Smart Library

Authors: Krish Vaswani, Pablo Wilson, Arjun Raguram, Electrical and Computer Engineering, Carnegie Mellon University

***Abstract*—**In the time of coronavirus, social distancing has become an important aspect of human life. Public spaces that once had space for plenty of people must now limit their intake because of social distancing. In order to organize the availability of different spaces, we introduce Smart Library. In this project, we set up a system of cameras in a public space which constantly monitor the availability of socially distant seating and transmit this information to a public website. This is achieved with four cameras sending information to a central node, which preprocesses the data and sends it to a server in the cloud to perform image recognition and parse the information. The information is then transmitted to the public through a website.

***Index Terms*—Camera, Sensor, Machine Learning**

I. Introduction

Libraries are a huge part of the lifestyle of a university student. It is a haven where students can go to do homework or focused studying for an upcoming exam. On the Carnegie Mellon University campus, one of these libraries is Sorrell's, which can get very crowded. Finding seats there can be difficult, even when you take into account social distancing protocols during a pandemic. An existing solution is a project called Carrel Corral, which detects the carrels available in Hunt Library using a system of sensors and LEDs that send information to an app. One problem with design is that it physically interferes with the space of the library with wired components, which could impede the work of a student. Also, this design only works in one place (Hunt Library).

Smart Library is a system that aims to help college students find proper seating in a library or public space through wireless methods. It consists of a wireless system of cameras connected to microprocessors that will take bird's-eye view images of a space and send them to a cloud server to run image detection of seats available for use. It also aims to analyze daily/weekly behaviors of a library of when seats tend to be available. The users of the Smart Library system will access the information obtained via a website. Our main goal is to use 4 images from 4 cameras to send it and run image detection on a cloud, as well as send the information to the website in less than or equal to 1 minute. This will translate to a camera frame rate of 1 frame per second. We will also aim to achieve a battery life of 72 hours and an object detection accuracy of 76.5%.

II. Design Requirements

Since the project was inspired by the lack of availability of space at Sorrell's Library at Carnegie Mellon University, our design requirements were derived from this one use case. By observation, we know the seats are usually occupied in the span of a few minutes after being left vacant. Therefore, it is necessary to update information at least once per minute. Our design choices were mainly based on this time constraint. Additionally, the funds acquired for this project was $600, and some design decisions were made to remain under budget.

Besides the time and budget constraints, there were two other minor constraints. If the batteries need to be replaced too often, it would become inconvenient for the people managing the library, and therefore Smart Library would become a burden to maintain. In order to ensure that the project remains relevant, we require the system to be able to function independently for 72 hours without changing the battery. Additionally, we have a requirement for the accuracy of image recognition for the project. The original plan for image recognition was based on the YOLO model, which has a top-1 accuracy of 76%. In our implementation, we modify this model to detect specific objects. Since this is more specialized, we also require the implementation to be at least as accurate as the original model that is more general.

Component	Time Allowance for Task Completion
Camera Node	20s
Central Node	30s
Cloud	10s

Fig. 1. Time Requirements for Each Project Segment

In order to ensure the design meets the time specifications, the requirements are further divided into three parts. Namely, camera nodes, the central node and the cloud. The breakdown of the timings for each section is given in Table 1. These timings are based on the specific function and hardware chosen for each portion, which are described in the Principle of Operation and System Description sections.

