Camerazzi

Author: Mimi Niou, Cornelia Chow, Adriel Mendoza

Electrical and Computer Engineering, Carnegie Mellon University

***Abstract*—This report details the conception and design of an autonomous robotic cameraman intended for social gatherings or professional events. Hiring a photographer often involves unavailability, unreliability, bias, and even latency in delivery of the photos. Our solution offers an automated robot that's always available, reliable, and unbiased. In addition, it ensures ample margins around faces and delivers photos instantly. Our robot does this efficiently with software algorithms, requiring lower power and less storage space than other implementations such as one that deletes subject-less photos after an event, and demonstrates the capabilities and advancements of electrical and computer engineering.**

***Index Terms*—Arduino, autonomous, camera, design, face detection, GPIO, iRobot, infrared, LCD, motor, OpenCV, PyCreate2, robot, Raspberry Pi, Roomba, sensors, thermal**

I. Introduction

IN this day and age, recording social and professional events through photographs is a common occurrence. However, cameramen often have high rates as well as busy schedules and may sometimes be biased when taking photos. Our project was inspired by a problem presented to us by the Robotics Club at CMU. This club organizes several events throughout the year and they always have trouble booking a photographer because of the challenges detailed in the abstract. It is for these reasons that student volunteers are typically sought out. As the number of volunteers has been steadily declining, we came up with the solution of designing a robot that could take on the duties of a photographer.

Camerazzi aims to be an unbiased, available, and reliable alternative to the typical cameraman that would be booked for an event. Our project provides a solution particularly suitable for recurring events due to reduction in costs and the constant availability. While other implementations like taking numerous snapshots randomly at an event are also viable, our robot uses the integration of software and hardware to efficiently take photos of the important subjects, the people, at an event. This way, we reduce the power consumption required, save storage space, and eliminate the need for post-analysis.

II. Design Requirements

In order to measure success for our robot and the pictures that it takes, we have defined the following requirements.

As an overarching requirement, we expect our robot to capture a minimum number of photos, with a given room size, number of people in the room, speed of the Roomba, and duration of time the Roomba is active. By researching standing crowd density [7] we have set an ideal crowd density of 1 people per 4 square meters (1/4 person per square meter). This will allow enough room for our robot to roam and capture photos at ideal distances.

To illustrate, we have here a sample space for the robot to roam of 2 meters by 4 meters.

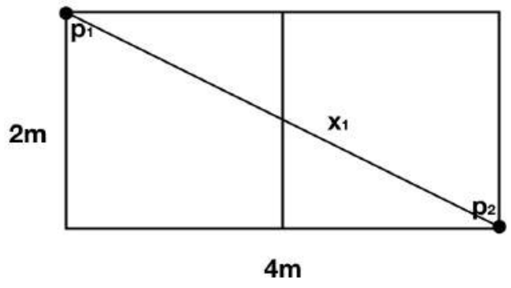


Fig. 1. Sample 2m x 4m space

Based on room density of 1 person per 4 square meters, there should be a maximum of 2 people in this space with our robot. To traverse the entire room in the shortest time, assuming that the people are at the farthest distance away from each other, the path is from point p_1 to point p_2.

$$x_1 = \sqrt{}(2^2 + 4^2) = 4.472135955 \qquad \text{(Eq. 1)}$$

The robot will move at a speed of 0.05 m/s, the fastest it can go without jitter with a structure on top of it.

$$time\ to\ get\ from\ p_1 to\ p_2 = \frac{4.472135955\ m}{0.05\ m/s} = 89.442719\ s$$
(Eq. 2)

$$camera\ adjustment\ slack\ time = 8\ s$$

$$LCD\ screen\ countdown\ time = 4\ s$$

$$allowance\ for\ additional\ slack\ time$$
$$(i.e. turning, moving\ backwards, etc) = 5\ s$$

$$\begin{aligned} &time\ elapsed\ between\ 2\ photos \\ &\quad = time\ to\ get\ from\ p_1 to\ p_2 \\ &\quad + camera\ adjustment\ slack\ time \\ &\quad + LCD\ screen\ countdown\ time \\ &\quad + additional\ slack\ time \\ &\quad = 89.442719\ s + 8\ s + 4\ s + 5\ s \\ &\quad = 106.442719\ s \end{aligned}$$
(Eq. 3)

$$\frac{1\ photo}{106.44\ s} = \frac{z}{3600\ s}$$
(Eq. 4)

$$z = 33.82\ photos$$ (Eq. 5)

Therefore, at least 33 photos should be attempted per hour if the room density is 1/4 person per square meter. Not all of these photos need to be uploaded to the cloud - only those that satisfy other low-level requirements. We simply want our robot to stop at least 33 times per hour and attempt to capture photos (which we will be able to determine based on print statements).

This requirement is imperative to the success of our project, as the main function of our robot is the ability to take photos at an event, so photos captured act as a key success factor of our project.

Our first low-level requirement is that the robot must be at least 3 feet away from humans in front of it. We set this requirement because academic papers such as Mumm & Mutlu's *Human-Robot Proxemics* [6] describe a comfortable distance between robots and humans as being approximately 3 feet. We will evaluate this requirement by laying out a measuring tape in front a stationary person, setting the robot to move towards the person, and checking to see how far the robot stops from the person.

Another requirement we've established is that robot stopping latency must be less than 1 second. We set this as our maximum latency because we want our robot to move at a speed of 50 mm per second. In our software, we will be detecting when the robot is 3 feet away from a human. If it moves 50mm within the 1 second latency, it is a negligible amount and the robot is still approximately 3 feet away. We will evaluate this requirement by recording the time that a stop command is sent to the robot, and the time that the robot actually stops.

The next requirement we've set is that the robot must be able to detect 90% of human faces in real time. We've set it to this value because we have found that using this algorithm on still images typically achieves a 95% success rate. But since we're using this algorithm on continuous video, we've slightly lowered the success rate to 90%. We will evaluate this measurement by manually counting the number of faces our algorithm detects at points in time during the demo, and dividing it by the number of faces that were actually present in the frames.

Another requirement we've set is that 100% of the photos taken must include a human. We have set this requirement because we don't want our robot to take extraneous photos when it should be spending its time taking useful photos. This is consistent with our decision to use this implementation, as opposed to one where take many photos and delete the ones that are not usable. We will evaluate this requirement by examining every picture taken at the demo and confirming that there is a human in the photo.

Our next requirement is that every photo must have a 5% margin between the borders of the face and the borders of the image. This is to ensure that no heads get cut off when the picture is taken. We will evaluate this requirement similarly to the previous requirement by examining every picture taken at the demo and confirming that there is a 5% margin between the face borders and the image borders.

