1. Introduction

1.1. Objective

Implementation of object detection through a machine learning application has the potential to act as feedback into a control loop capable of assisting the process to become fully automated, effectively converting traditionally manual processes for the streamlining of redundant work through an automated and predetermined workflow in dynamically changing environments. While advances in micro-scale fabrication, assembly and machining have taken place over the last few decades, so has the attention on developing automated processes [1], [2].

Optically Induced Dielectrophoresis (ODEP) is used to manipulate micro and nano scale particles within a suspension[3]. ODEP is a light actuated electrokinetic technique dependent on the frequency response of a suspended particle within a non-uniform electric field. It has been used in automated systems to separate healthy and cancerous cells in a fluid stream because the two cells response differently to the conditions which cause a DEP force. By flowing a mixture of healthy and cancer cells over a DEP electrode, the two will separate due to the forces applied to the cancer cells. This effectively separates the cells in an automated process. My objective is to implement machine vision as a solution to effectively separate two particles which respond the same to the presence of a DEP force. This will effectively create a sorting process that is no longer dependent on the particle's response.

To manipulate the particles in an ODEP system, light needs to be focused onto the surface of an ODEP chip where a photoconductive thin film becomes more conductive, resulting in an electric field gradient. Particles susceptible to the DEP force respond with either an attraction or repulsion to the induced electric field gradient. The behavior allows for the patterning of light onto the substrate surface to manipulate the particles in a predictable manner.

ODEP has the potential to be used for single cell analysis of bacterial or cancerous cells in automated systems, however when ODEP is used to separate two particles which respond the same to the electric field gradient. To accomplish sorting of two particles which respond the same, I have integrated an existing ODEP systems custom written software to receive the location and class of detected objects within the ODEP chip. The software then generates light patterns and moves the particle from its current position to a predesignated location based on the class. A cascade Haar-basis feature classifier was created for each class and implemented into a python script.

1.2. Significance

As stated, ODEP is effective as it is to manipulate particles and move them based on their response to the presence of a DEP force. An effective method of sorting and separating particles which response the same to ODEP in real-time is a novel endeavor as the process currently can only be done manually. Current DEP based publications have not demonstrated a system capable of producing the result based on the input of from a machine learning technique. Such a technique could open the door to further research on the application of sorting and identifying many different types of biological cells of interest, such as Escherichia Coli (*E. Coli*) or different types of cancer cells which responds the same to DEP[4].

2. Methodology

2.1. System Overview

The system overview as a block diagram is demonstrated in Fig. 1. The process begins by collecting images of the ODEP chip, via video stream, with a camera. Frames are then captured by the cascade classifier script and used to detect objects. The class and location of the detected objects are sent to a server set up within the ODEP systems in-house written software. The class and location data are then used to generate light patterns with movements to move the detected objects to predefined objectives. The generated images are passed to the projection system which scales them down and projects the images on to the surface of the ODEP chip.

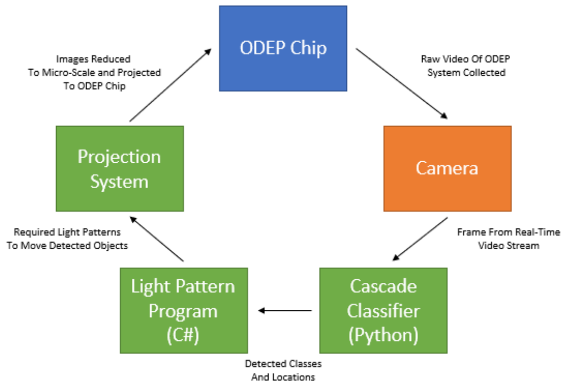

ODEP Chip
Raw Video Of ODEP System Collected
Camera
Frame From Real-Time Video Stream
Cascade Classifier (Python)
Detected Classes And Locations
Light Pattern Program (C#)
Required Light Patterns To Move Detected Objects
Projection System
Images Reduced To Micro-Scale and Projected To ODEP Chip

Figure 1 Experimental System Overview

2.2. Experimental Setup

The ODEP chip thin film layering, and real image are demonstrated in Fig. 2. The A-Si:H layer acts as the photoconductive thin film allowing for the creation of a strong electric field gradient within a microfluidic system in the presence of light. The ports play the role as an inlet and outlet to allow for the supply and collection of the 6/10 um bead mixture.