III. Principle of Operation

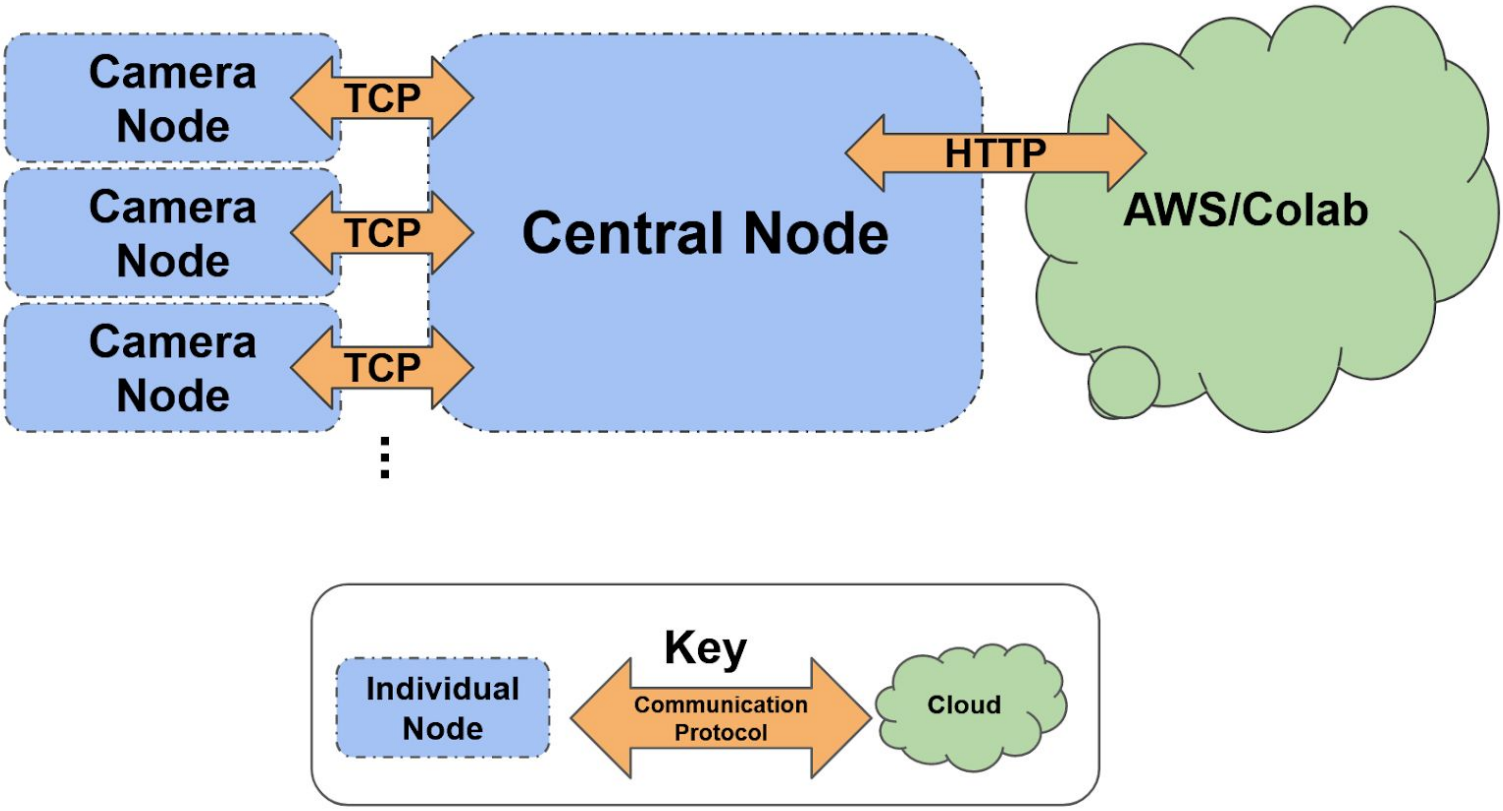


Fig. 2. Block Diagram of Node Network with 1 Camera Node Expanded for Detail

Our design operates by using a Node Network, consisting of many Camera Nodes and a single Central Node, and Machine Learning to determine where seats are available in a space. Our design utilizes a multitude of Camera Nodes to capture images of a location which then sends them over TCP to a singular Central Node. The Central Node then collates and sends the batch of images to our Machine Learning algorithm. The algorithm goes through and determines which seats are available and subsequently updates the website in realtime. A user of our system is able to visit the website and be able to make a decision about where to sit, without needing to visit the library first. An expanded block diagram of the entire system is available in section X.