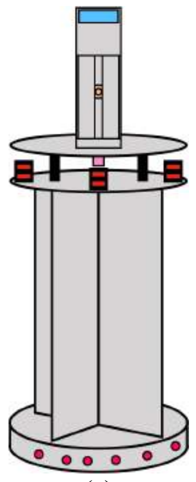
The next requirement is that the width of each face in the image must be at least 10% of the width of the image. We came up with this figure because after examining many photos, a face width that is 10% of the image is characteristic of a decent picture. Any less than this would mean that the person's face is out of focus in the shot, and it will appear as though we have taken a random picture. This requirement, in addition to our margin requirement, contributes toward the consistency aspect we wanted for our project. We will evaluate this requirement by examining every picture taken at the demo and confirming that the face width is at least 10% the width of the image.

Another requirement we've established is that 100% of the photos taken should be sent to the cloud folder over WiFi. We've set this requirement to ensure that our users received all the photos that were taken. This also helps us achieve the instant access to photos aspect we wanted for our robot. We will evaluate this requirement by recording the number of photos taken at the demo, and comparing it to the number of photos sent to the cloud folder. Originally we were attempting to use Google Drive, however, Dropbox is another platform with equivalent services that had easier integration, so we pivoted our solution towards using Dropbox instead.

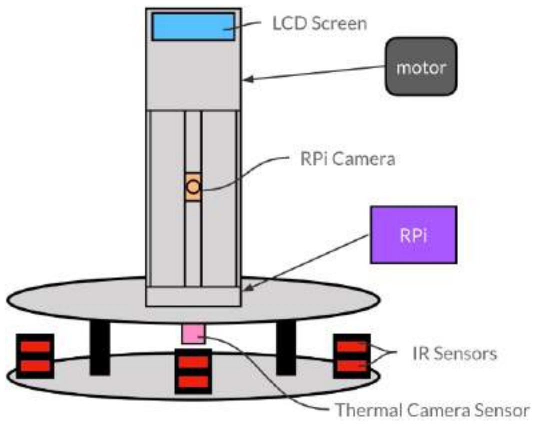
Our final product's success values are discussed in the final section of this report called Summary.

III. Architecture and Principle of Operation

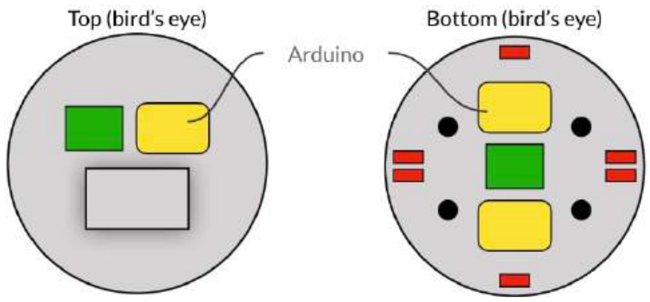
Below are diagrams of our system.



(a)



(b)



(c)

Fig. 2. System picture. (a) overall system. (b) zoom-in of top tower. (c) bird's-eye view of the tower's 2 platforms

For our mobile robot, the iRobot Create 2 Programmable Robot acts as the moving base, using its 6 built-in IR sensors to detect walls.

Our software-related system architecture revolves around the use of a Raspberry Pi, which is the controlling computer for the entirety of our software components. Hosted on the Raspberry Pi, are our motion control, image optimization, and collision detection software algorithms. Specifically, we used the PyCreate2 library to program the movement of the roomba. For photo optimization, we used OpenCV datasets on frames from a continuous video stream at 30fps to detect faces and ran our custom algorithm to determine when and where to capture photos. In addition, we used three Arduino UNOs to receive and process the thermal camera sensor and IR sensor signals over GPIO pins which relayed the data to the Raspberry Pi.

For photo capture, we used the Raspberry Pi Camera Module, which easily integrates onto the Raspberry Pi and is able to capture 8MP photos.

For collision detection, we originally intended to use 2 thermal cameras (1 facing the front, 1 facing the back) and 18 IR sensors. Since the design report, we pivoted to only using 1 thermal camera sensor and 12 IR sensors to detect when people or objects are in close proximity, so that our robot can stop moving and avoid collisions. We reduced the number of sensors because stacking them were getting to be too unstable. In addition, we realized we didn't need the back-facing thermal camera anymore - if our robot attempts to move back to adjust for margins, it only needs to know if something is there, human or not. In our final product, 6 IR sensors face forward and 6 IR sensors face backwards to detect objects when the robot moves forwards and backwards, and the single thermal camera sensor sits at the front of the robot just under the center IR sensors to determine when a human is in front of the robot.

For further control over image capabilities, we used a stepper motor and a custom-built wooden track for our camera to move up and down as specified by our image optimization algorithm. Our Raspberry Pi communicates with the stepper motor using the Arduino and GPIO pins as its communication channel. Our design report initially stated we would be using a telescoping or scissor-lift mechanism to move our camera up, however, this proved to be difficult and expensive to obtain. We created our own custom motorized camera track instead.

Lastly, we used an LCD screen, connected to the Raspberry Pi, to display a countdown prompt to indicate when a photo is about to be captured.

The block diagram on the next page (Fig. 3) demonstrates all of the above described components, as well as how they communicate and how information flows in our system.

