    ␣
↪+(tem['weighted_centroid-0'][1]-tem['weighted_centroid-0'][0])**2)
        if (np.abs(tem['centroid-0'][1]-tem['centroid-0'][0])<.2*np.
↪abs(tem['centroid-1'][1]-tem['centroid-1'][0])):
            d01=d
            d01w=dw
            Flag=True
            box=np.array([[int(np.min(tem['centroid-1'])-.2*d01),int(np.
↪min(tem['centroid-0'])-.8*d01)],\
                  [int(np.max(tem['centroid-1'])+.2*d01),int(np.
↪max(tem['centroid-0'])+.8*d12)]])
        elif (np.abs(tem['centroid-1'][1]-tem['centroid-1'][0])<.2*np.
↪abs(tem['centroid-0'][1]-tem['centroid-0'][0])):
            d12=d
            d12w=dw
            Flag=True
            box=np.array([[int(np.min(tem['centroid-1'])-.8*d12),int(np.
↪min(tem['centroid-0'])-.2*d12)],\
                  [int(np.max(tem['centroid-1'])+.8*d12),int(np.
↪max(tem['centroid-0'])+.2*d12)]])
        else:
            E_code='NHV'
            Flag=False
        coords= [[d01,d12,0,0,d01w,d12w,0,0]]
        if(nr==3):
            Flag=False
            E_code='N2'
            d12=np.
↪sqrt((tem['centroid-1'][1]-tem['centroid-1'][2])**2+(tem['centroid-0'][1]-tem['centroid-0']
            d02=np.
↪sqrt((tem['centroid-1'][2]-tem['centroid-1'][0])**2+(tem['centroid-0'][2]-tem['centroid-0']
            d12w=np.
↪sqrt((tem['weighted_centroid-1'][1]-tem['weighted_centroid-1'][2])**2\
                        ␣
↪+(tem['weighted_centroid-0'][1]-tem['weighted_centroid-0'][2])**2)
            d02w=np.
↪sqrt((tem['weighted_centroid-1'][2]-tem['weighted_centroid-1'][0])**2\
                        ␣
↪+(tem['weighted_centroid-0'][2]-tem['weighted_centroid-0'][0])**2)
            dh=np.sqrt(d01*d01+d12*d12)
            dhw=np.sqrt(d01w*d01w+d12w*d12w)
            box=np.array([[int(np.min(tem['centroid-1'])-.2*d01),int(np.
↪min(tem['centroid-0'])-.2*d12)],\
                   [int(np.max(tem['centroid-1'])+.2*d01),int(np.
↪max(tem['centroid-0'])+.2*d12)]])
        coords= [[d01,d12,d02,dh,d01w,d12w,d02w,dhw]]
```

```
    # Create the pandas DataFrame.
      coords=pd.DataFrame(coords, columns = ['hor dist', 'vert dist', 'diag',␣
↪'calc hypot', 'hor wt',\
                                             'vert wt','diag wt', 'calc hypot␣
↪wt'])
      box[box<0]=0
      if(max(tem['intensity_ratio'])<1.5):
          Flag=True
          E_code='IR'
      if(max(tem['circularity'])<.4):
          Flag=True
          E_code='CR'
      if (nr==3):
          a=np.abs(dh-d02)/d02
          b=np.abs(d01-d12)/d12

  # Check the angle of triangle. Should be a right angle.
          if np.logical_and.reduce([a<.1,np.logical_or.reduce([b<.2,(b>.
↪15)&(b<.6)])]):
              Flag=True
              E_code='NA'
          else:
              Flag=False
              E_code='N3A'

  # Doublecheck that the vertical dots align.
          if(np.abs(tem['centroid-1'][1]-tem['centroid-1'][2])>(.2*np.
↪abs(tem['centroid-1'][1]-tem['centroid-1'][0]))):
              Flag=False
              E_code='VA'
      coords['Laser_found']=Flag
      coords['Error']=E_code
      tr,tc=tem.shape

  # Initialize the values in coord with placeholders (useful in the case␣
↪where only two dots are found).
      for jj in range(3):
          tt='circ'+str(jj)
          coords[tt]=0
      for jj in range(3):
          tt='In_Rat'+str(jj)
          coords[tt]=0
      for jj in range(3):
          tt='area'+str(jj)
          coords[tt]=0
```

```
    # Now set the actual values.
        s=min(3,tr)
        for jj in range(s):
            tt='circ'+str(jj)
            coords[tt]=tem.at[jj,'circularity']
        for jj in range(s):
            tt='In_Rat'+str(jj)
            coords[tt]=tem.at[jj,'intensity_ratio']
        for jj in range(s):
            tt='area'+str(jj)
            coords[tt]=tem.at[jj,'area']
    return box,coords,Flag,E_code,tem
```

[ ]:
```
def dot_check_Duke(tem):
#
# Function to check for three dots that are in a straight line, if you want to␣
 ↪use this one change name to
# `dot_check`. (This function works for the Duke-baboon dataset)

# Looks for 3 spots in a straight line, or more to the point 3 objects whose␣
 ↪vertical
# spacing is less than 20% of their horizontal spacing. The input array, tem,␣
 ↪is either 2 or
# three elements long and is taken from rows of the particle properties array␣
 ↪(PTab or PTabgrn) If two
# elements are passed, the the function checks them for horizontal alignment␣
 ↪and computed the spacing between
# them in pixels.
#
    nr,nc=shape(tem)
    Flag=False
    E_code='NA'
    box=array([[0,0],[1,1]])
    box=box.astype(int)
    coords= [[1,2,3,4,5,6,7,8]]

    # Create the pandas DataFrame.
    coords=pd.DataFrame(coords, columns = ['dist1_2', 'dist2_3', 'total_dist',␣
 ↪'calc_total', 'wt_dist1_2',\
                                           'wt_dist2_3','wt_total_dist',␣
 ↪'calc_total_wt'])

    # Sort the three laser spots into the standard order.
    if(nr>=2):
        tem=tem.sort_values(by='centroid-1', ascending=0)
        tem.reset_index(drop=True, inplace=True)
```

```
    # Compute the distances.
        d01=np.
↪sqrt((tem['centroid-1'][1]-tem['centroid-1'][0])**2+(tem['centroid-0'][1]-tem['centroid-0']
        d01w=np.
↪sqrt((tem['weighted_centroid-1'][1]-tem['weighted_centroid-1'][0])**2\
                    ␣
↪+(tem['weighted_centroid-0'][1]-tem['weighted_centroid-0'][0])**2)
        coords= [[d01,0,1.94*d01,0,d01w,0,1.94*d01w,0]]
        if(nr>=3):
            d12=np.
↪sqrt((tem['centroid-1'][1]-tem['centroid-1'][2])**2+(tem['centroid-0'][1]-tem['centroid-0']
            d02=np.
↪sqrt((tem['centroid-1'][2]-tem['centroid-1'][0])**2+(tem['centroid-0'][2]-tem['centroid-0']
            d12w=np.
↪sqrt((tem['weighted_centroid-1'][1]-tem['weighted_centroid-1'][2])**2\
                          ␣
↪+(tem['weighted_centroid-0'][1]-tem['weighted_centroid-0'][2])**2)
            d02w=np.
↪sqrt((tem['weighted_centroid-1'][2]-tem['weighted_centroid-1'][0])**2\
                          ␣
↪+(tem['weighted_centroid-0'][2]-tem['weighted_centroid-0'][0])**2)
            dh=d01+d12
            dhw=d01w+d12w
            coords= [[d01,d12,d02,dh,d01w,d12w,d02w,dhw]]
            box=np.array([[int(np.min(tem['centroid-1'])-.2*d01),int(np.
↪min(tem['centroid-0'])-.9*d01)],\
                  [int(np.max(tem['centroid-1'])+.2*d01),int(np.
↪max(tem['centroid-0'])+.9*d01)]])