While our overall operation has remained the same since the design report, specific components carrying out the functions have changed slightly. For Camera Nodes, the LiPo battery was changed to a Portable Charger due to costs. This allowed each node to have a much greater lifespan at the cost of not being able to remotely monitor battery level. For the Machine Learning Algorithm, the YOLO Model was swapped to a Convolutional Neural Network due to the YOLO Model not being optimized for detecting objects from a bird's eye view.

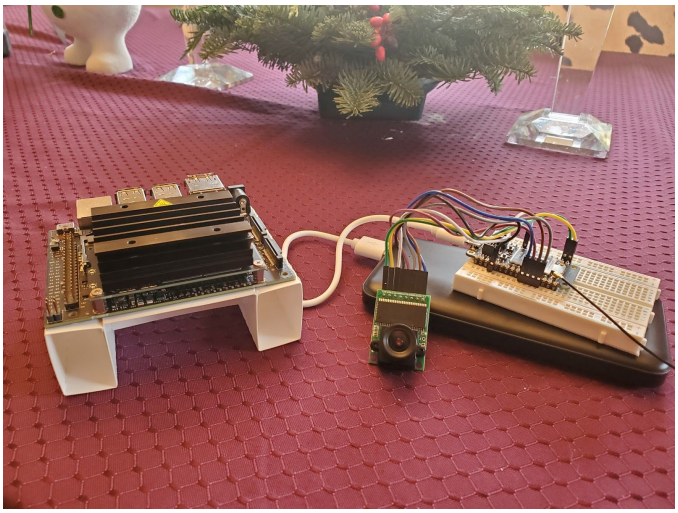


Fig. 3. Image of the Central Node (left) and a Camera Node without housing (right)

IV. Design Trade Studies

A. Camera Node

One design trade off that was made for the Camera Node was for the camera module. Ideally, we would like the highest resolution image possible, but we had to balance this with both cost and time to upload to the Central Node. We chose the OV5642 Camera Shield because it fit within our budget for each Camera Node and because the 5MP resolution it provides allows us to capture sufficiently high resolution images for the Object Detection Model and be able to upload the image over Wi-Fi at fast enough speeds to meet the upload time requirement. Another design trade off we made was choosing an MCU base for each Node. We decided on the Particle Argon because it comes with an integrated Wi-Fi chip and has sufficient processing power for the work it needs to do.

B. Central Node

One design trade off that was made for the central node was using an NVIDIA Jetson Nano instead of using another device like Raspberry Pi or INTEL NUC or NVIDIA TX2. One of the reasons is because of the use of image processing on the central node. As we work with the object detection algorithm in the cloud, we will have to determine what sort of image data pre-processing is needed to present the data to the model in an efficient and correct way. Image processing is generally faster on an NVIDIA Jetson Nano than on a Raspberry Pi, so any uncertainty with processing power or time will be accounted for here. Another reason for choosing the Jetson Nano over other NVIDIA devices like the TX2 or Jetson Xavier was due to a combination of portability and cost. The Jetson Nano is a relatively small computing device that costs around $99, which is much cheaper than the other NVIDIA alternatives. We only had a $600 budget in total, so getting something more expensive would not be ideal especially if we needed replacements.

The main reason we chose the Edimax 2-in-1 Wifi and Bluetooth Adapter is because there has been previous projects and records of it being used with a Jetson Nano and working properly. When it comes to using external devices, the computer needs to have the correct drivers available if not installed already with the operating system. Since the NVIDIA Jetson Nano does not have the same kind of processor as a personal laptop or desktop, the operating system and existing drivers may not be compatible, so it is important to use an external Wifi adapter where it is more or less confirmed to work by another individual. The Wifi adapter also has enough bandwidth and network speed for our purposes too, which was a necessity.

