AMATH 582 Homework 3

Yiyun Dong

Department of Physics
University of Washington
Email: yiyund@uw.edu
Github: yeewantung

Abstract

In this project, we process measurement data on 4 different motions of the paint can by 3 cameras. We first extract the trajectory of a particular point on the paint can. By analyzing the trajectories of measurements with PCA, we extract the information of the uncorrelated modes in the motion of the paint can.

1 Introduction and Overview

Principal component analysis (PCA) is a statistical procedure that uses an orthogonal transformation to convert a set of observations of possibly correlated variables into a set of values of linearly uncorrelated variables called principal components [6]. It is proven to be useful in data analysis, quantitative finance and neuroscience.

There are many methods for implementing PCA in reality. One idea is to the covariance matrix method and use an orthogonal transformation to diagonalize the covariance matrix. The orthogonal transformation can be singular value decomposition (SVD) when dealing with arbitrary complex matrix. It can also be eigenvalue decomposition when dealing with a real matrix with full rank.

In this project, we implement PCA with SVD to analyze measurement data on 4 different motions of the paint can by 3 cameras. We first extract the trajectory of a particular point on the paint can. By analyzing the trajectories of measurements with PCA, we extract the information of the uncorrelated modes in the motion of the paint can. The theory of PCA and SVD are further explained in Section 2.

2 Theoretical Background

2.1 Singular Value Decomposition (SVD)

As we learned from the lecture, every matrix $A \in \mathbb{C}^{m\times n}$ has a singular value decomposition (SVD),

$$A = U\Sigma V^*, \tag{1}$$

where U and V are two unitary matrix, and $Sigma$ is a diagonal matrix. The diagonal elements of Σ is called the singular values of matrix A. The column vectors of U and V are called left and right singular vectors respectively [**tft**, 1, 7]. Therefore, we can also write matrix A in a sum of rank-1 matrices,

$$A = \sum_{j=1}^{r} \sigma_j u_j v_j^*, \tag{2}$$

here r is the rank of matrix A.

In fact, SVD provides best approximation of a matrix A by matrices of lower rank [**tft**]. For $0 \le \nu \le r$, define

$$A_\nu = \sum_{j=1}^{\nu} \sigma_j u_j v_j^*, \tag{3}$$

then,

$$\|A - A_\nu\|_2 = \inf_{\substack{B \in \mathbb{C}^{m \times n} \\ rank(B) \leq \nu}} \|A - B\|_2 = \sigma_{\nu+1}. \tag{4}$$

Furthermore, the singular values σ_i are uniquely determined. If A is a square matrix and the singular values σ_i are distinct, the left and right singular vectors u_j and v_j are uniquely determined up to complex signs [**tft**]. This explains the existence and uniqueness of SVD for an arbitrary matrix.

2.2 Principal Component Analysis (PCA)

Principal component analysis (PCA) involves an orthogonal transformation to convert a set of observations of possibly correlated variables into a set of values of linearly uncorrelated variables called principal components [6]. Since the nature of SVD is a change of basis and it involves two unitary matrices U and V, so it is a natural orthogonal transformation for us to conduct PCA.

Consider a set of measurements of correlated variables,

$$X = \begin{pmatrix} x1 \\ x2 \\ \vdots \\ x_m \end{pmatrix}, \tag{5}$$

here x_j is a row vector representing a time sequence of some variable. Then we can find out how two measurements x_i and x_j are correlated by computing their covariance,

$$\sigma_{ij}^2 = \frac{1}{n-1} x_i x_j^T. \tag{6}$$

Also we can find out the amount of the dynamics of interest being captured for a certain measurement x_i by computing its variance,

$$\sigma_i^2 = \frac{1}{n-1} x_i x_i^T. \tag{7}$$

A covariance matrix can be constructed with covariances and variances of this set of measurements,

$$C_X = \frac{1}{n-1} X X^T. \tag{8}$$

In fact, the left unitary matrix U of SVD of the measurement matrix $X = U\Sigma V^*$ is the orthogonal transformation that transforms X into a set of uncorrelated variables Y, namely, $Y = U^* X$. We can justify this statement by looking at the covariance matrix of Y,

$$\begin{aligned} C_Y &= \frac{1}{n-1} Y Y^T \\ &= \frac{1}{n-1} U^* U \Sigma V^* V \Sigma^* U U^* \\ &= \frac{1}{n-1} \Sigma^2, \end{aligned} \tag{9}$$

Since the off-diagonal elements of C_Y are all zero, it means that each row of Y is uncorrelated to others. Then

$$Y = U^* X = \Sigma V^* \tag{10}$$

gives a way to obtain the uncorrelated variables, by relating right singular vectors of X to Y.

In conclusion, if we conduct a SVD on a set of measurements, we'll separate independent modes and extract the variance of each mode, which is shown useful in motion analysis [1].

3 Algorithm Implementation and Development

1. For an experiment, import the video recording the motion of object as several sets of data.
2. Crop and synchronize these video data so that the sets of data have the same length.
3. Trim background of frames in the sets of data to avoid the disturbance of background.
4. Extract the 2-dimensional trajectory of a particular point from each set of data, by using algorithm 1, 2, 3.
5. Subtract the mean of each trajectory for further calculation of variance, and stack the centered trajectories into a matrix A.
6. Compute the SVD of matrix A. Obtain the singular values, which are the square root of variances. Obtain the first two right singular vectors of A, which are the leading modes of motion of the object.
7. Plot the singular values and the modes of motion.

Algorithm 1: Extract the trajectory of a particular point from data w/o window

Fetch data from input
for Frame j in video data **do**
  Find maximum intensity point in frame j
  Save the point in the trajectory `trajx, trajy`
end for
return trajx, trajy

Algorithm 2: Extract the trajectory of a particular point from data w/ window

Fetch data, window size from input
Find maximum intensity point in frame 0
Save the point in the trajectory `trajx, trajy`
for Frame j in video data **do**
  Find maximum intensity point within the window centered at last trajectory point
  Save the point in the trajectory `trajx, trajy`
end for
return trajx, trajy

Algorithm 3: Approximate the trajectory when the point tracked is hidden

Fetch data, original trajectory extracted from input
Specify the color c of area closed to the point tracked
for Frame j need to be approximated in video data **do**
  Find the point that has the closest color to color c
  Update the point in the trajectory `trajx, trajy`
end for
return trajx, trajy

Algorithm 4: PCA on the motion of object

Import several sets of data of an experiment from video
Crop and synchronize sets of data
Trim background of each frame in sets of data
Extract the 2-dimensional trajectory of a particular point from each set of data
Subtract the mean of each trajectory
Stack the centered trajectories into a matrix A
Compute the SVD of A
Plot the singular values and first two right singular vectors of A

