SUPPLEMENTARY MATERIALS
Smartphone-based sensitive detection of SARS-CoV-2 from saline gargle samples via flow profile analysis on a paper microfluidic chip

Patarajarin Akarapipad[1,†], Kattika Kaarj[2,†], Lane E. Breshears[1,†], Katelyn Sosnowski[1,†], Jacob Baker[1], Brandon T. Nguyen[1], Ciara Eades[3], Jennifer L. Uhrlaub[4], Grace Quirk[5], Janko Nikolich-Žugich[4], Michael Worobey[5], and Jeong-Yeol Yoon[1,2,3,*]

[1]Department of Biomedical Engineering, The University of Arizona, Tucson, Arizona 85721, United States
[2]Department of Biosystems Engineering, The University of Arizona, Tucson, Arizona 85721, United States
[3]Department of Chemistry & Biochemistry, The University of Arizona, Tucson, Arizona 85721, United States
[4]Department of Immunobiology and Arizona Center on Aging, The University of Arizona College of Medicine, Tucson, Arizona 85724, United States
[5]Department of Ecology and Evolutionary Biology, The University of Arizona, Tucson, Arizona 85721, United States

[†]These authors contributed equally.
[*]Corresponding author. E-mail: jyyoon@arizona.edu.

Supplementary Methods

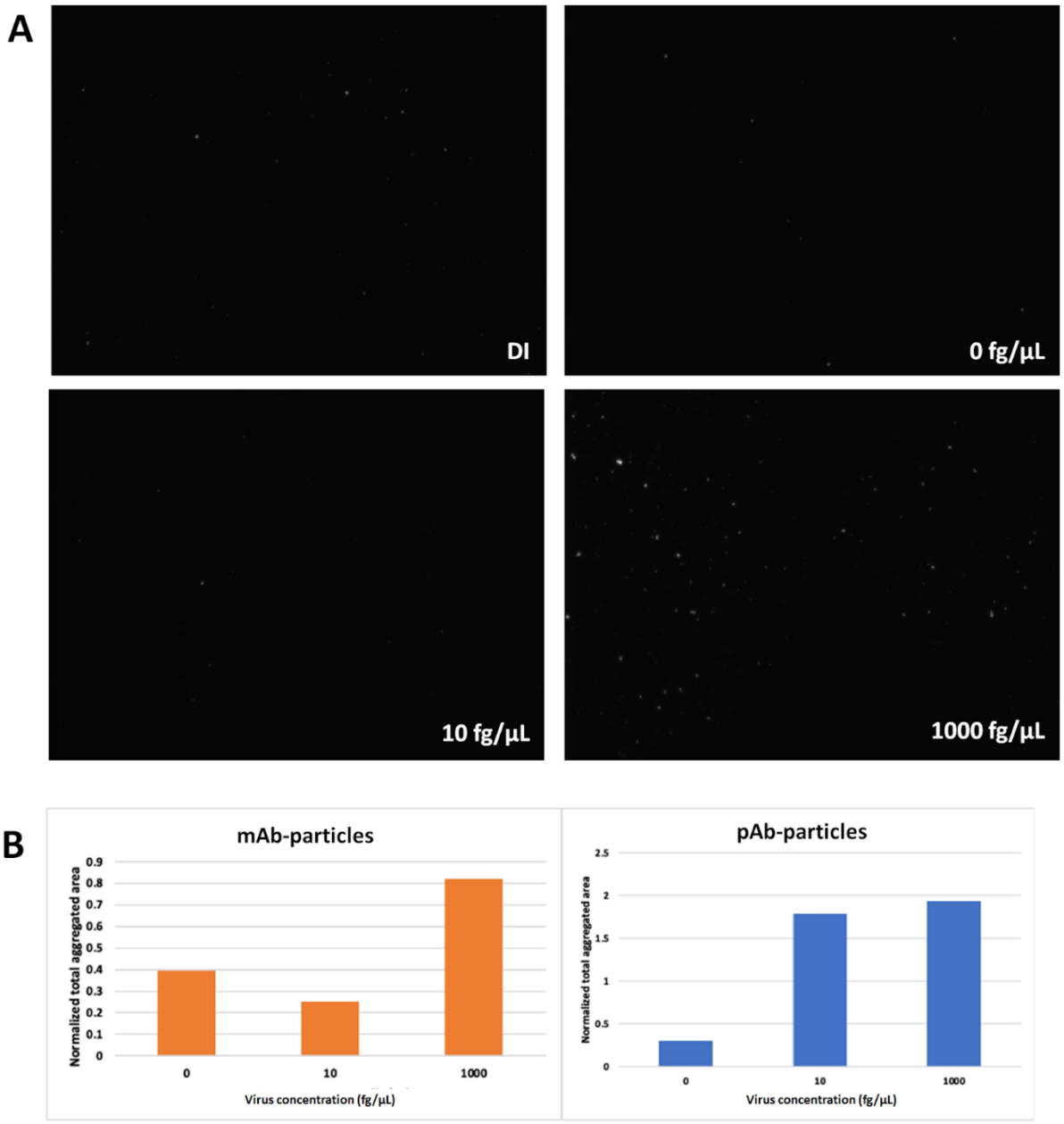
Pendant Droplet Experiment for Surface Tension Analysis

Surface tension of sample droplets was found using the optical pendant droplet method. Sample aliquots were loaded into a syringe fitted with a blunt needle tip (catalog #80086-154, VWR International, Radnor, PA, USA). The syringe was depressed as far as possible without completely discharging the sample. Photos were taken of the droplet shape at 0, 2, 4, 6, 8, and 10 seconds after depressing the syringe. The stabilized final value was chosen. This was accomplished by bolting a 3D-printed smartphone camera holder to the table, then fixing a lamp behind the droplet to assure consistent ambient illumination. Photos were uploaded to ImageJ for pendant drop analysis using a plugin (https://github.com/adaerr/pendent-drop). Photos were converted to the monochromatic 8-bit type, and a line segment was drawn across the length of the needle tip to set the scale to the known 1.27 mm tip diameter. A rectangle was drawn around the droplet before selecting the Pendant Drop plugin. The plugin was allowed to fit for the capillary length, tip x coordinate, tip y coordinate, and tip radius of curvature parameters, then the user selected "preview" and chose a starting capillary length between 1.0 and 3.0 before starting the analysis. The parameters were fitted automatically, and a surface tension value was displayed. The surface tension value was accepted as long as the pendant droplet fit was judged to be accurate (i.e. the red line generated around the droplet represented a close fit).

Bradford Assay for Total Protein Analysis

The Bradford Assay was used to determine the overall protein content of the saliva samples. A 96-well microwell plate (Cellstar 96 Well Cell Culture Plate, Greiner Bio-One, Frickenhausen, Germany) was used to conduct the Bradford Assay. The standard curve was developed using stock bovine serum albumin (BSA) concentrations from the Quick Start Bradford Protein Assay (Bio-Rad Laboratories, Inc., Hercules, CA, USA). Two additional higher concentrations of BSA were manually diluted into deionized water, which were later discarded as they did not form a linear trend and the sample protein contents were much lower. The microwell plate (Supplementary Figure S4) was loaded from rows A-C and columns 1-10 with 150 µL per well of the Bradford assay reagent. In rows A-C of column 1, 5µL of deionized water per well was added and mixed with a micropipette. Rows A-C of columns 2-10 were filled with 5 µL per well of the BSA standards and mixed. Then, 5 µL of the tested samples were placed in separate wells with 150 µL of the Bradford assay reagent and mixed. Due to the lack of saliva sample volume after running flow rate assays, the tested samples were able to fill between 1 and 3 wells (5-15 µL) while some samples did not have enough volume remaining to fill any wells. The analysis of this assay was done using an optical probe (BIF600-2, Ocean Insight, Orlando, FL, USA) connected to a spectrometer (USB400, Ocean Insight). The microwell plate was placed on a white background and the probe was used to measure the intensity of each well at 595 nm. Absorbance of each well was calculated using $A = \log_{10} (I_0 / I)$, where A is absorbance, I_0 represents the average intensity of the wells with deionized water, and I represents the average intensity of the target wells of the same concentration. When using the probe, the intensity values fluctuated substantially, which led

to inconclusive results. Therefore, it was decided to use an alternative method based on smartphone images analyzed using ImageJ. A photo in ambient lighting of the microwell plate against a white background was taken with a Samsung S10 (Samsung, San Jose, CA, USA) smartphone and uploaded into ImageJ. This photo type was changed to RGB stacking and only the red image was analyzed, as the Bradford dye emits a blue color for high protein content and therefore absorbs red. By using the ellipse tool, a circle with a diameter of 25 pixels was made. This circle was placed within each well in two different locations and the average intensity measurements over these two circles were taken. Intensities were further averaged between two researchers' results. From here, the absorbance was calculated as described above. A standard curve was made using the known BSA concentrations graphed against their absorbances. Then this standard curve was used to estimate the overall protein concentration in each saliva sample.