    # Check alignment and spacing of 3 dots.
            a=np.abs(dh-d02)/d02
            b=d01/d12
            if np.logical_and.reduce([a<.01,b>.95,b<1.2]):
                Flag=True
                E_code='NA'
            else:
                Flag=False
                E_code='N3A'
        E_code='HA2'

    # Create the pandas DataFrame.
        coords=pd.DataFrame(coords, columns = ['dist1_2', 'dist2_3',␣
↪'total_dist', 'calc_total', 'wt_dist1_2',\
                                               'wt_dist2_3','wt_total_dist',␣
↪'calc_total_wt'])
```

```
        if(nr==2):
            box=np.array([[int(np.min(tem['centroid-1'])-1.5*d01),int(np.
↪min(tem['centroid-0'])-.7*d01)],\
                  [int(np.max(tem['centroid-1'])+1.5*d01),int(np.
↪max(tem['centroid-0'])+.7*d01)]])
        box[box<0]=0

    # Doublecheck that the horizontal dots align. Horizontal deviation is not␣
↪more than 20% of the spacing and
    # the sum of the instances is within 1% of the total.
        Flag=False
        aa=True
        if(nr>=3):
            aa=np.abs(tem['centroid-0'][1]-tem['centroid-0'][2])<(.2*np.
↪abs(tem['centroid-1'][1]-tem['centroid-1'][2]))
            aa=np.logical_and.reduce([aa,a<.01,b>.95,b<1.2])
        bb=np.abs(tem['centroid-0'][1]-tem['centroid-0'][0])<(.2*np.
↪abs(tem['centroid-1'][1]-tem['centroid-1'][0]))
        if np.logical_or.reduce([aa,bb]):
            E_code='HA1'
        if np.logical_and.reduce([aa,bb]):
            Flag=True
            E_code='NA'
        if(max(tem['intensity_ratio'])<1.5):
            E_code='IR'
        if(max(tem['circularity'])<.4):
            E_code='CR'
        coords['Laser_found']=Flag
        coords['Error']=E_code
        tr,tc=tem.shape

    # Initialize the values in coord with placeholders (useful in the case␣
↪where only two dots are found).
        for jj in range(3):
            tt='circ'+str(jj)
            coords[tt]=0
        for jj in range(3):
            tt='In_Rat'+str(jj)
            coords[tt]=0
        for jj in range(3):
            tt='area'+str(jj)
            coords[tt]=0

    # Now set the actual values
        s=min(3,tr)
        for jj in range(s):
```

```
            tt='circ'+str(jj)
            coords[tt]=tem.at[jj,'circularity']
        for jj in range(s):
            tt='In_Rat'+str(jj)
            coords[tt]=tem.at[jj,'intensity_ratio']
        for jj in range(s):
            tt='area'+str(jj)
            coords[tt]=tem.at[jj,'area']
    return box,coords,Flag,E_code,tem
```

[ ]:
```
def dot_connect(PTabgrn,jk):
#
# This function takes as an input the index on one of the objects found in a␣
 ↪photograph, and to test if it is a laser
# spot follows the expected path to form the characteristic "L" shape of the␣
 ↪spot pattern. For any laser spot, there
# should be a second laser spot with the same coordinates on either the X or Y␣
 ↪axis.
#
    # Convert label into the corresponding index.
    jk=PTabgrn.loc[PTabgrn['label'] == jk].index[0]
    tem=PTabgrn[['centroid-0','centroid-1']]
    x=tem.at[jk,'centroid-0']
    y=tem.at[jk,'centroid-1']
    tem['Tx']=np.abs(tem['centroid-0']-x)/x<.025
    tem['Ty']=np.abs(tem['centroid-1']-y)/y<.025
    xpart=-1
    ypart=-1
    xF=False
    yF=False
    if (tem.sum()['Tx'])==2: # The x-partner.
        xp=tem.index[tem['Tx']]
        xpart=xp[0]
        xF=True
        if (xp[0]==jk):
            xpart=xp[1] # xpart is an index of the dataframe.
    if (tem.sum()['Ty'])==2: # The y-partner.
        yp=tem.index[tem['Ty']]
        yF=True
        ypart=yp[0]
        if (yp[0]==jk):
            ypart=yp[1] # ypart is an index of the dataframe.
    if ((xF) & (not yF)):  # The horizontal pair, and need to find the␣
 ↪y-partner of the new x-partner.
        y=tem.at[xpart,'centroid-1']
        tem['Ty']=np.abs(tem['centroid-1']-y)/y<.025
        if (tem.sum()['Ty'])==2: # The y-partner.
```

```
            yp=tem.index[tem['Ty']]
            ypart=yp[0]
            yF=True
            if (yp[0]==xpart):
                ypart=yp[1] # ypart is an index of the dataframe.
    if ((not xF) &(yF)):  # The vertical pair, and need to find the x-partner␣
↪of the new y-partner.
        x=tem.at[ypart,'centroid-0']
        tem['Tx']=np.abs(tem['centroid-0']-x)/x<.025
        if (tem.sum()['Tx'])==2: # The x-partner.
            xp=tem.index[tem['Tx']]
            xpart=xp[0]
            xF=True
            if (xp[0]==ypart):
                xpart=xp[1] # ypart is an index of the dataframe.
    return xF,yF,xpart,ypart
```

[ ]:
```
def resort_top3(PTabgrn,jk,min_dot):
    xF,yF,xpart,ypart=dot_connect(PTabgrn, tem.at[0,'label'])
    if xF&yF:
        PTabgrn.at[PTabgrn.loc[PTabgrn['label'] == xpart].index[0],'index']=-1
        PTabgrn.at[PTabgrn.loc[PTabgrn['label'] == ypart].index[0],'index']=-2
        PTabgrn.at[PTabgrn.loc[PTabgrn['label'] == tem.at[0,'label']].
↪index[0],'index']=-3
        PTabgrn=PTabgrn.sort_values(by='index', ascending=1)
        PTabgrn.reset_index(drop=True, inplace=True)
        tem=PTabgrn.head(min_dot)
        box,coords,Flag,E_code,tem=dot_check(tem)
        PTabgrn.drop(PTabgrn.index[:min_dot], inplace=True)
        PTabgrn=tem.append(PTabgrn)
        if not Flag:
            xF,yF,xpart,ypart=dot_connect(PTabgrn, tem.at[1,'label'])
            if xF&yF:
                PTabgrn.at[PTabgrn.loc[PTabgrn['label'] == xpart].
↪index[0],'index']=-4
                PTabgrn.at[PTabgrn.loc[PTabgrn['label'] == ypart].
↪index[0],'index']=-5
                PTabgrn.at[PTabgrn.loc[PTabgrn['label'] == tem.at[1,'label']].
↪index[0],'index']=-6
                PTabgrn=PTabgrn.sort_values(by='index', ascending=1)
                PTabgrn.reset_index(drop=True, inplace=True)
                tem=PTabgrn.head(min_dot)
                box,coords,Flag,E_code,tem=dot_check(tem)
                PTabgrn.drop(PTabgrn.index[:min_dot], inplace=True)
                PTabgrn=tem.append(PTabgrn)
                if not Flag:
                    xF,yF,xpart,ypart=dot_connect(PTabgrn, tem.at[2,'label'])
```

