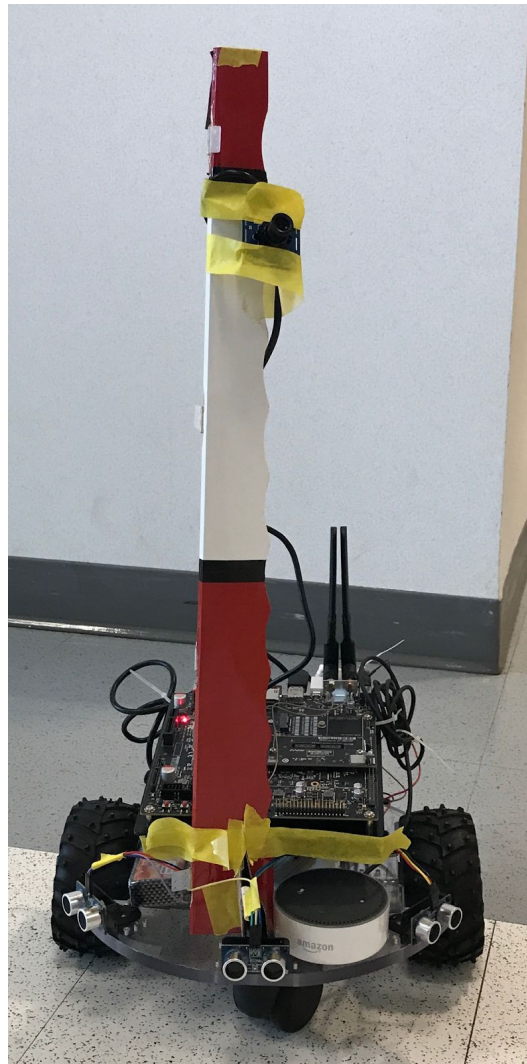


# Capstone Design Project: Autonomous Control of Interacting Robots



## Team 8

Omar Abdelkader  
Tauqir Abdullah  
Renee Adkins

# Table of Contents

<b>Introduction</b>	<b>3</b>
<b>Hardware</b>	<b>3</b>
<b>Software</b>	<b>4</b>
<b>Overview</b>	<b>7</b>
<b>Timeline</b>	<b>8</b>
Construction & Design	8
Goal	8
Implementation	8
Challenges	9
Controlled Motion & Obstacle Avoidance	10
Goal	10
Implementation	10
Challenges	11
Person Following	11
Goal	11
Implementation	11
Challenges	13
Person Identification	13
Goal	13
Implementation	14
Challenges	14
Bonus Features & Demo	15
Ambient Air Temperature Sensor	15
“Call for Help!”	16
<b>Conclusions</b>	<b>17</b>
<b>Resources</b>	<b>17</b>

# Introduction

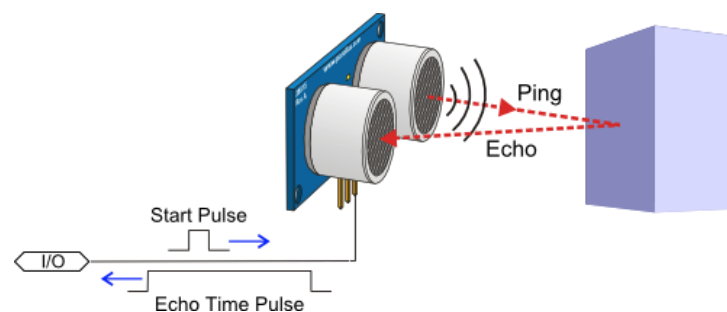
During the Spring 2018 semester, we worked together to design and implement an autonomous robot with onboard smart devices and sensor systems for the purpose of supporting the rehabilitation of disabled or elderly persons. The three of us are seniors studying Computer Engineering at the University of Maryland, College Park. Our experience in programming and circuitry from previous coursework allowed us to seamlessly context switch between various hardware and software tasks as they became necessary.

An assistive robot is one that performs a task or processes sensory information for the benefit of people with disabilities and older adults. In general, assistive robots compensate for some lost function(s). These include, but are not limited to but not limited to mobility, communicating, personal hygiene, child rearing etc. According to the International Federation of Robotics<sup>1</sup>, the total value of assistive robotics industry is expected to rise USD 97 million between 2016 and 2019.

The timetable for this project was planned around achieving several goals or milestones at different points during the semester. These milestones were to physically construct and design the robot, and achieve controlled motion and obstacle avoidance, person following, and person identification. In addition, each team was challenged to introduce additional features that made their implementation distinct or unique and demonstrate these capabilities during the final lab session of the semester.

## Hardware

The robot chassis is a circular plexiglass base about one foot in diameter. Attached on either side of the chassis are two gear motors with axles that easily connect to plastic wheels. In addition, a small wheel is attached at the front of the chassis to stabilize the robot. Motor signals are sent and received to the onboard Arduino via the Pololu dual motor controller. In addition, we equipped our robot with three front-facing Ping))) sensors to achieve persistent and accurate obstacle avoidance.

