OUR Journal: ODU Undergraduate Research Journal

Volume 9 Article 9

2022

ODU Undergraduate Students Addressing the Societal Problems of Parking Control, Classroom Seating, and Flood Monitoring in Hampton Roads

Stephanie K. Trusty
Old Dominion University

Gabriel Del Razo
Old Dominion University

Nathan Potter
Old Dominion University

Soad Ibrahim
Old Dominion University

Ayman Elmesalami
Old Dominion University

Recommended Citation
Trusty, Stephanie K.; Del Razo, Gabriel; Potter, Nathan; Ibrahim, Soad; and Elmesalami, Ayman (2022) "ODU Undergraduate Students Addressing the Societal Problems of Parking Control, Classroom Seating, and Flood Monitoring in Hampton Roads," *OUR Journal: ODU Undergraduate Research Journal*: Vol. 9, Article 9.
DOI: 10.25778/4w68-6786
Available at: https://digitalcommons.odu.edu/ourj/vol9/iss1/9

ODU Undergraduate Students Addressing the Societal Problems of Parking Control, Classroom Seating, and Flood Monitoring in Hampton Roads

By Stephanie Trusy, Gabriel Del Razo, Nathan Potter, and Soad Ibrahim

Faculty Advisor: Dr. Ayman Elmesalami, University Distinguished Teacher, ODU

ABSTRACT— During the summer of 2021, ODU undergraduate computer science students undertook image processing research projects. These projects focused on utilizing the Raspberry Pi computer and camera module to address three real-world problems concerning parking control, classroom seating, and flood monitoring. The parking lot occupancy project aimed to develop a system that monitors the occupancy of parking spaces in a lot and communicates the status of the lot drivers and the lot attendants. The COVID-19 classroom occupancy project sought to enforce social distancing protocols in a classroom environment by detecting seating violations and notifying the instructor and the impacted students of the violation. Designed for the Hampton Roads community, the flood detection project concerned the development of a vision system, controlled by the Raspberry Pi, that detects the flood levels of a particular location and determines if the flooding is low, moderate, or severe. This paper details the development of these projects and proposes future considerations and recommendations for further undergraduate study and improved real-world functionality.

Keywords: Raspberry Pi, image processing, classroom occupancy, parking lot occupancy, flood detection

Table of Contents

1. Introduction: Parking Control, Classroom Occupancy, Flood Monitoring

Over the summer of 2021, undergraduate computer science students at Old Dominion University undertook individual research projects. In addition to providing hands-on programming experiences for undergraduate students, these projects sought to address three relevant, real-world problems: determining the occupancy of a parking lot or garage during high-occupancy events (1.1); enforcing social distancing protocols in a classroom setting (1.2); and detecting flood levels on Hampton Roads roadways (1.3). Each of the resulting systems utilizes the Raspberry Pi computer and its camera module to capture image inputs for processing and communicates information about these images using light-emitting diodes (LEDs).

1.1 Parking Control

For the average driver, parking is often a tedious but unavoidable task. On average, Americans spend 17 hours a year searching for available parking (INRIX, Inc., 2017). In addition to its associated time consumption, parking also has notable economic and behavioral implications. A 2017 study conducted by INRIX found that, while searching for parking, drivers accumulate a cost of $345 in wasted time, fuel and emissions (INRIX, Inc., 2017). Additionally, parking availability and access often informs the driver’s experience and actions, resulting in them feeling stressed while parking, missing appointments, abandoning their trips, and even avoiding airports and sports and leisure activities due to parking (INRIX, Inc., 2017).

The purpose of the parking lot occupancy system is to simplify the parking experience for both the driver and the lot attendant. This system captures top-down images of a parking lot and uses image processing techniques to determine the lot’s occupancy. First, the system captures an initial reference image of the parking lot when it is empty. It then captures real-time images of the parking lot. The real-time images are compared to the reference image to determine the occupancy of the lot. The system quantifies this determination by subtracting the pixel values of the real-time image from the reference image. The subsequent value is compared to a heuristic threshold to ultimately determine whether the parking lot is full or open. The current state of the lot is communicated through light-emitting diodes: a green LED is turned on if the lot is open, and a red LED is turned on if the parking lot is at full capacity.

1.2 Classroom Occupancy

COVID-19 is a disease caused by severe acute respiratory syndrome coronavirus 2, which was discovered in Wuhan, China in 2019 (Centers for Disease Control and Prevention, 2021). In addition to vaccination and donning masks in public indoor spaces, 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, to limit the spread of COVID-19. 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 COVID-19 classroom occupancy detection system aims to support adherence to social distancing protocols within classroom environments.

1.3 Flood Monitoring

The Hampton Roads region is located in the middle of the Eastern seaboard in the commonwealth of Virginia and comprises ten independent cities and six counties (Hampton Roads Chamber, n.d.). Geographically, the region consists of multiple low-lying, coastal cities that are prone to flooding. According to the First National Flood Risk Assessment, conducted in 2020, 20% of properties in the city of Virginia Beach were at risk of flooding, as were 27% of

Norfolk properties, 24% of Portsmouth properties, and 33% of Hampton properties (First Street Foundation, Inc, 2020). Furthermore, land sinkage and rising sea levels are steadily putting Hampton Roads at increased flood risk. In a report on initiatives to combat the community's rising sea levels, the National Oceanic and Atmospheric Association states, "Virginia's Hampton Roads region is experiencing the highest rates of sea level rise along the entire Atlantic seaboard and is second only to New Orleans as the largest U.S. population center at risk" (NOAA Office for Coastal Management, n.d.). The region is expected to see a 1.7 foot rise in water levels by 2050 (Coutu, 2020).

Flooded roadways pose a significant risk for drivers and their vehicles. The CDC's National Center for Environmental Health notes that floods are the second deadliest weather phenomena in the United States, resulting in an average 98 deaths per year (National Center for Environmental Health, 2020). Most commonly, flood deaths occur when a vehicle is driven through hazardous flood waters (National Center for Environmental Health, 2020). According to the National Weather Service and the National Oceanic and Atmospheric Association, one foot of rushing water is capable of carrying away most cars and two feet of water can carry away most trucks and SUVs (National Weather Service & National Oceanic and Atmospheric Association, n.d.). Additionally, driving through a flooded street can cause significant damage to motor vehicles, including electrical damage such as short circuiting wires and computer malfunctions or damage to safety features resulting in malfunctioning antilock braking systems and airbag system failure (CARFAX, 2021). This has potential health and safety consequences even after the initial flooding event. CARFAX estimates that 378,000 flooded cars are back in use in the United States despite potential damage to mechanical, electrical, and safety features (CARFAX, 2021).

The flood detection project seeks to address the safety issues associated with flooded roadways in Hampton Roads. Its purpose is to develop a device and a compatible program that can determine the safety of a roadway by measuring the height of the floodwater and alert drivers of the current flood conditions with LED lights. The green LED is used to indicate low flooding. The yellow LED denotes moderate flooding, and the red LED denotes severe flooding.

Each of these projects provides an opportunity for undergraduate students to gain experience designing and implementing a solution to a timely, real-world problem. In addition to allowing students to utilize and expand upon the programming and software development skills they've garnered throughout their academic career, the project's hardware requirements provide hands-on experience with the Raspberry Pi computer, the Raspberry Pi Camera Module, and breadboard circuits.

2. Related Works

Ioannis Sakiotis designed the Parking Lot Occupancy Detector in the spring of 2021. This system was developed to determine if a parking lot is completely occupied or if it contains open spaces. Sakiotis developed this system with the Raspberry Pi computer. The system's methodology compares a real-time image of the parking lot to a base image of the parking lot, which depicts the lot when it is empty. The captured images are converted to grayscale to lower the complexity when comparing the pixel values between the base and real-time images. This comparison of the real-time image to the base parking lot image is then used to detect changes to the lot's occupancy. Sakiotis' system converts the images into arrays, then slices this image array into smaller two-dimensional arrays that contain the pixel values for each parking space. This methodology relies upon the Sum of Absolute Differences (SAD), a digital image processing technique that assesses the similarity between two regions of an image. Sakiotis utilizes the SAD technique in conjunction with a heuristic threshold when comparing the base parking space array to the real-time parking space array. Based on the derived calculation, if the difference is above this heuristic threshold, the space is determined to be occupied. If the difference is below the threshold, the space is determined to be open. If every space in the lot is occupied, a red LED is illuminated. Otherwise, a green LED is illuminated.

An implementation of the classroom occupancy detection system was completed by Ioannis Sakiotis in April 2021. The project sought to confirm required social distancing in a classroom environment by detecting the presence of vacant seats among students. The resulting system utilized the Raspberry Pi computer, Pi Camera Module, a breadboard, and a single green light-emitting diode.

The system first captures a top-down image of the classroom when there are no students seated in it, and then captures a real-time top-down photo of the classroom. In the segments of the image where students are seated, the system compares the pixel values. Sakiotis' implementation works under the assumption that a static camera position is maintained during program execution. It also assumes that the environment's lighting is consistent in both the initial image of the empty classroom and the real-time classroom image.

A heuristic threshold value is used to determine if the difference in pixel values at the seat locations is "little" or "significant." If the pixel difference is "little," the seat is determined to be vacant. A "significant" pixel difference indicates an occupied seat. If acceptable social distancing conditions are detected in the classroom, Sakiotis' system turns on a green LED. However, if it detects adjacent occupied seats, the green LED is turned off.