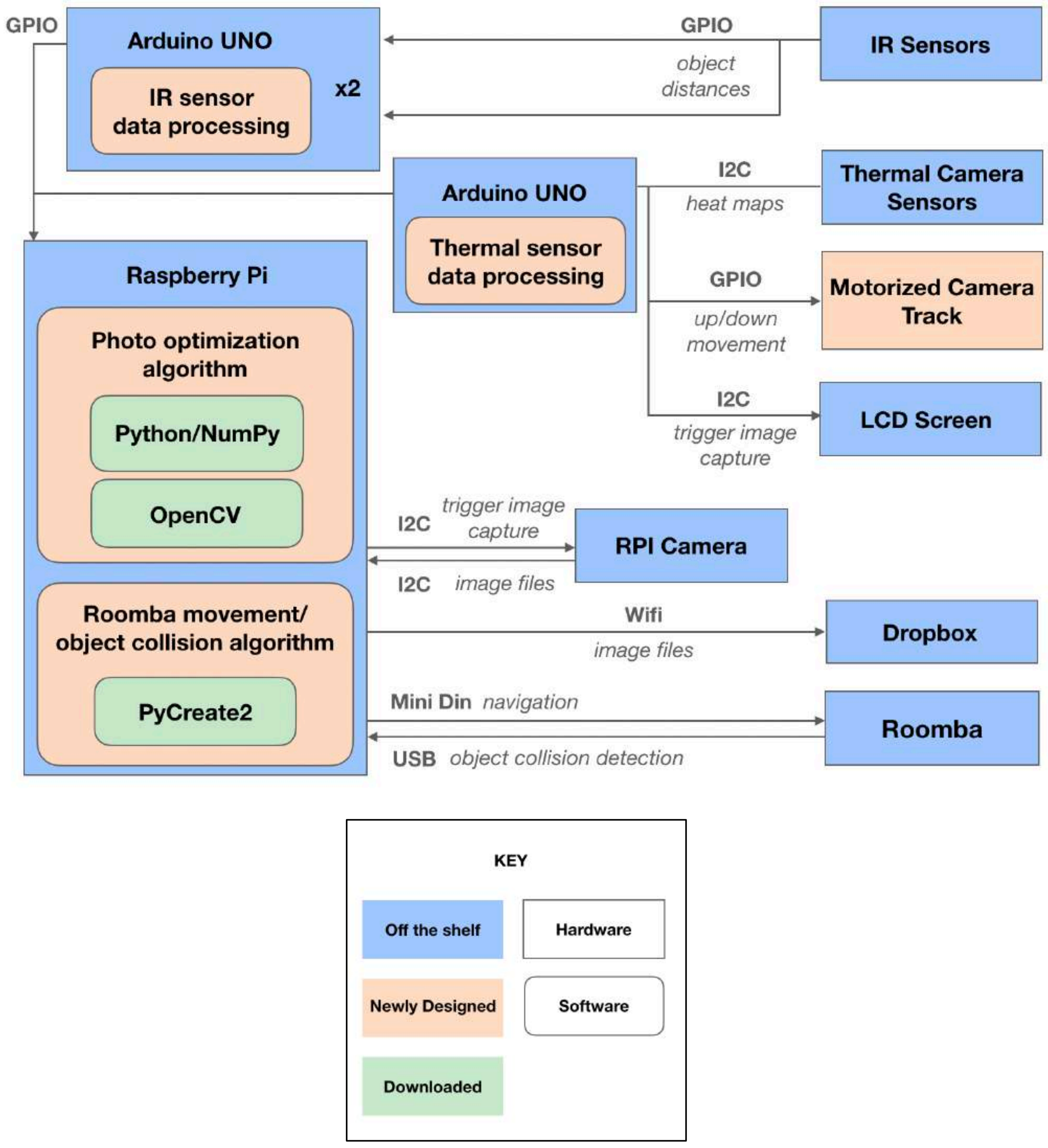


Fig. 3. Block Diagram with key

IV. Design Trade Studies

We decided to use the Sharp GP2Y0A02YK0F IR proximity sensor for obstacle detection because it was advertised as reliably providing data between a range of .5 feet to 5 feet. Since we are looking for obstacles within 3 feet, this sensor shows a clear advantage over other IR proximity sensors such as the UNCL4010 proximity sensor, which can only accurately detect collisions up to 7.5 inches away. We also found that this sensor's ability to bounce reflective light off the clothes of a human make it better than ultrasonic sensors. We put 2 IR proximity sensors on 6 different positions along our robot. We kept track of each last value reported by each sensor, and utilized a voting system between the 2 sensors in each position. We only acknowledged an IR trigger signal if it was high for 2 iterations of the sensor processing loop, and if both IR sensors in the same position were triggered. Doing this allowed us to reduce the chance of encountering outlier data coming from a sporadic sensor that occasionally exhibits erratic behavior. We decided on 6 IR sensor positions because we needed 2 facing front on the left, center, and right; and 2 facing back on the left, center and right. Since the robot would only need to move forward or backwards, we reasoned that we would only need to account for these 2 directions. The left, center, and right placements are meant to account for collisions that would happen either directly in front of/behind the robot, or at the edges of the robot. So with the 2 sensors per position, and 6 position on the robot, we needed 12 IR sensors.

In addition to the IR proximity sensor, we will be using the Adafruit AMG8833 8x8 Thermal Camera Sensor. We selected this particular sensor because it is able to detect heat signatures a far distance away (ie. up to 23 feet) and it has a fairly decent field of view (ie. 60°). This sensor was used in conjunction with our IR sensors to distinguish inanimate obstacles (eg. wall) from humans. For our collision detection mechanism, we need the IR proximity to detect for collisions, and we need the thermal camera sensor to determine whether that collision is a human or an object. We decided that we only needed one thermal camera sensor because only need to detect for humans in front of our robot. With the 60° field of view, we can capture what is seen by the 6 IR proximity sensors facing forward.

This diagram (Fig. 4) describes how we planned for the sensors to be positioned (red rectangles are the thermal camera sensors while yellow squares are the IR sensors). Refer to Fig. 2b and Fig. 2c for our final placement of sensors.

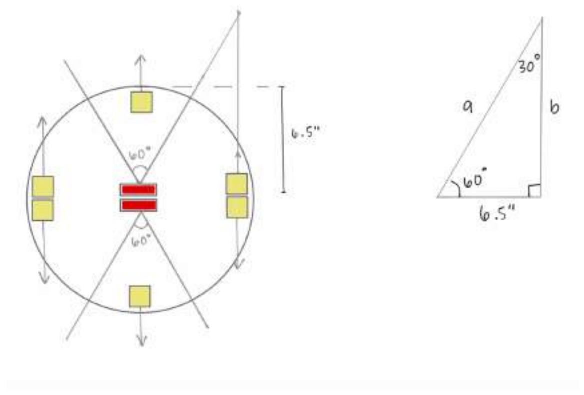


Fig. 4. Old sensor placement.

The distance between the intersection of the thermal camera sensor's field of view and the IR proximity sensor's field of view must be less 29.5" (36" - 6.5"). This is to ensure that the thermal camera will detect anything that is 36" away from the left or right IR proximity sensors. Our original design had our thermal camera sensor closer to the center of the robot, but even if we move our sensor to the front edge of the robot, we can meet this distance requirement. We know the angle between the right IR proximity sensor and the right side of the thermal camera's field of vision is 60°, and that distance between the IR sensor and the thermal camera sensor is 6.5", we can determine that the intersection point is 11.2583" away, which is less than 29.5". Thus we can conclude, that we only need one thermal camera sensor per direction.

$$a = 13" \quad \text{(Eq. 6)}$$
$$b = 11.2583" \quad \text{(Eq. 7)}$$

We had enough slack such that even after we pivoted the number of thermal sensors and moved the placement to the front instead of the center of the platform, the thermal camera and side IR sensors still intersect when a person is 3 ft away from the robot.

We decided to use 3 Arduino UNOs: 2 for IR sensor signal processing, and 1 for thermal camera signal processing, camera motor control, and LCD screen control. By putting signal processing on the Arduinos we are able to free up the Raspberry Pi's computational power for image processing which needs to be as fast as possible. The Arduino UNO has 6 pins for transmitting analog data, and since we are using 12 IR proximity sensors for collision detection, we need 2 Arduinos for that. Since LCD screen control and thermal camera signal communication happens over I^{2}C, the final Arduino has enough pins for thermal camera signal processing, camera motor control, and LCD screen control.