C. Cloud computing

One of the design trade offs that we needed to make for cloud computing was the size of the instance. AWS instances usually come in xlarge, 2xlarge, 4xlarge and 8xlarge sizes, with each one being roughly two times faster and more expensive than the last. In our preliminary research, we found that the machine learning algorithm chosen was designed to run quickly on a small graphics processing unit (GPU). So, the smallest instance was sufficient. Additionally, in our application, we would need to constantly run the model over an extended period of time. The cheapest option allows us to run the model for the longest time given a fixed budget. Therefore, the cheapest option, which is xlarge in this case, is the most optimal.

V. System Description

As mentioned previously, the system is divided into three parts, camera nodes, a central node and the cloud. The camera nodes take pictures periodically and send them to the central node, which then aggregates the pictures, performs preprocessing and sends the final product to the cloud. Then, object detection runs on the cloud to parse information, and display it on the website.

A. Camera Nodes

Each Camera Node consists of three hardware components: a Particle Argon, an OV5642 Camera Shield, and a 5V 10,000mAh portable battery. We decided to use the Particle Argon primarily due to its integrated Wi-Fi module, capable of reaching speeds up to 100Mbps, more than enough to meet the design requirements for upload speed. There will also be two software components: a synchronization protocol and image data conversion. Each Camera Node will be mounted high on a ceiling or wall to provide a bird's eye view.

The main purpose of each Camera Node is to capture images of the area from a bird's eye view and upload them to the Central Node. Since the Camera Shield was originally developed for Arduino platforms [4], the library had to be adapted to work for the Particle Argon. An adapted library was found for the Particle Photon [5] and was further adapted to be fully functional for our purposes. The Particle Argon is programmed with a synchronization protocol capable of synchronously taking images with the other Camera Nodes, such that images are taken and fed to the Central Node with minimal discrepancy between different Camera Nodes. The Particle Argon will also have lightweight data conversion

programming to convert the image data from the camera module to a form that the Central Node wants to receive over TCP.

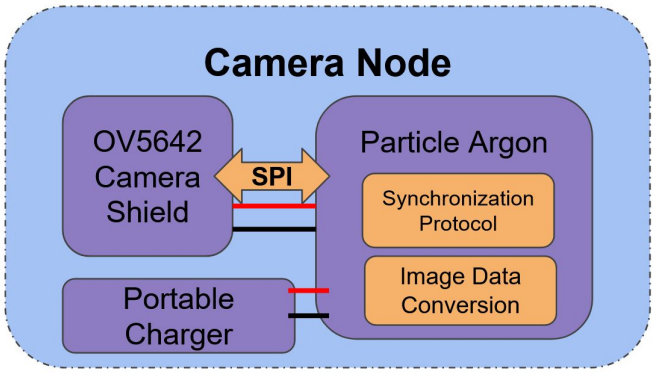


Fig. 4. Zoomed in Block Diagram of a Camera Node

The other purpose of each Camera Node is to survive at least 72 hours on its battery life. To achieve this, the synchronization protocol is designed to keep the Camera Node active only when necessary to avoid busy waiting and prolong the battery life.

B. Central Node

The central node consists of two hardware components: NVIDIA Jetson Nano and Edimax 2-in-1 Wifi and Bluetooth Adapter (EW-7611ULB). We decided to use the Jetson Nano instead of another device like a Raspberry Pi because running high-computational image processing we would use before sending the images to the cloud would have a better performance output on a Nano. The Edimax 2-in-1 is a known adapter to work well with the Jetson Nano drivers and has a network speed of 26 Mbps, which is enough for the project since we require 5-6 Mbps to adhere to the design requirements.

