Thermonitor

Author: Minji Kim, Iris Wang, Jiamin Wang: Electrical and Computer Engineering, Carnegie Mellon University

***Abstract*—**Thermonitor is a smart, contactless thermometer that can be installed at gateways to different locations across buildings for large organizations, such as educational institutions and corporations, to remotely monitor their community members' temperature and facilitate contact tracing. The display of the Thermonitor will alert the users to scan their ID, properly wear a mask, and measure their temperatures once they are correctly identified as a member of the organization. Then the Thermonitor will collect the temperature data accompanied with one's profile detected by their RFID tag. This data will be uploaded to a web application, which will match the temperatures to individual profiles and log the records to allow organization admins to access and manage the logs.

***Index Terms*—Azure, Computer Vision, Face Detection, Internet of Things, Machine Learning, Mask Detection, NFC, RFID, STM 32, Thermometer, Web Application**

I. Introduction

Currently, COVID-19, a novel acute respiratory illness is plaguing countries all over the world. Infections are rising, with no view of a vaccine in sight. One of the earliest warning signs of infection is fever. It is critical to regularly monitor body temperature in order to protect others, especially for diseases like COVID-19 where one is contagious several days before showing any symptoms at all. By noticing changes in one's own body temperature, we can take immediate measures to prevent further spread of the virus.

Thermometers in the current market are either standalone kiosks or handheld. Handheld thermometers require another person to hold the thermometer from a distance of up to 4 inches, which fails to conform to the mandated 6 ft for proper social distancing. Standalone kiosks are currently over $2000, which can be unaffordable to be installed at every gate way. Currently, there is no viable product that is both safe and affordable; Thermonitor aims to be both. Thermonitor is an end-to-end device where the user presents a valid RFID tag that triggers the start of temperature measurement. A monitor is used to broadcast the video stream back to the user with a bounding box that surrounds the face. The video stream is displayed at 7 frames per second (FPS), which is the rate at which images are shown on a display, to ensure viewing smoothness on the monitor. Thermonitor must measure the temperature within a distance of 50 cm (19.6 inches/1.64 feet) and this temperature is sampled multiple times. An 85% facial detection and 95% mask detection accuracy will be obtained. These algorithms ensure that the user is properly wearing a mask and alert them otherwise.

II. Design Requirements

The first requirement of Thermonitor is a capability to correctly read the RFID tag presented. This is necessary for us to send information from end-to-end on our device. It also serves as an indicator to know when the device should start scanning for faces and measuring temperatures. Secondly, it should be able to accurately measure the temperature of an individual standing in front of the device. It must also be able to properly detect faces in front of it using a bounding box to check whether or not the person is wearing a mask. All of this information must then be sent to our IoT platform for gathering and monitoring of data.

To verify that our design has met these specifications, we are performing several benchmark tests. We purchased sample RFID tags for testing the scanning functionality. The official STM NFC Tap app was used to read the unique string ID (UID) to confirm that both type 4 and type 5 ID tags were able to be properly scanned and recorded. We are aiming for 99% accuracy since misreads could occur occasionally.

To test the facial detection algorithm, we are putting both people and objects in front of the camera and seeing if their eyes or faces are detected by checking if a bounding box appears. In addition, we are testing faces with and without face masks in order to ensure it can still recognize people with only eyes as the tracking feature. If there are multiple people in the frame, the algorithm should only detect the main person in the front of the camera, the one who actually scanned their RFID. We are aiming for a 85% accuracy rate since the facial detection algorithm normally has a 95% accuracy, but since we are detecting it in real time video stream and we can only use a limited number of classifiers, we are lowering our accuracy rate. We are aiming for 95% mask detection rate since this algorithm is only being called after we determine there is a face. We are not aiming for 100% since we have a limited training set for mask detection.

To test temperature sensing, we are using various objects with different temperature ranges to test our degree measurement. We are aiming for ±0.2 °C from the intended temperature. Calibration with a real thermometer may be necessary if our sensor temperature measurement is off. Since sensor data needs to be transmitted, calculated, and displayed on the monitor, we are aiming for a 3 second time measurement. Handheld thermometers usually take around 1.5 seconds to display results and taking into account the processing time on the Jetson, we decided that 3 seconds is a good benchmark.

To test the FPS, we will be displaying the number of actual displayed frames from the running script. We are aiming for 10 FPS as that is a standard output within the frame rate capability of the camera model that we are using. We do not need an extremely high FPS since we are simply displaying the

processed video stream on the monitor and will be analyzing a stationary target.

On the cloud side, we are making logs accessible to the admin and ensuring that the correct profile is mapped to the RFID. We are using the sample RFIDs to test how easily accessible our external platform will be. To fully test our web application, we are conducting user testing on friends and family and adjust based on their feedback. We will have user surveys that we will periodically send out and adjust our web application accordingly. We are using MQTT and TCP protocols that already ensure reliable messaging. However, we still should account for unexpected data transmission issues, thus we are aiming for a 99% data transfer rate.

III. Architecture and/or Principle of Operation

We are using a microcontroller unit (MCU) to handle the logic behind our tag id verification. More specifically, we are using a STM32 Nucleo-L476RG board with a compatible expansion board X-NUCLEO-NFC05A1. The near field communication (NFC) expansion board is directly connected on top of the Nucleo board, and the boards are powered through the micro-USB port. We originally purchased the X-NUCLEO-NFC02A1 expansion board, but it lacked the library capabilities to extract the UID of the tag or device scanned. In the given middleware, the version 2 board only had the NDEF library while the version 5 board had both the NDEF and the RFAL library. With the RFAL library, we were able to write an FSM that provided the necessary logic to poll for a RFID scan and process the string before sending information to our Jetson Nano. As seen in Figure 2, if a RFID tag is read, then this immediately triggers the transmission of the UID string from the Nucleo board to the Jetson Nano. Thus, the Nucleo board is the slave device and is wired directly to the Jetson, the master device, to allow for serial communication.

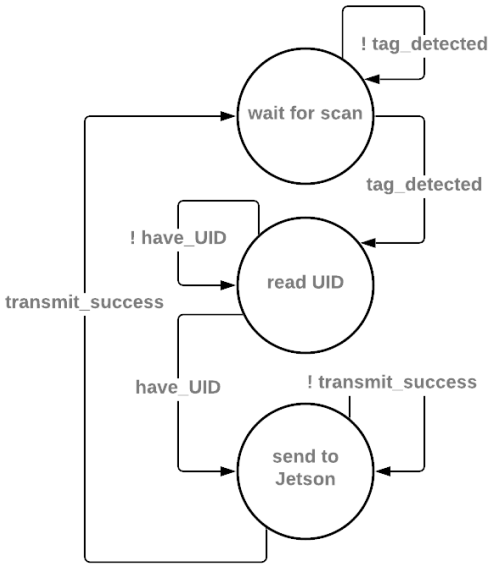

Fig. 2. FSM for RFID logic.