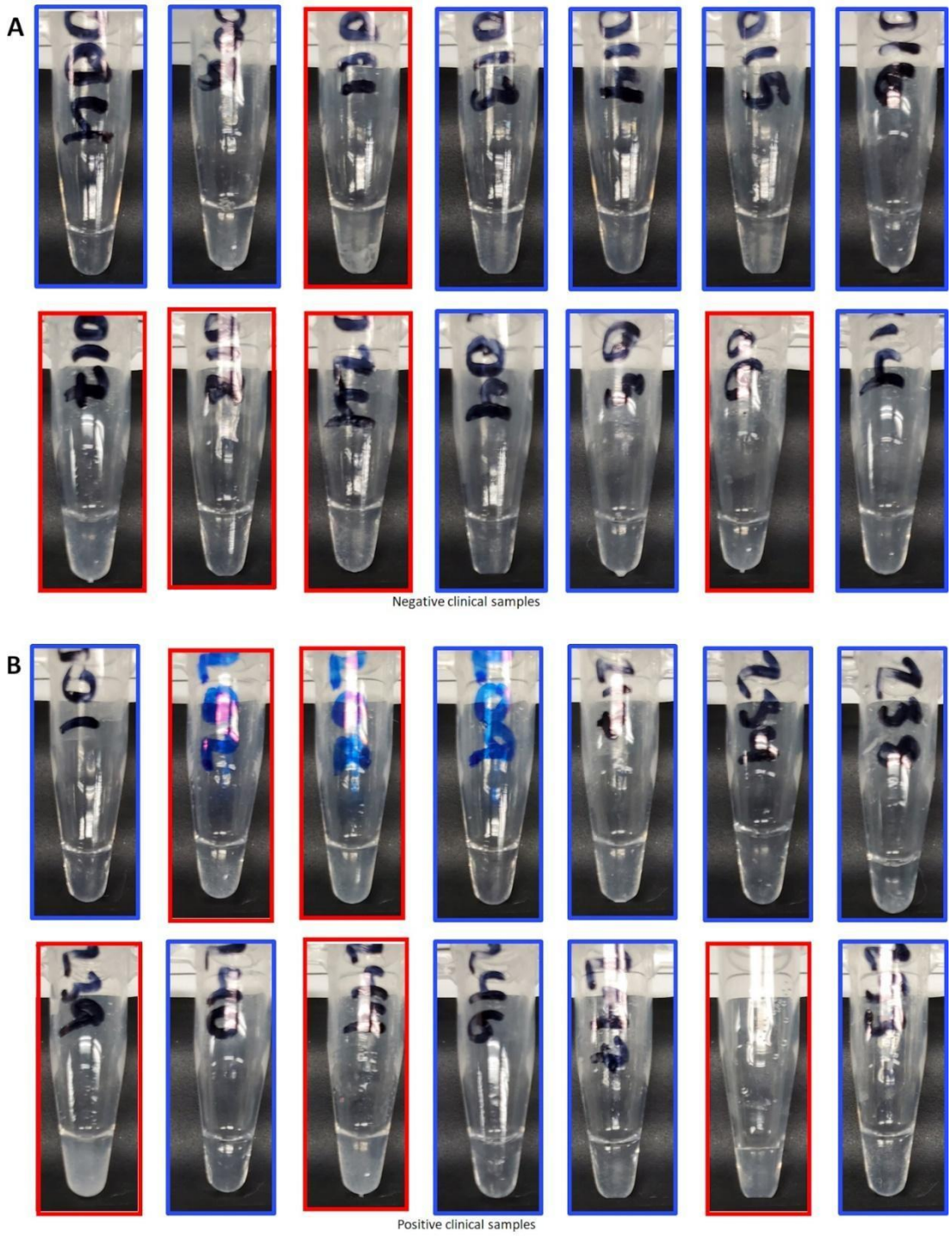


Supplementary Figure S1. Confirmation of immunoagglutination on microscope glass slides. (A) Fluorescence microscopic images of mAb-particles pre-mixed with DI water (without saliva), 0, 10, and 1,000 fg/µL SARS-CoV-2 in 1% v/v human pool saliva diluted in DI water. (B) Normalized pixel areas of immunoagglutinated particles using mAb- and pAb-particles.

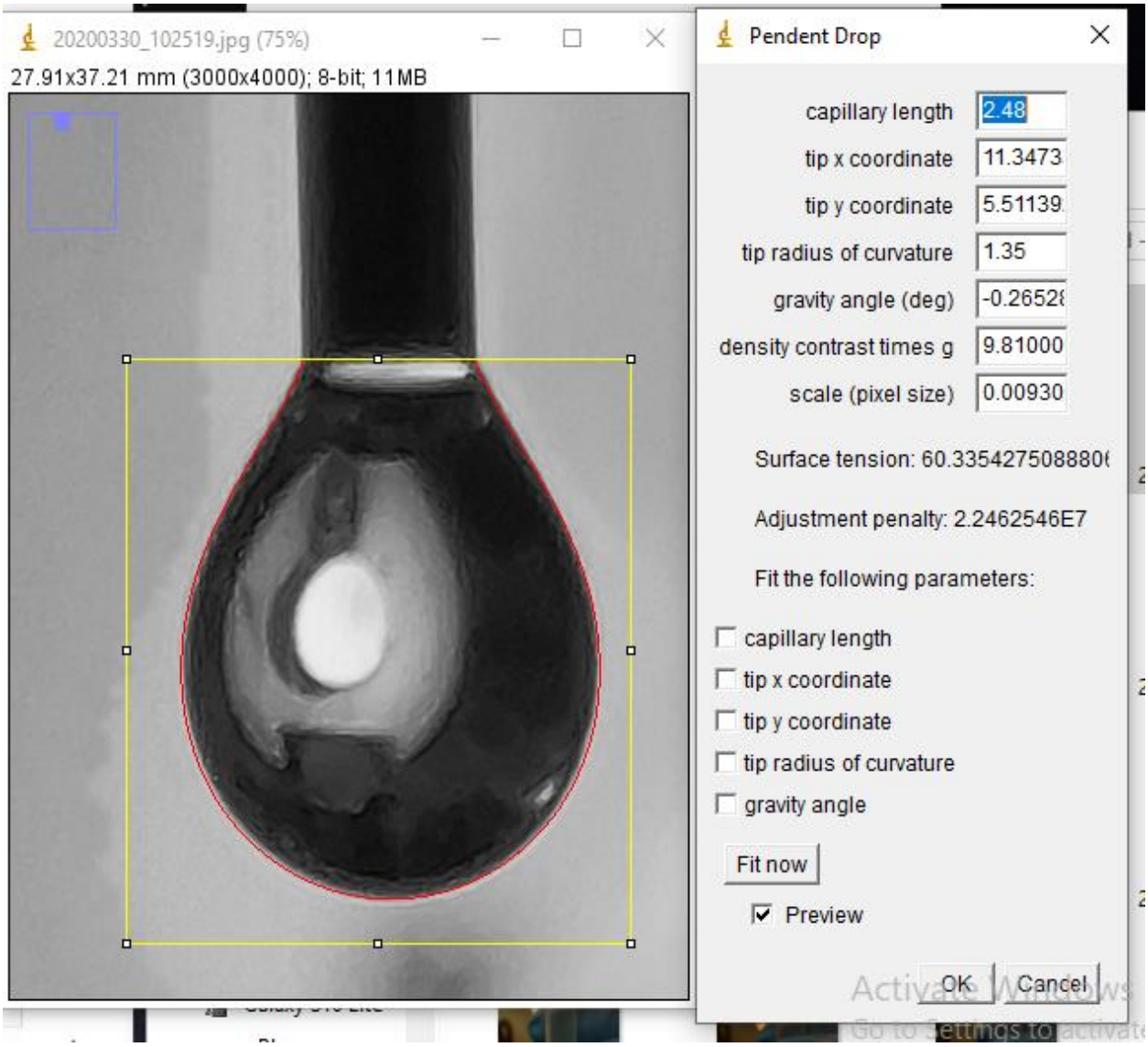
Supplementary Table S1. The clinical samples' IDs, Ct values, optical intensities (taken from Supplementary Figure S2), normalized intensities, last oral intake data (LOI), surface tension analysis, and protein concentration. Ct values are not available for negative samples. The optical intensities are the modes against the black background. The normalized intensities higher than 1.41 were deemed turbid (refer to Figure 4C), indicated by the orange-colored cells. Not all samples had LOI data provided. Surface tension was calculated using pendant drop analysis and not all samples had enough volume for this assay. Protein concentration was determined using Bradford assay analysis and not all samples had enough volume for this analysis.