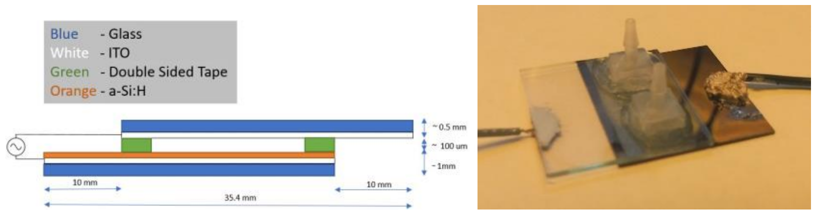


Figure 2 ODEP Chip Overview

From Fig. 3 the ODEP system can be seen to have 5 generalized parts labeled the camera, function generator, microscope, ODEP Chip, and Projection System. While the function generator itself is not in the image, the leads are labeled and can be seen attached to the side of the ODEP chip's mechanical stage. An objective and projector are included in what we call the projection system. We have modified the projector by removing the projection lens and mounted a 20x objective above the projector to act as a condenser, effectively shrinking the projected image down to the micro scale. We can then use mechanical stages to set the working distances for the camera/ microscope and condenser onto the surface of the ODEP chip.

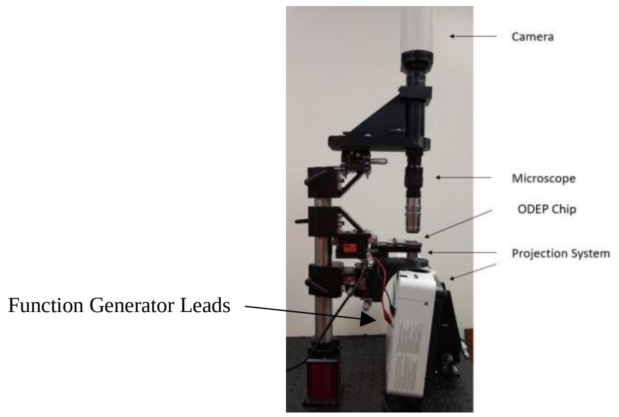


Figure 3 ODEP Setup Overview

Fig. 4 demonstrates the raw image captured from the ODEP chip. There are two regions in the space that have been sectioned off for use as isolation locations for sorted beads, which I will call "objective location". The rectangular light patterns create regions of space the beads can be confined to. The top region is assigned as the objective location for 10um beads, and the lower region is assigned as the objective location for 6um beads. A cross flow is achieved by driving a 3ml syringe with a bead mixture through the channel at a flow rate of 100 ul/hour using a Harvard programmable syringe pump.

