B.Tech Mini-Project Report

Bead Detection and Tracking

Under the guidance of

(Dr. Bosanta R Boruah)

Submitted by:
Tejas Sumukh KL
(Roll no.190121052)

Department of Physics
Indian Institute of Technology, Guwahati
April 2022

Content

1 Introduction

Optical Tweezers

Optical Trapping, also known as Optical Tweezers (OT), is a technique that uses light scattering to hold an object in place. They are based on the principle of light carrying momentum proportional to its energy and propagation direction. When a light ray / laser beam passes through an object, it bends and changes direction (called refraction) and alters its momentum. According to Newton’s third law, the object undergoes an equal and opposite momentum change, a reaction force, for the system to conserve the total momentum. Arthur Ashkin demonstrated that optical forces could displace and levitate micron-sized dielectric particles in both water and air. This work eventually led to the development of the single-beam gradient force optical trap or “optical tweezers”.

Application and Significances of OT

Optical tweezers have been used to trap dielectric spheres, viruses, bacteria, living cells, organelles, small metal particles, and even strands of DNA. They also include confinement and organization, tracking of movement (e.g. of bacteria), application and measurement of small forces, and altering of larger structures (such as cell membranes).

Two of the main applications of optical tweezers have been the study of molecular motors and the physical properties of DNA. In both areas, a biological specimen is biochemically attached to a micron-sized glass or polystyrene bead that is then trapped. By attaching a single molecular motor to such a bead, we have been able to probe motor properties such as: Does the motor take individual steps? What is the step size? How much force can the motor produce? Similarly, by attaching the beads to the ends of single pieces of DNA, experiments have measured the elasticity of the DNA, as well as the forces under which the DNA breaks or undergoes a phase transition.

Aim of the Project

Since most of the application of optical involves trapping and moving of an object, it is helpful to have a program to detect and track the motion of the bead from a stack of images or even from a video clip. So in this project we aim to develop a python program for detecting and tracking the bead from a stack of images and calculating the coordinates of the center of the bead. With a little modification, we can even use video clips instead of stacks of images as a video is basically stacks of images.

2 Methodology

In this section we will explain the methodology used for tracking the moving objects from the stacks of provided images.

2.1 Proposed Scheme

Background subtraction is the commonly used technique to detect moving regions in an image. This method is highly dependent on a good background model.
We can observe that the beads which we want to track are smaller in size compared to the background. Therefore, we can create a background image from the given stacks of images (that is an image with no bead) and then we can subtract each image with the background image.The resultant image will only consist of the pixels which have changed, which can be processed further for tracking the bead and calculating its center.

2.2 Background Model

First we will focus on how to create the background image from the given stacks of images. We have created the background from the method used in [1].
Our grayscale image is imported as a two dimensional array (one can think of it as an 2d matrix where each element a_{ij} contains the pixel value between 0 to 255). Now we are creating the background image by considering the mean of all the values at i^{th} ,j^{th} position of all the images. That is,

$$bg[i][j] = \frac{1}{N}\sum_{k=1}^{N} a_k[i][j]$$

Where: bg is the 2d matrix for the background image.

N is the total number of images in our stack.

a_k represent 2d matrix of k^{th} image

While taking mean is useful and computationally easy, we are going to use median instead of mean as it shows better results compared to mean.

Given below is the code for creating our background image.

```
# Loading Images from a Folder
imgs = load_images_from_folder(f"./Training Set 2/set{s}/")

# Creating Background Image
mean_background = np.median(imgs, axis=0).astype(np.uint8)
```