The Jetson is always polling for an incoming RFID transmission from the Nucleo board. Once a RFID has been received on the Jetson, a LED will turn on for a couple of seconds, green for indication of valid RFID string and red for invalid RFID string.

An IR sensor is used to scan for multiple accounts of temperature. The Raspberry Pi camera module's video stream is fed directly to the Jetson to begin facial detection and then mask detection. Initially, we planned on using OpenCV and TensorFlow for the algorithm, but due to Jetson memory issues and processing lagginess, we used YOLOv3 [1], an object detection algorithm that is time efficient and optimized for running on smaller devices.

As seen in Figure 3, the user must stand within the grey oval for an accurate measurement. A bounding box will indicate whether the user has their mask on, mask off, or if the mask is

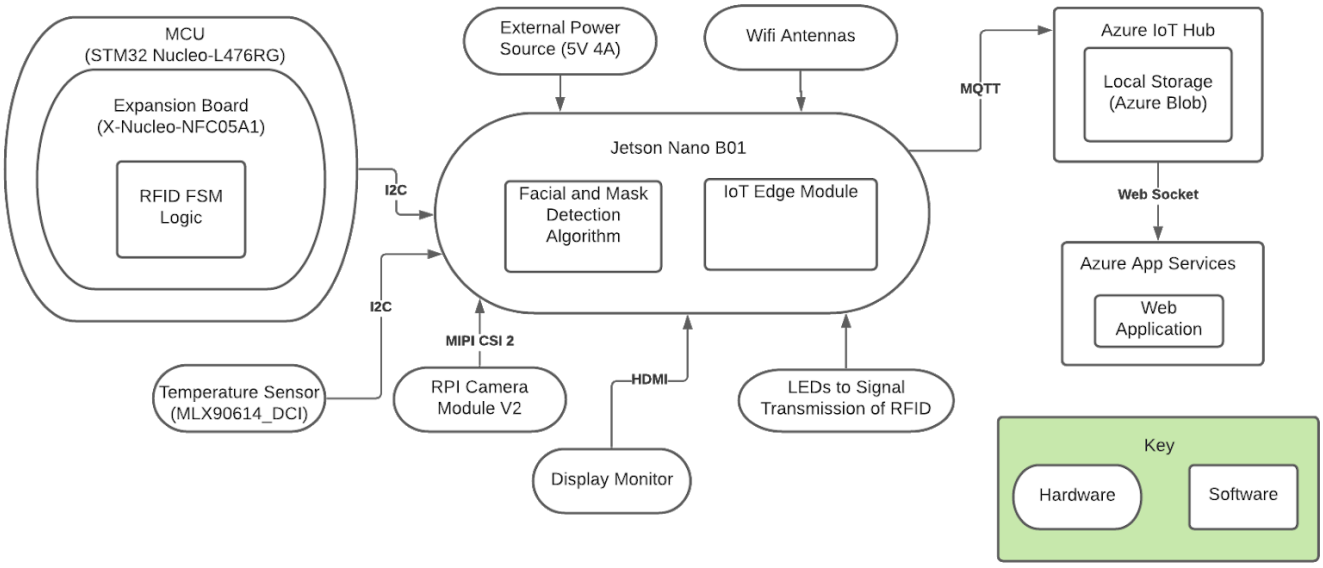

Fig. 1. Overall block diagram of our system.

worn improperly (e.g. the mask is only covering the mouth and not the nose). After a properly worn mask is detected, temperature taking will commence. Our temperature sensor is able to poll extremely quickly, thus removing the need for our limitation of having the user stand still for 3 seconds while the temperature measurement is happening. We are able to take 20 samples within 1 second. The temperature is then displayed on the monitor -- green for pass and red for fail. We are using a fever threshold of 38˚C, in accordance with CMU health guidelines.

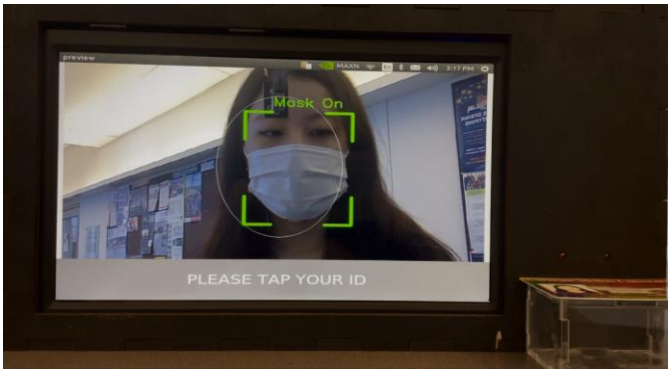

Fig. 3. Monitor display setup from the user's perspective.

After a successful temperature measurement, the RFID and the temperature will be sent to the web application. The Jetson Nano uses MQTT protocol to transmit user profile data to the cloud. This data is received by the Azure Hub, which is the cloud gateway for our IoT system. After unpacking the transmitted data and formatting it in the desired form, the data is sent to the backend of our web application. This web application is hosted on Azure App Service, where they provide APIs to facilitate data transmission between Azure IoT Hub and custom applications.

The web application is accessible through a desktop browser. We initially planned on creating a mobile application, but after implementation, we realized that the main piece of information to be displayed on the app is the list of logs in a table form. We created some mock-ups of the interface and concluded that a web application for desktop browsers was more adequate. Figure 4 provides an example of our web application wireframe.

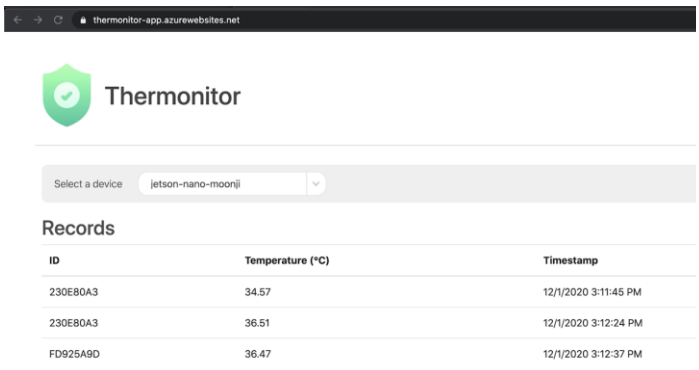

Fig. 4. Web application wireframe with individual records.

