People Counter: Count, Estimate, and Predict Occupancy of Rooms in Hallway

David Feng, Tianzhuo Li, Gary Qin

Department of Electrical and Computer Engineering, Carnegie Mellon University

***Abstract*—A system capable of estimating and predicting the occupancy of lab spaces at Carnegie Mellon's Hamerschlag 1300 wing. People Counter helps users save travel time by presenting a real-time estimation of room occupancy and an interactive element that predicts ahead of time whether the spaces will be busy. The system combines video feed processing, computer vision, as well as a web application deployed on server; For both estimation and prediction, People Counter surpassed the 80% accuracy benchmark, and we have shown that it can deliver up-to-date occupancy levels and useful prediction data for users.**

***Index Terms*— Arducam, classification, computer vision, Google Colaboratory, Ngrok, web application, object detection, prediction, Raspberry Pi, room occupancy, people movement, video processing**

I. Introduction

Several travel-related applications such as Google Maps and the New York MTA TrainTime mobile app have evolved in recent years to introduce real-time capacity tracking and estimation features. However, these applications currently do not combine the usage of historical data and real-time video processing techniques to make accurate predictions. Google Maps' prediction algorithm relies on users' historical location data [1], while the TrainTime mobile application is only able to show real-time occupancy data of the train cars [2] and does not have the capability to let users know whether their train is going to be packed ahead of time.

Thus, we propose People Counter, a system that uses both live camera feed to CV processing as well as analysis of historical data to count, estimate, and – most importantly – predict the occupancy levels of the Hamerschlag Hall 1300 hallway at Carnegie Mellon University. The system is an amalgamation of computer software and digital hardware technologies, as the live video feed will be captured by Arducam B0205 cameras and fed into the local backend through wireless connection. Then, a backend program that is able to recognize a person passing through a door for both entering and exiting scenarios through computer vision algorithms, namely Yolo and SORT, will be run on Google Colab, a cloud platform for interactive computing. The resulting data is stored on a ElephantSQL database management system in the cloud, and at last the occupancy and prediction data will be connected to a frontend web user interface through Django's model-view-controller (MVC) design pattern.

People Counter's intended users are students enrolled at Carnegie Mellon University who would like to learn about the current and predicted occupancy data of the 1300 hallway at Hamerschlag, which is among the most popular study spots for students in the Electrical & Computer Engineering department. Through our observation over the past several years of students who enter the hallway, many students would struggle to find a seat when the labs are in session or when the rooms are busy. They tend to quickly exit when no seats are available. Over time, the time wasted walking to and from study spaces like the 1300 hallway has become a major pain point for students. That is why we believe People Counter has the capability to solve this user pain point.

By interacting with People Counter's web application, users will not only be able to know the accurate current occupancy data of the Hamerschlag 1300 hallway but also obtain the predicted occupancy of the two individual lab spaces in the Hamerschlag 1300 wing at a time of the user's choosing up to 1 hour in advance. With this information, users will be able to make a better-informed decision about whether they would like to go study at the Hamerschlag labs. The project's ultimate goal is for our users to save valuable time during their day that they would otherwise spend on traveling to and from Hamerschlag without the data from People Counter, only to find a hallway packed with studious individuals at times.

II. Use-Case Requirements

We formulated the use-case requirements for People Counter based on the needs of our intended users. One of the critical use-case requirements for our system is to have a greater than 80% occupancy estimation accuracy for the two enclosed lab spaces on the Hamerschlag 1300 wing. The reason behind the 80% benchmark is due to the fact that the larger of the two lab rooms has a capacity of 50 people, while the smaller one only has a maximum occupancy of 25. The 80% benchmark would allow for a maximum of 5-people margin for error in the smaller lab space which would suffice the needs for our target users, as a difference in occupancy of 2 or 3 individuals would in general not affect a user's decision of whether or not they would want to spend time studying at the 1300 wing. However, if the estimation accuracy metric is set at a much lower mark, such as 50%, then it would not be sufficient for our use case, because the difference between 10 and 20 people in a room with 25 seats in total is significant for users who hope to find a less busy, relatively quiet place to do work. Additionally, the 80% benchmark must be reached consistently for at least 3 hours without dropping below the threshold, which is the amount of time that we believe constitute a middle ground between the maximum amount of time we are allowed to perform continuous testing and the minimum amount of time needed to show that our system is adequately consistent and accurate.

Other than real-time estimation, another key feature of the People Counter system is same-day occupancy prediction for the two lab rooms on the Hamerschlag 1300 wing. We plan to implement our prediction feature through categorization, characterizing occupancy levels of the rooms into 4 categories: “almost empty” (up to 19.9% full), “not busy” (20%-39.9% full), “busy” (40%-69.9% full), and “almost full” (70% full or more). The prediction feature is limited to 1 hour in advance and provided to the users in 15-minute intervals, so if the current time is 5:00PM, users will be able to view the predicted category of the rooms when the time is 5:15, 5:30, 5:45, and 6:00 on the web application. The aforementioned categorization and the percentage of capacity associated with each category are not set arbitrarily. Both lab rooms on the 1300 wings at Hamerschlag do not have individual, separate seating areas but rather feature contiguous work benches intended for better collaboration. If there are more than 70% of the total seats taken in one of the lab spaces, this would mean that the room is effectively almost full, because there will rarely be two consecutive occupied seats (having both neighboring seats taken is less ideal for individual studying) [3] when total occupancy is above 70%. On the other hand, when the occupancy levels of a room are below 40%, then it is guaranteed that there will be at least 1 seat in the room where there is at least 1 unoccupied neighboring seat. This would warrant a “not busy” categorization. Similar to the live estimation benchmark, we set the prediction accuracy to 80%, which means that when a user selects any time later during a particular day, the predicted occupancy categorization must exactly match the actual percentage of the rooms at the chosen time more than 80% of the time over a 15-hour period.

Lastly, another use-case requirement that, if not achieved, would impact the usability of our application is the overall latency of the system. Ideally, the latency from hardware video processing to end users shall be less than 1 minute (60 seconds) at any given time when the server of People Counter’s web application is running. Given the nature of people movement (moving in and out of a room on foot takes at least several seconds), we do not expect any non-malicious user to be checking People Counter’s web application in an interval less than 60 seconds for the up-to-date information regarding real-time occupancy estimation. However, given that occupancy in a room, especially multi-use spaces like the Hamerschlag 1300 labs, can shift swiftly (such as the start and end of class sessions), it could potentially be beneficial to users if the total latency of our system is consistently kept at under 60 seconds.

TABLE I. SUMMARY OF USE-CASE REQUIREMENTS

Name	Specification	Requirement
Estimation Accuracy	Users will receive via the web application an estimate of how many people are currently in the HH1300 rooms that is at least 80% accurate (i.e., a margin of error that is less than or equal to 20% from the actual number of people in the rooms. If there are in fact 50 people in total, People Counter’s estimate should fall between 40 and 60.) The 80% benchmark shall be reached continuously for at least 3 hours.	**80%**
Prediction Accuracy	Users will receive a categorical prediction out of the 4 possible values (almost empty, not busy, busy, almost full) for a time that is at most 60 minutes in the future from the current time (the 4 times displayed are 15, 30, 45, and 60 minutes from current time). We expect that the category users receive for all of these times falls into the accurate category 80% of the time.	**80%**
Latency	Users will receive the up-to-date occupancy estimation on the web application that has a maximum lag of 60 seconds (1 minute) whenever a change in the actual occupancy of the rooms occurs.	**60 seconds**

III. Architecture and/or Principle of Operation

Our product contains 4 main sub-systems: camera module, computer vision module, prediction module, and web application. Our camera module consists of two ArduCAM B0205 cameras, each connected to a Raspberry Pi, which will send live video feed to our backend through Ngrok web tunnel running on the Raspberry Pi. The camera set-up is shown in figure 1 on page 4. Since the design review, we added another camera into our camera module in order to improve accuracy of our computer vision module, we also started using Ngrok to transmit video footage in order to access the video feed on Google Colaboratory. Our computer vision module is deployed on Google Colab, which uses object detection and object tracking to count the number of people entering and exiting the four monitored rooms in Hamerschlag 1300 Wing. We update the count data related to each room every minute to a database hosted by ElephantSQL. Since the design review, we moved our computer vision module to Google Colab in order to improve processing speeds to meet our FPS requirements. We also started using a web hosted database instead of a local database in order to communicate data from the computer vision subsystem to our prediction module and web application.