In the spring of 2021, Mohammed Nauman Siddique developed a program and device that measures the water levels in a tank of water, which simulates a flooded roadway environment. Siddique utilizes a Raspberry Pi computer, camera, and breadboard to accomplish this. The system captures a 256 × 256 resolution image of the simulated environment and processes it to illuminate one of three LED lights. To determine the water levels in the tank, Siddique captured images of the tank at low, moderate, and high flood levels and used these images to train an image classification model. The system then compares incoming images against the trained model to determine current flood levels.

3. Background

Each project utilizes a Raspberry Pi 3 Model B computer, a Raspberry Pi Camera Module, and multiple LED breadboard circuits. This section details the components and functionality of this equipment.

3.1 Raspberry Pi 3 Model B

The first model in the third-generation Raspberry Pi, the Raspberry Pi 3 Model B, is a single-board computer capable of both Bluetooth and wireless local area network connectivity. The computer utilizes the BCM2837 Broadcom CPU, which incorporates a quad core ARM Cortex A53 cluster. The ARM cores found in this model run at 1.2 GHz, resulting in a computer that is approximately 50% faster than the Raspberry Pi 2.

The Model B includes 40 extended general-purpose input/output pins; a Camera Serial Interface port for Raspberry Pi Camera Modules; a Display Serial Interface port to connect a compatible touchscreen display; and a Micro SD port that loads the Raspberry Pi operating system and provides data storage. The computer's four USB ports can be used to connect a keyboard and mouse, and the full-size HDMI port can be used to connect a monitor or television.

Figure 3.1

Raspberry Pi 3 Model B

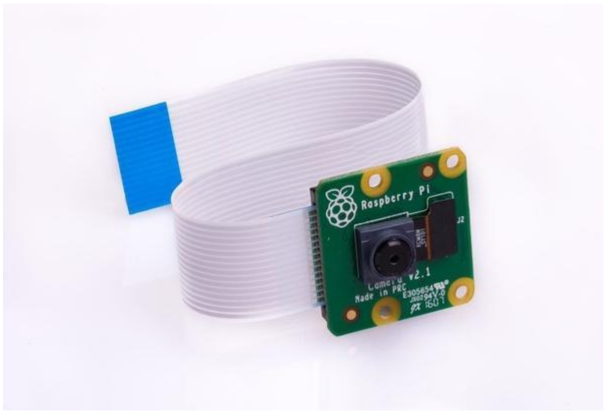
3.2 Raspberry Pi Camera Module

Each project utilizes a Raspberry Pi Camera Module to capture the photos necessary for image processing. These projects specifically make use of the Camera Module v2, the standard version of the Raspberry Pi Camera Module. Unlike the NoIR version, which can be paired with an infrared light source to capture images in the dark, the standard version of the Camera Module is solely designed to take pictures in natural light.

The v2 Camera Module supports both still capture and a variety of video modes. It attaches to the Pi's CSI port through a 15-centimeter ribbon cable. The Camera Module v2 utilizes a variety of sensors, including a Sony 8-megapixel image sensor and Sony's Exmor R sensors. The Exmor R sensor has a backlit architecture, allowing for smaller sensors and higher resolution without decreasing light sensitivity. The Raspberry Pi Camera Module v2, pictured below, is compatible with all Raspberry Pi 1, 2, 3, and 4 models.

Figure 3.2

Raspberry Pi Camera Module v2



3.3 GPIO Pins and LED Circuits

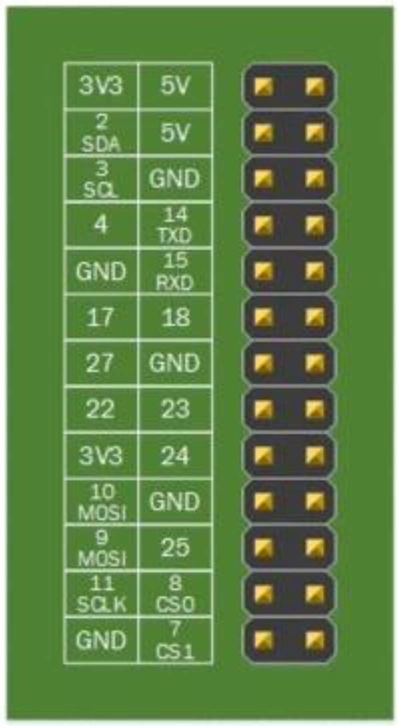
Breadboard circuits are an integral aspect of all three projects. These circuits function as an output for each system, communicating whether or not acceptable conditions are met using light-emitting diodes (LEDs). This section details how the breadboard LED circuits interface with the Raspberry Pi computer. Male-female jumper wires are used to connect the computer and the breadboard circuits.

LEDs are polarized components that allow current to flow in a single direction. They feature two terminals: an anode, which is the positive terminal, and a cathode, which is the negative terminal. In order to avoid damaging the diode, the flow of current through the component is limited by placing a resistor in series with the voltage channel and the diode's anode pin. The cathode is connected to the breadboard's ground channel.

As discussed in Section 3.1, the Raspberry Pi 3 Model B features 40 general-purpose input/output (GPIO) pins, which allow the computer to interface with and control electronic circuits. To connect the circuits to ground, the male end of the jumper wire is inserted into the breadboard's ground rail, while the female end is connected to one of the Pi's five GND pins. A second jumper wire is used to connect the circuit's positive supply with the male end connecting to the breadboard's voltage rail and the female end connecting to a GPIO pin on the Raspberry Pi. The following image depicts the layout of GPIO pins for the Raspberry Pi 3 Model B.

Figure 3.4

Raspberry Pi 3 Model B GPIO Layout



4. METHODOLOGY

Despite the similarities between the individual image processing projects, unique solutions were developed for each project. This section discusses the chosen methodology for each system, including hardware and software components, external packages and libraries.

4.1 Parking Lot Occupancy Controller Methodology

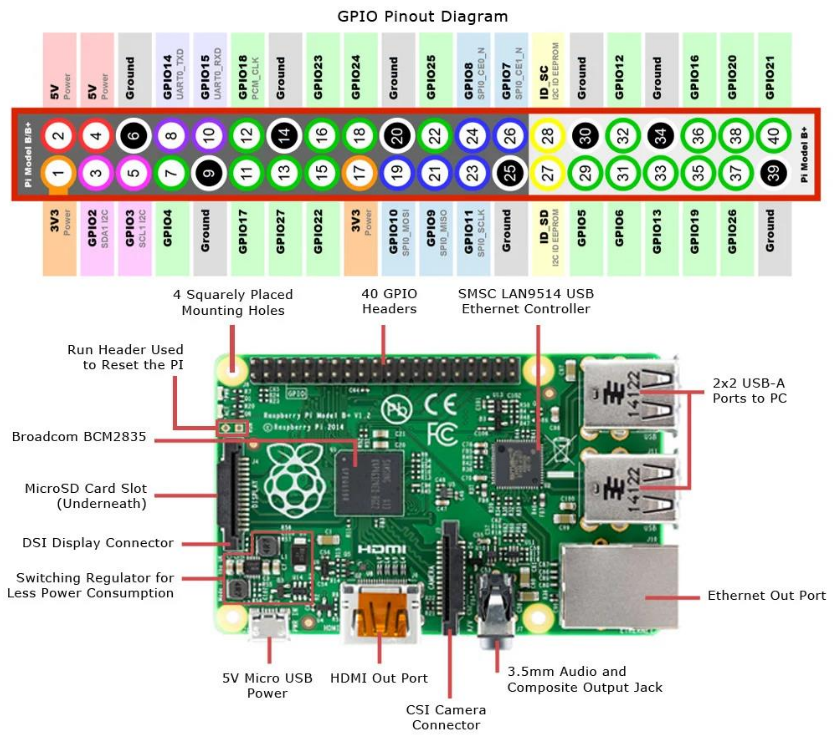
The parking lot occupancy controller is implemented using the Raspberry Pi 3 Model B computer and a compatible Camera Module. The system's output consists of two breadboard circuits, each of which illuminates a light-emitting diode. These breadboard circuits are wired to the Raspberry Pi, allowing the LEDs to be controlled with Python code.

4.1.1 Hardware Configuration

The system's methodology begins with designing the hardware setup to achieve the basic requirements of the system, which were to capture an overhead view of the parking lot and implement two functional LED circuits. This hardware design begins with the wiring of the LED circuits.

