Final Report: Magic Bus Passenger Demand Monitor

EECS 473
Fall 2021

Beiming Li, Manoj Handithavally Purushothama, Rohan Wagle, Yiteng Cai, Yukun Lou

I. Introduction and overview of project

In this project, we propose a real-time bus occupancy monitoring system, Magic Bus Passenger Demand Monitor. It would be composed of a network of embedded devices installed at the University of Michigan's campus bus system stations, counting the current number of people waiting. The distributed devices would send person counts to a central server owned by the university's bus authority office, for storage and statistical analysis. This real-time data would be provided at a fixed frequency (every minute), to help the transportation administrators decide whether they need to dispatch extra buses due to the unexpected high passenger volume. In the long term, the data of the number of passengers would contribute to decisions for new bus routes and schedules.

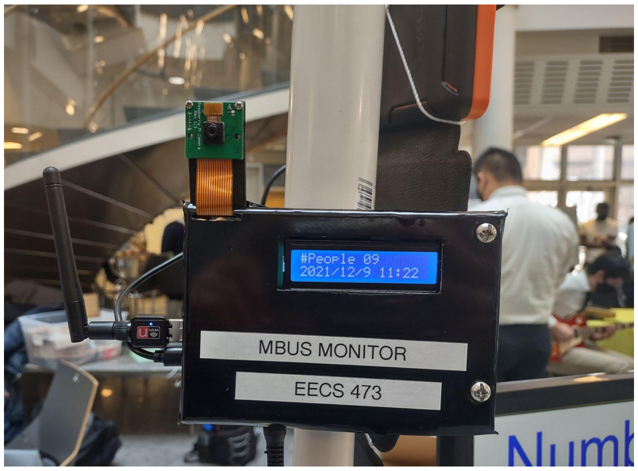

Figure 1. Fully Assembled Device

II. Description of project

The University of Michigan's campus bus system, Magic Bus, commonly known as MBus is a quick and efficient way for students to travel across both North and Central Campus. However, a common problem is that passengers will accumulate at the stations suddenly, without the bus system administrators nor the drivers becoming aware of this sudden demand. Currently, the Magic Bus system has a smartphone app which displays where each bus is currently located, for any given bus route in the system. However, a reverse app does not exist. That is, there does not exist an app for the bus administrators to count how many people are at each bus stop.

Therefore, we propose a small, inexpensive, solar powered device which could be mounted at each of the MBus stations to count the number of passengers waiting, on a minute-by-minute basis. The device would operate during the day, since that is usually when passenger demand can spike. Our device would be privacy oriented, so it would compute the number of people locally, and only send the person count, time stamp, and device ID to the server. This information could be used for real-time analysis and efficient bus dispatching. Additionally, this information could be used for long term analysis of bus passenger demands at different times of day during different days and semesters.

Project Constraints

The primary constraints of this project are power, size, and cost. The device should be solar powered with a battery backup, thus eliminating the need for running power lines to each station. Additionally, the device should be small enough to mount on existing bus station shelters or sign poles, without requiring additional structural reinforcements. Finally, the devices must be inexpensive enough to be deployed to nearly every bus station, and should not require frequent maintenance, which would increase the long term cost. The final constraint is privacy. As we mentioned, the privacy of the passengers waiting at the stations is important. People may not feel comfortable having their picture taken and sent to some remote server. Therefore, our device must process all image data locally and wipe each frame as soon as the number of people has been computed.

Person Detection and Counting

To meet the computing demands of this project without increasing the cost drastically, we use a Raspberry Pi Zero W 2 as the device which takes images and counts the number of people, because it is a well supported, low power Linux machine. We used the official Raspberry Pi Camera Module 1.3v, due to its low cost and 1080p resolution. Originally, we believed a Histogram of Gradients approach for detecting the number of people would be sufficient. However, we quickly found that not only was it unreliable in detecting people, but it was also incredibly slow too. We found the framerate was always less than 1 FPS. Therefore, we looked into a neural network approach of person detection, using the Python TensorFlowLite framework, directly on the device.

Considering the limited computing power, we carried out extensive research on finding a small and efficient model. We prioritized the single-shot object detection models, such as You-Only-Look-Once (YOLO) and Single-Shot-Detector (SSD), over two-shot detectors. The main reason is that under limited computing power (1GB RAM in our case), the single-shot detector achieves a comparatively higher frame rate with a modest exactness trade-off. Besides, our camera would be installed at a relatively high angle, in order to reduce the chance of people 'hiding' behind others. Therefore, the image we get would have a top-down view, which results in the difference between the size of people in the frames. After looking into two state-of-art single-shot detectors, we noticed that the SSD model removes fully connected layers and increases the number of feature maps with different sizes compared to YOLO. Such a pyramidal network structure should be beneficial in our case as it can improve the system's ability to detect objects of various sizes. Based on theory and testing results of multiple models (YOLO, SSD, EfficientDet), we selected a pre-trained SSD Mobilenet model and deployed it with the Tensorflow Lite framework.