ID	Ct	Mode of the optical intensities	Normalized intensities	Last Oral Intake (LOI)	Surface Tension Analysis (mN/mm)	Protein Concentration (mg/mL)
CVG00004	-	71	1.06	60	50.31	x
CVG00005	-	93	1.39	120	x	x
CVG00011	-	143	2.13	30	x	x
CVG00013	-	89	1.33	120	51.68	0.3109
CVG00014	-	85	1.27	60	x	x
CVG00015	-	82	1.22	120	45.42	0.1904
CVG00016	-	80	1.19	120	64.91	0.0871
CVG00017	-	108	1.61	120	x	x
CVG00018	-	97	1.45	120	x	0.0713
CVG00024	-	101	1.51	15	52.86	0.1608
CVG000104	-	80	1.19	120	x	-0.0269
CVG000105	-	91	1.36	20	x	x
CVG000106	-	122	1.82	10	49.53	0.2127
CVG000114	-	89	1.33	10	x	0.209
CVG00001	26	73	1.09	60	x	x
9900202	35	115	1.72	x	x	0.0206
9900208	29	110	1.64	x	51.05	0.4837

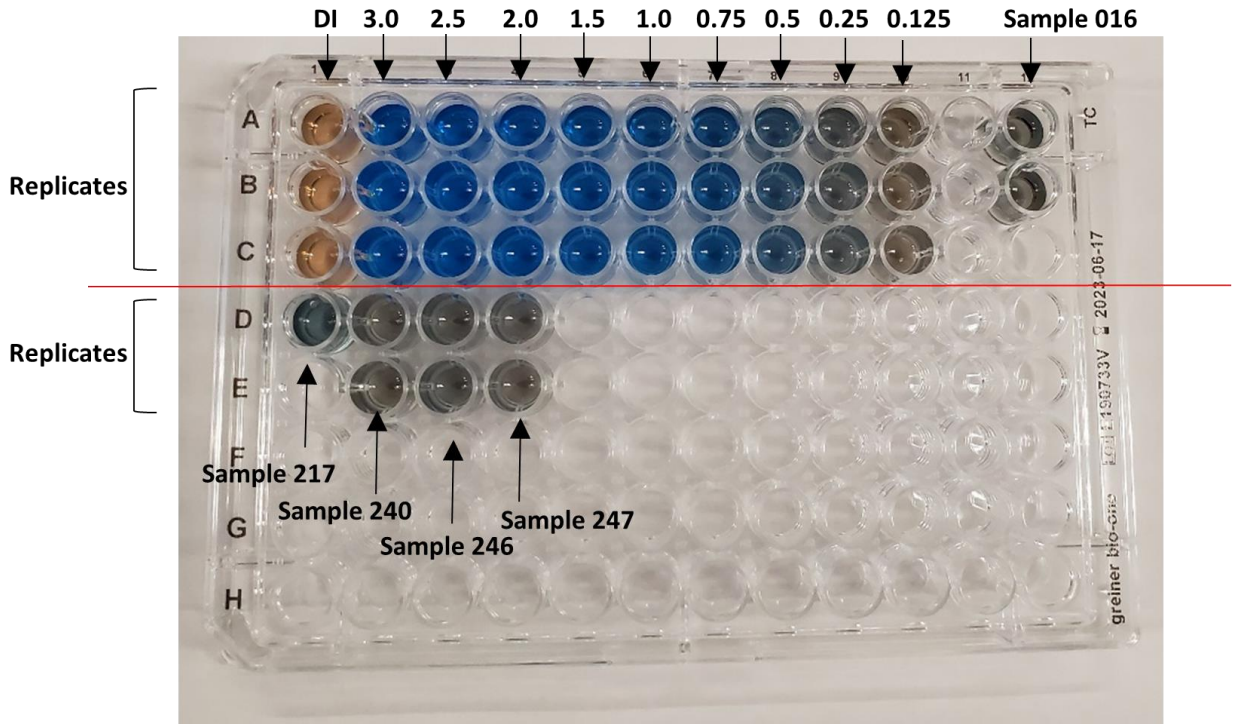
9900209	35	73	1.09	x	63.43	0.0444
9900217	31	93	1.39	x	52.92	0.4141
9900234	22	80	1.19	x	x	x
9900238	26	74	1.10	x	x	0.1939
9900239	31	139	2.07	10	45.67	0.4694
9900240	34	87	1.30	x	61.44	0.2572
9900244	33	130	1.94	x	45.19	0.304
9900246	36	80	1.19	x	53.09	0.2523
9900247	34	89	1.33	x	57.11	x
9900249	31	117	1.75	x	x	x
9900253	27	87	1.30	x	x	x
Empty tube		67	1.00	N/A	N/A	N/A



