# Classroom Occupancy Detection System

Stephanie Trusty

Old Dominion University

**Table of Contents**

List of Figures

Classroom Occupancy Detection System

Discovered in Wuhan, China in 2019, COVID-19 is a disease caused by the virus severe acute respiratory syndrome coronavirus 2 (SARS-CoV-2) (Centers for Disease Control and Prevention, 2021). In order to limit the spread of COVID-19, organizations such as the Centers for Disease Control and Prevention and the World Health Organization have advised the public to engage in social distancing, or physical distancing. According to the Centers for Disease Control and Prevention, social distancing entails maintaining a six foot distance from individuals who are not members of the same household in both indoor and outdoor settings (Centers for Disease Control and Prevention, 2021). The classroom detection system aims to enforce social distancing by capturing an overhead image of a classroom and determining if students are violating physical distancing protocol.

The system utilizes the RaspberryPi camera to capture overhead images of the classroom. If acceptable distancing conditions are detected, the system turns on a green light emitting diode (LED) to indicate this. To meet acceptable conditions, there must be at least one vacant seat between students. If adjacent occupied seats are detected, social distancing guidelines have been violated. In this case, a red light emitting diode is turned on to reflect the violation.

## 1. Problem Description

The primary objective of the classroom occupancy system is to capture an overhead image of a classroom, determine if there are any adjacent occupied seats in the classroom, and turn on the appropriate LED depending on whether or not a distancing violation was detected. Written in Python, the detection system is implemented using a Raspberry Pi 3 Model B

computer, a Raspberry Pi camera, a breadboard, two resistors, male-to-female jumper wires, and two light emitting diodes.

### 1.1 Raspberry Pi and the Camera Module

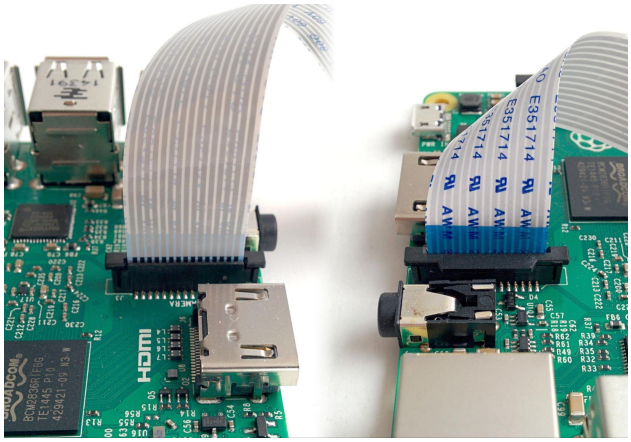
The Raspberry Pi 3 Model B is a single-board computer capable of both bluetooth and wireless local area network connectivity (Raspberry Pi Foundation, n.d.). The Model B has 40 general purpose input/output (GPIO) pins, a Camera Serial Interface (CSI) port for the Raspberry Pi camera, and a Micro SD port for data storage and loading the Raspberry Pi operating system (Raspberry Pi Foundation, n.d.). A keyboard and mouse can be connected to the computer through the USB ports, and a monitor can be connected using the HDMI port (Raspberry Pi Foundation, n.d.).

**Figure 1.1**
*Raspberry Pi 3 Model B*

In order to capture the overhead images necessary for image processing and occupancy detection, a Raspberry Pi Camera Module is connected via the CSI port, as seen in Figure 1.2. Specifically, the occupancy detection system makes use of the standard module, which is the Raspberry Pi Camera Module v2. This standard version of the camera module is designed to take pictures in natural light (Raspberry Pi Foundation, n.d.). Conversely, the NoIR version of the

camera module can be paired with an infrared light source to capture images in the dark. For the purposes of this system, however, the standard version is sufficient.

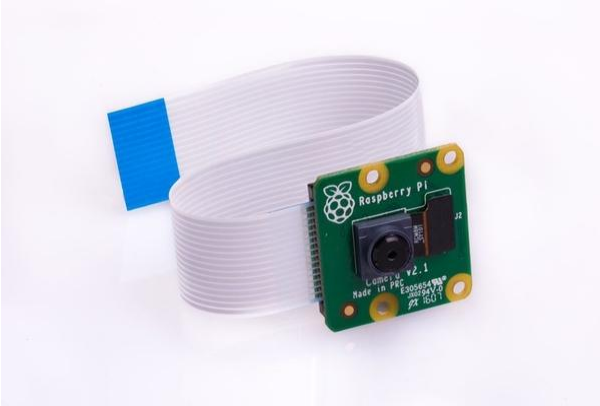
**Figure 1.2**
*Raspberry Pi Camera Module Port*




The Raspberry Pi Camera Module v2 has a Sony IMX219 8-megapixel image sensor (Raspberry Pi Foundation, n.d.). The Camera Module also makes use of Sony's Exmor R sensors, which has a backlit architecture (Raspberry Pi Foundation, n.d.). This architecture allows for smaller sensors and higher resolution without a decrease in light sensitivity (Sony, 2021). Capable of supporting various video modes as well as still capture, the v2 Camera Module attaches to the Raspberry Pi's CSI port through a 15-centimeter ribbon cable (Raspberry Pi Foundation, n.d.). The v2 Camera Module, pictured below, is compatible with all Raspberry Pi 1, 2, 3, and 4 models (Raspberry Pi Foundation, n.d.).

THIS SPACE INTENTIONALLY LEFT BLANK

**Figure 1.3**
*Raspberry Pi Camera Module v2*




In order to properly capture images of the simulated classroom environment, the Camera Module's ribbon cable is clamped to a (*insert wood dimensions*) piece of plywood using a binder clip. The ribbon cable and clip are positioned such that the Raspberry Pi camera faces the ground. The plywood rests on a table, allowing the camera to hang over it and capture overhead images. Figure 1.4 depicts the method used to mount the camera.