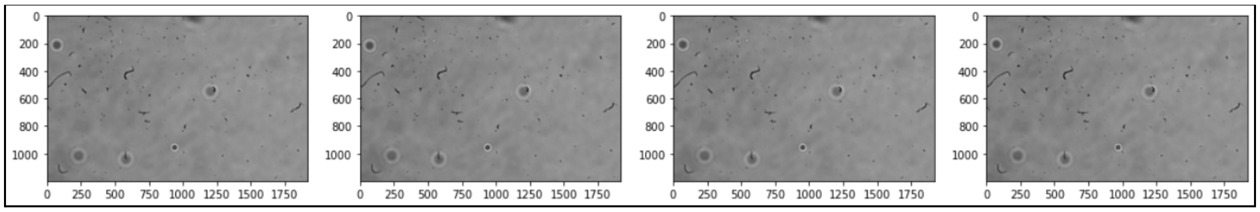


Fig: Input Images

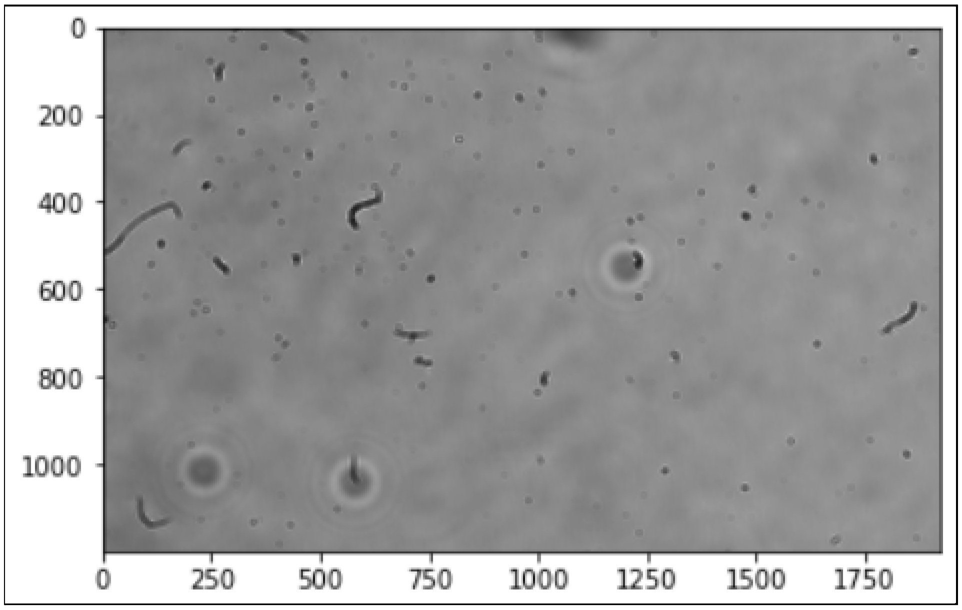


Fig: Background Image

2.3 Background Subtraction / Frame Differencing

Now that we have created our background image, we can do a background separation / frame differencing with each image to obtain the moving regions of the image.

$$diff_k[i][j] = |a_k[i][j] - bg[i][j]|$$

Where: $diff_k$ is the 2d matrix obtained after background subtraction, for the kth image.

bg is a 2d matrix of the background image.

a_k is a 2d matrix of kth image

We can Implement this frame differencing by using the following code

```
# Computing Frame Differencing
diff = cv.absdiff(img,mean_background)
```

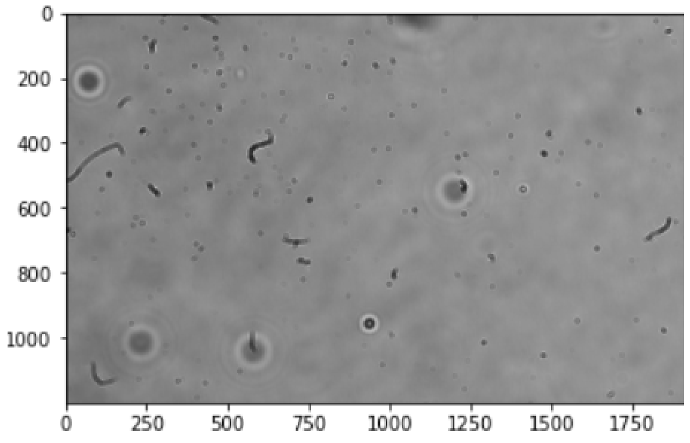


Fig: Input Image (img)

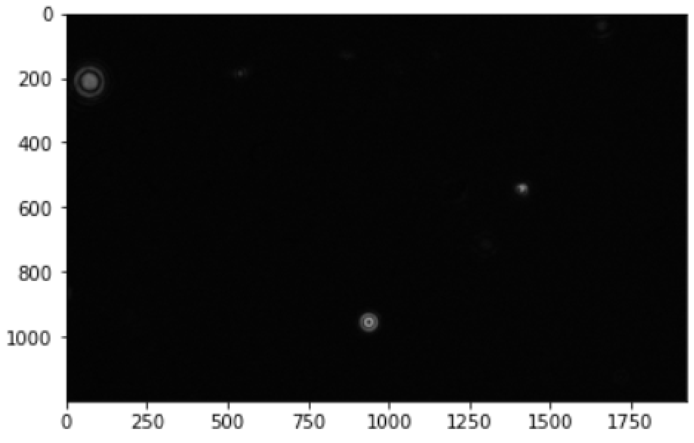


Fig: Subtracted Image (diff)

We have successfully implemented Background Separation. Now we are going to focus on image processing for tracking the bead.

