CSE 410 Report

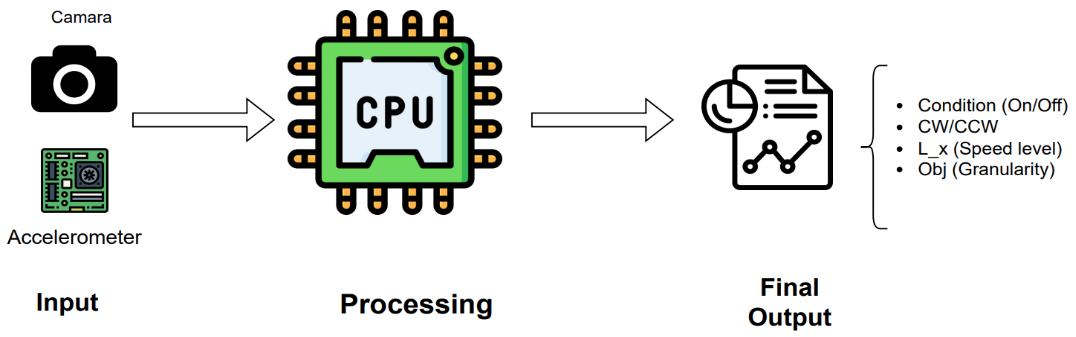
“Digital Twin”: Signal Processing

1. Project Introduction

In this project, our team's goal is to process signals received from the blender. The blender has two outputs, which we will receive as inputs. The first is camera data and the second is accelerometer data. We will then process and interpret the input data to obtain a result, which will be used in the real-time monitor for the blender. Doing this will allow the correlation of blender states and operation to end product states.

We aim to be able to monitor the blender's operational status (on or off), the blender's direction of rotation, determine the granularity of objects in the blender, and determine the speed of the blender. We then construct software to display the resultant data in a graphical user interface for the end user. This information will be used in the digital twin to allow for easier interpretation of the raw data coming from the physical device, and allow the digital twin to accurately reflect the real-time happenings of the blender.

2. Task Overview

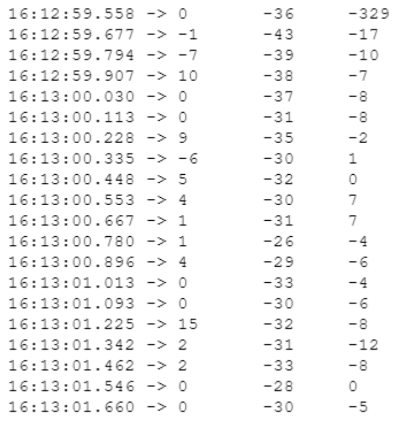


3. List of subtasks

Sub-task 1: Check the condition of the blender
Sub-task 2: Check the twist direction of blender
Sub-goal 3: Check Granularity of blender contents
Sub-task 4: Speed Checking
Sub-task 5: Design the GUI (user interface)

4. Sub-task 1

- Aim: Check the condition of the blender, the output will be On/Off
- Method:
  Firstly, this is the dummy data looks like, every 0.1 seconds a new list will append to this text file:



```
16:12:59.558 -> 0       -36     -329
16:12:59.677 -> -1      -43     -17
16:12:59.794 -> -7      -39     -10
16:12:59.907 -> 10      -38     -7
16:13:00.030 -> 0       -37     -8
16:13:00.113 -> 0       -31     -8
16:13:00.228 -> 9       -35     -2
16:13:00.335 -> -6      -30     1
16:13:00.448 -> 5       -32     0
16:13:00.553 -> 4       -30     7
16:13:00.667 -> 1       -31     7
16:13:00.780 -> 1       -26     -4
16:13:00.896 -> 4       -29     -6
16:13:01.013 -> 0       -33     -4
16:13:01.093 -> 0       -30     -6
16:13:01.225 -> 15      -32     -8
16:13:01.342 -> 2       -31     -12
16:13:01.462 -> 2       -33     -8
16:13:01.546 -> 0       -28     0
16:13:01.660 -> 0       -30     -5
```

Then, use codes to process the dummy data. Since every 0.1 seconds the sensor will send one list of information to our system. So once our system receives a new information list (ex. “`16:12:59.558 -> 0  -36   -329`”) from the sensor, the system will execute the following code:

```
import re
lst = "16:12:59.558 -> 0    -36 -329"
input_list = re.findall(r'-?\d+', lst)
print(input_list)
'''
Output: ['16', '12', '59', '558', '0', '-36', '-329']
'''
```

Then we already found out the time information and the coordinate information. Since the information list will be updated in every 0.1 seconds, so we don’t want to take care of the time right now, thus we just need to grab the last three values in the final output as the coordinate ([x,y,z]). Then send the coordinate (in [“0”,”-36”,”-329”] form) into the input of the following function.