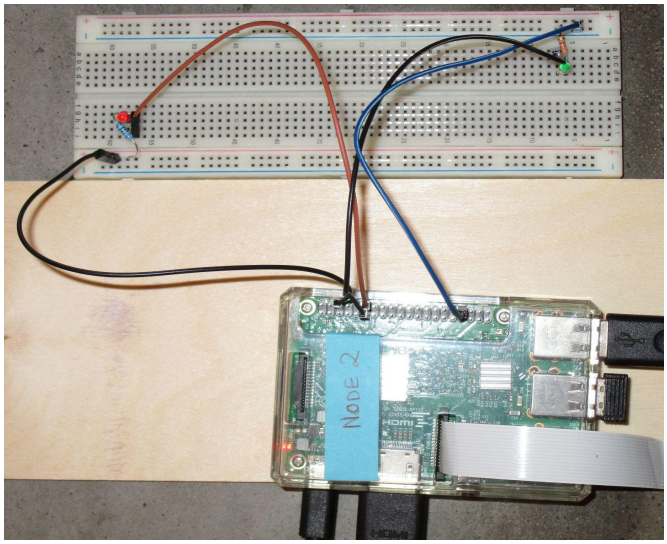
**Figure 1.4**
*Setup of Raspberry Pi and Camera Module*


THIS SPACE INTENTIONALLY LEFT BLANK

### 1.2 Breadboard and GPIO pins

To light the appropriate LEDS, the system includes a simple breadboard circuit. Each LED has a single resistor connected in series with the voltage channel and the diode's anode pin. This is done in order to limit the current passing through the LED. The LED's cathode pin is connected to ground using a male-female jumper wire.

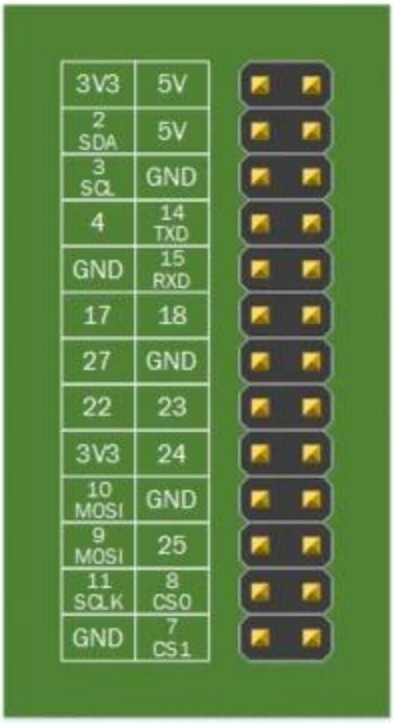
**Figure 1.5**
*Breadboard and LED circuits*




The Raspberry Pi's GPIO pins allow the computer to connect to and control electronic circuits. For both LED circuits, the male end of the jumper wire is connected to a hole on the breadboard's ground rail, while the female end is connected to one of the five GND pins found on the Raspberry Pi. To complete the circuit, the circuit's positive supply is connected to a GPIO pin. The green LED circuit and the red LED circuit use GPIO pins 17 and 18, respectively. Figure 1.6 , found below, depicts the GPIO layout found on the Raspberry Pi 3 Model B.

THIS SPACE INTENTIONALLY LEFT BLANK

**Figure 1.6**
*Raspberry Pi 3 Model B GPIO Layout*




## 1.3 Python Libraries and Packages

Written in Python, the classroom occupancy detection system is implemented using several Python libraries. These libraries will be briefly described here. Further details about the specific use of these libraries is available in Section 2, which details the system's algorithm flow and implementation.

### 1.3.1 PiCamera

The PiCamera library provides an interface for controlling the Raspberry Pi camera module with Python code. This package is available for Python 2.7 or above, and Python 3.2 or above. Within the system, this package is used to start a camera preview before taking a picture of the classroom; to stop the camera preview; and to capture the classroom image to a file.

THIS SPACE INTENTIONALLY LEFT BLANK

### 1.3.2 Python Imaging Library (PIL)

The Python Imaging Library provides imaging processing functionality to the Python interpreter. The classroom occupancy detection system utilizes the following PIL modules: Image, ImageChops, and ImageOps.

The Image module is used to represent a PIL image and provides functions to create new images, open an image from a file, and display an image. The system uses a function from this module to load the overhead classroom image that has been taken with the Raspberry Pi camera.

ImageChops provides channel operations, which are arithmetical image operations. The occupancy detection system utilizes the subtract function from this module. This function subtracts two PIL images, then divides the result by a scale value and adds an offset value, provided that values are given for the scale and offset. If these values are omitted, scale defaults to 1.0 and offset defaults to 0.0. The subtract function returns a PIL image that represents the differences between the two parameter images. See Section 2.3 for example images.

The ImageOps module provides image processing operations, such as functions to maximize image contrast, resize images, and colorize grayscale images. To reduce the complexity associated with comparing color images, the detection system uses a function from this module to convert the classroom pictures to grayscale images. See Section 2.2 for example images.

### 1.3.3 Numerical Python (NumPy)

Numerical Python, or NumPy, is an open-source Python library that provides multidimensional array and matrix data structures. The library provides a variety of methods to create arrays, perform mathematical operations on arrays, and otherwise manipulate the

numerical data stored within arrays. Within the classroom occupancy detection system, a NumPy array is used to represent the layout of the classroom. If a particular seat is occupied, a 1 is inserted into the array at the indices that represent that seat's location. For example, if a student is seated in the leftmost seat in the front row, a numerical value of 1 will be inserted into the array at the index (0, 0). Conversely, a value of 0 would be inserted at that location if the seat is empty.

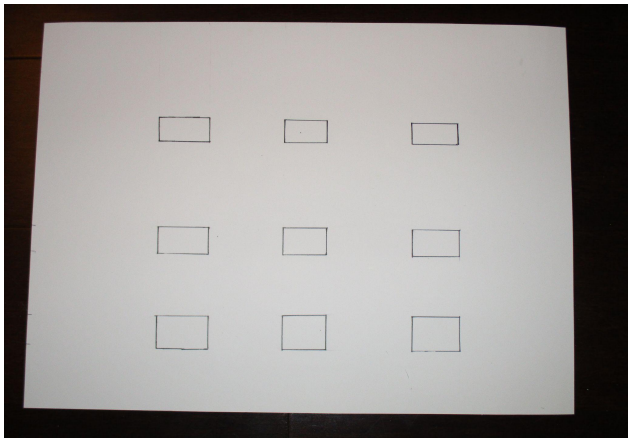
Additionally, the occupancy detection system also makes use of a NumPy function to convert an image to an array of pixel values. Slicing this array at the indices for a certain seat yields a smaller array of pixels for that particular seat. The system uses a NumPy function to compute the sum of a sliced array. The result of this calculation is used to make a determination of the seat's occupancy.