The prediction module uses a decision tree, which we trained on prior occupancy data for rooms in Hamerschlag 1300 Wing. We can predict future occupancy levels for the four monitored rooms within 1 hour based on the current time and occupancy. The web application uses the Django framework, and gets occupancy data from the database, it has access to the trained decision tree to get the predicted occupancy levels. CMU users can register for an account on the web app and access real-time occupancy and future occupancy data on the web application. Figure 2 on page 4 shows the detailed submodules within each subsystem. Please note an image of our physical system will be included in the Design Requirements Section.

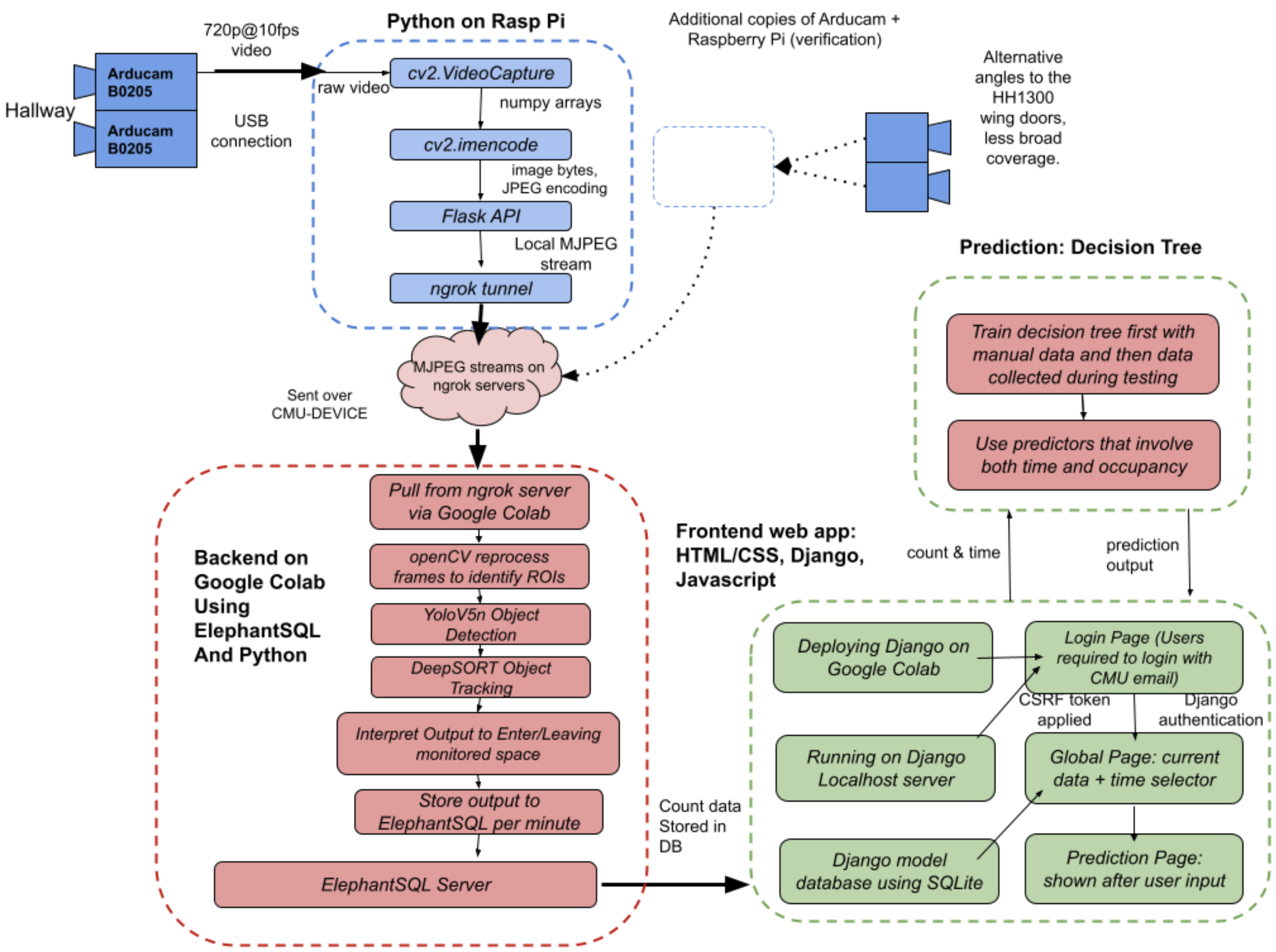

Hallway
Arducam B0205
Arducam B0205
720p@10fps video
USB connection
Python on Rasp Pi
raw video
cv2.VideoCapture
numpy arrays
cv2.imencode
image bytes, JPEG encoding
Flask API
Local MJPEG stream
ngrok tunnel
Additional copies of Arducam + Raspberry Pi (verification)
Alternative angles to the HH1300 wing doors, less broad coverage.
MJPEG streams on ngrok servers
Sent over CMU-DEVICE
Prediction: Decision Tree
Train decision tree first with manual data and then data collected during testing
Use predictors that involve both time and occupancy
Backend on Google Colab Using ElephantSQL And Python
Pull from ngrok server via Google Colab
openCV reprocess frames to identify ROIs
YoloV5n Object Detection
DeepSORT Object Tracking
Interpret Output to Enter/Leaving monitored space
Store output to ElephantSQL per minute
ElephantSQL Server
Count data Stored in DB
Frontend web app: HTML/CSS, Django, Javascript
count & time
prediction output
Deploying Django on Google Colab
Login Page (Users required to login with CMU email)
CSRF token applied
Django authentication
Running on Django Localhost server
Global Page: current data + time selector
Django model database using SQLite
Prediction Page: shown after user input

Fig. 1. Overall system block diagram

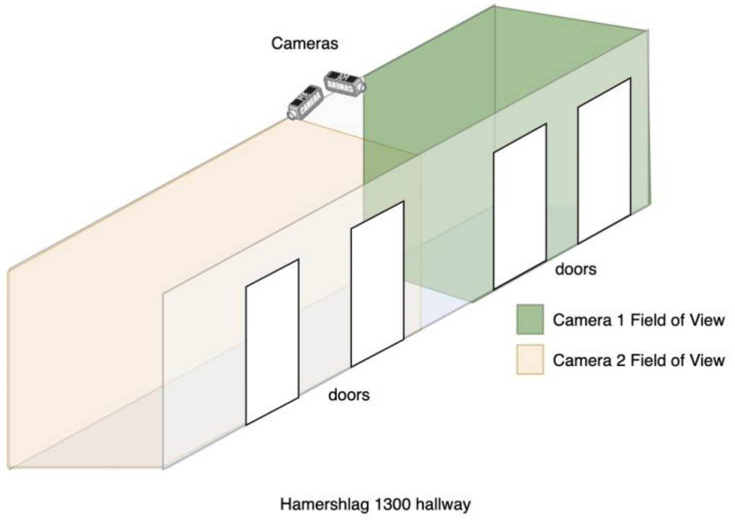


Fig. 2. Approximate location and setup of main cameras in HH1300 hallway

IV. Design Requirements

The primary goal of our design requirements is to ensure that the system we develop meets the estimation and prediction accuracy as well as the overall latency requirements previously mentioned for our use case of the Hamerschlag 1300 hallway.