The SSD Mobilenet we selected accepts a 300x300 RGB frame as the input, therefore we resized the 1280x720 images taken by the camera to meet the input requirement. After multiple testing, the model turns out to achieve a good frame rate (2 FPS) and a modest accuracy under normal light conditions. However, in our case, the number of people only needs to be updated per 60 seconds, which reduces the importance of a high frame rate. Hence, we decided to crop the 1280x720 into four 640x360 pieces and feed them to the model. The accuracy of the object detection turns out to be sufficiently improved, with a modest frame rate trade-off.

As stated above, we are aiming at a high accuracy but have a loose limitation on the frame rate. Alternatives such as using larger networks (i.e. two-shot detector) or further cropping the image (eight 320x360 pieces) might seem to be considerable. However, our RPi needs to run advanced networks under high current draw for an extremely long time, which introduces extra power consumption that our powering strategy could not afford. Moreover, further cropping the image would also introduce extra 'noises', since much more people might be cut in half or even cut into four pieces. During most of our testing, the eight croppings strategy resulted in a lower accuracy compared to the four croppings.

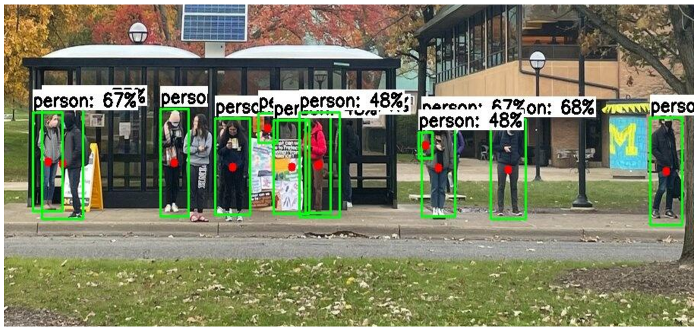

Figure 2. Sample Image Processing Result

Device Hardware

Originally, we considered both the Raspberry Pi 3B and Raspberry Pi Zero. We found the Raspberry Pi 3B, with its 64-bit ARMv7 Cortex-A53 1.4GHz quad core processor and 1GB of RAM, could surprisingly run an image frame through the neural network at approximately 2FPS. This was sufficient for our needs, because our device only takes an image every minute. However, we wanted to minimize the footprint of the overall device as much as possible, so we also experimented with the original Raspberry Pi Zero W. With only a single core ARMv6 CPU and 512 MB of RAM, this device could barely compile the image processing libraries, let alone actually run them. Then, during the middle of November, we were able to get ahold of the new Raspberry Pi Zero W 2. With the same ARMv7 quad core CPU as the Raspberry Pi 3B, but with 512 MB of RAM, this device was a welcomed compromise between the 3B and original Zero. We found the FPS of the person detection network to be slightly less than 2FPS, which was still sufficient for our needs.

We found the power consumption of the Raspberry Pi Zero 2 W to be quite reasonable. The maximum current the Raspberry Pi consumes (with camera) is 120mA, when it is pulling an image frame from the camera and running the person detection code. When idle, the device consumes around 80mA.

One caveat of using a Raspberry Pi Zero W for the detection is that its internal WiFi antenna is relatively weak. When testing our device at Pierpont station, we found that the MSetup WiFi network was out of range. Therefore, we used an external USB WiFi antenna which consumes a moderate 40mA current at 5V, but provides a strong connection to building WiFi hotspots when mounted outdoors. When the WiFi module is plugged in, the measured overall current consumption when running the detection code is around 200-220mA. This is fair considering we are running a full neural network inference on an image, but since our device runs off a solar panel and battery pack, we decided to lower our power consumption further. We did this by having an AVR microcontroller act as a watchdog for the Raspberry Pi, telling it to shutdown at night and wake up in the morning, to save power. Additionally, when Wifi is absolutely not available at stations, we add a GSM module (FONA 3G), which the AVR is responsible for managing.

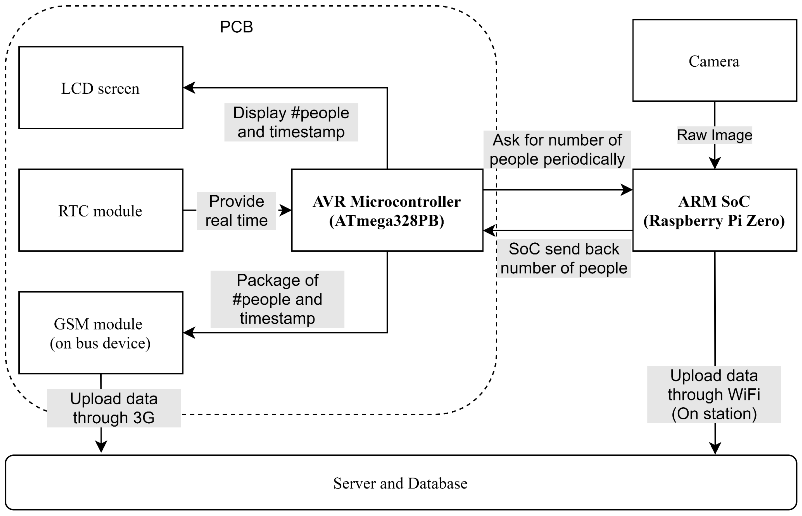
PCB
LCD screen
Display #people and timestamp
RTC module
Provide real time
AVR Microcontroller (ATmega328PB)
GSM module (on bus device)
Package of #people and timestamp
Upload data through 3G
Camera
Raw Image
Ask for number of people periodically
ARM SoC (Raspberry Pi Zero)
SoC send back number of people
Upload data through WiFi (On station)
Server and Database