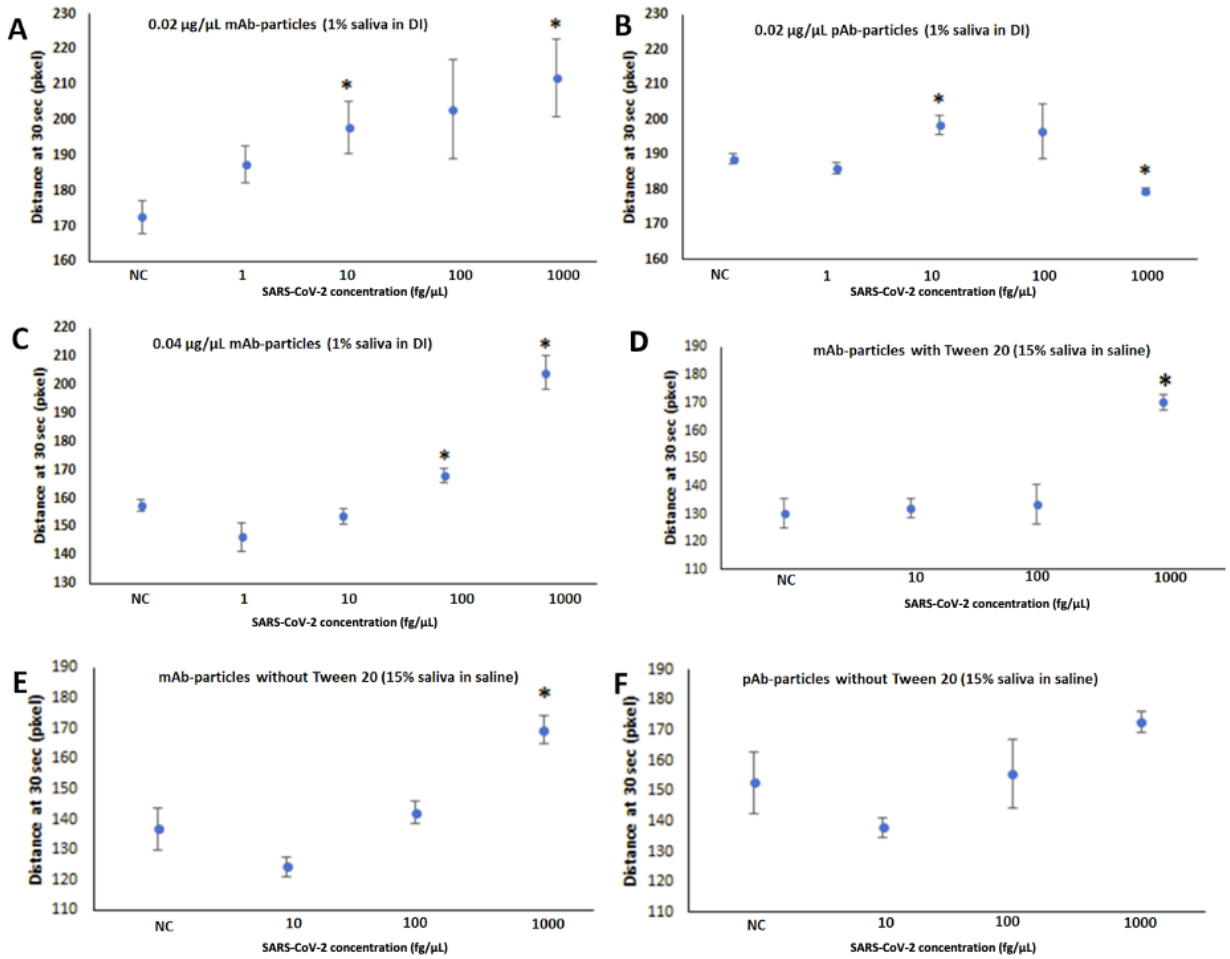
Supplementary Figure S2. Photographs of (A) positive and (B) negative clinical saline gargle samples, from left to right and top to bottom, organized in the same ID order as that shown in Supplementary Table S1. Red boxes indicate turbid samples, while blue boxes indicate relatively clear samples. (C) Empty tube used as a reference for normalization.



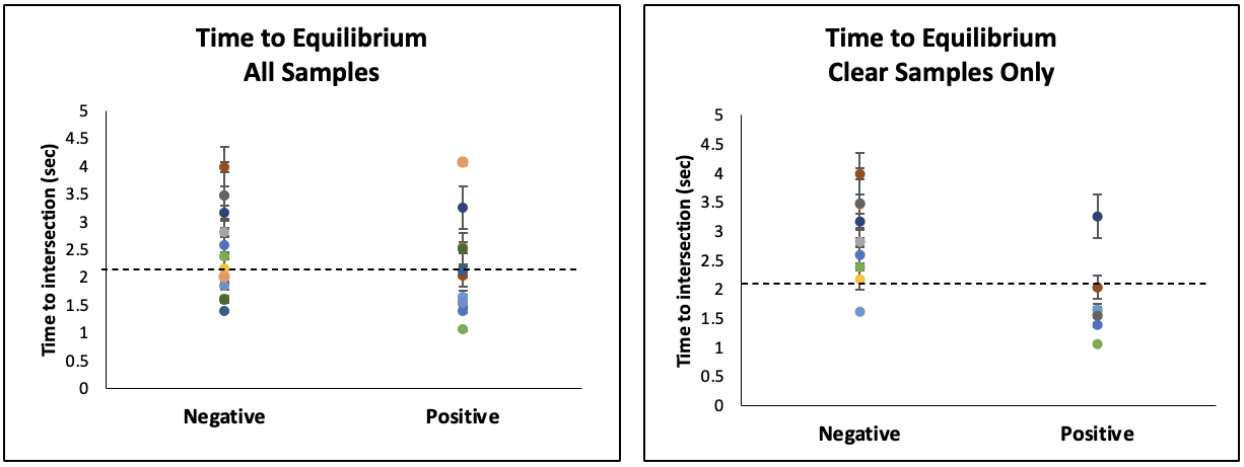
Supplementary Figure S3. Example image of a clinical sample suspended from the blunt needle tip and analyzed in ImageJ. The red outline around the droplet was adjusted to fit as perfectly as possible around the droplet, and then the surface tension was automatically calculated. Results from this method can be seen in Figure 4 and in Supplementary Table S1.



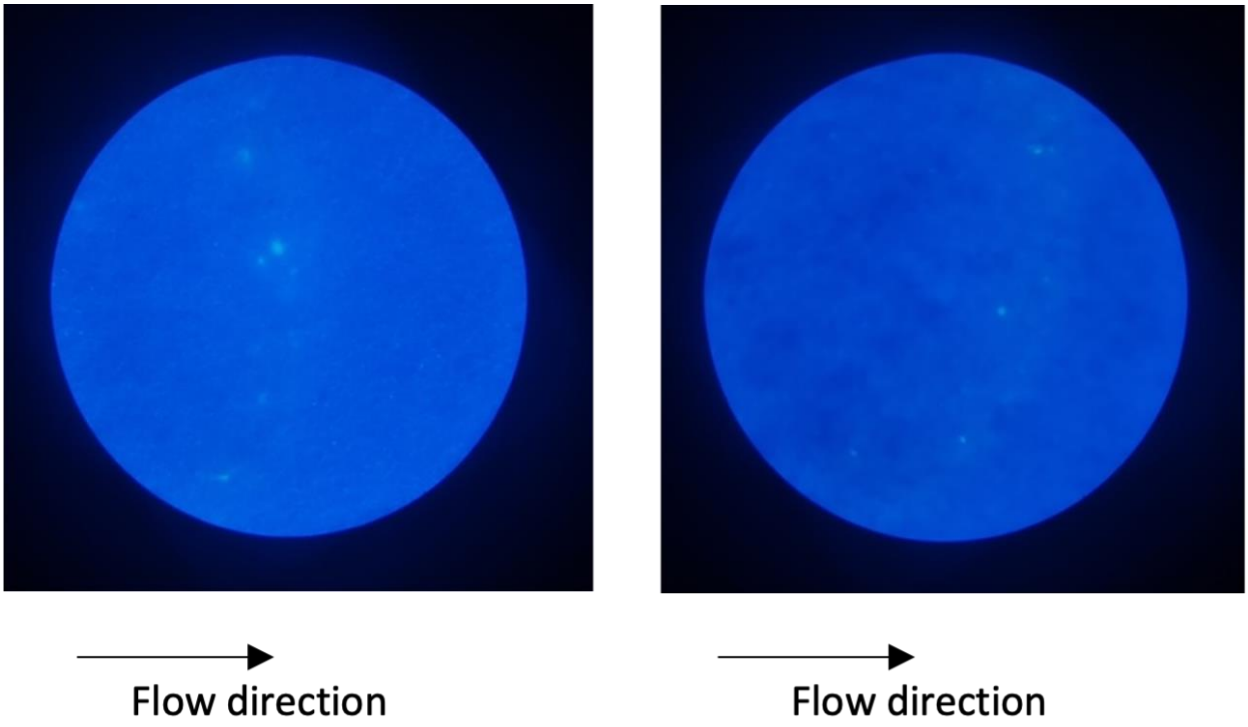
Supplementary Figure S4. Example Bradford assay microwell plate. Values are expressed in mg/mL BSA. Samples and standards become visibly blue with Bradford reagent for high protein concentrations. A new standard curve was generated each day of experiments (one per microwell plate). Due to low sample volume remaining after flow rate assays, some samples could only be tested 1-2 times (5µL sample per well).