We decided to use the Raspberry Pi 3 Model B, because it is WiFi and Bluetooth capability. By having WiFi capability, we are able to easily debug our code by simply powering the Raspberry Pi on campus and being on the CMU-DEVICE network. By having Bluetooth capability, we can pair our motorized stick with our Raspberry Pi, in order to move the camera up and down. This model also has a CSI port which allows us to seamlessly integrate our Camera module. Lastly, this model has the most USB ports (ie. 4), which works perfectly because we will need 4 ports to connect to the 3 Arduinos and the Roomba.

We decided to use the Raspberry Pi Camera Module V2 because of its ability to take 8 MP pictures, which we deemed sufficient for our purpose.

V. System Description

A. Flow

Although there are many different scenarios we have considered to be potential contexts for our robot, the behavior follows a general pattern of roaming, sensing for humans, adjusting, and capturing a photo. More specifically, Camerazzi begins roaming in a straight path until it either senses a human or collides with a wall or other object, using its thermal and ir sensors. In the case that Camerazzi senses a human, it uses OpenCV and the Raspberry Pi camera to scan for faces in view, moving the camera up and down. If faces are not detected, the robot will turn and continue roaming.

If one or more faces are detected, our algorithm runs a series of calculations to determine if the photo should be taken and how the robot should adjust its position and camera height. The details of this algorithm are detailed in Section C where we describe our photo optimization subsystem. Before photo capture, we display an output of "3...2...1… Smile! =)" onto an LCD screen mounted on the front of our robot to inform the subjects of the photo that a photo will be taken. In the case that Camerazzi collides with a wall or other object, it rotates by increments of 60º to the opposite direction and continues moving. Lastly, photos are immediately transferred to a Dropbox folder after they're captured.

Below we dive deeper into the specifics of each subsystem which allow us to achieve this overall behavior.

B. Susbsytem – Roomba Movement

Our robot moves via the Roomba as its base, communicating with the Raspberry Pi controller over Mini Din. In doing so, we must account for collisions with any object or wall and redirect the robot before contact. For collisions, the Roomba has built-in sensors we are utilizing. The table (Table I) below displays the sensors we interacted with and the range of their values that the sensor is able to provide.

TABLE I. RELEVEANT ROOMBA SENSORS

Sensor	Range	Index
bumps_wheeldrops	[0-15]	0
light_bumper_left	[0-4095]	36
light_bumper_front_left	[0-4095]	37
light_bumper_center_left	[0-4095]	38
light_bumper_center_right	[0-4095]	39
light_bumper_front_right	[0-4095]	40
light_bumper_right	[0-4095]	41

Using these sensors, our robot can detect a wall or floor barrier before it bumps into it. In order to avoid forces that may shake our robot structure upon collision, we wanted the robot to stop at least 1 cm away from walls. In addition, the robot should be rerouted by turning and moving forward away from the wall it just detected, thus avoiding contact and continuing its mission of taking photos.

The bumps_wheeldrops sensor's 0th and 1st bits are binary indicators for the state of the left and right bumpers. These should remain 0 throughout our robot's operation to avoid actually hitting a wall or object and shaking the robot. Because IR sensors are unreliable against certain materials as well as while the object in front of it is moving, the Roomba is sometimes unable to detect feet that are walking near it. This means that our robot has the potential to bump into people, especially on the sides where no IR sensors at the top of the robot are detecting obstacles. In this scenario, any time bumps_wheeldrops returns anything that is not a 0, meaning that the bumper was pressed and contact has been made, the robot pivots 180º and moves directly away.

There is a series of light_bumper values that returns data for each of the six sensors built into the Roomba, facing the front. Each of these sensors uses IR to retrieve the distance between it and the object it's detecting. The sensors are orientated in this way:

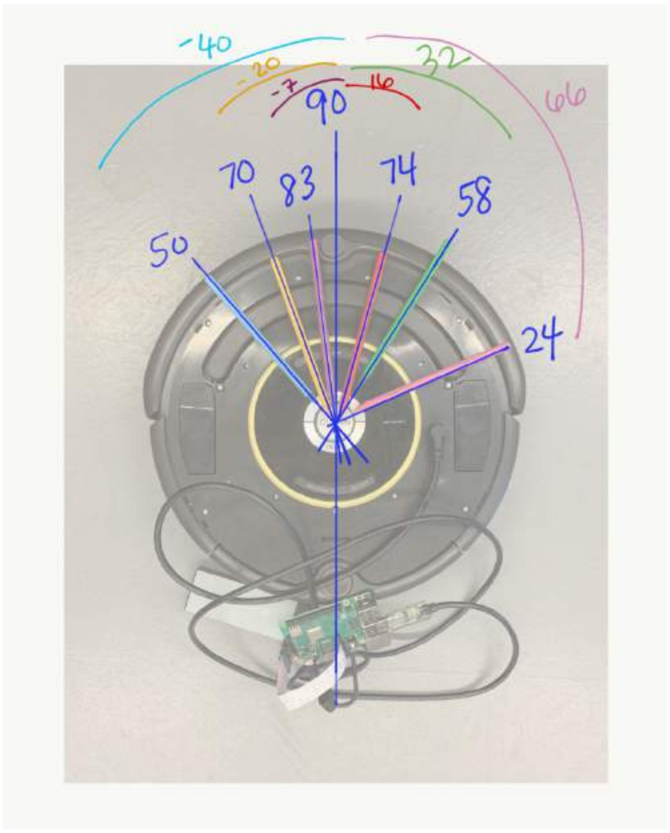


Fig. 5. Roomba sensor locations.

Setting 0º as directly center forward, the leftmost sensor is at -40º. Moving from left to right, the next sensor is at -20º, and the next sensor is at -7º. To the right of center, the next sensor is at 16º, then there is another sensor at 32º, and the last rightmost sensor is at 66º. Our goal in knowing the precise locations of these sensors is to turn 60º away from the sensor that gets triggered. More specifically, if it's any of the 3 sensors on the left that is triggered, the robot will turn 60º to the right from that sensor. If it's any of the 3 sensors on the right that is triggered, the robot will turn 60º to the left from the sensor. After turning away, it drives straight. This angle choice was to keep in accordance with the human collision detection algorithm as well as allow the robot enough angle to move forward without sensing or hitting the same wall (unless the robot is in a corner, then it might have to turn more times to move out of the corner).