Figure 3. System Architecture

The main AVR microcontroller of our device is the Atmega328PB on the PCB. The PCB integrates the LCD screen headers, the RTC module headers, the GSM module headers, the Raspberry Pi headers and the power management system. The AVR controller uses I2C to talk with Raspberry Pi and the RTC module. Every 40 seconds, the controller will send a request to the Pi so that the Pi will then capture an image and start the image processing to get the number of people. In the meantime, the controller would read a timestamp from the RTC module. The image processing operation will take less than 4 seconds end-to-end, and the result will be sent back to the controller. After that, the microcontroller will display the number of people, along with the corresponding timestamp, on the LCD screen, and then upload the data with the device's unique ID to the server. If the Raspberry Pi can connect to WiFi, it will directly upload the data to an intermediate web server, "dweet.io/dweet/for/gsm_mod", in JSON format. If WiFi is not available at the station, the Raspberry Pi will send the data to the AVR over I2C. The AVR unit will send the data to dweet.io over 3G by passing it to the GSM module via UART.

Server

Our "server" is a Python script which listens for new messages posted to the dweet.io page, via the Dweepy Python library. Once a new message is detected, the Python script will add the device ID, time stamp, and person count, from the message, into a Sqlite3 database running on the same machine. The script then regenerates a bar chart showing the average number of people seen over 20 minute intervals throughout the day, for a nice visualization of the data.

Power

To reduce power consumption, we make the AVR microcontroller shut down the Pi at night, by sending a message over I2C, which causes the Pi to run "sudo shutdown -h now", halting itself. It is not necessary to run the device at night, because the number of passengers is typically quite low. The AVR reactivates the Pi at 8am everyday, since this is when most students leave for classes. The reactivation process occurs by pulling the Pi's GPIO 3 pin to ground momentarily. This method of reactivation is recommended by the Pi's creators. The AVR connects this pin to ground via a BJT transistor. The transistor's base is connected to a digital output pin of the AVR, the collector is connected to a pullup resistor to 3.3V (Pi's HIGH voltage), and the emitter to ground. The Pi's GPIO 3 pin is connected to the collector. Thus, when the AVR activates the base pin, the GPIO pin is connected to ground, but when the AVR deactivates the base, the GPIO is pulled high.

Our device runs on 5V. With the WiFi adapter plugged in, the Raspberry Pi draws around 220mA max when actively taking an image or computing the number of people, around 120mA when idle, and around 70mA when shutdown. The PCB consumes around 70mA with the LCD plugged in and turned on. Over our 8am-5pm normal operating schedule, the Pi is active for 30 minutes total, idle for 8.5 hours total, and shutdown for 15 hours total. The PCB runs 24 hours, because the AVR needs to always be watching the Raspberry Pi to wake it up to shut it down. The total power consumed in a single day is therefore: (220mA*0.5h) + (120mA*8.5h) + (70mA*15h) + (70mA*24h) = 3860mAh=3.86Ah consumed per day. Our battery pack is rated as 20,000mAh, but with a conservative estimate of 50%, this would mean we could supply our device with (5V*20Ah*0.50)/(5V * 3.86Ah / day) = 2.59 days. However, the battery pack's onboard solar panel replenishes it with energy at 6W with mild overcast. So, we would need only (5V * 3.86Ah) / 6W = 3.22 hours of decent sunlight per day, to run our device without really tapping into the stored energy in the battery.

The AVR (Atmega328PB) on the PCB is programmed with C++ code through a 2x3 pin header on the PCB, which connects to the AVR's SPI bus pins. The program on the Pi is a Python script that runs at boot. When the power is plugged in, the Atmega328PB on the PCB initializes the LCD and GSM, then begins to send commands to the Pi. On boot, the Pi starts to listen to the I2C bus,to receive commands from the AVR.

III. Milestones, schedule, and budget

1. Milestones and schedule

1.1 Milestone 1

At this stage, we were on track with our schedule. We finished all our desired deliverables in milestone 1 and also brought forward the task of testing the capability of sending data using the GSM module. The detailed progress and issue are listed in table 1.

Deliverable	Progress and issue
Capable of getting the image processing result (Done)	We tried several opencv modules for detecting people and chose Single Shot Detector, which is capable of processing images.
Prototype able to read	Able to use python library

timestamp from RTC module (Done)	adafruit-circuitpython-pcf8523 and I2C to read timestamp from RTC module.
Show accuracy of people counting (Done in daylight)	In daylight conditions, people counting is quite accurate, but in dark, it almost cannot detect a person. We were trying the IR camera to make a difference.
Setup cell service plan (Not setup yet, but test the capability)	Since we did not yet want to start paying for the data plan, we used our own SIM cards to test and the GSM worked.
Selection of bus stations and buses where we would deploy our system (Done)	Select Pierpont Commons as the testing bus station, and Northwood, Commuter South as the testing buses.