Supplementary Figure S5. Assay optimization experiments with varying types and concentrations of SARS-CoV-2 antibodies. NC indicates negative control and * shows $p<0.05$ between sample and NC using one-tailed student's t-test with unequal variance. A general increasing trend can be observed for mAb-particles at 0.02 µg/µL with increased virus concentration, but the error bars also amplify (A). With pAb-particles at 0.02 µg/µL, however, a bell-shaped curve behavior is seen (B). Initially there is an increase in signal with virus concentration, reaching a maximum signal at 10 fg/µL before decreasing. The error bars are substantially smaller with the pAb-particles than with the mAb-particles. For both assays, the limit of detection (LOD) is 10 fg/µL or approximately 10 copies/µL. With mAb-particles at 0.04 µg/µL (C), the linear range shifted to higher concentrations, which is expected due to the higher number of antibodies available in this assay. The LOD is compromised to 100 fg/µL or approximately 100 copies/µL. The simulated saline gargle samples that are more similar to the clinical samples (15% saliva and 0.9% saline) show significantly compromised results, with the negative control samples showing slightly higher flow distances and compromised LODs (D-F). However, the addition of Tween 20 (0.5% w/v) significantly improved the results for mAb-particles (D: with Tween 20; E: without Tween 20). As shown in Figure 3 of the manuscript, the optimal conditions were determined to be 0.04 µg/µL pAb-particles with 0.5% w/v Tween 20.



Supplementary Figure S6. Time to constant velocity graphs of (left) all clinical samples and (right) just clear samples.



Supplementary Figure S7. Example photos showing more (left) and less (right) particle aggregation. Particles appear green under 460 nm (blue) excitation.

