ScottySeat

Aditi Raghavan, Mehar Goli, Chen Shen

Department of Electrical and Computer Engineering, Carnegie Mellon University

***Abstract*— Students waste time searching for study spaces, and the goal of ScottySeat is to streamline this process. Our solution is to capture images of study spaces and update a web application in real-time to reflect the seat availability in a given room. To accomplish this, we will be using YOLOv5, a pre-trained object detection algorithm to identify people and chairs.**

***Index Terms*—Computer vision, object detection, single board computer**

I. Introduction

Campus study spaces are a crucial commodity for many students at CMU. Whether it be for individual or group work - campus provides a reliable, safe location for students to work at. However, CMU has limited study space and free spots are hard to come by.

Looking at campus itself, popular areas tend to fill up quickly while smaller spots aren't widely known. With the size and complexity of CMU's campus, finding a single free space can take upwards of 30-40 minutes and even longer for group scenarios. Currently, classrooms/meeting rooms can be reserved but this doesn't account for open study areas that make up the majority of student study areas. Room reservations are usually restricted to organizations or students of the school where the room is located. Reservations also fill up quickly and don't allow for time flexibility to book closer to study time.

One potential solution is to work off-campus or at home, this isn't but this always isn't the most viable option. At home, students may have noisy roommates/neighbors. Off-campus locations like the public library and cafes may also be noisy or close early as well. Specifically for working in the evening and late into the night, campus is the safest option. For student groups as well, campus serves as the common congregation point that all group members can access if working at a teammate's house isn't possible or teammates live too far apart.

Our project seeks to ease the struggle in and reduce the time taken in finding campus study spots by providing students with easily accessible, real-time information on study spot availability. To do this, we will build a web platform showing digital maps of study areas. Each study area will have its own map indicating study spot locations and availability. Real-time camera footage of the areas will be monitored using computer vision to regularly update the site as well.

II. Use-Case Requirements

To meet our use case, we have created requirements for numerical accuracy, speed and spatial accuracy. Numerical accuracy in our use-case relates to the number of seats available in a study space. This is arguably the most important use-case requirement as low numerical accuracy can misguide users meaning that there would be no speed up in finding a study spot. Keeping this in mind, we aim to have 90% accuracy in detecting available seats. Our second use-case requirement relates to the update speed from capturing an image to the results being displayed on our webpage. For our solution to be useful, the data provided to the user should reflect the study rooms current capacity. After polling some potential end users and taking their feedback, we have decided that changes in seat availability must be reflected within 45 seconds of them occurring. Our last requirement is regarding the user interface. For users to be able to use our UI with ease, the seat mapping needs to be easy to interpret. This means that the seat mapping should closely reflect the positions of the real positions. This spatial accuracy use-case requirement is quantified as being able to map chair positions to seating charts with a 20% error margin. This metric will be measured by comparing the seat mapping with a camera with an overhead view, meaning no perspective correction is needed.

III. Architecture and/or Principle of Operation

Our system can be mainly divided into 3 subsystems, including the hardware module, the computer vision module, and the web application (UI) module, as shown in the diagram below (Figure 1).

The sensing module consists mainly of TedGem USB cameras. Such cameras will be deployed one for each study space being monitored. Its main purpose is to provide input for the CV algorithm for detection, and also provide training and testing data that would be used for verification. There are 2 kinds of positions in which the camera would be placed: an angled camera and an overhead camera (Figure 2).