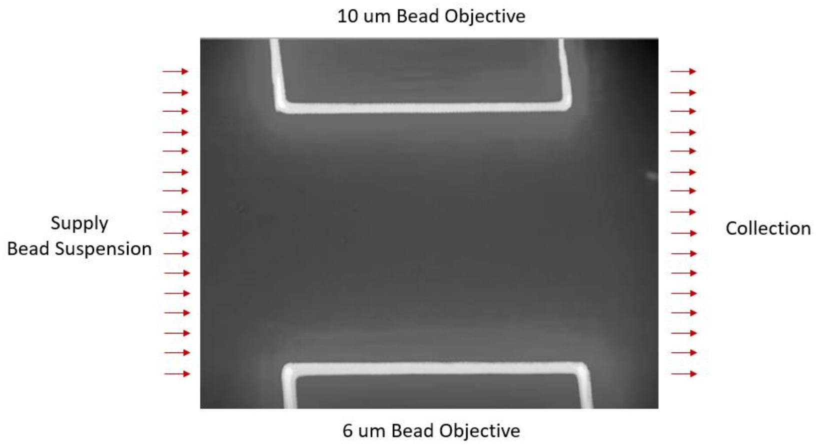


Figure 4 ODEP Chip Experimental Channel

2.3. Data Collection

The cascade classifier used to detect specific objects is trained using what are called positive and negative datasets. Positive datasets contain images limited to the object of interest only and negative images are any relevant image that does not contain the object of interest. It is important that each dataset is prepped and strategically chosen for training. For instance, negative images should not be of things which will not be encountered in the processed images. Therefore, it is best to choose negative images which include objects likely to produce a false positive. It is also important that the object of interest does not exist, even partially, in images contained within the negative dataset. Images withing the positive datasets should be cropped down to a size which removes as much of the background as possible to reduce the possibility of false positives. It is also important the positive images be as comprehensive of the variability seen in the object's characteristics. This means the object to be detected, should be represented in all possible variabilities pertaining to lighting, sizing, rotation, position within image, or focus.

The project objective required the development of datasets for 10 and 6 um beads. Four datasets of positive and negative images were built to train two cascade classifiers for each bead. Positive and negative images are both greyscale images originating from pictures taken of the beads from within the ODEP chip under operational conditions. Due to the similarity in shape and size of the two beads, positive images from each bead were included within the negative dataset for the other bead to reduce the number of false positives. Example images for positive and negative datasets are seen in Fig. 5 and Fig. 6, respectively.

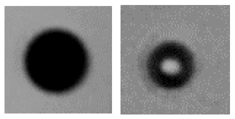


Figure 5 Positive Images for 10um (left) and 6um(right) beads

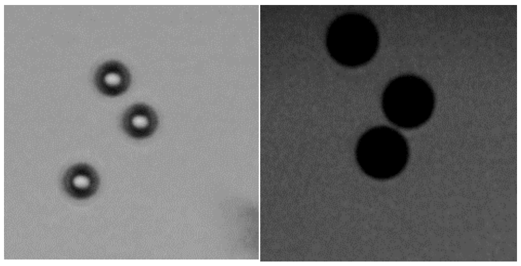


Figure 6 Negative Images for 10um (left) and 6um (right) beads

There were 387 positive and 430 negative images generated for the 6um bead classifier. The 10um bead classifier has 219 positive and 624 negative images generated for its classifier. Differences in dataset sizes reflect difficulties in false positives and missed classifications. For example, the 6um beads were often missed in testing cycles, so significantly more positive images were collected for training until the classifier was able to correctly identify them. While the cascade classifier does not require the training images to be the same size, it does require the images to have the same aspect ratio. A requirement to be discussed in the next section. The images are also relatively dark due to the nature of ODEP, which requires a dark background to create a strong electric field gradient at the site of incident light patterns.

2.4. Model Architecture

The cascade classifier was originally proposed by P. Viola and M. Jones [5] in 2001 for the rapid detection of objects within an image by using Haar-like feature detection. Object detection classifiers of the time were relatively slow, however images with the size 384 x 288 pixels were processed at 15 fps on a 700 MHz Pentium III machine using Viola and Jones cascade classier. This was upwards of 10 times faster than the closest method at the time. The cascade classifiers architecture is structured to be a supervised single class identifier, therefore the number of classifiers to be used must be the same as the number of classes. Their work provides three primary contributions to machine vision.

The first contribution of the classifier, is it is built using a variant of AdaBoost [6], where the selective characteristic features used in classifying is reduced to improve the computational complexity during training. For any image, the number of features can far outnumber the total number of pixels. The methodology works by first training on a very small number of features and validating them. If the true positive and false positive rates are within the set tolerances, the classifier will conclude training, otherwise it will iterate again with a larger or different set of features until tolerances are met.