```
                    if xF&yF:
                        PTabgrn.at[PTabgrn.loc[PTabgrn['label'] == xpart].
↪index[0],'index']=-7
                        PTabgrn.at[PTabgrn.loc[PTabgrn['label'] == ypart].
↪index[0],'index']=-8
                        PTabgrn.at[PTabgrn.loc[PTabgrn['label'] == tem.
↪at[2,'label']].index[0],'index']=-9
                        PTabgrn=PTabgrn.sort_values(by='index', ascending=1)
                        PTabgrn.reset_index(drop=True, inplace=True)
                        tem=PTabgrn.head(min_dot)
                        box,coords,Flag,E_code,tem=dot_check(tem)
                        PTabgrn.drop(PTabgrn.index[:min_dot], inplace=True)
                        PTabgrn=tem.append(PTabgrn)
        PTabgrn=PTabgrn.sort_values(by='index', ascending=1)
        PTabgrn.reset_index(drop=True, inplace=True)
    return
```

[ ]:
```
def clean_flash(im_box_t,im_box,tem1,GRfactor,min_d,GRBmean,grnmean):
#
# Broken, changes the input arrays in an unexpected way after other␣
↪modifications to the programs.  FIS THIS.

#
# This function is a final pass over the box containing the laser dots. The␣
↪purpose is to recompute the
# dot centers with a focus only on the very brightest part of the laser spot.␣
↪This eliminates weight given
# to green flares that often surround very bright laser spots
#
    gimg=np.array(im_box_t)
    a = np.logical_and.reduce([(gimg[:,:,1]<50),(gimg[:,:,0]>200)])
    greenr=a  #Looser parameters
    g_part,g_mask,gg_im_out,sizer=mask_image(np.
↪array(im_box),greenr,100,1000000) # Gorillas are always > 100000 pixels.

# Define masks:
# g_mask is the gorilla mask, gg_im_out is the image masked by the gorilla␣
↪(most of the time!).

# Now begin searching for the laser dots. Start with a fairly relaxed criteria␣
↪on first pass
# so that we catch all the laser dots + more green than we would like.
    gimg=np.array(gg_im_out)
    test0=1.0*np.ones(shape(gimg[:,:,0]))
    GRratio=np.divide(10.0*gimg[:,:,1],(gimg[:,:,0]+test0))
    GBratio=np.divide(10.0*gimg[:,:,1],(gimg[:,:,2]+test0))
```

```
    BRratio=np.divide(10.0*gimg[:,:,2],10.0*(gimg[:,:,0]+test0))
    GRB=np.multiply(GRratio,GBratio)
    GRB_sum=gimg[:,:,1]+gimg[:,:,0]+gimg[:,:,2]
    a = np.logical_and.reduce([(gimg[:,:,1]>dot_factor*100),(GRratio >␣
↪GRfactor*12),GBratio > 12])
    b = np.logical_and.reduce([(gimg[:,:,1]>200),(GRratio > 12),GBratio > 11])
    c = np.logical_and.reduce([(gimg[:,:,1]>240),(GRratio > 12),GBratio > 12])
    d = np.logical_and.reduce([(gimg[:,:,1]>120),(GRratio > 14),GBratio > 12])
    e = np.logical_and.reduce([(gimg[:,:,1]>dot_factor*70),(GRratio >␣
↪14),GBratio > 10])
    f = np.logical_and.reduce([(gimg[:,:,1]>250),(GRratio > 9),GBratio > 9])
    greenr=np.logical_or.reduce([c,f])   #Looser parameters
    minsize=20

# Create objects based on GR ratio
    greenr=ndi.binary_fill_holes(greenr)  # Fill holes.
    greenr=ndi.binary_erosion(greenr)  # Erode to smoooth edges.
    greenr=ndi.binary_erosion(greenr)  # Erode to smoooth edges.
    greenr=ndi.binary_erosion(greenr)  # Erode to smoooth edges.
    g_part,g_mask,x_im_out,sizer=mask_image(np.
↪array(gimg),greenr,minsize,1000000)

# Now work with object properties. First pass, evaluated solely on GR ratio␣
↪levels of individual pixels
    mask_c=g_mask
    num_obj,label_ratio,Props,PTabgrn1=prop_gen(mask_c,GRB_sum,'A_grn_int')
    num_obj,label_ratio,Props,PTab1=prop_gen(mask_c,GRB,'A_GRB_ratio')
    PTab1['weighted_centroid-0']=PTabgrn1['weighted_centroid-0']
    PTab1['weighted_centroid-1']=PTabgrn1['weighted_centroid-1']
    num_obji=num_obj

# Any manipulations are performed on the whole data frame, mask can be resized␣
↪to the
# original image (ratio mask) size and shape by the operation:␣
↪newmask=masks[g_mask]

# Check for and extract right triangle dot pattern
    PTabgrn1=PTabgrn1.sort_values(by='total_intensity', ascending=0)
    PTabgrn1.reset_index(drop=True, inplace=True)
    PTabgrn1.reset_index(drop=False, inplace=True)    # Create column (first)␣
↪with index before modifying with filters.
    if(num_obj>0):
        PTab1=PTab1.sort_values(by='total_intensity', ascending=0)
        PTab1.reset_index(drop=True, inplace=True)
        PTab1.reset_index(drop=False, inplace=True)
```