The following code will process the input coordinate (in [“0”,”-36”,”-329”] form) every 0.1 seconds:

```
while True:
        original_position = [0,-30,-6] # Set the original position
        previous_postion = [0,-30,-6] # Set the previous position
        counter = [] # List for storing the condition of the blender in 
every 0.1 second
        for _ in range(10):
            lst = input('Enter [a,b,c]:') # Here is the input
            input_list = re.findall(r'-?\d+', lst)
            if abs(int(input_list[0])-
int(previous_postion[0]))<=20 and abs(int(original_position[2])-
int(input_list[2])<=20): # This is the pattern found from dummy data, but
this section can be changed depends on the real situation.
                previous_postion = input_list # Change the
previous position data after processing
                counter.append("Off") # This situation is off
            else:
                previous_postion = input_list
                counter.append("On")

# Note: The original position is the location when the blender just started; the
previous position is the location from the latest previous list
```

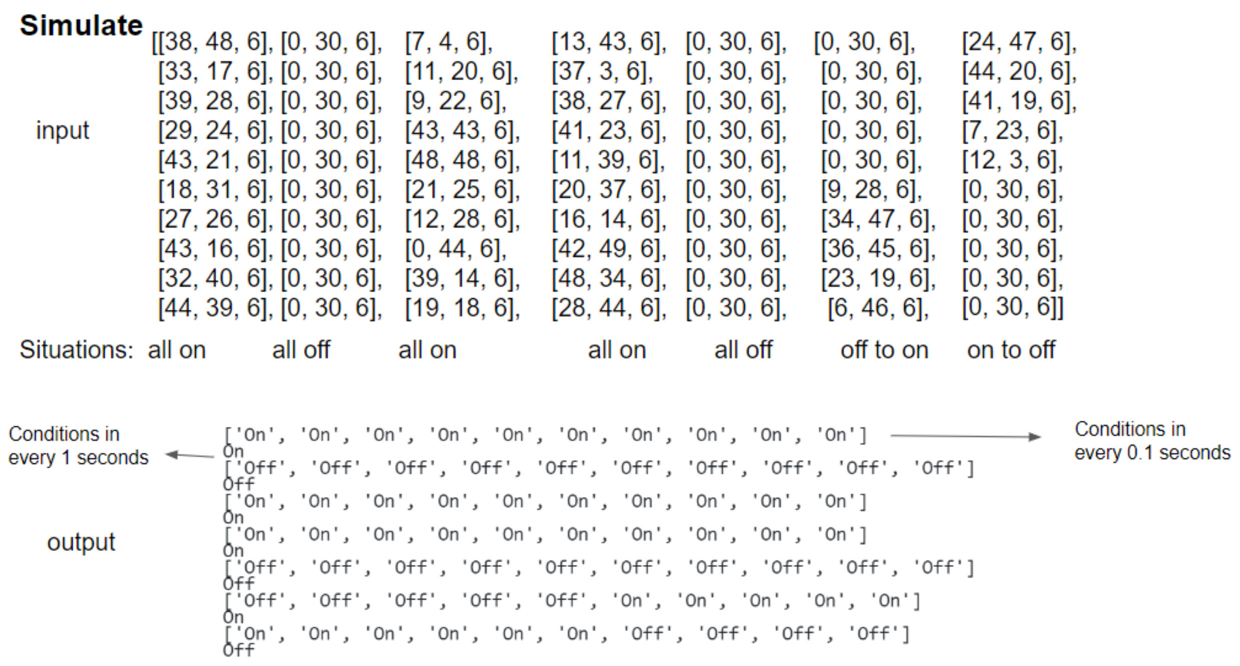
Basically, We want to compare the previous position with the current input position to check if the blender is working or not in 0.1 seconds duration. The original position is the location where the blender just started(The original position is making sure that the current input location has a piece of position information to compare with), and the previous position is the location that just finished processing. The checking processing mentioned in the code here is based on the dummy data, if necessary, users can change the code based on the pattern happens in the real case.

But the frequency 0.1 seconds per check is too high, we don't want the checking frequency to be too much and ignore the accuracy of the output. So this process use the `counter` list to store 10 elements (take about 1 second until full) then apply the following code to generate the condition result by analyze the feature from 10 elements of them:

```
result = counter
        # Condition = 'Off'
        if len(result) - 1 < result.count("On") or result.count("On") <
len(result) : # all On
            print('On')
        elif result.count('Off') <= 2: # accident move case during Off
            previous_postion = "The latest position"
            print('Off')
        elif result[0] == 'On' & result[-1] == 'Off': # from On to Off
            print('Off')
        elif result[0] == 'Off' & result[-1] == 'On': # from On to Off
            print("On")
        else:
            print("Off")
        counter = []
```

- Result (thoughts, insights, experience to share)
  These codes are able to analyze the condition (On/Off) of the blender based on the real-time data from the sensor every 0.1 seconds, and the output the real-time condition every 1 second. So this implements real-time monitoring for the blender.
  Thoughts: If there are more cases didn't cover in the current system, the developer can add more feature analysis functions in the original code. (Mentioned in the code block by comments)

  The picture below shows the result of these scripts by simulating real-life cases:



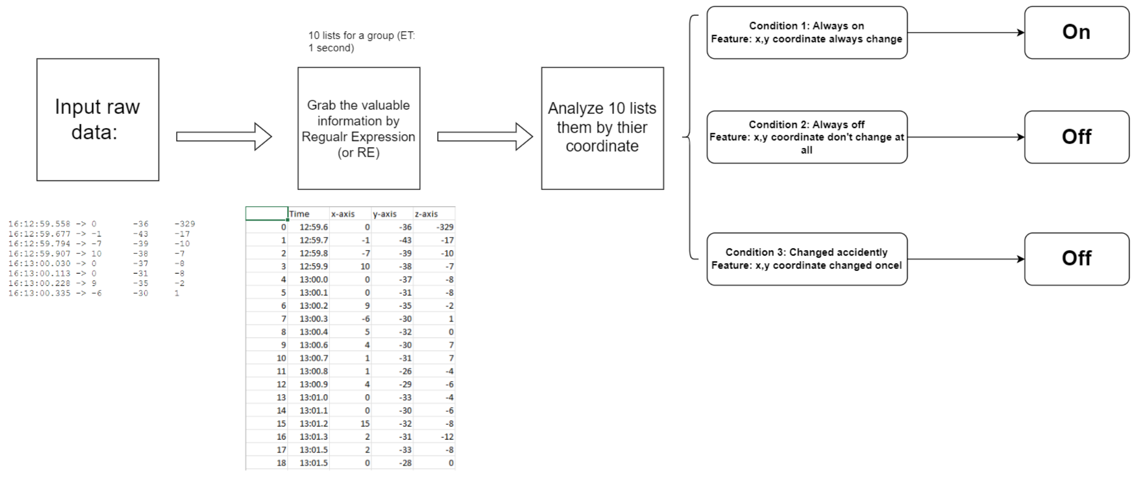
Simulate

```
input   [[38, 48, 6], [0, 30, 6], [7, 4, 6],   [13, 43, 6], [0, 30, 6], [0, 30, 6],  [24, 47, 6],
         [33, 17, 6], [0, 30, 6], [11, 20, 6], [37, 3, 6],  [0, 30, 6], [0, 30, 6],  [44, 20, 6],
         [39, 28, 6], [0, 30, 6], [9, 22, 6],  [38, 27, 6], [0, 30, 6], [0, 30, 6],  [41, 19, 6],
         [29, 24, 6], [0, 30, 6], [43, 43, 6], [41, 23, 6], [0, 30, 6], [0, 30, 6],  [7, 23, 6],
         [43, 21, 6], [0, 30, 6], [48, 48, 6], [11, 39, 6], [0, 30, 6], [0, 30, 6],  [12, 3, 6],
         [18, 31, 6], [0, 30, 6], [21, 25, 6], [20, 37, 6], [0, 30, 6], [9, 28, 6],  [0, 30, 6],
         [27, 26, 6], [0, 30, 6], [12, 28, 6], [16, 14, 6], [0, 30, 6], [34, 47, 6], [0, 30, 6],
         [43, 16, 6], [0, 30, 6], [0, 44, 6],  [42, 49, 6], [0, 30, 6], [36, 45, 6], [0, 30, 6],
         [32, 40, 6], [0, 30, 6], [39, 14, 6], [48, 34, 6], [0, 30, 6], [23, 19, 6], [0, 30, 6],
         [44, 39, 6], [0, 30, 6], [19, 18, 6], [28, 44, 6], [0, 30, 6], [6, 46, 6],  [0, 30, 6]]
Situations: all on    all off      all on       all on       all off     off to on    on to off
```

Conditions in every 1 seconds
Conditions in every 0.1 seconds
output

```
['On', 'On', 'On', 'On', 'On', 'On', 'On', 'On', 'On', 'On']
On
['Off', 'Off', 'Off', 'Off', 'Off', 'Off', 'Off', 'Off', 'Off', 'Off']
Off
['On', 'On', 'On', 'On', 'On', 'On', 'On', 'On', 'On', 'On']
On
['On', 'On', 'On', 'On', 'On', 'On', 'On', 'On', 'On', 'On']
On
['Off', 'Off', 'Off', 'Off', 'Off', 'Off', 'Off', 'Off', 'Off', 'Off']
Off
['Off', 'Off', 'Off', 'Off', 'Off', 'On', 'On', 'On', 'On', 'On']
On
['On', 'On', 'On', 'On', 'On', 'On', 'Off', 'Off', 'Off', 'Off']
Off
```

- Summary and Insight
  Flowchart:


Input raw data:
10 lists for a group (ET: 1 second)
Grab the valuable information by Regualr Expression (or RE)
Analyze 10 lists them by thier coordinate
Condition 1: Always on
Feature: x,y coordinate always change
On
Condition 2: Always off
Feature: x,y coordinate don't change at all
Off
Condition 3: Changed accidently
Feature: x,y coordinate changed oncel
Off

```
16:12:59.558 -> 0       -36     -329
16:12:59.677 -> -1      -43     -17
16:12:59.794 -> -7      -39     -10
16:12:59.907 -> 10      -38     -7
16:13:00.030 -> 0       -37     -8
16:13:00.113 -> 0       -31     -8
16:13:00.228 -> 9       -35     -2
16:13:00.335 -> -6      -30     1
```

	Time	x-axis	y-axis	z-axis
0	12:59.6	0	-36	-329
1	12:59.7	-1	-43	-17
2	12:59.8	-7	-39	-10
3	12:59.9	10	-38	-7
4	13:00.0	0	-37	-8
5	13:00.1	0	-31	-8
6	13:00.2	9	-35	-2
7	13:00.3	-6	-30	1
8	13:00.4	5	-32	0
9	13:00.6	4	-30	7
10	13:00.7	1	-31	7
11	13:00.8	1	-26	-4
12	13:00.9	4	-29	-6
13	13:01.0	0	-33	-4
14	13:01.1	0	-30	-6
15	13:01.2	15	-32	-8
16	13:01.3	2	-31	-12
17	13:01.5	2	-33	-8
18	13:01.5	0	-28	0