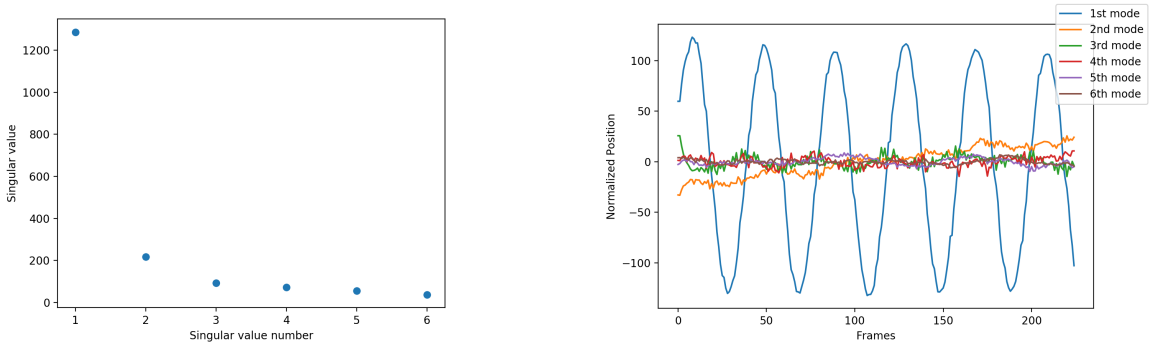
4 Computational Results

We are provided 4 tests corresponding to 4 different motions of the paint can. PCA are conducted on each test to extract the information of the motion of the paint can. The algorithm for PCA is listed in Algorithm 4.

For Test 1,2,3, the trajectories of the paint can are captured and recorded by tracking the trajectory of the flashlight placed on the top of the paint can. The algorithm to track the flashlight is Algorithm 2. For Test 4, the flashlight of the paint can could not be tracked for the entire time sequence. Thus, we implement Algorithm 1 and 3 to approximate the trajectory when the flashlight is not able to see.

4.1 Test 1

In Test 1, the paint can is oscillating in the vertical direction. There was very little noise in the data and the flashlight was in plain view for the entirety of all three videos, so it is possible accurately extract the paint can's position by tracking the flashlight. The results of SVD on Test 1 are shown in Figure 1. We can notice that, there is one leading harmonic mode that has variance much larger than the other modes. At the mean time, the leading harmonic mode has a fixed period of about 50 frames, whereas other modes corresponding other right singular vectors are basically noise.



(a) Square root of variance σ_i captured by modes

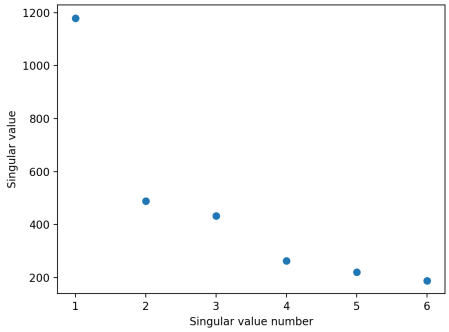
(b) Uncorrelated modes obtained by PCA

Figure 1: Motion decompostion of Test 1 by PCA

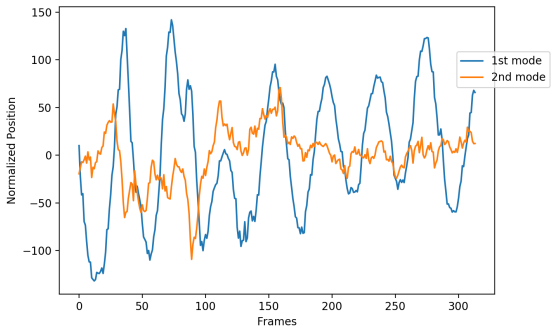
4.2 Test 2

In Test 2, the paint can is still oscillating in the vertical direction. However, the noise is introduced by shaking the cameras during measurements. We are not able to extract the paint can's position as accurate as before by tracking the flashlight. But SVD still works for this case, except for there is a considerable variance contributed by noise. The results of SVD on Test 2 are shown in Figure 2. We can notice that, there is one leading harmonic mode and a noise mode corresponding to shaking that have variance much

larger than the other modes. At the mean time, the leading harmonic mode still has a nearly uniform period, whereas other modes corresponding other right singular vectors are basically noise.



(a) Square root of variance σ_i captured by modes

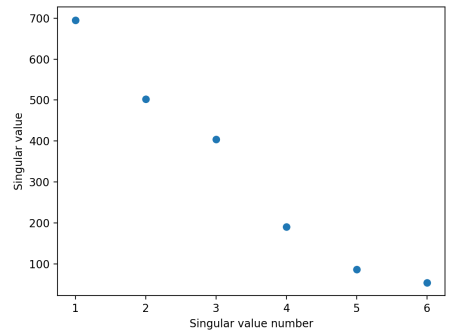


(b) First two uncorrelated modes obtained by PCA

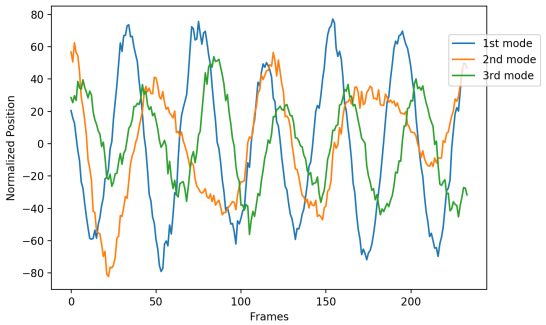
Figure 2: Motion decompostion of Test 2 by PCA

4.3 Test 3

In Test 3, the paint can is oscillating in the vertical direction and experiencing harmonic motions horizontally at the same time. The results of SVD on Test 3 are shown in Figure 3. We can notice that, there is three harmonic modes that have variance much larger than the rest of modes. At the mean time, the leading harmonic mode still has a uniform period of 50 frames, whereas other harmonic modes have different periods since they are representing horizontal motion.