Supplementary Code S1. Python script to recognize the flow distance profiles from the smartphone video clips.

```
#!/usr/bin/python
'''
Analyze flow of liquid on wax bound paper-based microfluidic chip
Authors:
   - Patarajarin Akarapipad as Biomedical Engineering M.S. Student
     patarajarina@email.arizona.edu
   - Jacob Clay Baker as Biomedical Engineering undergraduate student
     jacobbaker@email.arizona.edu
Created in Yoon Biosensors Lab, University of Arizona, 2020-present

Note: save the code in [file name].py and video in the same folder, and to execute the code, enter
 "[file name].py [video name].mp4" in the terminal
'''
import sys
import cv2
import numpy as np
import math
import time
from skimage import data# For Otsu's thresholding
from skimage.filters import threshold_multiotsu###########
import matplotlib###### If we are gonna do more stuff with improving image processing, use m
atplotlib for image might be interesting. However, right now I think this part is working pretty w
ell
import matplotlib.pyplot as plt########
import pathlib
import pandas as pd
import xlsxwriter
import openpyxl
from openpyxl.utils import get_column_letter
import operator
from scipy import integrate
###for slope
from statistics import mean
from matplotlib import style

#
def flow_measure(bw_channel_crop, pix_chann_width, count_time):#function that finds the begi
nning of the channel, measures three flow lines until hitting the flow front, then averages flow di
stance and returns time and distance.
    global frame_num, pixel_conversion, act_chann_width, flow_front_ch
    check = 0   #used to decide whether to enter flow measurement
    white = 255
    black = 0
```

```
    avg_distance=-1
    #derive dimension of crop image
    crop_hight, _ = bw_channel_crop.shape
    zero_start = 0
    y_distance = 1
    x_distance = 1
    x_distance_center =pix_chann_width/2
    vals = bw_channel_crop[y_distance, x_distance]#starts looking at coordinate (1,1)
    while(vals == white): #moves y coordinate to top of channel marker
        y_distance = y_distance + 1
        vals = bw_channel_crop[y_distance,x_distance]
        if vals == black:
            check = check +1

    while (vals == black): #moves y coordinate back to white part in the middle of the channel
        vals = bw_channel_crop[y_distance, x_distance]
        y_distance = y_distance + 1
        if vals == white:
            check = check +1
    flow_start = y_distance #now that we have found the beginning of the channel, we can set the
flow start position

    ##########3 let's make new x based on the actual channel width not the random crop
     ######### Identify the channel width (find x_channel_center)

    find_x_center = bw_channel_crop[math.floor(y_distance/2),x_distance]
    while(find_x_center == white):
        x_distance = x_distance + 1
        find_x_center = bw_channel_crop[math.floor(y_distance/2),x_distance]
        if find_x_center == black:
            chann_width_pixel = act_chann_width*pixel_conversion
            x_distance_center = x_distance + chann_width_pixel/2
    #print(x_distance)
    x_distance = math.floor(x_distance_center)
    x_distance_2 = math.floor(x_distance_center + (0. * pixel_conversion)) #pixel column 0.2mm
 to the right of the middle
    x_distance_3 = math.floor(x_distance_center - (0.5 * pixel_conversion)) #pixel column 0.2m
m to the left of the middle


    #begin flow is the number of pixels in the negative y direction
    flowing, flowing2, flowing3 = flow_start, flow_start,flow_start
    vals = bw_channel_crop[flowing,x_distance]
    vals_2 = bw_channel_crop[flowing2,x_distance_2]
    vals_3 = bw_channel_crop[flowing3,x_distance_3]
    # initializing
```

```
    middle_pix, right_pix, left_pix = 0, 0, 0
    if (check == 2): #only triggers if the flow_start position has been found
        while(vals == black):
            flowing = flowing+1
            zero_start = zero_start+1
            vals = bw_channel_crop[flowing,x_distance]
            if (vals == white):
                count_time = count_time + 0.2 #tracks flow time
                middle_pix = zero_start #tracks flow distance
                break
        zero_start=0
        while(vals_2 == black and flowing2 < (crop_hight - 1)):
            flowing2 = flowing2 + 1
            vals_2 = bw_channel_crop[flowing2, x_distance_2]
            zero_start = zero_start+1
            if (vals_2 == white):
                right_pix = zero_start
                break
        zero_start=0
        while(vals_3 == black  and flowing3 < crop_hight - 1):
            flowing3 = flowing3+1
            vals_3 = bw_channel_crop[flowing3,x_distance_3]
            zero_start = zero_start+1
            if (vals_3 == white):
                left_pix = zero_start
                break #************
        if left_pix > 0 and right_pix > 0 and middle_pix > 0 : #************
            if left_pix > middle_pix and right_pix > middle_pix and flow_front_ch == 0:# See time a
nd distance at concavity change
                flow_front_ch = 1
                print('Concavity changes at (time, center distance in pix, anddis/time: )', count_time, m
iddle_pix, count_time/middle_pix)
            #avg_distance = (left_pix + middle_pix + right_pix) / 3 #************ #averages three
flow distance markers
            avg_distance = middle_pix
    if avg_distance > 0 : #only returns time and distance if any progress has been made
        return ( round(count_time,2), round(avg_distance,2))

def best_fit_slope_and_intercept(xs,ys):
    m = (((mean(xs)*mean(ys)) - mean(xs*ys)) /
         ((mean(xs)*mean(xs)) - mean(xs*xs)))

    b = mean(ys) - m*mean(xs)

    return m, b
```

```
def single_channel_data(channel_num, channel_position, count_time, flow_array):
    threshTune = 0 # change here (~+-5) if the threshold is still not good
    global imageFrame, pixel_conversion, stop_time, frame_num, act_chann_width, act_chann_le
n, act_top_chann
    #Define channel
    pix_chann_width = act_chann_width * pixel_conversion
    pix_chann_len = act_chann_len * pixel_conversion
    pix_top_chann = math.floor(act_top_chann * pixel_conversion)
    top_chann = math.floor(pix_chann_len + pix_top_chann)
    left_chann = math.floor(channel_position - pix_chann_width/2) # dimension is not very precis
e, so move to the left by 10
    right_chann = math.ceil(channel_position + pix_chann_width/2)
    #Cropping the channel and automatically derive threshold values
    channel_crop = imageFrame[pix_top_chann: top_chann, left_chann : right_chann]
    gray_channel_crop = cv2.cvtColor(channel_crop, cv2.COLOR_BGR2GRAY) #Turn the chan
nel to gray scale
    blur = cv2.GaussianBlur(gray_channel_crop,(9,9),cv2.BORDER_DEFAULT) #Gaussian filter
ing
    thresholds = threshold_multiotsu(blur) # figure out Otsu thresholding values automatically
    thresh_1 = thresholds[1] # use the higher value
    thresh_2 = thresh_1 + threshTune # in case it needs to be adjusted, default of threshTune is 0
    bw_channel_crop = cv2.threshold(blur, thresh_2, 255, cv2.THRESH_BINARY)[1]
    # Create the name of each cannel
    name_channel = "Channel_Number_"
    name_channel = name_channel + str(channel_num)
    cv2.imshow(name_channel, bw_channel_crop)
    # Collect data every 0.2 s
    if(count_time <= stop_time and frame_num % 6 == 0): # 0.2s
        flow = flow_measure(bw_channel_crop, pix_chann_width, count_time)
        if flow is not None:
            count_time = flow[0]
            flow_array.append(flow)
    return flow_array, count_time

# Display results and graph after the flow of each channel reaches the stop_time for considering i
f the results look make sense
# Will return distance and time of each flow (at the stop time)
def display_results(channel_num, count_time, flow_array, run_one_time):
    global stop_time# write_xl
    lines_intersection_x, lines_intersection_y =0,0
    DiffSlope, DisSlope1, DisSlope2 = 0,0,0
    plot_x_time, plot_y_dist = [], [] # Define temporary arrays for plotting
    deriv_data = []#initializing
    if count_time == stop_time and run_one_time == 1:
        run_one_time = 2
        for each_data_set in flow_array:
```

```
            plot_x_time.append(each_data_set[0])
            plot_y_dist.append(each_data_set[1])
        last_distance = plot_y_dist[-1]
        # Create the name of each cannel
        name_channel = "Channel_Number_"
        name_channel = name_channel + str(channel_num)
        print("\n", name_channel,": Time (s)\n", plot_x_time, "\n",name_channel, ": Distance\n", pl
ot_y_dist) #**********
        plt.plot(plot_x_time, plot_y_dist, 'bo')
        plt.xlabel('Time (s)')
        plt.ylabel('Distance (pix)')
        plt.suptitle(name_channel)
        plt.show()
        #write_xl = write_xl + 1 #Keep track and until it turns to 4, start writing in excel

        ##dev
        time_in_sec = []
        dist_each_sec = []

        i=0
        newx = []
        for i in range(0,len(plot_x_time)-1):
            if i % 5 == 0:
                time_in_sec.append(plot_x_time[i])
                dist_each_sec.append(plot_y_dist[i])