Second, the classifier operates on a principal they call "integral image", which is the sum of all pixels above and to the left of a selected pixel location, following Equation 1:

$$ii(x,y) = \sum_{x' \le x, y' \le y} i(x',y') \qquad \textbf{[1]}$$

$$s(x,y) = s(x,y-1) + i(x,y) \qquad \textbf{[2]}$$

$$ii(x,y) = ii(x-1,y) + s(x,y) \qquad \textbf{[3]}$$

ii(x,y) is the integral image and i(x',y') is the original image. Equation 1 is predicated on the reoccurrences of Equation 2 and 3 where s(x,y) is the cumulative row sum and i(x,y) is the original image. As demonstrated in Fig. 7, the sum allows for the reduction of image sections into edge, line or four rectangle features, known as Haar-Basis features [7]. The features resemble kernels used in convolutional models. In my opinion, it seems the method simply arrives at the same solution as convolutional models but follows a different path.

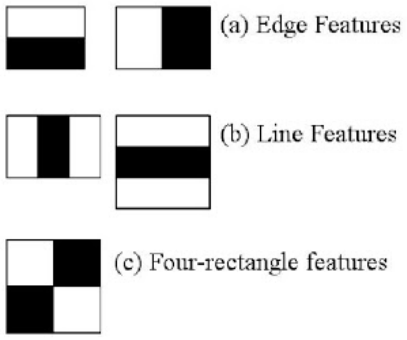
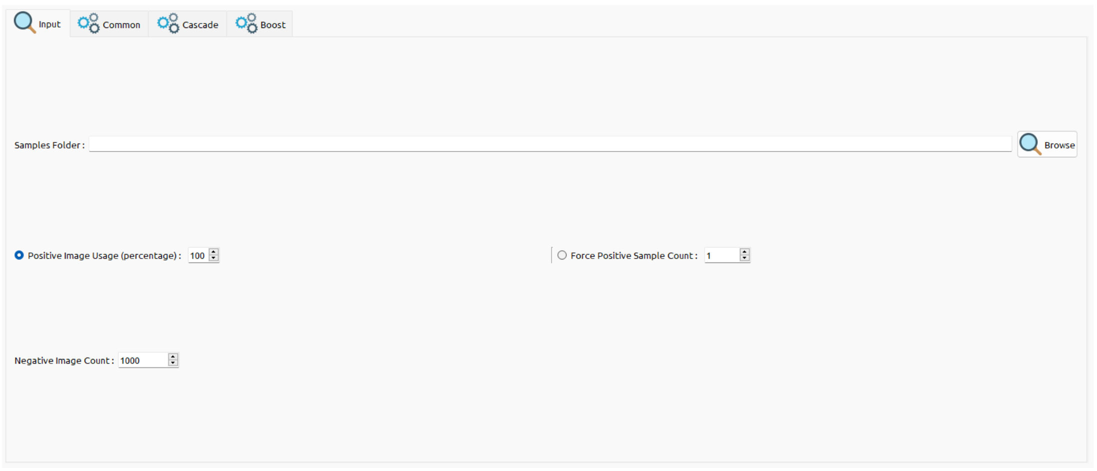


Figure 7 Haar-Basis Features [8]

The reduction of pixels into Haar-features reduces the calculations required on a kernel down to four pixels. This has an added benefits of freeing the feature from scale, allowing the detection of the same features at different scales. Additionally, reducing the time required to process an image.

The third contribution is the selective analysis of regions in an image which will have the highest probability of containing an object. Typical images have many regions that do not contain an object and processing these regions for the extraction of features is wasted resources and time. They achieve the reduction of computation by dividing the image into different regions which are reduced to simple features using what they describe as a "weak classifiers", within a cascade of classifiers to determine the probability the region will containing an object. Regions with a high probability of containing an object continue through the classifiers, while regions deemed as low probability are discarded and no longer analyzed. The structure of cascade classifiers is essentially a decision tree [9], contributing significantly to the speed of processing.