(a) Square root of variance σ_i captured by modes



(b) First three uncorrelated modes obtained by PCA

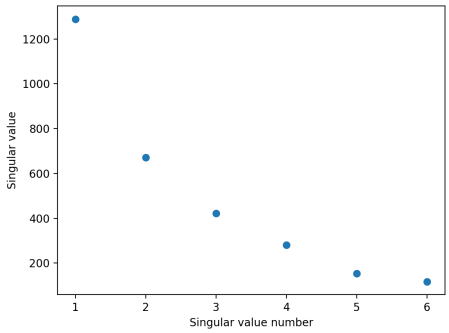
Figure 3: Motion decompostion of Test 3 by PCA

4.4 Test 4

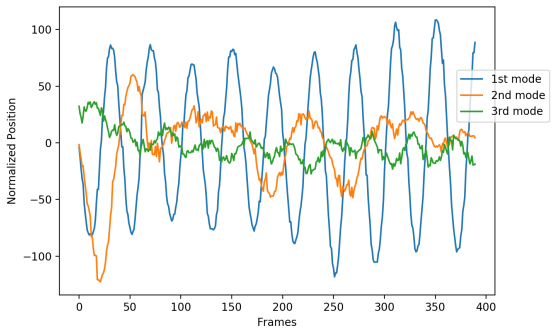
In Test 4, the paint can is oscillating in the vertical direction and experiencing self-rotation motion horizontally at the same time. We are not able to extract the paint can's position for the entire time sequence by tracking the flashlight. Instead, we notice that the flashlight handle has a unique color in the image. The position of the flashlight handle can be tracked by identifying its color and finding the closest color to it in each frame. The trajectory of the flashlight can be approximated by the position of the flashlight handle, when the flashlight is being rotated to the back of the paint can.

SVD still works for this case, except for there is a considerable variance contributed by noise, due to the incontinuity when merging two different trajectories we are tracking. The results of SVD on Test 4 are

shown in Figure 4. We can notice that, there is one leading harmonic modes that has variance much larger than the rest of modes, which is corresponding to the vertical oscillation. There is also a noticable second mode that has variance of $600 * 600$, which is corresponding to the in-homogeneous self-rotation. Notice that the leading harmonic mode has a uniform period of 40 frames, and the self-rotation mode is not uniform just as what we see in the video, whereas other modes are basically noise.



(a) Square root of variance σ_i captured by modes



(b) First three uncorrelated modes obtained by PCA

Figure 4: Motion decompostion of Test 4 by PCA

5 Summary and Conclusions

In this project, we use PCA to analyze the motion of 4 tests of the paint can. We render that PCA is useful in extracting the variances and uncorrelated modes in the motion of the paint can. The modes obtained by PCA are in fact representing the real motion of the paint can in 4 tests, which can be seen by simply playing the videos.

However, it took me a huge amount of time to track the motion of the paint can, especially for Test 4, when the tracking point is hidden sometimes. For compensation, another point is tracked instead and used to approximate the position of the tracking point during the time the tracking point is hidden. However, in reality, this requires lots of manual work. Algorithms for motion detection and tracking are especially an important basis for conducting PCA afterwards.

References

[1] Jose Nathan Kutz. *Data-driven modeling & scientific computation: methods for complex systems & big data*. Oxford University Press, 2013.

[2] *Matplotlib Reference*. URL: `https://matplotlib.org/api/_as_gen/matplotlib.pyplot.html`.

[3] *NumPy Reference*. URL: `https://docs.scipy.org/doc/numpy/reference`.

[4] *OpenCV Reference*. URL: `https://docs.opencv.org/3.4/index.html`.

[5] *SciPy Reference*. URL: `https://docs.scipy.org/doc/scipy/reference/index.html`.

[6] *Wikipedia - Principal component analysis*. URL: `https://en.wikipedia.org/wiki/Principal_component_analysis`.

[7] *Wikipedia - Singular value decomposition*. URL: `https://en.wikipedia.org/wiki/Singular_value_decomposition`.

Appendix A Python Functions used

Below are the Python functions implemented in the code with a brief explanation.

- `numpy.zeros(shape, dtype=float, order='C')` returns a new array of given shape and type, filled with zeros [3].
- `numpy.unravel_index(indices, shape, order='C')` converts a flat index or array of flat indices into a tuple of coordinate arrays [3].
- `numpy.argmax(a, axis=None, out=None)` returns the indices of the maximum values along an axis [3].
- `numpy.copy(a, order='K')` returns an array copy of the given object [3]. When assigning the values of a matrix A to another matrix B and changing the matrix elements of B, this function is used to avoid the change in the same elements of A.
- `numpy.linalg.norm(x, ord=None, axis=None, keepdims=False)` returns matrix or vector norm [3].
- `numpy.stack(arrays, axis=0, out=None)` joins a sequence of arrays along a new axis [3].
- `numpy.concatenate((a1, a2, ...), axis=0, out=None)` joins a sequence of arrays along an existing axis [3].
- `numpy.linalg.svd(a, full_matrices=True, compute_uv=True, hermitian=False)` computes Singular Value Decomposition [3].
- `numpy.flip(m, axis=None)` reverses the order of elements in an array along the given axis [3].
- `numpy.reshape(a, newshape, order='C')` gives a new shape without changing the data [3].
- `numpy.unique(ar, axis=None)` finds the unique elements of an array [3].
- `scipy.io.loadmat(file_name, mdict=None, appendmat=True, **kwargs)` loads MATLAB file [5].
- `matplotlib.pyplot.imshow(X, cmap=None)` displays an image, i.e. data on a 2D regular raster [2].
- `matplotlib.pyplot.scatter(x, y)` displays a scatter plot of y vs x with varying marker size and/or color [2].
- `cv2.cvtColor(src, bwsrc, cv::COLOR_RGB2GRAY)` converts an image from one color space (RGB) to another (grayscale) [4].