```
# dot_check tests whether a triplet of objects are positioned in the right␣
 ↪place to be a laser spot. It returns
# the bounding box for the laser spot as well as the centroid spacing in pixels.
 ↪ Choose which shape dot_check you need from above.
    tem=PTabgrn1.head(3)
    box1,coords,Flag,E_code,tem=dot_check(tem)
    if ((E_code=='HA1')&(min_dot==2)):
        Flag=True
        E_code='NA'
    return box1,Flag,E_code,coords,tem
```

0.1.2 Importing libraries

Before getting started, install scikit-image through your python installation's package manager. You know it is working by running:

[ ]:
```
%pylab inline
import skimage
import time
import skimage.io as skio
import skimage.color as skcolor
import numpy as np
from matplotlib import pyplot as plt
import os
import shutil as sh
from skimage.feature import canny
from skimage import measure
from scipy import ndimage as ndi
import pandas as pd
from PIL import Image, ImageDraw,ImageFont
```

0.2 Workflow

0.2.1 Step 1: Import the image of the Gorilla

In the 'image to process' folder place images that you want to process. We recommend for the first time to just do one image to ensure the process works.

We will primarily use `scikit-image` in this demo. First, we will use `skimage.io.imread` to read the image into memory. The image can be plotted easily using matplotlib's `imshow()` function. We can import matplotlib, numpy and a handful of other scientific libraries use the `%pylab inline` notebook magic function (see Importing Libraries Section).

There is no need to present the images after each step, which takes time, but these steps are illustrative of the process.

0.2.2 Step 2: Image properties

Scikit image will store this image as a numpy array. Because this is a color image, we expect the image to have three dimensions: two spatial dimensions for the x and y coordinates of the pixels, and four color coordinates the the red green, blue and alpha (opacity) channels of the color. For example:

`I(x,y,r,g,b) = (200, 400, 0.5, 0.2, 0.7, 1.0)`

Would mean that the pixel at position `x=200, y=400` has an rgb color of (0.5, 0.2, 0.7) and has an opacity of 1.0.
Opacity: some color images are stored as three channels, but some are stored with this fourth opacity channel. Scikit-image basically ignores the opacity. We can verify the dimensions and shape of the image easily in `numpy`.

Setting working directory and files to save outputs in. This step creates the working directory and automatically sets up subfolders in the working directory, within the folder 'Images_to_process' which you should have already created (see the beginning of this code). All images you wish to process should be placed in the 'Images to process' folder. They will be automatically moved by the code as they are completed.

This code block creates sub folders where the photos are automatically placed depending on the status after they are processed. The folders are as follows:

complete_2dots: photos that were successfully processed but only 2 dots were found

complete_not_found: photos that were succesfully processed but no dots were found

complete: photos that were successfully processed and 3 dots were found

dots_only: photos cropped to only the dots that were found

gorilla_mask: photos showing gorillas with the background removed

laserdotfind: for photos in which 3 dots were found a colation of 4 photos. 1) the original photo 2) the the masked photos 3) all the pixels that were identified as possible lasers 4) the 3 selected laser spots and red marks where the centre of the lasers spots have been identified.

laserdotfind_2dots: for photos in which 2 dots were found a colation of 4 photos. 1) the original photo 2) the the masked photos 3) all the pixels that were identified as possible lasers 4) the 3 selected laser spots and red marks where the centre of the lasers spots have been identified.

laserdotfind_probs: for photos in which there were problems findings the dots, these photos could still be succesful but are worth checking.

[ ]:
```
i_obj=0
i_found=0
blank=pd.Series(['      '])
t0=time.time()
path = os.getcwd()
root_dir="Images_to_process"
if not os.path.exists(root_dir+'/complete_2dots'):
    os.makedirs(root_dir+'/complete_2dots')
```

```
if not os.path.exists(root_dir+'/complete_not_found'):
    os.makedirs(root_dir+'/complete_not_found')
if not os.path.exists(root_dir+'/complete'):
    os.makedirs(root_dir+'/complete')
if not os.path.exists(root_dir+'/dots_only'):
    os.makedirs(root_dir+'/dots_only')
if not os.path.exists(root_dir+'/gorilla_mask'):
    os.makedirs(root_dir+'/gorilla_mask')
if not os.path.exists(root_dir+'/laserdotfind'):
    os.makedirs(root_dir+'/laserdotfind')
if not os.path.exists(root_dir+'/laserdotfind_2dots'):
    os.makedirs(root_dir+'/laserdotfind_2dots')
if not os.path.exists(root_dir+'/laserdotfind_probs'):
    os.makedirs(root_dir+'/laserdotfind_probs')
    print(root_dir)
# End of setup.

# Go through working dir
df = pd.DataFrame()
#df.to_csv(root_dir+'/'+'gorilla_photo_analysis_py.csv')
#df.to_csv(root_dir+'/'+'gorilla_photo_dots_found.csv')
grnpass=np.array(['A','B','C','D','E','F','G','H','I'])
```

[ ]:
```
print(path,root_dir,5)
```

Setting initial parameters. Once you have run the code the first time, check any photos that were unsuccesfull, in the 'laserdotfind_probs' folder. Then any photos that didn't work you can remove back to the main folder, adjust these parameters and run them again. The new results will be apended to the same csv output file.

`GRfactor` is a parameter that increases the value of green/red ratio for selecting pixels to be laser-like: - 1 works well. - 1.5 is a very large value. - <0.7 are low enough to not be very selective, but will select fairly flat spots. - 0.8 will pick up a lot of foliage.

`dot_factor` is a parameter that sets a floor on the minimum value of green accepted to be a laser spot: - 1 is small, but acceptable for selecting green values as low as 70 / 256, will pick up a lot of foliage - 1.5 raises that floor to 105 / 256 is considered a high value, good for very white spots.

`min_dot` tells how many laseer spots are being looked for, this is set to 3 for all our images. - min_dot=2: when looking for 2 laser spots only

`factor` is the green scale factor (allowed green), lower value of the factors allows more green from the background. - If the fur has more red in it (i.e. brown primates, increase this number).

`scale` is the factor by which the non-green intensity is lowered on each loop through the images, until 2 consecutive loop yield no new results.

`jkk`is the number of times we loop through the (remaining) set of pictures, lowering the scale on each loop.

[ ]:
```
GRfactor=1.   # Use 1.0 for GW. Use 1.1 for Duke.

dot_factor=1.5 # Use 1.5 for GW. Use 1.0 for Duke.

min_dot=3 # Usual number for three laser spots.

max_cut=10000

factor=1.02/.96/.96/.96

scale=0.95 #0.94 #0.95 #0.96  #0.97 for Duke, .95 for GW.

jkk=9
```

0.2.3 Main Code

[ ]:
```
npass=5
dir_0=-(npass-1)
dir_n=1
dir_m=0
for jk in range(jkk):
    p_num=grnpass[jk]
    factor=factor*scale
    dir_l = os.listdir(root_dir)
    newlist = []
    for names in dir_l:
        if names.endswith(".JPG"):
            newlist.append(names)
    dir_l=newlist
    if dir_0==0:
        dir_n=0
        dir_m=0
    else:
        dir_n=size(dir_l)
    print(p_num+'th-pass',dir_l,'number of files =', dir_n,',',dir_m)

    # Go through working directory.
    df = pd.DataFrame()
    file_num=0
    if dir_n==dir_m:
        if(dir_0==0):
            dir_n=0
        else:
            dir_0=dir_0+1
    else:
        dir_0=-(npass-1)
    dir_m=dir_n
```

```
    print(dir_n,dir_m)
    for img_ind in range(dir_n):
        img_name=dir_l[img_ind]
        file_num=file_num+1
        coords= [[1,2,3,4,5,6,7,8]]

    # Create the pandas DataFrame.
        coords=pd.DataFrame(coords, columns = ['hor dist', 'vert dist', 'diag',␣
↪'calc hypot', 'hor wt',\
                                            'vert wt','diag wt', 'calc hypot␣
↪wt'])
        Flag=False
        E_code='__'
        box=array([[0,0],[1,1]])
        box=box.astype(int)
        imgg = Image.open(root_dir+"/"+img_name)
        img=np.array(imgg)
        test=1.0*np.ones(shape(img[:,:,0]))

    # First refine the mask of the gorilla, since that is 1) where the laser is␣
↪likely to be and to be useful,
    # and 2) there is high contrast with the green laser spot.

