ECE 445 Final Report

Team 36 - Nikhil Mehta, Irfan Suzali, Abishek Venkit
ECE445 Final Report - Spring 2020
TA: Shuai Tang

Abstract

As online delivery services become more and more common, package theft has become a prevalent and growing problem. Our project focuses on this central problem area, and uses a computer vision based method to detect, monitor, and disarm the system. We based our design on an original design from Spring 2018 Team 9, who chose to solve this problem using a weight-sensing doormat to detect if a package is moved without authorization. Instead of this approach, we decided to use an entirely vision-based approach, using an object detection model to track packages and a face recognition model to disarm the system when the user's face is detected. This modification makes our design more functional, secure, and durable than the original.

Table of Contents

1 Motivation

1.1 Problem Statement

Ordering packages online is extremely commonplace these days, with over 87 billion packages shipped worldwide in 2018 [1]. However, the added convenience can also have drawbacks. One of the risks of ordering a package online is theft. When packages are dropped off outside of an unprotected residence, it is easy for a thief to steal it without being caught. 36% of Americans have experienced package theft, and the average cost to replace a stolen package is $109. This is costly and inconvenient for the customer and supplier [2]. The package delivery industry is incredibly large and growing at a fast pace, which tells us that this problem is significant, and an affordable, reliable, and secure solution would help a lot of people protect their packages from theft.

1.2 Solution

Our solution will implement computer vision to identify when packages are delivered and when they are picked up. This system will use a camera that is pointed at the location of delivery. The package will be delivered within a general area in the field of view of the camera. Once the package is delivered, the computer vision algorithm will detect its presence, and track it to ensure it stays in full view of the camera. The system will also notify the user via text using WiFi when it detects that a package has arrived. If anyone tries to move it, or if the package becomes obscured from the camera, an alarm will sound, the user is notified, and a short video recording will be taken of the offender. Furthermore, if anyone moves the actual camera system or tries to cover the camera, the alarm will sound as well. In order to disarm the alarm, the actual package recipient must show their face to the camera. This will disable the alarm from triggering for a short period, allowing the user to pick up their package. In order for this system to work at night, there will also be a motion sensored spotlight to illuminate the entryway.

This implementation will use a computer vision object identification and facial recognition algorithm running on the Raspberry Pi. This model will be trained on various standard package shapes and sizes, to ensure that the system can recognize any package with the appropriate marking. *The Raspberry Pi will only be used for the CV algorithm and storage system, and will transfer data to a separate microcontroller*. This microcontroller will communicate with the Raspberry Pi, and the alarm, WiFi, and spotlight systems. The alarm system will require speaker/audio circuitry, which includes a digital-to-analog converter, amplifier, and speaker. The notification system will require a WiFi module and external cellular device to push text messages to the user. The spotlight system will require a PIR motion sensor, ambient light sensor, and lighting circuitry. The entire device will need a power supply and voltage regulation. In order to disarm the alarm and retrieve your package, the system will utilize a face recognition model to turn off the alarm system when it recognizes the user's face. This will make the system seamless and easy to use, while still maintaining the security factors that are most important.

Package security is a large industry with several existing solutions on the market. However, despite the growing adoption rate of home security systems, only 17% of homes have some type of security system installed [3]. There are several reasons that people don't use home security systems. This includes the fact that they cost too much, they are difficult to use, and possibly infringe on privacy [4]. Related to the concern over high cost, many

home security systems are fully featured, and serve many purposes beyond just package security. This is great if you want all of these extra features and can afford it, but there are few options if you only want a way to secure your packages. Our system will also be easy to use, with a simple notification system and an easy way to disarm your system when you want to retrieve your package. The hardware will all be in one package, which will be easy to install and low cost.

Another alternative package delivery solution aimed at reducing theft is choosing to ship your package to a package locker, like a PO box, or the more recent Amazon Locker [5]. This allows you to secure your package in a remote facility until you unlock it and pick it up. This solution is less than ideal as well, because it forces you to travel to a separate destination everytime you want to pick up a package. A more modern and techy solution is using smart locks to allow delivery inside your home, like the Amazon Key [6]. This solution eliminates the risk of package theft after delivery, but introduces a new concern of letting someone into your home to deliver your package. These smart lock systems are also expensive and require installation into your door.

We have defined our target market for this product as anyone who orders packages regularly and wants an affordable way to monitor their package without investing in an expensive, fully-featured home security system. This is a large market, because such a small percentage of homes have home security systems. A major reason for this is the high price of home security systems, complicated usage, and fear of infringement on privacy. Considering our target market, the purchase conditions for a security system, and the many alternatives, our product excels when compared to the competition in a few key ways. Firstly, our system is focused on solving one specific common problem, allowing us to handle it in a complete and effective way. Other systems on the market have much broader goals, and thus become bloated and spread thin. Our system can focus more deeply on solving the problem of package theft. The next key strength comes as a byproduct of the first strength. By focusing on this single problem, we can make our system much more affordable than other competing products. Many consumers only want a system that will protect their packages from theft, and our product can provide that without any extra features or added cost. Finally, our product is easy to use and install, with simple features like face recognition to disarm and text notifications for package delivery and suspicious activity. Our product can seamlessly fit into our customers existing lifestyle and habits without a large learning curve or a complicated interface. These key benefits differentiate our product from the competition and provide value to our customers.

This project is an alternate solution to a problem proposed by Spring 2018 Team 9 [7]. Their original solution detected packages using a doormat with weight sensors, and detected potential thieves through the use of a PIR sensor. Their system would give a warning if someone approached the package and would take a picture of the approaching person. If the package is lifted from the mat, the weight sensors will detect it and trigger an alarm. If an alarm is triggered, the system will notify the user by sending a notification containing the captured photo to an Android app that is connected to this system over WiFi. The system is disabled using the app, or an RFID tag.

Our solution is different in a few key ways. First, our system uses only one camera to detect and track a package. Team 9's solution contains a camera, but also includes a weight sensor mat. We believe the mat is redundant and unnecessary if the goal is to detect and track a package. This can be done using the camera alone through computer vision. Eliminating the doormat also allows the package to be placed in a wider area in your entryway.

The old design will only work if the package is placed upon the doormat, but our solution will function as long as the package is in view of the camera mounted above the door. The field of view of the camera will be much wider than the doormat, making our solution more flexible and robust. Next, our design will be entirely housed in one unit that is mounted above your door, instead of having separate units on the floor and above the door. Having the whole system in one enclosed body reduces the risk of tampering with the system. Another difference is that if motion is detected near the package, our system will record a short video clip instead of just a picture. A video clip will give more context and information surrounding events in which an alarm is triggered. Our system will be disabled using facial recognition instead of an RFID tag as well, which further reduces cost because using facial recognition does not require any additional hardware, and will simply require software computation. Finally, our system does not need a dedicated app, and will instead send the user a text when packages are delivered, or if activity is detected. This will make the system more flexible for different users with different types of smartphones.

The differences listed all work to make our package security system a better, more effective solution to the pervasive problem of package theft. By restructuring the system into a single package that utilizes computer vision to monitor packages when delivered solves several problem points of the old system. The most important improvements of our system relate to usability and simplicity.

One deficiency of the original project design is the weight sensor mat. There are obvious limitations of this feature when it comes to usability and security, both of which are important factors for consumers when purchasing security systems. If a package is not placed within the bounds of the mat, it will not be detected, making the system essentially useless. In our system, the package can be placed anywhere within the field of the view of the camera, which is a much wider range. Secondly, the weight sensor can trigger false alarms if some foreign object like a leaf or branch happened to land on the mat. This is not an issue for our improved design, because our computer vision system is constantly looking for the package itself rather than a specific weight.

Another aspect that we have improved is the method of disarming the alarm when you want to retrieve your package. In the original design, the user had to use an Android app or an RFID tag to disarm the alarm and pick up their package. This method works perfectly fine, but adds an extra layer of complexity to the process of picking up a package. Using the app requires you to take out your phone and navigate to the app, and using an RFID tag requires you to carry another physical item with you at all times. Our solution to this problem is to use the existing camera and incorporate a face recognition model to disarm the system. This does not require the user to access their phone or carry any additional hardware. It will function seamlessly, and greatly improves the simplicity and utility of the product.

Overall, the features discussed are all extremely important to consumers when deciding to purchase a security system. Focusing on usability and simplicity make our product more enjoyable and painless to use, while maintaining robust security features that protect our customers' packages. Our improvements on the original product make it a better value and experience for the user.

1.3 High-Level Requirements

In order for this system to be successful, it would need to reduce the risk of package theft from households, and if packages are stolen anyways, the system would need to provide a video record of the thief.

- The video recording system must capture thefts reliably (90% +/- 5% theft detection rate) and must disarm reliably using facial recognition (80% +/- 5% true positive facial recognition rate, 90 % +/- 5% true negative rate)
- The notification system must notify the user reliably (90% +/- 5% WiFi transmission reliability given a valid outgoing message)
- The sensor and lighting system must illuminate the scene in the dark reliably, using the PIR and ambient light sensors (90% +/- 5% accurate in identifying and responding to darkness below a given threshold when movement is detected)

1.4 Visual Aid

Figure 1. Visual representation of how the system can be implemented at a typical home

The physical design consists of multiple components that will be mounted inside a robust enclosure near the user's entryway. On the front side of the enclosure, the camera, PIR sensor, ambient light sensor, and spotlight will stick out. On the back side of the enclosure, there will be a power cable that should be wired through the wall to be connected inside the user's home.