In accordance with the flow of Camerazzi, our robot will use the IR light bumper sensor values to turn and try a new path. Since the thermal camera sensors have a 60° field of view, the robot will turn right 60° to get a new view without overlap of the old view in order to scan for high human-like thermal signals. In essence, the robot will attempt to avoid detecting the same humans and taking their photos over and over. The robot should attempt to move towards a thermal signal to take photos of humans. If it repeatedly senses a wall, it will continue to turn to the right 60° at a time until it no longer senses a wall or object, and/or senses a human through thermal readings, and has a clear path to move forward. It is not turning 180° and simply moving away from the wall or obstacle because this may result in the robot bouncing back and forth along the same straight path in the room if its path just so happens to be unobstructed.

A major portion of our robot is detecting humans, for both taking photos of and avoiding collisions between our robot and humans. This section will outline the avoidance of collisions. (Refer to Photo Optimization Subsystem for description of detecting humans for purposes of taking photos.)

The robot uses both IR and thermal camera sensors for human detection, with the IR sensors' output pin connecting to analog pins on the Arduino (GPIO) and the thermal camera sensors communicating over I^{2}C with the Raspberry Pi. There are 3 sets of sensors facing the front of the robot and 3 sets of sensors facing the back of the robot (more details about number of sensors in Trade Design section). With the 13-inch diameter of the Roomba, having one set of sensors on the left-most side, one set in the middle, and one set on the right-most side will be able to detect any human that is standing in front of or behind the Roomba, even if the human is positioned perpendicular to the sensors. After detecting something is in front of the Roomba, the values we get from the IR sensors tell us the distance the human is away. The goal was to stop at least 3 feet away from any human (refer to Design Requirements section). Knowing that these voltages correspond with these output values from the IR sensors through the Arduino:

TABLE II. IR SENSOR VOLTAGE CORRELATION

Voltage	output from IR sensor through Arduino
0 V	0
5 V	1023

then 1 V corresponds to a value of 204.6 form the IR sensor through the Arduino.

$$3 \text{ feet} = 91.44 \text{ cm} \quad \text{(Eq. 8)}$$

$$\frac{2.5\,V - 0.4\,V}{150\,cm - 20\,cm} = \frac{21\,V}{130\,cm} = 0.01615385\,V/cm$$

(Eq. 9)

$$0.01615385\frac{V}{cm} \times 91.44\,cm = 1.47710804\,V$$

(Eq. 10)

$$1.47710804\,V \times 204.6 = 302.216304 \approx 302$$

(Eq. 11)

So once we get an output value of 302 from any of the IR sensors, the thermal camera sensor will then assist in determining if it's actually a human or if it's just a wall. If the data from the thermal camera tells us there's heat signature, the robot will stop and take a photo based on the Photo Optimization subsystem description. The camera has about a 60° field of view, so after it takes a photo, the goal is for it to discover a whole new view in order to detect and take photos of new subjects. Thus, after it takes a photo, it will rotate 60° to start with a new field of view. The robot will alternate turning left and right 60° in order to avoid going in circles if no bodies are detected.

Unfortunately, our IR sensors ended up only being able to detect around 18-20 inches, which is a lot lower than their documentation reported.

After a human is detected and the camera feed is analyzed, adjust according to instructions from photo optimization algorithm (see Photo Optimization Subsystem for details).

C. *Subsystem – Photo Optimization Algorithm*

Another main subsystem of Camerazzi is the photo optimization algorithm which determines when and where photos are captured. Specifically, our photo optimization algorithm uses data from the camera, thermal camera sensor, and IR sensors to determine whether it is in the right position to take a photo at a given time. As a result of the data, the robot moves forward or back, and the camera moves up and down on our motorized track, in order to satisfy our requirements for ideal photos. The thermal and IR sensors are used to identify when we believe there is a human in view of the robot. When a human is detected, the roomba stops and our face detection algorithm begins searching for faces using live video data from the camera.

If faces are detected, our photo optimization algorithm checks multiple parameters to determine how to adjust and capture the photo. First, we check if all faces are above the minimum width requirement of 10% image width. As a second check, we determine whether or not there is enough margin (10% image width) around all faces. If either of these requirements are not met, we adjust the position of the Roomba to attempt to satisfy them. For example, the Roomba moves forward if a given face width is too small, and backwards if the face width is too large. Similarly, if there is too little margin around a face to the left or right, the Roomba moves backwards, and if there is too much margin, the Roomba moves forwards. Lastly, if there is too little margin above or below a face, the motorized stick moves up or down accordingly. Due to the fact that adjusting to one constraint affects the value of the other variables, we have come up with a protocol that allows us to determine how the robot adjusts, or if it should turn and continue moving because the requirements cannot be satisfied. In the order of face width, horizontal margin, then vertical margin, Camerazzi adjusts its position or camera height slowly until the requirement is satisfied. After the given requirement is satisfied, we check if earlier requirements were made invalid due to the following adjustments. In that case, the ideal photo is not possible by our requirements, so Camerazzi turns and continues roaming. This protocol for adjusting the robot for ideal image capture provides a straightforward approach to handling edge cases and determining whether photos should be taken or not.

D. LCD Screen and Motorized Camera Track

Other components in our system include the LCD display and the motorized stick.

The LCD display is the robot's way of communicating to the people in front of it that it is going to take a picture. It is connected to an Arduino over the I^2C interface and will go through the following sequence: "3!", wait 1 second, "2!", wait 1 second, "1!", wait 1 second, "SMILE!" By providing this feedback to the human, the human will know when the photo has been taken.

The motorized camera track is a motor pulling on a string attached to the Raspberry Pi camera and will be used to adjust the camera's position to 3 different positions. These different positions assist in meeting the 5%+ margin requirement on the top and bottom of the image. If the bottom margin does not meet the 5% requirement, the Raspberry Pi will send a signal to the Arduino instructing the motor to pull up on the camera. Likewise, if the top margin does not meet the 5% requirement, the Raspberry Pi will send a signal to the Arduino instructing the motor to push the camera down. The camera will slide along a track on top of our robot to ensure that the camera doesn't wiggle from side to side.

E. Power

The iRobot Create 2 will be powered by its accompanying battery, which has a battery life of 3.5 hours.

Originally, we were going to have the Raspberry Pi supply power to the motor, the LCD screen, the sensors, the camera, and the Arduinos. Instead, we used a 3 10000 mA/h batteries to power everything. The battery powering our Raspberry Pi, camera module, one Arduino, the motor, the thermal sensor, and LCD screen dissipated power most quickly. With the Raspberry Pi and camera module with Wifi and Bluetooth on using 550 mA, the motor using 350 mA, the LCD screen using 120 mA, the thermal sensor using 50 mA, and an Arduino using 50 mA, we find maximum consumption was to be 1120 mA. With an ideal version of the batteries, we could provide power for a little bit over 9 hours. With a version that is 50% efficient, we will still be able to power our devices over 4.5 hours which is sufficient for our purposes.

