**MTRE 2610 Engineering Algorithms and Visualization – Dr. Kevin McFall**

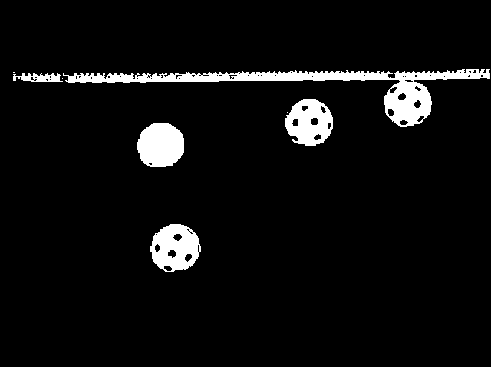
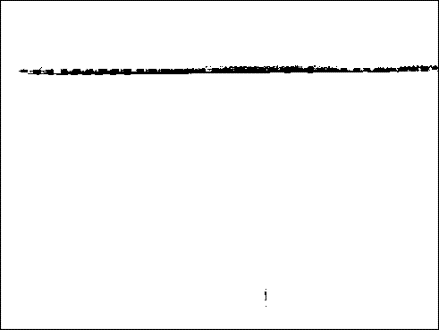
**Laboratory – Tracking in 2D**

**Introduction**

The goal of this laboratory exercise is to use track a ball’s movement in a video stream and estimate its actual speed.

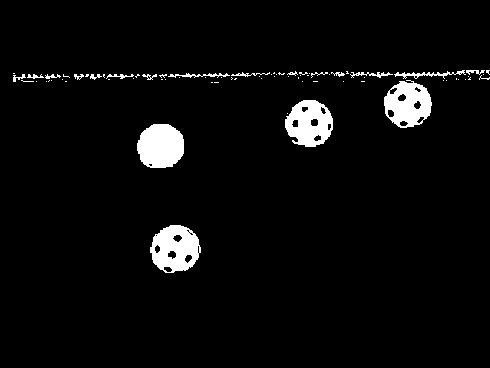
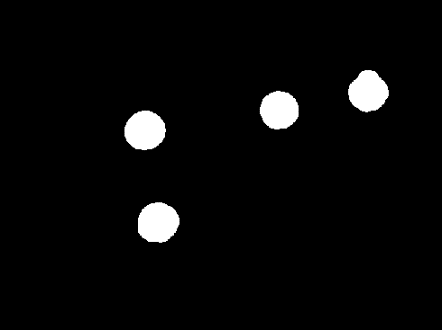
**Thresholding to Detect Balls**

Detecting the pink balls in the supplied image im30.png in Figure 1 (left) can be accomplished using color thresholding. Applying a threshold for high red value, Figure 1 (center), finds not only the balls but the yellow measuring tape as well (yellow is the combination of red and green). A low green threshold, Figure 1 (right) detects nearly everything except the measuring tape since it is the only thing with significant green content.

**  **

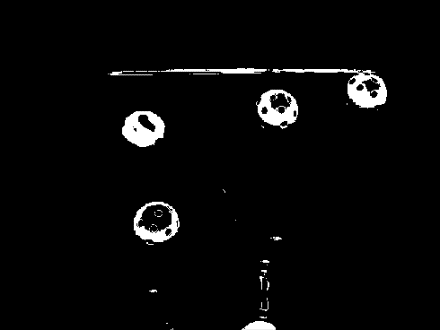
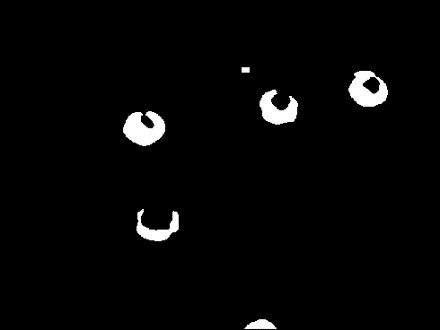
**Figure 1: Original image (left), high red values (center), and low green values (right).**

A logical AND of high red and low green, Figure 2 (left), eliminates some but not all of the measuring tape. Removing the remainder of the tape, Figure 2 (right), is best accomplished with a blur function that removes small groups of pixels and fills in holes.