A. Number of Cameras

In order to maximize the accuracy of the system as well as other use-case requirements, we have increased the scale and capability of our system after the design review stage to incorporate 4 total cameras. The final system that we build shall contain 2 main cameras (of which we will be measuring the estimation accuracy) and 2 validation cameras that will be used to test whether the output from the main cameras are consistent. Given that the validation cameras will have a field of view that is different from the main cameras, their purpose is to validate the flow of people in and out of the rooms in the HH1300 wing. During testing, the validation cameras and the main cameras will run simultaneously. After each test, the database file from the validation cameras will be matched with that of the main cameras through a script written in Python to compare the outputs from each set of cameras.

B. Location of Cameras

To image the Hamerschlag corridor completely, we will utilize a vision system that comprises a single camera placed directly above the main entrance way of the hallway. The camera will face the main hallway, with a slightly angled downward view. The purpose of positioning the camera at this location is to capture a single image that encompasses all room entrances, enabling our computer vision systems to receive as much information as possible. This approach aims to maximize the accuracy of our estimation and prediction systems though this maximized coverage. Fig. 3 shows all four cameras being installed in the HH1300 Hallway on three wood poles. From near to far on the right-hand side of the image: Wood pole with 2 main cameras facing opposite sides of the hallway as detailed in Fig. 2, validation camera on wood pole #2, and the other validation camera on wood pole #3.

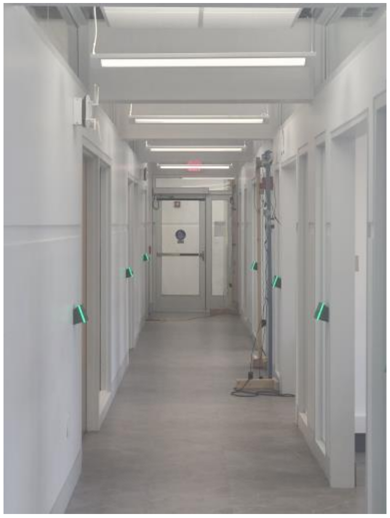


Fig. 3. Camera on wood poles located in Hamerschlag 1300 hallway

C. Camera Resolution

We have determined that the minimum resolution necessary for our computer vision algorithms to accurately estimate occupancy is 1280 by 720p. To arrive at this conclusion, we captured sample images at various resolutions using the Arducam camera, from the angle at which we plan to mount the camera during testing. We then utilized our Yolov5 detection algorithm to evaluate whether people were correctly detected in the varying resolution images. We observed that the accuracy of our detection algorithm began to decrease below a resolution of 720p, which led us to select this resolution for our computer vision pipeline. We also decided against using a higher resolution in order to reduce the Wi-Fi bandwidth requirements between the Raspberry Pi and the laptop running our CV system, so that we can consistently meet our latency requirement of 60 seconds.

D. Camera Frame Rate

To minimize the burden on our computer vision pipeline, we have decided to transmit our live video feed at a frame rate of 10 frames per second. If the frame rate were too high, it would be difficult for our local computation of the object detection and tracking on our laptop to keep up with the incoming frames. We believe that a frame rate of 10 fps is sufficient for our use case of the Hamerschlag 1300 hallway, as this is an indoor environment where pedestrians are unlikely to run at high speeds in a narrow hallway. We have determined that 10 frames per second is the minimum required frame rate for our vision pipeline to accurately estimate occupancy, as it is a low framerate where our DeepSORT object tracking algorithm can still follow walking people between frames, in order to meet our requirement of 80% estimation accuracy.

V. Design Trade Studies

During the design phase of our project, we took into account the implementation complexity, the fulfillment of our use-case and design requirements, and the practicality for our system when making design decisions. Our aim was to strike a balance between these factors with our design choices.

A. Number of Cameras

Initially, we designed our imaging system with only a single camera to cover all the doors in the Hamerschlag 1300 wing. However, after some rounds of testing, we shifted our design to use two cameras in conjunction instead, due to multiple reasons. Firstly, it was found that a singular camera was unable to sufficiently track all the doorways in the Hamerschlag 1300 wing. The addition of a second camera angle was critical in reducing the effects of occlusion between pedestrians, and allowed for much greater flexibility in camera positioning now that the coverage of doorways can be split between the two cameras. Secondly, it was discovered that the incorporation of a second camera was not as difficult to implement and synchronize as originally estimated. Due to the migration of the CV processing to the cloud, it became much easier to coordinate the multiple video streams from a single program, and to post our estimation results from the two cameras asynchronously to a shared cloud database. Overall, the inclusion of a second camera did not overly delay our development process, nor significantly spike latency issues. Thus, we have decided to revise our system to use two cameras, ensuring that we can still meet our use-case requirements consistently.

B. Camera Model

We selected the Arducam B0205 as the camera model for our imaging system. The specific camera model used in the imaging system is not crucial, as long as it fulfills our design requirements of covering the vertical span of the Hammerschlag hallway and exporting video at a minimum resolution of 1280 x 720p and a framerate of 10 FPS. We ultimately opted for the Arducam B0205 since it met these design requirements and had a small form factor that made it easy to mount during testing. Additionally, it could connect via USB to the Raspberry Pi, simplifying integration of the camera and microcontroller.

C. Microcontroller

When it came to choosing a microcontroller for our imaging system, we weighed up the options of the Raspberry Pi and Jetson Nano. Ultimately, we opted for the Raspberry Pi due to several reasons. Firstly, we felt that the extra capabilities provided by the GPU on the Jetson Nano would be unnecessary for our imaging system. The main function of the microcontroller in our design is to transmit live video from a connected camera over Wi-Fi, where the actual computer vision algorithms are executed. The video processing done on the microcontroller is minimal, involving only sending our video feed to the web application, thus negating the need for the processing capabilities of a GPU. Furthermore, the design requirements of our video pipeline are relatively low, with only a 720p resolution and a 10FPS framerate, further reducing the processing requirements of the microcontroller. Lastly, our team members were more familiar with the Raspberry Pi, having used the microcontroller before in other projects. As such, continuing to use the Raspberry Pi for our project would significantly reduce implementation time for our imaging system. For all these reasons, we then decided to select the Raspberry Pi for our microcontroller.

D. Object Detection Algorithm

We decided to choose DeepSORT tracking algorithm due to its ability to handle occlusion of objects and prevention of re-id of tracked objects by running feature extraction of tracked person [8], as well as the multiple object tracking capabilities, as we expect to track multiple people walking in the hallway. Having an accurate tracking algorithm is very important to achieving our accuracy requirement of 80% prediction accuracy. This is because if our tracker cannot track each person appearing in the video with high accuracy, there would be compounding error when we try to interpret the results of our CV system and result in compounding error in our estimations. Therefore, we have chosen to use DeepSORT as our object tracking algorithm. However, one drawback is that in order to perform feature matching on tracked objects, DeepSORT runs a pretrained CNN, which could lead to increasing computational complexity, potentially leading to failing our latency requirements. As mentioned in the previous section on Object Detection model, after switching to a smaller detection model and deploying our computer vision module onto Google Colab, we were able to meet our processing latency requirements of at least 10 frames per second. During initial testing, we also explored other tracking algorithms such as the MedianFlow tracking algorithm, which is a Lucas-Kanade based algorithm built into the OpenCV library. However, while the processing speed was faster than DeepSORT, it was less accurate with more frequent ID switching, causing cascading errors in counting accuracy, therefore we eventually chose to use DeepSORT as our tracking algorithm despite higher latency (since we were still able to meet our latency requirements).

E. Location of Computer Vision Processing

During our design phase for the project, we initially decided against the possibility of performing object detection and tracking on the cloud, mainly due to concerns over high latency and cost. However, after switching to two cameras for the main vision system, our local laptop was unable to handle the increased load from double the amount of input frames, causing the system to miss our original 60 second latency target. As a result, transferring our computer vision processing program from a local laptop to the cloud became necessary.