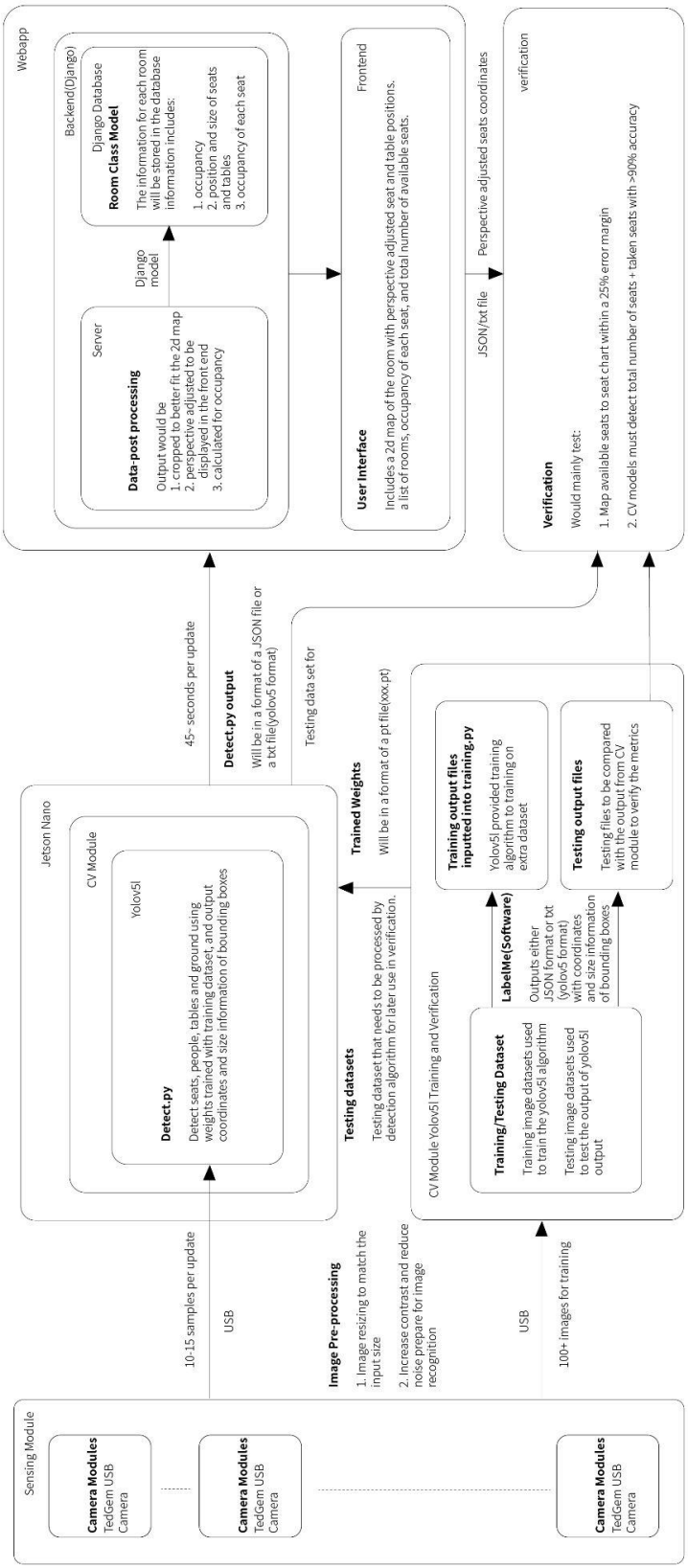

Figure 2: System Block Diagram

The angled camera would be responsible for gaining input for detection, the training data, which is used to train for further training the weights of the CV model, and also the testing data, which would be used to verify the accuracy of the model output. Both the training and testing dataset would be processed later with labeling tools such as the Computer Vision Automated Tool (CVAT). The overhead head camera would only be used for collecting testing dataset for verification of the accuracy of the position of the seat after the perspective adjustment on the server side of the webapp.

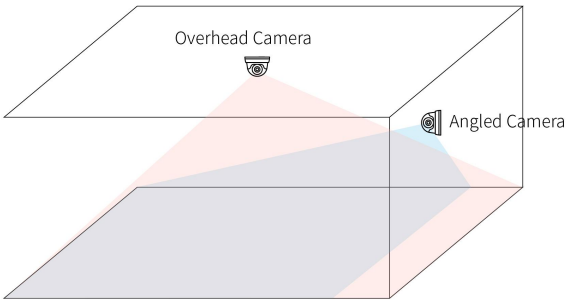

Figure 1: Overhead and Angled Cameras

The pipeline of our solution is that we will capture images using our angle camera, sample a frame every few seconds, preprocess it, run it through a custom trained version of YOLOv5, an object detection algorithm. We will take 10-15 samples, and take the highest confidence samples, post process them and update the web application.

IV. Design Requirements

To achieve our first use-case requirement, we will need a single high quality camera per room and must have a highly accurate object detection algorithm.

For the camera footage, we believe that a 1080p resolution to be able to accurately detect objects and their edges. The camera needs to have a large field of view to be and be placed correctly so that it captures the entirety of the room in a single shot. Our set-up must work in a variety of lighting conditions so the camera should have automated exposure adjustment.