5. Sub-task 2

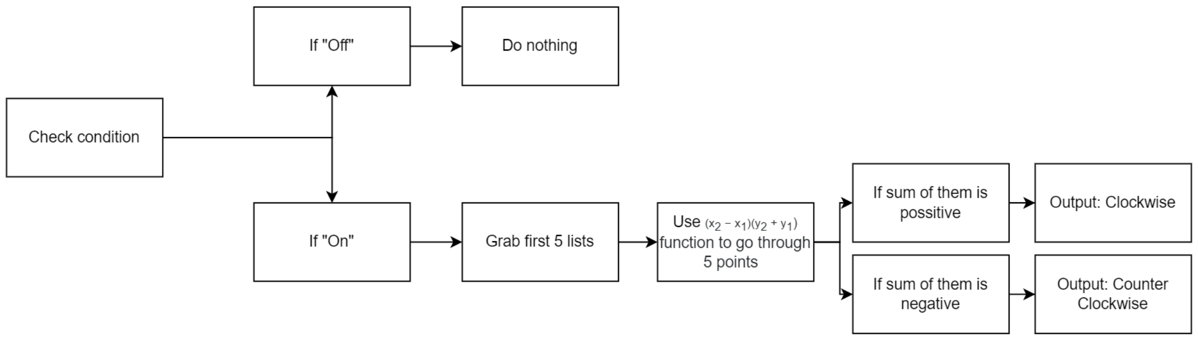
- Aim: Detect the twist direction of the blender
- Method:
    a. When the blender is on, grab the first 5 coordinates of point
    b. Use the function (x2 − x1)(y2 + y1) to go through these 5 points
    c. If the sum of all functions is positive, this means the direction is clockwise, otherwise, c. clockwise

Here is the code for implementing these methods:

```
def cw_ccw_check():
    store = [] #ex. [[1,2,3],[0,2,3],[7,1,3]]
    result = 0 # result will add all calculated value up
    cool = []
    while len(store)<5:
        cool = input("input").replace("[",'').replace("]",'').split(',') #
formatting the input
        store.append(cool) # store the input value into the store list
    for i in range(len(store)-1): # Go through all of elements in the store
list
        a=(int(store[i+1][0]) - int(store[i][0])) * (int(store[i+1][1]) +
int(store[i][1])) # apply the function
        result = a+ result # Add up all values
        print(a)
    if result<0: # If the sum of all values <0, this means the blender is
twisting in CCW
        print("Blender is working in CCW")
    else:# If the sum of all values >0, this means the blender is twisting
```

```
in CCW
        print("Blender is working in CW")
```

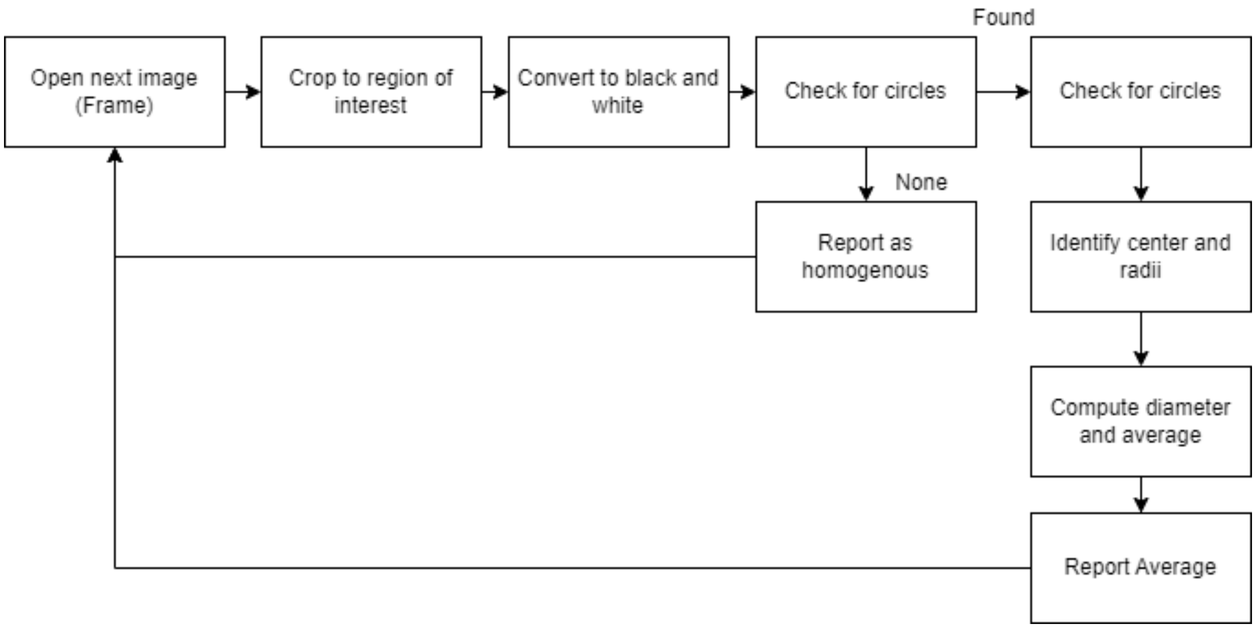
- Result (thoughts, insights, experience to share)
  These scripts are able to detect if the blender is twisting Counter Clockwise or Clockwise.
- Summary and Insight:
  Flowchart:


Check condition
If "Off"
Do nothing
If "On"
Grab first 5 lists
Use $(x_2 - x_1)(y_2 + y_1)$ function to go through 5 points
If sum of them is possitive
Output: Clockwise
If sum of them is negative
Output: Counter Clockwise

6. Sub-task 3

- Aim: This task aimed to check the granularity of the contents in the blender. Granularity was defined as the area of the particles in the blender, in mm^2.

- Method: The python package openCV was used to implement the computer vision portion. The code begins by reading the image. For video streams, the video would be separated into its constituent frames and analyzed one frame at a time. The image is then converted to grayscale for analysis.