The cascade model was trained in windows OS using Cascade Trainer Gui created by Amin Ahmadi [10]. As the name suggests, the trainer is a graphical user interface (GUI) used to train cascade models. A parent directory containing the positive and negative datasets, labeled "p" and "n" respectively, are supplied to the samples folder textbox on the input tab on the GUI for each class, see Fig. 8. The boost and decision tree parameters are set within the boost tab, which have values of GAB, 0.995, 0.5, 0.95, 1,100 for the boost type, minimal hit rate, maximal false alarm rate, weight trim rate, maximal depth weak tree, and maximal weak trees, respectively. Cascade settings for the sample width and height, and the features to use can be altered on the cascade tab which have values of 40, 40, HAAR, and all for the sample width, sample height, feature type, and HAAR feature types, respectively. Upon successful training, the GUI exports the model as an xml file within the parent directory of the training dataset subdirectories.



Samples Folder :

● Positive Image Usage (percentage) : 100

○ Force Positive Sample Count : 1

Negative Image Count : 1000

Figure 8 Main Page of Cascade Classifier Training GUI

A sample data set for testing was collected alongside the other positive and negative data sets to test the developed model. Testing against still images was done in the GUI by running an image or set of images through the classifier model. Results are given as images where they are output into a folder at a specified location. However, since the project goal is the implementation of the classifier for real time analysis and sorting in an ODEP system, the performance of the classifier is categorized two different ways.

3. Results and Discussion

This section will discuss the process, errors, and results of developing the datasets, software integration between the Python based classifier and C# based ODEP software, and experimental results of the complete system in a real-time application.

3.1. Datasets

Development of a dataset for training proved to be learning experience in and of itself, having it's own challenges and revelations on how to improve the performance of the classifier. The results discussed in this section are the result of many iterations which will not be discussed due to the sheer number volume of discussion which would result. Due to the iterative and time-consuming process of editing and changing datasets, some discussions will have results to discuss which no example exists.

An initial challenge was the false positive rate of empty space. My first positive images were images with a significant amount of background and mostly images containing multiple beads. I found when the classifier was tested, the results would include false positives occurring in empty spaces. Another challenge is demonstrated in Fig. 9 was false positive of 6-um beads which seem to occur at the edge of 10-um beads while missing 6-um beads. This was an interesting problem because it seemed to suggest the classifier was only looking at features existing at the edge of the bead and not using enough features to describe the entirety of the bead itself. I realized one factor contributing to the problems must be due to the amount of background shown in the positive dataset, as they were training the classifier to look at characteristics occurring between and on the edge of corresponding beads. To combat this issue, the positive datasets were cropped so the images would be of one bead filling most of the image, I also generated more images of the background to add in the negative dataset. These changes significantly reduced the false positive rate, as demonstrated in Figs. 9 & 10.

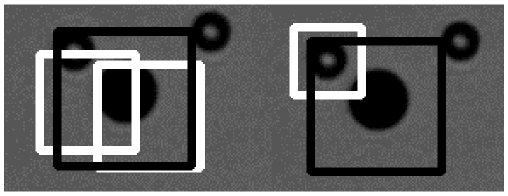


Figure 9 False Positives (left) And Improved Classification (right). 6-um Bead (White Square) 10-um Bead (Black Square)

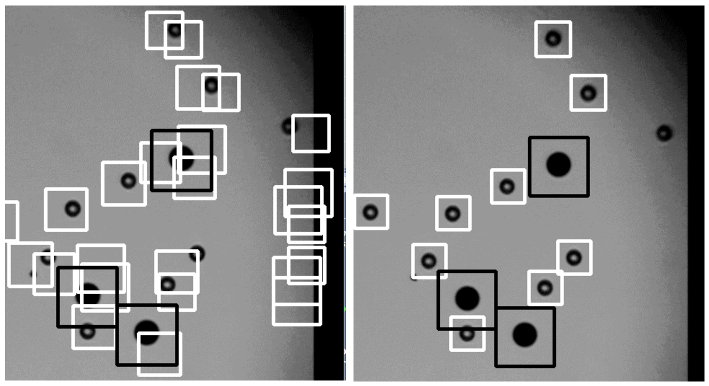


Figure 10 False Positives (left) And Improved Classification (right). 6-um Bead (White Square) 10-um Bead (Black Square)

While false positives were improved by editing the positive images in the dataset and including more background images in the negative dataset, another problem that is not as easily solved are missed objects, which can also be seen in the Fig 9 & 10. In Fig. 11, the unclassified 6um beads and 10-um beads are caused by two known issues, blurring and image brightness.