To achieve our first use-case requirement, the object detection algorithm must be able to identify chairs and people sitting on chairs with 90% accuracy. To help aid with achieving this accuracy. We plan to sample at least 20 times over a minute and use the detected objects with the highest confidence. If the highest confidence detected in an image is less than 80% we will ignore the images taken in that minute. The algorithm will need to be able to also discern clearly between nearby objects with high accuracy and will also need to have very high accuracy on occluded objects.

To achieve our second-use case requirement, speed, we will need a relatively fast algorithm which does not compromise accuracy for speed. The website should automatically refresh every 45 seconds and we are giving 1 second for final changes to be reflected on the page. Given that we have a 44 seconds window between a change occurring and an update, and aim to have 20 samples/min, we will have ~14 samples between the change and the update. This means the algorithm needs to be able to run and complete within 3 seconds.

To achieve our spatial accuracy requirement, we will need to ensure that the bounding boxes placed around the objects by the computer vision model are accurate. Additionally, we will need to carry out perception correction as a final post processing step before displaying the maps to users.

V. Design Trade Studies

A. Hardware

The motivation for using hardware, instead of software for running our computer vision model was due to privacy. Our project captures images in a public space, and to help ensure privacy of the users, we thought it was best to use a single board computer. This way images will not have to be sent over the network, reducing the possibility of hacking, and the images can be easily deleted after use. The only data being sent over the network will be the locations of chairs and tables and so users can safely participate in our project while remaining anonymous.

The choice of using a single board computer (SBC) compared to an FPGA was that it would be much easier to scale an SBC model versus an FPGA model. Additionally, networking on an FPGA is non-trivial and translating the computer vision models would be out of the scope for the given project timeline.

The main difference between the NVIDIA Jetson Nano 2GB Developer Kit and the Raspberry Pi is their compute power. Since we intended on using machine learning to identify objects, a larger GPU is preferred as it is able to parallelize many of the matrix operations needed in machine learning. While analyzing YOLOv5, researchers found that the RPi, out of the box had an FPS of 1.6 FPS while the Jetson Nano had an FPS of 5 FPS [1]. Considering that we are using 2 cameras for our MVP, and have some preprocessing and postprocessing to do on our input image, using the RPi would possibly become a limiting factor in our speed requirement. A solution to this might be to use a smaller YOLO model but scaling down the algorithm results in a reduced accuracy according to benchmarks released by Ultralytics [2]. Another possibility was to use the RPi along with the Intel Neural Compute Stick 2. According to Feng et al., the RPi + NCS2 outperforms the Jetson Nano on both mean confidence and FPS, when using YOLOv3 [3]. However, the Intel NCS2 is out of stock, and for our project timeline, we needed our hardware as soon as possible.

B. Computer Vision Model

The motivation behind computer vision as opposed to a physical method such as seat sensors was scalability. Scalability is a simpler process with computer vision, by adding a few new cameras to a space rather than individual sensors to every chair in the area.

The main goal of our computer vision model is to detect chairs, tables and people in a given room from a given image. Chairs will be oriented in different positions and will likely be partially or mostly occluded by a table or a person. The images themselves will be given every three seconds and the model will also run on a Jetson Nano. As such some requirements we had for a given CV model were as follows:

- Classify and locate multiple objects in an image
- Scalable to multiple object classes
- Implementable on Jetson Nano
- Can be built to run in under 2-3 seconds
- Can handle object occlusion and orientation variability
- Optimally, have the highest starting accuracy with lowest resource usage.

Overall, we considered a variety of object detection architectures: namely neural network-based methods (Faster R-CNN, Yolo) and feature detection-based methods (BRIEF, SIFT, ORB). All the architectures noted can be implemented in Python through the use of common libraries and thus can run on the Jetson Nano. Specifically, the underlying libraries used are OpenCV (BRIEF/SIFT/ORB feature descriptors are built in), and Tensorflow/Pytorch for neural network models. They all also run relatively quickly - in close to real time. .