    # Define masks based on the lack of green in the gorilla fur.
        gr=img[:,:,1]
        bl=img[:,:,2]
        rd=img[:,:,0]
        min_cut=0.008*gr.size

    # Will make 3 passes with inceasing levels of green allowed. Being too␣
↪strict often results in a mask that is only
    # part of the gorilla
        a=np.logical_or.reduce([rd>(factor*gr),bl>(factor*gr)])
        black=np.logical_and.reduce([bl<70,gr<30,rd<70,bl>1,gr>1,rd>1,bl>1])
        gray=np.logical_and.reduce([bl<130,gr<130,rd<130,bl>100,gr>100,rd>100])
        greenr = np.logical_and.reduce([bl>40,gr>40,rd>40,a])   # Testing␣
↪against relative color levels.
        greenr=np.logical_or.reduce([greenr,black,gray])
        greenr=ndi.binary_fill_holes(greenr)  # Fill holes so the laser spot is␣
↪not excluded.
#    greenr=ndi.binary_dilation(greenr)  #apply dilation filter  --- # of␣
↪applications by trial and error
#    greenr=ndi.binary_dilation(greenr)  #apply dilation filter
#    greenr=np.logical_not(greenr)   # invert to switch frm green filter to not␣
↪green filter
```

```
        g_part,g_mask,gg_im_out,sizer=mask_image(np.
↪array(imgg),greenr,min_cut,gr.size) # Gorillas are always > 100000 pixels.
        n_ape=g_part
        ape_size=min(np.bincount(g_mask.ravel()))/gr.size
    # Mask photo to restrict analysis to gorilla.
        x,y=gg_im_out.size
        result=Image.new('RGB',(x*1,y*2))
        result.paste(imgg,(x*0,y*0))
        result.paste(gg_im_out,(x*0,y*1))
        skio.imsave(root_dir+"/"+"gorilla_mask/mask_"+p_num+img_name,np.
↪asanyarray(gg_im_out))  #save to file for checking

    # Define masks:
    # g_mask is the gorilla mask, gg_im_out is the image masked by the gorilla␣
↪(mostly!)

    # Now begin hunting for laser dots. Strategy will be to make a fairly␣
↪relaxed criteria on first pass
    # so that we catch all the laser dots + more green than we would like.
        gimg=np.array(gg_im_out)
        GRratio=np.divide(10.0*gimg[:,:,1],(gimg[:,:,0]+test))
        GBratio=np.divide(10.0*gimg[:,:,1],(gimg[:,:,2]+test))
        BRratio=np.divide(10.0*gimg[:,:,2],10.0*(gimg[:,:,0]+test))
        GRB=np.multiply(GRratio,GBratio)
        GRB_sum=gimg[:,:,1]+gimg[:,:,0]+gimg[:,:,2]
#     greenr = 1*(GRratio > 2) & 1*(img[:,:,1]>50)
#     greenr = 1*(GRratio > 2)&1*(GRratio < 200)&1*(GBratio > 1.7) & 1*(img[:,:
↪,1]>50)
#     greenr = 1*(GRratio > 2)&1*(GBratio>1.2)&1*(GBratio<2.5)+1*((img[:,:
↪,1]>240)\
#                             &(GBratio>1.5)&(GRratio > 2)) #Strict parameterss
        a = np.logical_and.reduce([(gimg[:,:,1]>dot_factor*100),(GRratio >␣
↪GRfactor*12),GBratio > GRfactor*12])
        b = np.logical_and.reduce([(gimg[:,:,1]>200),(GRratio >␣
↪GRfactor*12),GBratio > 11])
        c = np.logical_and.reduce([(gimg[:,:,1]>240),(GRratio >␣
↪GRfactor*11),GBratio > 10])
#         d = np.logical_and.reduce([(gimg[:,:,1]>120),(GRratio >␣
↪GRfactor*14),GBratio > 12])
        e = np.logical_and.reduce([(gimg[:,:,1]>dot_factor*70),(GRratio >␣
↪GRfactor*14),GBratio > 10])
        f = np.logical_and.reduce([(gimg[:,:,1]>250),(GRratio >=␣
↪GRfactor*10),GBratio >= 10])   # more for GW
        greenr=np.logical_or.reduce([a,b,c,e,f])   ###Looser parameters
#     a = np.logical_and.reduce([(gimg[:,:,1]>70),(GRB > 2.5)])
#     b = np.logical_and.reduce([(gimg[:,:,1]>200),(GRB > 1.3)])
```

```
#    c = np.logical_and.reduce([(gimg[:,:,1]>240),(GRB > 1.1)])
#        greenr=np.logical_or.reduce([b,c,d,e])   ###Looser parameters
#
    # Create objects based on GR ratio
        greenr=ndi.binary_dilation(greenr)  # Apply dilation filter.
        greenr=ndi.binary_dilation(greenr)  # Apply dilation filter.
        greenr=ndi.binary_fill_holes(greenr) # Fill holes.
        greenr=ndi.binary_erosion(greenr)   # Erode.
        g_part,g_mask,g_im_out,sizer=mask_image(np.
 ↪array(gimg),greenr,10,max_cut)

    # Create array of particles, defined by any arrangement of contiguous,␣
 ↪non-zero pixels.
        print("File=",img_name,file_num,"files out of␣
 ↪",dir_n,',',dir_m,"program found",g_part,'dots on first pass')

    # Now work with object properties. First pass, evaluated solely on GR ratio␣
 ↪levels of individual pixels.
        mask_c=g_mask
        num_obj,label_ratio,Propsgrn,PTabgrn=prop_gen(mask_c,gimg[:,:
 ↪,1],p_num+'_grn_int')
        num_obj,label_ratio,Props,PTab=prop_gen(mask_c,GRB,p_num+'_GRB_ratio')
        PTab['weighted_centroid-0']=PTabgrn['weighted_centroid-0']
        PTab['weighted_centroid-1']=PTabgrn['weighted_centroid-1']
        num_obji=num_obj

# Any manipulations are performed on the whole data frame, mask can be resized␣
 ↪to the
# original image (ratio mask) size and shape by the operation:␣
 ↪newmask=masks[g_mask]

# Check for and extract right triangle dot pattern
        PTabgrn=PTabgrn.sort_values(by='total_intensity', ascending=0)
        PTabgrn.reset_index(drop=True, inplace=True)
        PTabgrn.reset_index(drop=False, inplace=True) # Create column (first)␣
 ↪with index before modifying with filters.

# Write to file directory depending on results.
        if(num_obj>0):
            i_obj=i_obj+1
            if(i_obj==1):
                PTabgrn.to_csv(root_dir+'/'+'gorilla_photo_analysis_py.csv',␣
 ↪mode = 'a')
                blank.to_csv(root_dir+'/'+'gorilla_photo_analysis_py.csv', mode␣
 ↪= 'a')
            elif(i_obj>1):
```

```
                PTabgrn.to_csv(root_dir+'/'+'gorilla_photo_analysis_py.csv',␣
↪mode = 'a', header = False)
                blank.to_csv(root_dir+'/'+'gorilla_photo_analysis_py.csv', mode␣
↪= 'a', header = False)
        PTab=PTab.sort_values(by='total_intensity', ascending=0)
        PTab.reset_index(drop=True, inplace=True)
        PTab.reset_index(drop=False, inplace=True)

# dot_check tests whether a triplet of objects are positioned in the right␣
↪place to be a laser spot. It returns
# the bounding box for the laser spot as well as the centroid spacing in pixels.