VI. Project Management

A. Schedule

Leading up to the mid semester demo, our schedule followed our initial Gantt chart very closely. The only task we had to add was doing some assembly of our robot before the demo, which we hadn't initially accounted for. In addition, it took a little longer than expected for the LCD screen integration, Roomba movement, and collision detection, however we were able to finish all of the tasks before the mid semester demo. As a result of those extended tasks, we had to push back our task for adjusting based on image margins until after the mid semester demo. We were able to catch up on this task by working together in the following week. Lastly, due to structure redesigns, we had to spend a significant amount of time rebuilding the base of our robot structure leading up to the final demo, which left us less time for testing. We also had to add tasks for our demo barriers and final preparation before our public demo.

A simplified version of our Gantt chart is on the next page (Fig. 6).

B. Team Member Responsibilities

Mimi

As a software engineer, Mimi was responsible for the face detection portion of this project using OpenCV and NumPy on the Raspberry Pi. She also did image analysis to determine the margins of the current frame and send that data to Cornelia to move the base of the robot for adjustments. After movement, Mimi checked the margins again and then ultimately captured the photo. Then, she transmitted the photo to a Dropbox folder over wifi. Due to the complexity and scale of the physical structure of the robot, Mimi ended up doing a lot of construction as well - from cutting wood to fitting pieces together to soldering and wiring IR sensors to assembling the entire structure.

Cornelia

As a software engineer, Cornelia was responsible for the Roomba movement portion of this project as well as overall integration of sensor data and the photo optimization algorithm. Using PyCreate2, an iRobot Roomba opcode library, she programmed the Roomba's initial movement, processing thermal and IR sensor data for human collision detection in order to stop, and processing face detection/margin data from Mimi to adjust the Roomba's position. Due to the complexity and scale of the physical structure of the robot, Cornelia ended up helping with soldering, construction, and assembly as well.

Adriel

As a hardware engineer, Adriel was responsible for setting up the Raspberry Pi and Arduinos. He programmed the Arduinos to receive sensor data and send GPIO signals to the Raspberry Pi. More importantly, he was responsible for all the interconnects between the components of this project - the wiring of the Arduinos' GPIO pins to the Raspberry Pi which involved using logic-level shifters. He also worked on building and assembling the physical structure of the robot, including the motorized camera track, LCD display enclosure, soldering of IR sensors, laser-cutting the attachments between the Roomba and the robot's top.

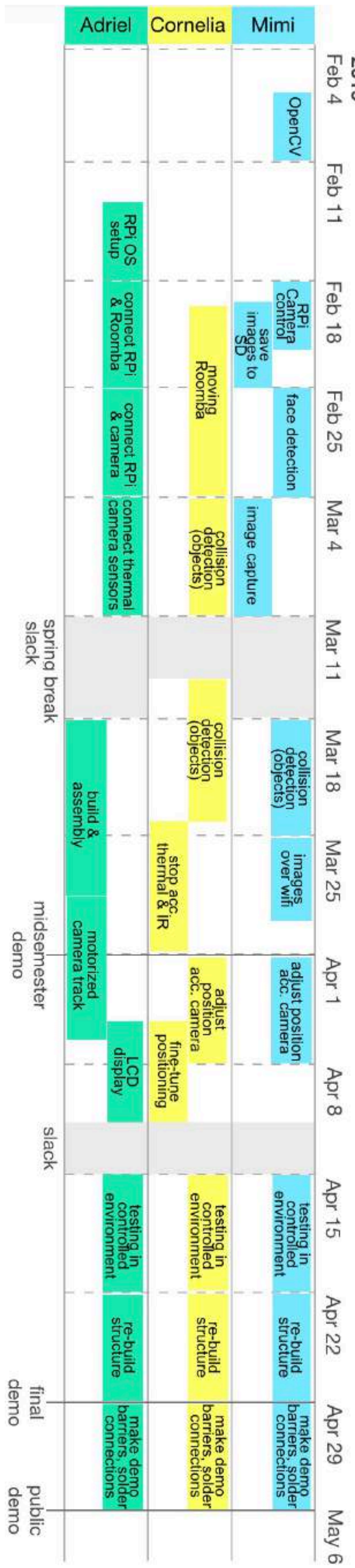


Fig. 6. Gantt chart

C. Budget

Our budget spreadsheet is located at the end of this report (Table III). Parts that we acquired without purchased is also located at the end of this report (Table IV).

Notable tools we used to accomplish our project include soldering iron, laser-cutter, band saw, wood filers, Visual Studio Code, and Arduino IDE.

D. Risk Management

There were a few major design risks that arose throughout the semester, relating to proximity sensing, robot stability, and photo quality. One of the most challenging elements of our project was finding a way to get accurate and reliable proximity sensor data. Throughout the semester, we had to mitigate the risk of our robot not being able to detect collisions with objects and people by finding the right proximity sensors and continually doing testing and improving our software. Some specific ways we mitigated against this risk included having backup ways to sense proximity (ultrasonic, IR, OpenCV), removing outlying data, and comparing sensor values in time and physical proximity to get more reliable data. We also soldered our IR sensor wires to a soldering protoboard before transporting our robot to ensure that connections would remain intact.

Another significant risk we faced was the stability of our robot. After testing with our initial design approach of a tripod base, we determined that the tripod would be too unstable to support our hardware components on the top of the robot. As a result, we had to redesign our structure to mitigate the risk of our robot falling over during activity. We did so by using wood, which is thicker and more sturdy than the thin plastic tripod legs, to construct a base with increased surface area on top and bottom. We also introduced padding and used hot glue to secure any loose or wobbly areas.

Lastly, photo quality was a risk that arose during testing as we came across blurry, crooked, and dark photos. To mitigate the risk of such photos being captured, we tried to address each of the source problems including poor lighting, camera position, and robot stability. First, we tested in our demo space to ensure there would be appropriate lighting. We also introduced pauses in our program to allow our camera time to focus on subjects. Lastly, as stated above, we improved our robot's stability to avoid blurriness of photos.

The design risks mentioned above introduced some schedule related risks, as we had to adjust our tasks and catch up on work as we updated our design. To mitigate any risks of not meeting the deadlines we'd set for ourselves in our Gantt chart, we worked outside of class each week to catch up on additional tasks that had yet to be completed. We also worked as a team and helped each other accomplish any tasks that were taking longer than expected. In addition, we updated our Gantt chart weekly to accurately demonstrate our completed and scheduled work so that we could stay on track and plan for the future more precisely. Fortunately, we had no major risks related to our budget, as we were able to borrow many resources from around campus.

VII. Related Work

A similar project we found when doing research for our project is called the "Roomberry Surveillance Robot," which uses a Roomba, and Raspberry Pi Zero, and a camera, to send video through a web interface [1]. This project has many similarities to our project in terms of the hardware components it uses, and the functionalities it achieves such as a movable camera, saving photos, and serving up photos through Wi-Fi. However, this project lacks a lot of the more complex features we hope to include, such as optimizing photos and the Roomba position for human faces, and taking care of collision detection.