Open next image (Frame)
Crop to region of interest
Convert to black and white
Check for circles
Found
Check for circles
None
Report as homogenous
Identify center and radii
Compute diameter and average
Report Average

```
img = cv2.imread('example.jpg')
gray = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
```

We then setup the parameters for estimated radii and distance.

```
minDist = 100
param1 = 30
param2 = 5
minRadius = 40
maxRadius = 100
```

The HoughCircles method was used to implement the actual circle detection.

```
circles = cv2.HoughCircles(blurred, cv2.HOUGH_GRADIENT, 1, minDist, param1=param1, param2=param2, minRadius=minRadius, maxRadius=maxRadius)
```

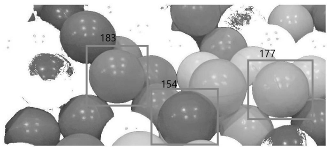
Hough circles were chosen after comparing a few different methods. This method returns a list of vectors containing the detected circles. We then iterate through the list of returned circles and display a bounding box around them.

```
for c in circles:
    x, y, widthPixel, heightPixel = cv2.boundingRect(c)
    realWidth = widthPixel / WIDTH_REF * 20.0
    realHeight = heightPixel / HEIGHT_REF * 20.0
    cv2.rectangle(res,(x,y),(x+widthPixel,y+heightPixel), (255, 0, 0), 2)
    cv2.putText(res, "{0:.1f}".format(realWidth) +" x " + "{0:.1f}".format(realHeight) + " mm
    print(realWidth, " x ", realHeight)
```

Height is determined by using the pixel count of the camera and the distance from the object. It is used as a constant with the pixel information returned by the

HoughCircles method. The radii in the image are then totaled and averaged to determine the granularity.

- Result (thoughts, insights, experience to share)



The above images show the result of a trial. A ball pit was used as the image. A few assumptions were made in determining the method:

1. The samples are spherical and break down into smaller spheres before losing circularity and being homogenous
2. The camera captures a profile that reveals circles rather than spheres
3. The camera and blender position would be stationary, allowing size to be determined.

If assumption 4 is not possible, the program would instead determine granularity as pixel area and return the result as a dimensionless number to be referenced against itself. The program was able to detect some circles and measure their diameter in pixels. More parameter tuning would increase the amount of detectable circles. Further, applying different blurs to reduce may have helped.

Outside of the HoughCircles method, two others were tried:

1. FindContours: This method located curves along boundaries. This method detected the edges of the image (a container). When cropping was applied to fix this, the method struggled with identifying overlapping balls as a single object.
2. SimpleBlobDetector: This method detects blobs in an image. By specifying parameters (area, circularity, inertia, and convexity) it can be used to accurately determine circles from other shapes. It worked on black and white images, but struggled when images with color were made binary. Adjusting the parameters allowed either over or underestimation of the amount of circles repeatedly.

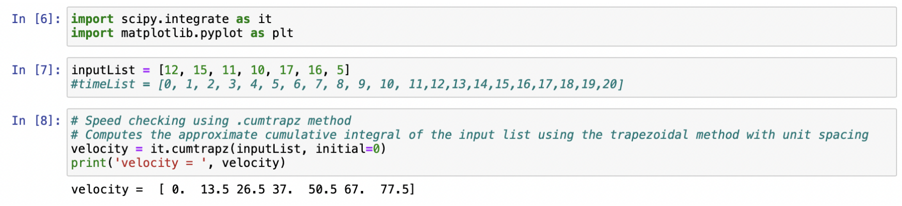
- Summary and Insight

The end result was a program able to crop an image and reliably detect and measure circles within the region. Finding more efficient ways to do this would increase the utility of the program as image analysis can be a compute-heavy task. Further, tuning the parameters used to specific conditions (lighting, distance, samples) would improve the accuracy of the program. Advanced methods, such as AI, can be used to learn from the detection this program does and continually improve the results.

7. Sub-task 4

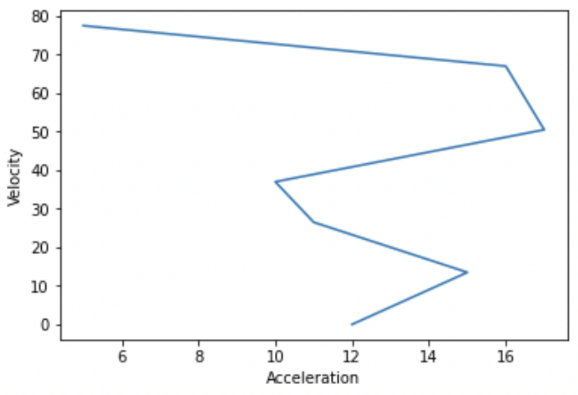
a. Aim: This task aimed to get the velocity of the blender through the use of accelerometer data.

b. Method: We first find the list of acceleration through the accelerometer. We take a list of this accelerometer data. We then use the .cumtrapz method. This method computes the cumulative integral of a function over a set of data points. We used the data we got from the accelerometer and took the integral of it to get an estimate of the velocity of the blender.



```
In [6]: import scipy.integrate as it
        import matplotlib.pyplot as plt

In [7]: inputList = [12, 15, 11, 10, 17, 16, 5]
        #timeList = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11,12,13,14,15,16,17,18,19,20]

In [8]: # Speed checking using .cumtrapz method
        # Computes the approximate cumulative integral of the input list using the trapezoidal method with unit spacing
        velocity = it.cumtrapz(inputList, initial=0)
        print('velocity = ', velocity)

        velocity =  [ 0.   13.5 26.5 37.   50.5 67.   77.5]
```

c. Results: The result of the method that we used is an array of values representing the cumulative integral of a function at each point in the dataset. These values represent the velocity from the acceleration.



d. Summary and Insight: Finished the code for speed checking. We used a very different approach to get the velocity compared to how I thought we would get this data. Working on a project and trying to figure out how to extract data out of raw data and making it into useful information seems really interesting to me. At first I wasn't sure how we would get the velocity, but basic physics knowledge came into play and I was able to find the velocity taking the integral of the acceleration.