# Write to file
        if(num_obj>0):
            PTab.to_csv(root_dir+'/'+'gorilla_photo_analysis_py.csv', mode =␣
↪'a', header = False)
            blank.to_csv(root_dir+'/'+'gorilla_photo_analysis_py.csv', mode =␣
↪'a', header = False)
        iter_num=0
        Flag=False
        if (g_part>=min_dot):
            tem=PTab.head(min_dot)
            iter_num=iter_num+1      #1
            box,coords,Flag,E_code,tem=dot_check(tem)
            PTab.drop(PTab.index[:min_dot], inplace=True)
            PTab=tem.append(PTab)
            if not Flag:

# Eliminate the non-circular dots
                area = [prop.area for prop in Props]
                per = [prop.perimeter for prop in Props]
                meanint=[prop.mean_intensity for prop in Propsgrn]
                bbox=[prop.bbox_area for prop in Props]
                circ = np.array(area) * math.pi * 4 / np.array(per) / np.
↪array(per)
                circ = np.array(area) * math.pi * 4 / np.array(per) / np.
↪array(per)
                circ[circ == inf] = 0
                a = circ > .4
                b =circ < 1.5
                circsqr=.25*math.pi*np.array(bbox) / np.array(area)
                c=circsqr < 1.5
                circularity=np.logical_and.reduce([a,b,c])
                print('circ min,max=',np.min(circ),np.max(circ),np.
↪min(circsqr),np.max(circsqr))
                circles = np.append([0], circularity)
```

```
                mask_c=circles[label_ratio]
                ␣
↪num_obj,label_ratio,Props,PTab=prop_gen(mask_c,GRB,p_num+'_GRB_ratio')
                num_obj,label_ratio,Propsgrn,PTabgrn=prop_gen(mask_c,gimg[:,:
↪,1],p_num+'_grn_int')
                num_obj,label_ratio,Propsred,PTabred=prop_gen(mask_c,gimg[:,:
↪,0],p_num+'_red_int')
                num_obj,label_ratio,Propsblu,PTabblu=prop_gen(mask_c,gimg[:,:
↪,2],p_num+'_blu_int')
                PTab['GBR_ratio']=PTabgrn['mean_intensity']/(.
↪5*PTabred['mean_intensity']+.5*PTabblu['mean_intensity'])
                PTab['mean_green']=PTabgrn['mean_intensity']
                ␣
↪PTab['mean_bright']=PTabgrn['mean_intensity']+PTabred['mean_intensity']+PTabblu['mean_inten
                PTab['weighted_centroid-0']=PTabgrn['weighted_centroid-0']
                PTab['weighted_centroid-1']=PTabgrn['weighted_centroid-1']
                PTab=PTab.sort_values(by='total_intensity', ascending=0) # Sort␣
↪by intensity
                PTab.reset_index(drop=True, inplace=True)
                PTab.reset_index(drop=False, inplace=True)    # Create column␣
↪(first) with index before modifying with filters.
                PTabgrn=PTabgrn.sort_values(by='total_intensity', ascending=0) ␣
↪ # Sort df by intensity.
                PTabgrn.reset_index(drop=True, inplace=True)  # Reindex to␣
↪match sorting order.
                PTabgrn.reset_index(drop=False, inplace=True)    # Create␣
↪column (first) with index before modifying with filters.
                tem=PTab.head(min_dot)
                iter_num=iter_num+1 #2
                # eliminate this check ?   redundant with first one in the␣
↪cascading series of filters
                box,coords,Flag,E_code,tem=dot_check(tem)
                PTab.drop(PTab.index[:min_dot], inplace=True)
                PTab=tem.append(PTab)
                PTabgrn.to_csv(root_dir+'/'+'gorilla_photo_analysis_py.csv',␣
↪mode = 'a', header = False)
                blank.to_csv(root_dir+'/'+'gorilla_photo_analysis_py.csv', mode␣
↪= 'a')
                PTab.to_csv(root_dir+'/'+'gorilla_photo_analysis_py.csv', mode␣
↪= 'a', header = False)
                blank.to_csv(root_dir+'/'+'gorilla_photo_analysis_py.csv', mode␣
↪= 'a')
                if not Flag:
                # Eliminate small area dots
                    tem=PTab.head(4)
                    tst=np.min(tem['area'])/5
```

```
                    PTab = PTab[PTab['area'] >= tst]
                    PTab.reset_index(drop=True, inplace=True)
                    # Sort by intensity
                    PTab=PTab.sort_values(by='total_intensity', ascending=0)
                    PTab.reset_index(drop=True, inplace=True)
                    tem=PTab.head(min_dot)
                    iter_num=iter_num+1 #3
                    box,coords,Flag,E_code,tem=dot_check(tem)
                    PTab.drop(PTab.index[:min_dot], inplace=True)
                    PTab=tem.append(PTab)
                    PTab.to_csv(root_dir+'/'+'gorilla_photo_analysis_py.csv',␣
↪mode = 'a', header = False)
                    blank.to_csv(root_dir+'/'+'gorilla_photo_analysis_py.csv',␣
↪mode = 'a')
                    if not Flag:
                # Eliminate LARGE bbox object.
                        PTab = PTab[PTab['circsqr'] <= 1.35]
                        PTab.reset_index(drop=True, inplace=True)
                # Sort by total intensity.
                        PTab=PTab.sort_values(by='total_intensity', ascending=0)
                        PTab.reset_index(drop=True, inplace=True)
                        tem=PTab.head(min_dot)
                        iter_num=iter_num+1 #4
                        box,coords,Flag,E_code,tem=dot_check(tem)
                        PTab.drop(PTab.index[:min_dot], inplace=True)
                        PTab=tem.append(PTab)
                        PTab.to_csv(root_dir+'/'+'gorilla_photo_analysis_py.
↪csv', mode = 'a', header = False)
                        blank.to_csv(root_dir+'/'+'gorilla_photo_analysis_py.
↪csv', mode = 'a')
                        if not Flag:
                # Sort by mean intensity.
                            PTab=PTab.sort_values(by='mean_intensity',␣
↪ascending=0)
                            PTab.reset_index(drop=True, inplace=True)
                            tem=PTab.head(min_dot)
                            iter_num=iter_num+1    #5
                            box,coords,Flag,E_code,tem=dot_check(tem)
                            PTab.drop(PTab.index[:min_dot], inplace=True)
                            PTab=tem.append(PTab)
                            PTab.to_csv(root_dir+'/'+'gorilla_photo_analysis_py.
↪csv', mode = 'a', header = False)
                            blank.to_csv(root_dir+'/
↪'+'gorilla_photo_analysis_py.csv', mode = 'a')
                            if not Flag:
                # Sort by intensity ratio.
```