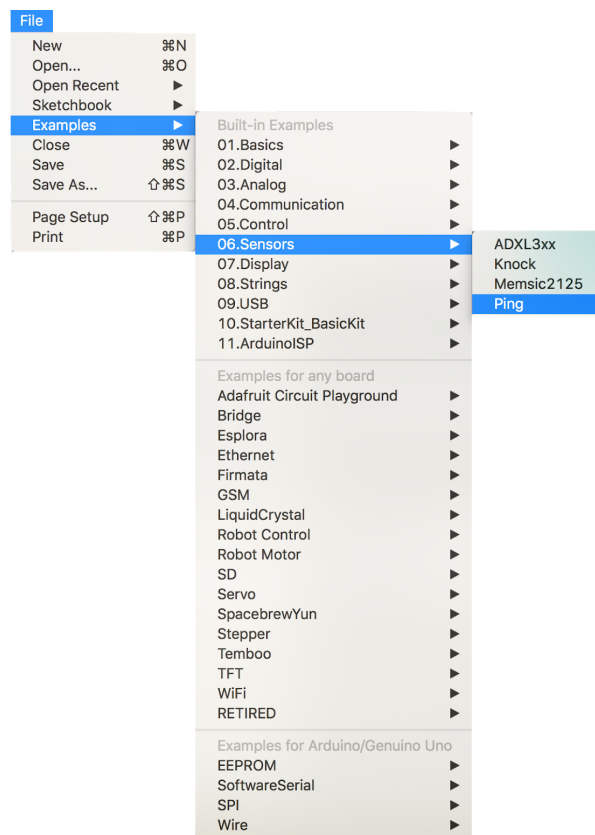


**Fig. 1** Ping))) Sensor mechanism

Computer vision and image processing efforts are handled by the onboard NVIDIA Jetson TX2 GPU and a simple USB camera. Voice-based interactions with the robot are done using the onboard Amazon Echo Dot. Finally, we integrated a standard NTC thermistor to measure the ambient air temperature of the robot's surroundings for one of our bonus features. All the hardware components were powered by a single rechargeable 12V battery.

## Software

All Arduino programming was written in C/C++ using Arduino's IDE. This was ideal because it allowed us to display log statements directly to the serial monitor during testing. In addition, basic templates and boilerplate code for operating the Ping))) sensors are available in the Arduino library.



**Fig. 2** The Arduino IDE comes with prewritten Ping))) sensor code underneath the File > 06.Sensors > Ping tabs

For Python, we employed a variety of different software libraries and tools to operate our robot. For image processing, we used OpenCV, an open-source library primarily designed for real-time computer vision. OpenCV is widely adopted and has great user support and documentation, which made it a logical choice to incorporate into our workflow.

For Amazon Echo integration, we used If This Then That (IFTTT), an online platform for integrating various apps and devices together using a series of triggers and actions.

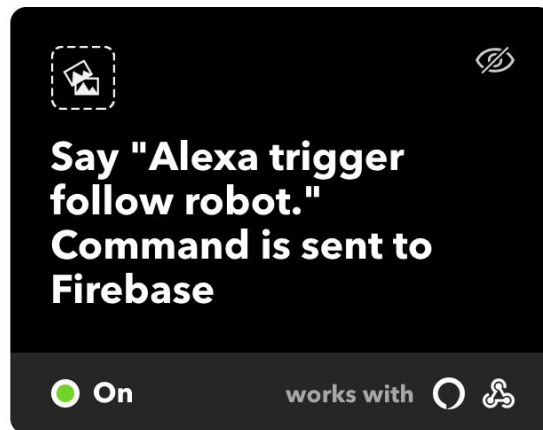


Fig. 3 Example IFTTT applet

We used Google Firebase as an intermediary to store our Amazon Echo commands off device in order to best integrate with IFTTT's infrastructure.