### 1.4 Limitations and Constraints

Before discussing the finer details of the solution, it is imperative to understand the limitations present in the testing and implementation of the classroom occupancy detection system. The most significant constraint is the absence of a real classroom. The classroom environment was simulated with a simple two-dimensional drawing. Simple squares represent the location of a seat, with a total of nine seats in the classroom. The figure below shows the classroom drawing used for development and testing.

THIS SPACE INTENTIONALLY LEFT BLANK

**Figure 1.7**
*Two-Dimensional Depiction of the Classroom Layout*



Additionally, students were represented by placing small toy figures on top of the drawing at seat locations. This is significant because of the simplicity of the simulated environment. The system is designed to detect adjacent occupancy in an environment with completely stationary seat locations. Furthermore, testing of the system was limited to a simple classroom layout consisting of three rows and three columns of seats. While the solution is designed to accommodate classrooms with more rows and columns, as well as those with fewer rows and columns, it is not implemented with more complex seat configurations in mind. For example, the system would be unable to accurately detect adjacent occupied seats in a circular layout.

## 2. Solution and Implementation

The classroom occupancy detection system determines whether or not a seat is occupied by capturing an overhead image of the classroom in its current state and comparing it to an image of the classroom when it is empty. This section details the solution's algorithm and necessary input files, as well as challenges encountered during development and implementation.

THIS SPACE INTENTIONALLY LEFT BLANK

### 2.1 Initial Challenges

The most significant challenge that arose during the development of the system concerned the issue of detecting adjacent seats. After comparing the current image of the classroom with the image of the empty classroom using image subtraction, the occupancy detection system needed to be able to determine if the occupied seats were located next to one another. Other methods were attempted to address this problem before the current solution was implemented. However, these alternate methods ultimately proved to be ineffective.

The first attempted solution aimed to utilize edge detection to address the matter of adjacent occupancy detection. The intention behind this idea was to identify the rectangular contours that represent the seat locations in the image. For each contour, the system would obtain the upper, lower, rightmost, and leftmost coordinate for the contour, then check the pixel values within that space in the same manner that the current implementation does. This proposed solution was attempted using OpenCV, an open-source library that provides computer vision functionality. Furthermore, this method was attempted in order to provide a more robust implementation that could automatically detect seat locations.

However, there proved to be several issues in using this method. Firstly, the system makes use of image subtraction in order to produce a simplified image that only displays the differences between the empty classroom and the current state of the classroom. As such PIL's subtraction function produces an image in which the unchanged pixels, or the features that are the same in both classroom images, are now depicted with a pixel value of zero. The system's implementation assumes that the classroom is largely stationary. As a result, the rectangular squares that represent the seat locations would be depicted as black pixels, with only the contours

of the students being detected. Furthermore, it should be noted that OpenCV's contour detection algorithm identifies all individual contours. Using this method would result in extraneous information or even incomplete information, as the student's entire body is not guaranteed to be treated as a single contour. In fact, considering factors such as clothing and personal belongings, it is highly likely that different parts of the student's body would be treated as unique and distinct contours.

By using this method, these additional factors would have to be taken into account in order for the occupancy detection system to make a decision based on this information. Rather than using the seat location to determine occupancy, the student's body itself would have to be considered in order to determine the location of a seat. If the solution were to be applied to a real world scenario, this presents another issue. The classroom occupancy detection system continuously captures photos and detects distancing violations until it is terminated by a user. As such, the system would likely capture images of students as they are entering the classroom and approaching seats. In this context, using the presence of a person's body to define the location of a seat would result in inaccurate detection results. Furthermore, this method requires the system to differentiate between humans and non-human objects that may be entering the classroom, adding additional complexity to the problem.

Though edge detection and contour detection were quickly ruled out, the second attempted solution still aimed to resolve the problem without requiring additional input files. However, the second implementation attempt was directly designed around the use of image subtraction. This method entailed converting the subtracted image to a NumPy array, iterating through the array to search for a cluster of non-zero pixels, and finally searching for an adjacent

cluster of non-zero pixels. Several challenges arose when utilizing this approach, however. Firstly, in order to detect adjacent students, this method requires that a threshold distance be defined. If the system simply looks for non-zero pixels along the same x-axis or y-axis, occupied seats that are not directly next to each other but are still in the same row or column would be counted, and the system would falsely report a distancing violation. To avoid this, the system would still need an additional input value: an integer value that represents the pixel distance between students. Depending upon the layout of the room, multiple distance values would need to be provided. In terms of reducing the number of input resources that the end-user would need to provide, this solution would offer few additional benefits over the final implementation that was chosen. Additionally, this method proved to have slow execution times, since the system had to iterate through the entire image to search for adjacent students.

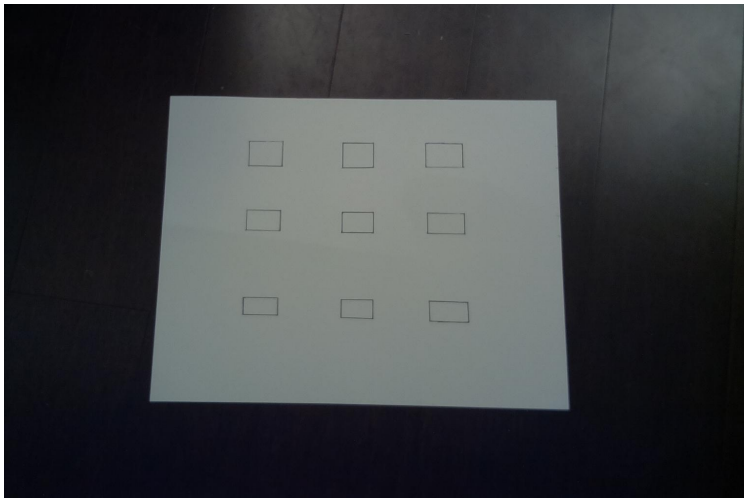
In terms of real-world functionality, this solution presents the similar problems as the contour and edge detection solution. Since the system would be looking for groupings of non-zero pixels, any item that wasn't present in the unoccupied classroom image would be represented with non-zero pixels in the subtracted image. For example, if a student placed their bag on the desk, the classroom occupancy detection system would look for an adjacent group of pixels within a certain distance. This has the potential to result in inaccurate violation detection.