2.4 Noise Reduction

The obtained subtracted image contains a high level of noise. To reduce the noise in the image, we are going to perform a blur/smoothing operation.
As mentioned above, we can reduce the noise in the image by smoothing or blurring the image, and there are many techniques available for performing the said operation. For our program we are going to use median blurring.

For performing median blur operations we will need to define something called kernel size (In this case, the kernel is an unity matrix with an odd number of rows and columns). Now we will consider the matrix elements of the image which are overlapped with the kernel size and replace the center of the overlapped image with its median. Once we perform this for all the matrix elements of the image, we will obtain the median blur of the image.

For example, consider an image matrix of dimension 6x6 and we have chosen the kernel size to be 3x3.

1	25	3	0	0	0
4	15	6	24	2	0
7	48	9	11	41	0
153	127	35	0	1	0
32	126	187	0	22	0
185	34	123	0	234	0

We have overlapped the kernel matrix with the image matrix for the (2,2)th image elemen.Now we are going to the value at 2,2 with the median of the overlapped matrix, which in this case will replace 15 with 7.

We can calculate the median blur using the following code

```
# Computing Blur for the diff image where 17 is the kernel size
blur = cv.medianBlur(diff,17)
```

For a more detailed explanation, please refer to [3].

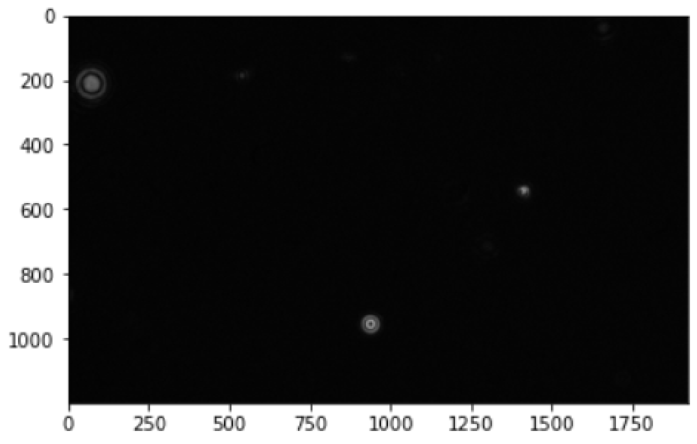


Fig: Input Image(Diff)

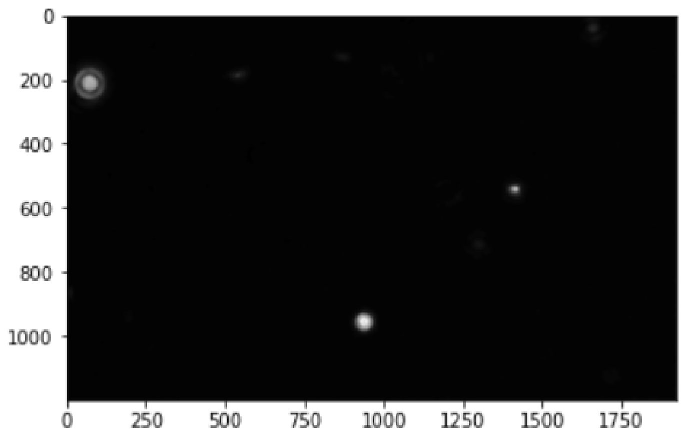


Fig: Blurred Image (blur)

2.5 Pixel Transformation

From the above section we obtained a noise reduced image, now we have to increase the contrast of the image to highlight the bright pixels of the images.
We can increase the contrast and brightness of any image by multiplication and addition with a constant, that is

$$img_k[i][j] \ = \ \alpha * img_k[i][j] \ + \ \beta$$

Where: img_k is the 2d matrix of the kth image.