From there, the feature detection methods tend to have the lowest resource usage but also come with a number of risk areas due to how object detection is ultimately done. With feature detectors such as BRIEF/SIFT/ORB - object detection is generally performed through image matching [5]. The feature descriptors detect various 'features' in a source image (i.e. corner points, areas of high contrast and variability) and a thresholding model such as KNN is used to match those points to an object in a test image. Thus, it operates essentially as a one to one image matching architecture. This makes it difficult for these methods to account for object occlusion if not enough points are present. In our use case especially, chairs can't be moved by a user to help object detection so the system needs to account for heavy object occlusion. The system also becomes difficult to scale to multiple object classes. Different sets of descriptors and models for each object class, that is for different types of tables, different chair designs will be needed. Lastly, image matching for object detection also makes this method unideal for person detection as well, a separate system will be needed to track people in the scene.

In contrast, the neural networks are pre-trained to work on multiple object classes with multiple objects per image. For example, YOLO in particular is trained to detect 80 different object classes. In testing we can see, out-of-the-box YOLO is able to detect tables, chairs, people and even some objects such as the TV and a backpack as well. We can also note that YOLO is able to detect the chairs despite heavy occlusion.

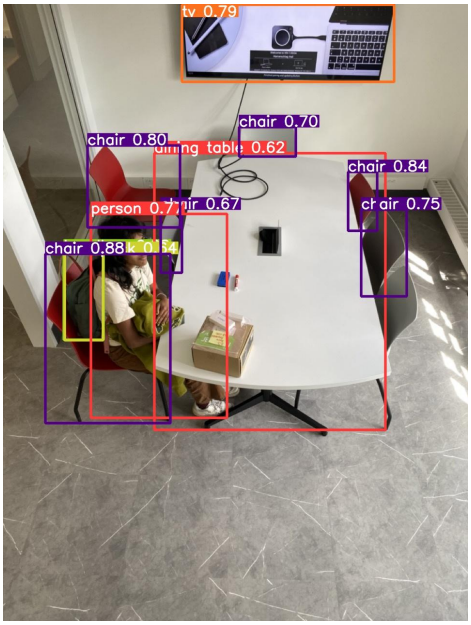

Figure 3: Sample Yolov5L output

It is possible to build a feature descriptor-based architecture to work for our use case, however with the issues that will need to be accounted for and the time that will be taken for training of either system type - a neural network based model is preferable for our case.

Of Faster-RCNN, YOLO - Faster R-CNN is noted to have the highest precision and slowest time. The creators of YOLO themselves found that a Faster R-CNN with VGG-16 architecture achieved a mAP of 73.2% with a speed of 7 frames per second as opposed to standard YOLO achieving 63.4 mAP and 45 FPS (both are when tested on COCO2007 - a CV dataset) [6]. Here mAP refers to Mean Average Precision, here it tells us the system with the highest true positive to false positive ratio. To see whether Faster R-CNN or YOLO will prove more accurate for our use case. We tested Facebook's Detectron 2 Faster R-CNN architecture and Ultralytic's YOLOv5L architecture locally using 6 iPhone test images of our study space. Each image showed a single human, one table and 7 chairs. Testing showed, YOLO was able to accurately detect more objects than Faster R-CNN even with lower resource use, though Faster R-CNN did generally seem to have higher confidence in its results through a qualitative analysis. Through this we determined we will finally use YOLOv5L for our project.

Table 1: Detection Testing off Faster RCNN vs Yolov5L

Architecture	**Human Detected?**	**Table Detected?**	**Chairs Detected**	**Objects Detected**
Faster RCNN + ResNet 50	6/6	6/6	18/42	30/54
Faster RCNN - ResNext 101	6/6	6/6	25/42	31/54
Yolov5L	6/6	6/6	39/42	39/54

VI. System Implementation

A. Subsystem A - Computer Vision

The computer vision pipeline comprises three sections: preprocessing, CV model and postprocessing (sample consolidation), denoted with bold text for the start of each section. Overall, the system will take in an image sampled from the camera feed and output list of object detected and confidence level for each object, and coordinates for a bounding box around each box.