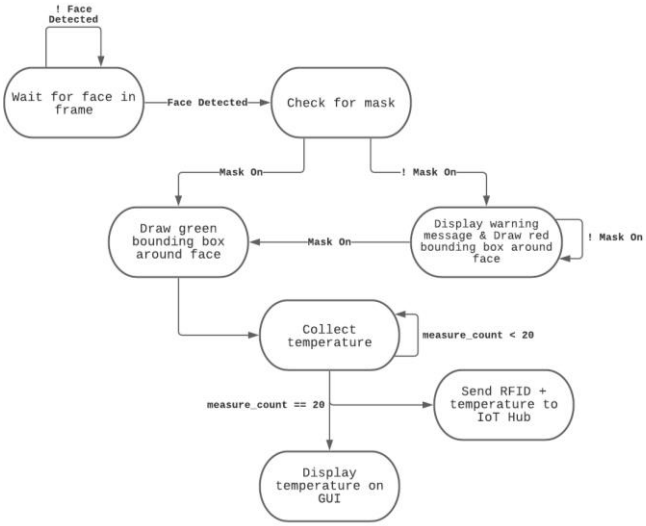

Fig. 5. FSM for overall system.

IV. Design Trade Studies

These are some components and design trade-offs we took into consideration when choosing which best fit our system.

A. Temperature Measurement

From our previous design, we decided to change our temperature measuring hardware from a FLIR IR camera to a MLX90614_DCI medical grade IR sensor. We made this change since IR cameras within our price range had around 2 degrees of error, which is significant when referring to human body temperatures. More accurate IR cameras ranged from around $1000 to $2000, which exceeds our budget. The IR sensor in contrast measures temperature with an accuracy of ±0.2 °C. It measures the surface temperature by detecting infrared radiation energy and wavelength distribution. The sensor can detect object temperatures from a range of up to 50cm, which still conforms to our project goals of ensuring safe social distancing temperature measurement, since our kiosk does not require human operation.

For temperature validation, we verified our IR sensor against a handheld IR thermometer. Although the IR sensor was factory calibrated, the actual temperature measured had a consistent offset of around 2.4 °C. The graph in Figure 6 shows the calibration measurements we took to get the average offset. The average difference was added to our final IR sensor reading for calibration.

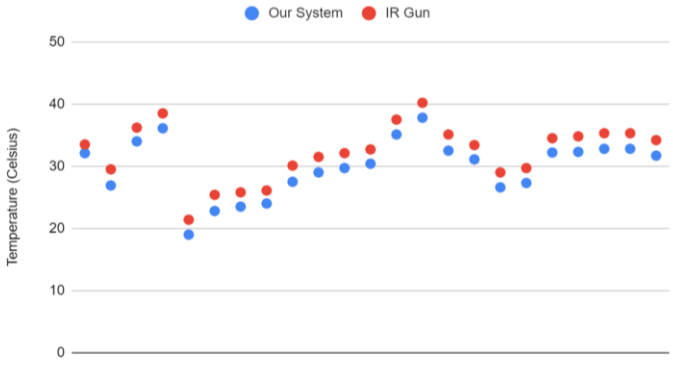

Fig. 6. Handheld Thermometer vs. IR sensor temperature measurements.

B. *Serial Communication Protocol*

We evaluated different serial communication protocols to determine which one is best suited for sending RFID tag ids from the Nucleo board to the Jetson Nano. The two that were under consideration were I2C and SPI. We ended up choosing I2C over SPI because it supports multiple masters to multiple slaves on the bus, making the project more scalable. In addition, I2C has ACK and NACK bits to support error handling, which the SPI protocol does not have. Our primary form of identification is done through the RFID tag ids. Thus, error handling is necessary to prevent inaccurate storage of information and ensure that the correct identification is stored in our local and cloud systems. We were able to achieve our desired 100% RFID transmission rate by scanning multiple RFIDs and receiving them on the Jetson Nano.

C. *Microcontroller Unit*

We originally planned on using an Arduino, Raspberry Pi, or FPGA as our main processor for the RFID verification process. We ruled out the FPGA early on because the functionality we wished to achieve was not able to fully utilize the capabilities of an FPGA. From getting some feedback from the Teaching Assistants, we pivoted towards using an MCU. The first board that came to mind was to use an Arduino since we have previously worked with this microcontroller in past classes. We also considered using a Raspberry Pi but ended up ruling that out. This is discussed in the next section. In addition, Arduino and Raspberry Pi boards are more for experimenting instead of creating an end-to-end prototype. We wanted to target our product to be in the industrial setting, and thus we ended up moving towards using an embedded board. The STM32 series seemed promising since it seemed to be a popular choice with various expansion boards. We wanted to aim for a low power series that would also be compatible with an NFC/RFID expansion board, so we ended up choosing the STM32 Nucleo L476RG.

D. *Jetson Nano*

We chose the Jetson Nano as our core processor over the Raspberry Pi because of its high performance optimized for object detection and image processing while being lightweight. We determined that the Jetson Nano would be the better choice since it could process around 15 frames per second versus the Raspberry Pi's 1 frame per second. With the higher frame processing speed, we would be able to recognize the face and mask faster and be able to match our goal of scanning a person's temperature at a rate fast enough to compete with handheld thermometers.

E. *Face and Mask Detection Algorithm*

We started off implementing our Face and Mask detection algorithm with Haar Cascades using Viola Jones Method. We used eye classifiers in order to detect a face, and if there existed one, checked if the user was wearing a mask if the nose classifier was present. This turned out to be a very inaccurate approach. We were achieving around 40-50% accuracy since we were only able to use a small number of classifiers. For regular detection algorithms, they use around 38 strong classifiers for different features on the face. In addition, the analyzed results lagged in real-time, since the processing took longer than expected. This is because the algorithm can't be run in parallel due to dependencies on previous processing iterations. This caused our FPS to drop since the algorithm had to process a frame, and then display it.

The second approach we took was with TensorFlow. We tried using pre-trained models in our algorithm to improve the accuracy of the algorithm, but the Jetson Nano did not have enough compute capability to run the models. Using the pre-trained models froze the screen of the Jetson Nano and indicated "Out Of Memory" warnings.

Finally, we pivoted to using YOLOv3, which is a real-time object detection framework that is optimized for running on devices with limited computational capacity, like our Jetson Nano. This proved to be a much faster approach since it scans the entire image at runtime as opposed to the Viola Jones algorithm, which performs multiple scans of the entire image. YOLOv3 was the best choice for our use case since it struggles with detecting small objects, but since we are ensuring that the user's face is always a certain size, it has no problems detecting it. YOLOv3 also utilizes trained models, which we trained with datasets [2] of masked, no masked and incorrect masked faces. We had to keep supplementing some of the datasets in order to meet our accuracy metrics. However, trained models definitely boosted the accuracy of the face and mask detection.