Table 1. Milestone 1 Progress and Issues

1.2 Milestone 2

At this stage, we were a little behind our schedule. The major issue was the PCB assembly. Our first version of PCB was not feasible because the ATmega2560 chip that we designed around had gone out of stock before we could order it. So, we had to improvise and prototype our circuit with a Atmega328PB processor, which took more time than expected. We chose the Atmega328PB, because we already had some scrap boards with chips we could salvage, as well as being able to order more if necessary, since they were in stock.

Having prototyped with a different processor, we had to revise our PCB design and then order the new version. The delivery took another week. Also, around this time, we had been testing with the Raspberry Pi NoIR camera, which we hoped would let us run the device at night. However, the IR LEDs on the camera provided such little light that it diffused over just a few meters, which made it useless across the street of a bus station. So, we had to alter the plan and only have our device operate strictly during the day.

The detailed progress and issues are listed in table 2.

Deliverable	**Progress and issue**
PCB assembled (Ordered	The PCB was ordered, but has not arrived yet. Our first

but not assembled)	version PCB used Atmega2560, but it was out of stock. We had to make the second version, using Atmega328PB.
Prototype able to send data via GSM module (Done)	We are able to send the data obtained from Raspberry Pi and Atmega328PB to a web server through the GSM module.
Prototype able to read GPS data (Removed)	We decided to remove this part from our design, since the current MBus system has the GPS data itself. Furthermore, we realized the prospect of putting the device on the bus was not very useful, so a GPS was unlikely to be needed.
Camera tuned for best performance (Only focus on daylight condition)	During the daytime, we made it. We are supposed to use IR light at night, but it does not perform well. Given the time, budget, and power constraints, we pivot our design to only function during the daytime.
Server software prototype built (Partially done)	We used dweet.io, an intermediate website, to serve as the HTTP POST target for the devices. We planned to write a Python web server which saves data to a SqlLite database, but not done yet at that stage.
Hardware interface module code completed and tested (Done)	Each hardware interface module is tested successfully on a breadboard. This included the bidirectional logic shifting circuit for 3.3V/5V shifting for I2C communication between the Raspberry Pi and AVR.

Table 2. Milestone 2 Progress and Issues

2. Budgets

We have modified our budget. What we stuck with were the GSM module (including breakout board, antenna and battery), RTC module and the camera. Compared to the original budget, we removed the stepping voltage regulator for a LiPo battery to power the Pi and PCB, and instead used a USB battery bank with a built-in solar panel. We replaced the Arduino Nano board with our self-designed PCB, together with the microprocessor and surface mounted components. In addition, since we do fewer functions on the Raspberry Pi, and due to the power issue, we use Pi Zero 2 instead of Pi 3B+, which saves us on the budget. For the power supply, we chose a solar panel and battery, which is cheaper and of better performance, but not the 5000mAh LiPo battery and charger in the original design. The total cost for a device that has a wifi antenna and GSM module is $247.7. The GSM module is expensive as it has a built-in GPS module, which is actually unnecessary. We can further cut the cost by replacing it with a normal one. The budget we spent on testing, such as the USB Multimeter, previous version of PCB, are counted as miscellaneous in the original budget.

IV. Lessons learned

We were able to achieve most of our intended goals that we planned out at the beginning of the project. Initially, we were quite worried about the accuracy of the detection. We had to go through several detection algorithms and models, checking their feasibility not only in terms of accuracy but also efficiency. The SSD (Single Shot Detector) model that we ended up using was the most accurate, efficient, and reliable for our project.

Having to design our PCB in the midst of a shortage of chips and other components was a major barrier and inconvenience. We had to redesign our PCB and a good portion of our circuit because of it, since by the time we completed one design, the parts we designed with had already sold out. We should have thought about the availability of components when we were choosing parts on our PCB. Had we purchased the parts before actually designing and ordering the PCB, we would have saved a lot of time and effort.

We immediately realized some of the mistakes that we made in the first version of our PCB after we assembled and tested a few. Although we rectified the problems, like correcting the reset pin connection, changing the Micro USB power jack to a DC barrel jack and ordered a new set of PCBs, it would be nice if we can go back in time and have a heads-up on this. It would have saved us a lot of time.

We do not fully implement all of our expectations of our proposal. For example, we only focused on the device at the bus station but not on the bus, so we cancelled the GPS module. We only detect people during the daytime because of the camera limitation and power cost. During the process, we realized that we tried to reach too many goals and some of them are hard. If it is possible, we should learn the lesson that do not overestimate the functions we can accomplish in a certain timestamp and within a certain budget.

Each of us learns some lessons in the project, with details listed in Table 3.

