Supporting Materials

Calibration Process

To convert lengths in pixels to metric units, an object of known length must be identified and measured in the frame. The ArUco marker shown in Figure 2b was printed out and placed in the same plane as the plasma plume. We measured the side length of the ArUco marker using a standard ruler to get the known side length, pressing the 'c' key while running the program to calibrate the camera position. This triggered the program to find an ArUco marker if one was present in the frame and identified the pixel locations of the four corners. From the pixel locations of the corners, we calculated the lengths (in pixels) of all four sides using the distance formula, averaged them together, and then divided by the physically known side length in cm. This calibration process returned a px/cm value and stored it for future use. Once calibrated, the ArUco marker was removed from the camera frame. Recalibration was done if the distance between the plasma plume and camera changed.

Equipment

APPJ source was housed in a dark plastic box. Data was collected inside the box to minimize air disturbances on the plasma plume's shape and provide a solid background image, making recognition easier. A cell phone (iPhone 11) was attached to a tripod inside the box and fed its video stream to the laptop running the CV program. This was accomplished using the Camo application.

Main Program

```
#from cmath import sqrt
#from operator import delitem
#from turtle import width
import cv2
import numpy as np
import time

from aruco_calibrate import find_aruco
from object_detector import *
# Load Object Detector
detector = HomogeneousBgDetector()

record_data = False
num_points = 0
plumeLen = []
global pixels_per_cm

def nothing(x):
    pass

cap = cv2.VideoCapture(0)

cv2.namedWindow("Tracking")
cv2.createTrackbar("LH", "Tracking", 120, 180, nothing)
cv2.createTrackbar("LS", "Tracking", 119, 255, nothing)
cv2.createTrackbar("LV", "Tracking", 115, 255, nothing)
cv2.createTrackbar("UH", "Tracking", 255, 180, nothing)
cv2.createTrackbar("US", "Tracking", 255, 255, nothing)
```

```
cv2.createTrackbar("UV", "Tracking", 255, 255, nothing)
#cv2.createTrackbar("Data Coll", "Tracking", 0, 255, nothing)



def makeVideo(frame):

    hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)

    l_h = cv2.getTrackbarPos("LH", "Tracking")
    l_s = cv2.getTrackbarPos("LS", "Tracking")
    l_v = cv2.getTrackbarPos("LV", "Tracking")

    u_h = cv2.getTrackbarPos("UH", "Tracking")
    u_s = cv2.getTrackbarPos("US", "Tracking")
    u_v = cv2.getTrackbarPos("UV", "Tracking")

    l_b = np.array([l_h, l_s, l_v])
    u_b = np.array([u_h, u_s, u_v])

    mask = cv2.inRange(hsv, l_b, u_b)
    res = cv2.bitwise_and(frame, frame, mask=mask)

    contours = detector.detect_objects(res)
    for cnt in contours:
        # Get rect
        rect = cv2.minAreaRect(cnt)
        global w
        global h
        (x, y), (w, h), angle = rect
        # Display rectangle
        box = cv2.boxPoints(rect)
        box = np.int0(box)
        cv2.circle(res, (int(x), int(y)), 5, (0, 0, 255), -1)
        cv2.polylines(res, [box], True, (255, 0, 0), 2)

        object_width = w
        object_height = h

        cv2.putText(res, "Width {} px".format(round(object_width, 1)), (int(x - 100), int(y - 20)), cv2.FONT_HERSHEY_PLAIN,
2, (100, 200, 0), 2)
        cv2.putText(res, "Height {} px".format(round(object_height, 1)), (int(x - 100), int(y + 15)), cv2.FONT_HERSHEY_PLAIN,
2, (100, 200, 0), 2)

    cv2.imshow("frame", frame)
    cv2.imshow("mask", mask)
    cv2.imshow("res", res)
```

```
while True:
    _, frame = cap.read()

    makeVideo(frame)

    key = cv2.waitKey(1)



    #------------------------------------KEY FUNCTIONALITY----------------------------
    if key == 27:
        break

    #if you hit c, program will detect aruco and print the 4 corners
    if key== ord('c'):
        corners = find_aruco(frame,annotate=True)
        print(corners)
        #[topLeft, topRight, bottomRight, bottomLeft]
        '''
        [1      2]

        [4      3]
        '''
        try:
            x1, y1 = corners[0]
            x2, y2 = corners[1]
            x3, y3 = corners[2]
            x4, y4 = corners[3]

            topSide = np.sqrt((x2-x1)**2 + (y2-y1)**2)
            rightSide = np.sqrt((x2-x3)**2 + (y2-y3)**2)
            bottomSide = np.sqrt((x4-x3)**2 + (y4-y3)**2)
            leftSide = np.sqrt((x4-x1)**2 + (y4-y1)**2)

            avgLength = (topSide + rightSide + bottomSide + leftSide)/4
            pixels_per_cm = (avgLength)/0.8   #pxels per 5 cm (length of aruco) small aruco is 0.8 cm
            print(pixels_per_cm)
            print(f"calibrated at: {pixels_per_cm} px/cm")
        except:
            x1, y1 = [0,0]
            x2, y2 = [0,0]
            x3, y3 = [0,0]
            x4, y4 = [0,0]
            pixels_per_cm = 0
            print(pixels_per_cm)
            print('no aruco')
```

```
#-----------------FILE NAME--------------
#voltage = 'vol-vol'
#current = 'cur-cur'
#filename = voltage + 'volt_' + current + 'amp.txt'
run_num = 'NUMBER'
filename = 'run_' + run_num + '.txt'
numDataPts = 100
#-----------------------------------------

#if space bar collect data
#plumeLen = []
#press space bar to start data collection

#if press space bar and not currently taking data, set record data to true
if key == 32 and not record_data:
    record_data = True
    global start_time
    start_time = time.time()
#append w to plume length

if record_data:
    data_pont = []
    #data_pont.append(time.time())
    data_time = round(time.time() - start_time, 3)
    data_pont.append(data_time)
    if w>h:
        data_pont.append(round(w/pixels_per_cm , 3))
    else:
        data_pont.append(round(h/pixels_per_cm , 3))
    print(data_pont)
    plumeLen.append(data_pont)
    #plumeLen.append(w)
#once we have taken all the data points,
if len(plumeLen) >= numDataPts:
    record_data = False     #stop recording
    data = np.array(plumeLen)       #convert to np arryay
    #if len(data) > 0:
    saver = np.savetxt(filename, data)  #save that shit
    print('saved')
    plumeLen = []   #reset plume length array so there is no builldup



#press p and parameters will pring out
if key == ord('p'):
    l_h = cv2.getTrackbarPos("LH", "Tracking")
    l_s = cv2.getTrackbarPos("LS", "Tracking")
    l_v = cv2.getTrackbarPos("LV", "Tracking")
```

```
u_h = cv2.getTrackbarPos("UH", "Tracking")
u_s = cv2.getTrackbarPos("US", "Tracking")
u_v = cv2.getTrackbarPos("UV", "Tracking")

print(f"Lower hue: {l_h}")
print(f"Lover sat: {l_s}")
print(f"Lower value: {l_v}")
print(f"Upper Hue: {u_h}")
print(f"Upper Sat: {u_s}")
print(f"Upper Value: {u_v}")
print('---------')
```