We tested our algorithm by having multiple people stand in front of our machine and gathering data on the face and mask detection rate. We also used pictures of humans and non-humans since it was not plausible to get a big enough sample size of real people due to the pandemic. The total sample size we used is 20.

The face detection exceeded our expected metrics with an accuracy of 90%. We also achieved 95% accuracy for correctly detecting an individual with mask on and mask off, which is what we desired when going into this project. However, we only achieved a 60% accuracy for incorrect mask usage. This is due to not having enough training data with people wearing their masks incorrectly. The detailed metrics data collected describing each false-positive and false-negative scenario are presented in Table 1.

TABLE I. FACE AND MASK FALSE DETECTION METRICS

Expected Result	Actual Result	Rate (%)
Mask On	Mask Off	0
Mask On	Mask Improper	0
Mask Off	Mask On	0
Mask Off	Mask Improper	5
Mask Improper	Mask Off	5
Mask Improper	Mask On	40

F. User Experience

We scratched out the idea of using a wake up signal. Originally, we wanted to use the signal to decrease the power consumption of the Jetson Nano. For example, if the Jetson has not received an RFID transmission for 10-15 minutes it would go into idle state. But when a new user comes to scan their RFID, they would have to wait for the Nucleo to power on the Jetson Nano. However, with the way that Jetson Nano handles disabling auto power on, booting the system backup could take up to 2 minutes. This is not ideal from a user's perspective because students or employees would want to quickly get into a room without wasting any of their valuable time. Thus, we decided to take out the wake up signal functionality and prioritize creating a user-friendly interface.

Through testing, we also realized that a higher resolution video output stream resulted in a lower FPS. Originally, our video stream had around 4 FPS and every time the user moved; the response would occur 3 to 5 seconds later. After some user testing, we noticed that the difference in resolution was not as noticeable to the user as the difference in FPS. As a result, we concluded to prioritize FPS over the resolution and settled with 800 by 600 pixels. We also found that we were able to achieve 10 FPS, our desired metric if we created a smaller window size. Even though we desired 10 FPS, we decided that having the video stream displayed on the entire monitor is necessary to improve user experiences. We were able to achieve 7 FPS and the response would occur within 0.5 seconds of any movement.

G. Microsoft Azure

Both Azure and AWS have a cloud service that enables reliable communications between IoT applications and connected devices. We picked Azure over AWS since there is more documentation on using Azure IoT hub with hardware devices like ours, and since none of us have any experience with IoT, felt more comfortable having existing documentation to reference when necessary.

We were able to achieve our desired 100% message transfer rate from the device to our web application. This was done by monitoring all sent messages and ensuring they were reflected on our application. In addition, our user experience feedback surveys were highly positive, indicating that they were able to use the site easily. We received an average of 90% satisfaction rate for user experience and 100% satisfaction rate for overall website design.

V. SYSTEM DESCRIPTION

The project consists of a software and hardware component. On the hardware side, a microcontroller unit (MCU) was used to handle interactions with a RFID tag. The Jetson processes a video stream for facial and mask detection. This information will then be sent to the cloud using Microsoft Azure.

A. Hardware Components

The STM32CubeIDE is used to compile and download code to the microcontroller. The expansion board provided the drivers for X-CUBE-NFC05. Although the expansion board is compatible with the Nucleo board, necessary changes and modifications were made to ensure that the drivers worked properly. This took some time since some files were different and we had to look through individual files and make sure that the necessary files were placed in the correct folder path. UART and puTTY were used for print debugging.

Standard HAL drivers were downloaded that were specific to our board. The HAL library is used to provide generic APIs, such as initializing and configuring the necessary peripherals, managing data transfers, and managing communication errors. Middleware folders with NDEF and RFAL libraries were also provided. This served as the main documentation for working with NFC tags. An FSM was written to poll for an RFID scan, format the string, and transmit it one byte at a time to the Jetson Nano. The Nucleo board was configured as a slave device with a slave address in the I2C interface written.

LEDs were connected to the GPIO pins of the Jetson Nano to blink during transmission and hold for 1 second after transmission has been completed. We originally directly connected the LEDs with pull up resistors to the Jetson, but soon discovered that the brightness was very lacking due to low current. We followed a tutorial to ensure that we had the correct transistors and resistors needed to build the circuit [3]. Then, we purchased a pack of P2N2222 transistors to amplify the current supply to the LEDs. LEDs were put into our design after some user testing feedback, where the user wanted more visual affirmation that the RFID had been scanned.

To perform our facial and mask detection, we are using the Raspberry Pi camera module V2 as our main video feed. We chose this model for our camera as there exists a lot of documentation for the integration of these two components. This is directly connected to the Jetson through J13 for camera connector #1. The Raspberry Pi camera module V2 communicates with the Jetson Nano through the MIPI CSI-2 interface. It is a high-speed protocol used for point to point image and video transmission between cameras and host devices. We also installed a Wi-Fi card, Intel Dual Band Wireless-AC 8265, and Wi-Fi antennas onto the Jetson Nano for mobility instead of needing to tether it to a router switch. We can then connect the Jetson Nano to the cloud for our IoT platform. The video stream is broadcast back to the user through

a display monitor that is connected by an HDMI to HDMI cable on J6. The display monitor has a bounding box to confirm the face of interest, the measured temperature, and other necessary information. The IR temperature sensor, MLX90614_DCI, is connected to the Jetson through one of the GPIO pins on J41. The sensor uses the I2C interface to communicate, so we are connecting the corresponding SDA and SCL pins of the MLX90614_DCI to the pins on the Jetson Nano.

To communicate from the IR temperature sensor and the Jetson Nano, we originally relied on a built in library - i2cdetect. This library is supposed to identify the address where the connected temperature sensor device resides (e.g. 0x5a). However, after connecting it multiple times, nothing was showing up on i2cdetect. At first, we thought that the temperature sensor was broken. We tried troubleshooting by attaching it to a Raspberry Pi and Arduino, and realized we were receiving temperature measurements, so the sensor was working properly. We then theorized that our Jetson Nano i2c pins were faulty, so we tried connecting the temperature sensor to another Jetson Nano. However, that Nano also could not sense the temperature sensor. Finally, we tried simply reading bytes from the supposed address and started receiving data. We realized that it was a problem with the library and that our hardware components were working all along. We transitioned to a smbus Python library to help us read from the proper registers and started obtaining correct temperature data [4].