Preprocessing: For the preprocessing of the image, we will perform some denoising and contrast increase, along with brightness increase as well. These preprocessing steps were determined based on a test image from the camera feed

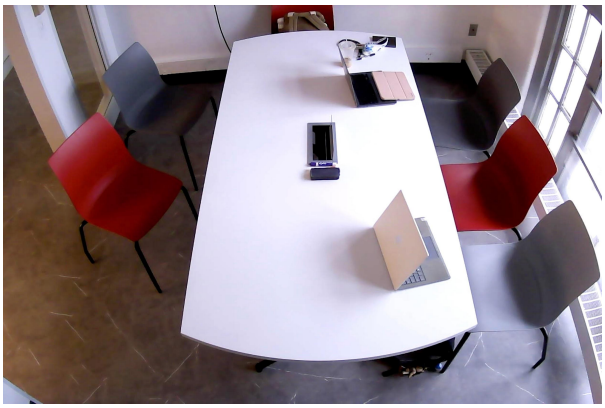

Figure 4: Initial webcam test image retrieved by standing on a chair to simulate camera angle

As can be seen in the test image, the image has some graininess and many chairs tend to blend into the floor. Denoising and contrast increase can help to mitigate these factors. Beyond this, the brightness increase is to account for differences in lighting conditions throughout the day. Further testing will be done to see exactly how much denoising/contrast/brightness will be helpful towards object detection.

CV Model: From there, the sampled image will be run through a pre-trained YOLOv5l model to retrieve bounding boxes for objects detected, the classes of the objects detected and the confidence level in the classifications made for each object. Below is a visual representation of the data that will be outputted - except through the form of a list.

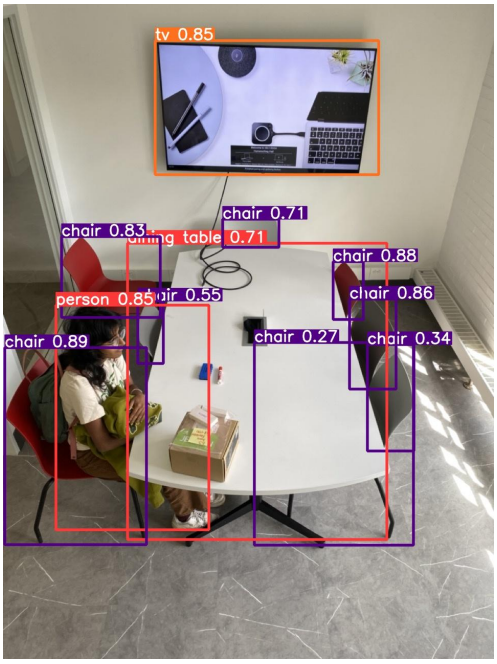

Figure 5: Sample YOLOv5 output using iPhone test image

The YOLOv5 model will be trained further using a combination of a custom dataset and the open-source ImageNet dataset. Ultralytics Yolov5L is pre-trained to work across 80 object classes on the open-source COCO image dataset. So, initially the model will be trained further on a different popular dataset ImageNet to be optimized for the table, person and chair classes. Any further training needed will be done using the custom dataset collected. The custom dataset will be image data collected of the study space using the webcam. Specifically, the dataset will contain images of different numbers of chairs in the study space, with chairs being occluded by the table, by people sitting in the chairs. The dataset will be prepared for training by manually drawing bounding boxes and tagging the boxes with the object classification using CVAT.

Post-Processing: From here the post-processing of the CV pipeline is centered around consolidating the 14 samples the system takes for each update. Each processed image data will be temporarily stored in a local folder until a batch of 14 samples are collected. From there the 14 images and their data will be consolidated into a single image data JSON file that will be sent for use by the web application. The folder will then be emptied in preparation for the next batch of images.

The exact methodology for consolidating the 14 images will be tested; however, we do have a potential algorithm to use. All 14 images will be thresholded to only keep detected objects with at least high confidence. The ‘high confidence’ threshold is yet to be determined, however we only want objects with high confidence to increase accuracy potential. From there objects from all 14 images are compiled together. Objects appearing in at least 60% of images will be kept. Specifically, if more than 60% of images have an object where the bounding box overlaps significantly (at least 70% to

account for minor position changes), then the object will be kept and the bounding box of highest confidence used for that object. The motivation behind this secondary step is that while we do want high confidence detections, objects are sometimes covered between frames - so we want to account for cases of non-detection due to excess occlusion as well.

B. Subsystem B - Related Hardware Components

To run our computer vision algorithms, we will use a NVIDIA Jetson Nano 2GB Developer Kit. Along with this, we plan to use 2 1080p TedGem USB Cameras to capture our images. The website will be hosted locally, and we will connect to the CMU network using ethernet.

The Jetson will create two instances of a camera object, and open the streams using the gstreamer library. The gstreamer library will capture a frame from the camera and feed each image into an instance of YOLOv5. The website will be hosted using the python in-built library, http.server

C. Subsystem C - Web Application

In order for the output of the CV algorithm to be displayed in a user-friendly way, a web application interface has been created. It consists of 2 parts: frontend and backend.