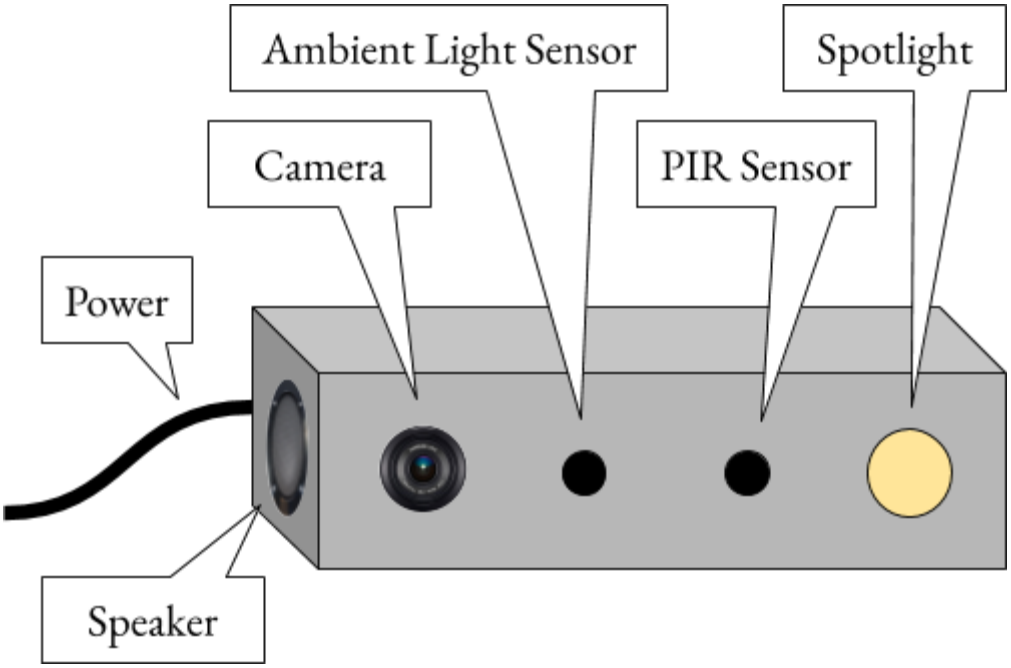


Figure 2. Physical Design

As shown in Figure 1, this unit will be mounted above the front door. This will keep all electronics secure and out of reach. In case someone tries to cut power to the system, or tamper with the electronics inside, the camera would capture it, since it faces outwards and covers the entire entryway. Additionally, since the unit is mounted above the door, it will also be safe from weather-related stress, including rain and wind. The enclosure will prevent water from affecting the internal components. These features will make the system robust and durable for long-term usage.

1.5 Block Diagram

Our block diagram displays the distinct different modules in our design. The perception module is composed of the Raspberry Pi and the Raspberry Pi Camera, and will be connected to the control and power supply modules. The control module contains a PIC32 microcontroller. The motion sensor module will contain a PIR motion sensor, ambient light sensor, and a spotlight, and will interface with the control module to send and receive data. The alarm module will contain a DAC to convert a PWM signal to analog, and a medium sized speaker which will output the alarm signal. The WiFi module will contain a dedicated WiFi chip that will interface with the PIC32 over UART, and a LED status light to indicate transmission. Finally, the power supply module will allow us to convert outlet power (120V) into 5V and 3.3V for usage by the Raspberry Pi, PIC32, LED, speaker, and other components.

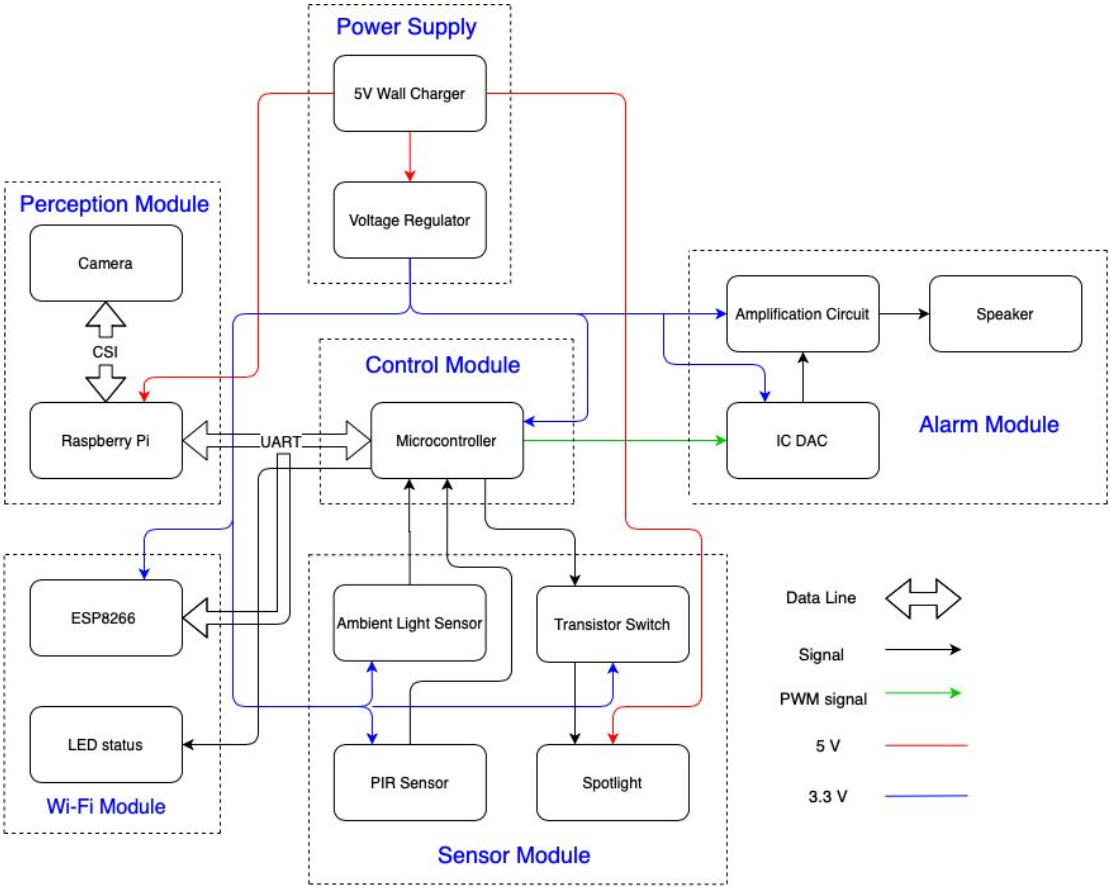


Figure 3. Block Diagram

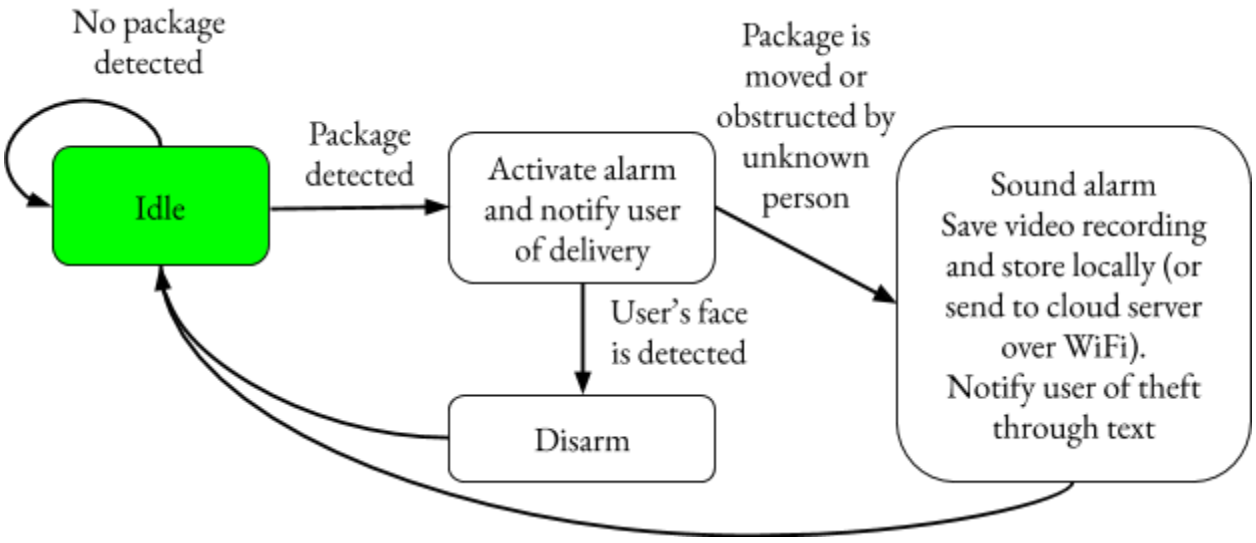


Figure 4. Flow diagram of perception alarm system

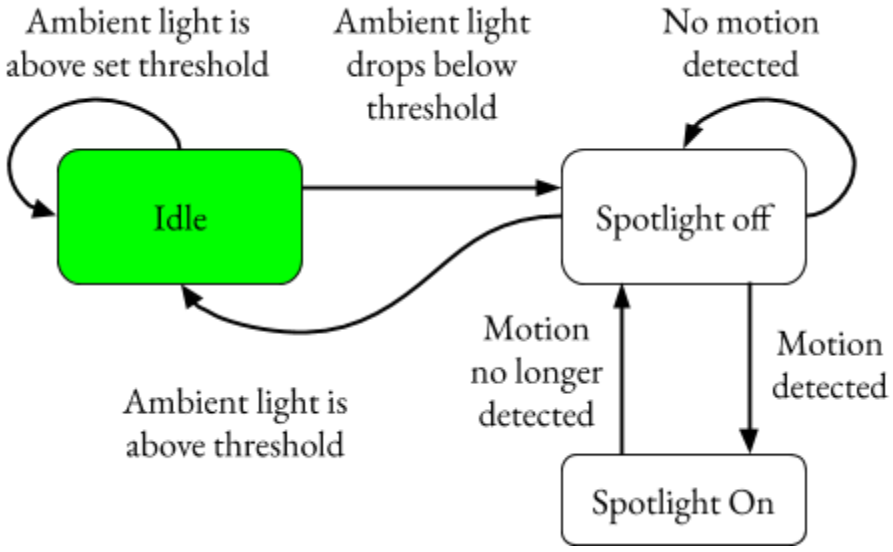


Figure 5. Flow diagram of ambient light sensor spotlight system

2 Implementation

For this new project, we were unable to implement any of the hardware components due to lack of test equipment and other restrictions. However, we were able to develop a test for one of our main software subcomponents. Specifically, we were able to develop and test the face recognition part of our system, and simulate its function with test videos taken from a smartphone. In our actual product, video will be captured by a Raspberry Pi Camera, but since we were unable to implement any hardware, we tested on smartphone video and ran the computation on a laptop instead.