Appendix B Python Codes

```
#!/usr/bin/env python
# coding: utf-8

# In[1]:


# import packages
from scipy.io import loadmat
import numpy as np
import matplotlib.pyplot as plt
import cv2


# In[244]:
```

```
get_ipython().run_line_magic('matplotlib', 'notebook')
get_ipython().run_line_magic('matplotlib', 'notebook')


# # Function Definition

# In[3]:


def loadgsdata(filename, variable = None):
    '''
    Load .mat file and turn it into grayscale
    filename: (STRING) file name of .mat file
    variable: (STRING) specific variable name to import in .mat file
    Return value
    data_gray: (NUMPY ARRAY) garyscale image/video data
    '''

    # load mat
    if variable == None:
        data = loadmat(filename)
    else:
        data = loadmat(filename)[variable]

    # grab dimensions of data
    idx = data.shape
    idx_gray = (idx[0], idx[1], idx[-1])
    data_gray = np.zeros(idx_gray, dtype = np.uint8)

    # turn data into grayscale
    for i in range(idx[-1]):
        data_gray[:,:,i] = cv2.cvtColor(data[:,:,:,i], cv2.COLOR_RGB2GRAY)

    return data_gray


# In[58]:


def bg_darken(y, edge_l, edge_r, edge_u, edge_d):
    '''
    Eliminate a portion of background manually
    y: (NUMPY ARRAY) data of a set of images
    edge_l: (INT) pixel # of left edge
    edge_r: (INT) pixel # of right edge
    edge_u: (INT) pixel # of upper edge
    edge_d: (INT) pixel # of lower edge
    Return value
    yn: (NUMPY ARRAY) background-darkened data
    '''
    # grab dimensions of data
    idx = y.shape # idx[0]: height, idx[1]: width
```

```
    # in case the input is not integer
    edge_l = int(edge_l)
    edge_r = int(edge_r)
    edge_u = int(edge_u)
    edge_d = int(edge_d)

    # determine the effectiveness of edges
    if (edge_l >= edge_r) or (edge_u >= edge_d):
        print('Error: Incorrect edge values. The left (up) edge must have smaller value than the right
        return y
    elif ( (0 <= edge_l <= idx[1]) or (0 <= edge_r <= idx[1])           or (0 <= edge_u <= idx[0]) or (
        print('Error: Incorrect edge values. Number should be in the region of image. ')
        return y

    # set region outside edge to zero intensity
    if len(idx) == 2: # image
        yn = y.copy()
        yn[:,:edge_l] = 0
        yn[:,edge_r:] = 0
        yn[:edge_u,:] = 0
        yn[edge_d:,:] = 0

    elif len(idx) == 3: # video

        yn = np.zeros(idx)
        # set elements to zero if below threshold
        for i in range(idx[-1]):
            yn[:,:,i] = y[:,:,i].copy()
            yn[:,:edge_l,i] = 0
            yn[:,edge_r:,i] = 0
            yn[:edge_u,:,i] = 0
            yn[edge_d:,:,i] = 0

    else:
        yn = y
        print('Error: Wrong dimensions. Input data must be a video or image in grayscale. ')

    return yn


# In[1316]:


def traj_track(y, win_x, win_y, win2_start = 0, win2 = 0):
    '''
    Extract the trajectory of an object, i.e. find maximum intensity in the window
    y: (NUMPY ARRAY) data of a set of images w/ background treatment
    Return value
    trajx: (NUMPY ARRAY) (Dim: frames*1) the x-trajectory of the object
    trajy: (NUMPY ARRAY) (Dim: frames*1) the y-trajectory of the object
    win_x: (INT) window size of point tracking in x direction
    win_y: (INT) window size of point tracking in y direction
    win2: (INT) window size change of point tracking for 2nd window
```

```
win2_start: (INT) frame # that 2nd window takes place
'''
# grab dimensions of data
idx = y.shape # idx[0]: height, idx[1]: width

# justify the effectiveness of windowsize
if (win_x <= 0) or (win_x >= min(idx[0], idx[1])/2) or (win_y <= 0) or (win_y >= min(idx[0], idx[1])
    print('Error: Wrong window size.')
    return 0, 0

if ( (win_x - abs(win2)) <= 0) or ( (win_x + abs(win2)) >= min(idx[0], idx[1])/2) or    ( (win_y - a
    print('Error: Wrong window size change.')
    return 0, 0

if len(idx) == 2: # image

    max_idx = np.unravel_index(y.argmax(), idx)
    trajx = max_idx[1]
    trajy = max_idx[0]

elif len(idx) == 3: # video

    trajx = np.zeros(idx[-1])
    trajy = np.zeros(idx[-1])

    # first image: traj = max intensity
    max_idx = np.unravel_index(y[:,:,0].argmax(), idx[:-1])
    trajx[0] = max_idx[1] # Note: the order is reversed for image
    trajy[0] = max_idx[0]

    # search only in the neighbor to avoid other illumination
    for i in range(1, idx[-1]): # i: frame number

        # window size
        window_x = win_x + (i >= win2_start) * win2
        window_y = win_y + (i >= win2_start) * win2

        # neighbor
        xmin = trajx[i-1] - window_x
        xmax = trajx[i-1] + window_x
        ymin = trajy[i-1] - window_y
        ymax = trajy[i-1] + window_y

        # justify the effectiveness of data range
        if xmin < 0:
            xmin = 0
            xmax = xmin + 2*window_x
        if xmax >= idx[1]:
            xmax = idx[1]
            xmin = xmax - 2*window_x
        if ymin < 0:
            ymin = 0
            ymax = ymin + 2*window_y
        if ymax >= idx[0]:
```

```
                ymax = idx[0]
                ymin = ymax - 2*window_y

            # find max intensity
            data_w = y[(int)(ymin):(int)(ymax), (int)(xmin):(int)(xmax), i].copy()
            max_idx = np.unravel_index(data_w.argmax(), data_w.shape)
            trajx[i] = trajx[i-1] - window_x + max_idx[1] # Note: the order is reversed for image
            trajy[i] = trajy[i-1] - window_y + max_idx[0]

    else:
        trajx = 0
        trajy = 0