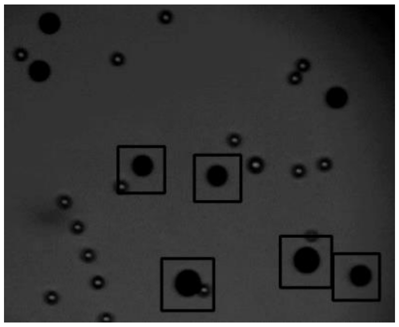


Figure 11 Missed 6-um Objects Due to Focus and Poor Contrast. 6um Bead (white square) 10-um bead (black square)

The image is taken slightly out of focus, causing the surrounding bead edge to blur. It is self-evident from previous experiments that the defining characteristics of the 6um are established around the edge of the bead, blurring the object results in the degradation of the edge features and thus it being missed by the classifier. The second contributing factor was the brightness of the background, which cannot be changed easily due to the configuration of the ODEP setup. For example, the dark corners around the image are due to a truncation of the image from the alignment of the microscope and camera. I have been wrestling with minimizing or eliminating this issue since the development of the ODEP system. To fix missed objects within the image due to focus, I would need to either significantly increase the size of the training datasets to include objects at different focal lengths or confine my experimental setup to only work at a narrow focal length. To address this issue, I chose the latter. I believe an increase in the image brightness could be addressed as a preprocessing technique; however, I did not pursue this in my final solution.

3.2. Software Integration

The development of a Python script for object detection and an existing C# program for creation and movement of light patterns requires a methodology of communication to transfer the location and class of the detected objects. The communication protocol is implemented by first detecting class objects within a video stream in real time, creating a dictionary with the parameters for each detected object, converting the dictionary data into an JSON object, serializing the JSON object via an established TCP Connection to a server inside the light pattern program, and finally deserializing the JSON object within a C# class object. This process is demonstrated in Fig. 12. Choosing to use a TCP connection between the two programs has the benefit of doing the analysis on a specialized and dedicated computer, as opposed to sharing resources to run both the Python script and C# program.

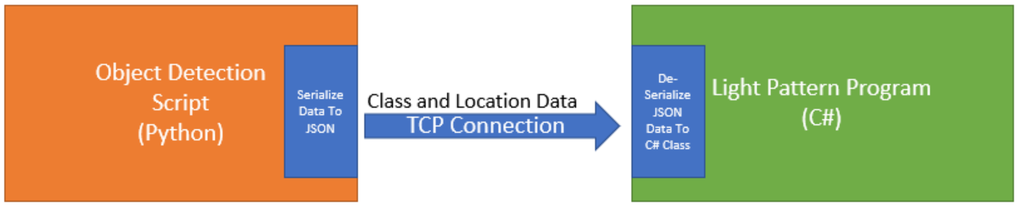
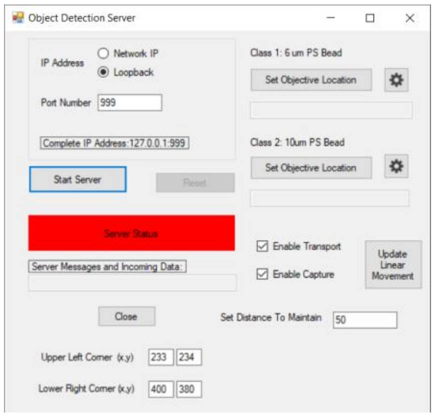

Object Detection Script (Python)
Serialize Data To JSON
Class and Location Data
TCP Connection
De-Serialize JSON Data To C# Class
Light Pattern Program (C#)

Figure 12 Block Diagram of Communication Protocol

The Python code is relatively simple in comparison to the C# code as it does nothing more than an infinite loop of detecting objects within a captured frame, transmitting that data to the C# program, and displaying indicating the location of the detected objects in a video stream on the host computers monitor. It is the job of the C# program to determine how to use the data. The program was already written for the purpose of smooth translation of object within an ODEP system, so methods of transporting an object from point 1 to point 2 were utilized for this project. It operates by creating a light shape to capture the particle and moving the object to a pre-defined objective location based on the objects class. Fig. 13 demonstrates the GUI interface created for the server on the C# program.