Beiming	SQL database design, machine learning, communication protocols
Manoj	PCB design, circuit prototyping, communication protocols
Rohan	PCB design, SQL database design, surface mount soldering, machine learning
Yiteng	PCB design, communication protocols, soldering
Yukun	PCB design, soldering PCB, communication protocols

Table 3. Technical Lessons Learnt

V. Contributions of each member of team

Beiming: 20%

I researched, selected, deployed and implemented object detection software on our Raspberry Pi Zero 2, including evaluating different neural network models. I tested the performance and accuracy of our device with teammates indoors and at the bus station. I then decided on the selection of the camera we used and modified the object detection strategy based on the field testing results. I also developed the communication software for Raspberry Pi and AVR, to manage the interaction between two devices, which involved the creation of our own message signal protocol. I participated in debugging the first version of PCB, and assisted Rohan in testing breadboard circuits and developing server software, including the Dweet message sniffer, SQL database creation, and data visualization scripts. I also worked on the complete measurement of our device's PCB and footprint, when we designed the CAD models for our enclosure.

Manoj: 20%

I prototyped and tested the interface between the Raspberry Pi and our PCB's AVR chip, including the message signal protocol over I2C. This included figuring out different communication protocols (I2C, SPI, etc) and the required circuit for the two devices to communicate safely, such as the logic level shifting between 3.3V (Pi) and

5V (AVR). I also worked on the design of the BJT circuit for shutting down and waking up the Pi using the AVR. I also contributed to the PCB design and debugging, GSM interface for pushing the data to Dweet.io, soldering the PCBs, and overall assembly of the system, including the final expo model of our device. I also assisted Rohan in working on the CAD models and 3D printing our enclosure and camera mount.

Yukun: 20%

I first figured out the protocol of the RTC module, including hardcoding the timestamp and reading the current time, which was used both for the data logging, as well as for the shutdown/wakeup procedure of the Raspberry Pi, conducted by the AVR chip. Then, I designed the schematic and layout of the PCB in Eagle, and did part of the soldering and testing. I designed the poster for the expo, and compiled the information for our milestone meetings' reports. I also participated in the field testing of our device, to measure the performance of object detection models under different lighting conditions.

Yiteng: 20%

I designed the layout of our PCB in Eagle, soldered them, and made the board work properly during testing. I was also responsible for writing the prototype and production software for the GSM module and testing it thoroughly to ensure it worked reliably. After that, I wrote a program to make the AVR work with every component besides WiFi, including the I2C communication with the RPi (designed my own message signal protocol), the GSM/AVR interface, and the Raspberry Pi shutdown/wakeup AVR procedure.

Rohan: 20%

I developed the Python server listening for the new messages posted to Dweet.io. I created a SQL server to log these results, and a Python script to visualize the data. I helped evaluate/prototype the GSM with Yiteng, by finding the AT code sequence to get HTTPS requests to send. I prototyped the logic level shifting circuit which allowed the Raspberry Pi and AVR to communicate over I2C, and the BJT transistor circuit used for the AVR to reboot the Pi. I assisted in the early computer vision work, especially the Histogram of Gradients approach to detecting people. I helped assemble the PCBs, and found a method of removing bridges between pins on the soldered chips. I did the Autodesk Inventor CAD work for the enclosure.

VI. Cost of Manufacture

Item	Quantity	Cost/Unit	Total Cost
0.1uF Capacitor (0603)	1	$0.11	$0.11
0.1uF Capacitor (0805)	4	$0.09	$0.35
22pF Capacitor (0805)	2	$0.38	$0.75
47uF Electrolytic Capacitor	2	$0.36	$0.71
8MHz Crystal Oscillator	1	$0.90	$0.90
10K SMD Resistor (0603)	5	$0.12	$0.61
10K SMD Resistor (0805)	2	$0.03	$0.05
1K SMD Resistor (0603)	3	$0.04	$0.11
220 SMD Resistor (0603)	1	$0.03	$0.03
Header 18 Pin Male	1	$1.10	$1.10
2x3 Header Pin Male	2	$0.19	$0.38
BSS138 MOSFET	2	$0.22	$0.44
M7 Diode	1	$0.03	$0.03
Green LED	1	$0.18	$0.18
Yellow LED	1	$0.18	$0.18
5V LDO Regulator	1	$0.25	$0.25
DC Barrel Jack	1	$0.46	$0.46
LCD	1	$4.25	$4.25
RTC Module	1	$5.95	$5.95
Raspberry Pi Zero W 2	1	$15.00	$15.00
WiFi Antenna	1	$24.99	$24.99
2N3904	1	$0.05	$0.05
Tactile Switch	1	$0.10	$0.10
Raspberry Pi Camera	1	$6.00	$6.00
Atmega328PB	1	$1.43	$1.43

Potentiometer	1	$0.60	$0.60
USB A Female connector	1	$1.25	$1.25
PCB	1	$1.50	$1.50
Grand Total			**$67.77**

Table 4. Cost of Manufacture

This cost is slightly steep, because we want these devices to be deployed at each bus stop. However, we could drop the cost by removing the LCD, as well as building our own RTC (when the parts become available again) instead of using a breakout board. We had to use the RTC breakout for this project, because the raw parts to build our own in our PCB weren't available, due to the chip shortage.

We also had a 3G FONA GSM module which was $130 total, including its own battery. This was extremely expensive and unnecessary, because we only realized after buying it that any cell phone store will sell you a $30 USB 4G modem that you put your sim card into. This would have plugged into the Raspberry Pi instead of the Wifi module, if you wanted GSM, rather than having a separate GSM board plugged into the AVR. In a real product, we could have used a real-time clock with an alarm feature to tell the Pi to shut down, and have a second alarm trigger the BJT transistor to wake up the Pi again. This, coupled with a USB 4G modem would eliminate the need for an AVR chip altogether.