We also started implementation of the PIC32 dispatcher code, completing the UART communication interface and writing pseudocode for the analog input sensor interactions. We could not test this code properly without the PIC32, but the logic described in Figures 4 and 5 is complete. This code appears in the appendix.

2.1 Face Recognition Model Background

In order to test the face recognition software subsystem, we had to implement the face recognition model pipeline. This includes a face detection model to identify where the face or faces can be found within an image, a face recognition training function, and the face recognition function itself. This process is broken down in the flow diagram below.

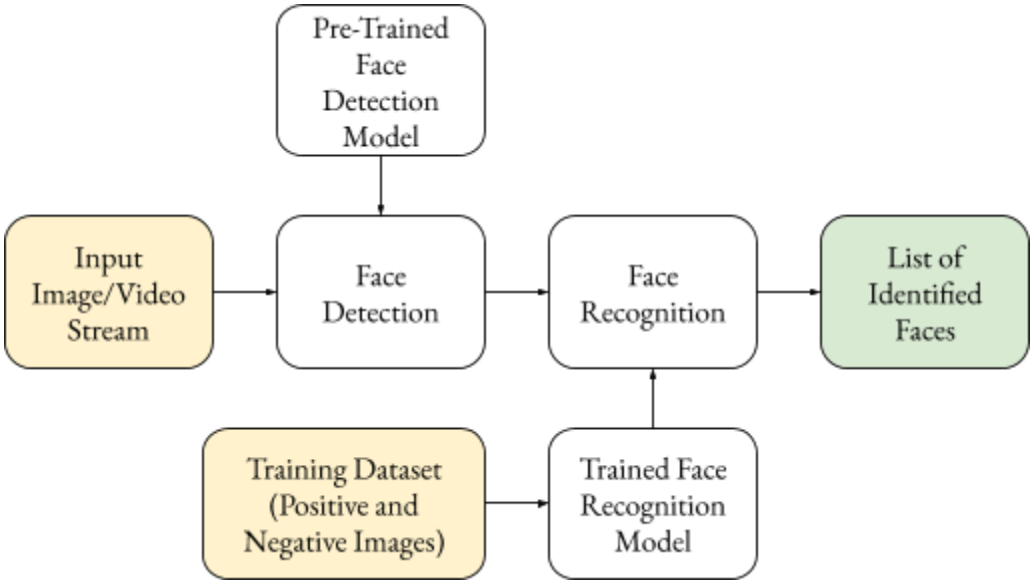


Figure 6. Face Recognition Model Flow Diagram

The first computational step in this process is the face detection function. This is needed to extract where the face or faces are in an input image. This step does not actually extract any information about the faces in an image. It simply determines whether faces exist in the image at all, and outputs their location. Because of this, the face detection model is almost universal, and we decided to use a pre-trained face detection deep neural net

from the OpenCV library [8]. This model is based on the Single Shot Detector framework with a ResNet base network. Face detection is a fast and accurate task, and does not require any modification to fit this specific purpose.

The next computational step is to properly identify each detected face. For the purposes of our product, this step will identify whether the face in an image is the user, or someone unknown. This information is then used to either disarm the system, or sound the alarm. Unlike the face detection step, the face recognition step is specialized to each user, and must be trained on a set of images of that specific user, as well as a random set of non-authorized faces. The model will then be trained based on this information. This trained model will now be able to identify faces, completing the pipeline. To accomplish the face recognition task, we used an open source project called OpenFace [9]. This model quantifies facial images, and can accurately classify based on training data. Before using the model to classify faces, you must train it in a one-time process that generates a file of embeddings that classify faces [10]. Once this file is generated, the model can be used to identify faces on an input image or video stream.

We decided to use a model based entirely on OpenCV because it is most similar to the implementation we would have applied to our Raspberry Pi [11]. Although we were not able to implement facial recognition on the Raspberry Pi, our simulation and results are representative of how the model generally would have worked when embedded on the Pi.

2.2 Face Recognition Model Implementation and Results

After learning about how the face recognition model works, we implemented it using Python. The first step of our implementation was training the model on a dataset of faces. We decided to train it on images of Nikhil. Using this dataset, we generated a file of embeddings, and referenced it for each future test. For the actual device, this process would happen when you first set it up, and will be automated. After mounting the device above your door, it would walk you through the process of looking at the camera, allowing the camera to capture several images of you. The system would then train itself based on this set of images. For this simulation, we did this process manually.

We then tested the face recognition model on a sample image. Figure 7 shows the results of this test, accurately identifying Nikhil in the image, and labelling the other face as unknown.

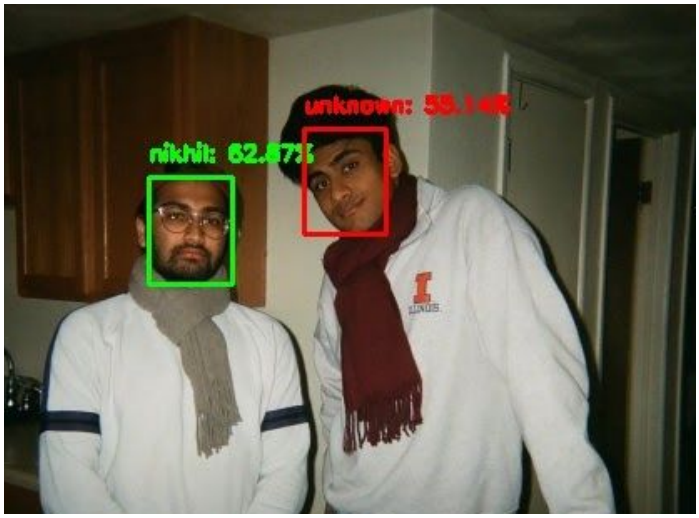


Figure 7. Face recognition test on a sample image

We then moved on to testing the model on a video that would be similar to actual footage from our device. We recorded a video of Nikhil walking up to the door of his home, as if he was arriving home and wanted to disarm the alarm system. The screenshots below show the progression of this video, from walking up to the door, to the point where the face comes into the frame and is positively recognized.

Figure 8. Screenshots of face recognition test on a sample video

As shown in these examples, the face recognition system is accurate and can locate and identify faces based on its training. In addition to this, it can function in real-time and identify faces quickly without much lag. We tested the model on a live webcam feed and maintained a smooth video stream, even while identifying faces. The next step for this would have been to port the model over to the Raspberry Pi, and continue testing with the Raspberry Pi Camera module. We would then implement an object detection model to detect packages as well,

completing the computer vision subsystem of our device. We were unable to do this, but the success of our face recognition tests are promising, and imply that the computer vision models on the Raspberry Pi would have worked as well. An object detection model would be functionally similar to the face recognition model, with a slightly different pipeline. The model would need to be trained on many images of packages, and would then detect packages in the input image using that information.