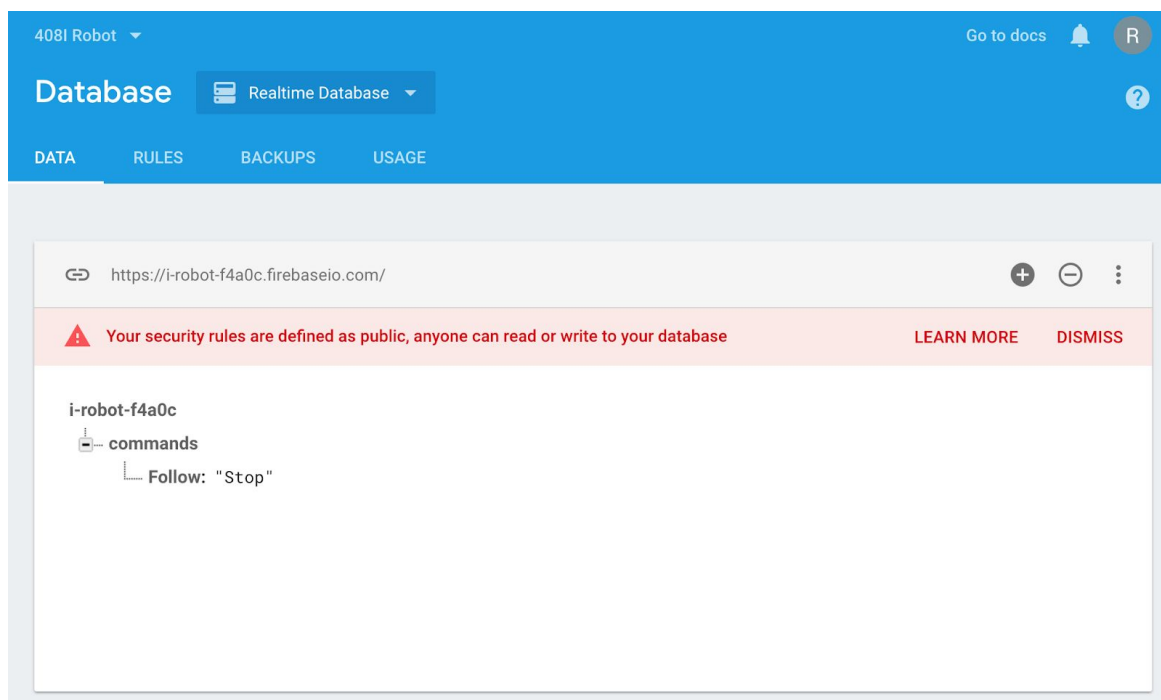


Fig. 4 Example Snapshot of Firebase

Each applet used for directly controlling the robot's motion is sent to Firebase with the corresponding command. Figures 3 and 4 show one of the applet's used to control the robot's motion and the corresponding command in Firebase. When saying "Alexa trigger Robot Stop,"

the JSON object {Follow: Stop} is sent to Firebase. The command is fetched by the NVIDIA Jetson via Firebase’s Python API and then processed in the main navigation algorithm.













Person Following	<div><div><div><div></div><div>Say "Alexa trigger follow robot." Command is sent to Firebase</div><div><div><div><div></div></div><div>On</div></div><div>works with</div><div><div><div></div></div><div></div></div></div></div></div><div><div><div><div></div><div>Follow Tauqir</div><div><div><div><div></div></div><div>On</div></div><div>works with</div><div><div><div></div></div><div></div></div></div></div></div></div><div><div><div><div></div><div>Follow Renee</div><div><div><div><div></div></div><div>On</div></div><div>works with</div><div><div><div></div></div><div></div></div></div></div></div><div><div><div><div></div><div>Robot Come Here</div><div><div><div><div></div></div><div>On</div></div><div>works with</div><div><div><div></div></div><div></div></div></div></div></div></div></div></div>
Action	<div><div><div><div></div><div>Action: Emergency Help Text &amp; Call</div><div><div><div><div></div></div><div>On</div></div><div>works with</div><div><div><div></div></div><div></div></div></div></div></div></div>
Features	<div><div><div><div></div><div>Temperature Gauge</div><div><div><div><div></div></div><div>On</div></div><div>works with</div><div><div><div></div></div><div></div></div></div></div></div></div>

Fig. 5 IFTTT Applets used for our robot implementation

IFTTT was not the only option for communication with the Alexa. Another framework was Flask-App which uses a localhost to communicate with the Alexa via ngrok. The IFTTT framework was able to accommodate the Echo as well as the thermistor that was used to measure the temperature in a room. A hotspot was used with the Alexa since it isn’t built to be used with wifi networks that require credentials. Other teams were able to get the Echo working

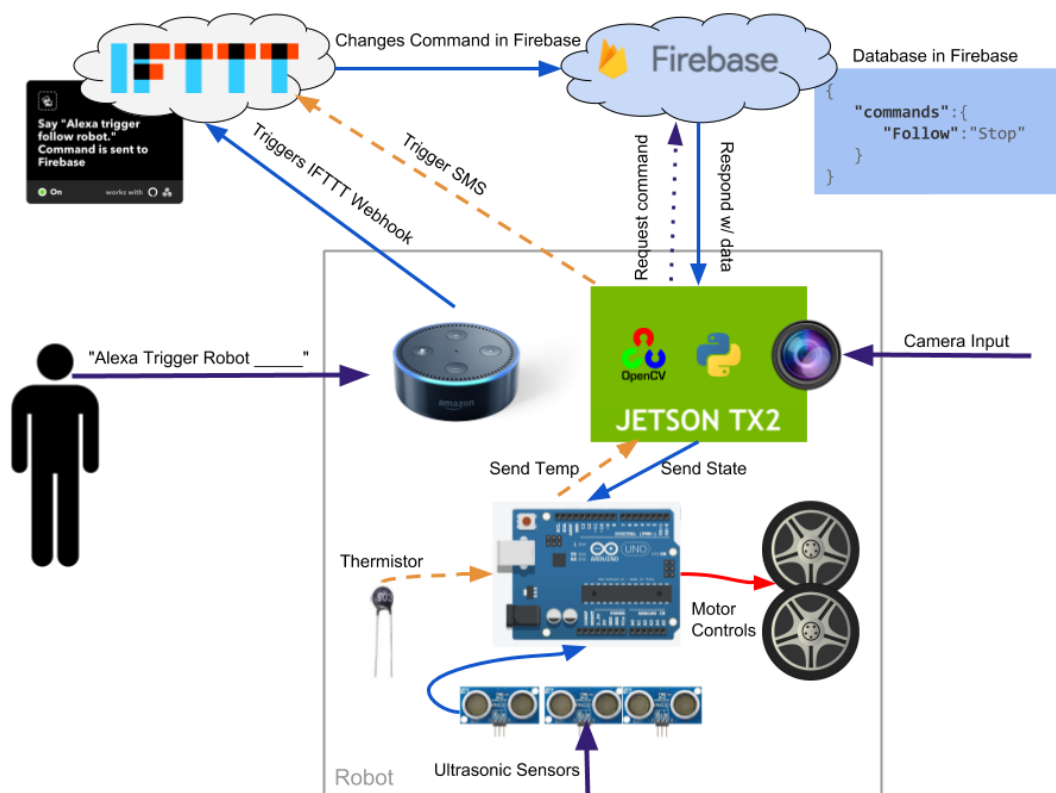
with umd instead of umd-secure, but we could not, so we used a hotspot for the entire semester.