IP Address: ( ) Network IP [✓ Loopback]
Port Number: 999
Complete IP Address:127.0.0.1:999
Server Messages and Incoming Data:
Class 1: 6 um PS Bead
Class 2: 10um PS Bead
☑ Enable Transport
☑ Enable Capture
Set Distance To Maintain: 50
Upper Left Corner (x,y): 233 234
Lower Right Corner (x,y): 400 380

Figure 103 Server GUI on C# Program

The server is designed so the client program (Python script) can run on the network or on the same computer by initializing the server on the localhost or on the LAN. Loopback is necessary if the python script is running on the same machine, however the LAN server is needed if the script is running remotely. This can be set by selecting the 'Network IP' or the 'Loopback' radio-button. The 'Set Objective location' button on under each class allows the user to designate the objective location to transport each detected object to. The 'distance to maintain' textbox sets a specified distance to attempt to maintain between captured objects. This feature does cause a few bugs when the program is in operation, however with some work it can be fixed. It is important the distance between light patterns is maintained as each pattern can interfere with the ability of another light pattern to maintain control of a particle. Checkboxes to enable capture and transportation of particles are included to enable some user control in case a failure were to occur during development. The button to 'Update Linear Movement' causes the program to begin transporting the objects after the transportation button has been reenabled if objects were detected and captured while off. Four textboxes are placed corresponding to two labels called 'Upper Left Corner' and 'Lower Right Corner'. These boxes allow for scaling of the bead's location in the captured image frame to the projected image created in the program. If further development were to be pursued, corrections to the internal code would be applied to allow for one-to-one scaling to eliminate these parameters.

3.3. Experimental Results

Fig 14 demonstrates successful sorting of 10-um and 6um particles from the crossflow. In frame 1 of the figure, two 10-um beads can be seen after being captured and moved to the location of their class objective. In frame two, one of the beads begins to cross the border into the objective region while the other follows closely behind. In frame 3, it can be observed that both 10-um beads have almost entirely passed into the objective location and a 6-um bead that has entered the passage has just been captured. Frame 4 shows the two rings moving the 10-um beads have disappeared and the 6-um bead is entering the objective location. Frame 5 shows the successful capture of the 6-um bead in its corresponding objective. The translation occurred over the span of approximately 22 seconds.

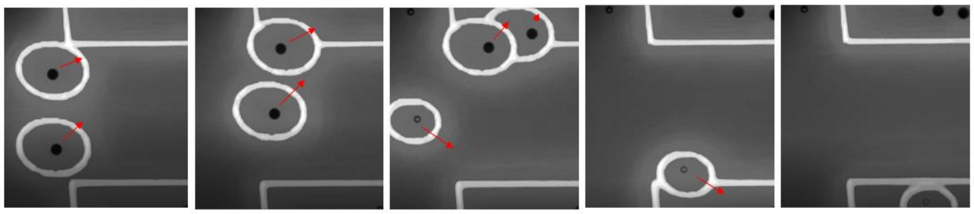


Figure 14 Time Sequence of Automated Sorting over 22 seconds. Read from left to right

While the sequence demonstrates successful capturing and translating of beads to the correct objective, it does not mean there were not inaccuracies in the real-time classification of the beads. It was an unintended but welcomed result that some measures the C# program took to prevent duplicated rings also prevented momentary false positives from creating useless problematic light rings. This demonstrates that while it is ideal to optimize the classifier to eliminate false positives, performance in a real-time application can be effectively increased or leveraged through programmatic means. Fig. 15 demonstrates frames containing squares with false positive identifiers which did not result in the creation of a light ring due to the detected objects proximity to another ring.