**2.2 Input Files**

In order to perform the necessary image processing and to determine seat occupancy, two materials must be prepared in advance: an empty classroom image and a text file consisting of seat locations. The empty classroom image is used as a point of comparison for the current image of the classroom that is captured while the system is running. Without a picture of the empty

classroom, the occupancy detection system is unable to determine if a seat is currently occupied. Figure 2.1 shows the empty classroom image that was used for testing and development purposes. This image was captured with the Raspberry Pi Camera Module.

**Figure 2.1**
*Empty Classroom Image*



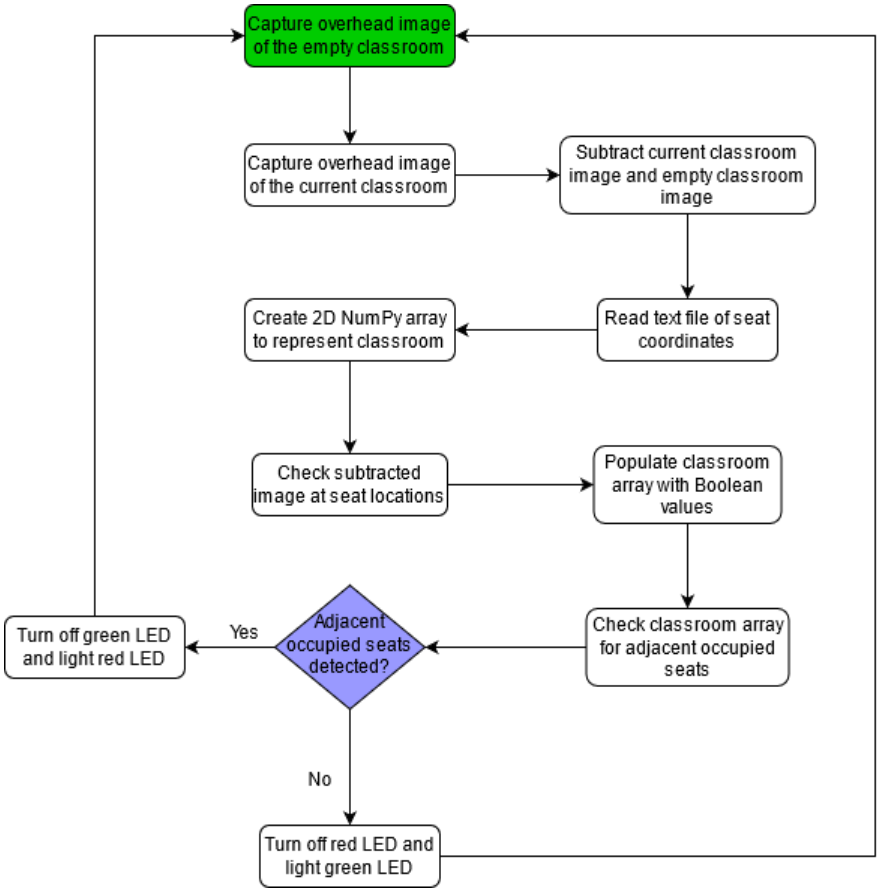
The text file of seat coordinates defines the areas where a student may be seated. Each line of the file should contain the upper, lower, right, and left coordinates for a single seat. These coordinates are used to create slices of the larger NumPy array. This allows the system to compute a sum of the individual seat array and determine whether or not that seat is occupied.

**2.3 Algorithm Flow**

The classroom occupancy detection system is designed to repeatedly execute its algorithm until the user elects to terminate the process. The system captures an image of the classroom in its current state. This current classroom image is subtracted from the empty classroom image, which is provided before the system begins execution. The text file of seat coordinates, which was also provided before execution, is read by the system and a two-dimensional NumPy array is created based on the coordinates from this file. At each seat

location, the pixel values are summed and the array is populated based on these values. Once the entire array has been populated, the system checks for adjacent occupied seats. Based on the result of this, the appropriate LED is turned on and a message is displayed to the console. The system captures a new photo of the classroom and this process repeats. The figure below details the algorithm process.

**Figure 2.2**
*Classroom Occupancy Detection Algorithm Flow*



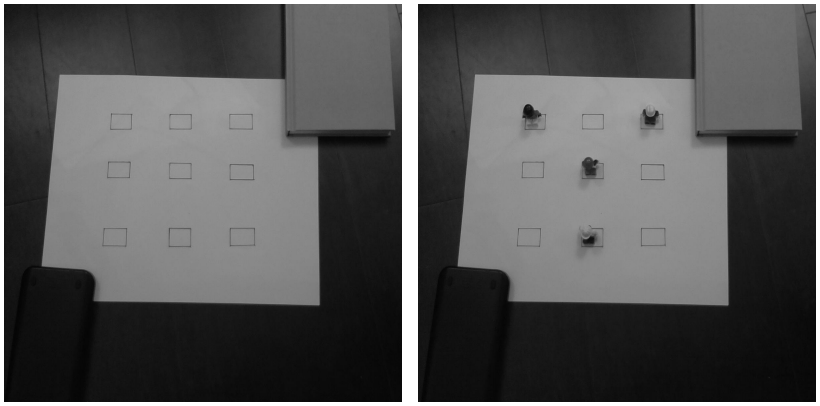

### 2.3.1 Image Capture and Processing

First, an overhead image of the classroom is captured using the Raspberry Pi camera. The method responsible for this functionality displays a preview of the classroom for five seconds,

then captures and saves the photo to the Raspberry Pi's home folder. This image of the classroom in its current state is converted to grayscale to reduce complexity. Similarly, the image of the empty classroom is also converted to grayscale. The two classroom images are subtracted from one another using the subtraction method the Python Imaging Library. This function returns a PIL image that only depicts the pixels that are different between the two parameter images.