                newx.append((plot_x_time[i] + plot_x_time[i+5]) / 2)
        dydx = np.diff(dist_each_sec)/np.diff(time_in_sec)

        ########### Slope of the first 3 ponints of Diff1
        xs = np.array(time_in_sec[0:3], dtype=np.float64)
        ys = np.array(dydx[0:3], dtype=np.float64)
        DiffSlope, y_intercept = best_fit_slope_and_intercept(xs,ys)
        print('\nSlope is ', DiffSlope)
        print('Vel: ', ys)
        print('time: ', xs)
        ###Plot
        regression_line = [(DiffSlope*x)+y_intercept for x in xs]
        style.use('ggplot')
        plt.scatter(time_in_sec[:-1],dydx,color='#003F72')
        plt.plot(xs, regression_line)
        plt.show()

        #for excel
        deriv_excel_time = ['Time(s)']
        #deriv_excel_time.append(time_in_sec[:-1])
        deriv_excel_time=deriv_excel_time+time_in_sec[:-1]
```

```
deriv_excel_vel = ['Vel(pix/s)']
#deriv_excel_vel.append(dydx)
deriv_excel_vel=deriv_excel_vel+list(dydx)
deriv_data=[deriv_excel_time,deriv_excel_vel]
##

######## 2nd Diff

dy2d2x = np.diff(dydx)/np.diff(time_in_sec[:-1])
plt.plot(time_in_sec[:-2], dy2d2x, 'bo')
plt.xlabel('Time (s)')
plt.ylabel('Diff2 (pix/s2)')
name_channel_dif=name_channel+'dif'
plt.suptitle(name_channel_dif+'2')
plt.show()

########## Intersection baby style
dxdy10_15s = dydx[-5:] # vel at last 5 second
avgdxdy10_15s = sum(dxdy10_15s)/len(dxdy10_15s)
ChangeAtTime =0
for x in dydx:
    ChangeAtTime=ChangeAtTime+1
    if x >= avgdxdy10_15s + 3:
        break
print('Time is ',ChangeAtTime)
#Got the time, next look at distance data
element_number=0
for time_point in plot_x_time:
    if time_point <= ChangeAtTime:
        element_number =element_number+1
###############
element_number_shift = element_number
bestFit_distance1st = plot_y_dist[0:element_number_shift]
bestFit_time1st = plot_x_time[0:element_number_shift]
bestFit_distance2nd = plot_y_dist[element_number_shift:] # ignore 5 points
bestFit_time2nd = plot_x_time[element_number_shift:]
#let's plot them!
# get slopes and intersections
xs1 = np.array(bestFit_time1st, dtype=np.float64)
ys1 = np.array(bestFit_distance1st, dtype=np.float64)
xs2 = np.array(bestFit_time2nd, dtype=np.float64)
ys2 = np.array(bestFit_distance2nd, dtype=np.float64)
DisSlope1, y_intercept1 = best_fit_slope_and_intercept(xs1,ys1)
DisSlope2, y_intercept2 = best_fit_slope_and_intercept(xs2,ys2)
print('\nSlope are ', DisSlope1, DisSlope2)
print('Vel: ', ys1, ys2)
```

```
        print('time: ', xs1, xs2)
        ###Plot
        regression_line1 = [(DisSlope1*x)+y_intercept1 for x in plot_x_time[0:len(xs1)+8]]
        regression_line2 = [(DisSlope2*x)+y_intercept2 for x in plot_x_time]
        style.use('ggplot')
        plt.scatter(plot_x_time,plot_y_dist,color='#003F72')
        plt.plot(plot_x_time[0:len(xs1)+8], regression_line1)
        plt.plot(plot_x_time, regression_line2)
        plt.show()
        ###Alright, get the intersection
        lines_intersection_x = (y_intercept2-y_intercept1)/(DisSlope1-DisSlope2)
        lines_intersection_y = DisSlope2*lines_intersection_x +y_intercept2
        print('The intersection point is (s, pix) ', lines_intersection_x, lines_intersection_y/pixel_con
version)


    return deriv_data, run_one_time, lines_intersection_x, lines_intersection_y, DiffSlope, DisSlo
pe1, DisSlope2

def orientation_correction():
    global imageFrame
    global sq_A, sq_B, sq_C, old_sqA, old_sqB, old_sqC, angle_off_deg, r_x, r_y, r_w, r_h
   ##################################################################33
    # Set range for red color and define mask
    hsvFrame = cv2.cvtColor(imageFrame, cv2.COLOR_BGR2HSV)
    red_lower = np.array([2, 100, 110], np.uint8) #[136, 87, 111]----------2, 100, 110
    red_upper = np.array([30, 255, 255], np.uint8)
    red_mask = cv2.inRange(hsvFrame, red_lower, red_upper)
    kernal = np.ones((5, 5), "uint8")
    # For red color
    red_mask = cv2.dilate(red_mask, kernal)
    res_red = cv2.bitwise_and(imageFrame, imageFrame,
                              mask = red_mask)
    # Creating contour to track red color
    contours, hierarchy = cv2.findContours(red_mask, cv2.RETR_TREE, cv2.CHAIN_APPROX
_SIMPLE)

    i =1# counting squares
    for pic, contour in enumerate(contours):
        area = cv2.contourArea(contour)
        #print(area)
        if(area > 200 and area < 500):
        #if(area > 600 and area < 900):
            x, y, w, h = cv2.boundingRect(contour)
            imageFrame = cv2.rectangle(imageFrame, (x, y), (x + w, y + h), (0, 0, 255), 2)
            r_x = x
```

```
            r_y = y
            r_w = w
            r_h = h
            # label each

            if ((0.5<r_w/r_h < 2 or  0.5<r_h/r_w <2)): #and thisFrame ==checkCurrentFrame
                cX = r_x + r_w/2
                cY = r_y + r_h/2
                if i == 1:
                    if len(sq_A) != 0:
                        old_sqA = sq_A
                    sq_A = [cX, cY]
                    i = i+1
                elif i == 2:
                    if len(sq_B)  != 0:
                        old_sqB = sq_B
                    sq_B = [cX, cY]
                    i = i+1
                elif i == 3:
                    if  len(sq_C) != 0:
                        old_sqC = sq_C
                    sq_C = [cX, cY]
                    i = i+1
                    #thisFrame = thisFrame+1
                cv2.putText(imageFrame, "Check", (r_x, r_y), cv2.FONT_HERSHEY_SIMPLEX, 0.5
, (0, 0, 255))
    if i != 4:
        sq_A = old_sqA
        sq_B = old_sqB
        sq_C = old_sqC

    if  len(sq_A)  != 0 and len(sq_B)  != 0 and len(sq_C)  != 0 and sq_A != sq_B and sq_B!=sq_C
 and sq_C!=sq_A:
        lineAB = math.sqrt((sq_A[0]  - sq_B[0])**2 +(sq_A[1]  - sq_B[1])**2)
        lineBC = math.sqrt((sq_B[0]  - sq_C[0])**2 +(sq_B[1]  - sq_C[1])**2)
        lineAC = math.sqrt((sq_A[0]  - sq_C[0])**2 +(sq_A[1]  - sq_C[1])**2)
        DictLine = {'AB':lineAB, 'BC':lineBC, 'AC':lineAC}
        hypotenuse = max(DictLine.items(), key=operator.itemgetter(1))[0]
        DictLine.pop(hypotenuse)
        adjacent = max(DictLine.items(), key=operator.itemgetter(1))[0]
        opposite = min(DictLine.items(), key=operator.itemgetter(1))[0]

        for char1 in hypotenuse:
            for char2 in adjacent:
                if char1 == char2:
                    topleft = char2
```

```
        for char3 in opposite:
            for char4 in adjacent:
                if char3 == char4:
                    topright = char4

        if topleft == 'A':
            topleft = sq_A
        elif topleft == 'B':
            topleft = sq_B
        elif topleft == 'C':
            topleft = sq_C

        if topright == 'A' and topleft != 'A':
            topright = sq_A
        elif topright == 'B' and topleft != 'B':
            topright = sq_B
        elif topright == 'C' and topleft != 'C':
            topright = sq_C

        tanX = 0
        if topleft[0] != topright[0]:
            tanX = (topleft[1] - topright[1])/(topleft[0] - topright[0])
        angle_off_rad = math.atan(tanX)# return in radians, best case scenario is 0

        # get angle in degree
        row,col,ht = imageFrame.shape
        scale = 1
        old_angle = angle_off_deg # store previous data to produce less extra movement
        angle_off_deg = angle_off_rad*360/(2*math.pi)
        if abs(old_angle-angle_off_deg) < 0.5 or abs(old_angle-angle_off_deg) > 30:
            angle_off_deg =old_angle
        #print(angle_off_deg)

        matrix = cv2.getRotationMatrix2D((col/2,row/2), angle_off_deg, scale)
        imageFrame = cv2.warpAffine(imageFrame,matrix,(col,row))

def write_xl_fn(FILE_NAME,data_val1, data_val2, data_val3, data_val4, data_val5, data_val6,d
ata_val7,data_val8,data_deriv):
    global write_xl, text
    data_n_1 =  text
    data_n_2 = 'Deri_slope_3pts'
    data_n_3 = 'Slope1'
    data_n_4 = 'Slope2'
    data_n_5 = 'Intersection_time_point'
    data_n_6 = 'Intersection_dist_point'
```

```
data_n_7 = 'Time(s)'
data_n_8 = 'Dist(pix)'
plot_x_time,plot_y_dist =[],[]

for each_data_set in data_val8:
    plot_x_time.append(each_data_set[0])
    plot_y_dist.append(each_data_set[1])
## deriv append
data_derivX = data_deriv[0]
data_derivY = data_deriv[1]
plot_x_time = plot_x_time + data_derivX
plot_y_dist =plot_y_dist +data_derivY
######

file = pathlib.Path(FILE_NAME)
if not file.exists ():
    workbook = xlsxwriter.Workbook(FILE_NAME)
    worksheet = workbook.add_worksheet()
    workbook.close()
if file.exists ():
    book = openpyxl.load_workbook(FILE_NAME)
    sheet = book.active

    cell_num = 1
    column_let1 = get_column_letter(1)
    next_column = get_column_letter(2)
    cell_loc = column_let1 + str(1)
    cell_value = sheet[cell_loc]
    while cell_value.value != None: #move to the next column until found empty cell
        column_let1 = get_column_letter(cell_num)
        cell_loc = column_let1 + str(1)
        next_column = get_column_letter(cell_num+1)
        cell_value = sheet[cell_loc]
        cell_num = 1 + cell_num

    for i in range(1,8):
        if i != 7:
            nameit = 'data_n_' + str(i)