When we first thought about performing our CV tasks on the cloud, we thought about using the Google Cloud Platform, but we found that it lacked the necessary video processing modules and was not supported by our CMU budget. Eventually, we discovered a workaround that did not involve using GCP. Instead, we used the Flask API for Python and ngrok to redirect a web stream of the camera feed from the Raspberry Pi to a VM running in Google Colab for processing. During testing, we measured the latency of our system and found that, contrary to our initial assumptions, we were able to reliably meet our 60-second latency requirement. This was due to the drastically improved performance provided by the GPUs on Google Colab, and the protection against network variance from the integrated video buffer in the Flask API. Since the benefits of computing on the cloud now outweighed the initial costs of moving from local computation, we decided to continue with our web tunnel setup throughout the remainder of our project.

F. Prediction Algorithm

While there were many available machine learning classification algorithms for us to use, we decided that in order to meet People Counter’s use-case requirements, it was ideal to apply decision tree classification. In general, for decision tree algorithms, there was no need to produce real valued outputs, as prediction only needs categories for occupancy. Over a short development cycle (14 weeks), we could not gather enough data to train a vastly more complex classification system such as neural networks. Additionally, given our 60-second latency requirement, we need to be able to predict in a relatively short amount of time, as other parts of our system already require a large number of computational resources. Furthermore, decision tree classification supports multiple categories during classification, making our use-case requirement a suitable goal. We believe that occupancy is highly correlated to time-based attributes such as time of the day, day in the week, classes currently being held in the classrooms, and these are usable features for the decision tree to learn in order to accurately predict and output a category-based result.

VI. System Implementation

A. Camera Module

The architecture of our hardware components mainly consists of the Arducam camera module and our Raspberry Pi microcontroller. The configuration of the video stream from the camera will be done in by the Python program, with OpenCV setting the video connection to have a resolution of 1280 x 720p. OpenCV will also be responsible for controlling the framerate of the video pipeline, pulling new frames from the camera at a rate 10 frames per second. These cameras and microcontrollers were mounted on wooden poles at angles that would provide the clearest images possible for later computer vision processing, with each camera pointing down at a slightly lowered angle. During our test runs, we calibrated the position of these wooden frames to ensure that the pedestrians' images were clear and not overlapping or going out of focus.

After capturing the individual frames using Python, OpenCV encodes the frames into JPEG format before posting the frames to a web application. The original raw format of the images stored by OpenCV is not suitable for transmitting over a standard Wi-Fi connection due to its storage inefficiency. Therefore, the images are converted into JPEG format to reduce the bandwidth requirements for the CMU Wi-Fi. The recording program passes the video frames to the Flask API in Python, which allows for the video to be available via HTTP request. The video stream is provided as a continuous MJPEG buffer, from which the receiving end can segment to obtain the individual JPEG frames and perform the necessary processing. Due to only being able to run Google Colab programs within a VM, a web tunnel was necessary in order to facilitate communication between the Flask server on the Raspberry Pi and the Google Colab instance. To achieve this, Ngrok is used to direct our web server with the camera feed from localhost on the Raspberry Pi, over the CMU-DEVICE network, to a generated URL accessible from any device, including the VM on Google Colab.

B. Computer Vision Module

There are three main parts to our computer vision module: Object Detection, Object Tracking, and translating tracks into occupancy data. We set up our camera in a way such that we did not need to define the region of interest, and each frame from the video feed gets resized in our Camera Module to the correct dimensions. Thus, we could simply fetch the frames sent through Ngrok web tunnel in our computer vision module and process it with our object detection model.

Before we could process the video with object detection and tracking models, we needed to fetch the video from Ngrok web tunnel. This was done through starting background threads in our backend python code in Google Colab. These threads will continuously fetch byte data of frames from the Ngrok tunnel URL by using GET requests and reconstruct the image using OpenCV's function "imdecode". This part of our code was inspired by Han's project [4]. After fetching the data, we will process each frame with our object detection model.

We chose to use YOLOv5n as our object detection model. YOLO is an object detection model that is pre-trained on the COCO dataset, with the capability to detect 80 different types of objects, including people. At first, we ran the YOLOv5n model on our local computers to get a grasp on the detection accuracy. From the footage we collected, we determined that the pretrained model was detecting people well enough as under most cases with little occlusion, the model was able to detect people and draw bounding boxes around them accurately. We used pytorch to load the YOLO model in our python backend, as it is the library that YOLOv5 was built to support. After running each frame through our object detection model, we will store the bounding boxes of people detected in the frame with confidence level above a threshold. We decided to set a 40 % confidence threshold in our model as after testing detection with different confidence levels on images, we found that 40% gives little false positives while still picking up people walking in the hallway even when they are partially occluded (which often happens when they are entering/exiting doors). The stored bounding box information for each frame gets passed into the object tracker in order to track the movement of each person being detected in the frame.

After we have found the bounding boxes for people in each frame, we pass the frames into a multi-object tracking algorithm to track the movement of each person over multiple frames. We used a tracking algorithm called DeepSORT, which is available as a python library called "deep-sort-realtime". In our computer vision module running on Google Colab, we initialized two instances of the tracking algorithm for each of the cameras, as each tracker needs to keep states specific to the video feed of each camera to track the movement of people. The algorithm continuously takes in bounding box coordinates and the input frame, and outputs a list of tracks in the form of bounding boxes corresponding to each person tracked in the current frame, as well as a unique track id to identify each person that has appeared in the video. The figure below shows an example of the bounding box produced by our object detection and tracking system. As shown in the figure tracking algorithm labels each person with a unique ID which allows us to track the movement of people in the frame. With this information, we can count people entering and exiting doors in Hamerschlag 1300 Wing.

In order to translate bounding boxes and unique identifiers of each tracked person into people entering and exiting different classrooms, there is some additional information that we need to track, namely the boundary of the doors and the direction that each person in the frame is moving towards.

For each of the entrances, we will define a range of x and y coordinates that identifies the bottom of the door. For instance, if there is a door at the top left corner of an image, and points on the image are represented with coordinates x (in range 0 to image_width starting on 0 from the left), and y (in range 0 to image_height starting on 0 from the top), then we will model the bottom of the door as line from (x1, y1) representing one end of the line, and (x2, y2) as the other end of the line. In order to accurately determine the bounding coordinates that define each door, we inspected video footage on python using openCV, and used mouse click events to select the coordinates. Once we have determined the coordinates for the doors, it will remain constant as our cameras are set up at fixed locations and have fixed field of view.

We also maintain a dictionary of unique track_ids and their corresponding movements, done by comparing coordinates bounding boxes between the past several frames. In a given frame, if we detect that the bounding box coordinates of a tracked object crosses the line of the entrance, we will check the movement of that track_id stored in our dictionary (see Fig.4 for example bounding box). If the movement of the track is away from the center of the image (assuming that the Hamerschlag 1300 hallway is placed in the middle of the image, and doors are on the side of the hallway), then we can say that the person is entering the room, and we would increase the count for the room. Whereas if the movement of the person across a given number of tracked frames is towards the center of the image, and away from the line that indicates the entrance to a door, we will treat that as a person exiting the room. For cases where a person appears in the region surrounding the line that defines a door, if we do not have historical data associated with the person, we will treat that as a person leaving the given room as well, since a person appearing at the door without previous detection is likely to be out of the field of vision of the camera to start with (meaning that the person was not in the hallway hence exiting the room).

In addition, we also kept lists of track_ids that entered and left rooms in order to prevent double counting. This is necessary as there could be multiple frames where a person's bounding box crosses the boundary defined by the door, we need to ensure that each track_id could only be counted for going into a room and going out of a room at most once to prevent miscounts. In our system, if a track_id has already been counted as going into a room or leaving a room, it cannot be counted for the respective actions again. This is the correct behavior as if a person enters a room, the track with the current track_id for the person ends as they leave the field of view of the camera, and the next time they reappear into the field of view of the camera, they will be assigned a new track_id. Therefore, each tracked object can at most enter a room once before being identified as a new tracked object, which suggests that each tracked object can at most leave any room once.

After translating bounding boxes outputted by the object tracker, we periodically store the updated counts to our ElephantSQL database as specified by our use case requirements of updating information to the users every minute. Periodically updating the database also avoids excessive network overheads from modifying the web hosted database. The stored information could be retrieved by the web application to display the most recent data every minute.