A similar product we found on the market is the "Double Robotics Double 2 Telepresence Robot," which is used for virtual interaction with remote individuals [2]. This robot is similar to our robot in shape and size, as well as the capability to take photos with a 5MP camera. However, this telepresence robot varies drastically in terms of cost and movement, as it is priced over $3000, and must be remote controlled for movement.

A product with a similar use case to our project is called the "Polycam Player," which was developed by Nikon company MRMC, and is designed to take the place of cameramen who specifically capture action shots in sports settings [3] [4]. The system uses face and limb detection to track players and get close up action shots and video of game play. This project shares many similarities with our system in terms of using feature detection to capture quality photos of humans.

Lastly, an article that we found, called "Autonomous Mobile Robotics Research for Daily-Life Environment," details the research of a university team that experimented with multiple types of robots in different environments [5]. This article was informative to us, due to its explanation of key functions and behaviors of indoor, autonomous, mobile robots.

VIII. Summary

We achieved our high level requirement. Across multiple testing sessions and our final public demo, our robot stopped to attempt photo capture an average of 41 times per hour.

Table V at the end of this document describes our metrics and validation from before the final demo.

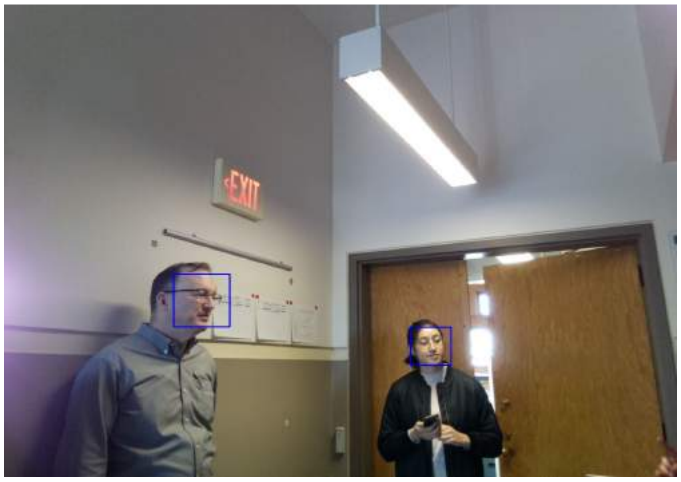
In order to measure the collision detection mechanism of our robot, we set it 6 feet away from a human. Then, we ran code that would have the robot move forward and stop when IR sensors and thermal sensors were triggered. The distance from the torso of the human to our robot was measured in each trial. We also let the robot approach the human from different angles. We found the average value of our trials was ~18 inches. We set the IR trigger value to be 18 inches, because even with no obstacles in front of our robot, the IR sensors was only able to report a value of 18-20 inches. Increasing our IR trigger value would result in many more false positives and our robot would never be able to take a picture. The IR sensors did not meet the advertised sensing distance, and thus we were not able to satisfy the success value we set for ourselves. If we had more time, we would have liked to test different sensors.

In order to measure stopping latency, we allowed the robot to run its algorithm in front of a human. We used a stopwatch to measure the time between the program printing that it detected a human and the robot actually halting. After 10 trials, we found the average to be around 0.14 sec which is well below our maximum latency requirement of 1 second.

In order to determine the face detection and photo capture mechanisms, we allowed our robot to roam the testing environment until 100 photos were taken. For testing, photos were taken with a blue rectangle outlined around what OpenCV identified as a face. The photos were analyzed after the 100 photos were taken, and the percentage of faces correctly outlined was recorded (face detection) and the percentage of photos with faces was recorded (photo capture). We found that out of the 100 photos taken, 83.6% of the faces were correctly outlined, and 92.7% of the photos had actual faces in them. Face detection was testing OpenCV accuracy - online documentation reported a 90-95% accuracy for still images, but since we're running it on frames from a continuous stream, we weren't sure if OpenCV would still result in the same accuracy. Even though we lowered our success value, we still found that its accuracy was lower than what we expected. This difference could be attributed to poor lighting conditions, camera movement causing blurriness, and low photo resolution which resulted in false positives (windows being labeled as faces) and true negatives (faces that did not get detected). The photo capture metric was testing the overall robot accuracy, from detecting a face, adjusting the robot's position, to actually capturing the photo. Due to humans moving between face detection and photo capture, our tested value was lower than our ambitious 100% expected value.

For image margins, we compared the number of rectangles outlining faces with enough margin (5%+ pixel-count) to the number of rectangles without enough margins. Our photos have a resolution of 1296px by 972px. 5% is about 65px for the left and right margins and about 48px for the top and bottom margins. We were able to meet our image margin requirement on 100% of the faces that were accurately identified by OpenCV.

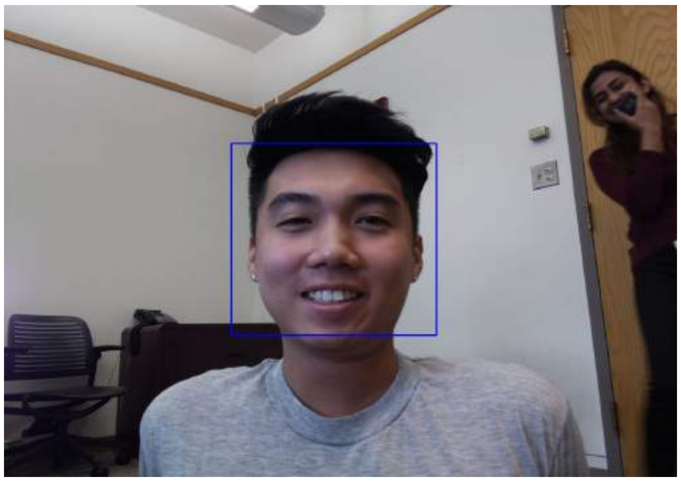
The following are some of the photos our robot was able to capture during testing (with the blue rectangles we used to outline faces that OpenCV detected) and our final public demo:



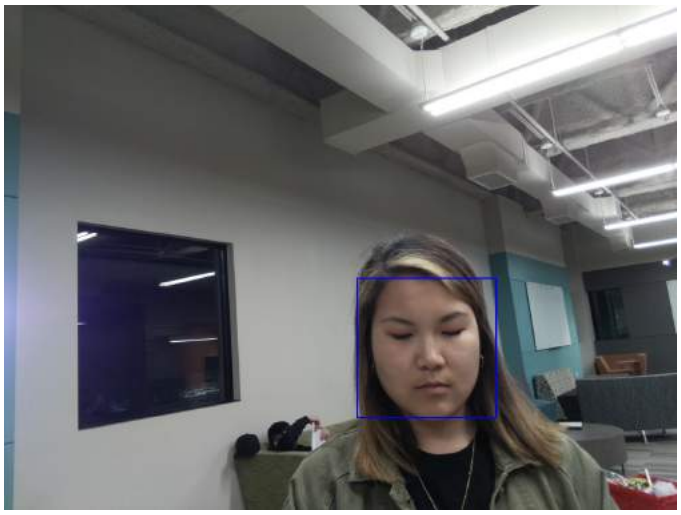
(a)