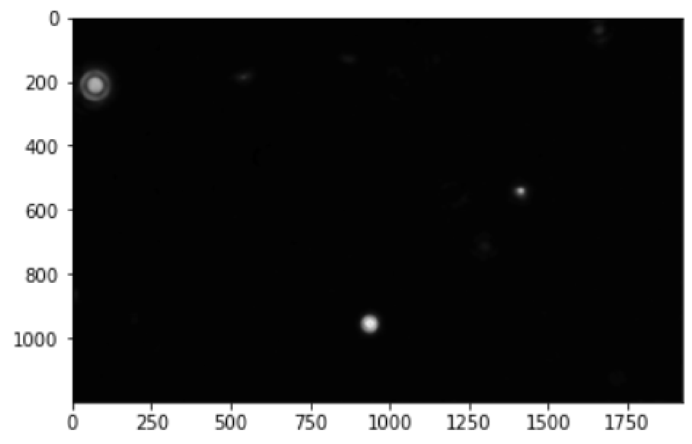
α is a parameter which is used to control contrast of the image, $\alpha > 0$.

β is a parameter which is used to control the brightness of the image.

We can perform this pixel transformation using the following code

```
ad_img = cv.convertScaleAbs(img,alpha = 5,beta = 0)
```

For a more detailed explanation, please refer to [4].



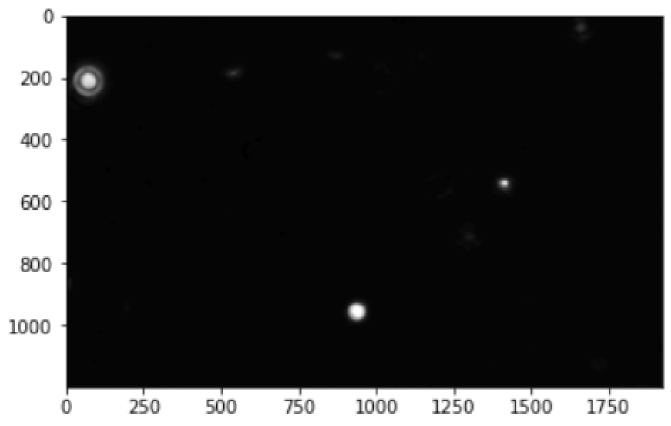


Fig: Input Image (img)

Fig: Pixel Transformed Image(ad_img)

2.6 Image Thresholding

Now we will have to convert the adjusted image or a binary image. Meaning pixel values of the image will either be 0 or 255.

To perform this operation, we will need a threshold value of the pixel. We will access each pixel of the image and check if the pixel value is greater or lesser than the threshold value. If the value is less than the threshold, then we will change the value of the pixel to 0. If it is greater than the threshold then we will change its value to 255. After performing this operation, we will obtain a binary image.

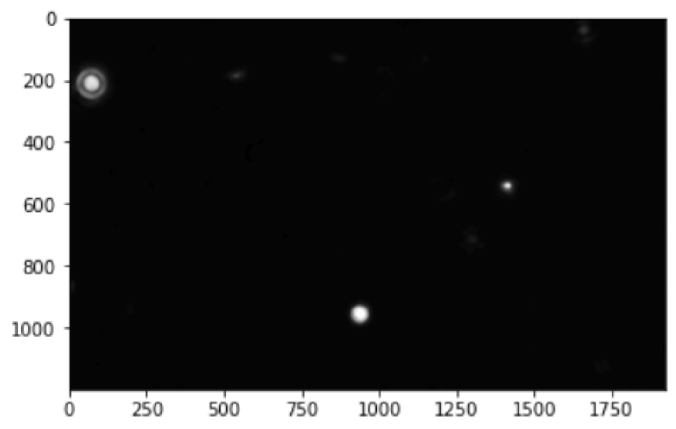
$$img_k[i][j] \quad = \begin{cases} 255 & if\ img_k[i][j] > threshold \\ 0 & if\ img_k[i][j] \leq threshold \end{cases}$$

Where: img_k is the 2d matrix of the k^{th} image.

We can implement image thresholding using the following code

```
# Image Thresholding
_,thresh = cv.threshold(img,threshold,256,cv.THRESH_BINARY)
```

For a more detailed explanation, please refer to [5].