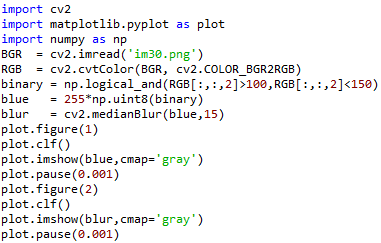
 

**Figure 2: Logical AND of high red and low green (left) and then applying a blur (right) to identify only the balls.**

The following sample code produces a similar although inferior result, see Figure 3, by allowing only blue values between 100 and 150.

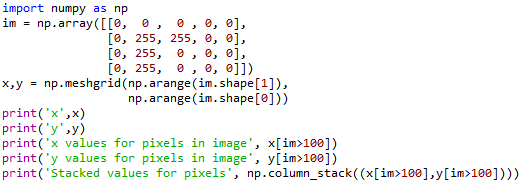
**Figure 3: Color threshold for blue between 100 and 150 (left) with a blur applied (right).**



Note that applying thresholds results in the Numpy array of Booleans binary that is then converted back to an array of 8-bit unsigned integers with max value of 255. Increasing the 15 parameter in medianBlur increases the amount of blur, and vice versa.

**Detecting Ball Centers**

A properly thresholded and blurred image in Figure 2 (right) identifies pixels in the image corresponding to the four balls. A convenient method for determining the four ball centers is with the *k*-means clustering algorithm, which requires the *x*,*y* positions of all ball pixels. Consider the code below and its output for a simplified image im where 255 values represent ball pixels.



x [[0 1 2 3 4]

[0 1 2 3 4]

[0 1 2 3 4]

[0 1 2 3 4]]

y [[0 0 0 0 0]

[1 1 1 1 1]

[2 2 2 2 2]

[3 3 3 3 3]]

x values for pixels in image [1 2 1 1]

y values for pixels in image [1 1 2 3]

Stacked values for pixels [[1 1]

[2 1]

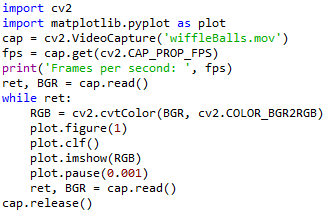
[1 2]

[1 3]]

A meshgrid is first generated representing the *x*,*y* positions of every pixel in the image. The *x* and *y* values corresponding to the 255 values are found using logical indexing on the im matrix, and column\_stack places them in the required format for *k*-means clustering where each row represents the *x*,*y* position of one ball pixel.

**Reading Frames from Video**

Video is essentially a series of images played with a particular frequency. OpenCV includes VideoCapture objects from which images can be extracted from given video files, such as the supplied wiffleBalls.mov. The code below displays the playback frequency in frames per second and reads through the video displaying each frame on the screen. VideoCapture.read() returns two variables, the first a Boolean indicating if the read was successful, and the second the image read.



**Tracking the Moving Ball**

One of the balls moves in the video and the others are stationary. Estimating the moving ball’s speed is possible by measuring the distance its detected circle moves from frame to frame. To accomplish this, store the circle centers in a variable that can be compared against centers detected in the next frame. Unfortunately, the clustering algorithm does not keep track of which ball is which, and so balls may be listed in different orders from frame to frame. For each frame, loop through each of the four circle centers from the previous frame and find the current circle closest to it. Assume these circles represent the same ball in both frames. Whichever circle center has moved the most (stationary circles may move slightly due to noise or a shaking camera) must be the moving ball. Store the distance the non-stationary ball has moved for each frame. Use the pixel-to-foot calibration and the frame rate to convert the pixel ball movements into speeds in ft/s. This information can be used to plot the speed of the moving ball over time.

**Unit conversions**

It is desired to plot the speed of the ball in ft/s on the vertical axis and time on the horizontal axis. Movement measurements described above produce ball speed in units of pixels/frame. The recording frames per second returned by the VideoCapture object allows conversion from frame to seconds. To convert from pixels to feet, mark two locations in the image a known distance apart on the measuring tape, and calculate the number of pixels between these points (perhaps using matplotlib.pyplot.ginput).