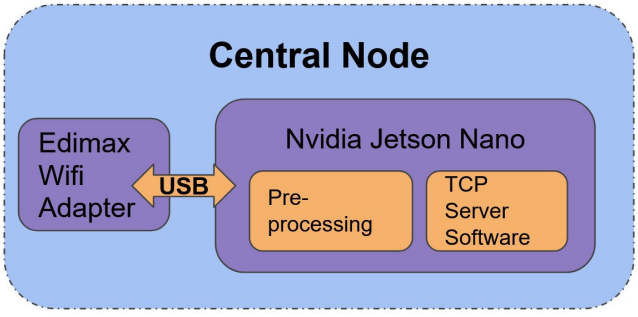


Fig. 5. Zoomed in Block Diagram of the Central Node

The main purpose of the central node is to receive packets wirelessly from camera nodes and send packets wirelessly to the cloud. The Nano has server-side socket programming that receives TCP packets synchronously and concurrently from the four camera nodes. The reason that TCP was chosen over UDP is because with TCP, packet loss is not an issue. Even though UDP offers a faster alternative to TCP, there still lies the risk of packet loss, and we require the central node to take a maximum of 30s of processing time. The speed of TCP suffices in meeting our requirements. For concurrency, we use an iterative selection based model where the central node scans connections to see which ones have data to be read from and iteratively reads data from each connection that is ready to be read. We designed our own network application protocol to enable proper synchronous communication between the central node and camera nodes. The central node will communicate with the cloud and send data packets via HTTP requests since the cloud hosts the website as well.

Another purpose of the central node is to conduct image pre-processing on packets before sending image data to the cloud. This includes any pre-processing that would make the data presentation to the Convolutional Neural Networks (discussed in section C) sufficient as well as image stitching. Image stitching is an algorithm that takes multiple images and connects them together to create one image that acts as a map of an entire area. For the purpose of Smart Library, it will be used to combine 4 images into one overall map of a public space. This will help make the AWS model run smoothly.

C. Cloud

The cloud runs two pieces of software, a machine learning model and a website. Cloud computing was chosen because it enables renting computing resources. This is especially helpful with machine learning models like the one we use in this project. Cloud computing gives us cheap access to a Graphics Processing Unit (GPU), which is required for machine learning.

The specific cloud computing resource we chose for this project is Amazon Web Services (AWS) and Google Colab. Google Colab offers free access to a GPU, making it good for development of the machine learning model. However, Colab is also inconsistent in giving its resources, so we will have to move the project to AWS for the final stages of development and for deployment. AWS offers access to an NVIDIA T4 GPU with the g4dn.xlarge hardware instance. This instance costs $0.526 per hour and allows us over 175 hours on the budget allocated for cloud computing, which is more than what is required for the project. Additionally, g4dn.xlarge has a fast network speed of 25 gigabits per second, which reduces latency when communicating with the other parts of the

project. For software, the Ubuntu Deep Learning AMI will be used, because it comes pre configured with common deep learning libraries like PyTorch and OpenCV.

The cloud runs a machine learning algorithm which was originally to be adapted from the YOLO model [1], because it is an object detection model that could identify seats. However, in practice this model was found to be ineffective, because of the difference in the datasets used. The YOLO model was pre-trained on a set of naturally occurring images. However, for the purpose of this project, the model needed to be run on aerial view images. As a result, the pre-trained YOLO model did not generalize well. This would have been the case with other pre-trained models as well. In order to overcome this, we used convolutional neural networks trained from scratch. Since they are completely trained on our dataset, they would give us more freedom to control the kind of data being input. However, they would also require more data and be less sophisticated than other pre-trained models.

The updated convolutional neural network was trained as a classification model as opposed to an object detection model. This would not identify the locations of vacant and occupied seats, but instead given the locations it can identify whether a person is sitting in the seat. This compromise was made because an object detection model would have required much more data than was available and collecting the data would have taken a lot of time. Additionally, this worked well with the setup of the hardware. Seating locations and the cameras were in a relatively fixed position, so inputting the locations of seats during deployment was a feasible option.