Figure 4.1

Raspberry Pi Component Diagram



[Diagram of Raspberry Pi GPIO Pinouts and Components]. (n.d.). https://www.jameco.com/Jameco/workshop/circuitnotes/raspberry_pi_circuit_note_fig2a.jpg

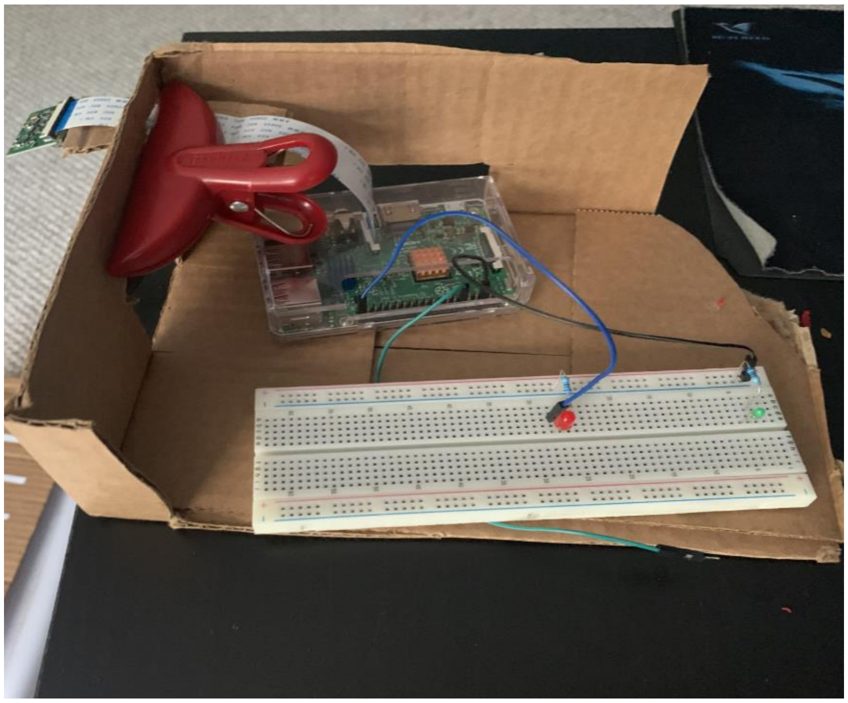
The output circuits interface with the Raspberry Pi through the computer's general-purpose input/output (GPIO) pins, featured in Figure 4.1. The LED connections are then tested with a simple Python program, utilizing the GPIOZero library. GPIO Zero is a Python library that provides a simple interface to access the Raspberry Pi's GPIO pins (Nuttall, 2021). This library is installed by default on the Rasbian operating system, the Raspberry Pi OS.

The camera installation is the final component of the system's physical setup. The camera was installed by lifting the fastening bar on the Camera Module port of the Raspberry Pi, inserting the camera's ribbon cable into the port, and then lowering the bar to secure the connection. To ensure the proper connection of the Camera Module, a simple Python program was used for testing. This program utilized the PiCamera Python library to test the functionality of the camera.

For this system to be effective, it needs to compare the pixel values between two images to detect changes in the parking lot's occupancy. In order to successfully accomplish this, the pixel coordinates must match between the image of the empty parking lot and the image of the parking lot in its current state. As such, a fixed camera position was needed to achieve consistent pixel coordinates. As portrayed in Figure 4.2, a housing unit with a stabilizing brace is used to mount and support the system's hardware. The housing unit is designed out of a cardboard box. Sections in the front and side of the cardboard box are extracted to accommodate the connection wires and the Camera Module. The camera is positioned to capture overhead images of the parking lot and is supported by a piece of cardboard secured with a plastic clip.

Figure 4.2

Physical Setup - Raspberry Pi, Camera Module, and Breadboard



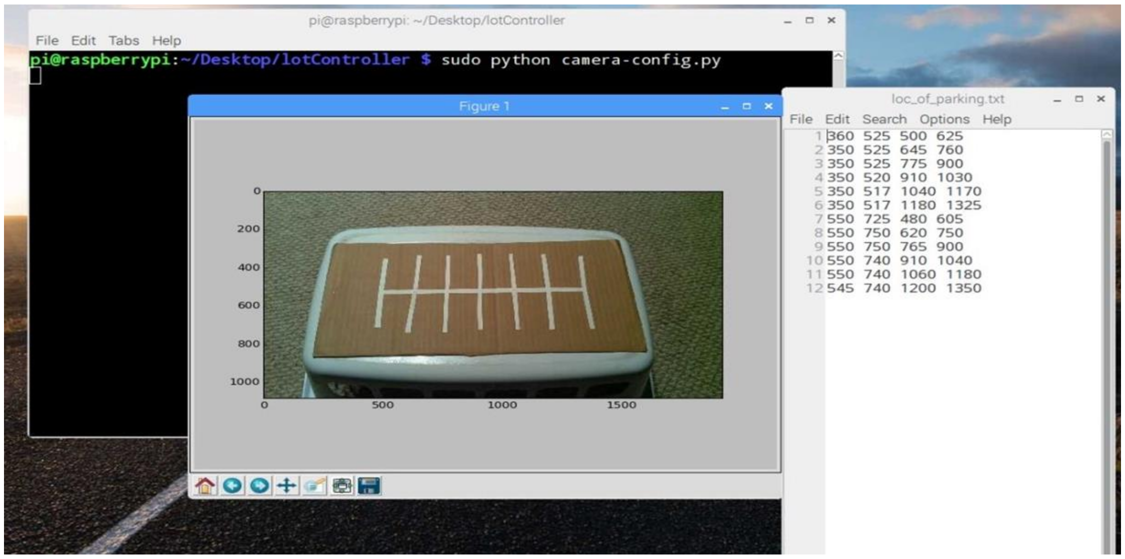
4.1.2 Software Configuration

The system’s program consists of two components: the controller configuration and the main program. The controller configuration begins system execution and is responsible for capturing and analyzing the reference image of the parking lot. The configuration process is completed by the end-user. Within the main program, the system runs autonomously. This section of the system captures a live image of the parking lot, compares it to the reference image, and displays the quantitative results, as well as lights the LED representing the parking lot's current occupancy.

The controller configuration component configures the system to the parking lot that the user wishes to monitor. This process begins by capturing a reference image of the parking lot when it is completely empty. When the controller is later executed, this simplifies the process of determining whether an individual parking space is occupied or not. The second configuration task requires the user to analyze the reference image and record sets of pixel coordinates that correlate to each parking space in the lot. The program utilizes two Python libraries to achieve this. NumPy, a library that provides multidimensional array and matrix data structures, is used to convert the reference image into an array. The array consists of each pixel value that comprises the image. Each pixel consists of three color channels: red, green, and blue. Each of these channels has a value ranging from 0 to 255, with a value of 0 denoting zero light and a value of 255 denoting maximum light. The program then uses Matplotlib to plot the NumPy array. Matplotlib is a plotting library that allows for interactive visualization of arrays (Hunter et al., 2021). In this case, the array is plotted to display the reference image. The plot has x-axis and y-axis values that describe the resolution of the image. The user can then use the tools provided by the Matplotlib library to zoom in on each parking space individually to find the coordinate set that describes the location of a space. The user will need to record each coordinate set into a text file and store that file in the same directory as the system's program. This text file is opened and read during the controller execution and is used to locate the parking spaces in the real-time image. This emphasizes the importance of the camera position being consistent between each image.

Figure 4.3

Configuration Controller - Plotted Reference Image and Coordinate Text File


pi@raspberrypi: ~/Desktop/lotController
File Edit Tabs Help
Figure 1
loc_of_parking.txt
File Edit Search Options Help

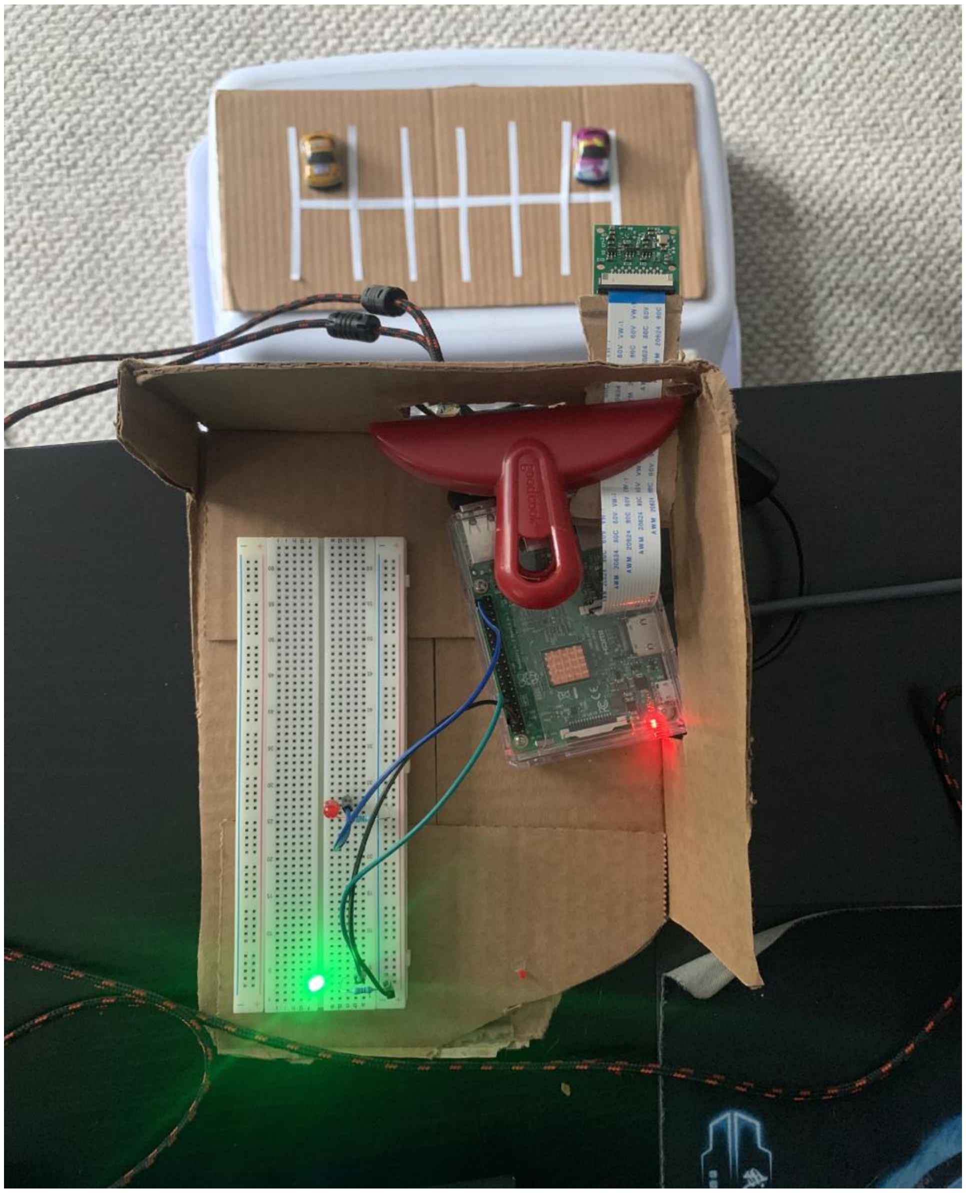
```
pi@raspberrypi:~/Desktop/lotController $ sudo python camera-config.py
```

```
360  525  500  625
350  525  645  760
350  525  775  900
350  520  910  1030
350  517  1040  1170
350  517  1180  1325
550  725  480  605
550  750  620  750
550  750  765  900
550  740  910  1040
550  740  1060  1180
545  740  1200  1350
```

After completing the configuration of the controller, the system then executes the main program. The main program starts by initializing the GPIO pins that are connected to the LED circuits. It then imports the reference image previously captured during the controller configuration phase.