        print('Error: Wrong dimensions. Input data must be a video or image in grayscale. ')

    return trajx, trajy


# In[818]:


def traj_track_s(y):
    '''
    Extract the trajectory of an object, i.e. find maximum intensity in each image
    Simple version: w/o usage of window
    y: (NUMPY ARRAY) data of a set of images w/ background treatment
    Return value
    trajx: (NUMPY ARRAY) (Dim: frames*1) the x-trajectory of the object
    trajy: (NUMPY ARRAY) (Dim: frames*1) the y-trajectory of the object
    '''
    # grab dimensions of data
    idx = y.shape # idx[0]: height, idx[1]: width

    if len(idx) == 2: # image

        max_idx = np.unravel_index(y.argmax(), idx)
        trajx = max_idx[1]
        trajy = max_idx[0]

    elif len(idx) == 3: # video

        trajx = np.zeros(idx[-1])
        trajy = np.zeros(idx[-1])

        for i in range(idx[-1]): # i: frame number

            # find max intensity
            max_idx = np.unravel_index(y[:,:,i].argmax(), idx[:-1])
            trajx[i] = max_idx[1] # Note: the order is reversed for image
            trajy[i] = max_idx[0]

    else:
        trajx = 0
        trajy = 0
        print('Error: Wrong dimensions. Input data must be a video or image in grayscale. ')
```

```
    return trajx, trajy


# In[1721]:


def traj_repair(y, window, interval, trajx0, trajy0, color):
    '''
    Repair the trajectory of an object with color identifier
    Work with traj_track_s(y)
    Input:
        y: (NUMPY ARRAY) data in RGB of a set of images w/ background treatment
        window: (List) [xmin, xmax, ymin, ymax]
        interval = [start, end]: (List) interval of those frames with tracking points hidden
        trajx0: (NUMPY ARRAY) the original x-trajectory of the object
        trajy0: (NUMPY ARRAY) the original y-trajectory of the object
        color: (List) [red, green, blue]
    Output:
        trajx: (NUMPY ARRAY) (Dim: frames*1) the x-trajectory of the object
        trajy: (NUMPY ARRAY) (Dim: frames*1) the y-trajectory of the object
    '''
    # grab dimensions of data
    idx = y.shape # idx[0]: height, idx[1]: width
    # convert to numpy array
    color = np.array(color)
    # window
    [xmin, xmax, ymin, ymax] = window

    # determine the effectiveness of bdrys
    if (xmin >= xmax) or (ymin >= ymax):
        print('Error: Invalid boundary values. ')
        return 0, 0
    elif ( (0 <= xmin <= idx[1]) or (0 <= xmax <= idx[1])           or (0 <= ymin <= idx[0]) or (0 <= y
        print('Error: Boundary values exceeding the range. ')
        return 0, 0

    if len(idx) == 4: # video

        trajx = trajx0.copy()
        trajy = trajy0.copy()

        for inc in interval:

            tstart = inc[0]
            tend = inc[1] + 1

            for i in range(tstart, tend):

                # find closest color in plot
                data = y[ymin:ymax, xmin:xmax, :, i].copy()
                dist = np.linalg.norm((data - color), axis = 2)
                min_idx = np.unravel_index(dist.argmin(), dist.shape)
                trajx[i] = xmin + min_idx[1] # Note: the order is reversed for image
```

```
                trajy[i] = ymin + min_idx[0]

    else:
        trajx = 0
        trajy = 0
        print('Error: Wrong dimensions. Input data must be a video in RGB. ')

    return trajx, trajy


# In[1954]:


def spike_remover(mode0, threshold):
    '''
    Remove the spikes for better plot, and smoothen the curve
    Input:
        mode0: (NUMPY ARRAY) a vector of mode
        threshold: minimum value for relative spike intensity
    Outputput:
        mode: (NUMPY ARRAY) a vector of mode with spikes removed
    '''

    mode = mode0.copy()

    for i in range(1, len(mode)-2):

        if abs(mode[i] - mode[i-1]) >= threshold:
            # whether next one is also in the peak
            if abs(mode[i+1] - mode[i]) <= threshold:
                mode[i] = (2*mode[i-1] + mode[i+2])/3
                mode[i+1] = (mode[i-1] + 2*mode[i+2])/3

            else:
                mode[i] = (mode[i-1] + mode[i+1])/2

    return mode


# # Test 1

# In[125]:


# load data from files
cam11 = loadgsdata('datasets/cam1_1.mat', variable = 'vidFrames1_1')
cam21 = loadgsdata('datasets/cam2_1.mat', variable = 'vidFrames2_1')
cam31 = loadgsdata('datasets/cam3_1.mat', variable = 'vidFrames3_1')


# In[150]:


# starting & ending points for test
```

```
len11 = (cam11.shape)[-1]
len21 = (cam21.shape)[-1]
len31 = (cam31.shape)[-1]
shift11 = 1
shift21 = 10
shift31 = 0
len_t1 = min([len11-shift11, len21-shift21, len31-shift31])

# crop & synchronize data
cam11n = cam11[:,:,shift11:(len_t1 + shift11)].copy()
cam21n = cam21[:,:,shift21:(len_t1 + shift21)].copy()
cam31n = cam31[:,:,shift31:(len_t1 + shift31)].copy()


# In[192]:


# plot grayscaled video segment for reference
plt.figure(2)
fnum = 41 # frame number
data_plot = cam11n[:,:,fnum]
plt.imshow(data_plot, cmap = plt.get_cmap("gray"))
# maximum intensity
maxidx = np.unravel_index(data_plot.argmax(), data_plot.shape)
plt.plot(maxidx[1], maxidx[0], 'go')
plt.show()


# In[185]:


# background darkening
cam11nd = bg_darken(cam11n, 300, 400, 50, 390)
cam21nd = bg_darken(cam21n, 220, 380, 50, 320)
cam31nd = bg_darken(cam31n, 200, 425, 200, 400)


# In[1317]:


# extract trajectory of paint can
trajx11, trajy11 = traj_track(cam11nd, 25, 25)
trajx21, trajy21 = traj_track(cam21nd, 25, 25)
trajx31, trajy31 = traj_track(cam31nd, 15, 15)