Finally, we used a Python library called `facial_recognition` to detect and identify the faces of our team members. The `facial_recognition` library uses a similar framework to that of “You Only Look Once”, in that it only needs one photo of a person to be able to recognize them for future samples.

We will dive deeper into how we actually used these tools and software packages, and the challenges we ran into when using them in the sections below.

## Overview

The image below represents the entire data flow of our robot, end-to-end. Please refer to figure 6 in the event that you need clarification regarding the interaction of each component in our system.



**Fig. 6** Data flow of our robot

# Timeline

The course was broken down into the following milestones.

- Construction & Design
  - January 24 - January 30
- Controlled Motion & Obstacle Avoidance
  - January 30 - February 13
- Person Following
  - February 13 - February 27
- Person Identification
  - February 27 - March 27
- Bonus Challenges & Demo
  - March 27 - May 8

Each milestone was met with varying degrees of success and obstacles. In the following sections, we break down our approach to each milestone, the challenges we faced, and how we eventually worked our way around them.

## Construction & Design

### Goal

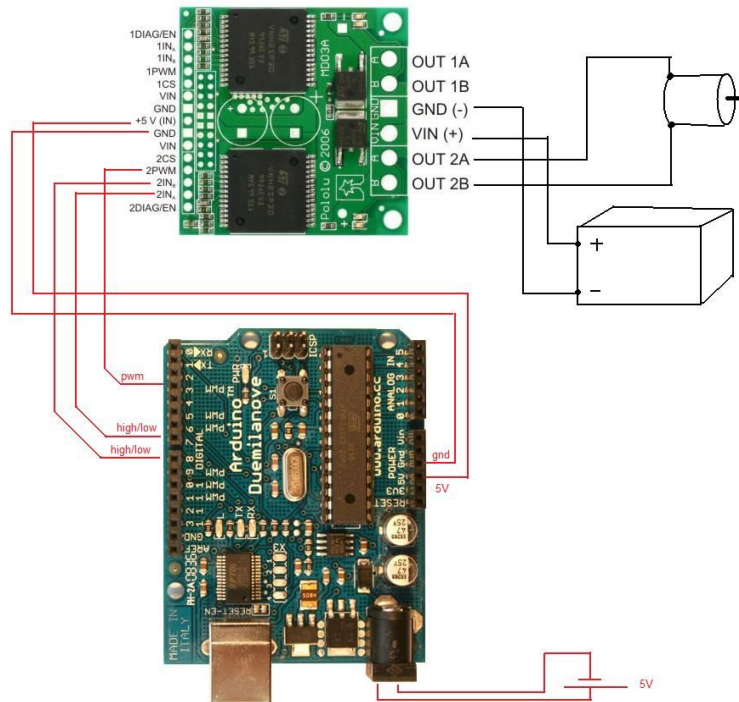
**Each team is given starter materials to begin wiring and mounting components to the robot.**

The primary objective of this milestone was to cleanly wire the starter materials (most notably, the battery, Ping))) sensors, motors, and motor controller) to the Arduino onboard the robot. In addition, a second objective of this milestone was to properly allocate space such that both the weight of the robot was properly distributed, and any future components, such as the NVIDIA Jetson TX2 and Amazon Echo Dot, could be seamlessly integrated without forcing us to readjust much.

### Implementation

We used the circuit diagram and starter code from this blog<sup>2</sup> as a jumping off point for this effort. We hoped that defining a sensible and color-coded wiring schema early on in the project would establish a precedent for properly adding any components in the future. Below is a graphical representation of the circuit described above.





**Fig. 7** Circuit Diagram for integrating the Arduino and Pololu Motor Controller

## Challenges

Each team was given a blank robot chassis with two motors and a front wheel attached upon arriving to the first lab session of the class. Our team was one of the last groups to receive primary wheels to attach to the motor axles due to being in the last lab section of the day. Unfortunately, by the time we found the wheels, we found that the large ones had all been taken. Though the smaller wheels were functional, we had to spend time remounting the motors such that the smaller wheels did not grind up against the robot base when moving. The smaller wheels also ended up adversely affecting the weight distribution of the robot as it was a lot closer to the ground. Towards the end of the semester, however, we located an extra pair of the larger wheels in the lab and pivoted our design to incorporate them.

Because of our strong desire to organize the wiring and intelligently preallocate the empty space on the robot, this portion of the timeline took us longer than expected. As with all long-term projects, thinking ahead is critical in order to protect oneself from unexpected consequences, but this mindset can become debilitating at a certain point. Perfection is rarely achievable, especially when working on a project in a new space to many of us in robotics. At some point, we had to accept that what we had put together was satisfactory enough to move on to the next stage in the project timeline.