**Laboratory exercise procedure**

Load the im30.mat image file and adjust threshold values and blur amounts to obtain a result similar to that in Figure 2 (right). Use *k*-means clustering with the detected pixels to find the ball centers and display them on the image. Still working with im30.mat, write a nested for-loop (or a single for-loop using element-wise operators) to generate the distance matrix between each of the four balls in the image. Such matrices, as the one appearing in Figure 2, were popular in road atlases to easily visualize distances between major cities before GPSs became popular. The value in the cell defined by row *i* and column *j* is the distance between city *i* and city *j*. Notice the matrix is symmetric about the zero values along the diagonal. Symmetry occurs because the distance between city *i* and city *j* is the same as the distance between cities *j* and *i*.

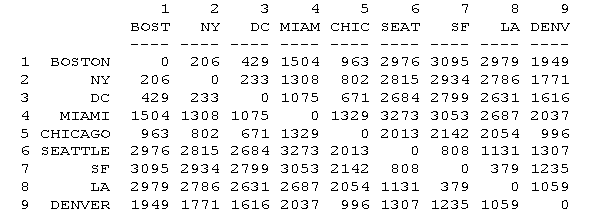
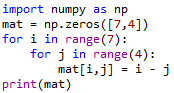


Figure 2: Distance matrix for US cities.

Generating the distance matrix requires traversing each position in the matrix, which is simplest using nested for loops. As a review, consider the code



looping through seven rows and four columns, setting a value in the matrix mat to the difference between the row and column number. The results is

increasing *j* →

0 -1 -2 -3

1 0 -1 -2

2 1 0 -1

increasing *i* →

3 2 1 0

4 3 2 1

5 4 3 2

6 5 4 3

Generating the distance matrix requires a similar nested loop structure where the value in mat will be the distance from center *i* to center *j* rather than the difference between row and column number.

Now load the wiffleBalls.mov video file instead and loop through every frame. Check that *k*-means clustering with the previously identified threshold and blur parameters properly detect all four balls in each frame, making adjustments as necessary. The trick now is to identify which of the four detected centers is the moving ball in each frame. To accomplish this, store the old ball center values from the previous frame and compare them to the detected centers in the current frame. Complete the distance matrix in Figure 3 where each row represents a ball location in the previous frame and each column a center in the current frame. This implies that the value at row *i* and column *j* will be the distance between the *i*th previous center and the *j*th current center. This matrix is similar to the distance matrix from im30.mat, but has several key differences:

* The matrix has no zero values since an unsteady camera will never place balls in the exact same pixel location
* The small, but non-zero, values representing the distances between old and current locations of the same ball will not necessarily appear on the diagonal since the clustering algorithm does not always return the detected circles in the same order
* The matrix does not contain two entries of the exact same distance on opposite sides of the diagonal because the distance between the *i*th previous center and the *j*th current center is slightly different than the distance between the *j*th previous center and the *i*th current center (this would be true even if detected circles were returned in the same order)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Current center 1 | Current center 2 | Current center 3 | Current center 4 |
| Previous center 1 |  |  |  |  |
| Previous center 2 |  |  |  |  |
| Previous center 3 |  |  |  |  |
| Previous center 4 |  |  |  |  |

Figure 3: Distance matrix comparing old ball centers from a previous frame with new ball centers in the current frame.

The resulting distance matrix aids first with identifying how much each ball has moved, and then by determining how much the moving ball travelled. The minimum value in each column represents how much each of the four balls has moved from the previous frame. This must be the case assuming three balls are stationary and the fourth moves less than one ball diameter per frame. Finally, the maximum value of the minimum in each column is therefore the distance the moving ball has moved. Three balls (or all four in the first several frames before the moving ball is kicked) will experience “motion” only due to noise or an unsteady camera, whereas the ball with the largest frame-to-frame distance must be the moving ball (or a near-zero effect due to noise if none are moving).

As a final task, complete the following pseudo-code to generate a plot of the moving ball’s speed as a function of time.

previousCenters = detected centers from first frame

while still have frames to read

currentCenters = detected centers from current frame

for i in range(4)

for j in range(4)

Set one element in the distance matrix

Compute minimum of each column of distance matrix

Store the maximum value of the minimum in each column of distance matrix

Update previousCenters

Plot speed as a function of time

**Grading rubric**

1. 25 points: Adjust threshold and blur for im30.mat and use the results to display the ball centers in the image using *k*-means clustering
2. 25 points: Display the distance matrix for the detected balls in im30.mat
3. 25 points: Loop through frames in wiffleBalls.mov, with exactly the four balls detected in every frame
4. 25 points: Plot estimated speed of the moving ball over time