Layer Number	Downsampling Kernel	Number of input channels
1	2x2	3
2	2x2	4
3	2x2	4
4	2x3	4
5	5x5	4
6	5x5	4
Final	Sigmoid	1

Fig. 6. Architecture and Layer Description of the Model

The above table specifies the architecture of the model. The amount of data available was the main constraint that drove the design choices for the model, since we were only training the model on data that we collected. The model accepts a 400x600 pixel image, which is then fed into the six numbered layers. Each layer would downsample the image with a specific kernel so that at the end it is reduced to a single scalar value, representing the probability of a person being in the picture. Each numbered layer consisted of a depthwise convolutional layer, a 3x3 grouped convolutional layer, a ReLU activation and a max-pooling layer. Splitting the convolutional layers into a depthwise layer and a grouped layer is a common practice in machine learning to reduce the number of parameters needed to be trained. It was followed by a max pooling layer for downsampling, which was chosen over a convolutional layer due to the fact that it did not contain any parameters and would therefore require less data to be trained. A total of four channels was chosen for all the layers except the final layer, because of the limited data. The final layer had only one channel to reduce the output to a scalar value, which was then passed through a sigmoid function to be in the range of 0-1 in order to represent a probability.

VI. Project Management

A. Schedule

Our Schedule contains four primary task groups: Camera Nodes, Central Node, Object Detection Model, and Website. By the nature of the project, the hardware must be complete before the Object Detection Model can be properly trained, thus the Camera Nodes and Central Node task groups were tackled first, concurrently. After Camera Nodes had achieved its milestones, work on the Object Detection Model task group began in full force. The Website task group is designed so that tasks may be worked on between gaps in other task groups. All task groups were scheduled to be completed by the last week of November.

We initially encountered setbacks due to shipping delays of our components since the initial phases of our project relies heavily on having working nodes to provide our ML algorithm trainable data. Since our original schedule was aggressive, with a final milestone scheduled for the end of November, this shifted our timeline and had us finishing by the end of the semester. Despite difficulties revolving around being so far apart and vastly different time zones, we were able to finish all of our respective tasks on time.

B. Team Member Responsibilities

Each team member took on primary responsibility for a task group. Pablo took the lead on the Camera Nodes, Arjun took the lead on the Central Node, and Krish took the lead on the

Object Detection Model. Secondary responsibilities include integration between task groups and the Website task group. Pablo and Arjun worked together to develop the communication between the Camera Nodes and the Central Node. Arjun and Krish worked together to develop the image pre-processing and transfer to the Object Detection Model from the Central Node. Krish and Pablo worked together to find best image capturing practices to feed to the Object Detection Model. Krish worked on the Website in the gaps between Task Groups while awaiting the completion of the Hardware tasks.

C. *Budget*

Our project came in under budget due to changes since the design report. We changed to portable chargers rather than LiPo batteries and a voltage boosting circuit which allowed us to be well under the $100 budget we had set for each Camera Node. Additionally, AWS credits became a non-cost. These made up for going slightly over budget on the Central Node and purchasing no longer needed LiPo Batteries by a wide margin.

System Subgroup	Component	Cost	Quantity	Total
Camera Node	Camera Module	39.99	4	$ 159.96
Camera Node	Particle Argon	27.92	4	$ 111.68
Camera Node	Portable Charger	17.99	2	$ 35.98
Camera Node	LiPo Battery x3	9.34	1	$ 9.34
Central Node	Jeston Nano	99.00	1	$ 99.00
Central Node	Wi-Fi Adapter	12.34	2	$ 24.68
Cloud	AWS Credits			$ -
Total				**$ 440.64**
Camera Node Total				**$ 76.91**
Central Node Total				**$ 111.34**
Cloud Total				**$ -**

Fig. 7. Budget Table (Portable Charger should read x2)

D. *Risk Management*

The way we mitigated design risk was to have a generic system architecture with clearly defined needed features. No single aspect of our design is locked to a specific component; our components were chosen solely due to the fact they could achieve the features we wanted for the cheapest price and/or easiest integration. For example, our Central Node can be replaced by any hardware component(s) that can connect to the internet and do image processing and our Camera Nodes can be replaced by any hardware component(s) that can capture images and transmit them to the Central Node. A goal of our design was to be able to be as flexible as possible, so that it should be near arbitrary to scale up or swap components.