B. *Jetson Nano*

Jetson Nano serves as the software processor. It provides three main purposes — (1) Receival of RFID (2) Face and mask detection and (3) IoT connection.

i. *Receival of RFID*

The Jetson receives the RFID byte string from the STM32 Nucleo Board. On transmission, the Jetson sends a signal to either the green or red LED indicating a valid or invalid RFID. LEDs were added for the user to visually confirm that their RFID has been accepted and as an extra indicator that mask detection and temperature scanning should commence. The Jetson then displays “RFID SCANNED. Please hold still. Taking temperature…” to indicate that the user should place their face into the gray oval drawn on the screen. The oval is placed in the optimal position for the user to get their temperature measured, as it is aligned with the camera and the IR sensor. It is drawn to ensure that users are correctly positioned for our device to analyze them.

ii. *Face and Mask Detection*

The Jetson receives the video stream inputs [5] from the RGB Raspberry Pi Module V2 camera and runs the facial and mask detection algorithm on each frame. The detection algorithm is implemented in Python using popular computer vision libraries -- OpenCV and NumPy.

We are using YOLOv3, a fast object detection algorithm, which stands for You Only Look Once. It is named for being able to detect an object by scanning the image once at runtime, compared to most other object detection algorithms like Viola Jones which require multiple stages of processing. Only one forward pass through the neural network is necessary for detection. We trained the YOLOv3 model with datasets of people with no mask, improper mask, and correctly worn mask.

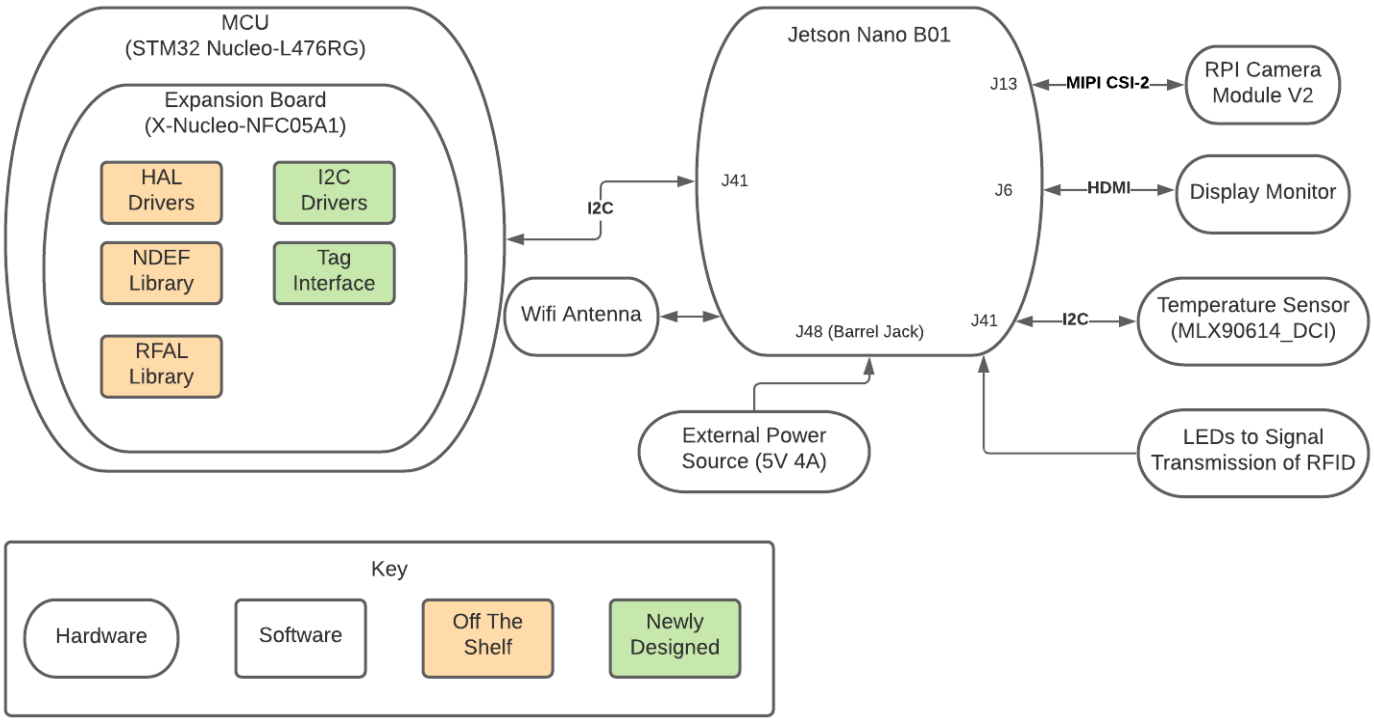

Fig. 7. This is the hardware block diagram for the MCU and Jetson Nano.

The single convolutional neural network (CNN) then scans the image once, predicts bounding boxes and the confidence levels for those boxes and then outputs recognized objects with the boxes.

We decided to add the limitation of having only one user recognized at a time, depending on the size of their bounding box. This is because of our use case -- since Thermonitor could be placed in an area where many people are walking by, we do not want the program to detect their faces and send their temperatures to the web application since the scanned RFID does not correspond to the people walking by. Only the target user should be identified. The largest person in the frame will be considered the target user, and the program will only base the mask detection and temperature sensing based on them.

The program will display the results of the detection above the detected face, which is outlined with a red or green box, depending on the results, that follows the user when they move. It will either output “Mask On”, “No Mask”, or “Improper Mask”. After detection of a mask, the program waits until the user is correctly wearing a mask and positioned in the gray oval before starting the temperature measurement. After a second, the temperature will be displayed on the screen, with the colored bar at the bottom displaying green (PASS) or red (FAIL) depending on the measured temperature. We are using a threshold of 38˚C for a failed temperature, as that is CMU’s health guidelines for a fever. Finally, the RFID and temperature string are then sent to the IoT hub for client viewing.

Fig. 8. Mask on and proper temperature

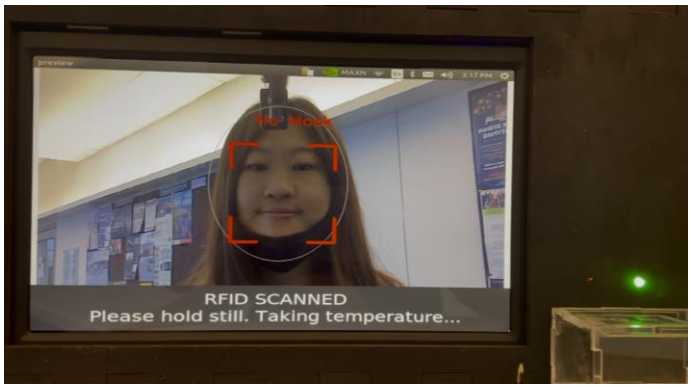

Fig. 9. No mask.