8. Sub-task 5

- Aim: Implement a real-time monitor for monitoring the blender machine, with Power State, Direction, Speed, Granularity, Real-time Camera and Real-time Sensor features.
- Method:
    a. Firstly, design the GUI and divide them into six sections, then I chose to use the Tkinter package to implement these designs. The sample code for implementing a section is down below: (The full code is on my GitHub page under “GUI design” file)

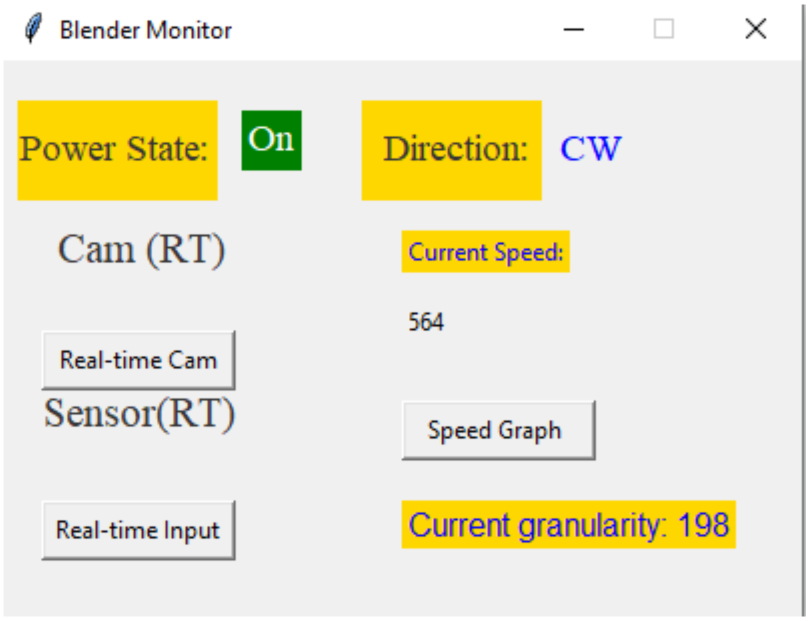
```
GLabel_402=tk.Label(root,bg='gold')
ft = tkFont.Font(family='Times',size=14)
GLabel_402["font"] = ft
GLabel_402["fg"] = "#333333"
GLabel_402["justify"] = "center"
GLabel_402["text"] = "Power State: "
GLabel_402.place(x=8,y=20,width=100,height=50)
```

    b. For Power State, Direction, Speed and Granularity: Call the functions designed before. Sample code for calling the functions designed before:

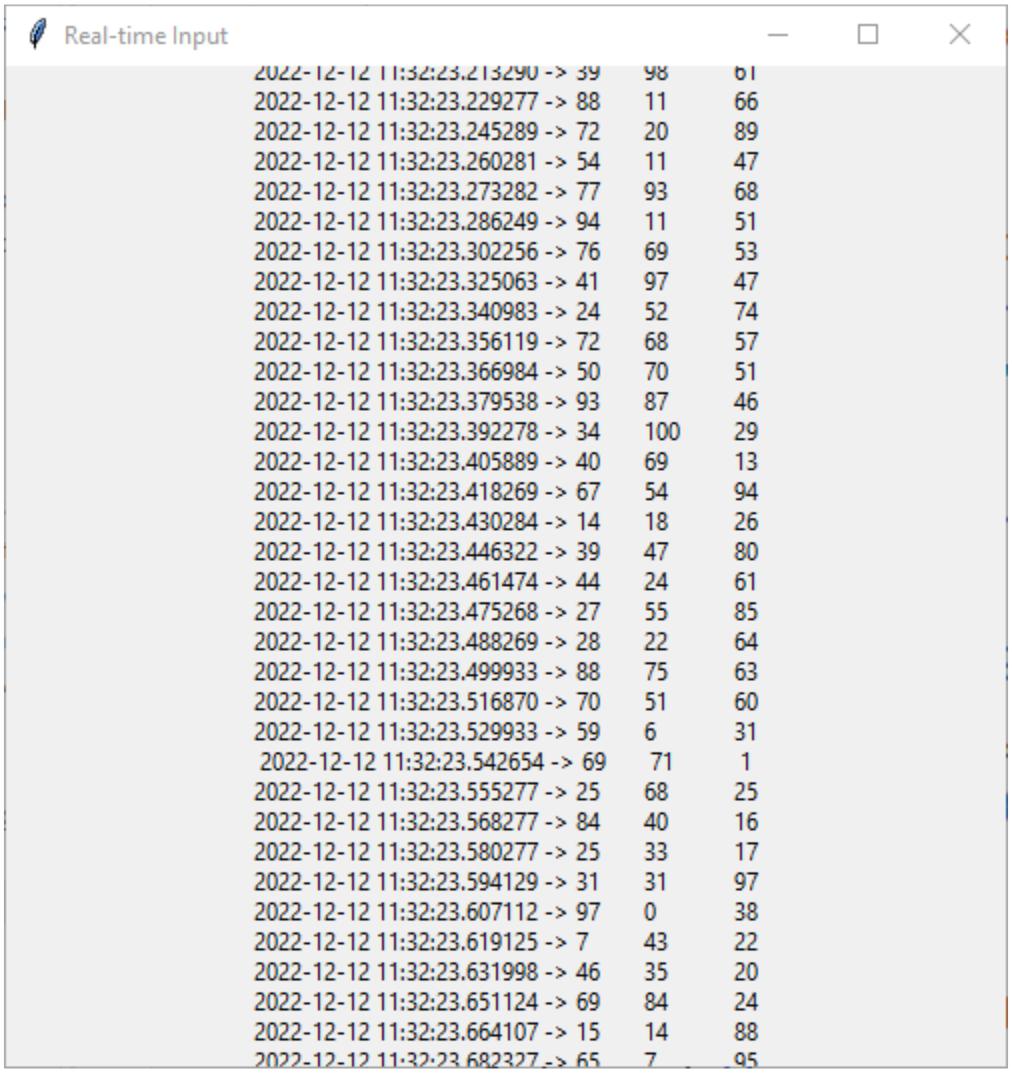
```
GLabel_403=tk.Label(root)
ft = tkFont.Font(family='Times',size=14)
GLabel_403["font"] = ft
GLabel_403["fg"] = "white"
GLabel_403["justify"] = "center"
GLabel_403["text"] = demo(input) # add functions here
GLabel_403['bg'] = "green"
GLabel_403.place(x=120,y=25,width=30,height=30)


def demo(input):
   if input == "On":
     return "On" # Return String type here
   else:
     return "Off" # Return String type here
```