Figure 4.4

Real-Time Image of the Parking Lot with Two Cars



Next, the program captures a real-time image of the parking lot. The real-time image is converted to a NumPy array, similar to the reference image. As previously noted, each image captured is in RGB format with three color channels that comprise each pixel. To simplify the complexity of comparison between the reference image and the real-time image, the program removes both the red and blue channels from each image, keeping only the green channel. It should be noted that there is no significant difference in system efficiency or execution by keeping or removing a different set of channels. The goal of removing channels is to lower the total pixel value so that the difference between the comparisons is lower.

With the green color channel extracted from both the reference and the real-time images, the program then reads the text file containing the coordinates sets for the parking spaces. The program uses the coordinate set to slice the array at the parking space's coordinate location and store the two-dimensional array slice into a list. This process is performed for both the reference image and the real-time image. As such, there are two lists of arrays: one for the reference parking spaces and another for the real-time spaces. The two lists are compared using the Sum of Absolute Differences (SAD) formula. This statistical technique is used to determine the degree of change between the two images. It compares the pixel value in the reference image and subtracts it from the pixel value at the same location in the real-time image. It then takes the absolute value of that difference and repeats the process for all pixels in the images. This will derive a final quantitative value that describes the difference between the two images. The program finds the SAD for each parking space and stores it in a new list. It then iterates through the list of SAD values, comparing each value to a heuristic threshold. This threshold was developed through trial testing of the system. If the SAD is below the threshold, the correlating parking space is noted as "open." If it is above the threshold, the space is noted as "occupied." After the iteration of the SAD list is complete, the program outputs the results to the screen, describing each parking space as shown below in Figure 4.5.

Figure 4.5

Sample Console Output

- Parking Space 1: The SAD between the images: 1028757
- Parking Space 2: The SAD between the images: 98644 (OPEN)

- Parking Space 3: The SAD between the images: 143499 (OPEN)
- Parking Space 4: The SAD between the images: 137998 (OPEN)
- Parking Space 5: The SAD between the images: 157849 (OPEN)
- Parking Space 6: The SAD between the images: 807202
- Parking Space 7: The SAD between the images: 69269 (OPEN)
- Parking Space 8: The SAD between the images: 80465 (OPEN)
- Parking Space 9: The SAD between the images: 127150 (OPEN)
- Parking Space 10: The SAD between the images: 100623 (OPEN)
- Parking Space 11: The SAD between the images: 95051 (OPEN)
- Parking Space 12: The SAD between the images: 157508 (OPEN)
- Number of open parking spaces: 10

The program concludes by illuminating the appropriate LED. If the number of open spaces is not zero, the program lights the green LED, indicating that the parking lot has open spaces. Otherwise, the red LED is illuminated, indicating that there are no open spaces and the parking lot is full.

4.2 COVID-19 Classroom Occupancy Detection System Methodology

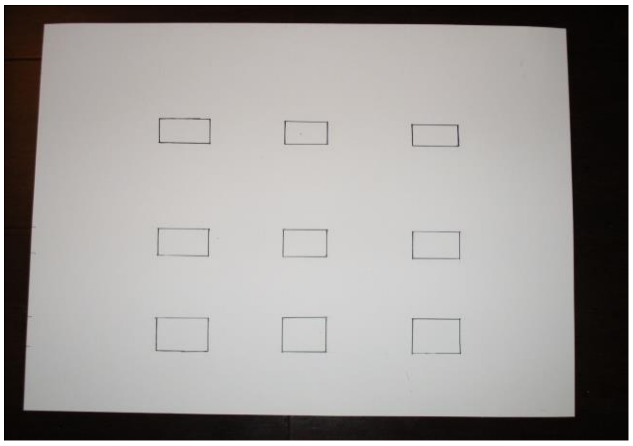
The classroom occupancy detection system uses the Raspberry Pi computer and its Camera Module to capture overhead images of a simulated classroom environment. If the system detects acceptable physical distancing conditions, a green light-emitting diode is turned on to reflect this. In order to meet acceptable distancing requirements, there must be at least one vacant seat between students. Any instance of adjacent occupied seats constitutes a violation of social distancing protocols. In the event that the system detects a violation, a red light-emitting diode is turned on to indicate this. Additionally, the system also displays a message to the console, indicating the number of students affected by the violation.

4.2.1 Depicting the Classroom Environment

For the implementation and testing of the classroom occupancy detection system, a classroom environment was simulated using a two-dimensional drawing on a piece of 27.94 x 35.56 centimeter poster board with squares denoting the location of a seat. A simple 3 row, 3 column layout was chosen, resulting in a total of nine seats in the classroom. The figure below shows the classroom environment used for development and testing.

Figure 4.6

Two-Dimensional Depiction of the Classroom Layout



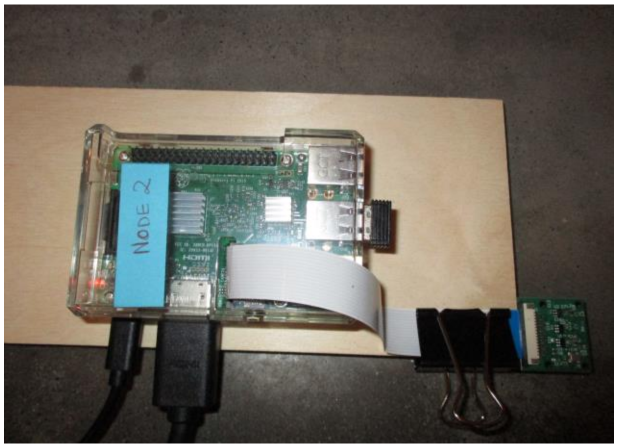
To represent a classroom occupied by seated students, small toy figures were placed on top of the drawing at seat locations.

4.2.2 Raspberry Pi, Camera Module, and Breadboard Configuration

To effectively capture overhead images of the simulated classroom, the Camera Module’s ribbon cable is clamped to a 6.3 x 101 x 304 millimeter 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. The simulated classroom is positioned on the ground beneath the camera at a distance of approximately 74.93 centimeters. Figure 4.7 depicts the method used to mount the camera.

Figure 4.7

Setup of Raspberry Pi and Camera Module

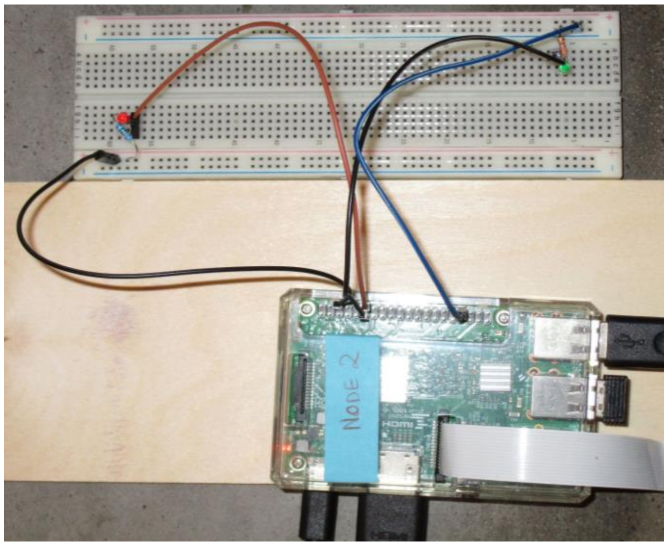


To light the appropriate LEDs, the system includes two simple breadboard circuits. Each LED has a single resistor connected in series with the voltage channel and the diode's anode pin. This is done to limit the current passing through the LED. The LED's cathode pin is connected to ground using a male-female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

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 4.8 depicts the layout of the system's LED circuits.

Figure 4.8

Breadboard and LED Circuits



4.2.4 Python Libraries and Packages

Written in Python, the classroom occupancy detection system utilizes various Python libraries and packages. This section will briefly detail the functionality of each library, as well as their use within the classroom occupancy detection system.

4.2.4.1 PiCamera

Available for Python 2.7 or above and Python 3.2 or above, the PiCamera library provides an interface for controlling the Raspberry Pi Camera Module with Python code (Jones, 2018). The system uses this package to start a camera preview prior to taking a picture of the classroom. Additionally, the PiCamera package is also used to stop the camera preview and to capture the classroom image to a file.