2.3 Circuit Schematics

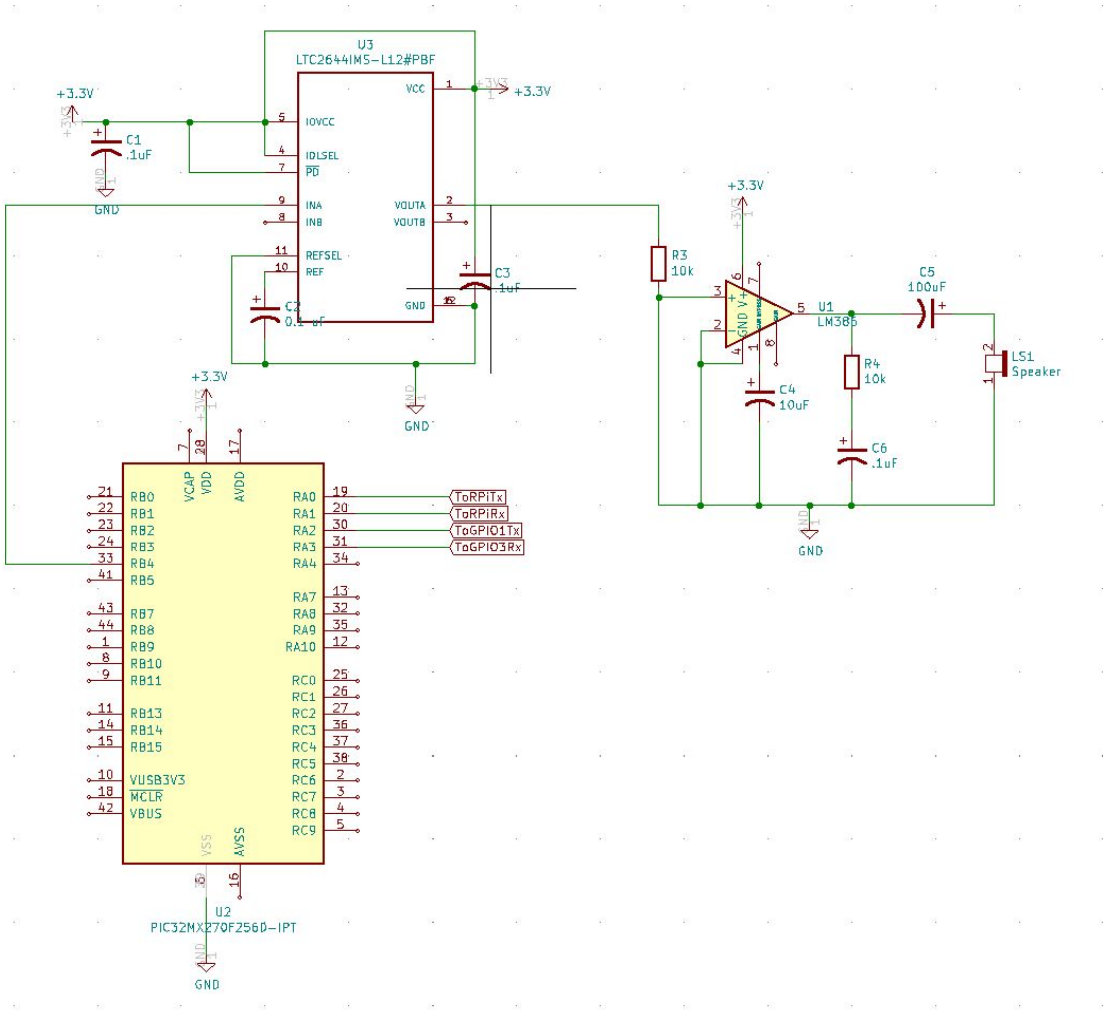


Figure 9. Circuit Schematic for Control and Alarm Module

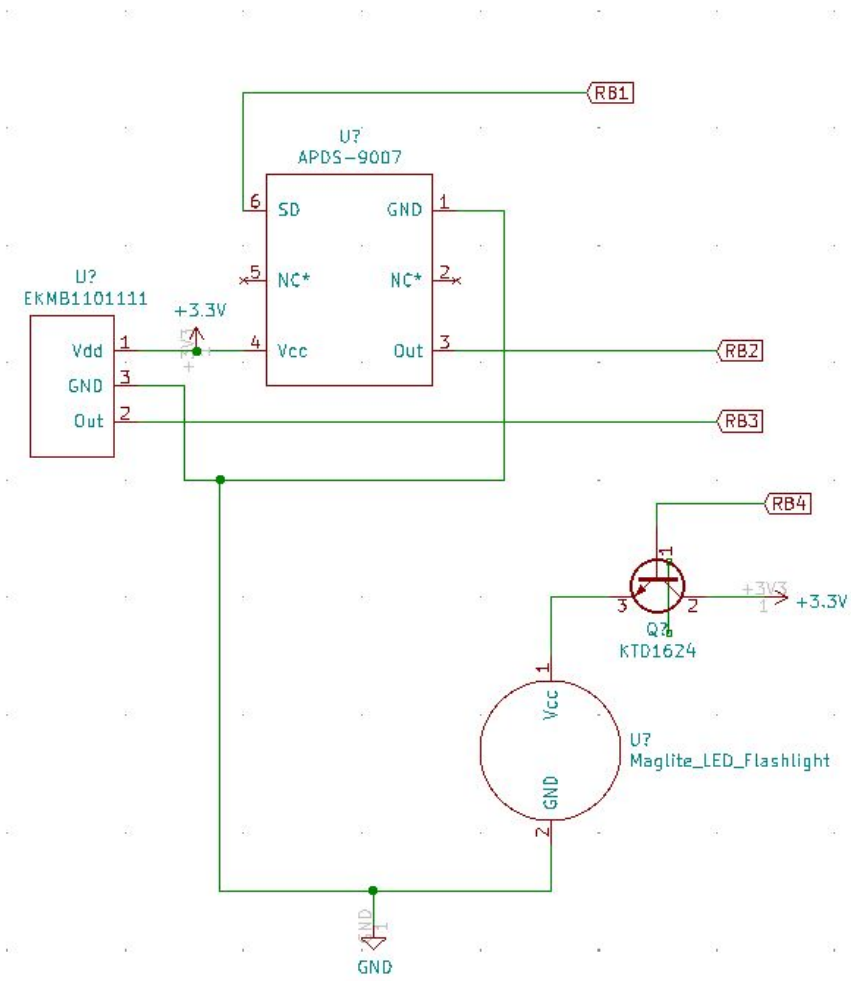


Figure 10. Circuit Schematic for Sensor Module

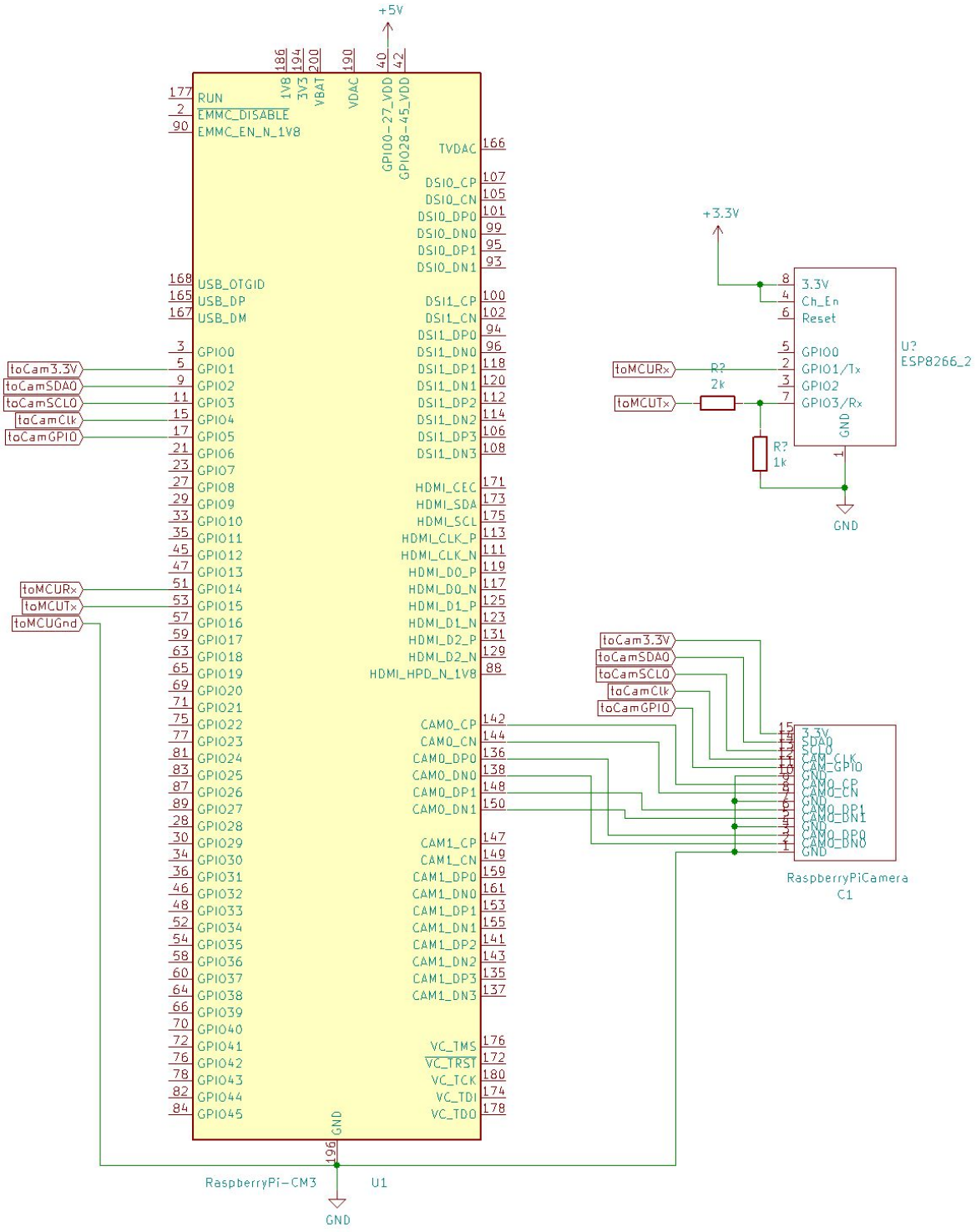


Figure 11. Circuit Schematic for Perception and Wi-fi Module

2.4 Tolerance Analysis

The success of the project is determined by the reliability of our perception and sensor module in tracking a package and detecting the presence of other objects within a scene with varying lighting conditions. To successfully achieve this, we require our system to be robust enough to tolerate a range of operating conditions. In any case, we would want our system to be able to track a package and its status at all times. Failure to recognize an object will ultimately render our system useless.