For example, consider the following figures, which were captured with the Raspberry Pi Camera Module and converted to grayscale using the aforementioned process:

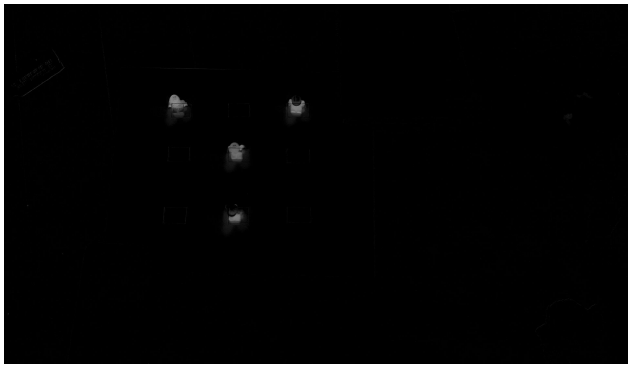
**Figure 2.3**
*Sample Parameter PIL Images for the Subtraction Function*



Subtracting these two images from each other using the Python Imaging Library function results in a simplified image that only depicts the toy figures with non-zero pixels. This can be seen in Figure 2.4, which only displays the pixels that are not common to both of the parameter images.

THIS SPACE INTENTIONALLY LEFT BLANK

**Figure 2.4**
*Sample Image Returned by the PIL Subtraction Function*



### 2.3.2 NumPy Arrays and Occupancy Detection

In order to determine if adjacent seats are occupied, the coordinates of a seat must be provided. This is done by reading a text file that denotes the leftmost, rightmost, upper, and lower coordinates of each seat in the classroom. All lines of the text file are read, with each line denoting coordinates for a single seat. Each line is stripped of white spaces and commas. The coordinates are then appended to a list. The resulting list of seat coordinates is then used to establish the layout of the classroom.

For each set of coordinates, two pairs of tuples are created: a tuple containing the upper and lower coordinates and a tuple containing the left and right coordinates. The upper and lower tuple is added to a list of y-coordinate values, and the left and right tuple is added to a list of x-coordinates. Seats that are in the same row will have the same y-coordinates, whereas seats in the same column will have the same x-coordinates. As such, duplicate coordinate tuples are removed from the two lists. The lengths of the x-coordinate list and the y-coordinate list are obtained and stored. The length of the x-coordinate list denotes the number of columns in the

classroom's layout, and the length of the y-coordinate list denotes the number of rows. These values will be used to create the two-dimensional NumPy array that represents the classroom.

Next, the list of seat coordinates is used to check the subtracted image for occupied seats. For each set of coordinates in the list, the sum of pixel values in that section of the subtracted image is obtained. If the sum is above the given threshold, the occupancy flag is set to 1, meaning that the seat is occupied. Otherwise, the flag is 0, denoting an unoccupied space. The current value of the occupancy flag is appended to a list. Once this process has been completed for each set of seat coordinates, the list of flags is converted into a one-dimensional array. Using the values for the classroom's number of rows and number of columns, the array is then reshaped to reflect the classroom's layout.

Consider the subtraction parameter images in Figure 2.3. Given a text file of seat coordinates for this classroom layout, the system would produce a 3x3 NumPy array. The populated array for the current state of this classroom is shown below in Figure 2.5.

**Figure 2.5**
*NumPy Array Based on Figure 2.3*

```
[[1 0 1]
 [0 1 0]
 [0 1 0]]
Adjacent students detected. Number of compromised students: 2
```

### 2.3.3 Detecting Adjacency with NumPy Array

With the classroom array populated and reshaped, the system iterates through the array. During iteration, it checks for an occupied seat, which is denoted by a value of 1. If the seat is occupied, the system then checks for an occupied seat in the next row and in the next column. If an adjacent occupied seat is detected, the indices of both seats are added to a list of compromised

seats. Once the entire array has been checked, any duplicate seat indices are removed from the list. The length of this list is used to print a message to the console that states the number of compromised students in the classroom. If the length is greater than 0, the system sets the GPIO pin connected to the red LED to HIGH. Otherwise, the pin connected to the green LED is set to HIGH instead.

## 3. Discussion and Results

The system's functionality was tested with various configurations of occupied seats, all within the three-by-three simulated classroom discussed in Section 1.4. Three particular scenarios will be discussed here: a classroom with no seating violations and zero compromised students, a classroom with a single seating violation and two compromised students, and a classroom with multiple seating violations and nine compromised students.

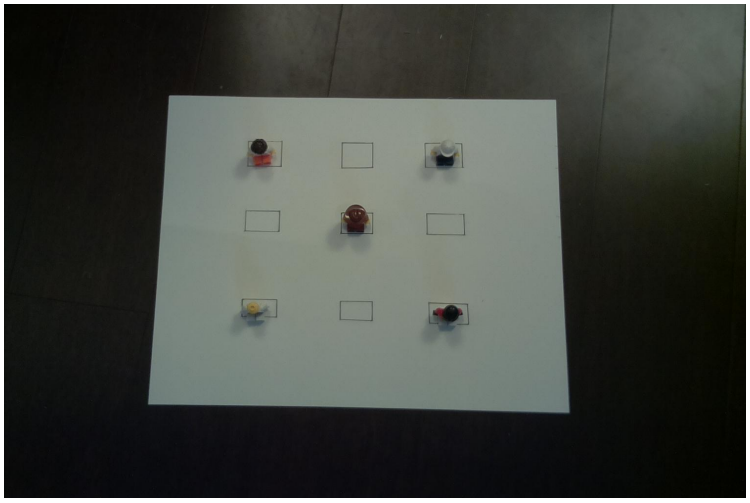
It should be noted that, for testing purposes, the NumPy array that represents the classroom's layout will also be printed to the console. This is to ensure that the system correctly populates the array in a method that is consistent with the classroom image. In a real-world context, this information would not be directly presented to the end-user.