(b)

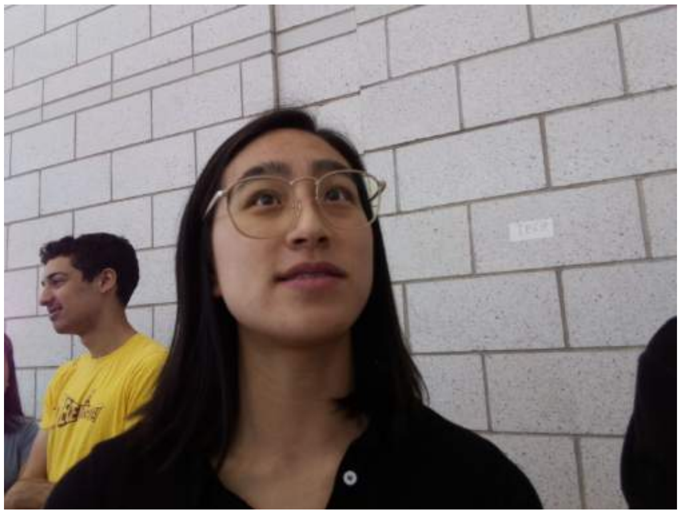
(e)


(c)

(f)


(d)

(g)



(h)

Fig. 7. (a-d) in-lab testing with blue outlines. (e) testing without blue outlines, multiple faces in 1 frame. (f-h) final public demo without blue outlnes.

Our robot consistently uploaded each photo it captured which satisfied the margin requirements to a Dropbox folder, reaching our 100% image upload requirement. As long as a decent WiFi connection was established on the Raspberry Pi (which was 100% of the time we were on CMU campus), anytime our program said a photo was captured and uploaded, it would appear in the cloud folder.

Apple sent 3 engineers to attend our final demo and was kind enough to sponsor prizes to award the top teams. Our team received an Honorable Mention. It was really nice to be recognized for our hard work and be 1 of the 6 teams invited to demo to engineers at Apple headquarters in Cupertino over video-chat.

A. Future work

Given more time to work on Camerazzi, some improvements we'd like to make include further strengthening of our robot structure, integrating more reliable and far reaching IR sensors, using a higher resolution camera, and adding a light ring to ensure good lighting. In addition, we would like to fine tune our algorithm to more efficiently adjust the robot's position before taking a picture, to reduce the time spent in front of a human, and increase the number of photos that can be taken in a given time span.

B. Lessons Learned

We learned a lot of valuable lessons over the past semester. One that really helped us in terms of budget was scavenging the parts we needed from other projects, classes, departments. Having a clear understanding of how long each specific task will take is also extremely useful to allow for more accurate scheduling and reduce the risk of any bottlenecks. Because parts orders are only placed every other day, even delaying the form submission by a day can put you behind, so don't procrastinate on filling out order forms! This will ensure timely delivery of materials you need and keep you on schedule. One lesson that's more specific to our project is to test the sensors you're considering before ordering them in bulk to ensure that they meet your design requirements and to take into account the method of connection needed for sensors. We had a lot of problems soldering our sensors and getting good connections from them and would've used other sensors had we known the ones we ended up getting would be so hard to work with.

REFERENCES

[1] https://www.hackster.io/danimaciasperea/roomberry-surveillance-robot-Roomba-pi-zero-camera-c056f9
[2] https://www.bhphotovideo.com/c/product/1296165-REG/double_robotics_1012dr_universal_360_camera_mount.html
[3] https://www.technologyreview.com/the-download/610711/a-robotic-camera-system-films-sports-like-a-human-does/
[4] https://www.techradar.com/news/nikons-robotic-cameras-are-designed-to-give-sports-teams-the-competitive-edge
[5] https://www.sciencedirect.com/science/article/pii/S1474667017331099
[6] http://www.cs.cmu.edu/~illah/CLASSDOCS/p331-mumm.pdf
[7] http://www.gkstill.com/Support/crowd-density/CrowdDensity-1.html

TABLE III. BUDGET SPREADSHEET

Part	Source	Quantity	Unit Price	Total Price
Raspberry Pi 3 Model B+	Amazon	1	$42.99	$42.99
Raspberry Pi Camera Module V2	Amazon	1	$24.68	$24.68
Sharp GP2Y0A02YK0F IR proximity sensor	RobotShop	17	$13.02	$221.34
Adafruit AMG8833 8x8 Thermal Camera Sensor	Amazon	2	$43.99	$87.98
Micro SD Card	Amazon	1	$11.39	$11.39
USB extension cord	Amazon	1	$7.07	$7.07
Tripod	Amazon	1	$14.99	$14.99
Balsa Wood	Amazon	1	$12.75	$12.75
Stepper Motor NEMA-17	Adafruit	1	$19.95	$19.95
Motor Breakout Board	Adafruit	1	$9.34	$9.34
Camera Ribbon	Amazon	1	$7.95	$7.95
Barreljack-to-USB (3-pack)	Amazon	1	$5.95	$5.95
PVC Pipe	Home Depot	1	$3.93	$3.93
Plywood 12x12x.25 (2-pack)	CMU Art Store	1	$9.89	$9.89
Plywood .75”x2’x4’	Home Depot	1	$22.49	$22.49

TABLE IV. PARTS ACQUIRED WITHOUT PURCHASE

Part	Quantity	Contributor
Raspberry Pi 3 Model B	1	ECE Department
Logic-level shifter	2	Ideate Room
iRobot Create 2 w/ USB serial cable	1	Robo Club
Arduino UNO	3	Cornelia, ECE Department
Sharp GP2Y0A02YK0F IR proximity sensor	1	Ideate Room
Soldering protoboard	2	Makerspace
Power bank	3	Mimi, Cornelia, Adriel

TABLE V. METRICS AND VALIDATION

Tested feature	**Metric**	**Success Value**	**Tested Value**
Face detection	Percentage of faces detected correctly in real time	90%+	83.60%
Photo capture	Percentage of photos with faces	100%	92.70%
Image margins	Moves to optimal position to ensure image margins	5%+ margin all the way around	5%+ margin all the way around
Collision detection	Distance from human when it's detected	At least 3 ft away	18 in away
Roomba stopping latency	Time between human detection and Roomba halting	< 1 sec	< 1 sec
Image transfer	Images wirelessly transferred to designated folders	100%	100%