# Controlled Motion & Obstacle Avoidance

## Goal

**Each team's robot must travel through the lab avoiding obstacles and people.**

The main idea of this goal is to prevent the robot from crashing into anything when following a given trajectory or person. As the robot follows a person, it may encounter any number of obstacles including chairs, walls, or even moving objects like other people and pets. In short, the desired motion should be quick enough to have the robot traverse smoothly but also avoid obstacles in an intelligent way.

## Implementation

The obstacle avoidance algorithm can be broken down into five distinct cases: "No Obstacle", "Obstacle Straight Ahead", "Obstacle to the Left", "Obstacle to the Right", and "Obstacle to the Left, Right, and Middle".

In the event of "No Obstacle" the robot is unconstrained and can move freely. The direction the robot takes when no obstacle is present depends on the input read from the NVIDIA Jetson. If the Jetson does not send a command the robot will rotate in place. When the Jetson powers on, it sends last found command on firebase to the Arduino. If the last command was to stop, for example, the robot will go into the STOP state where it does not move. If there is a command to follow someone it will receive the NA state. In the NA state the robot will spin in-place at a slow rate until it sees a face. When it sees a face, the Jetson provides the next steps to move towards the person if they match the command to follow. (See Person Identification)

When the middle ultrasonic sensor is blocked, the case of Obstacle Straight Ahead is triggered. After it is triggered the robot reverses and then attempts to turn in the direction of the command previous given command. When the left or right ultrasonic sensors is blocked, the cases of Obstacle Left and Obstacle Right are triggered, respectively. With both commands, we start by reversing, and then turning in the opposite direction. The turn is tuned such that the robot ends up parallel to the obstacle. This is done in a while loop to continue turning until the obstacle is no longer blocking the sensor. Finally, the last case is if there are obstacles on all sides (Left Right and Middle). In this scenario, the robot reverses for a few seconds, and then turns right. The reason we always reverse is to prevent the robot from turning into the obstacle and getting stuck in an infinite obstacle loop. As the robot turns the obstacle may get closer on one side as well as a new obstacle appearing on the other side. So when it is between two obstacles it can fall into a loop of trying to avoid either side. By reversing and turning we have a better chance of moving away from one of the obstacles.

## Challenges

We had trouble finding matching speeds for the motors. When set to the same pulse width modulation intensity, the wheels moved at different speeds which made the robot prone to spinning instead of moving forward, as desired. In addition, our robot had trouble avoiding obstacles when it was surrounded on all sides. We used front, left, and right facing ping sensors and if all three were blocked, the robot would oscillate between forward and backward motion. Sometimes it would take several tries for it to attempt to locate an unobstructed path and continue moving but other times it would never find an obstructed path and stop moving.

In hindsight, we could have avoided these problems had we started the Arduino guidance code from scratch. Instead, we used a past team's code and by the end we wished we wrote fresh code based on the other teams' rather than blindly using theirs for a large part of the semester. Integrating previous code can add some substantial technical debt. In order to gain the benefits of the navigation code from the previous team we had to trim many of the complex ideas they had built into it. This left us with barebones code that used old helper functions. Every robot is different, however, and each one has a different "personality", for lack of a better term. For example, as mentioned earlier, our motors did not match speeds. So the existing values from the previous team's code were useful to get started, but required some fiddling with to become useful. It was also important to look at the values for the ultrasonic sensors as they may not give the same readings.

## Person Following

### Goal

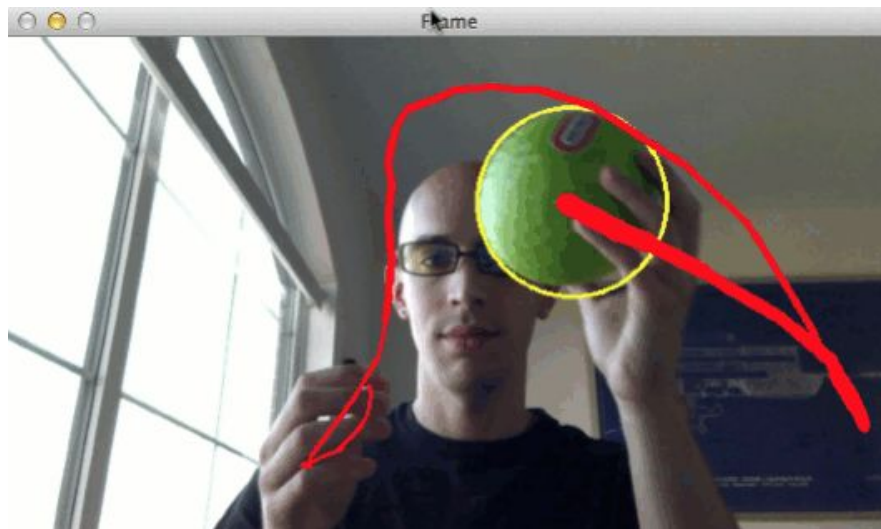
**Each team's robot must follow a person walking in the laboratory and hallway outside the lab.**

Though this milestone could imply person identification as well, most teams, including ourselves deciding to approach blob tracking as a means for person following first. For us, this meant following whoever is holding a tennis ball.