### 3.1 Case Study 1: No Seating Violations

The first test case involves executing the classroom occupancy detection system on a classroom with multiple students who are properly social distancing. Given the layout of the simulated classroom, a maximum of five students can be seated in the room without violating the social distancing protocol. For this scenario, two toy figures are placed in the left and right seat in the front row; two more are positioned in the left and right seat in the back row; and a single

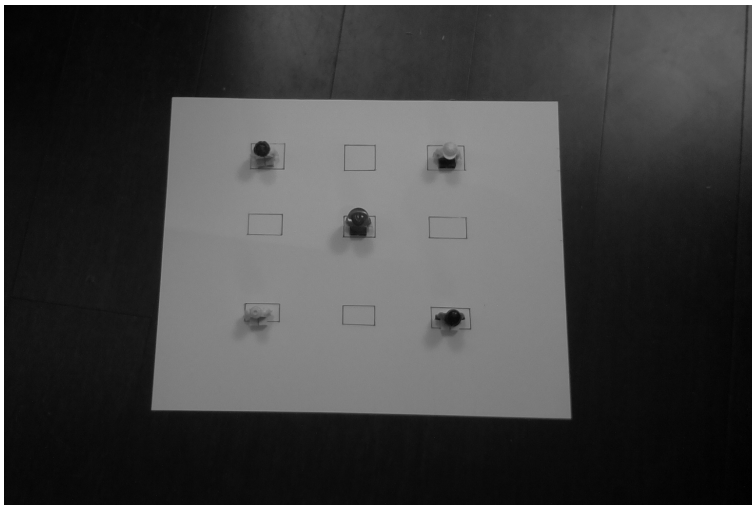
figure is placed in the center seat in the middle row. The figure below shows the seating configuration for this case study, which was captured with the Raspberry Pi Camera Module.

**Figure 3.1**
*Seating Layout for Test Case 1 (No Seating Violations)*



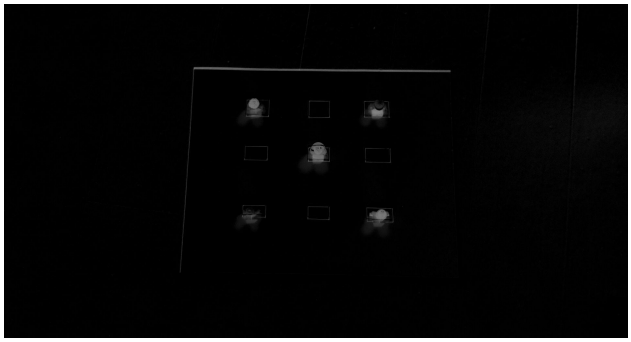
The goal of this test is to ensure that the system can confirm that all seated students are abiding by the physical distancing requirements and can light the green LED on the breadboard to convey this information. Converting this image to grayscale results in the following image:

**Figure 3.2**
*Seating Layout after Grayscale Conversion*



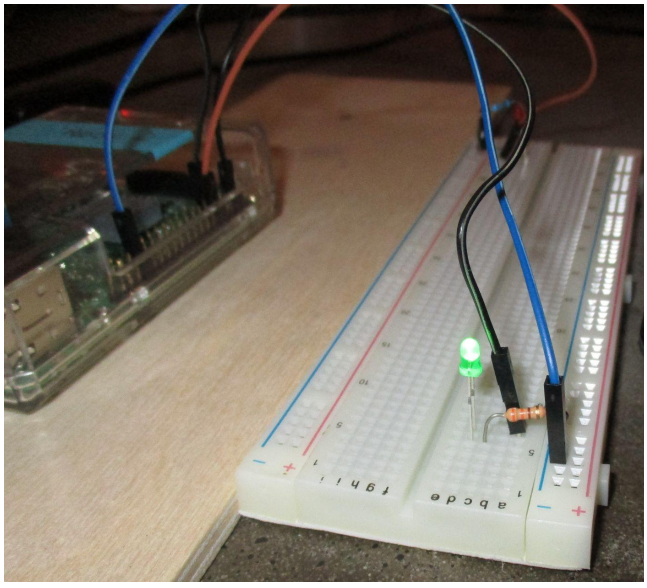
This image is subtracted from the empty classroom image shown in Figure 2.1. This operation results in the PIL image shown in Figure 3.3.

**Figure 3.3**
*Subtracted Image for Test Case 1*



The test produces the desired results, as the green LED has been turned on and the console message confirms that there are no students at risk.

**Figure 3.4**
*LED Output for Test Case 1*



THIS SPACE INTENTIONALLY LEFT BLANK

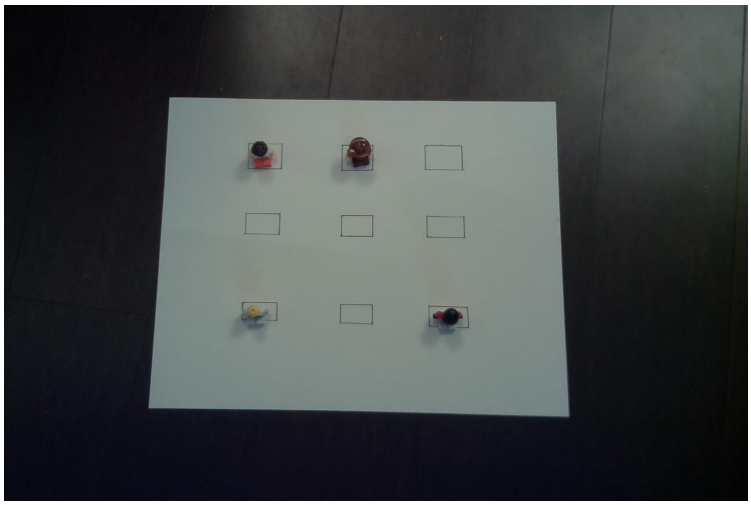
**Figure 3.5**
*Console Output for Test Case 1*

```
[[1 0 1]
 [0 1 0]
 [1 0 1]]
No adjacent students detected.
```

### 3.2 Case Study 2: A Single Seating Violation

The conditions for this test are as follows: two students in the front row are properly distanced from each other, with a single empty seat between the two of them. However, the two students seated in the back row are directly next to each other.