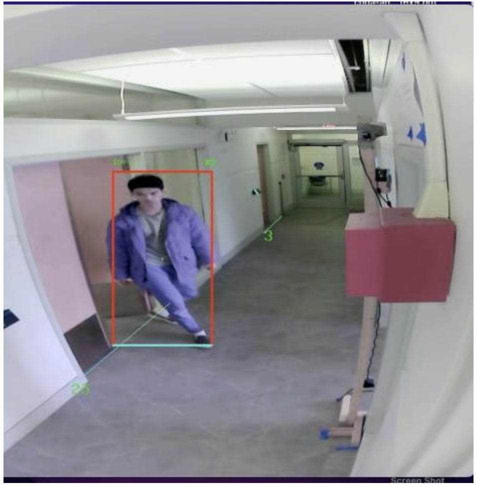


Fig. 4. People Counter's CV output bounding box of a person

C. *Prediction Module*

We trained a decision tree classifier in order to predict the future occupancy in 4 different categories: almost empty, not busy, busy, and almost full. We decided to use categorical prediction because it most likely does not matter to the user whether there are 24 or 25 people out of a room with a maximum capacity of 50 people, it would be more useful to tell the users if the room is almost full or almost empty for them to decide if they want to come to the room to study or not. To train the decision tree, we collected data manually for the occupancy of the rooms in Hamerschlag 1300 wing at different times of the day and on different days of the week. The collected data will be used as a reference point to build a dataset manually. To train the decision tree, we labeled existing data based on features on time-based attributes such as time of the day, weekend vs. weekday, if there is class currently being held in the space, occupancy level in the room during previous hour, etc.

We included occupancy levels in the room during the previous hour in order to support inputting current count as an attribute to predict future occupancy and avoid the output of the decision tree being deterministic based on time. We decided to use the ID3 algorithm to build the decision tree and split on the attribute that gives the most information gain. We created a dataset with 300 data points for training our model and tuned the decision tree using cross validation to get the optimal depth. The trained tree will be stored in the backend and could be used to make predictions of future occupancy levels in almost real-time, since the prediction only requires labeling the data and traversing down the trained decision tree to make a prediction. This would allow us to meet the latency requirements for our system, where the user could get updated data every 60 seconds. During program execution, occupancy data for each room is labeled with the aforementioned labels. Since most of the attributes are time based, we simply translated system time during execution to the labels, and for the prior occupancy data, we queried the database to get the prior count information. The labeled data gets passed into the prediction module to output an occupancy category to be displayed on the web application. Due to time constraints of this project, we were not able to use data collected from our backend to train our decision tree, as there was not enough information. If we had a longer period of time to collect data with our system, we could also optimize the decision tree further by training on more data points.

D. Web Application Module

The web application provides a secure, usable, and interactive platform for users to view and interact with the data that is processed by our backend database. The computed data output from the backend will be integrated with the frontend using Python as well as the Django framework so that the data can be deployed on the server. The basic user interface and frontend styling for the web application is done on HTML/CSS; we will also be using JavaScript frameworks to create an asynchronous web application and ensure the main page will be able to automatically update without having to manually refresh the page for updated information.

For the login page, the web application makes use of the Django authentication package and customized decorators to allow new users to register and old users to login. It will protect the system against malicious users by ensuring that all users will need to register using an Andrew email address provided by CMU. After a user logs in, they will be directed to a home page which shows real-time information regarding occupancy at the Hamerschlag 1300 wing. The database and forms are sent from the cloud-based ElephantSQL database to Django’s Model-View-Controller (MVC) system architecture through a custom Python script, and the relevant data is transmitted and constantly updated over 60-second intervals through the database.

VII. Test, Verification and Validation

The testing and verification of our system was completed in three main phases.

A. Tests on Pre-recorded Footage

Our testing process began with pre-recorded footage captured by the Raspberry Pi vision system, mounted on top of the main Hamerschlag hallway doorway. Tests conducted during this initial testing phase were much shorter and simpler at around 5 minutes each, to allow for easy manual verification and retesting of new developments. For each preliminary test, the footage was saved on an external storage device, and inputted into the computer vision system after the fact to generate the estimations. The main aim of the tests was to observe if the computer vision algorithms could detect and track pedestrians inside the testing environment. After we have verified that the estimated occupancy in each room was within our 80% threshold for the entirety of the recording, we moved on to the next phase of testing.

B. Tests on Live Footage

In the second phase of testing, we simulated the Hamerschlag environment's typical activity by conducting tests for up to 180 minutes straight. After our preliminary tests, we switched to using two cameras for our main vision system when we found that it provided better estimation accuracy, as well as moved our CV processing framework online in order to meet our design requirements for CV processing speed. Additionally, we began mounting our cameras on movable wooden poles, to increase the number of positions in which we can conduct our tests without hurting overall reproducibility. We also began to use auxiliary verification cameras as a secondary measure of our estimation accuracy. These cameras were placed at alternate angles to the main cameras and had a more limited view of the entire hallway environment, but a more advantageous view of specific doors. The additional estimates provided by these verification cameras running in parallel were another baseline in which we could evaluate the estimation accuracy of our system during our test runs.

Our main priority was to ensure that our computer vision pipeline could maintain consistent performance that matches the speed of the live video feeds. We also evaluate the connection between the Raspberry Pi and Google Colab, as well as the ability of our pipeline to operate continuously over an extended period of time. Another objective during this phase was to collect training data for our predictive module. We recorded occupancy levels from the auxiliary cameras during our testing sessions and stored them in a database for future use in building our decision tree model. At this stage, we did not yet have the web application fully operational, and so we did not test our predictive module at the time. Our goal for this testing phase was to ensure that our primary computer vision system could estimate occupancy within 80% accuracy of the actual occupancy (determined either manually or with the verification cameras).

C. Tests with Prediction Module

In the final phase of our testing, we fully integrated our prediction component. The testing setup remained the same as the previous phase, in which we placed our camera mounts with the main and verification cameras in the same locations. At the beginning of each hour of the testing process, the prediction algorithm forecasted which of the four capacity categories each room in the Hamerschlag wing would fall into one hour in the future. At the start of the following hour, this prediction will be compared to a manual measurement of the occupancy, or an estimate provided by the verification cameras if manual verification was unavailable at the time. Our validation target for the predictive model during this testing phase was to accurately predict the occupancy categories of the three rooms with auxiliary cameras, one hour into the future, at an 80% success rate throughout the testing process.

D. *Results*

TABLE II. SUMMARY OF FINAL TESTING RESULTS

Run length (minutes)	Manual Verification	Automatic Verification
25	96.55%	N/A
10	81.82%	88.89%
20	89.19%	93.94%
30	87.81%	91.67%
60	91.07%	96.08%
90	93.62%	88.64%
170	N/A	91.40%
180	N/A	93.33%

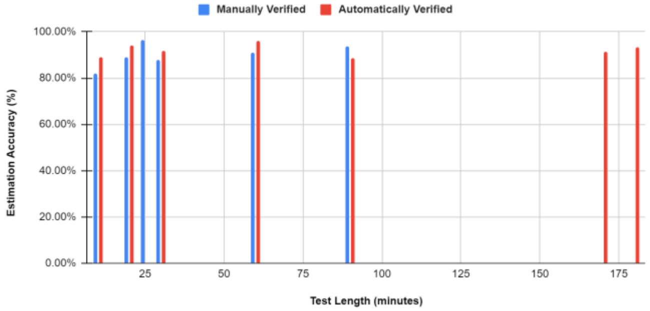


Fig. 5. Final testing results visualized in double column chart

Table II and Fig. 5 demonstrate the estimation accuracy results of our testing runs, shown as a table and graph. We employed two verification methods during our testing: manual and automatic. Manual verification involved watching the stored recordings of the testing runs to determine the actual changes in occupancy. In contrast, the automatic method used the occupancy changes tracked by the verification cameras during the tests as the ground truth for the changes in occupancy. Both methods were performed for the majority of the shorter tests that were under 100 minutes, except for one run in which the verification cameras were not available. For the longer runs, it became infeasible to manually review such a lengthy video, and so only automatic verification was performed.