VII. Parts and Budget

Item	Quantity	Price	Tax	Total Cost
Real Time Clock	1	$5.95	$0.00	$5.95
3G Cellular Breakout	1	$79.95	$0.00	$79.95
LiPo battery	1	$9.95	$0.00	$9.95
GSM antenna	1	$2.95	$0.00	$2.95
GPS antenna	1	$17.95	$7.71	$25.66
Fisheye camera	1	$18.99	$1.14	$20.13
GPS SMA/uFL adaptor	1	$5.99	$0.36	$6.35
Raspberry Pi Zero 2	1	$15.00	$1.26	$16.26
Coincell batteries	1	$4.27	$0.26	$4.53
PCB Parts	1	$54.40	$3.73	$58.13
PCB LEDs	1	$7.36	$0.44	$7.80

M7 diodes	1	$6.19	$0.37	$6.56
PCB Parts	1	$12.71	$0.76	$13.47
PCB Stencils	1	$22.90	$0.00	$22.90
RTC Module#1	1	$11.99	$0.72	$12.71
RTC Module#2	1	$11.99	$0.72	$12.71
Solar Panel	1	$38.69	$2.32	$41.01
Cell Service	1	$25.00	$1.99	$26.99
USB Multimeter	1	$9.99	$0.60	$10.59
USB Wifi Adapter	1	$24.99	$1.50	$26.49
Raspberry Pi Zero 2 W	1	$15.00	$1.38	$16.38
Buttons and USB Headers	1	$11.75	$1.54	$13.29
Female RPI GPIO Header	1	$7.50	$1.16	$8.66
ATmega2560	1	$16.81	$0.60	$17.41
PVC Pipe & Screws	1	$37.86	$2.27	$40.13
	Total:	$476.13	$30.83	
	Sub Total:	$506.96		

Table 5.Parts and Budget

VIII. References and Citations

1. Interface codes

1.1 Interfaces on PCB

```
/* print long sentence on lcd screen, delay 2s each screen
 * @param data the entire data to be printed on lcd
 */
void printLong(String data);

/* turn on GPRS first before transmitting data
 * GPRS should only be turned on once
 */
int turnGPRS();

// gsm initialization
void GSM_init();

// post data to website
int postData();

// check if there's signal
```

```
int check_network();

/* send message to pi through I2C
 * @param str command to be received by pi, "down", "count", "ack", "nack"
 */
void Send_Command(char* str);

// read timestamp from rtc
void read_time();

// read count from pi through I2C
void Read_Count()

// main function to be called by timer
bool function_to_call(void*)
```

1.2 Camera Module and I2C

```
#define MODEL_NAME = "coco_ssd_mobilenet_v1"
#define GRAPH_NAME = "detect.tflite"
#define LABELMAP_NAME = "labelmap.txt"
#define CWD_PATH  // which is your working directory
# Func: Initialization of the TFLite Interpreter, including loading the
Tensorflow Lite model, preplanning tensor allocations to optimize inference
and loading the label map
def TFLite_init():

# Para: Initialized TFLite Interpreter, and all the available labels supported
by the pretrained Mobilenet-SSD model
# Func: Triggered RPi camera to take images, count the number of people by
passing the images to the ml model and return the number of people in the
frame
def Count(interpreter: Interpreter, labels: list) -> int:

# Para: id and tick are default parameters specialized by the pigpiod library
# Func: a call back function triggered whenever RPi received an income i2c
message from AVR;
#       performs various action based on the command from AVR;
#       the actions including saving the data to the log file, uploading data
to web server, send back number of people, etc
def i2c(id, tick):
```