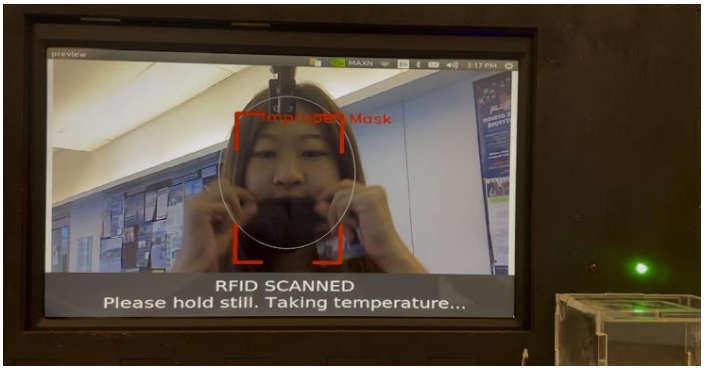

Fig. 10. Improper Mask.

Fig. 11. Multiple people in frame.

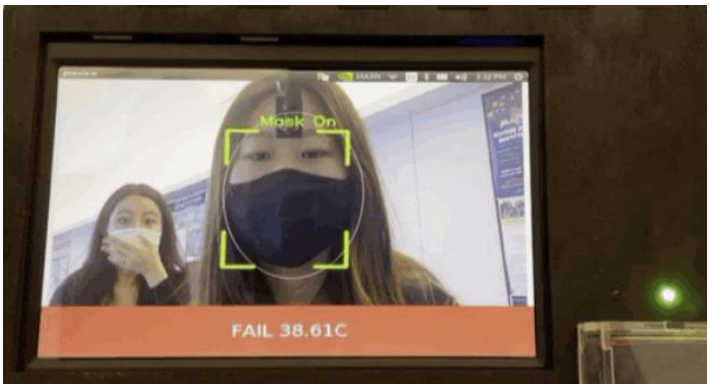

Fig. 12. Failed Temperature.

iii . IoT connection

For the IoT functionality, we are deploying the IoT Edge Module directly on the Nano. This allows the device to easily connect to Azure IoT hub, which is the cloud gateway that is compatible with IoT Edge devices. We are using the MQTT protocol to send messages to the IoT Hub. After we receive the messages, the Hub stores the information in a local storage, Azure Blob, to allow periodic bulk updates in case of internet connectivity issues. Then, the IoT Hub packs the data and sends it over to our Web Application’s back-nd service, which is also hosted by Azure. The back-end communicates with the front-end of the application to manage and display the collected temperature data. The back-end service is implemented using Node.js, while the front-end is built using HTML and JavaScript. Additionally, our database is implemented using

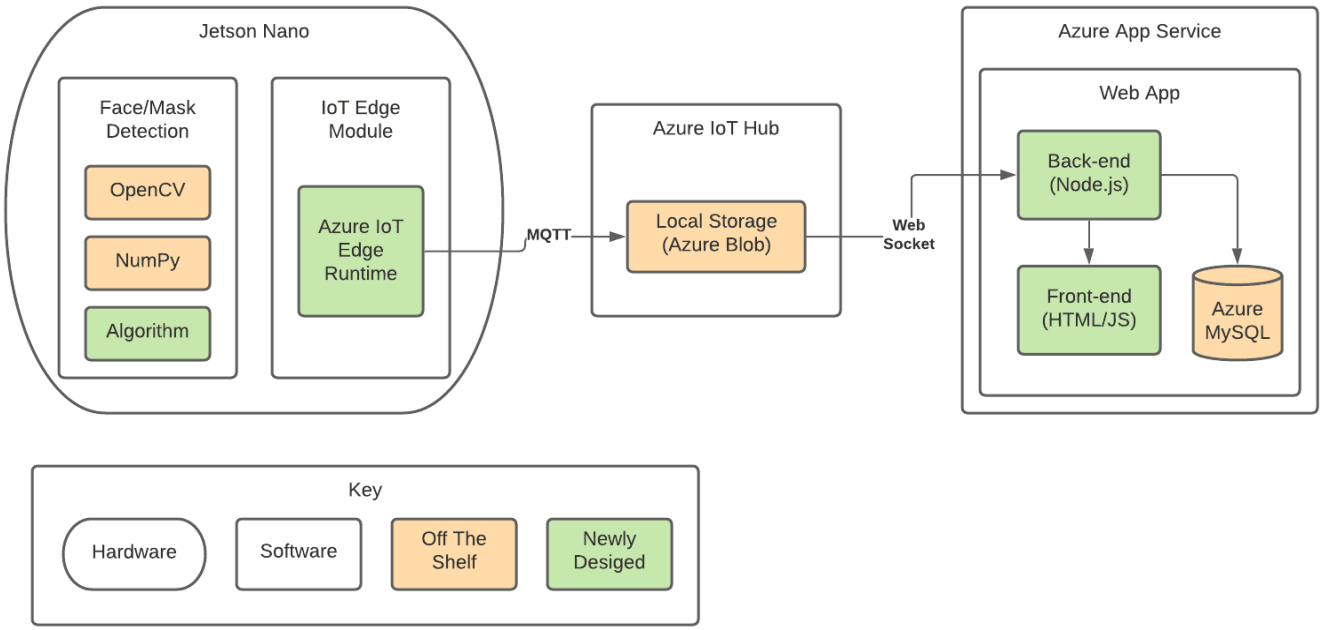

Fig. 13. This is the software block diagram for the Jetson Nano and IoT.

Azure MySQL. Finally, we are using Azure App Service to deploy our web application, since it easily integrates with the IoT Hub and provides a safe way to send and receive data through web sockets.

C. Enclosure

To make all our different components into a single product, we built a laser cut wood enclosure made in TechSpark. It houses all of our different hardware components, so that our project will be a standalone product with only a single power cord coming out the back, compared to needing three different outlets for our Jetson Nano, monitor and Nucleo board. Figure 14 shows the whiteboard idea of what we wanted. We considered making separate enclosures for each of our components but in the end, we decided that we wanted to have one complete singular enclosure for visual simplicity. The hardest part was making sure that the temperature sensor was close enough to the user while the camera still had a good enough angle to view the user. As seen in Figure 15 and 16, our final product was designed in CAD, laser cut, and painted. A clear acrylic enclosure was also added to the RFID scanner because we discovered that users tend to physically tap their ID on the scanner. The tap would sometimes disconnect the scanner from the microcontroller, and thus terminate the RFID scanning process. With the clear enclosure, users can comfortably tap their ID without touching the actual RFID scanner.