Our testing runs were able to surpass the use case requirement of 80% estimation accuracy, with the average estimation accuracy of all the runs being 90.33%. The accuracy results obtained from automatic verification were comparable to those from manual verification for the runs where both methods were used, with the average margin of error between the two methods being 5.13%. As such, we can confidently say that our longer testing runs exceeded our 80% estimation accuracy target despite having not manually verified them, considering that the automatically verified accuracy for both runs were more than twice margin of error from the target 80%.

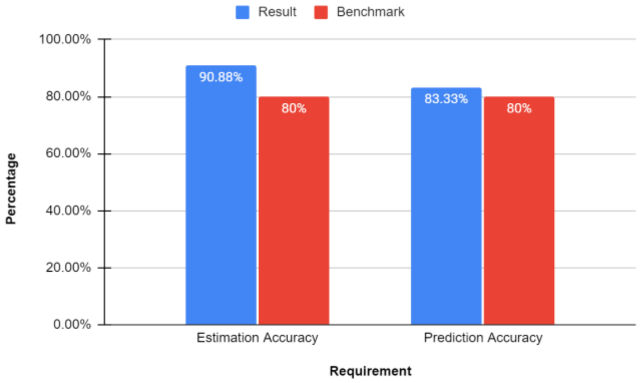


Fig. 6. Final testing results comparing estimation and prediction accuracy visualized in double column chart

Fig. 6 is a graph of our final estimation and prediction results compared to their use case requirements. The verification of the prediction system was done on the combination of all the test runs, after we had gathered as much occupancy data to train our final model as possible. Overall, our test runs, our final model accurately predicted the future occupancy of the Hamerschlag wing rooms 339 out of 373 times, for a final accuracy of 83.33%. This satisfied our use-case requirement of predicting within the correct category at least 80% of the time. Since our project had a shorter timeline and because our model improved over time with additional test data, it is likely that we could have achieved even higher predictive accuracy if we had more time for testing.

Lastly, through the design and optimization of our hardware and software components, along with the selection of lightweight computer vision algorithms, we were easily able to meet our use-case requirement for system latency. Through our test runs, our system had an end-to-end latency of under 3 seconds on average, and a maximum latency of 10 seconds during times of high network load, and the data transfer latency from hardware to the web application is exactly 60 seconds, which we have manually set in our ElephantSQL database script to have 1 update per 60 seconds.

A more comprehensive testing log and notes can be found in Table IV, the last component of this final report.

VIII. PROJECT MANAGEMENT

A. *Schedule*

The detailed schedule for the project is presented in the second to last page of the document. Since the submission of the Design Review report, the team has made some changes to the overall schedule of the project development cycle. On April 14, 2023, we were told that plans with our original setup, which included adhering cameras to the walls of the hallway at Hamerschlag Hall, cannot be continued. In light of this abrupt change near the end of the semester, we had to adapt our testing and development schedule to fit the time needed to revamp parts of our system (including building and installing wood poles, moving backend testing to cloud, as well as changing the door logic since the camera angles have changed). The hardware setup and configuration were integrated with the backend logic in Week 9 (March 13-19), and the backend logic and database were connected to the web application in Week 15 (April 24-30).

B. Team Member Responsibilities

David Feng worked on configuring the camera modules with the Raspberry Pi 4's, calibrating the camera configuration to the testing environment of Hamerschlag Hall 1300 wing, setting up the Ngrok tunnels, and connecting the video feed to the backend software through wireless internet.

Tianzhuo Li is in charge of developing and running the CV-based algorithm for object detection. Additionally, he will work on translating the CV output to estimation data and implementing the prediction algorithm with machine learning models in the backend. Tianzhuo also contributed to the prediction algorithm and the connection of the backend database to the front end.

Gary Qin is responsible for connecting backend data output to the local frontend as well as the server-side web application powered by Django. In addition, he will be working on developing the entire web application user interface using HTML/CSS. Gary's secondary responsibility includes working on parts of the prediction subsystem and testing.

All members roughly spent the same amount of time being physically present in HH1300 to test our backend logic.

C. Bill of Materials and Budget

The bill of materials used for this project and the budget allocation is displayed in detail in Table II in the last page of this document.

D. Risk Management (used to be Risk Mitigation Plans in Design Document)

The team has identified 4 critical risk factors for this project: the speed and efficiency of the backend in processing live video feed, the potential of algorithmic bias being present in the computer vision algorithm, the privacy concerns of people present in the camera frame and the users who interact with the web application, and the overall challenges of designing a project that exhibits both beneficence and non-maleficence to the people involved and their surroundings.

We are committed to improving the performance of our system and optimizing the latency of individual subsystems. One of the aforementioned trade-offs include moving backend testing from local to Google Colaboratory using Ngrok tunnels. As a result of this change, we were able to significantly reduce the risk of the system lagging behind as we were able to increase the average frames per second (FPS) measurement to well above 10 because Google Colab, when coupled with the GPU-enabled CV processing techniques, processes video frames much faster than relying on local CPU.

During the testing and validation stage of our project, we constantly kept a vigilant eye on any potential trace of algorithmic bias present in the CV algorithm when performing object detection. We were on the lookout of any sort of algorithmic bias, including discrimination of race, gender, and height, that causes the algorithm to be incapable of detecting a person under various confidence levels. After finding no trace of algorithmic bias, we also slightly lowered the confidence level parameter of the backend detection algorithm as a precautionary approach to dealing with bias during testing and validation of our integrated system.

We care about the privacy of students, staff, and faculty members whom the system interacts with one way or another. Therefore, we have not stored any live camera feed in a database during testing – the camera feeds are sent to the software backend for processing only. When validating the accuracy of our system, we relied on the percentage of matched outputs between the main cameras and the set of validation cameras rather than putting an emphasis on viewing the video output and manually counting the number of people who enter or exit doors located within the HH1300 hallway. We mounted the camera in a way during testing so that it does not intrude into the normal movement of people in the Hamerschlag 1300 wing, and we also put away the cameras and any peripheral hardware, including wood poles used to secure the camera location, whenever testing is not in session.

Lastly, the purpose of People Counter is to bring accurate information to users. In the meantime, we would not want to do any harm to the people that may be involved and the environment where testing and deployment take place. When we were notified by the department and course staff that we would not be allowed to directly tape our cameras to the walls of the HH1300 hallway, we responded swiftly by taking down our original setup and building wood poles which allow us to mount the cameras and Raspberry Pi's. Even though the notice occurred abruptly near the end of the semester, we mitigated the potential risks by moving quickly to a new setup. In hindsight, our project was able to improve through this change, as the added wood poles granted more flexibility to the location of the system, allowing it to be deployed elsewhere, such as for the final demo.

IX. Ethical Issues

Given the nature of our project idea and design being centered around the movement of humans, there are a number of areas where the project could have been exposed to ethical challenges. Notable ethical issues include the use of cameras in a public setting, the usage and manipulation of a database focused on keeping track of the number of people, as well as algorithmic bias.

Given that People Counter largely depends on the use of cameras to track people moving in and out of the rooms in the HH1300 hallway, people who enter the tracking camera's field view could have been the most vulnerable to failure or misapplication. While the cameras were only installed in the Hamerschlag 1300 wing, there still likely be people who are unaware of the cameras' presence and those who are reluctant to be captured by a camera. We took a number of precautions to ensure that people's privacy is protected and not encroached on. For example, we printed out signs that explicitly state the purpose of the cameras as well as an assurance that the cameras will not be turned on outside of testing times. After switching the system to include wood poles as a method to fix the cameras in place, we only moved the wood poles to the hallway when all 3 team members were physically present in the HH1300 wing for testing, and we moved the poles back to a corner of the ECE capstone room once testing of the day was completed. We also purchased the smallest cameras possible to ensure they would not cause physical harm to people and do not interfere with the normal movement of people in the hallway. By taking these necessary steps, we were able to minimize the impact that our cameras may have created on people who come into the HH1300 hallway.