### Implementation

We implemented person following mainly using OpenCV library functions on the NVIDIA Jetson. This tutorial<sup>3</sup> from PyImageSearch taught us how to detect the presence of a bright green ball using computer vision techniques, and also track the ball as it moves around in frame by drawing its previous positions as a contrail. The photo below is taken directly from the blog and shows the programmer moving a ball in front of his camera. The yellow circle denotes the

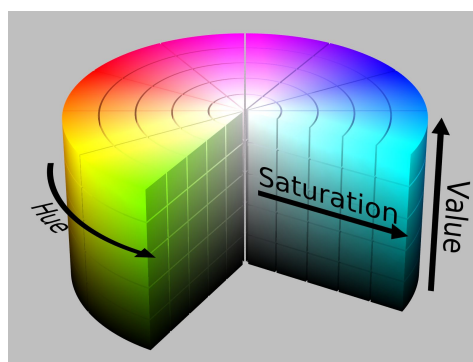
minimum enclosing circle around the ball at any time, and the red contrail shows the previous 64 positions of the ball at any time. Thicker contrail lines denote more recent positions.



**Fig. 8** Blob tracking using OpenCV's `inRange` function

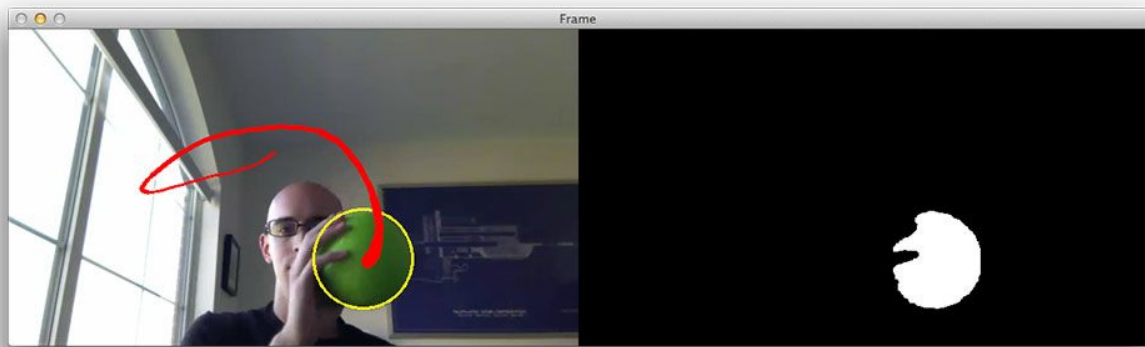
Luckily, the author documents the code very well, which made it very easy to customize this example to our needs and purposes.

For historical reasons, OpenCV defaults to the BGR color space. BGR is equivalent to RGB, however the blue and red channels are physically flipped. In order to better measure image intensity, we convert each frame to HSV. We do this because color spaces like HSV separate the image intensity from the image color, which is particularly useful in this application. Rather than measuring color, HSV represents the hue, saturation, and value of an image.



**Fig. 9** HSV color cylinder

Using OpenCV's `inRange` function, we can localize the green ball in frame. We first supply the lower HSV color boundaries for the color green, followed by the upper HSV boundaries. The output of `cv2.inRange` is a binary mask.



**Fig. 10** This binary mask only allows objects in the green HSV range we defined to pass through

We can then identify the (x,y) centroid coordinates of the largest contour in any particular frame and send the information to the Arduino. We partition the frame into five equal sections, and depending on the x coordinate location within the frame, we decide upon one of five states.

Fast Left	Slow Left	Fast Forward	Slow Right	Fast Right
-----------	-----------	--------------	------------	------------

## Challenges

The main challenge we ran into at this stage of the project was deciding whether to transfer the NVIDIA Jetson from the development board to the much more portable carrier board. The primary motivation behind transferring to the carrier board is that it would dramatically reduce the physical surface area the Jetson would require. At this point in the project, we did not have much real estate left on the robot. Doing this, however, would require us to reflash the device. This means that we would have had to start with a fresh install of Ubuntu, and we would have lost much of our progress. Despite following version control best practices by using GitHub<sup>4</sup>, we would have still had to reinstall many packages on the Jetson and risked creating compatibility issues that were not worth the trouble of pursuing this track any further. In addition, we heard that the transfer process was taking some teams as long as 15 hours to complete. In the end, we asked Jay, the lab engineer, if he could build us an elevated platform for which we could place the development board onto.

## Person Identification

### Goal

**Each team's robot must demonstrate the ability to identify a specific person and respond to (voice) commands from that person.**

The goal of this section was to implement person identification by means of video and or audio (voice) recognition. Currently, the Amazon Echo Dot does not support voice command recognition, however we were able to implement facial recognition using a Python library called `facial_recognition`<sup>5</sup>.

## Implementation

The `facial_recognition` library only needs one photo of each subject in order to classify them in the future. Due to the poor lighting conditions and the instability of our camera unit, however, we took several photos in order to have as many encodings as possible. Displaying the facial recognition results looks exactly like this.