# In[1318]:


# plot extracted trajectory (sanity check)
plt.figure(5, figsize = (10,4))
plt.subplot(1,3,1)
plt.plot(trajy11)
plt.subplot(1,3,2)
```

```
plt.plot(trajy21)
plt.subplot(1,3,3)
plt.plot(trajx31)
plt.show()


# In[1151]:


# svd
a1 = np.stack((trajx11, trajy11, trajx21, trajy21, trajx31, trajy31))
for i in range(6): a1[i] = a1[i] - a1[i].mean()
u1, s1, v1 = np.linalg.svd(a1)


# In[1978]:


# plot singular values & vectors
plt.figure(6)
plt.scatter(np.array([1,2,3,4,5,6]),s1) # sqrt of variance
plt.xlabel('Singular value number')
plt.ylabel('Singular value')
plt.show()
plt.figure(7)
plt.plot(v1[0]*s1[0], label = '1st mode')
plt.plot(v1[1]*s1[1], label = '2nd mode')
plt.plot(v1[2]*s1[2], label = '3rd mode')
plt.plot(v1[3]*s1[3], label = '4th mode')
plt.plot(v1[4]*s1[4], label = '5th mode')
plt.plot(v1[5]*s1[5], label = '6th mode')
plt.legend(bbox_to_anchor=(0.9,0.7))
plt.xlabel('Frames')
plt.ylabel('Normalized Position')
plt.show()


# # Test 2

# In[302]:


# load data from files
cam12 = loadgsdata('datasets/cam1_2.mat', variable = 'vidFrames1_2')
cam22 = loadgsdata('datasets/cam2_2.mat', variable = 'vidFrames2_2')
cam32 = loadgsdata('datasets/cam3_2.mat', variable = 'vidFrames3_2')


# In[303]:


# starting & ending points for test
len12 = (cam12.shape)[-1]
len22 = (cam22.shape)[-1]
```

```
len32 = (cam32.shape)[-1]
shift12 = 0
shift22 = 23
shift32 = 2
len_t2 = min([len12-shift12, len22-shift22, len32-shift32])

# crop & synchronize data
cam12n = cam12[:,:,shift12:(len_t2 + shift12)].copy()
cam22n = cam22[:,:,shift22:(len_t2 + shift22)].copy()
cam32n = cam32[:,:,shift32:(len_t2 + shift32)].copy()


# In[487]:


# plot grayscaled video segment for reference
plt.figure(11)
fnum = 266 # frame number
data_plot = cam32nd[:,:,fnum]
plt.imshow(data_plot, cmap = plt.get_cmap("gray"))
# maximum intensity
maxidx = np.unravel_index(data_plot.argmax(), data_plot.shape)
plt.plot(maxidx[1], maxidx[0], 'go')
plt.plot(trajx32[fnum], trajy32[fnum], 'co')
plt.show()


# In[393]:


# background darkening
cam12nd = bg_darken(cam12n, 280, 450, 200, 430)
cam22nd = bg_darken(cam22n, 200, 420, 50, 430)
cam32nd = bg_darken(cam32n, 250, 500, 180, 320)


# In[1319]:


# extract trajectory of paint can
trajx12, trajy12 = traj_track(cam12nd, 25, 25)
trajx22, trajy22 = traj_track(cam22nd, 35, 35)
trajx32, trajy32 = traj_track(cam32nd, 25, 25, win2_start = 260, win2 = 5)


# In[1320]:


# plot extracted trajectory (sanity check)
plt.figure(13, figsize = (10,4))
plt.subplot(1,3,1)
plt.plot(trajy12)
plt.subplot(1,3,2)
plt.plot(trajy22)
```

```
plt.subplot(1,3,3)
plt.plot(trajx32)
plt.show()


# In[1143]:


# svd
a2 = np.stack((trajx12, trajy12, trajx22, trajy22, trajx32, trajy32))
for i in range(6): a2[i] = a2[i] - a2[i].mean()
u2, s2, v2 = np.linalg.svd(a2)


# In[1980]:


# plot singular values & vectors
plt.figure(16)
plt.scatter(np.array([1,2,3,4,5,6]),s2) # sqrt of variance
plt.xlabel('Singular value number')
plt.ylabel('Singular value')
plt.show()
plt.figure(17)
plt.plot(v2[0]*s2[0], label = '1st mode')
plt.plot(v2[1]*s2[1], label = '2nd mode')
#plt.plot(v2[2]*s2[2], label = '3rd mode')
#plt.plot(v2[3]*s2[3], label = '4th mode')
#plt.plot(v2[4]*s2[4], label = '5th mode')
#plt.plot(v2[5]*s2[5], label = '6th mode')
plt.legend(bbox_to_anchor=(0.9,0.7))
plt.xlabel('Frames')
plt.ylabel('Normalized Position')
plt.show()


# # Test 3

# In[516]:


# load data from files
cam13 = loadgsdata('datasets/cam1_3.mat', variable = 'vidFrames1_3')
cam23 = loadgsdata('datasets/cam2_3.mat', variable = 'vidFrames2_3')
cam33 = loadgsdata('datasets/cam3_3.mat', variable = 'vidFrames3_3')


# In[521]:


# starting & ending points for test
len13 = (cam13.shape)[-1]
len23 = (cam23.shape)[-1]
len33 = (cam33.shape)[-1]
```

```
shift13 = 5
shift23 = 28
shift33 = 0
len_t3 = min([len13-shift13, len23-shift23, len33-shift33])

# crop & synchronize data
cam13n = cam13[:,:,shift13:(len_t3 + shift13)].copy()
cam23n = cam23[:,:,shift23:(len_t3 + shift23)].copy()
cam33n = cam33[:,:,shift33:(len_t3 + shift33)].copy()


# In[781]:


# plot grayscaled video segment for reference
plt.figure(21)
fnum = 200 # frame number
data_plot = cam33nd[:,:,fnum]
plt.imshow(data_plot, cmap = plt.get_cmap("gray"))

# maximum intensity
maxidx = np.unravel_index(data_plot.argmax(), data_plot.shape)
plt.plot(maxidx[1], maxidx[0], 'go')
# plot calculated trajectory
plt.plot(trajx33[fnum], trajy33[fnum], 'co')
plt.show()


# In[762]:


# background darkening
cam13nd = bg_darken(cam13n, 200, 400, 150, 400)
cam23nd = bg_darken(cam23n, 200, 450, 200, 400)
cam33nd = bg_darken(cam33n, 250, 450, 150, 400)


# In[1321]:


# extract trajectory of paint can (have to use a lower point for cam 1 & 2)
trajx13, trajy13 = traj_track(np.flip(cam13nd, 2), 25, 25, win2_start = 169, win2 = -5) # flip for bett
trajx13 = np.flip(trajx13)
trajy13 = np.flip(trajy13)
trajx23, trajy23 = traj_track(cam23nd, 15, 15)
trajx33, trajy33 = traj_track(cam33nd, 25, 25)


# In[1322]:


# plot extracted trajectory (sanity check)
plt.figure(23, figsize = (10,4))
plt.subplot(1,3,1)
```

```
plt.plot(trajy13)
plt.subplot(1,3,2)
plt.plot(trajy23)
plt.subplot(1,3,3)
plt.plot(trajx33)
plt.show()


# In[1161]:


# svd
a3 = np.stack((trajx13, trajy13, trajx23, trajy23, trajx33, trajy33))
for i in range(6): a3[i] = a3[i] - a3[i].mean()
u3, s3, v3 = np.linalg.svd(a3)


# In[1983]:


# plot singular values & vectors
plt.figure(26)
plt.scatter(np.array([1,2,3,4,5,6]),s3)
plt.xlabel('Singular value number')
plt.ylabel('Singular value')