4.2.4.2 Python Imaging Library

The Python Imaging Library provides image processing functionality to the Python interpreter (Clark, 2021). The classroom occupancy detection system utilizes three specific PIL modules: Image, ImageOps, and ImageChops.

The Image module is used to represent an PIL image and provides functions to create new images, open an image from a file, and display an image. Once an overhead image has been taken of the classroom in its current state, the system uses PIL's Image module to load the current classroom image.

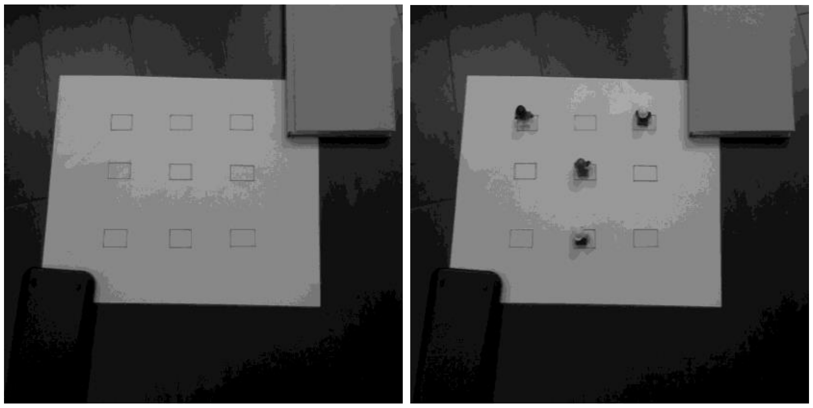
The ImageOps module provides image processing capabilities, such as functions to colorize grayscale images, maximize image contrast, and resize images. In order to reduce the complexity associated with comparing color images, the classroom occupancy detection system uses a function from this module to convert the classroom images to grayscale prior to performing the ImageChops subtraction function.

ImageChops provides channel operations, or arithmetical image operations, such as addition, subtraction, multiplication, and division. In particular, the occupancy detection system uses the subtraction function from this module. This function subtracts two PIL images from one another and 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 not explicitly provided, a default value of 1.0 is used for the scale, and a default value of 0.0 is used for the offset. The subtraction function returns a PIL image that represents the differences between the two parameter images. This resulting image only depicts the pixels that are not common to both parameter images. All pixels that are common to both images are depicted as black or zero-value pixels.

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

Figure 4.9

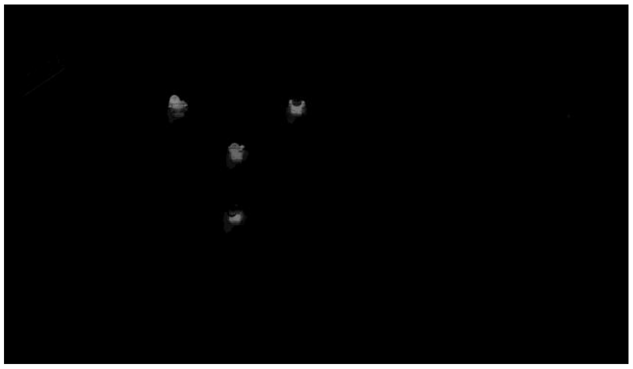
Sample Parameter PIL Images for the Subtraction Function



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 4.10, which only displays the pixels that are not common to both of the parameter images.

Figure 4.10

Sample Image Returned by the PIL Subtraction Function



4.2.4.3 Numerical Python

Numerical Python, also known as NumPy, is an open-source Python library that provides multidimensional array and matrix data structures (The NumPy Community, 2021). The library includes methods to create arrays, perform mathematical operations on arrays, and otherwise manipulate the numerical data within an array. The classroom occupancy detection system uses a NumPy array to represent the seating configuration of a classroom as well as the occupancy of each seat. If a seat is currently occupied, the system inserts a value of 1 into the array at the indices that represent that seat's location in the room. Consider an example in which a student is seated in the leftmost seat in the front row. To reflect this seat's occupancy, a numerical value of 1 will be inserted into the array at index (0.0).

Additionally, the classroom occupancy detection system 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.

4.2.1 Algorithm Flow

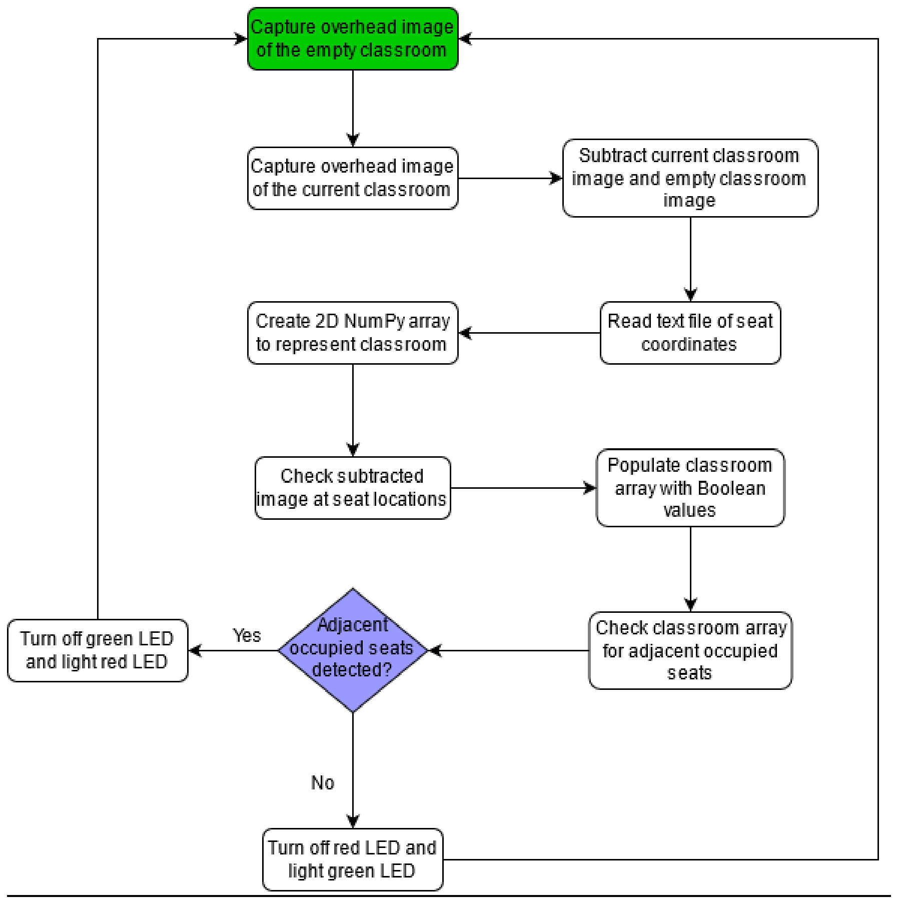
The COVID-19 classroom occupancy detection system is designed to execute repeatedly until the end-user elects to terminate the process. First, the system captures an overhead image of the classroom in its current state using the Raspberry Pi Camera Module. A preview of the classroom is displayed for five seconds. After that, the photo is captured and saved to the Raspberry Pi's home folder. To reduce complexity, this image of the classroom in its current state is converted to grayscale. Likewise, the image of the empty classroom, which was captured prior to program execution and provided as an input file, is also converted to grayscale. These two classroom images are then subtracted from another, resulting in an image that only depicts the pixels that are different between the two images.

Next, the system reads the text file of seat coordinates, which was also provided as an input file prior to execution. This file is used to create a two-dimensional array that reflects the classroom's layout. The subtracted image is converted to an array and sliced at each seat location. This creates a smaller array of pixel values representative of that single seat. For each seat location array, the pixel values are summed, and this sum is compared against a threshold value that was provided as an input. If the sum is larger than the threshold value, a value of 1 is populated into the classroom array at that seat's location denoting that the seat is occupied. Otherwise, a value of 0 is placed into the array to represent an empty seat.

Once the entire classroom array has been populated, the classroom occupancy detection system checks for adjacent occupied seats or adjacent values of 1 in the array. Based on the results of this, the appropriate LED is turned on and a message is displayed to the console. If the user has not chosen to terminate the process, the system captures a new photo of the classroom and this process repeats. The figure below details the algorithm process for the classroom occupancy detection system.