```
                                PTab=PTab.sort_values(by='intensity_ratio',␣
↪ascending=0)
                                PTab.reset_index(drop=True, inplace=True)
                                tem=PTab.head(min_dot)
                                iter_num=iter_num+1    #6
                                box,coords,Flag,E_code,tem=dot_check(tem)
                                PTab.drop(PTab.index[:min_dot], inplace=True)
                                PTab=tem.append(PTab)
                                PTab.to_csv(root_dir+'/
↪'+'gorilla_photo_analysis_py.csv', mode = 'a', header = False)
                                blank.to_csv(root_dir+'/
↪'+'gorilla_photo_analysis_py.csv', mode = 'a')
                                if not Flag:
                # Sort by mean(G)/(mean(R) +mean(B)).
                                    PTab=PTab.sort_values(by='GBR_ratio',␣
↪ascending=0)
                                    PTab.reset_index(drop=True, inplace=True)
                                    tem=PTab.head(min_dot)
                                    iter_num=iter_num+1    #7
                                    box,coords,Flag,E_code,tem=dot_check(tem)
                                    PTab.drop(PTab.index[:min_dot],␣
↪inplace=True)
                                    PTab=tem.append(PTab)
                                    PTab.to_csv(root_dir+'/
↪'+'gorilla_photo_analysis_py.csv', mode = 'a', header = False)
                                    blank.to_csv(root_dir+'/
↪'+'gorilla_photo_analysis_py.csv', mode = 'a')
                                    if not Flag:
                # Sort by mean_green.
                                        PTab=PTab.sort_values(by='mean_green',␣
↪ascending=0)
                                        PTab.reset_index(drop=True,␣
↪inplace=True)
                                        tem=PTab.head(min_dot)
                                        iter_num=iter_num+1      #8
                                    ␣
↪box,coords,Flag,E_code,tem=dot_check(tem)
                                        PTab.drop(PTab.index[:min_dot],␣
↪inplace=True)
                                        PTab=tem.append(PTab)
                                        PTab.to_csv(root_dir+'/
↪'+'gorilla_photo_analysis_py.csv', mode = 'a', header = False)
                                        blank.to_csv(root_dir+'/
↪'+'gorilla_photo_analysis_py.csv', mode = 'a')
                                        if not Flag:
                # Sort by mean_bright.
```

```
                                                PTab=PTab.
↪sort_values(by='mean_bright', ascending=0)
                                                PTab.reset_index(drop=True,␣
↪inplace=True)
                                                tem=PTab.head(min_dot)
                                                iter_num=iter_num+1      #9
                                               ␣
↪box,coords,Flag,E_code,tem=dot_check(tem)
                                                PTab.drop(PTab.index[:min_dot],␣
↪inplace=True)
                                                PTab=tem.append(PTab)
                                                PTab.to_csv(root_dir+'/
↪'+'gorilla_photo_analysis_py.csv', mode = 'a', header = False)
                                                blank.to_csv(root_dir+'/
↪'+'gorilla_photo_analysis_py.csv', mode = 'a')
                                                if not Flag:
                # Sort by circularity.
                                                    PTab=PTab.
↪sort_values(by='circularity', ascending=0)
                                                    PTab.reset_index(drop=True,␣
↪inplace=True)
                                                    tem=PTab.head(min_dot)
                                                    iter_num=iter_num+1      #10
                                                   ␣
↪box,coords,Flag,E_code,tem=dot_check(tem)
                                                    PTab.drop(PTab.index[:min_dot],␣
↪inplace=True)
                                                    PTab=tem.append(PTab)
                                                    PTab.to_csv(root_dir+'/
↪'+'gorilla_photo_analysis_py.csv', mode = 'a', header = False)
                                                    blank.to_csv(root_dir+'/
↪'+'gorilla_photo_analysis_py.csv', mode = 'a')
                                                    if not Flag:
                # Sort by area.
                                                        PTab=PTab.
↪sort_values(by='area', ascending=0)
                                                        PTab.reset_index(drop=True,␣
↪inplace=True)
                                                        tem=PTab.head(min_dot)
                                                        iter_num=iter_num+1      #11
                                                       ␣
↪box,coords,Flag,E_code,tem=dot_check(tem)
                                                        PTab.drop(PTab.index[:
↪min_dot], inplace=True)
                                                        PTab=tem.append(PTab)
```

```
                                                    PTab.to_csv(root_dir+'/
↪'+'gorilla_photo_analysis_py.csv', mode = 'a', header = False)
                                                    blank.to_csv(root_dir+'/
↪'+'gorilla_photo_analysis_py.csv', mode = 'a')
        nr,tc=shape(PTab)
        print('There are ',nr,' objects after the last measureprops operation')


# Clean up and write to file.
        if Flag:
            tr,tc=PTab.shape
            s=min(6,tr)
            for i in range(s):
                x=PTabgrn.loc[PTabgrn['label'] == PTab.at[i,'label']].index[0]
                PTabgrn.at[PTabgrn.loc[PTabgrn['label'] == PTab.at[i,'label']].
↪index[0],'index']=i-s
            PTabgrn=PTabgrn.sort_values(by='index', ascending=1)
            PTabgrn.reset_index(drop=True, inplace=True)
            tem=PTabgrn.head(min_dot)
            box,coords,Flag0,E_code0,tem=dot_check(tem)
            PTabgrn.drop(PTabgrn.index[:min_dot], inplace=True)
            PTabgrn=tem.append(PTabgrn)
            tr,tc=tem.shape
            areatest=max(tem['area'])
            coords['pic area']=gr.size
            coords['pass_num'] = p_num
            coords['iter_num'] = [iter_num]
            coords['el. time']=[time.time()-t0]
            box_area=(box[1,0]-box[0,0])*(box[1,1]-box[0,1])
            coords['box_area']=[box_area]
            if(box_area/areatest>400):
                Flag=False
                E_code='BA'
            coords['Laser_found']=Flag
            coords['Error']=E_code
        print('preflash',Flag,E_code)

    # Annotate figure with labels
        x,y=g_im_out.size
        image=g_im_out   #result
        idraw = ImageDraw.Draw(image)
        font = ImageFont.truetype('KOI8_FIN.ttf', int(y/50))  # Specified font␣
↪size.
        fs1=int(y/500)
        for ij in np.arange(nr):
            font = ImageFont.truetype('KOI8_FIN.ttf', int(y/50))  # Specified␣
↪font size.
```

```
            idraw.text(((PTab['centroid-1'][ij]+15), PTab['centroid-0'][ij]),␣
↪"{:.0f}".format(PTab['label'][ij]),fill='white',font=font)
            font = ImageFont.truetype('KOI8_FIN.ttf', fs1)  # Specified font␣
↪size.
            idraw.text(((PTab['centroid-1'][ij]-int(fs1/3.5)),␣
↪PTab['centroid-0'][ij]-int(fs1/2.5)), "o",fill='red',font=font)
            idraw.text(((PTab['weighted_centroid-1'][ij]-int(fs1/3.5)),␣
↪PTab['weighted_centroid-0'][ij]-int(fs1/2.5)), "x",fill='red',font=font)
    # Images.
        x,y=g_im_out.size
        image=g_im_out    #result
        idraw = ImageDraw.Draw(image)
        font = ImageFont.truetype('KOI8_FIN.ttf', int(y/20))  # Specified font␣
↪size.
        idraw.text((int(.75*x), int(.9*y)), "{:.0f}".format(g_part)+"␣
↪dots",fill='white',font=font)
        result=Image.new('RGB',(x*2,y*2))
        result.paste(imgg,(x*0,y*0))
        result.paste(g_im_out,(x*0,y*1))
        result.paste(gg_im_out,(x*1,y*0))
        if not Flag:
            result.paste(gg_im_out,(x*1,y*1))
            skio.imsave(root_dir+"/"+"laserdotfind/result_"+img_name,np.
↪asanyarray(result))
        elif Flag:

    # Get rid of extra green flares that surround very bright laser spots.
    # Create im_box here as a bbox of the above fig, without the numbers  ??or␣
↪maybe use the one below

            im_box=Image.fromarray(img)
            xx=np.array(imgg)
            grnmean=np.mean(xx[box[0,1]:box[1,1],box[0,0]:box[1,0],1])
            grnstd=np.std(xx[box[0,1]:box[1,1],box[0,0]:box[1,0],1])
            grnmean=grnmean+2*grnstd
            GRBmean=np.mean(GRB[box[0,1]:box[1,1],box[0,0]:box[1,0]])
            GRBstd=np.std(GRB[box[0,1]:box[1,1],box[0,0]:box[1,0]])
            if (np.average(tem['area'])>1000):
                im_box_t=im_box
                img1 = ImageDraw.Draw(im_box_t)
                img1.rectangle([box[0,0],box[0,1],box[1,0],box[1,1]], fill␣
↪="red", outline ="red")
                box1,Flag,E_code,coords,tem=clean_flash(im_box_t,Image.
↪fromarray(img),tem,GRfactor,min_dot,GRBmean,grnmean)
                print(Flag,E_code)
                if (E_code=='IR'):
```

```
                    E_code='NA'
            im_box=Image.fromarray(img)
            idraw = ImageDraw.Draw(im_box)
            font = ImageFont.truetype('KOI8_FIN.ttf', fs1)  # Specified font␣
↪size.
            tr,tc=tem.shape
            for ij in np.arange(tr):
                idraw.text(((tem['centroid-1'][ij]-int(fs1/3.5)),␣
↪tem['centroid-0'][ij]-int(fs1/2.5)), "o",fill='red',font=font)
                idraw.text(((tem['weighted_centroid-1'][ij]-int(fs1/3.5)),␣
↪tem['weighted_centroid-0'][ij]-int(fs1/2.5)), "x",fill='red',font=font)
            im_box=im_box.crop((box[0,0],box[0,1],box[1,0],box[1,1]))
            im_box = im_box.resize(g_im_out.size)
            result.paste(im_box,(x*1,y*1))
            skio.imsave(root_dir+"/dots_only/dots_"+img_name,np.
↪asanyarray(im_box))
            skio.imsave(root_dir+"/laserdotfind/result_"+img_name,np.
↪asanyarray(result))
            print(grnmean,grnstd,GRBmean,GRBstd)
            if Flag:
                print('postflash',Flag,E_code)
    # Result
                areatest=max(tem['area'])
                coords['pic area']=gr.size
                coords['pass_num'] = p_num
                coords['iter_num'] = [iter_num]
                coords['el. time']=[time.time()-t0]
                box_area=(box[1,0]-box[0,0])*(box[1,1]-box[0,1])
                coords['box_area']=[box_area]
                if(box_area/areatest>400):
                    Flag=False
                    E_code='BA'
                coords['Laser_found']=Flag
                coords['Error']=E_code
                print(grnmean,GRBmean)

    # Move figure to complete folder if 3 dots are found.
                if (jk==0):
                    gg1=gg_im_out
                    gg2=gg_im_out
                elif (jk==1):
                    gg2=gg_im_out
                    gg1=Image.open(root_dir+"/"+"gorilla_mask/mask_A"+img_name)
                elif (jk>1):
                    gg1=Image.open(root_dir+"/"+"gorilla_mask/
↪mask_"+grnpass[jk-2]+img_name)
```

```
                    gg2=Image.open(root_dir+"/"+"gorilla_mask/
↪mask_"+grnpass[jk-1]+img_name)

    # Create composite of sequentially more green-allowed primate images.
                x,y=imgg.size
                result=Image.new('RGB',(x*2,y*2))
                result.paste(imgg,(x*0,y*0))
                result.paste(gg1,(x*1,y*0))
                result.paste(gg2,(x*0,y*1))
                result.paste(gg_im_out,(x*1,y*1))

    # Result.
                skio.imsave(root_dir+"/"+"gorilla_mask/mask_"+img_name,np.
↪asanyarray(result))
            if (tr<min_dot):
                E_code='NA2'
                coords['Error']=E_code
            if (E_code=='NA'):
                Flag=True
                dest = sh.move(root_dir+"/"+img_name,root_dir+"/"+"complete/
↪"+img_name)
            elif (jk>3):
                dest = sh.move(root_dir+"/"+"laserdotfind/
↪result_"+img_name,root_dir+"/"+"laserdotfind_probs/result_"+img_name)
                dest = sh.move(root_dir+"/"+img_name,root_dir+"/
↪"+"complete_not_found/"+img_name)
            if ((E_code=='NA')&(tr==2)):
                dest = sh.move(root_dir+"/"+"laserdotfind/
↪result_"+img_name,root_dir+"/"+"laserdotfind_2dots/result_"+img_name)
                dest = sh.move(root_dir+'/complete/'+img_name,root_dir+"/
↪"+"complete_2dots/"+img_name)
            i_found=i_found+1
            coords.insert(0, "num_apes", n_ape, True)
            coords.insert(0,"ape_size",ape_size,True)
            coords.insert(0, "FileName", [img_name], True)
            coords.insert(0,"directory",[root_dir],True)
            if(i_found==1):
                coords.to_csv(root_dir+'/'+'gorilla_photo_dots_found.csv', mode␣
↪= 'a')
            elif(i_found>1):
                coords.to_csv(root_dir+'/'+'gorilla_photo_dots_found.csv', mode␣
↪= 'a', header = False)

# Record if no dots are found, if on the last round.
        if (jk==jkk-1):
            if not Flag:
```

```
                #  dest = sh.move(root_dir+"/"+img_name,root_dir+"/
↪"+"complete_not_found/"+img_name)
                coords.insert(0, "num_apes", n_ape, True)
                coords.insert(0,"ape_size",ape_size,True)
                coords.insert(0, "FileName", [img_name], True)
                coords.insert(0,"directory",[root_dir],True)
                tr,tc=tem.shape
                areatest=0
                coords['Laser_found']=Flag
                coords['Error']=E_code
                coords['pic area']=gr.size
                coords['pass_num'] = p_num
                coords['iter_num'] = [iter_num]
                coords['el. time']=[time.time()-t0]
                box_area=(box[1,0]-box[0,0])*(box[1,1]-box[0,1])
                coords['box_area']=[box_area]
                i_found=i_found+1
                if(i_found==1):
                    coords.to_csv(root_dir+'/'+'gorilla_photo_dots_found.csv',␣
↪mode = 'a')
                elif(i_found>1):
                    coords.to_csv(root_dir+'/'+'gorilla_photo_dots_found.csv',␣
↪mode = 'a', header = False)
```

0.2.4 Final output

Once succesfully run, all photos will be moved in to their respective folders and two csv files will be generated.

gorilla_photo_dots_found: is a csv file with the distance recorded between the 3 laser spots. The horizontal distance is in the 'hor dist' column and the vertical is in the 'vert dist' column. The excel file also records the weighted values e.g. 'hort wt' and 'vert wt'. We found that the weighted values were more similar to manual measures.

gorilla_photo_analysis_py: is a csv file recording all the details of every dot found in each photo