    c. For Real-Time Sensor: Once the system gets the data, then the sensor will be displayed a new window line by line.

    Once click the “Real-time Input” button:



A new window will pop out and display the input data like this:



```
2022-12-12 11:32:23.213290 -> 39    98     61
2022-12-12 11:32:23.229277 -> 88    11     66
2022-12-12 11:32:23.245289 -> 72    20     89
2022-12-12 11:32:23.260281 -> 54    11     47
2022-12-12 11:32:23.273282 -> 77    93     68
2022-12-12 11:32:23.286249 -> 94    11     51
2022-12-12 11:32:23.302256 -> 76    69     53
2022-12-12 11:32:23.325063 -> 41    97     47
2022-12-12 11:32:23.340983 -> 24    52     74
2022-12-12 11:32:23.356119 -> 72    68     57
2022-12-12 11:32:23.366984 -> 50    70     51
2022-12-12 11:32:23.379538 -> 93    87     46
2022-12-12 11:32:23.392278 -> 34    100    29
2022-12-12 11:32:23.405889 -> 40    69     13
2022-12-12 11:32:23.418269 -> 67    54     94
2022-12-12 11:32:23.430284 -> 14    18     26
2022-12-12 11:32:23.446322 -> 39    47     80
2022-12-12 11:32:23.461474 -> 44    24     61
2022-12-12 11:32:23.475268 -> 27    55     85
2022-12-12 11:32:23.488269 -> 28    22     64
2022-12-12 11:32:23.499933 -> 88    75     63
2022-12-12 11:32:23.516870 -> 70    51     60
2022-12-12 11:32:23.529933 -> 59    6      31
 2022-12-12 11:32:23.542654 -> 69    71     1
2022-12-12 11:32:23.555277 -> 25    68     25
2022-12-12 11:32:23.568277 -> 84    40     16
2022-12-12 11:32:23.580277 -> 25    33     17
2022-12-12 11:32:23.594129 -> 31    31     97
2022-12-12 11:32:23.607112 -> 97    0      38
2022-12-12 11:32:23.619125 -> 7     43     22
2022-12-12 11:32:23.631998 -> 46    35     20
2022-12-12 11:32:23.651124 -> 69    84     24
2022-12-12 11:32:23.664107 -> 15    14     88
2022-12-12 11:32:23.682327 -> 65    7      95
```

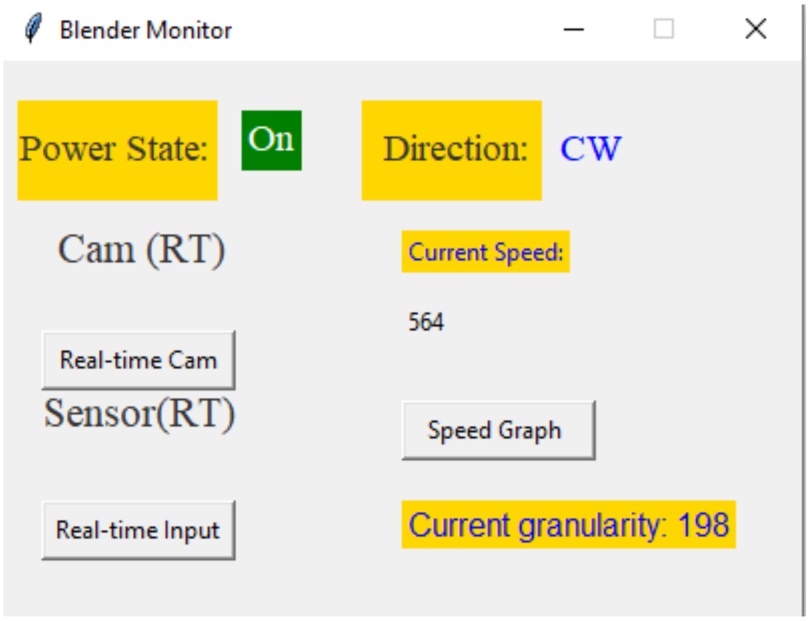
Here is the code for implementing them: (Demo sample)

```
# Show realtime input on the main window
        GLabel_1192 = tk.Button(root, text = 'Real-time Input',
command= App.realtime_input)
        GLabel_1192.place(x=20,y=220,width=97,height=30)
```

```
# Show the real time input on the pop-out window
    def realtime_input():
        newWindow = Toplevel(root)
        newWindow.title("Real-time Input")
        newWindow.geometry("500x500")

        def set_label():
            currentTime = datetime.datetime.now()
            label['text'] += str(currentTime) + " ->
"+str(random.randint(0,100))+"  "+str(random.randint(0,100))+"
"+str(random.randint(0,100))+"\n" # This is the Demo
            newWindow.after(10, set_label)
        label = tk.Label(master = newWindow, text="")
        label.pack()
        set_label()
```