Backend:

The backend side, namely the server side, with every 45 seconds of update, is firstly responsible for parsing in the output file by the CV module which contains position information of chairs, people and tables, and the bounding box sizes of chairs and tables. Then, the server will apply post processing to the coordinates. This process involves cropping the image so that the image mainly consists of the ground and removing the walls from the image. Then, an algorithm to adjust the perspective due to using an angled camera will be applied to adjust for the position difference between an overhead view and an angled view. After this, the occupancy of each seat will be calculated thanks to the adjustment done by the previous process. The next process is to store all that information into the database, and assign all that information to the corresponding room, so that any request the server receives can be easily processed by extracting the corresponding information from the database.

Frontend:

The frontend, namely the website, on loading, will send a request to the server side requesting the information of a certain room with a "POST" request since the name of the room has to be sent along with the request. The server will extract the response from the database and respond with json format, and the website will be updated with the information and display a map of the seats and tables of the requested room. The total counts of availability and occupancy will also be updated. When the user clicks into another room, another request will be sent to the server side and subsequently update the room with the information.

A layout of the frontend interface:

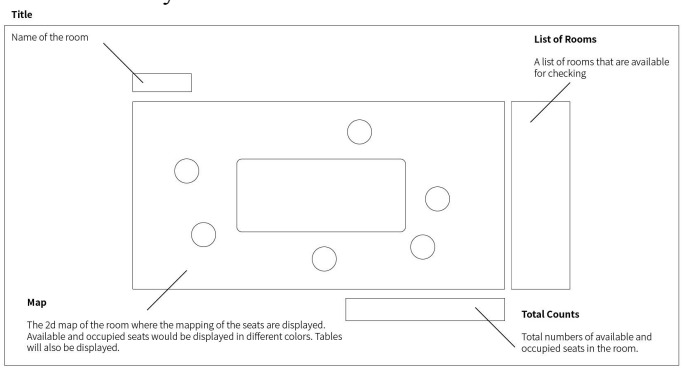

Figure 6: Web Application Layout

VII. Test, Verification and Validation

A. Tests for Use- Case: Object Accuracy

The computer vision model must detect the total number of seas with 90% accuracy. This test relates directly to our use-case requirement on accuracy. To carry out this test, we will take pictures of our study spaces we plan to use in our MVP to verify this. We will capture 100 images of the study space, using different lighting conditions, different chair orientations and introducing other objects like books, laptops and bags. We will label this data using a tool like the Computer Vision Annotating Tool to assist us in the labeling process. We will then feed this into our computer vision model and compare the outputs with our labeled image.

B. Tests for Use-Case: Speed

To ensure we make our speed requirement, we need the entire pipeline to run in under 30 seconds. As derived in section IV, the computer vision model needs to run under 3 seconds with 2 cameras running. To test this, we will use the python time library to verify this. We can reuse the dataset used for testing our computer vision accuracy. Another larger tes will involve verifying that the time between an image being captured and an update being reflected on the website. We can use the same timing libraries to check this, and can artificially feed in an image where the number of chairs occupied changes.

C. Tests for Use-Case: Spatial Accuracy

To ensure we meet the spatial accuracy requirements, we need to be able to validate that our mappings accurately reflect their real positions. To achieve this, we will take images from directly overhead, and calculate the center points of the chairs. We will then compare this with the final seat mappings. We will calculate the error of all objects in the room and aim to have an average of less than 20%.

VIII. Project Management

A. Schedule

A detailed copy of our updated schedule is presented in Figure 8. We are currently on the last steps of our design phase for computer vision, while all other areas are in the building phase. We plan to do an initial integration after break to make sure pipeline components are working together. Overall we will reach our MVP by Week 9. We will first work with one room, having the initial system working by Week 7 (latest mid-Week 8) and then scaling to two rooms by Week 9. Week 10 to 11 is left as slack time for any last improvements to the system and for final documentation prep.

B. Team Member Responsibilities

Team member responsibilities, primarily, are delegated based on each member's expertise - Aditi: Hardware, Mehar: Computer Vision, Chen:Web Interface. Larger sections like CV and integration/interfacing tasks, are also split across members to make sure work is evenly split. Throughout our timeline as well, members will be working together for tasks that may need more hands.