Figure 4.11

COVID-19 Classroom Occupancy Detection Algorithm Flow


Capture overhead image of the empty classroom
Capture overhead image of the current classroom
Subtract current classroom image and empty classroom image
Read text file of seat coordinates
Create 2D NumPy array to represent classroom
Check subtracted image at seat locations
Populate classroom array with Boolean values
Check classroom array for adjacent occupied seats
Adjacent occupied seats detected?
Yes
Turn off green LED and light red LED
No
Turn off red LED and light green LED

4.3 Flood Detection System Methodology

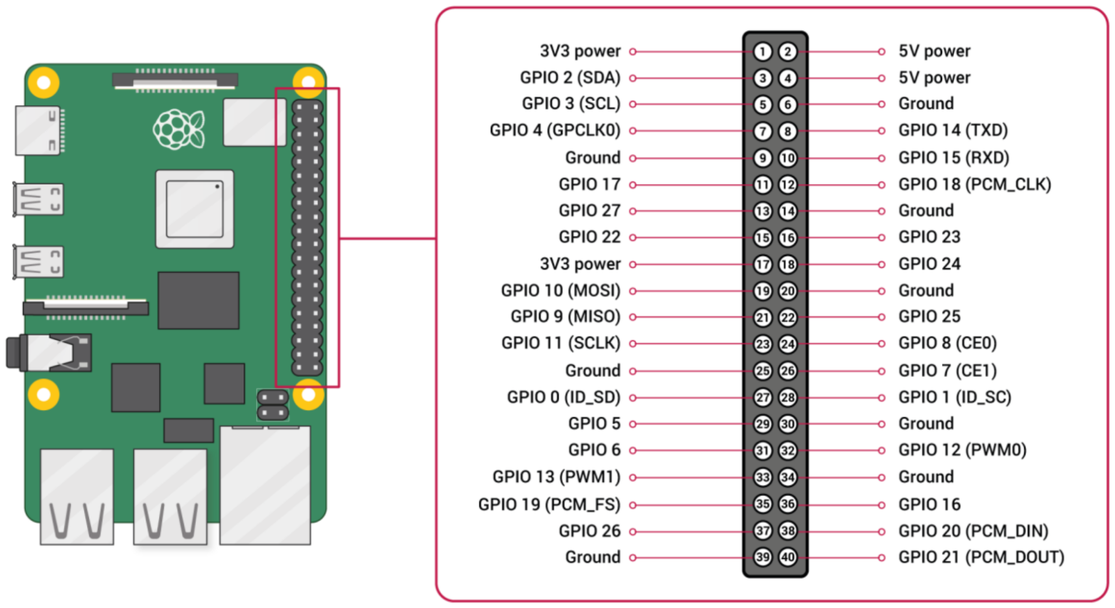
The Raspberry Pi 3 Model B with a compatible Camera Module is used to implement the flood detection project. The flood environment is achieved using a fish tank with a block of concrete inside to simulate a roadway. Water is then poured into the tank to simulate the flood water entering the environment. The Raspberry Pi Camera Module is placed approximately two feet away and is positioned to capture images of the front of the tank. System output is achieved with breadboard circuits to control the three LEDs used to communicate flood levels.

4.3.1 Hardware Configuration and SSH Connection

The Raspberry Pi's GPIO pins are used to control components on the breadboard. For this project, three different GPIO pins and a ground pin are used to design a circuit that can control three LEDs. Figure 4.12 shows the layout of the Raspberry Pi's GPIO pins.

Figure 4.12

Raspberry Pi GPIO Pin Layout



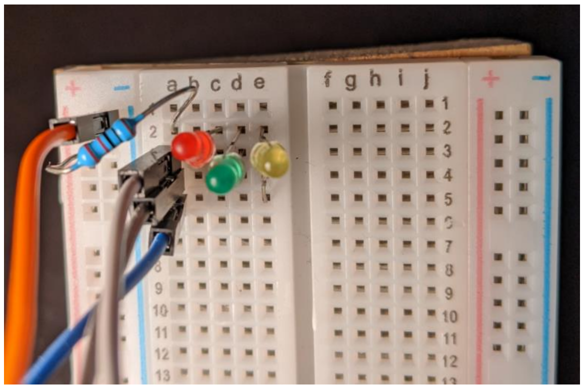
[Raspberry Pi GPIO Pinout Diagram]. (n.d.). https://learn.sparkfun.com/tutorials/introduction-to-the-raspberry-pi-gpio-and-physical-computing/gpio-pins-overview

Starting with one LED, a circuit is constructed by connecting a male-to-female jumper lead from the GPIO pin to the negative line. On the negative line, a resistor is connected in series on the breadboard. The negative end of a LED light is connected on the same row as the resistor, and the positive end is connected to its own row. On the same row as the positive end of the LED, a male-female jumper is connected to another GPIO pin. With this setup, the LED circuit is tested using a Python script that can control the diode. This step is repeated for the other two

LEDs. It should be noted that the positive side of each LED is connected to its own GPIO pin. Figure 4.13 shows the completed circuit used for the system.

Figure 4.13

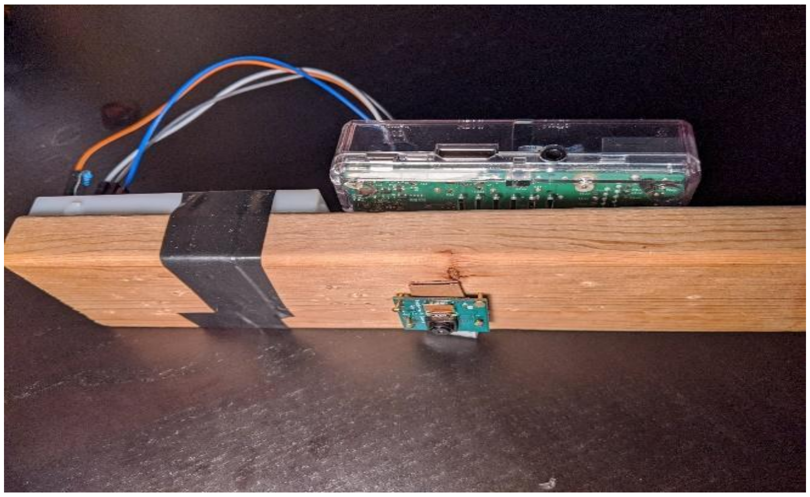
Breadboard with Three LED Circuits Connected



To provide image capturing functionality to the system, the Camera Module's ribbon cable is inserted into the camera port on the Raspberry Pi. Once connected, the camera is then enabled. This is accomplished by accessing the Preferences category on the Raspberry Pi, and then selecting the Raspberry Pi configuration menu. On the Interfaces tab, the Camera option should be enabled. To complete the hardware setup, the breadboard and camera are mounted to a block of wood. The purpose of mounting the device on a block of wood is to keep the components from shifting between photos and to ensure that the camera is stable while capturing photos. Figure 4.14 shows the completed device. The device is then placed at a two-foot distance from the tank.

Figure 4.14

Flood Detection Device



It is imperative that the Camera Module and its connected components maintain a stationary position during system execution. To send code and commands to the Pi, a Secure Shell (SSH) connection is used. Enabling SSH is achieved in a similar manner to enabling the camera. With the SSH connection active, an SSH client program like PuTTY can be used to access the Raspberry Pi's command line interface. Establishing this SSH connection allowed software development to be completed at a traditional workstation running a more sophisticated IDE than the one provided on the Rasbian OS.

4.3.2 Software Configuration and OpenCV

The first part of the program to be implemented was the code that controls the LEDs on the device. Figure 4.15 shows the tank as seen by the Pi Camera Module. It shows the water at the medium level and the light shining on the tank to represent the sun. The car is in the tank to simulate a car on a road that might be in danger depending on the flood level. The block of concrete is a simulated road.

Figure 4.15

The Tank at a Medium Water Level

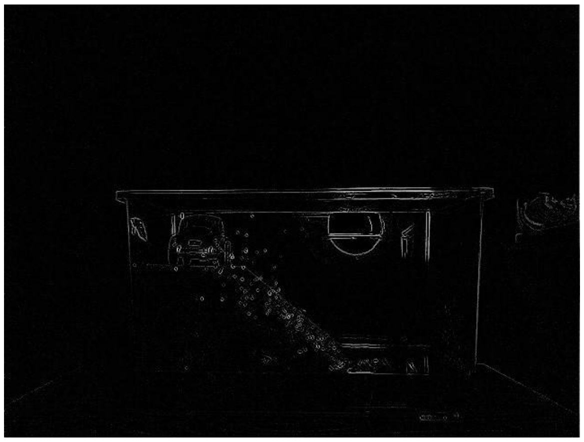
The OpenCV Python library is heavily utilized in the implementation of the flood detection system. OpenCV is an open-source computer vision library with a focus on real-time applications (OpenCV Team, 2021). After taking a picture of the tank, the image is saved and then opened for processing. It is converted to a black-and-white image before being further processed. During the processing step, an OpenCV filter is applied to the image to ensure that only the outlines of objects are depicted. This ensures that the water level within the tank is clearly visible. In all pictures, the lighting is enhanced to make the outlines clearer.

4.3.3 Selection of Edge Detection Filter

During development, various filters were tested, and their results were compared. The first method used was the Laplacian filter. This filter is an edge detector algorithm that computes the second derivative of an image and measures the change from the first derivative of the image (Bradski & Kaehler, 2008). Figure 4.16 depicts an image of the tank in low flood levels with the Laplacian filter applied.

Figure 4.16

Laplacian Filter with Low Water Level



The Laplacian filter removed a significant portion of the image's background. Additionally, the remaining outlines were not clear, and a substantial amount of noise remained. As such, this filter was not ideal for the system's goals.

The next method used was the Sobel filters. The Sobel filter calculates the approximate derivatives of the image: one for horizontal changes and one for vertical changes (Bradski & Kaehler, 2008). Using the rate of change from high to low image intensity at each pixel, the filter draws a new image based on the rate of change (Bradski & Kaehler, 2008). Within the OpenCV library, the Sobel filter can be applied to the x-axis or the y-axis. For testing purposes, both were combined in attempt to provide a clean outline of the tank. Figures 4.17, 4.18, and 4.19 depict Sobel filtered images of the tank in low water levels. In these images, the filter is applied to the x-axis, the y-axis, and both axes, respectively.