2.4.1 PIR Sensor

To analyze the constraints of our system, we will first take a look at our PIR sensor. The sensor is required to pick up movement by any human entity within a certain range. The following are the specifications of PIR sensor:

PIR sensor Field of View (FOV):

- Maximum detection range: 5 m (16.404 ft)
- Horizontal FOV = 94°
- Vertical FOV = 82°

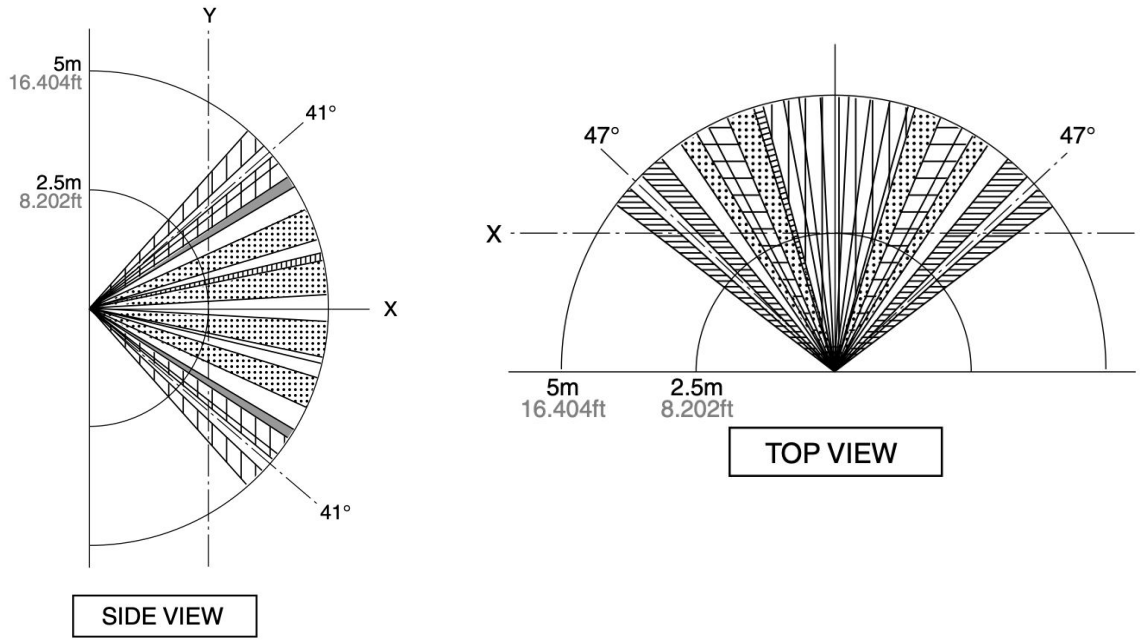
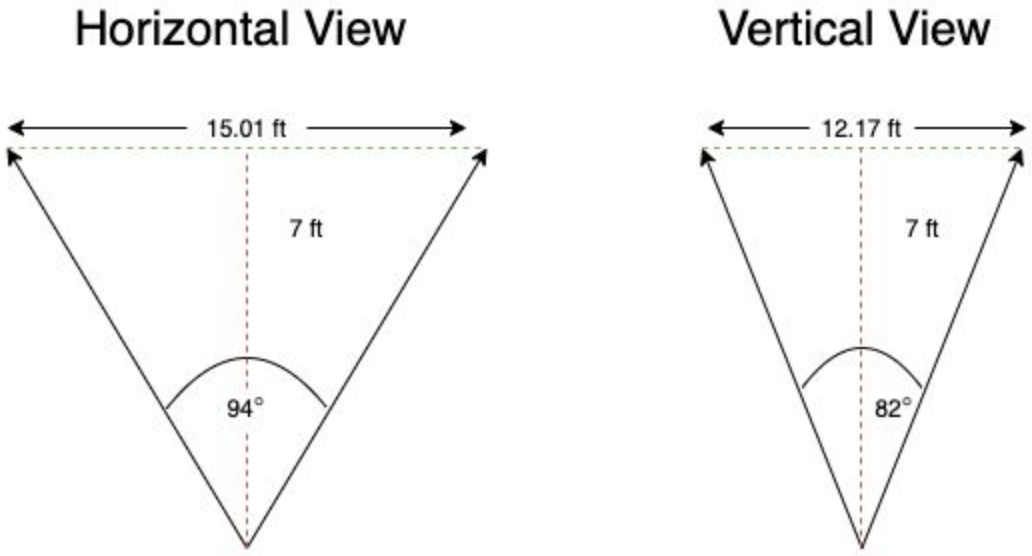


Figure 12. Detection performance of PIR sensor

Given that the maximum distance of detection for the PIR sensor is 16.404 feet, it is well within our desired operating height for our system. Theoretically, any distance less than 16.404 feet from the desired package

placement area would work. However, since we will be using the PIR sensor to also switch a spotlight on during low light conditions, we would want to extend the detection range further from the package placement area.

As shown in Figure 1, the system will be placed above the door frame, with an operating height of approximately 7 feet. This would be our desired operating range. Considering this height, we can calculate the effective range of the sensor. Below is a visual representation of the FOV, when the distance from the PIR sensor to the package is 7 feet. The calculations (1) and (2) are used to determine the field of view window.



$$tan\,(94/2\circ) * 7\,ft * 2 = 15.01\,ft \quad (1)$$

$$tan\,(82/2\circ) * 7\,ft * 2 = 12.17\,ft \quad (2)$$

Based on the calculations, at an operating height of 7 feet, we are able to achieve a wide range of detection with 12.17 ft in vertical height and 15.01 ft in horizontal width. This gives our system the ability to detect human presence beyond the scene in which a package may be initially placed. We are aware that with a wide FOV, we may have an increased number of false positives when detecting object motion. We would prefer our system to produce false positives when detecting object motion to add robustness to our system. We have agreed that the effective FOV of the PIR sensor is optimal for our use case.

2.4.2 Light Ambient Sensor

Another component we need to monitor tolerance levels for is the ambient light sensor. The ambient light sensor will amplify the photodiode output signal and convert it into a logarithmic current output. The sensor is able to provide a photocurrent response to a wide dynamic range of 3 lux to 70k lux. The sensitivity of the sensor allows us to work within a large range of lighting levels. Our challenge is to determine a threshold lux in which we would like to operate at. The logarithmic current output is advantageous because we will be determining a lux threshold at lower brightness levels. These smaller changes will be easier to be detected as it provides a good relative resolution over the entiere ambient light brightness range as shown in Figure 9.

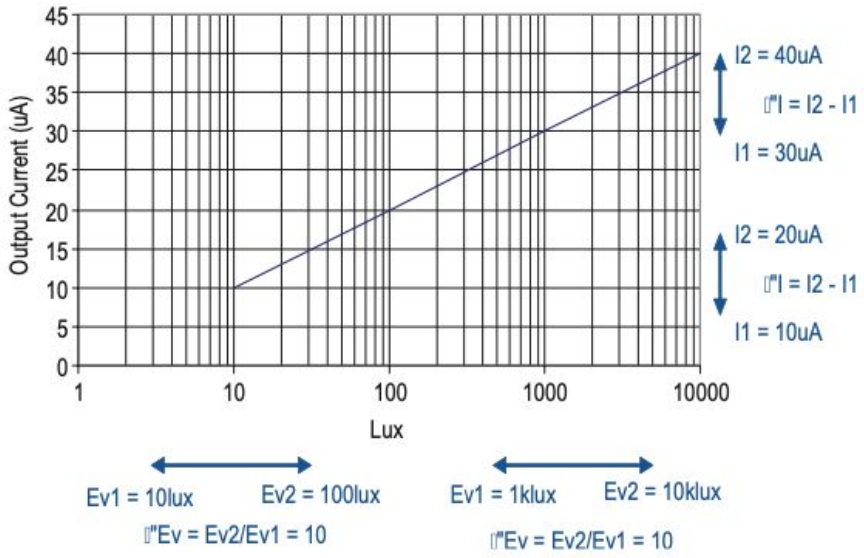


Figure 13. Feature of Logarithmic Output

Determining the ambient threshold brightness level will require further testing with the perception module to ensure optimum results.

2.4.3 Software

It is also important to note that there will be some acceptable tolerance ranges on the software aspect of the computer vision algorithms. This has to do with image resolution and distance from the subject. Based on the given metrics of our camera, as well as the OpenCV algorithms we will be using, our image resolution will be in an acceptable range as long as our sensors satisfy the above constraints.

2.5 Bill of Materials

Table 1. Breakdown of parts required with expenses

Module	Parts	Cost Per Unit	Reason
Control	PIC32MX270F256D Microcontroller	$4.09	Microcontroller for managing peripherals and I/O
Power Supply	5V 10A Wall Charger	$19.88	Provide 5V power
	Voltage Regulator	$2.77	5V to 3.3V converter
Perception	Raspberry Pi	$35.00	Perform computer vision component
	Raspberry Pi Camera Module v2	$25.25	Image/Video input
Speaker	PCB 8 Ohm Speaker	$2.50	Audio output
	LM386 Op Amp	$1.05	Amplification circuit
	IC DAC	$7.21	Digital to analog converter
Wi-fi	ESP8266 Wi-fi module	$6.95	Enable wifi capabilities for microcontroller
Sensor	PIR Sensor	$2.29	Motion detection
	APDS-9007-020 Light Sensor	$2.07	Ambient light detection
	Maglite LED Flashlight	$24.98	Spotlight for illumination
	Grand Total	**$134.04**	

3 Conclusions

3.1 Implementation Summary

Due to the circumstances and restrictions this semester, we were unable to implement much of our design. Specifically, we were unable to implement hardware components, our PCB, and run computation on our Raspberry Pi. However, we did simulate part of our software subsystem, and found promising results. Our tests with face recognition were successful, and we gathered enough positive data that we are hopeful that our actual hardware product would have worked as well. We tested the face recognition model on images, pre-recorded video, and on a real-time webcam feed, and all resulted in smooth, consistent, and accurate face tracking. This is an important part of our design and contributes to our first high level requirement of accurate video tracking and disarm capabilities. If this function was not accurate or efficient, the usability of our system would have suffered greatly.