Aditi:

- Jetson Nano + Camera physical pipeline
- Interfacing from Jetson to Web App
- Packaging CV models for Jetson Nano
- Yolov5 Data Collection

Mehar:

- Image Preprocessing for Noise Reduction, Contrast
- Image Postprocessing for Sampling Consolidation
- Yolov5 Model Training + Data Collection

Chen:

- Web Application Development
- Translating CV output for Seat Map
- Interfacing from Jetson to Web App

C. Bill of Materials and Budget

The bill of materials and budget can be found in Figure 7.

D. Risk Mitigation Plans

One of the risks we have is the occlusion of the chair by a person when the person is sitting on the chair. This is the one of the main reasons we are using an angled camera instead of an overhead camera, just so that we have a better chance to make more parts of the chair visible to the camera. In order to mitigate this, we can:

1. Acquire/Build a custom dataset on mostly covered chairs so that the algorithm could better recognize a covered chair as a chair.
2. A backup plan to show only empty seats in the room, as from the user standpoint, the most important information is whether or not there are available seats in the room.

Another risk we have is that we might be overly sensitive to changes. Consider the case where a person leaves the room for 3 minutes for restroom, or stands up for 5 minutes to have a rest, the change would be immediately updated on the map and thus might be misleading to users who came all the way just to find out that the seats are actually not available. This can be mitigated by:

1. Adding an additional state besides "occupied" and "available", which holds the seat for 5 minutes after the seat transitions from the "occupied" state.
2. Another way is to detect any items on the table close to the area where the seat is located, and doing this will require a lot more calculation power, and possibility resulting in seats that are constantly falsely occupied due to some object present on the table.

IX. Related Work

We are mainly aware of 2 projects that are similar to ours: one 18500 Fall 2020 Team B4 Smart Library, and one from Fall 2021 Team A3 FreeSeats. Ours differs from Smart Library as we have dynamic seat mapping which accounts for more cases and situations, while FreeSeats are using sensors mounted on a chair to detect and monitor each chair and reflect it on an app.

X. Summary

With the implementation of our design, students no longer have to travel around campus to find a study space as they could do so just by a single click. Our solution is to place cameras in study rooms and use computer vision to identify available seats. We will display the available seats and where they are located on a web application. We hope to save users time, and more efficiently utilize all of the study spaces CMU has to offer. Our largest challenge will be achieving our 90% accuracy in identifying seats, there are known unknowns here as we have some ways to mitigate this risk but we do not know to what extent they will help us and whether they will work as expected.

Glossary of Acronyms

AJAX - Asynchronous JavaScript And XML
CV– Computer Vision
R-CNN - Region-Based Convolutional Neural Network
ML - Machine Learning
RPi – Raspberry Pi
JSON - JavaScript Object Notation
XML - Extensible Markup Language
YOLO -You Only Look Once

References

[1] Bochkovskiy, Alexey, Chien-Yao Wang, and Hong-Yuan Mark Liao. "Yolov4: Optimal speed and accuracy of object detection." *arXiv preprint arXiv:2004.10934* , 2020.

[2] Ultralytics, “Ultralytics/yolov5: Yolov5 in PyTorch > ONNX > CoreML > TFLite,” *GitHub*. [Online]. Available: https://github.com/ultralytics/yolov5. [Accessed: 14-Oct-2022].

[3] H. Feng, G. Mu, S. Zhong, P. Zhang and T. Yuan, "Benchmark Analysis of YOLO Performance on Edge Intelligence Devices," 2021 Cross Strait Radio Science and Wireless Technology Conference (CSRSWTC), 2021, pp. 319-321, doi: 10.1109/CSRSWTC52801.2021.9631594.

[4] Redmon, Joseph, et al. "You only look once: Unified, real-time object detection." *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016.

[5] S. A. K. Tareen and Z. Saleem, "A comparative analysis of SIFT, SURF, KAZE, AKAZE, ORB, and BRISK," 2018 International Conference on Computing, Mathematics and Engineering Technologies (iCoMET), 2018, pp. 1-10, doi: 10.1109/ICOMET.2018.8346440.

[6] J. Redmon, S. Divvala, R. Girshick and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016, pp. 779-788, doi: 10.1109/CVPR.2016.91.