Figure 4.17

Sobel X-axis Filter with Low Water Level

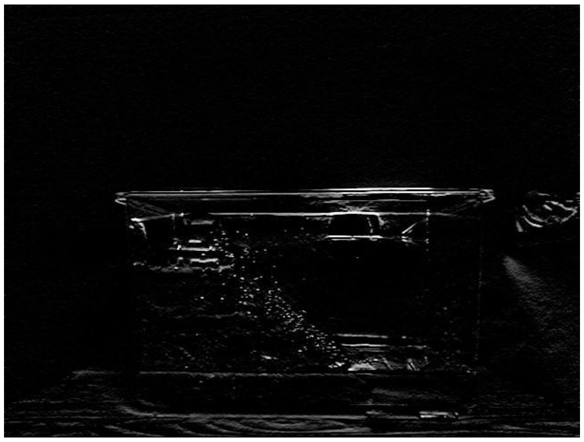


Figure 4.18

Sobel Y-axis Filter with Low Water Level

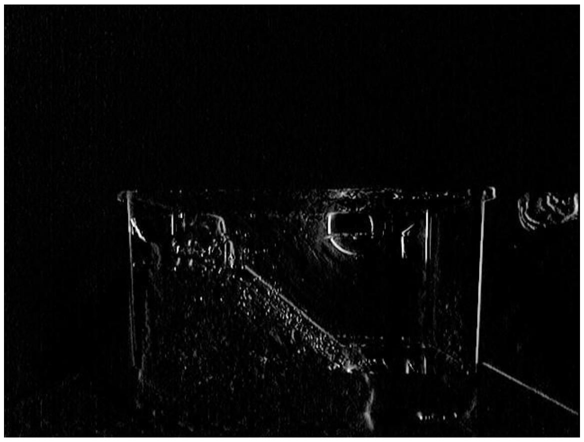
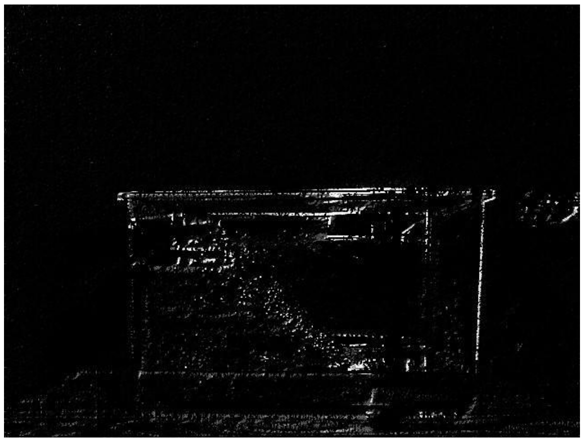


Figure 4.19

Sobel Combined X-axis and Y-axis Filter with Low Water Level



Of the three Sobel filters, the x-axis filter provides the clearest and most suitable images. With this filter, the water level was depicted as a clean horizontal line. However, it also produced a significant amount of noise around the water level line. Similarly, this filter removes much of the tank and the objects within it, making it difficult to understand the image's contents.

The last method tested was the Canny outline. The Canny outline is a multi-stage, edge detection algorithm. This algorithm removes the noise in the image, finds the intensity gradient of the image, removes unwanted pixels, and finally performs hysteresis thresholding to distinguish between edges and non-edges (The OpenCV Team, 2021). The Canny outline was tested on images of the tank at various water levels. This filter yielded images with bold lines and clearly defined water levels. Figures 4.20, 4.21, and 4.22 show the Canny outline filter applied to low, medium, and high water levels.

Figure 4.20

Canny Outline with Low Water Level

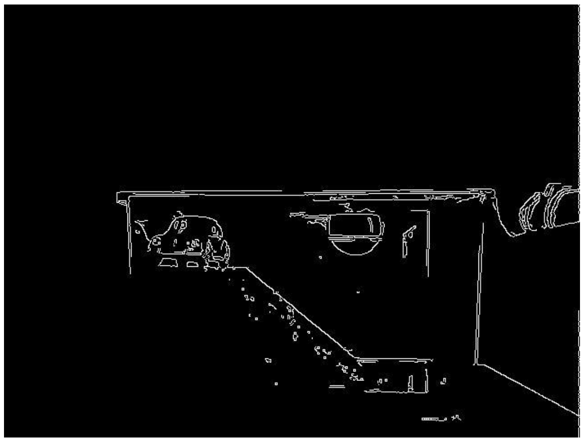


Figure 4.20

Canny Outline with Medium Water Level

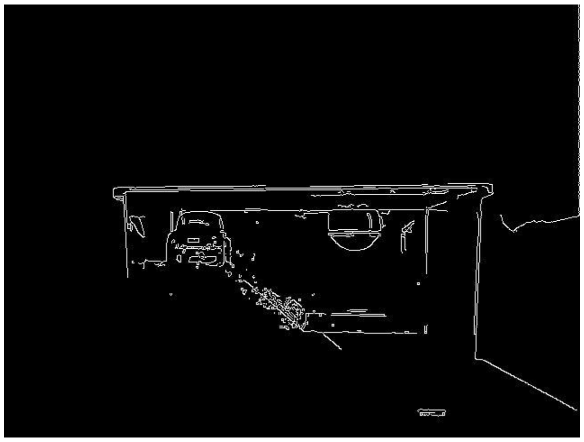
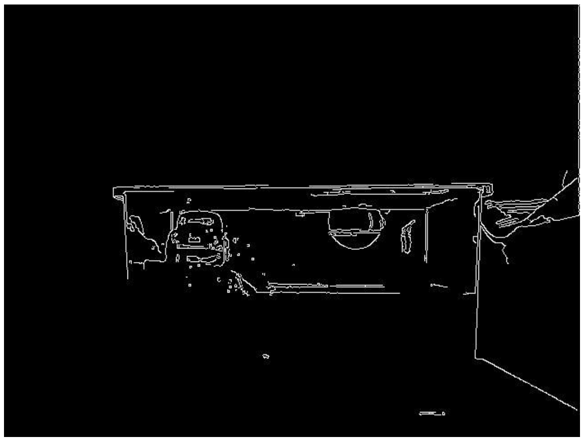


Figure 4.21

Canny Outline with High Water Level



The Canny outline clearly shows clean lines of objects in the picture, especially as compared to the Laplacian and Sobel filters. During the testing of these OpenCV filters, it was noted that all three filters were extremely sensitive to any drops of water on the tank and would outline these droplets as well. Given the solid, defined lines that it produced, the Canny outline was chosen for the flood detection system.

The system detects the water level by first cropping the image. The bottom of the cropped image corresponds to the bottom of the tank and has the value of 0. The top of the cropped image is where the road starts. With this cropped outline the system scans each vertical line looking for the first white pixel of the image and computing an average of these pixels as the scan continues. With this accomplished, the system returns that average value. If the average falls within the bottom third of the cropped image, the green LED will be illuminated, denoting that the water level in the image is significantly below street level. If the value is in the middle of the image, the yellow LED is turned on, as water levels are encroaching into the roads and are at risk of creating hazardous conditions. Finally, if the average falls within the top third, the red LED is illuminated, communicating that the current flood levels pose notable risk to drivers and their vehicles.

5. Results and Discussion

This section details the accuracy and performance of each system as observed during development and testing. Any constraints and limitations currently impacting a system's functionality are also detailed here.

5.1 Parking Lot Occupancy Controller Results

The parking lot occupancy controller system is capable of successfully determining if a parking space is open or occupied and subsequently determining the overall occupancy of the parking lot. Currently, the system's performance is impacted by the elapsed time between the initial capturing of the reference image and the capturing of the real-time image. The amount of natural light captured in both the reference image and the real-time image can cause discrepancies in the SAD values, thus affecting the system's detection capabilities. However, this issue can be mitigated by maintaining a static camera position between image captures. Since the system is centered around the coordinates of the parking spaces in the reference image, these coordinate positions must be consistent between the reference image and any images captured during system execution.

5.2 COVID-19 Classroom Occupancy Detection System Results

Utilizing the three-row, three-column simulated classroom environment described in Section 4.2.1, the classroom occupancy detection system was tested using various seating configurations. These test case configurations assess system performance in a classroom setting with no physical distancing violations; a setting with a single violation; and a setting with multiple violations. Furthermore, the test cases ran concurrently during a single instance of system execution to ensure that the system can adapt between acceptable and unacceptable conditions. For testing purposes, the NumPy array that represents the classroom's current seating layout is output to the console. This is to confirm that the system has produced an array that is reflective of the classroom image that was captured. Additionally, all images captured during system execution are provided to confirm accurate functionality. It should be noted that during real-world execution the captured images and the NumPy array would not be presented to the end-user.

The three test cases produced accurate LED outputs and the desired console messages. Additionally, the NumPy arrays associated with each test case accurately reflected the chosen seating configurations. Since each test case ran concurrently, the positioning of both the Camera Module and the simulated classroom seats were consistent. As such, each test case utilized the same set of seating coordinates. Much like the parking lot occupancy project, the performance of the classroom occupancy detection system is dependent upon fixed camera and seat positions. Changing either would impact detection accuracy and require an updated seat coordinate file. Though the system passed each test case, additional development and subsequent testing is needed to produce a flexible system capable of adjusting to changes in camera and seat positioning.

5.3 Flood Detection System Results

Overall, the flood detection system accurately detects water levels in the tank and turns on the correct LED. However, it should be noted the accuracy of this detection is dependent upon lighting and water clarity within the tank. During development, the system performed optimally when the camera was mounted on a flat surface, approximately three feet away from the tank. Furthermore, the tank itself needs to be placed in a well-lit environment to ensure that optimal images are captured during system performance. Given these constraints, decreases in detection performance and accuracy were observed as variables, such as lighting and camera distance, changed.

6. Conclusion

The parking lot controller system, COVID-19 classroom occupancy detection system, and flood detection system provide valuable research and problem-solving experience that can be applied in further undergraduate and graduate education as well as the professional workforce.