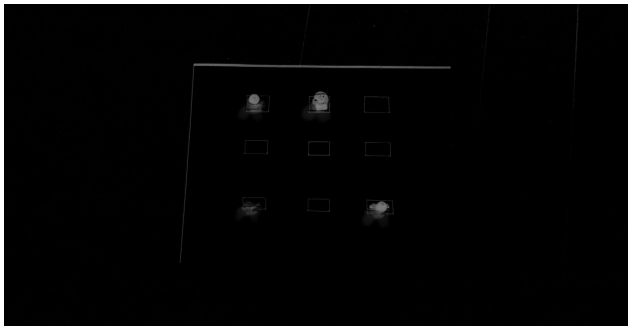
**Figure 3.6**
*Seating Layout for Test Case 2 (Single Seating Violation)*



Though this scenario is primarily concerned with ensuring that the system can correctly detect violations, this test is also executed directly after the first case test. This is to ensure that the system can continuously respond to changes in the classroom throughout execution. If the classroom moves from a state of compliance to one of violation, the detection system also needs to ensure that the green LED is turned off before communicating that there has been a violation.

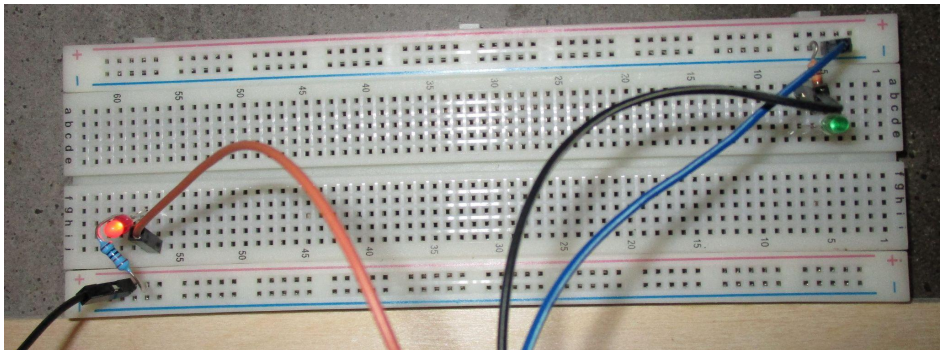
Subtracting the test image from the empty classroom image produces the following PIL image, shown in Figure 3.7.

**Figure 3.7**
*Subtracted Image for Test Case 2*



Due to the distancing violation in the back row, the system turns on the red LED and the console message informs the end-user that there are two students at risk. This confirms that the system is equipped to handle scenarios where the classroom contains both compliant and at-risk students. The LED output and the console output for this scenario can be seen in Figure 3.8 and Figure 3.9, respectively.

**Figure 3.8**
*LED Output for Test Case 2*



THIS SPACE INTENTIONALLY LEFT BLANK

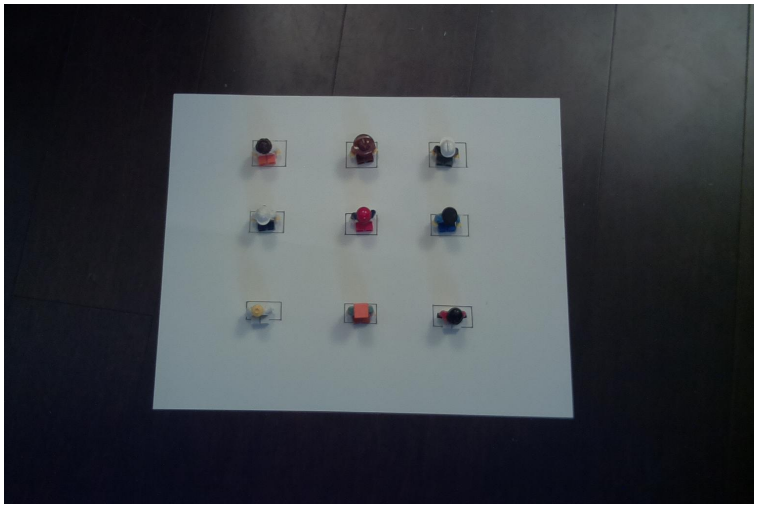
**Figure 3.9**
*Console Output for Test Case 2*

```
[[1 1 0]
 [0 0 0]
 [1 0 1]]
Adjacent students detected. Number of compromised students: 2
```

### 3.3 Case Study 3: Multiple Seating Violations

This final test case is concerned with simulating a scenario in which multiple seating violations are occurring. The core goal of this case study is to ensure that the system accurately determines the number of compromised students in a scenario where an individual student is seated next to more than one other student. The figure below depicts the seating configuration for this case study.

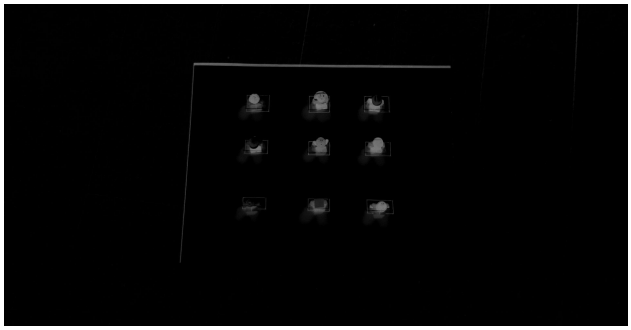
**Figure 3.10**
*Seating Layout for Test Case 3 (Multiple Seating Violations)*



Subtracting this test image for the empty classroom image shown in Figure 2.1 results in the following:

THIS SPACE INTENTIONALLY LEFT BLANK

**Figure 3.11**
*Subtracted Image for Test Case 3*



For this test case, the red LED is turned on and the console message is displayed, informing the user that there are nine compromised students in the classroom. The LED output and console output can be seen in Figure 3.12 and Figure 3.13.