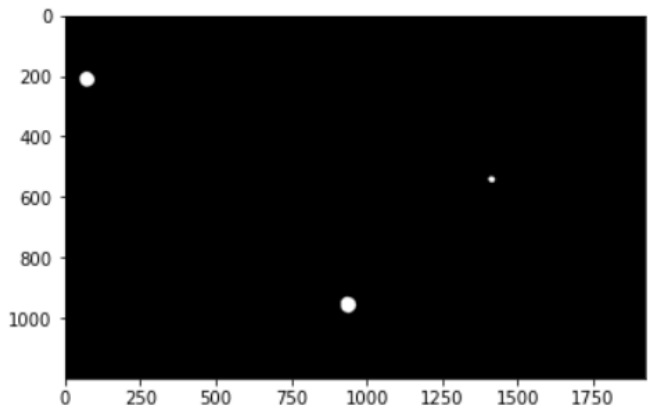


Fig: Input Image(img)

Fig: Binary Image (thresh)

2.7 Calculating Threshold

While binarizing an image, one should be careful while choosing the threshold value. Even in our case a single threshold value is not sufficient for a different variety of images. To overcome the above issue, we have come up with a simple but still efficient solution.

Firstly we have calculated the histogram for the image[]. Basically it contains an array of length 256, where each index contains the value of the number of pixels which has the intensity equal to the index. Once we have a histogram array, we simply find the mean of the index with non zero values.

The function below demonstrates the said concept.

```
def get_thresh(img):
    hist = cv.calcHist([img],[0],None,[256],[0,256])
    n,x = 0,0

    for i in range(len(hist)):
        if(hist[i]):
            x += i
            n += 1
    return(int(x/n))
```

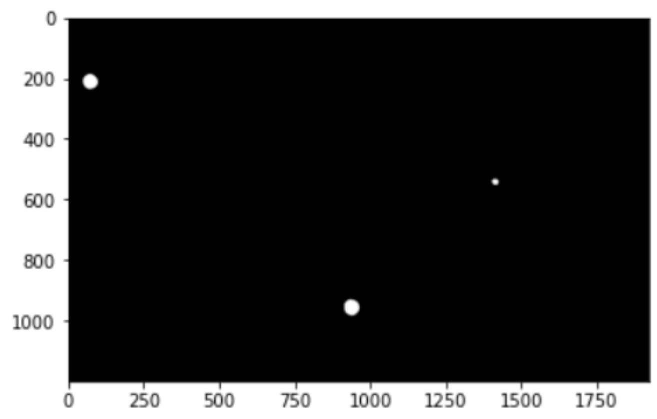
2.8 Morphological Operations

There are many types of morphological operations available. For our program we will use the dilation operation.
The basic purpose of dilation of an image is to increase the brightness of the image around the edges. The implementation of dilation is similar to blurring of the image, only difference being that we will replace the center pixel with the maximum value of the pixel in the over lapped region.

We can dilate an image using the following code

```
# Image Dilation
dilated_img = cv.dilate(img, None, iterations=2)
```

For a more detailed explanation, please refer to [6].



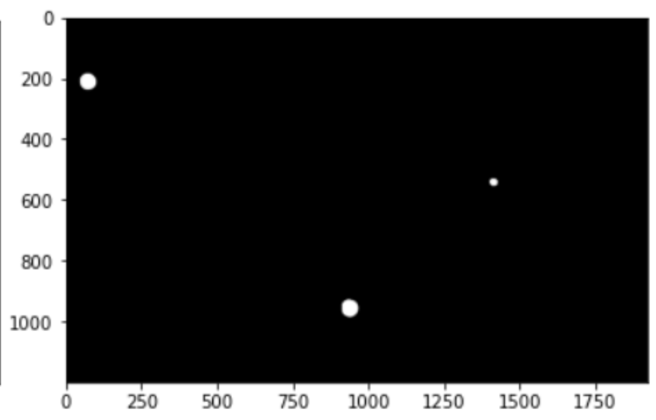


Fig: Input Image (img)

Fig: Dilated Image (dilated_img)

With this we are done with tracking the bead. Following the methodology, we will track not only the bead which we are interested in, but we are tracking all the moving objects in the images. Now we will focus on bead selection and calculating the center of the selected bead.

2.9 Calculating Contours

Till now we have tracked and isolated the moving objects in the images, now we will find the center of all the isolated objects. To do that we will need to calculate contours.
Over here, we can consider contours to be all the points of the edges of each object (in reality contours are more than just edges of an object). We can calculate contours using the method mentioned in [7].

We can find contours of the image using the following code

```
# Calculating Contours from the Image
contours,_= cv.findContours(img,cv.RETR_LIST,cv.CHAIN_APPROX_SIMPLE)
```

For a more detailed explanation, please refer to [7].