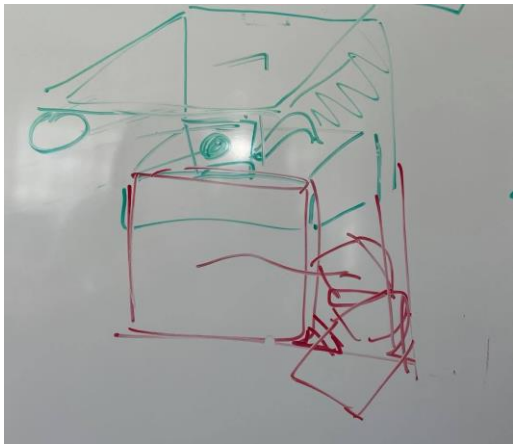

Fig. 14. Whiteboard idea.

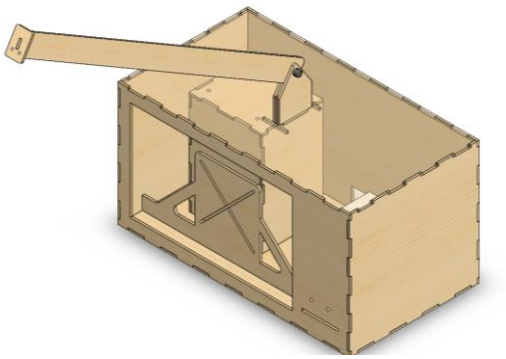

Fig. 15. CAD file of enclosure.

Fig. 16. Final laser cut and painted product.

VI. Project Management

A. Schedule

The Gantt chart is split up in three different sections: initial set up, MVP, and final project (Appendix Fig. A1). During initial set up, we focused on getting familiar with the hardware and the software packages that are available to us. This is also the time when we submitted our bill of materials and were just waiting on shipment. Due to shipment delays and placing incorrect orders, our Gantt chart did shift back by a couple days, but this did not affect our work significantly. Next is MVP, these are the tasks that we aimed to finish by the first demo. During this process, we must validate each of the components or code we write before trying to integrate everything together. We decided to push for a better face and mask detection before we started the IoT side because this was the core of our project. In the final weeks, we worked on developing our web application and building the enclosure.

B. Team Member Responsibilities

Jiamin focused on working with the RFID scanner and Nucleo board. She made sure that the RFID scanner is properly connected to the Nucleo board, so that the scanned ID is eventually communicated to the Jetson Nano. In addition, she coordinated with TechSpark to laser cut the enclosure.

Iris and Minji focused on the Jetson Nano board. They made sure that the camera modules and temperature sensor are fully integrated with the Jetson and that the facial and mask detection algorithms reached our required accuracy. They also handled the display monitor so that the information recorded by the Jetson is broadcast back to the user.

After face and mask detection reached the required accuracy, Minji and Iris focused on the cloud and IoT side. They handled the communication from Jetson to cloud, more specifically the encryption and decryption of the messages. They made sure that the messages were sent correctly and at a reasonable speed. They also created a real-time updated web application with the temperature and RFID logs.

C. Budget

Table 1 presents all the materials that were purchased for this project. We were given a budget of $600.00 and we have used about 84% of the allotted amount.

TABLE II. Bill of Materials

Component	Purchase Details			
	Supplier	***Cost (per unit)***	***Quantity***	***Cost***
Jetson Nano Developer Kit	Amazon	99.00	1	99.00
Jetson Nano Wi-Fi Antenna	Amazon	23.57	1	23.57
Micro SD Card (64GB)	Amazon	11.99	1	11.99
SD Card Reader	Amazon	12.99	1	12.99
RPi Camera Module V2	Amazon	20.00	1	20.00
Acrylic Camera Holder Case for RPi	Amazon	9.91	1	9.91
IR Sensor MLX90614_DCI	Mouser Electronics	60.00	1	60.00
X-NUCLEO-NFC02A1 RFID Scanner	STM Electronics	9.19	1	9.19
X-NUCLEO-NFC05A1 RFID Scanner	STM Electronics	14.09	1	14.09
STM32 NUCLEO-L476RG	STM Electronics	14.31	1	14.31
10.1 inch Portable Monitor	Amazon	119.99	1	115.99
Thermometer Gun	Amazon	26.98	1	27.98
ST25-TAG-BAG-A	Mouser Electronics	3.75	1	3.75
ST25-TAG-BAG-U	Mouser Electronics	12.50	1	12.50
Jumper Pins	Amazon	6.99	1	6.99
Wires	Amazon	8.49	1	8.49
HDMI Cable	Amazon	7.00	1	7.00
Power Strip with USB Ports	Amazon	14.99	1	14.99
Laser Cut Materials (Wood and Acrylic)	TechSpark	23.50	1	23.50
Total				506.23

D. Risk Management

We have mitigated several risk factors in our project. The first one pertains to the accuracy of the mask detection algorithm. Since there will be varying forms of masks, we may have higher false positives and false negatives when detecting masked faces. To address this issue, we collected more training sets with a variety of masks and trained our model with the supplemented data in order to achieve our target accuracy percentages. If our YOLOv3 algorithm did not work, we were planning on falling back to the Haar Cascade method which was

slow and inaccurate, but still worked. Luckily, we were able to exceed our target accuracy metrics.

We didn't anticipate any issues on the RFID side, but we encountered compatibility issues with the expansion board since it didn't have the capabilities, we needed to complete the necessary tasks. From a schedule standpoint, a lot of time and effort was dedicated to debugging this specific issue. However, after we bought a new compatible expansion board, the integration process with the Nucleo board was much smoother and put us back on track.

We anticipated that the IR sensor may detect inaccurate temperatures. To ensure the accuracy of the temperature data we collect, we increased the time interval to collect 20 sample As discussed in section IV, an average offset was also added to the final temperature result due to an offset we discovered when calibrating with a store bought IR thermometer gun.

Since CMU went remote after Thanksgiving break, we were concerned that TechSpark would not be available for use for building our enclosure. Thus, we attempted to mitigate this risk by having our design planned out before Thanksgiving in order to give ourselves ample time to find a backup plan if TechSpark ended up closing. Several COVID-19 cases at TechSpark did put us a bit behind schedule, but we were able to parallelize the work and improve other parts of the system.

VII. Related Work

There are several other projects that overlap with parts of our system. There is a face and mask detection system built on the Raspberry Pi, although they use TensorFlow instead of YOLOv3 [6]. In addition, there are projects that use the FLIR Lepton camera we were considering at first [7]. It is also a contactless thermometer combined with a face detection project that checks if the user has a fever. Similarly, there are projects using the same medical grade IR sensor as us, such as a YouTube tutorial from educ8s.tv [8].