Item	Manufacturer	Model Number	Quantity	Cost	Notes
Jetson Nano 2GB	NVIDIA	P3541	1	0	From course list
USB Cameras	TedGem	CE0140_01	2	0	From course list
USB Extension Cable	AINOPE	N/A	1	13.99	amzn1.sym.67f8cf21-ade
Ethernet Cable	Amazon Basics	HL-001764	1	9.76	Back-up for WIFI adapter
WiFi Adapter	Edimax	EW-7811Un V2	1	9.99	zn1.sym.67f8cf21-ade4-
Micro-USB to USB C	Cable Matters	201003-BLK-1m	1	7.99	m.67f8cf21-ade4-4299-b
Reclosable Fastners	3M	N/A	1	7.88	To mount and unmount cameras
Colab Subscription	Google	N/A	2	10	$10 Montly subscription for 2 months
			Total	69.61	
			Budget Left	530.39	

Figure 7: Table of Materials and Costs

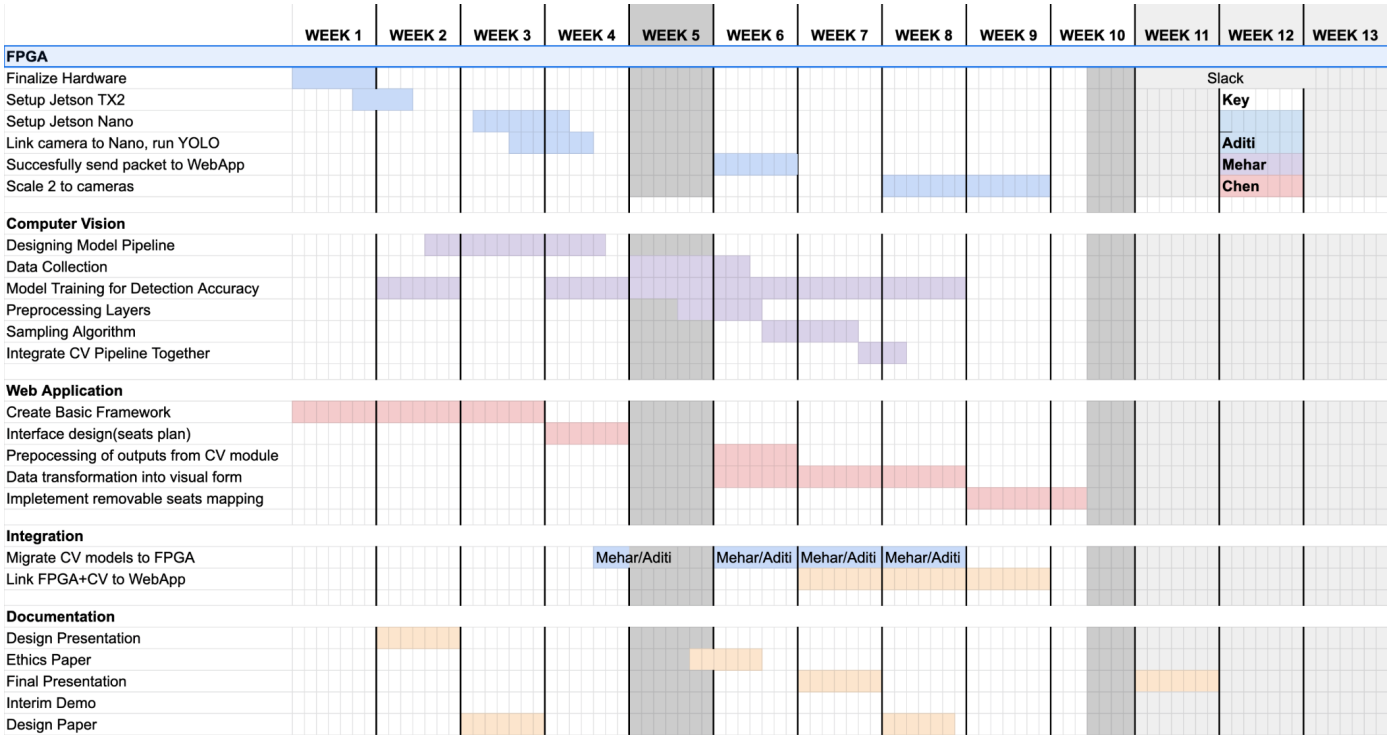

Figure 8: Project Schedule With Major Benchmarks