In formulating the database tables for estimation and prediction, we also considered the ethical concerns related to usage and manipulation of data, especially ones that may be private. In the end, we decided to not include any information other than time and the number of people, namely video feed and the identity of individuals. Compared to the one most available to us (which includes video feeds), the choice of not storing any live video in our dataset helps protect the privacy of individuals who enter the field view of our cameras. Specifically, while our cameras are unable to automatically blur out people's faces and their physical traits, we did not incorporate any private or personal information of people into the process of producing a working project.

Additionally, we took many steps to make sure our Django-based web application is secure. We added a custom decorator to the login and register functions so that the scope of the web application is only limited to people who possess an email address that ends with "@andrew.cmu.edu". Furthermore, we made sure that the web application will not be susceptible to XSS or CSRF attacks by implementing CSRF tokens and sanitizing data inputs.

Finally, we can report that after over 65 hours of cumulative testing time, we have not observed any trace of noticeable algorithmic bias in the CV algorithms we have used. Potential algorithmic bias could have been harmful to the stakeholders of the project, as it could potentially discriminate against the gender, racial background, height, and body type of people being captured by the cameras. In conclusion, we believe that we have minimized the ethical impact of People Counter and created a project that is both beneficent and non-maleficent to the CMU community.

X. Related Work

Even though we believe that the estimation and prediction of usage and occupancy in enclosed spaces have a high demand and a variety of potential use cases, there does not seem to be a considerable number of existing applications or systems in the market. Specifically, we found no similar products currently being used on Carnegie Mellon's campus that help estimate and predict the occupancy of lab spaces and lecture rooms. Nevertheless, there exist systems and products that are slightly different from People Counter in both use cases and implementation methods.

Zensors, a spin-off company from CMU's School of Computer Science, uses existing surveillance cameras near airport security checkpoints to estimate the wait time needed for passengers to go through security [9]. It uses computer vision algorithms deployed on AWS cloud to process live information captured on monitoring cameras, and the computed output is displayed on a big TV screen in front of the security lines at Pittsburgh International Airport that gives travelers a real-time estimation of the expected security wait times. To our best knowledge, in contrast to People Counter's prediction feature, Zensors' airport security product does not predict the wait time of security lines in advance and does not have an interactive web application that allows users to select a future time for wait time prediction.

In addition, the paper by Saralegui et al. describes a case study of using IoT sensors and energy consumption of rooms to predict the rooms' future usage [10]. The researchers utilize a variety of parameters and historical data of an IoT smart home system, namely thermal behavior, energy usage, and humidity, to monitor and predict room occupancy data. The results show the prediction accuracy maintaining at around an 79% clip across multiple tests. This study shows that there are more ways to estimate and predict occupancy in rooms, but the accuracy is somewhat limited given that certain people movement can be unpredictable.

XI. Summary

We have shown People Counter to be a feasible system to estimate and predict the number of people in multiple enclosed spaces within one single hallway. As mentioned earlier in the report, the system has met and, in some circumstances, exceeded expectations from both a design point of view and the use cases. Originally planned for the incorporation of 2 cameras in total, People Counter improved in its scale after the design review stage when we added 2 more validation cameras to the system to provide it with more stability and a better platform for accuracy verification. Given that the unstable nature of many open-source CV libraries and algorithms, there were some limitations to the accuracy of the system in terms of both estimation and prediction. There could also have been certain manual human errors that caused a potential decrease in estimation accuracy. We manually drawn the "lines" that defined a door's boundary during testing, and even though we spent many hours optimizing the lines for each door that connected individual rooms to the hallway, the fact that there was no automated method to define the entry points of doors that would yield the highest detection accuracy with respect to our cameras could have slightly harmed the total accuracy of the system.

A. *Future work*

We will try to continue working on the project after the semester and to further improve the scope of its use cases so that it can be deployed elsewhere on campus as long as our personal budgets can sustain. The public demo is a great opportunity for us to test out the feasibility of having our project implemented in a non-enclosed setting, as the Wiegand Gym at

CMU is a large space without separate rooms.
As of now, the project has already been upgraded in a sense that it has become much more technically complex that we had previously envisioned. People Counter's design leverages the combination of live camera feed, a backend that computes occupancy data with CV and machine learning algorithms, and an interactive web-based application to deliver both real-time estimation and 1-hour predictions of the occupancy levels at Hamerschlag Hall's 1300 lab spaces at CMU.

On a compact college campus like that of Carnegie Mellon, it can be difficult for students to find an ideal place to do some work or relax. For them, it would be a huge waste of time and energy if they come to their favorite spot to study, like the Hamerschlag 1300 wing for ECE students, only to find a packed room.

In the future, we believe that a more mature version of People Counter can help fully resolve this issue, as students will have remote access to real-time occupancy data and will even be able to view the 1-hour predicted occupancy of the lab spaces. With the help from People Counter, students can make a calculated decision with better confidence on whether they would like to travel to Hamerschlag 1300 wing to spend their time, and thereafter they will be able to find more time doing what they truly enjoy, instead of wasting much of their valuable time on the road.

B. Lessons Learned

Validation was definitely one of the more challenging phases of the project, and we may even argue that validating People Counter was more difficult than integrating a working system. Even though manually verifying the accuracy may have been the most accurate method, it was rather unrealistic given we would not be able to prove the level of accuracy achieved in one single test is sustainable long-term. Therefore, we came up with a Python script that allowed for automatic matching of outputs from the database of the main cameras and that of the validation cameras. This method was a lot easier in terms of validating the system, as some of our tests that were longer in length did not have to be manually verified anymore. We were glad that we made the changes to our system after the design review phase, as the alterations have definitely paid off – we were able to successfully test all components of the system and achieve most, if not all, of the goals we set out for ourselves.

Glossary of Acronyms

ECE – Electrical and Computer Engineering
HH – Hamerschlag Hall
FPS – Frames per Second
RPi – Raspberry Pi
XSS – Cross-Site Scripting
GCP – Google Cloud Platform
CSRF – Cross-Site Request Forgery
CV – Computer Vision
VM – Virtual Machine
MVC - Model-View-Controller

References

[1] "Popular Times, wait times, and visit duration," *Google Business Profile Help*. [Online]. Available: https://support.google.com/business/answer/6263531?hl=en#:~:text=To%20determine%20popular%20times%2C%20wait,enough%20visits%20from%20these%20users. (accessed Mar 03, 2023).

[2] "MTA unveils new capacity tracking and real-time location features in Metro-North Traintime App," *MTA*. [Online]. Available: https://new.mta.info/press-release/mta-unveils-new-capacity-tracking-and-real-time-location-features-in-metro-north-traintime-app. (accessed Mar 03, 2023).

[3] L. Lim, M. Kim, J. Choi, and C. Zimring, "Seat-choosing behaviors and visibility," *JSTOR*. [Online]. Available: https://www.jstor.org/stable/26893773. (accessed Mar 04, 2023).

[4] J. Han, "'Squirrel Squirter 2000: Running an Object Detection Model on Raspberry Pi at 30+ FPS.,'" *Medium*, Sep. 04, 2019. https://medium.com/@jayhan_81187/squirrel-squirter-2000-running-an-object-detection-model-on-raspberry-pi-at-30-fps-e38625f3f747. (accessed May 03, 2023).

[5] M. G. Naftali, J. S. Sulistyawan, and K. Julian, "Comparison of Object Detection Algorithms for Street-level Objects," Aug. 2022. (accessed Mar 04, 2023).

[6] "YOLOv5 on CPUs: Sparsifying to Achieve GPU-Level Performance and a Smaller Footprint," *Neural Magic - Software-Delivered AI*, 07-Sep-2022. [Online]. Available: https://neuralmagic.com/blog/benchmark-yolov5-on-cpus-with-deepsparse/. (accessed Mar 03, 2023).