The way we planned for schedule risk was by setting ourselves with an aggressive timeline so that we would finish early if everything went to plan. We mitigated risk of unanticipated delays this way by allowing our schedule to shift by a couple weeks and still be on track to finish before the deadline. Additionally, we split our tasks into groups so that we can easily adapt and adjust when any one task gets behind schedule, as well as having primary task groups be staggered, so that there is always a team member available to assist if needed.

We managed resource risk by ordering duplicate parts and avoiding purchasing all parts at once. Even though our design for a fully functioning node network contains at least four Camera Nodes, our design will still function with just one, albeit with less area covered. Furthermore, full functionality can be tested with just two Camera Nodes, so we initially only purchased enough components for two Nodes in case we ran into a design problem and had to pivot to a different design for our Camera Nodes. For our Central Node, due to the high cost of the Jetson Nano, we were not able to purchase duplicate nodes. However, we only purchased one of the Wifi and Bluetooth Adapter, so that Arjun could confirm the part worked for our design before Pablo purchased one to be put into the Node Network on campus.

VII. Related Work

One related project was done in Fall 2019 of CMU Capstone 18500 by Team A4. The project was called Carrel Corral. The main goal of this project was to find available study carrels on the third floor of Hunt Library on CMU campus. Their approach was to have a system of ultrasonic and infrared sensors at each carrel connected to a microprocessor to determine whether there was a person there, as well as have a mobile app that can communicate this information to possible student users. They also considered the case where a student places their backpack at a carrel but does not actually sit there for a long period of time.

VIII. Summary

A. *Future work*

Although this project is part of a semester-long course, we might continue working on it after the semester ends. Throughout the semester, we plan on demonstrating the project in Sorrell's Library at Carnegie Mellon University. However, this can be extended to any general space. One particular type of space is restaurants, who might want to monitor seating availability in order to automatically assign tables without the need for a host. This would require modifying the system to fit into multiple different

environments. Specifically for object detection, data would have to be collected at a much larger scale from multiple locations in order to build a model that can generalize more easily.

Another modification that could be made is integration with existing hardware. It is common for public spaces to have security cameras. For installation in such a space, we could remove the requirement for our own cameras in lieu of the security cameras that have already been installed. This would require a change in hardware along with compatibility with the specific security system of the space. The advantage with this method is that our system would be less invasive since it reduces the requirement of installing additional hardware.