plt.show()
plt.figure(27)
plt.plot(v3[0]*s3[0], label = '1st mode')
plt.plot(v3[1]*s3[1], label = '2nd mode')
plt.plot(v3[2]*s3[2], label = '3rd mode')
#plt.plot(v3[3]*s3[3], label = '4th mode')
#plt.plot(v3[4]*s3[4], label = '5th mode')
#plt.plot(v3[5]*s3[5], label = '6th mode')
plt.legend(bbox_to_anchor=(0.9,0.7))
plt.xlabel('Frames')
plt.ylabel('Normalized Position')
plt.show()


# # Test 4

# ## Import & synchronize data

# In[1208]:


# load color data from files
cam14c = loadmat('datasets/cam1_4.mat')['vidFrames1_4']
cam24c = loadmat('datasets/cam2_4.mat')['vidFrames2_4']
cam34c = loadmat('datasets/cam3_4.mat')['vidFrames3_4']


# In[790]:
```

```
# load grayscale data from files
cam14 = loadgsdata('datasets/cam1_4.mat', variable = 'vidFrames1_4')
cam24 = loadgsdata('datasets/cam2_4.mat', variable = 'vidFrames2_4')
cam34 = loadgsdata('datasets/cam3_4.mat', variable = 'vidFrames3_4')


# In[791]:


# starting & ending points for test
len14 = (cam14.shape)[-1]
len24 = (cam24.shape)[-1]
len34 = (cam34.shape)[-1]
shift14 = 2
shift24 = 7
shift34 = 0
len_t4 = min([len14-shift14, len24-shift24, len34-shift34])

# crop & synchronize data
cam14n = cam14[:,:,shift14:(len_t4 + shift14)].copy()
cam24n = cam24[:,:,shift24:(len_t4 + shift24)].copy()
cam34n = cam34[:,:,shift34:(len_t4 + shift34)].copy()


# In[1209]:


# crop & synchronize color data
cam14cn = cam14c[:,:,:,shift14:(len_t4 + shift14)].copy()
cam24cn = cam24c[:,:,:,shift24:(len_t4 + shift24)].copy()
cam34cn = cam34c[:,:,:,shift34:(len_t4 + shift34)].copy()


# In[1890]:


# plot with R, G, B channel
plt.figure(51)
fnum = 10 # frame number

data_plot = cam24cn[:,:,:,fnum]
plt.imshow(data_plot)

# plot calculated trajectory
plt.plot(trajx24[fnum], trajy24[fnum], 'co')
plt.plot(trajx24rrr[fnum], trajy24rrr[fnum], 'mo')
plt.show()


# ## Extract trajectory with traj_track_s(y)

# In[796]:
```

```
# background darkening
cam14nd = bg_darken(cam14n, 300, 460, 200, 430)
cam24nd = bg_darken(cam24n, 200, 430, 50, 400)
cam34nd = bg_darken(cam34n, 250, 500, 120, 300)


# In[1255]:


# extract trajectory of paint can (have to use a lower point for cam 1 & 2)
trajx14, trajy14 = traj_track_s(cam14nd)
trajx24, trajy24 = traj_track_s(cam24nd)
trajx34, trajy34 = traj_track_s(cam34nd)


# In[1081]:


# plot extracted trajectory (problematic region are shaded)
plt.figure(33, figsize = (10,4))

plt.subplot(1,3,1)
plt.plot(trajy14)
vspan1 = np.array([[0, 13], [66, 153], [183, 235], [251, 251], [263, 332], [368, 389]])
vlines1 = np.ndarray.flatten(vspan1)
for x in vlines1:
    plt.axvline(x, color='r')
for x in vspan1:
    plt.axvspan(x[0], x[1], alpha=0.15, color='red')

plt.subplot(1,3,2)
plt.plot(trajy24)
vspan2 = np.array([[0, 22], [33, 35], [40, 41], [58, 58], [66, 153], [165, 165], [173, 173],
vlines2 = np.ndarray.flatten(vspan2)
for x in vlines2:
    plt.axvline(x, color='r')
for x in vspan2:
    plt.axvspan(x[0], x[1], alpha=0.15, color='red')

plt.subplot(1,3,3)
plt.plot(trajx34)
vspan3 = np.array([[0, 34], [125, 125], [131, 132], [169, 211], [247, 288], [300, 302]])
vlines3 = np.ndarray.flatten(vspan3)
for x in vlines3:
    plt.axvline(x, color='r')
for x in vspan3:
    plt.axvspan(x[0], x[1], alpha=0.15, color='red')

plt.show()


# ## Repair the trajectory in the regions with flashlight hidden
```

```
# In[1827]:


#flashlight1 = [217, 153, 160] # lighter
flashlight1 = [209, 146, 152] # darker
trajx14rrr, trajy14rrr = traj_repair(cam14cn, [300, 460, 200, 430], vspan1, trajx14, trajy14, flashligh


# In[1888]:


flashlight2 = [220, 176, 188] # darker
trajx24rrr, trajy24rrr = traj_repair(cam24cn, [200, 430, 90, 400], vspan2, trajx24, trajy24, flashlight


# In[1723]:


flashlight3 = [193, 155, 147]
trajx34rrr, trajy34rrr = traj_repair(cam34cn, [250, 500, 120, 300], vspan3, trajx34, trajy34, flashligh


# ## Compare the repaired trajectory to the original

# In[1830]:


# original vs repair (cam1)
plt.figure(35, figsize = (9,4))
plt.subplot(1,2,1)
plt.plot(trajy14, label = 'original')
plt.plot(trajy14rrr, label = 'repair')
vspan1 = np.array([[0, 13], [66, 153], [183, 235], [251, 251], [263, 332], [368, 389]])
vlines1 = np.ndarray.flatten(vspan1)
for x in vlines1:
    plt.axvline(x, color='r')
for x in vspan1:
    plt.axvspan(x[0], x[1], alpha=0.15, color='red')
plt.legend()

plt.subplot(1,2,2)
plt.plot(trajx14, label = 'original')
plt.plot(trajx14rrr, label = 'repair')
for x in vlines1:
    plt.axvline(x, color='r')
for x in vspan1:
    plt.axvspan(x[0], x[1], alpha=0.15, color='red')
plt.legend()

plt.show()


# In[1829]:
```

```
# original vs repair (cam3)
plt.figure(36, figsize = (9,4))
plt.subplot(1,2,1)
plt.plot(trajx34, label = 'original')
plt.plot(trajx34rrr, label = 'repair')
vspan3 = np.array([[0, 34], [125, 125], [131, 132], [169, 211], [247, 288], [300, 302]])
vlines3 = np.ndarray.flatten(vspan3)
for x in vlines3:
    plt.axvline(x, color='r')
for x in vspan3:
    plt.axvspan(x[0], x[1], alpha=0.15, color='red')
plt.legend()

plt.subplot(1,2,2)
plt.plot(trajy34, label = 'original')
plt.plot(trajy34rrr, label = 'repair')
for x in vlines3:
    plt.axvline(x, color='r')
for x in vspan3:
    plt.axvspan(x[0], x[1], alpha=0.15, color='red')
plt.legend()

plt.show()


# In[1889]:


# original vs repair (cam2)
plt.figure(37, figsize = (9,4))
plt.subplot(1,2,1)
plt.plot(trajx24, label = 'original')
plt.plot(trajx24rrr, label = 'repair')
vspan2 = np.array([[0, 22], [33, 35], [40, 41], [58, 58], [66, 153], [165, 165], [173, 173],