            input_data = 'data_val' + str(i)
            sheet[column_let1 + str(i)] = locals()[nameit]
            sheet[next_column + str(i)] = locals()[input_data]
        elif i == 7:
            nameit = 'data_n_' + str(i)
            sheet[column_let1 + str(i)] = locals()[nameit]
            nameit8 = data_n_8#'data_n_' + str(i)
            sheet[next_column + str(i)] = nameit8
```

```
            i = i+1
            curr_row_time = i
            curr_row_dist = i
            for time_dat in plot_x_time:
                sheet[column_let1 + str(curr_row_time)] = time_dat
                curr_row_time = curr_row_time+1
            for dist_dat in plot_y_dist:
                sheet[next_column + str(curr_row_dist)] = dist_dat
                curr_row_dist = curr_row_dist+1
        book.save(FILE_NAME)
    write_xl=write_xl+1


######
# Hopefully, we can find a way to store large data base and somehow derive the flow pattern mo
del and info
# https://www.youtube.com/watch?v=3fOXIbycAmc
# def flow_math_model()

def main():
    if len(sys.argv) < 2:
        video_capture = cv2.VideoCapture("12_8_20covidflowkk/20201208_121849.mp4") #****
**********
    else:
        video_capture = cv2.VideoCapture(sys.argv[1])
        video_name = sys.argv[1]

    #derive frame rate
    framespersecond= int(video_capture.get(cv2.CAP_PROP_FPS))
    print("The total number of frames in this video is ", framespersecond)# 30frames/sec

    first_frame_length = 1
    global text
    ###### For orieantation correction
    global sq_A, sq_B, sq_C, old_sqA, old_sqB, old_sqC, angle_off_deg, r_x, r_y, r_w, r_h, imag
eFrame
    sq_A, sq_B, sq_C = [], [], []
    old_sqA, old_sqB, old_sqC = [], [], []
    angle_off_deg =0# initialized
    r_x, r_y, r_w, r_h = 0,0,1,1
    #####

    actual_width = 34
    actual_height = 28 # might not need but keep it just in case
    actual_edge_distance = 8 #distance from edge of device to first channel
    actual_channel_distance = 6 #distance from one channel to the next
```

```
    global act_chann_width, act_chann_len, act_top_chann # actual distances based on the channe
l model
    act_chann_width, act_chann_len, act_top_chann = 5.4, 20, 4
    #mm to pixels converstion rate: 15.56 pixels/mm
    global imageFrame, frame_num, pixel_conversion, stop_time, write_xl, flow_front_ch
    flow_front_ch=0
    frame_num, write_xl = 0, 0
    stop_time = 15 # Specify time that data should be collected
    # initialize count_time for each channel
    count_time1, count_time2, count_time3, count_time4 = 0, 0, 0, 0
    # Define array for collecting flow in each channel
    flow_array1, flow_array2, flow_array3, flow_array4 = [(0,0)], [(0,0)], [(0,0)], [(0,0)]
    deriv_data1,deriv_data2,deriv_data3,deriv_data4 = [],[],[],[]
    #Define number of channel
    channel_num1, channel_num2, channel_num3, channel_num4 = 1,2,3,4
    # for displaying results only once when the flow reaches stop_time
    run_one_time1, run_one_time2,run_one_time3, run_one_time4 = 1,1,1,1


    while True:

        ret, imageFrame = video_capture.read()
        if not ret:#VDO exists or not
            break

        frame_num = frame_num + 1

        orientation_correction() # correct orientation based on 3 red squares

        if first_frame_length == 1:
            initial_im_w, initial_im_h, initial_im_color = imageFrame.shape
            first_frame_length = 2
            print("initial_im_dim",initial_im_w, initial_im_h)
            # Width and height of the vdo
            g_x = 0
            g_y = 0
            g_w = initial_im_h
            g_h = initial_im_w
        # Convert the imageFrame in BGR(RGB color space) to HSV(hue-saturation-
value) color space
        hsvFrame = cv2.cvtColor(imageFrame, cv2.COLOR_BGR2HSV)
        # Set range for green color and define mask
        green_lower = np.array([25, 70, 100], np.uint8) #25, 52, 72
        green_upper = np.array([102, 255, 255], np.uint8) #102,255,255
        green_mask = cv2.inRange(hsvFrame, green_lower, green_upper)
```

```
        kernal = np.ones((5, 5), "uint8")

        # For green color
        green_mask = cv2.dilate(green_mask, kernal)
        # Creating contour to track green color
        contours, hierarchy = cv2.findContours(green_mask,
                                               cv2.RETR_TREE,
                                               cv2.CHAIN_APPROX_SIMPLE)

        for pic, contour in enumerate(contours):
            area = cv2.contourArea(contour)
            #print(area)
            #if(area > 150000 and area < 199000):
            if(area > 195000 and area < 229000): #***************  #defines the border of the chip i
f the green rectangle is within a certain area boundary.
            #if(area > 440000 and area < 500000):
                x, y, w, h = cv2.boundingRect(contour)
                #*******************
                g_x = x
                g_y = y
                g_w = w+x
                g_h = h+y

                imageFrame = cv2.rectangle(imageFrame, (x, y),
                                           (x + w, y + h),
                                           (0, 255, 0), 2)

                cv2.putText(imageFrame, "Green Colour", (x, y),
                            cv2.FONT_HERSHEY_SIMPLEX,
                            1.0, (0, 255, 0))

        imageFrame = imageFrame[g_y : g_h, g_x : g_w ] #crops the rectangleq
        pixel_conversion = ((g_w - g_x) / actual_width)
        pixel_edge_distance = pixel_conversion * actual_edge_distance
        pixel_channel_distance = pixel_conversion * actual_channel_distance
        # pixel_width = (g_w - g_x) pixel_height = (g_h - g_y)
        channel1_position = pixel_edge_distance +3 #**************
        channel2_position = pixel_edge_distance + pixel_channel_distance - 1  #**************
        channel3_position = pixel_edge_distance + (pixel_channel_distance * 2) - 3 #************
**
        channel4_position = pixel_edge_distance + (pixel_channel_distance * 3) - 5  #***********
***

        # This function will provide distance and current time of the flow for each frame
        flow_array1, count_time1 = single_channel_data( channel_num1, channel1_position, count
_time1, flow_array1)
```

```
    flow_array2, count_time2 = single_channel_data( channel_num2, channel2_position, count
_time2, flow_array2)
    flow_array3, count_time3 = single_channel_data( channel_num3, channel3_position, count
_time3, flow_array3)
    flow_array4, count_time4 = single_channel_data( channel_num4, channel4_position, count
_time4, flow_array4)

    # Data set can be access here, trigger after each channel meets it stop_time
    # Derive the final distance and displaying data and graph
    if run_one_time1 == 1:
        deriv_data1, run_one_time1, time_intersect1, dist_intersect1,DiffSlope1, DisSlope11, Di
sSlope21 = display_results(channel_num1, count_time1, flow_array1, run_one_time1)
        if run_one_time1 !=1:
            write_xl =7
    if run_one_time2 == 1:
        deriv_data2, run_one_time2, time_intersect2, dist_intersect2,DiffSlope2, DisSlope12, Di
sSlope22 = display_results(channel_num2, count_time2, flow_array2, run_one_time2)
        if run_one_time2 !=1:
            write_xl =5
    if run_one_time3 == 1:
        deriv_data3, run_one_time3, time_intersect3, dist_intersect3,DiffSlope3, DisSlope13, Di
sSlope23 = display_results(channel_num3, count_time3, flow_array3, run_one_time3)
        if run_one_time3 !=1:
            write_xl =3
    if run_one_time4 == 1:
        deriv_data4, run_one_time4, time_intersect4, dist_intersect4,DiffSlope4, DisSlope14, Di
sSlope24 = display_results(channel_num4, count_time4, flow_array4, run_one_time4)
        if run_one_time4 !=1:
            write_xl=1
    # Print the whole chip
    cv2.imshow("Chip found", imageFrame)

    ######name the file
    FILE_NAME = "6-8_Turbidity.xlsx"
    if write_xl == 1:
        text = 'First result'
        write_xl_fn(FILE_NAME,video_name, DiffSlope4, DisSlope14, DisSlope24, time_inters
ect4, dist_intersect4/pixel_conversion,count_time4,flow_array4,deriv_data4) #dist_intersect4/pix
el_conversion
    if write_xl ==3:
        text = 'Second result'
        write_xl_fn(FILE_NAME,video_name, DiffSlope3, DisSlope13, DisSlope23, time_inters
ect3, dist_intersect3/pixel_conversion,count_time3,flow_array3,deriv_data3)
    if write_xl ==5:
        text = 'Tird result'
        write_xl_fn(FILE_NAME,video_name, DiffSlope2, DisSlope12, DisSlope22, time_inters
```

```
ect2, dist_intersect2/pixel_conversion,count_time2,flow_array2,deriv_data2)
        if write_xl == 7:
            text = 'Forth result'
            write_xl_fn(FILE_NAME,video_name, DiffSlope1, DisSlope11, DisSlope21, time_inters
ect1, dist_intersect1/pixel_conversion,count_time1,flow_array1,deriv_data1)

        ######## Jacob, put Excel stuff here


        if cv2.waitKey(10) & 0xFF == ord('q'):
            cap.release()
            #cv2.destroyAllWindows()
            break

    video_capture.release()
    # cv2.destroyAllWindows()

if __name__ == "__main__":
    # execute only if run as a script
    main()
```