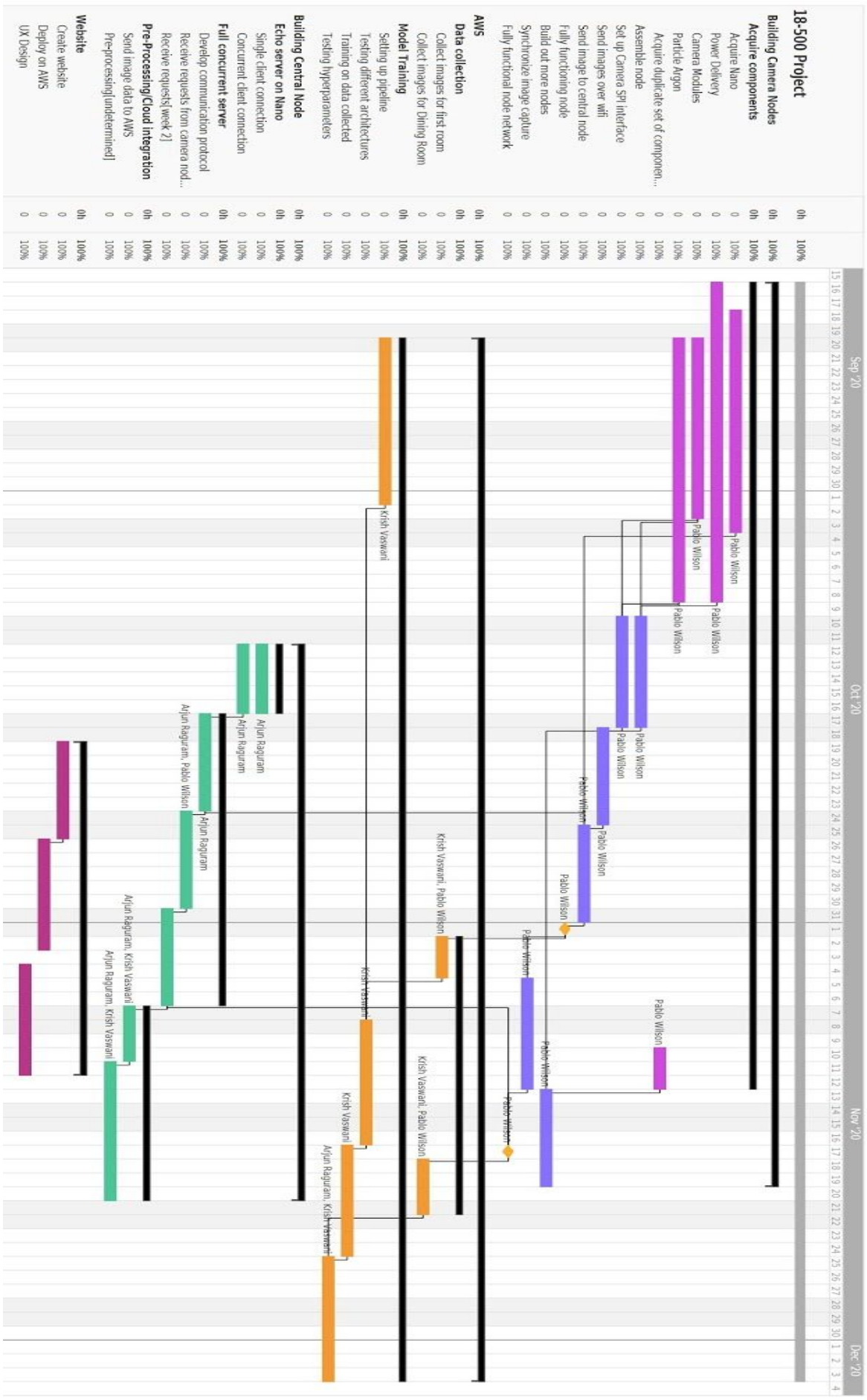
B. Lessons Learned

Throughout the semester, we learned how to deal with issues related to the COVID-19 pandemic. First, all three members of the project were in different locations and time zones, so we had to figure out an adequate communication method. There were not many hours that overlapped between our timezones, but on meeting days, all three of us would get on a call together and discuss information for those few hours. Also, in order to set up test equipment in a public place such as Sorrells Library in CMU, you have to go through a formal process to get permission to do so, as well as adhere to the specific permissions one may or may not be given. Not doing this could result in a dilemma due to privacy concerns of those in the public place.

REFERENCES

[1] You Only Look Once: Unified, Real-Time Object Detection, https://arxiv.org/abs/1506.02640v5
[2] National Semiconductor Inc., www.national.com.
[3] CMU 18500 Capstone Team A4: Carrel Corral, http://course.ece.cmu.edu/~ece500/projects/f19-teama4/proposal/
[4] Arducam Library: https://github.com/ArduCAM/Arduino
[5] Arducam Library adapted for Particle Photon and example code: https://github.com/jbrichau/particle_arducam

IX. Schedule Diagram



X. Expanded Block Diagram