In addition to our software simulations, we also developed a detailed tolerance analysis for our PIR sensor component, and our ambient light sensor. These components are integral parts of our motion-sensing spotlight subsystem and have somewhat specific tolerances to each of them. The PIR motion sensor has a certain field of view that is limited by its hardware, and we had to make sure it was wide enough to sense motion in the whole entryway area. The ambient light sensor also has tolerances on the amount of ambient light it can detect, measured in lux. It is required for this sensor to be able to detect when ambient light drops below a certain threshold, indicating that it is nighttime. Because we could not test this ourselves, we developed the tolerance analysis, and specific requirements for each sensor that we would test for.

Finally, we created circuit schematics for our whole system. The schematic diagrams show each component of our system, including the power supply, perception subsystem, motion-sensing spotlight subsystem, WiFi subsystem, and speaker subsystem. These schematics would be the basis of our PCB design.

3.2 Unknowns, Uncertainties, Testing Needed

All hardware testing and integration could not be completed because of a lack of access to the lab and our parts. If we were to have access to the lab and our parts, our first course of action would be to test and verify our sensors and voltage regulator. We would ensure they perform as intended straight out of the box and that there are no defective components. After this, we would need to complete our software implementation and start integrating software and hardware components. The programming of the PIC32 would require a dedicated device that can access the pins, and after programming and testing is complete, the PIC32 would be soldered onto the PCB. Because this process is new for all of us, it would require some trial and error. Once the hardware components are tested and integrated with software, we would test modules and eventually move on to complete device tests. We would build our way up to these larger tests so that we can debug individual components easily before connecting everything together.

Ultimately, given an entire semester, we would be able to test, refine, and hopefully create a working prototype. Still, even without access to any hardware or lab equipment, we were able to make significant progress on the software components of our design.

3.3 Ethics and Safety

We have an obligation to our profession to uphold the highest level of ethical and professional conduct. We stand to follow and commit ourselves to the guidelines stated by the IEEE Code of Ethics. Safety of the user is of utmost importance especially since there are significant hardware components situated on the body of the user. There is a potential danger of hardware components short circuiting and overheating after prolonged usage that could cause harm to the user. We intend to design our product with these risks in mind in accordance to the IEEE Code of Ethics #1 - "To hold public safety first and to disclose factors of our project that might endanger the public" [12]. Mitigating these risks are our main priority. To avoid overheating, we will ensure that all components operate in low power mode when not in use. Additionally, hardware components will be spaced out accordingly within our designed enclosure so that electrical contact is avoided risking a short circuit and ultimately device malfunction. The enclosure should uphold OSHA provision standards 1910.303(b)(7)(i) stating "Unless identified for use in the operating environment, no conductors or equipment shall be located in damp or wet locations; where exposed to gases, fumes, vapors, liquids, or other agents that have a deteriorating effect on the conductors or equipment; or where exposed to excessive temperatures."[13] to provide protection against any case of exposure to liquids that could cause a short circuit. This is critical as the device will most likely be placed outdoors where it will be exposed to the elements. The sensor components have humidity conditions which if not upheld may produce performance issues.

We acknowledge that there is a certain degree of error that can arise from object identification. The core of the project depends on users being able to trust our system to identify and label an object in its scene with a high level of accuracy. To adhere to the IEEE Code of Ethics #3 - "To be honest and realistic in stating claims or estimates based on the available data."[12], it is our duty to be honest of the estimates provided from the available data provided to us. To uphold this, we will ensure that our system has a reliable output and is able to verify and identify different packages. This will be done by training our object recognition model extensively with a variety of packages of all shapes and sizes. In addition, we will also be using a facial recognition algorithm to authorize user access to the system. This will require a significant degree of accuracy as to ensure proper access is given to authorized users. The standard of accuracy in which we will be aiming for will be high to ensure the overall security of our anti-theft system.

Finally, our product would not be possible without the advances in computer vision algorithms developed by pioneers before us. In accordance to the IEEE Code of Ethics #7 - "To seek, accept, and offer honest criticism of technical work, to acknowledge and correct errors, and to credit properly the contributions of others." [12], we would like to formally acknowledge and give due credit to those who have contributed to the open source software, OpenCV.

3.4 Future Improvements

If given ample time to work on our second project, we would not only be able to complete our initial implementation, but we could expand on the software and hardware to make the product more practical and polished. Initially, we would complete the original project following these steps:

1. Design PCB for PIC32 and sensors
2. Complete sensor verification and testing
3. Complete PIC32 dispatch code (interacting with sensors and Raspberry Pi)
4. Integrate Raspberry Pi, sensors, and PIC32
5. Design hardware enclose and mounting system

After completing our initial design, we would thoroughly test the product in various scenarios, evaluating the successes and shortcoming of our device. In order to improve our device, we would try to implement the following 3 additional features:

1. Convert Raspberry Pi Code to run on a dedicated GPU using CUDA
2. Develop a mobile app to deal with user notifications
3. Use a cloud hosting service to back up video recordings in real time

We selected these features because each one improves the practicality and usefulness of the product, but were too involved for us to implement in a single semester. Using GPUs instead of the Raspberry Pi would majorly speed up the computer vision algorithm runtime. While we were not able to test the actual runtime of our code on a Raspberry Pi, we expected it to run significantly slower than on a normal computer. This would cause a delay between thefts and alarms, or face recognition and disarm, a significant issue that could affect the accuracy and practicality of our product. A dedicated GPU will run even faster than a standard laptop, although it would take significant expertise to implement correctly. The use of a mobile app would increase the usability and versatility of user interactions with the product. Instead of receiving simple SMS text messages from the device, the user can get notifications that include images, they can access the history of alerts in the past, and they can change settings to vary what events will trigger notifications. They could even change the alarm tone, or vary the light threshold. There are numerous possibilities. One issue to note with this improvement is that it may increase cost such that our product is no longer a cheap alternative to a full-fledge security system. If the app were implemented, we would need to do a cost-benefit analysis to ensure it fits the product and the target market. Finally, a cloud hosting service is a secure and reliable way to store theft data. If a thief were to somehow destroy our original security system, the recorded data would be lost. Backing up data on the cloud in real time resolves this issue, and lets the user view recordings online with ease. Each one of these improvements refines our prototype and moves it closer to a market-ready product.

Progress Made on First Project

In addition to the work done on our second project, we made some progress on our first project (EyeCU - Assistive Eyewear with Computer Vision) as well. Similar to the face recognition simulation we did for our second project, we also completed a simulation for the OCR (Optical Character Recognition) model required for our first project. The same restrictions applied to the development on our first project, in that we were not able to implement or test any hardware. We instead simulated the software components only on our laptops.

The OCR model is explained in the flow diagram below. Unlike the face recognition model, there is no need to provide a training dataset since recognizing english text only requires recognizing the relatively small set of characters.

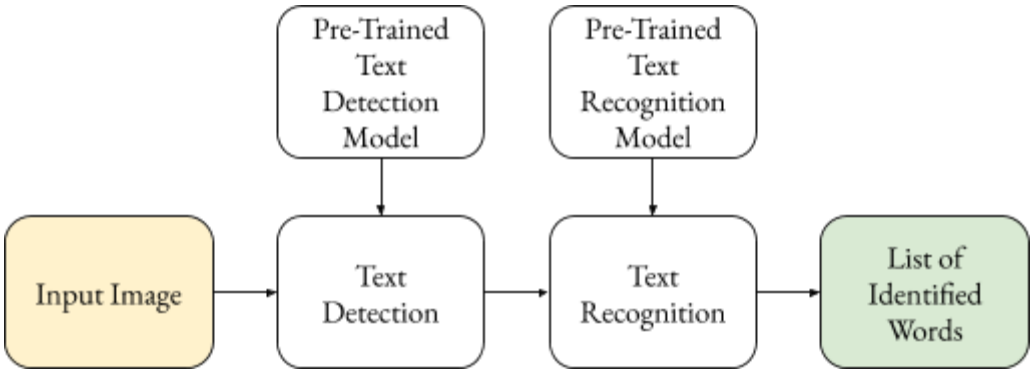


Figure 14. Flow diagram for OCR

The text detection task was completed using the pre-trained EAST text detection model [14]. This model detects and locates individual segmented words within an image. Each detected word is then quantified using the OpenCV Tesseract model [15].

After implementing the model, we tested on various images. We found that the OCR model worked well on standard, simple fonts, but struggled on more unique, specialized fonts.

Figure 15. Results of OCR Testing

References

[1] K. Buchholz and F. Richter, “Infographic: 87 Billion Parcels Were Shipped in 2018,” *Statista Infographics*, 08-Nov-2019. [Online]. Available: https://www.statista.com/chart/10922/parcel-shipping-volume-and-parcel-spend-in-selected-countries/. [Accessed: 03-Apr-2020].

[2] Kaza, “6 Shocking Stats about Package Theft,” *Smiota*, 17-Feb-2020. [Online]. Available: https://smiota.com/resources/6-shocking-stats-package-theft/. [Accessed: 03-Apr-2020].

[3] “Burglary Statistics: The Hard Numbers,” *National Council For Home Safety and Security*, 19-Dec-2019. [Online]. Available: https://www.alarms.org/burglary-statistics/. [Accessed: 03-Apr-2020].

[4] “5 Reasons Why Homeowners Don't Have Home Security Systems,” *ADT Home Security and Alarm Systems | ProtectYourHome.com*. [Online]. Available: https://www.protectyourhome.com/blog/articles/2014/march/5-reasons-why-homeowners-dont-have-home-security-systems. [Accessed: 03-Apr-2020].

[5] H. Blodget, “Here's A Picture Of Amazon Locker, The New Delivery Box Amazon Is Using To Take Over The World,” *Business Insider*, 24-Aug-2012. [Online]. Available: https://www.businessinsider.com/amazon-locker-2012-8. [Accessed: 03-Apr-2020].