3.0 Calculating Center

Once we have the contours for each object, we can use the center of mass concept to find the coordinates of the objects.
Basically we have the coordinates of the edges of the object (i.e contours). We can assume that they have equal weight and calculate the center of mass of the object using the formula:

$$x_{cm} = \frac{\sum_{k=1}^{N} m_k i_k}{\sum_{k=1}^{N} m_k}, \quad y_{cm} = \frac{\sum_{k=1}^{N} m_k j_k}{\sum_{k=1}^{N} m_k}$$

Where: m_k is the mass of the kth pixel (we can assume mass of all pixels to be 1).

i_k is the x coordinate (or in terms of array, 0th position) of the kth pixel

j_k is the y coordinate (or in terms of array, 1th position) of the kth pixel

(we are talking about coordinates of the edges of the object here. The coordinates will be an array of the form [x,y])

We can implement the above concept using the code:

```
# Calculating the Centers of the Object
(x,y),rad = cv.minEnclosingCircle(contour)
```

For a more detailed explanation, please refer to [8].

3.1 Object Selection

As mentioned before, we have found the center of not only the bead which we are interested in but of all the objects which are moving. Now object selection is quite straightforward. Since we have the coordinates of the center of all the objects, we just have to check the distance traveled by each object per frame/ image and we have to consider the coordinates of the object whose difference in center is within a set tolerance.

3 Results and Conclusion

In this section, we will look at a set of input images and the output images of the program.

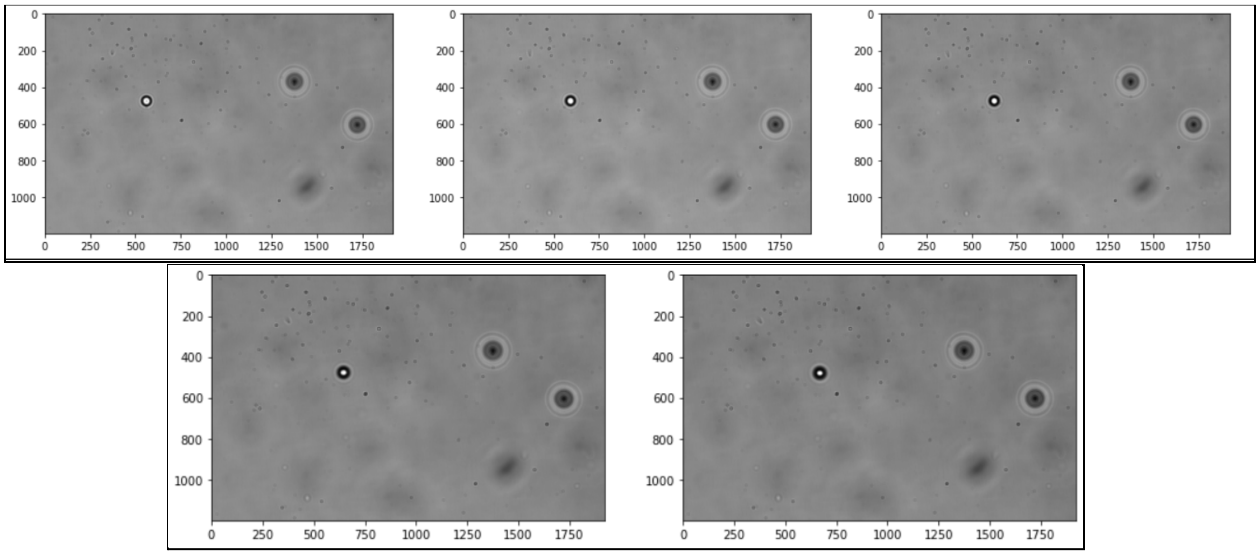


Fig: Input Images, We can clearly see the movement of the beads.