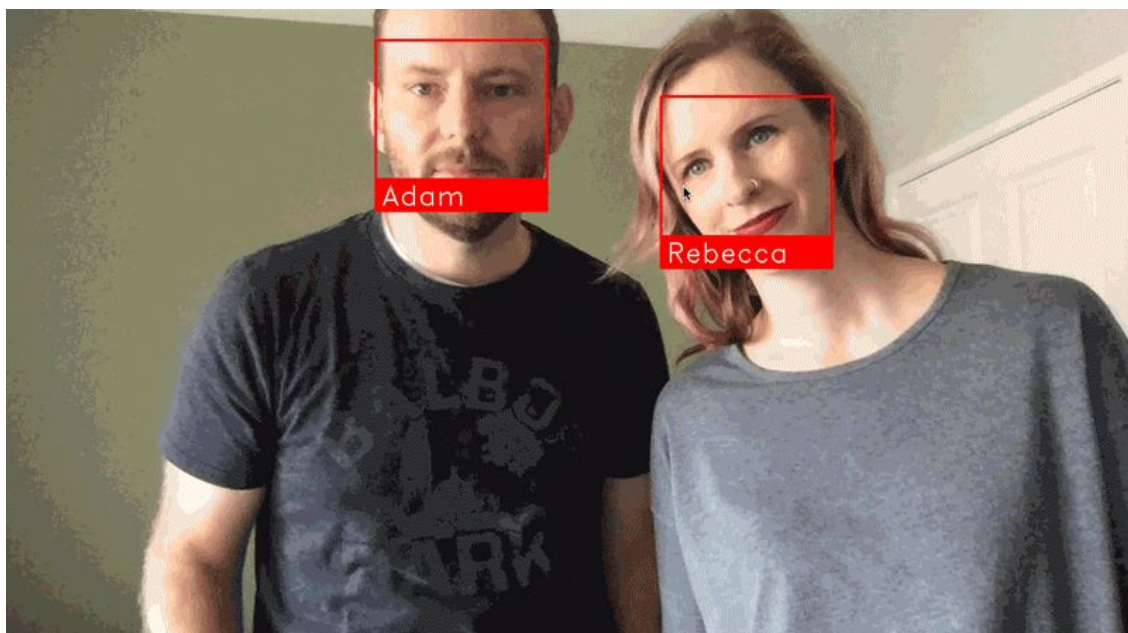


Fig. 11 Facial recognition sample from a single video frame

## Challenges

For our team, the `facial_recognition` library had trouble identifying faces when we were using the fisheye camera lens. We suspect that the models the classifiers use were trained on rectilinear photos, and thus had trouble identifying classical facial features in the distorted frame. We decided to swap the lens with that of a traditional web camera to solve this issue, but this presented some new issues. With the camera position so low, and no fisheye lens to see upwards, we were unable to follow or identify our subject. To fix this, we added support “mast” and wired the camera up so that it was closer to eye-level.

In addition, as with many facial recognition and detection classifiers, the performance tends to suffer with darker skinned subjects. Only of our teammates was lighter skinned, so demonstrations involving this integral feature were only consistent when he was the primary subject.

We also had some trouble deciding how to follow a subject after they had turned around. It is impractical, and one could even argue dangerous, for someone to have to be constantly walking backwards in order for the robot to follow them. In order to get around this, we hacked a quick fix that identified the color shirt of the user. That way, if no face was in frame, the robot would fall back to following the largest contour of the same color, assuming that it was the same person.

## Bonus Features & Demo

Each team demoed their robots' ability to meet each of the criteria stated above and showcased their bonus features in the final lab session of the semester. Check out the YouTube video below for a brief demonstration of the person following and obstacle avoidance capabilities of our robot.



Fig. 12 <https://www.youtube.com/watch?v=58ssK7U3kx4>

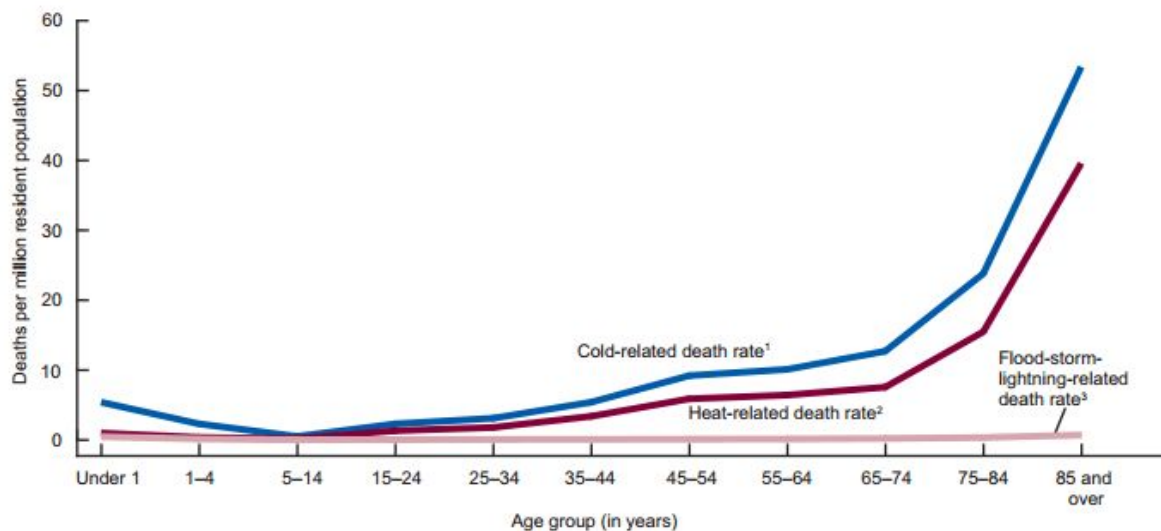
The photo shown above is a screen grab from the YouTube video demo. Click the link in the caption to see the entire demonstration.