2. PCB Design.

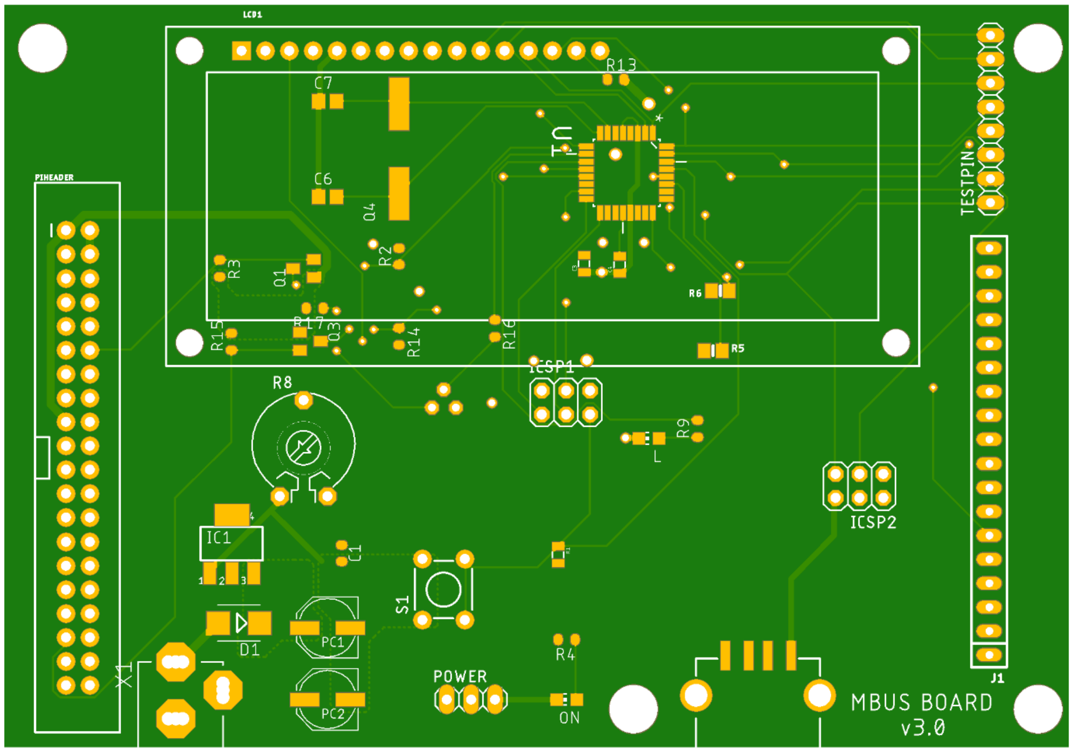

Figure 4. Design of PCB

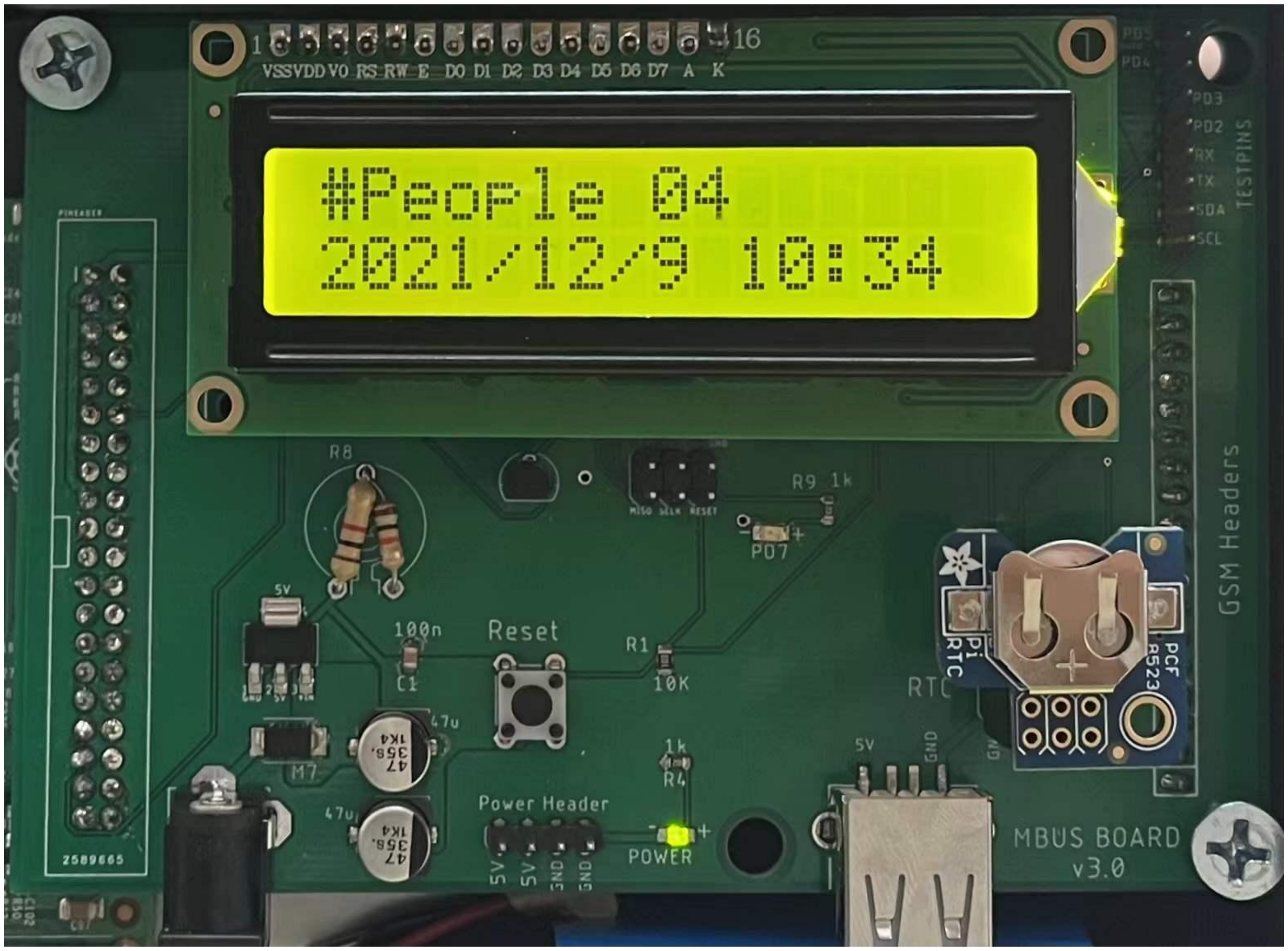

Figure 5. Soldered version of PCB

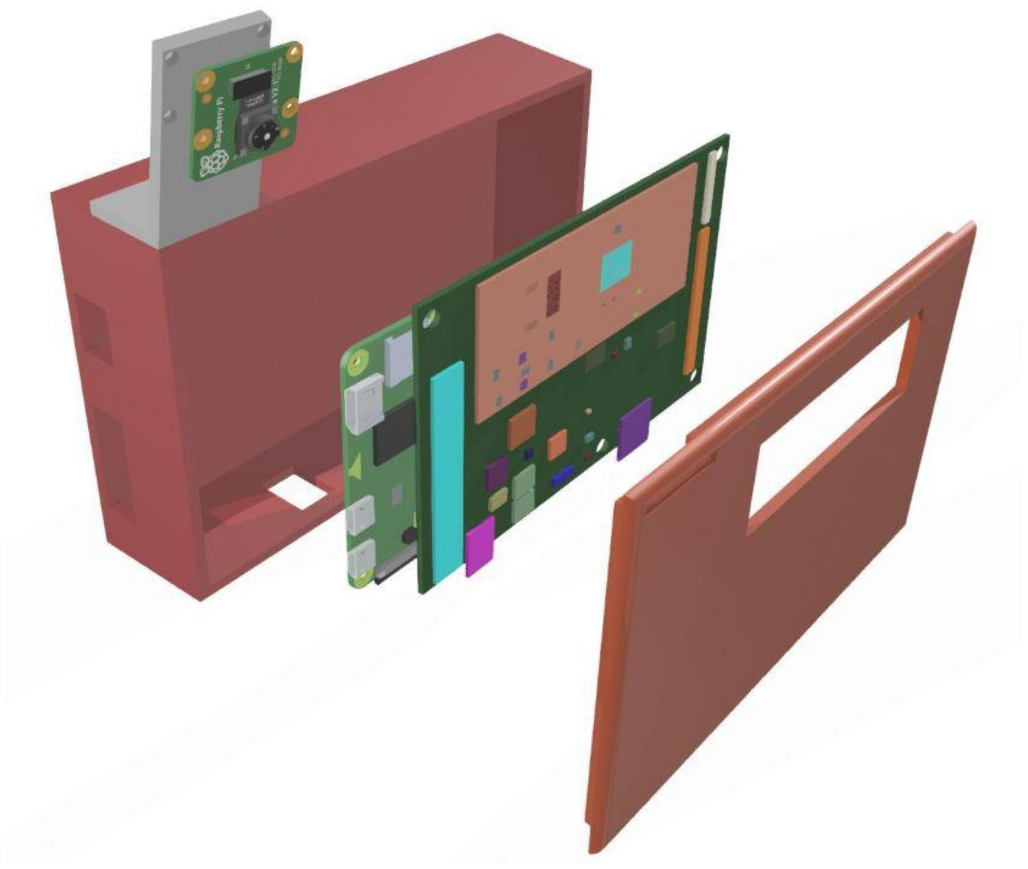

Figure 6. CAD Assembly of our enclosure with PCB and RPi inside

3. Acknowledgement

We used these libraries for our project

- ➢ Bidirectional logic circuit: https://learn.sparkfun.com/tutorials/bi-directional-logic-level-converter-hookup-guide/all
- ➢ Arduino Schematic: https://www.arduino.cc/en/uploads/Main/arduino-uno-schematic.pdf
- ➢ These libraries come from Arduino:
  LiquidCrystal.h: https://github.com/arduino-libraries/LiquidCrystal
  Wire.h: https://github.com/esp8266/Arduino/blob/master/libraries/Wire/Wire.h
  arduino-timer.h: https://github.com/Megunolink/MLP/blob/master/ArduinoTimer.h
  RTClib.h: https://github.com/adafruit/RTClib
- ➢ The FONA library *Adafruit_FONA.h* we used is a modified version based on the official library. It can be found at this Github link: https://github.com/botletics/Adafruit_FONA
- ➢ We used the pre-trained Mobilenet-SSD model from Tensorflow Hub, and similar models such as YOLOv5 and EfficientDet for testing and comparing. The object detection Python code we developed is based on the official TensorFlow Lite image classification example and open source code written by Evan Juras available on Github: https://github.com/EdjeElectronics/TensorFlow-Object-Detection-API-Tutorial-Train-Multiple-Objects-Windows-10.
- ➢ Library support using the RPi as I2C target device: https://github.com/joan2937/pigpio

We would like to thank Dr. Mark Brehob, Matt Smith for their constant guidance and encouragement, GSIs Benjamin Cyr, Cameron Kabacinski, Luke Cohen for providing assistance and support during our project. Many thanks to Stryker and University of Michigan's department of Electrical and Computer Engineering for the funding, which made the project possible.

Our code and PCB files can be found in our Github repository: https://github.com/edwinli0626/MBus_monitor/ . The "main" branch holds the image processing code and Raspberry Pi specific Python scripts. The "pcb" branch holds both the pcb schematic and board files, as well as the AVR C++ code. The “server” branch holds the Python scripts for the server and plotting software.