[7] Ahmed, Khaled R. *Smart Pothole Detection Using Deep Learning Based on Dilated Convolution*. https://www.researchgate.net/publication/357093620_Smart_Pothole_Detection_Using_Deep_Learning_Based_on_Dilated_Convolution. (accessed Mar 03, 2023).

[8] Wojke, Nicolai, et al. "Simple Online and Realtime Tracking with a Deep Association Metric." *ArXiv.org*, 21 Mar. 2017, https://arxiv.org/abs/1703.07402. (accessed Mar 03, 2023).

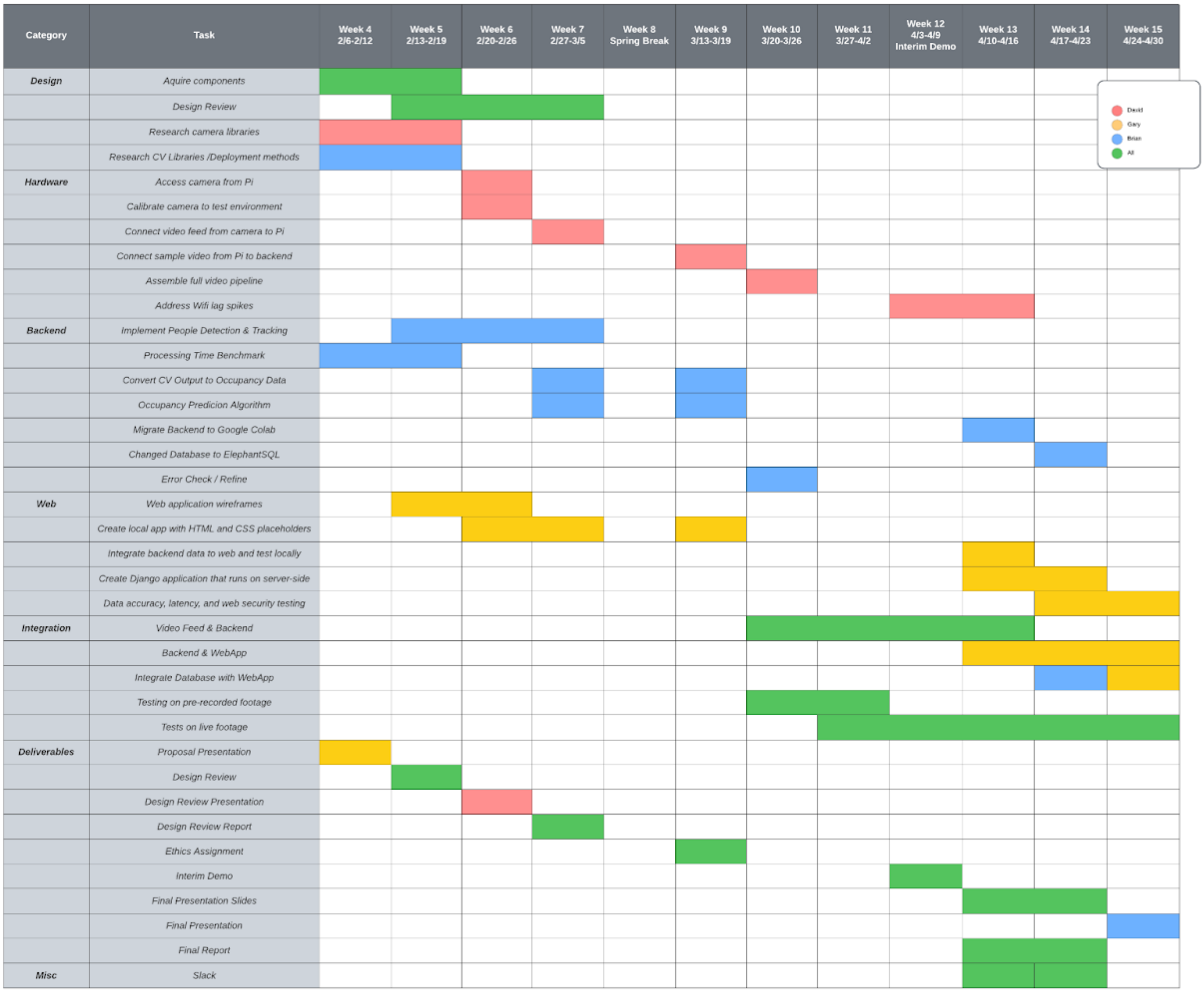
[9] "About Zensors Vision AI," *www.zensors.com*. https://www.zensors.com/company/about-zensors-vision-ai (accessed May 03, 2023).

[10] U. Saralegui, M. Antón, O. Arbelaitz, and J. Muguerza, "Smart meeting room usage information and prediction by modelling occupancy profiles," *Sensors*, vol. 19, no. 2, Jan. 2019. (accessed Mar 04, 2023).

TABLE III. BILL OF MATERIALS

Description	Model #	Manufacturer	Quantity	Cost @	Total
Arducam 1080P Camera [1]	B0829HZ3Q7	Arducam	4	$34.99	$139.96
Arducam 4K 8MP Camera [2]	B09BR1RNSN	Arducam	1	$38.99	$38.99
10ft Extension Cord [1]	L6LAC034-DT-R	AmazonBasics	3	$8.87	$26.61
60yd Duck Tape [1]	394475	Shurtape Tech.	1	$7.95	$7.95
Raspberry Pi 2-8GB [1]	N/A	Raspberry Pi	4	$0	$0
Grand Total					**$213.51**

1. Added 2 1080P cameras, 3 extension cords, 1 duck tape, and 2 Raspberry Pi's after design review report in light of increasing the scope of the project to incorporate 4 cameras in total
2. Not used in final system, as one of the nails of the camera's rotatable base fell off during testing



Category	Task	Week 4 2/6-2/12	Week 5 2/13-2/19	Week 6 2/20-2/26	Week 7 2/27-3/5	Week 8 Spring Break	Week 9 3/13-3/19	Week 10 3/20-3/26	Week 11 3/27-4/2	Week 12 4/3-4/9 Interim Demo	Week 13 4/10-4/16	Week 14 4/17-4/23	Week 15 4/24-4/30
Design	*Aquire components*	🟩	🟩										
	Design Review		🟩	🟩	🟩								
	Research camera libraries	🟥	🟥										
	Research CV Libraries /Deployment methods	🟦	🟦										
Hardware	*Access camera from Pi*			🟥									
	Calibrate camera to test environment			🟥									
	Connect video feed from camera to Pi				🟥								
	Connect sample video from Pi to backend						🟥						
	Assemble full video pipeline							🟥					
	Address Wifi lag spikes									🟥	🟥		
Backend	*Implement People Detection & Tracking*		🟦	🟦	🟦								
	Processing Time Benchmark	🟦	🟦										
	Convert CV Output to Occupancy Data				🟦		🟦						
	Occupancy Predicion Algorithm				🟦		🟦						
	Migrate Backend to Google Colab										🟦		
	Changed Database to ElephantSQL											🟦	
	Error Check / Refine							🟦					
Web	*Web application wireframes*		🟨	🟨									
	Create local app with HTML and CSS placeholders			🟨	🟨		🟨						
	Integrate backend data to web and test locally										🟨		
	Create Django application that runs on server-side										🟨	🟨	
	Data accuracy, latency, and web security testing											🟨	🟨
Integration	*Video Feed & Backend*							🟩	🟩	🟩	🟩		
	Backend & WebApp										🟨	🟨	🟨
	Integrate Database with WebApp											🟦	🟨
	Testing on pre-recorded footage							🟩	🟩				
	Tests on live footage								🟩	🟩	🟩	🟩	🟩
Deliverables	*Proposal Presentation*	🟨											
	Design Review		🟩										
	Design Review Presentation			🟥									
	Design Review Report				🟩								
	Ethics Assignment						🟩						
	Interim Demo									🟩			
	Final Presentation Slides										🟩	🟩	
	Final Presentation												🟦
	Final Report										🟩	🟩	
Misc	*Slack*										🟩	🟩	

🔴 David
🟡 Gary
🔵 Brian
🟢 All

Link to Comprehensive Test Log