d. For Real-Time Camera: Use Opencv package to grab the images from the camera and use PhotoImage and after function to show the real-time video from the camera

   Once click the “Real-time Cam” button:



A new window will pop out and display the input data like this:

Here is the code for implementing them: (Demo sample)

```
# Show realtime camera on the main window
        GLabel_120 = tk.Button(root, text = 'Real-time Cam',
command= App.cam_video)
```

```
        GLabel_120.place(x=20,y=135,width=97,height=30)
        label3=tk.Label(root,text='Current
Speed:',bg='gold',fg='blue',)
        label3.place(x=200,y=85)
```

```
# Show the real time camera on the pop-out window
def cam_video():
        # Solution: Click the link and check the real-time camara data
        newWindow = Toplevel(root)
        width, height = 800, 600
        cap = cv2.VideoCapture(0)
        cap.set(cv2.CAP_PROP_FRAME_WIDTH, width)
        cap.set(cv2.CAP_PROP_FRAME_HEIGHT, height)
        newWindow.bind('<Escape>', lambda e: newWindow.quit())
        lmain = tk.Label(newWindow)
        lmain.pack()
        def show_frame():
            _, frame = cap.read()
            frame = cv2.flip(frame, 1)
            cv2image = cv2.cvtColor(frame, cv2.COLOR_BGR2RGBA)
            img = Image.fromarray(cv2image)
            imgtk = ImageTk.PhotoImage(image=img, master = newWindow, width =
width, height = height)
            lmain.imgtk = imgtk
            lmain.configure(image=imgtk)
            lmain.after(10, show_frame)
        show_frame()
```

- Result (thoughts, insights, experience to share)
  Final result video Demo:
  https://youtu.be/Qlg62t8a3yM
  Thoughts & experience to share: after clicking the "Real-time Cam" button it always takes so much time to pop out a new window and display the real-time camera's view, if the part can be improved that will be better.
- Summary and Insight:
  Finished all codes for GUI and components. By building this GUI, I learned how to use Opencv package and some tools related to building GUI, such as Tkinter. The only thing left is to combine the previous codes with GUI codes together and test them out.

9. Project Summary:

- Yuzhang Huang: For this project, I already finished the code for checking condition and direction, also the code for GUI. The only thing left is combining these codes and testing them out. For personal experience, I had no experience with OpenCV and GUI design before this class. But from this class, I not only learned how to use them, but also learned how to use them to solve problems in real life. Also from this class, I learned how to decompose a big project into several small sub-goals, and then study and learn about each sub-goal. For example, retrieving the camera from the computer is a relatively complicated process, however, after my research and study, I found that I could achieve the goal of displaying the real-time camera by constantly generating new images and setting the refresh frame rate. Finally, I also learned how to organize a small team in this class and communicate with my teammates. In conclusion, this course helps me to connect my computer science knowledge with real-life problems and helps me to understand how to do research with a team.

- Yusuf Elazzazi: Throughout this project I learned how to collaborate with other to complete an open goal. Determining a problem statement, tasks and subtasks, and deadlines were all important to being successful. Further, I learned about different sensor methods and how to process the data received from them. I learned most about image processing and the many ways to accomplish a task, each with different tradeoffs. Learning the data formats of images, how color is encoded, and how to threshold and filter images for optimal analysis will be useful skills in the future. Collaborating with other teams was also a good experience, as we all had separate tasks but were guided by an overarching goal.

- Fardin Apurbo: This project was a great learning experience for me. Coming into this class, I had very little knowledge of what was happening. It took me a while to get the concepts down but now it made me much more confident to start a project from scratch and slowly figure out ways to implement and make the program better. We used various tools and techniques to get the desired results like google collab, accelerometer, cameras as well as get raw data from these sources and make something useful out of it. We have built models to simulate the performance of the blender under various conditions. The digital twin blender could be used to understand the performance of the blender so we could improve on our model to make the blender perform better. Understanding these raw data could help to understand and optimize the performance of the blender and design better algorithms to control the blender. I used a method called the .cumtrapz method in python that takes the cumulative integral of a given dataset, I used the accelerometer data to create a list and found the integral of them using the .cumtrapz method to find the velocity of the blender.
  Using different tools like the camera to get the picture of the particles inside the blender and then using image processing to determine the granularity of the particles in the blender was very interesting as well as it opens up other

possibilities that we could apply to this blender for other applications.
This project really helped me overcome my fear of starting projects from scratch as this is my first time working with a group of students to get something done. We communicated with different teams like the hardware team to get raw data from the devices that were connected to the blender, we also talked about where we should place the devices for optimal performance and accurate measurements.

Materials

- Yuzhang Huang: https://github.com/githuberuser/CSE-410.git
- https://colab.research.google.com/drive/14XiAmn5JYiBEdaUScqkVXHzFghCIVnYG?usp=sharing