[6] T. Haselton, “Amazon Key changes how packages are delivered - just beware of your dog,” *CNBC*, 16-Nov-2017. [Online]. Available: https://www.cnbc.com/2017/11/16/amazon-key-in-home-delivery-review.html. [Accessed: 03-Apr-2020].

[7] J. Bianco, J. Graft, and J. Simonaitis, “Package Anti-Theft System,” Feb. 2018. [Online]. Available: https://courses.engr.illinois.edu/ece445/getfile.asp?id=12482. [Accessed: 03-April-2020].

[8] Rosebrock, A., 2020. Face Detection With Opencv And Deep Learning - Pyimagesearch. [online] PyImageSearch. Available at: <https://www.pyimagesearch.com/2018/02/26/face-detection-with-opencv-and-deep-learning/> [Accessed 8 May 2020].

[9] OpenFace. [Online]. Available: https://cmusatyalab.github.io/openface/. [Accessed: 08-May-2020].

[10] Rosebrock, A., 2020. Opencv Face Recognition - Pyimagesearch. [online] PyImageSearch. Available at: <https://www.pyimagesearch.com/2018/09/24/opencv-face-recognition/> [Accessed 8 May 2020].

[11] A. Rosebrock, “Raspberry Pi Face Recognition,” *PyImageSearch*, 25-Jun-2018. [Online]. Available: https://www.pyimagesearch.com/2018/06/25/raspberry-pi-face-recognition/. [Accessed: 17-Apr-2020].

[12] “IEEE Code of Ethics,” *IEEE*. [Online]. Available: https://www.ieee.org/about/corporate/governance/p7-8.html. [Accessed: 03-Apr-2020].

[13] Osha.gov. (2020). *1910.303 - General.* | *Occupational Safety and Health Administration*. [online] Available at: https://www.osha.gov/laws-regs/regulations/standardnumber/1910/1910.303 [Accessed 28 Feb. 2020].

[14] "OCRTesseract Class Reference," OpenCV. [Online]. Available: https://docs.opencv.org/3.4/d7/ddc/classcv_1_1text_1_1OCRTesseract.html. [Accessed: 28-Feb-2020].

[15] Rosebrock, A., 2020. Opencv OCR And Text Recognition With Tesseract - Pyimagesearch. [online] PyImageSearch. Available at: <https://www.pyimagesearch.com/2018/09/17/opencv-ocr-and-text-recognition-with-tesseract/> [Accessed 8 May 2020].

Appendix

Facial Recognition Code (Project 2)

```
# USAGE
# python recognize_video.py --detector face_detection_model \
#       --embedding-model openface_nn4.small2.v1.t7 \
#       --recognizer output/recognizer.pickle \
#       --le output/le.pickle

# import the necessary packages
from imutils.video import VideoStream
from imutils.video import FileVideoStream
from imutils.video import FPS
from threading import Thread
import numpy as np
import argparse
import imutils
import pickle
import time
import cv2
import os

# construct the argument parser and parse the arguments
ap = argparse.ArgumentParser()
ap.add_argument("-d", "--detector", required=True,
       help="path to OpenCV's deep learning face detector")
ap.add_argument("-m", "--embedding-model", required=True,
       help="path to OpenCV's deep learning face embedding model")
ap.add_argument("-r", "--recognizer", required=True,
       help="path to model trained to recognize faces")
ap.add_argument("-l", "--le", required=True,
       help="path to label encoder")
ap.add_argument("-c", "--confidence", type=float, default=0.5,
       help="minimum probability to filter weak detections")
ap.add_argument("-i", "--input", help="input video filepath")
ap.add_argument("-o", "--output", help="output path")
args = vars(ap.parse_args())

# load our serialized face detector from disk
print("[INFO] loading face detector...")
protoPath = os.path.sep.join([args["detector"], "deploy.prototxt"])
modelPath = os.path.sep.join([args["detector"],
       "res10_300x300_ssd_iter_140000.caffemodel"])
detector = cv2.dnn.readNetFromCaffe(protoPath, modelPath)

# load our serialized face embedding model from disk
```

```
print("[INFO] loading face recognizer...")
embedder = cv2.dnn.readNetFromTorch(args["embedding_model"])

# load the actual face recognition model along with the label encoder
recognizer = pickle.loads(open(args["recognizer"], "rb").read())
le = pickle.loads(open(args["le"], "rb").read())

# initialize the video stream, then allow the camera sensor to warm up
if args["input"]:
	print("[INFO] reading input video...")
	vs = FileVideoStream(args["input"]).start()
	time.sleep(2.0)
else:
	print("[INFO] starting video stream...")
	vs = VideoStream(src=0).start()
	time.sleep(2.0)

check_output = False
if args["output"]:
	fourcc = cv2.VideoWriter_fourcc(*'MJPG')
	videowriter = None
	check_output = True

# start the FPS throughput estimator
fps = FPS().start()

# loop over frames from the video file stream
while True:
	# grab the frame from the threaded video stream
	grabbed, frame = vs.read()
	if not grabbed:
		break

	# resize the frame to have a width of 600 pixels (while
	# maintaining the aspect ratio), and then grab the image
	# dimensions
	frame = imutils.resize(frame, width=600)
	(h, w) = frame.shape[:2]

	if (check_output) and (not videowriter):
		videowriter = cv2.VideoWriter(args["output"],fourcc,20,(w,h),True)

	# construct a blob from the image
	imageBlob = cv2.dnn.blobFromImage(
		cv2.resize(frame, (300, 300)), 1.0, (300, 300),
		(104.0, 177.0, 123.0), swapRB=False, crop=False)

	# apply OpenCV's deep learning-based face detector to localize
	# faces in the input image
	detector.setInput(imageBlob)
```

```
detections = detector.forward()

# loop over the detections
for i in range(0, detections.shape[2]):
	# extract the confidence (i.e., probability) associated with
	# the prediction
	confidence = detections[0, 0, i, 2]

	# filter out weak detections
	if confidence > args["confidence"]:
		# compute the (x, y)-coordinates of the bounding box for
		# the face
		box = detections[0, 0, i, 3:7] * np.array([w, h, w, h])
		(startX, startY, endX, endY) = box.astype("int")

		# extract the face ROI
		face = frame[startY:endY, startX:endX]
		(fH, fW) = face.shape[:2]

		# ensure the face width and height are sufficiently large
		if fW < 20 or fH < 20:
			continue

		# construct a blob for the face ROI, then pass the blob
		# through our face embedding model to obtain the 128-d
		# quantification of the face
		faceBlob = cv2.dnn.blobFromImage(face, 1.0 / 255,
			(96, 96), (0, 0, 0), swapRB=True, crop=False)
		embedder.setInput(faceBlob)
		vec = embedder.forward()

		# perform classification to recognize the face
		preds = recognizer.predict_proba(vec)[0]
		j = np.argmax(preds)
		proba = preds[j]
		name = le.classes_[j]

		if name == "nikhil":
			color = (0, 255, 0)
		else:
			color = (0, 0, 255)

		# draw the bounding box of the face along with the
		# associated probability
		text = "{}: {:.2f}%".format(name, proba * 100)
		y = startY - 10 if startY - 10 > 10 else startY + 10
		cv2.rectangle(frame, (startX, startY), (endX, endY),
			color, 2)
		cv2.putText(frame, text, (startX, y),
			cv2.FONT_HERSHEY_SIMPLEX, 0.45, color, 2)
```

```
        # update the FPS counter
        fps.update()

        if check_output:
                videowriter.write(frame)
        else:
                # show the output frame
                cv2.imshow("Frame", frame)
                key = cv2.waitKey(1) & 0xFF

                # if the `q` key was pressed, break from the loop
                if key == ord("q"):
                        break

# stop the timer and display FPS information
fps.stop()
print("[INFO] elapsed time: {:.2f}".format(fps.elapsed()))
print("[INFO] approx. FPS: {:.2f}".format(fps.fps()))

# do a bit of cleanup
cv2.destroyAllWindows()
vs.stop()
if check_output:
        videowriter.release()
```