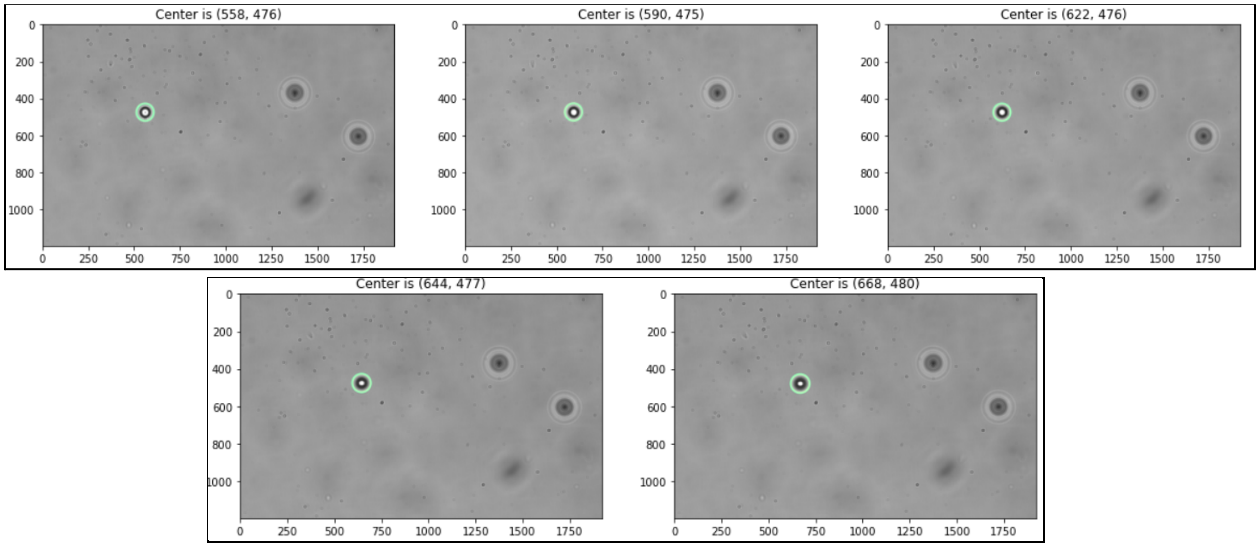


Fig: The output of the program after tracking the bead.

As we can see, our program was successful in detecting, tracking and calculating the center coordinates of the beads in the images.

Conclusion

We have successfully developed a program to detect, track and calculate the coordinates of the center of the beads. This is simple and low in computational power as it does not use any computation demanding techniques such as deep learning or neural networks. However our program will not be able to detect static objects using this technique.The background imagel and the moving objects should be pretty distinguishable. The lighting should also be good. Else, we may face issues like double detections for a single object.

4 References

[1] International Journal of Computer Applications (0975 – 8887) Volume 102– No.7, September 2014 20 Moving Object Detection and Segmentation using Frame differencing and Summing Technique.

[2] https://debuggercafe.com/moving-object-detection-using-frame-differencing-with-opencv/

[3] https://docs.opencv.org/3.4/dc/dd3/tutorial_gausian_median_blur_bilateral_filter.html