**Figure 3.12**
*LED Output for Test Case 3*

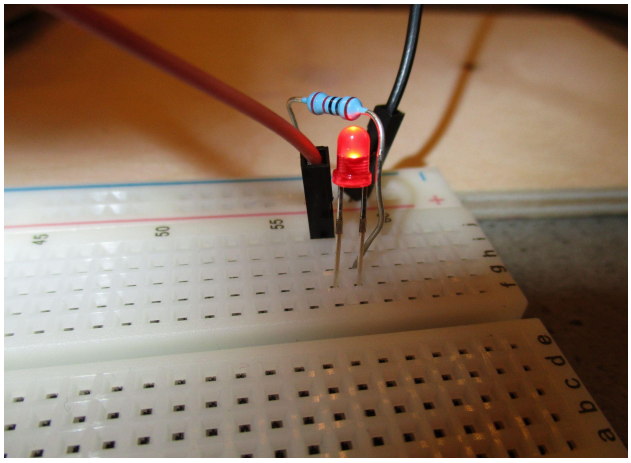


**Figure 3.13**
*Console Output for Test Case 3*

```
[[1 1 1]
 [1 1 1]
 [1 1 1]]
Adjacent students detected. Number of compromised students: 9
```

THIS SPACE INTENTIONALLY LEFT BLANK

## 4. Future Considerations

As discussed in Section 1.4, the current implementation of the classroom occupancy detection system is designed to operate in a completely stationary classroom. In the event that the layout of the classroom changes, the system is not equipped to immediately respond to these changes. An updated empty classroom image and an updated text file of seat coordinates would need to be provided for the system to function accurately. Additionally, the system's design does not account for personal belongings and student behaviors. For example, if a student were to place their bag or other personal belongings in an empty seat, the detection system would likely identify this as an occupied seat. As such, it would falsely report a seating violation because the system is simply designed to look for significant pixel changes.

Machine learning image recognition tools should be considered to improve the current implementation of the classroom occupancy detection system. An object detection or image labelling model could be used to identify and differentiate between humans and non-human objects in the classroom image. This would have multiple benefits. Firstly, object detection capabilities would allow the system to detect desks and chairs within the classroom and determine the location of these objects. Utilizing a trained object detection model on the image of the empty classroom would allow the system to identify seat locations without the use of an input text file. These locations can then be stored and used to check for non-zero pixels in the subtracted image. Incorporating image labelling or image classification functionality would allow the system to label non-human objects that enter a seat's location and ignore them, ensuring that a seat isn't falsely marked as occupied. These changes would allow for a more

robust detection system that is better suited to a real-world context and that requires fewer input files from the end-user.

## Conclusion

The classroom occupancy detection system successfully utilizes the Raspberry Pi Camera Module to capture overhead images of a simulated classroom environment and detects the presence of occupied seats to determine if the students are complying with social distancing guidelines set by organizations such as the Centers for Disease Control and Prevention. Once an overhead image was captured, this current photo of the classroom was subtracted from the empty classroom image, which was provided to the system as an input file. Using a text file of seat coordinates, which was also provided as an input file, the system created a two-dimensional NumPy array to represent the classroom's layout. The subtracted image was converted to an NumPy array and sliced at the given seat coordinates. For each seat array, a sum is taken and compared against a threshold value to determine if a seat is occupied and to populate the classroom array with the appropriate Boolean value. In order to ensure proper functionality, this system was tested under three different scenarios using the classroom layout shown in Figure 2.1.

The first scenario tested a classroom under compliant conditions. The seating for this test case consists of five occupied seats, all of which are properly distanced from each other. This test case produced an array that was representative of the seating layout and turned on the green LED, as expected.

The second case study aimed to ensure that the system was capable of detecting a single distancing violation amidst otherwise compliant conditions. Additionally, this test was

implemented directly after the first test case to ensure that the detection system could turn off one LED before turning on the other. This seating layout consisted of two properly distanced students in the back row, and two adjacent students in the front row. Again, the test resulted in an accurate NumPy array and console message stating that there are two compromised students in the classroom. The green LED was turned off and the red LED was turned on.

The final case study aimed to ensure that the system could accurately determine the number of compromised students, even in a scenario where a student is seated next to more than one other student. In this scenario, every seat in the classroom was occupied. This test also produced a proper NumPy array and illuminated the red LED.

Though the classroom occupancy detection system produced accurate results under all three case study scenarios, there are significant limitations present that may hinder the system's functionality and usability in real-world applications. The current implementation is only ideal for stationary, unchanging classroom environments with uniform layouts that can be easily represented as a two-dimensional NumPy array. Furthermore, the solution does not account for other objects and belongings being introduced into the classroom environment. There is no protocol to distinguish between human and non-human objects, which could result in inaccurate detection results in a real world scenario. In order to produce a system suitable for real world applications, it is recommended that the current image processing method be replaced by a trained object detection or image classification model. Doing so would eliminate the need for the end-user to prepare a seat coordinate file and would allow the system to ensure that the detected object is in fact a student.

# References

Centers for Disease Control and Prevention. (2021, June 11). *How to Protect Yourself and Others*. Centers for Disease Control and Prevention. https://www.cdc.gov/coronavirus/2019-ncov/prevent-getting-sick/prevention.html

Centers for Disease Control and Prevention. (2021, July 9). *About COVID-19 Frequently Asked Conditions*. Centers for Disease Control and Prevention. https://www.cdc.gov/coronavirus/2019-ncov/faq.html#Basics

Clark, A. (2021, July 6). *Pillow (PIL Fork) 8.3.1 Documentation*. Pillow (PIL Fork). https://pillow.readthedocs.io/en/stable/

Jones, D. (2017, February 25). *Picamera 1.13 Documentation*. Picamera. https://picamera.readthedocs.io/en/release-1.13/

The NumPy Community. (2021, June 22). *NumPy v1.21 Manual*. NumPy. https://numpy.org/doc/stable/user/whatisnumpy.html

Raspberry Pi Foundation. (n.d.). *Camera Module V2*. Raspberry Pi Foundation. https://www.raspberrypi.org/products/camera-module-v2/

Raspberry Pi Foundation. (n.d.). *Raspberry Pi 3 Model B*. Raspberry Pi Foundation. https://www.raspberrypi.org/products/raspberry-pi-3-model-b/

Sony. (2021, March 30). *What is the difference between Exmor and Exmor R sensors?* Sony Support. https://www.sony.com/electronics/support/articles/00070962

TensorFlow. (2021, May 15). *Image Classification*. TensorFlow. https://www.tensorflow.org/lite/examples/image_classification/overview#model_description

TensorFlow. (2021, May 18). *Object Detection*. TensorFlow.

https://www.tensorflow.org/lite/examples/object_detection/overview