PIC32 Dispatcher Code (Project 2)

```
//pragma statements and UART configs taken from below link
//https://people.ece.cornell.edu/land/courses/ece4760/PIC32/PLIB_examples/plib_examples/uart
/uart_basic/source/main.c
// Section: Includes
// *****************************************************************************
// *****************************************************************************
#include <GenericTypeDefs.h>
#include <plib.h>
// *****************************************************************************
// *****************************************************************************
// Section: Configuration bits
// *****************************************************************************
// *****************************************************************************
#pragma config FPLLODIV = DIV_1, FPLLMUL = MUL_20, FPLLIDIV = DIV_2, FWDTEN = OFF, FCKSM =
CSECME, FPBDIV = DIV_1
#pragma config OSCIOFNC = ON, POSCMOD = XT, FSOSCEN = ON, FNOSC = PRIPLL
#pragma config CP = OFF, BWP = OFF, PWP = OFF

// *****************************************************************************
// *****************************************************************************
// Section: System Macros
```

```
// ****************************************************************************
// ****************************************************************************
#define GetSystemClock()                    (80000000ul)
#define GetPeripheralClock()          (GetSystemClock()/(1 << OSCCONbits.PBDIV))
#define GetInstructionClock()         (GetSystemClock())

UINT32 GetDataBuffer(char *buffer, UINT32 max_size, UINT32 UART_NUM);
void SendDataBuffer(const char *buffer, UINT32 size, UINT32 UART_NUM);
UINT32 activate_alarm = 0;

#include "PIC32MX270_STD.h"
#include <string.h>

int main(void) {

    //UART1 configs
    UARTConfigure(UART1, UART_ENABLE_PINS_TX_RX_ONLY);
    UARTSetFifoMode(UART1, UART_INTERRUPT_ON_TX_NOT_FULL | UART_INTERRUPT_ON_RX_NOT_EMPTY);
    UARTSetLineControl(UART1, UART_DATA_SIZE_8_BITS | UART_PARITY_NONE | UART_STOP_BITS_1);
    UARTSetDataRate(UART1, GetPeripheralClock(), 57600);
    UARTEnable(UART1, UART_ENABLE_FLAGS(UART_PERIPHERAL | UART_RX | UART_TX));
    //UART2 configs
    UARTConfigure(UART2, UART_ENABLE_PINS_TX_RX_ONLY);
    UARTSetFifoMode(UART2, UART_INTERRUPT_ON_TX_NOT_FULL | UART_INTERRUPT_ON_RX_NOT_EMPTY);
    UARTSetLineControl(UART2, UART_DATA_SIZE_8_BITS | UART_PARITY_NONE | UART_STOP_BITS_1);
    UARTSetDataRate(UART2, GetPeripheralClock(), 57600);
    UARTEnable(UART2, UART_ENABLE_FLAGS(UART_PERIPHERAL | UART_RX | UART_TX));

    while(1) {
        //read Rpi input data over UART1
        //directive buffer ("THEFT", "DELIVERY", "USER")
        char Rpi_data[] = itoa(GetDataBuffer(buf, 1024, 1));
        char str_theft[] = "THEFT";
        char str_delivery[] = "DELIVERY";
        char str_user[] = "USER";
        if (strcmp(Rpi_data, str_delivery) == 0) {
            //send notification to WIFI module over UART2
            SendDataBuffer("Delivery Occurred", sizeof("Delivery Occurred"), 2);
            activate_alarm = 1;
        }
        if (strcmp(Rpi_data, str_theft) == 0 && activate_alarm) {
            //send notification to WIFI module over UART2
            SendDataBuffer("Theft Occurred", sizeof("Theft Occurred"), 2);
            //function not implemented
            soundAlarm();
        }
        if (strcmp(Rpi_data, str_user) == 0) {
            activate_alarm = 0;
        }
```

```
        //pseudo code for analog inputs (functions not implemented)
        UINT32 light_level_low = readAnalog1() > light_threshold ? 0 : 1;
        UINT32 movement_detected = readAnalog2() > movement_threshold ? 1 : 0;
        if (light_level_low && movement_detected){
            activateSpotlight(); //turns on spotlight for 10 seconds before polling motion
        }

    }
}

//functions modified from below link
//https://people.ece.cornell.edu/land/courses/ece4760/PIC32/PLIB_examples/plib_examples/uart
/uart_basic/source/main.c
//send data over UART
void SendDataBuffer(const char *buffer, UINT32 size, UINT32 UART_NUM){
    if (UART_NUM == 1) {
        while(size){
            while(!UARTTransmitterIsReady(UART1));
            UARTSendDataByte(UART1, *buffer);
            buffer++;
            size--;
        }
        while(!UARTTransmissionHasCompleted(UART1));
    }
    if (UART_NUM == 2) {
        while(size){
            while(!UARTTransmitterIsReady(UART2));
            UARTSendDataByte(UART2, *buffer);
            buffer++;
            size--;
        }
        while(!UARTTransmissionHasCompleted(UART2));
    }
}
//receive data from UART
UINT32 GetDataBuffer(char *buffer, UINT32 max_size, UINT32 UART_NUM){
    UINT32 num_char;
    num_char = 0;
    while(num_char < max_size){
        UINT8 character;
        if (UART_NUM == 1){
            while(!UARTReceivedDataIsAvailable(UART1));
            character = UARTGetDataByte(UART1);
        }
        if (UART_NUM == 2){
            while(!UARTReceivedDataIsAvailable(UART2));
            character = UARTGetDataByte(UART2);
        }
        if(character == '\r')
            break;
```

```
        *buffer = character;
        buffer++;
        num_char++;
    }
    return num_char;
}
```

OCR Code (Project 1)

```
# USAGE
# python recognize_video.py --detector face_detection_model \
#       --embedding-model openface_nn4.small2.v1.t7 \
#       --recognizer output/recognizer.pickle \
#       --le output/le.pickle

# import the necessary packages
from imutils.video import VideoStream
from imutils.video import FileVideoStream
from imutils.video import FPS
from threading import Thread
import numpy as np
import argparse
import imutils
import pickle
import time
import cv2
import os

# construct the argument parser and parse the arguments
ap = argparse.ArgumentParser()
ap.add_argument("-d", "--detector", required=True,
        help="path to OpenCV's deep learning face detector")
ap.add_argument("-m", "--embedding-model", required=True,
        help="path to OpenCV's deep learning face embedding model")
ap.add_argument("-r", "--recognizer", required=True,
        help="path to model trained to recognize faces")
ap.add_argument("-l", "--le", required=True,
        help="path to label encoder")
ap.add_argument("-c", "--confidence", type=float, default=0.5,
        help="minimum probability to filter weak detections")
ap.add_argument("-i", "--input", help="input video filepath")
ap.add_argument("-o", "--output", help="output path")
args = vars(ap.parse_args())

# load our serialized face detector from disk
print("[INFO] loading face detector...")
protoPath = os.path.sep.join([args["detector"], "deploy.prototxt"])
```

```
modelPath = os.path.sep.join([args["detector"],
	"res10_300x300_ssd_iter_140000.caffemodel"])
detector = cv2.dnn.readNetFromCaffe(protoPath, modelPath)

# load our serialized face embedding model from disk
print("[INFO] loading face recognizer...")
embedder = cv2.dnn.readNetFromTorch(args["embedding_model"])

# load the actual face recognition model along with the label encoder
recognizer = pickle.loads(open(args["recognizer"], "rb").read())
le = pickle.loads(open(args["le"], "rb").read())

# initialize the video stream, then allow the camera sensor to warm up
if args["input"]:
	print("[INFO] reading input video...")
	vs = FileVideoStream(args["input"]).start()
	time.sleep(2.0)
else:
	print("[INFO] starting video stream...")
	vs = VideoStream(src=0).start()
	time.sleep(2.0)

check_output = False
if args["output"]:
	fourcc = cv2.VideoWriter_fourcc(*'MJPG')
	videowriter = None
	check_output = True

# start the FPS throughput estimator
fps = FPS().start()

# loop over frames from the video file stream
while True:
	# grab the frame from the threaded video stream
	grabbed, frame = vs.read()
	if not grabbed:
		break

	# resize the frame to have a width of 600 pixels (while
	# maintaining the aspect ratio), and then grab the image
	# dimensions
	frame = imutils.resize(frame, width=600)
	(h, w) = frame.shape[:2]

	if (check_output) and (not videowriter):
		videowriter = cv2.VideoWriter(args["output"],fourcc,20,(w,h),True)

	# construct a blob from the image
	imageBlob = cv2.dnn.blobFromImage(
		cv2.resize(frame, (300, 300)), 1.0, (300, 300),
```

```
        (104.0, 177.0, 123.0), swapRB=False, crop=False)

    # apply OpenCV's deep learning-based face detector to localize
    # faces in the input image
    detector.setInput(imageBlob)
    detections = detector.forward()

    # loop over the detections
    for i in range(0, detections.shape[2]):
        # extract the confidence (i.e., probability) associated with
        # the prediction
        confidence = detections[0, 0, i, 2]

        # filter out weak detections
        if confidence > args["confidence"]:
            # compute the (x, y)-coordinates of the bounding box for
            # the face
            box = detections[0, 0, i, 3:7] * np.array([w, h, w, h])
            (startX, startY, endX, endY) = box.astype("int")

            # extract the face ROI
            face = frame[startY:endY, startX:endX]
            (fH, fW) = face.shape[:2]

            # ensure the face width and height are sufficiently large
            if fW < 20 or fH < 20:
                continue

            # construct a blob for the face ROI, then pass the blob
            # through our face embedding model to obtain the 128-d
            # quantification of the face
            faceBlob = cv2.dnn.blobFromImage(face, 1.0 / 255,
                (96, 96), (0, 0, 0), swapRB=True, crop=False)
            embedder.setInput(faceBlob)
            vec = embedder.forward()

            # perform classification to recognize the face
            preds = recognizer.predict_proba(vec)[0]
            j = np.argmax(preds)
            proba = preds[j]
            name = le.classes_[j]

            if name == "nikhil":
                color = (0, 255, 0)
            else:
                color = (0, 0, 255)

            # draw the bounding box of the face along with the
            # associated probability
            text = "{}: {:.2f}%".format(name, proba * 100)
```

```
                        y = startY - 10 if startY - 10 > 10 else startY + 10
                        cv2.rectangle(frame, (startX, startY), (endX, endY),
                                color, 2)
                        cv2.putText(frame, text, (startX, y),
                                cv2.FONT_HERSHEY_SIMPLEX, 0.45, color, 2)

        # update the FPS counter
        fps.update()

        if check_output:
                videowriter.write(frame)
        else:
                # show the output frame
                cv2.imshow("Frame", frame)
                key = cv2.waitKey(1) & 0xFF

                # if the `q` key was pressed, break from the loop
                if key == ord("q"):
                        break

# stop the timer and display FPS information
fps.stop()
print("[INFO] elapsed time: {:.2f}".format(fps.elapsed()))
print("[INFO] approx. FPS: {:.2f}".format(fps.fps()))

# do a bit of cleanup
cv2.destroyAllWindows()
vs.stop()
if check_output:
        videowriter.release()
```