VIII. Summary

Our system is able to achieve everything we envisioned for it at the start of the semester. It is able to retrieve a user's RFID, detect their face and mask, take their temperature and have it all shown on a web application.

We had an original goal to hit 10 FPS but were only able to achieve 7 FPS. If given more time, we would like to improve the FPS of the video stream without sacrificing the screen resolution. In addition, since we directed most of our focus to ensuring accuracy for our face and mask detection algorithm, we would like to improve the user interface and create a database to store the user profiles if time permits.

A. Lessons Learned

We learned that the integration of all of our components was a much more difficult process than expected. We definitely encountered some roadblocks when integrating the components and had to spend a lot of time drawing out the FSM and thinking about what fit into where. If we drew this out as the first step, before we started coding, it would have avoided a lot of confusion and provided more clarity in our actions.

When we first started our project, there was a lack of good documentation. Sometimes, the language used in the documentation did not clarify anything and led to more confusion. Working with hardware we usually depend on reading datasheets to obtain the most accurate and representative information. However, we could never find one singular document that described all the features of the hardware and often had to perform Google searches with specific keywords to find the right assistance online.

One last lesson we learned is that we cannot blindly rely on the built-in libraries. We encountered many cases where we had to deal with buggy behaviors of the libraries that we initially thought was correct. As discussed in section V, the i2cdetect feature was one of our major roadblocks where we spent a lot of time and energy to debug a working sensor.

B. Acknowledgements

We would like to thank TechSpark, especially Brian Lee, for helping us in developing the enclosure design.

References

[1] *Python wrapper for YOLOv3* https://github.com/madhawav/YOLO3-4-Py.

[2] *Mask Dataset* https://drive.google.com/drive/folders/1aAXDTl5kMPKAHE08WKGP2PifIdc21-ZG

[3] *Jetson Nano GPIO*, JetsonHacks, https://www.jetsonhacks.com/2019/06/07/jetson-nano-gpio/.

[4] Shawn Hymel, *Experiment 4: I2C Temperature Sensor*, Sparkfun, https://learn.sparkfun.com/tutorials/python-programming-tutorial-getting-started-with-the-raspberry-pi/experiment-4-i2c-temperature-sensor.

[5] *Raspberry Pi Video Stream Code* https://github.com/JetsonHacksNano/CSI-Camera

[6] *Face Mask Detection using Raspberry Pi*, ElectronicWings, https://www.electronicwings.com/users/ptksuraj99/projects/437/face-mask-detection-using-raspberry-pi.

[7] *Fever*, https://github.com/maxbbraun/fever.

[8] educ8s.tv, *Arduino Project: IR thermometer using the MLX90614 IR temperature sensor from icstation.com*, https://www.youtube.com/watch?v=F2ZCUrR-oss&list=PL4KWmkNpjC3B7iMqkTVDQxApwqh7McqcT&index=19&t=0s.

IX. Appendix

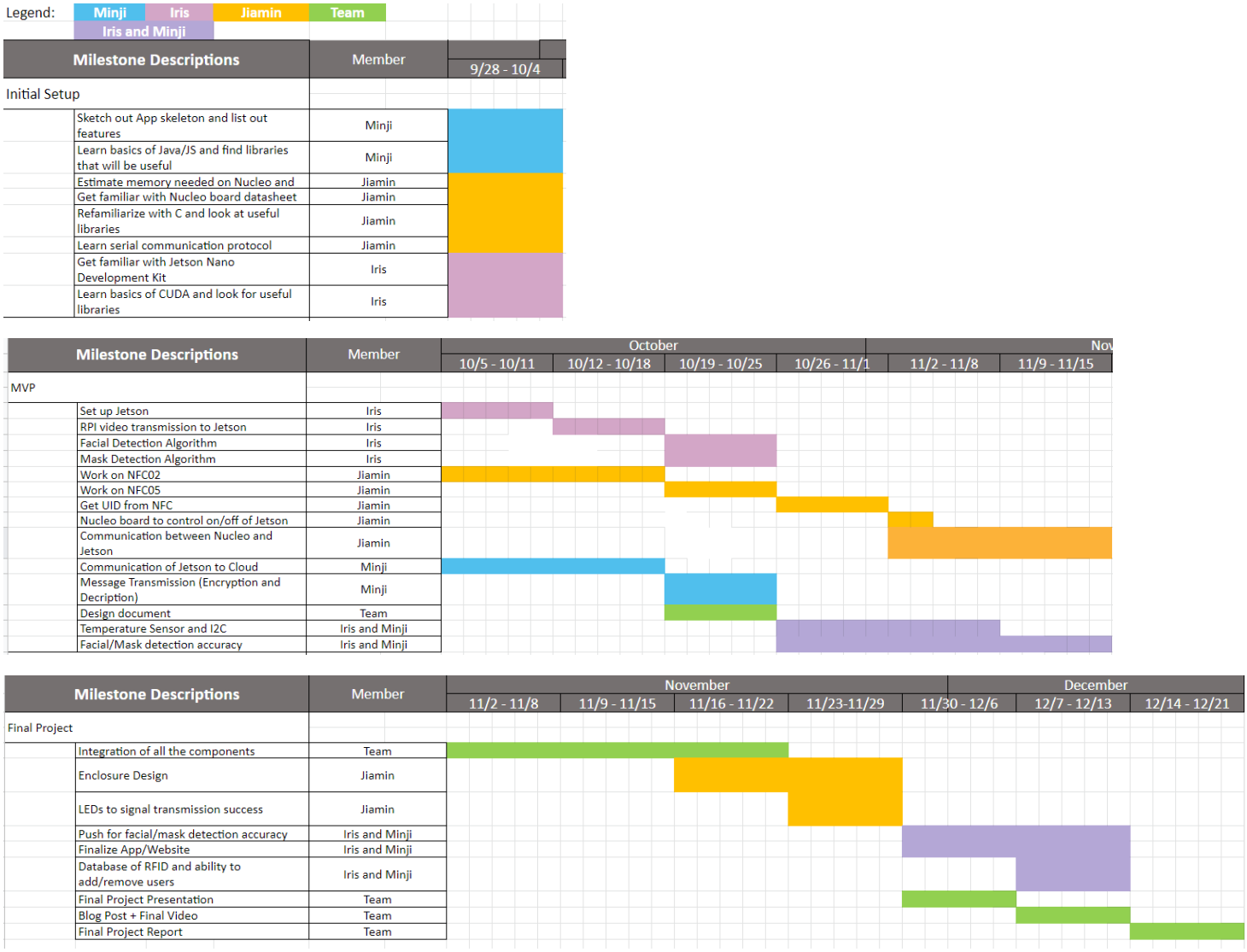

Fig. A1. Gantt Chart