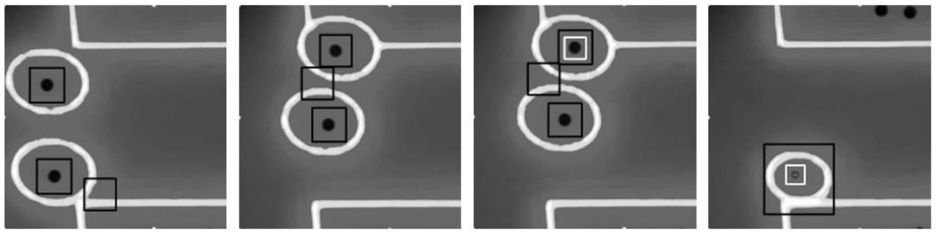


Figure 115 Frames Containing Misclassified Samples. Black Squares Represent a 10-um Bead Classification While White Squares Represent 6um beads.

Of the four frames containing false positive results, only one frame displays a false positive box for a 6um bead. All frames contained a false positive identifier for the 10-um bead. It is an interesting result that the false positive identifier for all 10-um bead occurred around light shapes, particularly in spaces confined by two light shapes. The consistency of the false positives occurring consistently in similar conditions suggests the datasets could be expanded to include similar images to reduce or eliminate the false positives. In addition, it is also possible the condition to pass on the false positive rate or “maximal false alarm rate” could be decreased to eliminate the occurrence of this type of false positive.

References

[1] S. Chen, J. Chen, X. Zhang, Z.-Y. Li, and J. Li, “Kirigami/origami: unfolding the new regime of advanced 3D microfabrication/nanofabrication with ‘folding,’” *Light Sci. Appl.*, vol. 9, no. 1, p. 75, Dec. 2020, doi: 10.1038/s41377-020-0309-9.

[2] A. Hsu *et al.*, “Automated 2D micro-assembly using diamagnetically levitated milli-robots,” in *2017 International Conference on Manipulation, Automation and Robotics at Small Scales (MARSS)*, Montreal, QC, Jul. 2017, pp. 1–6. doi: 10.1109/MARSS.2017.8001926.

[3] Pei Yu Chiou, Zehao Chang, and M. C. Wu, “A novel optoelectronic tweezer using light induced dielectrophoresis,” in *2003 IEEE/LEOS International Conference on Optical MEMS (Cat. No.03EX682)*, Waikoloa, HI, USA, 2003, pp. 8–9. doi: 10.1109/OMEMS.2003.1233441.

[4] M. Dudaie *et al.*, “Label-free discrimination and selection of cancer cells from blood during flow using holography-induced dielectrophoresis,” *J. Biophotonics*, vol. 13, no. 11, Nov. 2020, doi: 10.1002/jbio.202000151.

[5] P. Viola and M. Jones, “Rapid object detection using a boosted cascade of simple features,” in *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, Kauai, HI, USA, 2001, vol. 1, p. I-511-I–518. doi: 10.1109/CVPR.2001.990517.
[6] Y. Freund and R. E. Schapire, “A desicion-theoretic generalization of on-line learning and an application to boosting,” in *Computational Learning Theory*, vol. 904, P. Vitányi, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 1995, pp. 23–37. doi: 10.1007/3-540-59119-2_166.
[7] C. P. Papageorgiou, M. Oren, and T. Poggio, “A general framework for object detection,” in *Sixth International Conference on Computer Vision (IEEE Cat. No.98CH36271)*, Bombay, India, 1998, pp. 555–562. doi: 10.1109/ICCV.1998.710772.
[8] “Cascade Classifier,” *opencv.org*, [Online]. Available: https://docs.opencv.org/4.x/db/d28/tutorial_cascade_classifier.html
[9] Y. Amit, D. Geman, and K. Wilder, “Joint induction of shape features and tree classifiers,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 19, no. 11, pp. 1300–1305, Nov. 1997, doi: 10.1109/34.632990.
[10] A. Ahmadi, “CASCADE TRAINER GUI”, [Online]. Available: https://amin-ahmadi.com/cascade-trainer-gui/