vlines2 = np.ndarray.flatten(vspan2)
for x in vlines2:
    plt.axvline(x, color='r')
for x in vspan2:
    plt.axvspan(x[0], x[1], alpha=0.15, color='red')
plt.legend()

plt.subplot(1,2,2)
plt.plot(trajy24, label = 'original')
plt.plot(trajy24rrr, label = 'repair')
for x in vlines2:
    plt.axvline(x, color='r')
for x in vspan2:
    plt.axvspan(x[0], x[1], alpha=0.15, color='red')
plt.legend()

plt.show()
```

```
# # SVD

# In[1891]:


# svd
a4 = np.stack((trajx14rrr, trajy14rrr, trajx24rrr, trajy24rrr, trajx34rrr, trajy34rrr))
for i in range(6): a4[i] = a4[i] - a4[i].mean()
u4, s4, v4 = np.linalg.svd(a4)


# In[1989]:


# plot singular values & vectors
plt.figure(66)
plt.scatter(np.array([1,2,3,4,5,6]), s4)
plt.xlabel('Singular value number')
plt.ylabel('Singular value')
plt.show()
plt.figure(67)
plt.plot(v4[2])
plt.plot(v4n[2])
plt.xlabel('Frames')
plt.ylabel('Normalized Position')
plt.show()


# # Remove spikes due to incontinuity in repairing the trajectory

# In[1988]:


# remove spikes
v4n = v4.copy()
# 1st mode
v4n[0] = spike_remover(v4[0], 0.02)
v4n[0][289:295] = spike_remover(v4n[0][289:295], 0.01)
# 2nd mode
v4n[1] = spike_remover(v4[1], 0.035)
v4n[1][169:175] = spike_remover(v4n[1][169:175], 0.01)
# 3rd mode
v4n[2] = spike_remover(v4[2], 0.035)


# In[1991]:


# plot right singular vector (smoothened) (2 modes)
plt.figure(68)
plt.scatter(np.array([1,2,3,4,5,6]), s4)
plt.xlabel('Singular value number')
plt.ylabel('Singular value')
plt.show()
```

```
plt.figure(69)
plt.plot(v4n[0]*s4[0], label = '1st mode')
plt.plot(v4n[1]*s4[1], label = '2nd mode')
plt.plot(v4n[2]*s4[2], label = '3rd mode')
plt.legend(bbox_to_anchor=(0.9,0.8))
plt.xlabel('Frames')
plt.ylabel('Normalized Position')

plt.show()


# ## Supplement: determine the color of the flashlight handle (used and saved in flashlight1,2,3 during

# ### Color of flashlight, Cam 3, Test 4

# In[1625]:


plt.figure(104)
fnum = 15
plt.imshow(cam34cn[:,:,:,fnum])
plt.axvline(310)
plt.axvline(340)
plt.axhline(220)
plt.axhline(190)
plt.show()


# In[1630]:


flashcolor0 = cam34cn[231:240,360:367,:,0].reshape((63,3))
flashcolor5 = cam34cn[229:236,338:344,:,5].reshape((42,3))
flashcolor10 = cam34cn[217:221,319:324,:,10].reshape((20,3))
flashcolor15 = cam34cn[203:212,320:326,:,15].reshape((54,3))


# In[1629]:


plt.figure(105)
fnum = 15
plt.imshow(cam34cn[190:220,310:340,:,fnum])
plt.axvline(10)
plt.axvline(15)
plt.axhline(13)
plt.axhline(21)
plt.show()


# In[1659]:
```

```
flashcolor = np.concatenate((flashcolor0, flashcolor5, flashcolor10, flashcolor15)).copy()
#flashcolor = np.unique(flashcolor, axis = 0)


# In[1660]:


flashcolor.mean(axis = 0)


# In[1675]:


plt.figure(108)
plt.subplot(1,2,1)
test = np.concatenate((flashcolor, np.array([[0,0,0]]))).reshape((15,12,3))
plt.imshow(test)
plt.subplot(1,2,2)
plt.imshow(np.unique(flashcolor, axis = 0).reshape((15,11,3)))
plt.show()


# In[1969]:


flashcoloru = np.unique(flashcolor, axis = 0).reshape((15,11,3))
flashcoloru[12:15].reshape(33, 3).mean(axis = 0)


# In[1970]:


# flashlight3
plt.figure(109)
plt.imshow(np.array([[[193, 155, 147]]]))
plt.show()


# ### Color of flashlight, Cam 1, Test 4

# In[1832]:


plt.figure(114)
fnum = 70
plt.imshow(cam14cn[:,:,:,fnum])
plt.axvline(380)
plt.axvline(350)
plt.axhline(350)
plt.axhline(320)
plt.show()


# In[1797]:
```

```
flashcolor0 = cam14cn[263:271,394:401,:,0].reshape((56,3))
flashcolor00 = cam14cn[260:269,401:406,:,0].reshape((45,3))
flashcolor10 = cam14cn[251:258,413:423,:,10].reshape((70,3))
flashcolor1010 = cam14cn[250:254,423:430,:,10].reshape((28,3))
flashcolor70 = cam14cn[330:338,355:362,:,70].reshape((56,3))
flashcolor7070 = cam14cn[334:340,362:366,:,70].reshape((24,3))


# In[1796]:


plt.figure(115)
fnum = 70
plt.imshow(cam14cn[320:350,350:380,:,fnum])
plt.axvline(15)
plt.axvline(12)
plt.axhline(14) # add to first index
plt.axhline(19)
plt.show()


# In[1799]:


flashcolor1 = np.concatenate((flashcolor0, flashcolor00, flashcolor10, flashcolor1010, flashcolor70, fla


# In[1825]:


flashcolor11 = np.concatenate((flashcolor00, flashcolor1010, flashcolor70)).copy()


# In[1800]:


flashcolor1.mean(axis = 0)


# In[1826]:


# flashlight1 (finally used)
flashcolor11.mean(axis = 0)


# In[1808]:


test = np.concatenate((np.unique(flashcolor1, axis = 0), np.array([[0,0,0]])))
plt.figure(118)
plt.imshow(test.reshape((15,18,3)))
```

```
plt.show()


# In[1814]:


# flashlight1
plt.figure(119)
plt.imshow(np.array([[[217, 153, 160]]]))
plt.show()


# ### Color of flashlight, Cam 2, Test 4

# In[1864]:


plt.figure(154)
fnum = 70
plt.imshow(cam24cn[:,:,:,fnum])
plt.axvline(305)
plt.axvline(270)
plt.axhline(255)
plt.axhline(235)
plt.show()


# In[1874]:


flashcolor0 = cam24cn[200:205,289:294,:,0].reshape((25,3))
flashcolor5 = cam24cn[160:165,324:330,:,5].reshape((30,3))
flashcolor70 = cam24cn[245:250,286:298,:,70].reshape((60,3))


# In[1873]:


plt.figure(155)
fnum = 70
plt.imshow(cam24cn[235:255,270:305,:,fnum])
plt.axvline(27)
plt.axvline(16)
plt.axhline(14) # add to first index
plt.axhline(10)
plt.show()


# In[1876]:


flashcolor2 = np.concatenate((flashcolor0, flashcolor5, flashcolor70)).copy()
flashcolor2.mean(axis = 0)
```

```
# In[1882]:


test = np.unique(np.concatenate((flashcolor2, np.array([[0,0,0],[1,1,1]]) )), axis = 0)
plt.figure(158)
plt.imshow(test.reshape((23,5,3)))
plt.show()


# In[1883]:


# flashlight2
plt.figure(159)
plt.imshow(np.array([[[220, 176, 188]]]))
plt.show()
```