[4] https://docs.opencv.org/3.4/d3/dc1/tutorial_basic_linear_transform.html

[5] https://docs.opencv.org/4.x/d7/d4d/tutorial_py_thresholding.html

[6] https://docs.opencv.org/3.4/db/df6/tutorial_erosion_dilatation.html

[7] https://docs.opencv.org/3.4/dd/d49/tutorial_py_contour_features.html

[8] https://docs.opencv.org/3.4/d8/dbc/tutorial_histogram_calculation.html

[9] https://lumicks.com/knowledge/what-are-optical-tweezers/

5 Appendix

Python Implementation

```python
import os
import cv2 as cv
import numpy as np
import matplotlib.pyplot as plt
```

```python
def show(img):
    return plt.imshow(img,cmap="gray")

def img_adj(img,a = 3,b=-550):
    return cv.convertScaleAbs(img, alpha=a, beta=b)

def binarize(img,threshold=128):
    _,thresh = cv.threshold(img,threshold,256,cv.THRESH_BINARY)
    return thresh

def create_blank(img):
    return np.zeros(img.shape,dtype = np.uint8)

def get_contour(img):
    contours,_= cv.findContours(img,cv.RETR_LIST,cv.CHAIN_APPROX_SIMPLE)
    return(contours)

def get_thresh(img):
    hist = cv.calcHist([img],[0],None,[256],[0,256])
    n,x = 0,0

    for i in range(len(hist)):
        if(hist[i]):
            x += i
            n += 1
    return(int(x/n))

def load_images_from_folder(folder):
    images = []
    for filename in os.listdir(folder):
        img = cv.imread(os.path.join(folder,filename),cv.IMREAD_GRAYSCALE)
        if img is not None:
            images.append(img)
    return images

def detect_bead(img,iteration):
    contours = get_contour(img)

    for i in range(iteration):
        pos = []
        b = create_blank(img)

        for contour in contours:
            (x,y),rad = cv.minEnclosingCircle(contour)
            x,y,rad = int(x),int(y),int(rad)
            pos.append((x,y,rad))
            cv.circle(b, (x,y), rad, (57, 255, 20), -1)

        blur = cv.medianBlur(b,17)
        contours = get_contour(blur)
    return(b,pos)
```

```
def draw_circles(ini,fin,ps,save):
    p_imgs = []

    if(save):dims = find_strip_dim(p_pos)

    for i in range(ini,fin):
        t =  cv.merge([imgs[i],imgs[i],imgs[i]])
        blank = create_blank(t)

        for j in range(len(ps[i])):
            cv.circle(blank, (ps[i][j][0],ps[i][j][1]), ps[i][j][2]+15, (57, 255, 20

            if(ini+1==fin):
                blank = cv.putText(blank, f"{j}", (ps[i][j][0],ps[i][j][1]-ps[i][j][

        p_imgs.append(cv.bitwise_or(blank,t))

        if(save):
            cv.imwrite(op_des+f"/Center is {i,(ps[i][j][0],ps[i][j][1])}.png",p_imgs

    return (p_imgs)

def show_highlight(fr,ps,save=False):
    if fr == n:
        return(draw_circles(0,n,ps,save))
    else:
        return(draw_circles(fr,fr+1,ps,save)[0])

def find_strip_dim(ps):
    x_max,y_max = ps[0][0][0],ps[0][0][1]
    x_min,y_min = ps[0][0][0],ps[0][0][1]
    r = 0
    for i in range(n):
        for j in range(len(ps[i])):
            x_max,y_max = max(x_max,ps[i][j][0]),max(y_max,ps[i][j][1])
            x_min,y_min = min(x_min,ps[i][j][0]),min(y_min,ps[i][j][1])
            r = max(ps[i][j][2],r)
    return(y_min-3*r,y_max+3*r,x_min-3*r,x_max+3*r)
```

```
def process_img(img,bg):
    diff = cv.absdiff(img, bg)
    blur = cv.medianBlur(diff,17)
    adjested_img = img_adj(blur,5,0)

    threshold = get_thresh(adjested_img)

    threshed_img = binarize(adjested_img,threshold)
    dilated_img = cv.dilate(threshed_img, None, iterations=2)

    return(detect_bead(dilated_img,3))
```

```
# Loading Images from a Folder
s = 2
# C:\Users\neutr\Desktop\Mini Project\Training Set 2\set2
op_des = "./Procced Img/set2"
ip_des = "./Training Set 2/set2/"
imgs = load_images_from_folder(ip_des)
n = len(imgs)

mean_background = np.median(imgs, axis=0).astype(np.uint8)
```

```python
# Calculating Center Coordinates of all moving objects
pos = []

for i in range(n):
    img,temp = process_img(imgs[i],mean_background)
    pos.append(temp)
```

```python
# Calculating Maximun,Minimum and total number of moving objects
nop = [len(i) for i in pos]
ma = max(nop)
mi = min(nop)
mfn = 0

print(f"Max Number of Moving Object = {ma} and Minimum Number of Moving Object is =

# Finding the Frame where there is maximum number of moving objects
for i in range(len(nop)):
    if nop[i]==ma:
        mfn = i
        break

# Showing the Maximum Number of Moving Object Detected by the Program
show(show_highlight(mfn,pos))
```

```python
# Selecting one object and storing its coordinates
ob = 0
tol = 100
ob_pos = pos[mfn][ob]
p_pos = []
lp = 0

for i in range(mfn,-1,-1):
    for j in range(len(pos[i])):
        if( abs(pos[i][j][0]-ob_pos[0]) < tol and abs(pos[i][j][1]-ob_pos[1]) < tol)

            p_pos.insert(0,[pos[i][j]])
            ob_pos = pos[i][j]
            break

    if lp == len(p_pos):
        p_pos.insert(0,[(0,0,1)])
        lp = len(p_pos)
    else:lp = len(p_pos)


ob_pos = pos[mfn][ob]
for i in range(mfn+1,n):
    for j in range(len(pos[i])):

        if( abs(pos[i][j][0]-ob_pos[0]) < tol and abs(pos[i][j][1]-ob_pos[1]) < tol
            p_pos.append([pos[i][j]])
            ob_pos = pos[i][j]
            break

    if lp == len(p_pos):
        p_pos.append([(0,0,1)])
        lp = len(p_pos)
    else:lp = len(p_pos)
```

```python
# Getting highlighted Images of the object and Saving them
p_imgs = show_highlight(n,p_pos,True)
```