## Ambient Air Temperature Sensor

According to a report<sup>6</sup> conducted by the Center for Disease Control and Prevention, temperature related fatality rates are much higher for older adults. "Among adults, heat-related and cold-related death rates increased with age, particularly for those aged 75 and over," the report says. The study's conclusion is consistent with the findings in other studies which show



older adults, especially those who are socially isolated, sick and/or poor, tend to be more susceptible to extreme temperatures.



**Fig. 13** Mortality rates for weather-related deaths, by age: United States, 2006–2010 (CDC)

As you can see from the graph above, the number of weather-related deaths jumps up quite dramatically for older adults. With this in mind, we decided to incorporate temperature sensing into our robot by using the NTC temperature sensor from the Arduino starter kit. We calibrated the sensor to the average human body temperature (98.6°F) and set hot and cold thresholds of 90°F and 60°F, respectively. If the temperature hit either extreme, then an IFTTT trigger is activated to alert the user's primary caregiver via text message. Below is a screenshot of what one of these messages was structured to look like.

The temperature is too HOT at  
102.55 degrees

**Fig. 14** Sample text message warning the primary caregiver that the temperature is too hot

Using pyfttt<sup>7</sup>, a Python package for interfacing with the IFTTT webhooks channel, incorporating this feature simply required a few extra lines to our Python navigation script. Reading in the temperature values off the Arduino pin was also trivial to do. In the end, we were very pleased with how simple this feature was to implement, yet how relevant the issues it tackled are to present day eldercare.

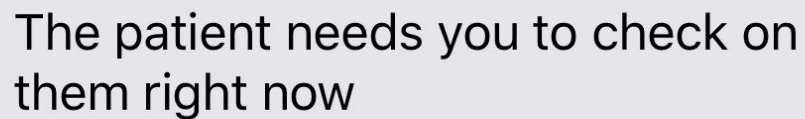
“Call for Help!”

Another standard feature in most assistive robotics implementations is that of personal emergency response and home medical alerts. This feature was notably commercially



popularized by Life Alert's suite of products. In our implementation, the user can invoke this feature by asking the Amazon Echo Dot to call for help. Doing this results in a phone call and text message being sent to the user's primary caregiver. In the event that the phone call is not answered, a voicemail is left notifying the caregiver to check on the patient right away. A screenshot of the text that would be sent is shown below.

Tue, May 8, 2:38 PM



The patient needs you to check on them right now

Fig. 15 This text message is sent if the user calls for help

## Conclusions

We learned that designing a robot of this stature end-to-end really does require a team with strong communication. Having differing schedules made it challenging to meet or discuss outside of class, but defining expectations early, and redefining them as different hurdles presented themselves, was the key to building a successful robot.

As far as hardware goes, we believe that it would have been worth exploring different ways we could incorporate the Xbox Kinect. Out of the box, the Xbox Kinect provides the ability to detect both people and voices. In addition, motion and pose detection for the Xbox Kinect is also very powerful. Another hardware component we would have liked to have experimented with would have been a thermal camera. Depending on how strong the device is, some researchers<sup>8</sup> have shown that thermal cameras are capable of extracting heart rate, or lack thereof, from a simple video stream.

Though there are many things we would have done differently had we been able to do this project from the beginning, we believe that the class would not have been as fulfilling without all the challenges and obstacles we faced along the way. We would like to thank Professor Blankenship for his encouragement and for organizing the class in such a way where we could feel free to experiment and design without feeling too constrained by a rubric or guidelines. We would also like to thank the lab engineer Jay Renner, for building us numerous components we used for the robot and helping us with general software and hardware issues. Finally, we would like thank our Graduate Teaching Assistant, Helene Nguewou-Hyousse, for her guidance and for opening the lab outside of class hours so we could continue to work on the robot.

# Resources

1. <https://ifr.org/ifr-press-releases/news/31-million-robots-helping-in-households-worldwide-by-2019>
2. <https://zilinzen.wordpress.com/2009/12/19/pololu-motor-controller-testing-and-troubleshooting/>
3. <https://www.pyimagesearch.com/2015/09/14/ball-tracking-with-opencv/>
4. <https://github.com/omikader/ENEE408I>
5. [https://pythonhosted.org/face\\_recognition/readme.html](https://pythonhosted.org/face_recognition/readme.html)
6. [https://www.washingtonpost.com/news/capital-weather-gang/wp/2014/08/04/cold-kills-more-than-heat-cdc-says-but-researchers-caution-it-depends/?noredirect=on&utm\\_term=.26bf022931d2](https://www.washingtonpost.com/news/capital-weather-gang/wp/2014/08/04/cold-kills-more-than-heat-cdc-says-but-researchers-caution-it-depends/?noredirect=on&utm_term=.26bf022931d2)
7. <https://pypi.org/project/pyfttt/>
8. <https://pdfs.semanticscholar.org/04aa/4343267cdc469abecaeb5cf9e14619001876.pdf>