These projects present students with a topical task and ask them to design and implement a system. Their respective development processes provide exposure to image processing concepts and techniques, such as the Sum of Absolute Differences and edge detection filters, by encouraging students to research these topics and then apply them within their own system. In the course of implementing the solutions discussed in this paper, students gained practical experience interfacing with the Raspberry Pi and its components, constructing breadboard circuits, and establishing and using an SSH connection.

Students undertaking these research projects encounter various issues during system development. Consider the solutions previously discussed for the parking lot controller and classroom occupancy problems. One of the most significant challenges in implementing the classroom occupancy detection system involved the detection of seat locations. The initial method, which involved using edge detection to determine the seat coordinates without additional user input, proved to be ineffective, and, ultimately, an input file of seat coordinates was incorporated into the system. A notable issue that arose during development of the parking lot controller concerned quantifying parking space occupancy. Early stages of this system attempted to address this by computing an average of the parking space's pixels and then subtracting the real-time image's average from the reference image's average. However, the averaging of pixels proved not to be a scalable technique and resulted in inaccurate system performance. Ultimately, space occupancy was quantified using the Sum of Absolute Differences technique discussed in this project's methodology section. In the case of both projects, reaching the final implementation from their initial methodology entailed several iterations of research, faculty and graduate student feedback, and testing. Given the iterative nature of software development practices like Agile, undergraduate students can utilize this practical experience in future software engineering roles and courses.

In addition to the broad technical skills that are developed and refined throughout the research project, students can also gain a foundational understanding of specific domain topics. The flood detection project, for example, provided a basic understanding of Hampton Roads' geographic features and associated flood risks. Likewise, the parking lot occupancy and classroom detection projects gave undergraduate students a domain-specific knowledge. These skills and background knowledge can prepare undergraduate students for a wide range of future work and may be especially useful for students wishing to use programming in public infrastructure applications.

7. Future Work

Though each project was successfully completed and implemented, limitations are present in each resulting system. These limitations present opportunities for future work by undergraduate students. This section details recommendations to improve these respective systems and to expand the projects for future undergraduate study.

7.1 Parking Lot Occupancy Controller

Rather than simply capturing and processing static, top-down images of a lot, the parking lot occupancy project can be expanded to analyze a livestream of the parking lot. In addition to changing how the system receives the input information, such a change could expand on the methods used to analyze and process the input data. Using object detection methods, undergraduate students could develop a system capable of detecting vehicles entering the lot, occupying a space, and exiting the lot. Instead of manually inputting coordinates to isolate an individual parking space, a method for recognizing the spaces without user interaction could be explored which would make the system autonomous aside from physical system maintenance.

7.2 COVID-19 Classroom Occupancy Detection System

In its current state, the COVID-19 classroom occupancy detection system is designed to operate in a completely stationary environment. This implementation is dependent upon a text file of seat coordinates that informs the system where it should check for an occupied seat. As such, if the layout of the classroom were to change, the system would be unable to respond to these changes in real time. Both an updated image of the empty classroom and an updated text file of seat coordinates would need to be provided for the system to function accurately.

Additionally, the current system implementation has not been tested on a real-world classroom environment. The system's current design does not account for student behaviors or personal belongings and other inanimate objects that would realistically impact functionality. For example, the system would not be able to distinguish between a student sitting in a seat and a bag or other inanimate object that has been placed in a seat. Both of these events would result in the detection of an occupied seat by the system.

Addressing these limitations would not only result in a more robust system appropriate for real-world classroom environments, it would also provide additional challenges for undergraduate students that wish to gain more experience. Machine learning image recognition tools, such as an object detection model or an image labeling model, could be used to identify and differentiate between humans and inanimate objects in the classroom image. By using a trained object detection model that can identify desks and chairs and determine the location of these objects, the system would be able to check a seat's location for a student without the use of an input text file. Image labeling or image classification functionality would allow the classroom occupancy detection system to label non-human objects that have entered a seat's location and ignore them, ensuring that the seat isn't incorrectly marked as occupied. These proposed solutions would give undergraduate students the opportunity to work with machine learning tools or even train their own machine learning model. This would provide valuable experience that the student could utilize in later coursework, other research projects, or in the workplace.

7.3 Flood Detection System

The flood detection project can be expanded to include live detection of water levels. Students could slowly fill the tank and watch the LED circuit output change in real time according to the water levels. The project requirements could also be adapted to place more emphasis on the SSH connection. This would teach students about computer networking concepts and give them experience establishing remote connections between machines. Enabling remote connectivity would also make the system more viable in real-life cases. In real-world applications, the flood detection device would be placed in areas with increased flood risks, thereby limiting physical access to the system, especially during inclement weather. The SSH connection would allow the system's operators and engineers to remotely implement system updates. As such, the project would be more realistic and more robust if students were required to connect to the Raspberry Pi via SSH connection rather than the computer's HDMI port.

REFERENCES

Barrett, D. J., Byrnes, R. G., & Silverman, R. E. (2005). *SSH, the Secure Shell the Definitive Guide*. O'Reilly Media, Inc. Web.

Bradski, G., & Kaehler, A. (2008). *Learning OpenCV*. O'Reilly Media, Inc. Web

CARFAX. (2021, September 16). *Free Flood Check® & How to Avoid Flooded Cars*. Carfax. Retrieved January 9, 2022, from https://www.carfax.com/press/resources/flooded-cars

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 Questions*. Centers for Disease Control and Prevention. https://www.cdc.gov/coronavirus/2019-ncov/faq.html#Basics

Clark, A. (2021, September 2). *Pillow (PIL Fork) Documentation Release 8.3.2*. Pillow (PIL Fork). https://pillow.readthedocs.io/en/stable/

Coutu, P. (2020, February 9). *Sea level rise continues to accelerate; Hampton Roads should prepare for 1.7 feet by 2050, report says*. The Virginian-Pilot. Retrieved January 9, 2022, from https://www.pilotonline.com/news/environment/vp-nw-sea-level-rise-hampton-roads-20200209-gskdls6j7rafbdvopxqzdnexxu-story.html

First Street Foundation, Inc. (2020). *The First National Flood Risk Assessment: Defining America's Growing Risk*. First Street Foundation. https://assets.firststreet.org/uploads/2020/06/first_street_foundation__first_national_flood_risk_assessment.pdf

Hampton Roads Chamber. (n.d.). *Hampton Roads*. Hampton Roads Chamber. Retrieved January 9, 2022, from https://www.hrchamber.com/page/hampton-roads/

Helber, S. (2021, April 17). *Virginia Beach ranks top for cities at serious risk of flooding in Virginia*. WTKR. Retrieved January 9, 2022, from https://www.wtkr.com/news/virginia-beach-ranks-top-for-cities-at-serious-risk-of-flooding-in-virginia

Hunter, J., Dale, D., Firing, E., Droettboom, M., & The Matplotlib Development Team. (2021, August 13). *Matplotlib Release 3.4.3*. Matplotlib. https://matplotlib.org/stable/contents.html

INRIX, Inc. (2017). *New INRIX Study Finds Parking is the Largest Cost of Driving*. INRIX. Retrieved January 4, 2022, from https://inrix.com/press-releases/cod-us/

INRIX, Inc. (2017, July 12). *Searching for Parking Costs Americans $73 Billion a Year*. INRIX. Retrieved January 4, 2022, from https://inrix.com/press-releases/parking-pain-us/

Jones, D. (2018, November 3). *Picamera 1.13 Documentation*. Picamera. https://picamera.readthedocs.io/en/release-1.13/

National Center for Environmental Health. (2020, March 27). *Dangers of flooding and tips for how you can protect yourself | Environmental Health Toolkits | NCEH*. CDC. Retrieved January 9, 2022, from https://www.cdc.gov/nceh/toolkits/floods/default.html

National Center for Environmental Health. (2020, December 21). *Precipitation Extremes: Heavy Rainfall, Flooding, and Droughts*. CDC. Retrieved January 9, 2022, from https://www.cdc.gov/climateandhealth/effects/precipitation_extremes.htm

National Weather Service & National Oceanic and Atmospheric Association. (n.d.). *+Turn Around Don't Drown® Warning Signs*. National Weather Service. Retrieved January 9, 2022, from https://www.weather.gov/safety/flood-turn-around-dont-drown

NOAA Office for Coastal Management. (n.d.). *Hampton Roads' Sea Level Rise Adaptation Advances on Multiple Fronts*. NOAA Office for Coastal Management. Retrieved January 9, 2022, from https://coast.noaa.gov/states/stories/sea-level-rise-adaptation-advances-on-multiple-fronts.html

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

Nuttall, B. (2021, March 18). *GPIO Zero Documentation*. GPIO Zero. https://gpiozero.readthedocs.io/en/stable/

OpenCV Team. (2021, July 19). *OpenCV Modules 4.5.3*. OpenCV. https://docs.opencv.org/4.5.3/index.html

OpenCV Team. (2021, September 9). *OpenCV: Canny Edge Detection*. OpenCV. https://docs.opencv.org/3.4/da/d22/tutorial_py_canny.html

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

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

Sakiotis, I. (2021, April). *Parking Lot/Classroom Occupancy Detection Using Raspberry Pi* [PowerPoint slides].

Siddique, M. N. (2021, April). *Flood Detection using Raspberry Pi* [PowerPoint slides].

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

SparkFun Electronics & Blom, J. (n.d.). *Polarity*. SparkFun. https://learn.sparkfun.com/tutorials/polarity/all

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