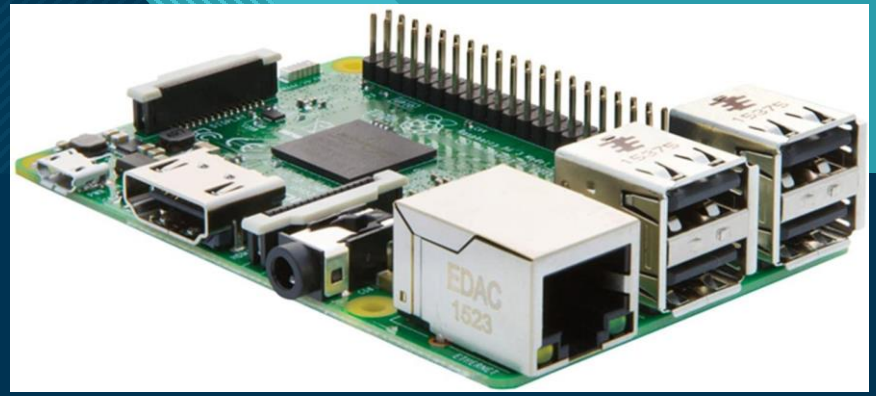
IoT Security Project

Created by Talisa Powell and Bradley Allen

CSCIU-509

4/20/22

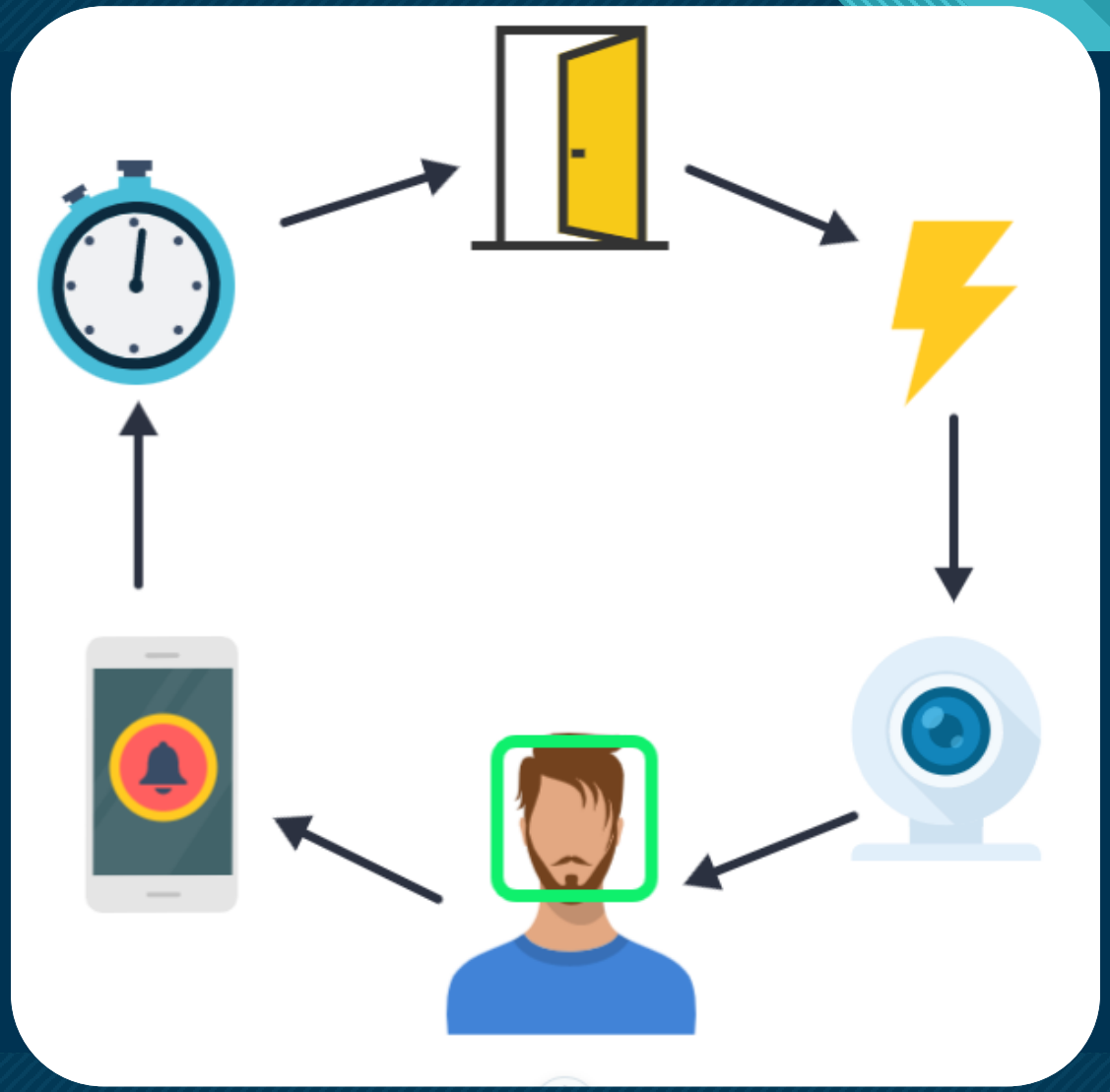
Project Overview



Our project was made using:

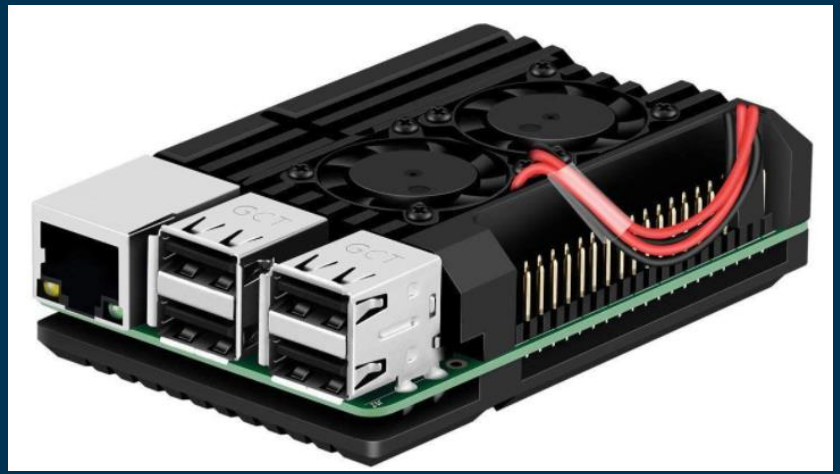
- Raspberry Pi 3B+
- Reed switch (magnetic relay)
- Breadboard
- Webcam

How It Works

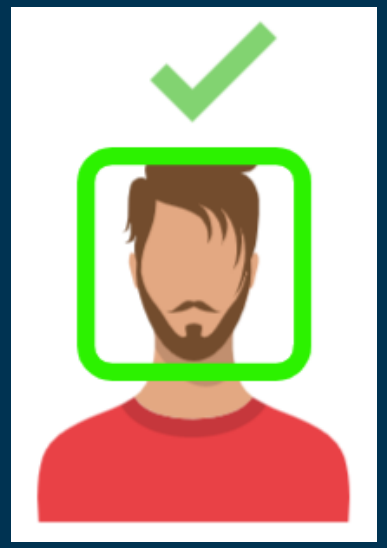


Challenges

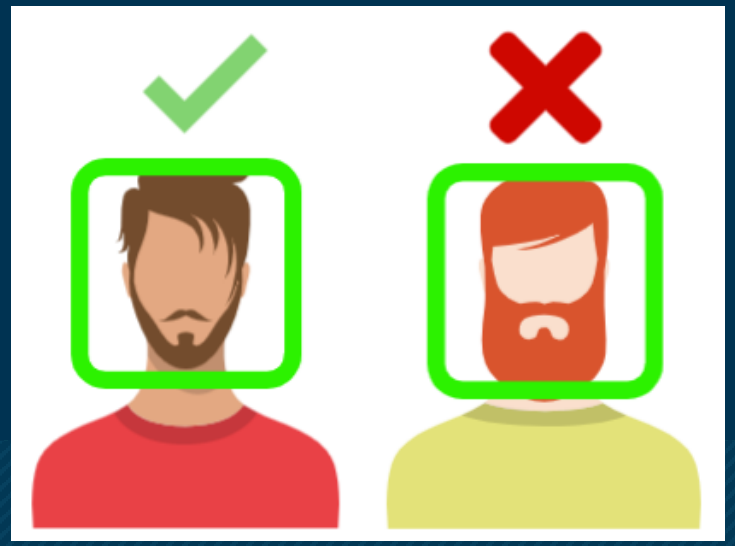
- Overheating
- Shorted reed switch
- Low voltage
- Frame rate
- Segment fault

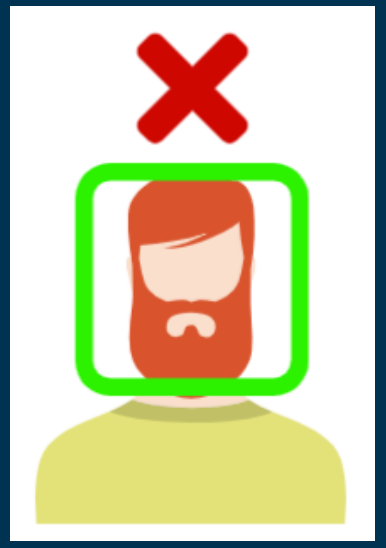


Capabilities



No Notification





Notifies

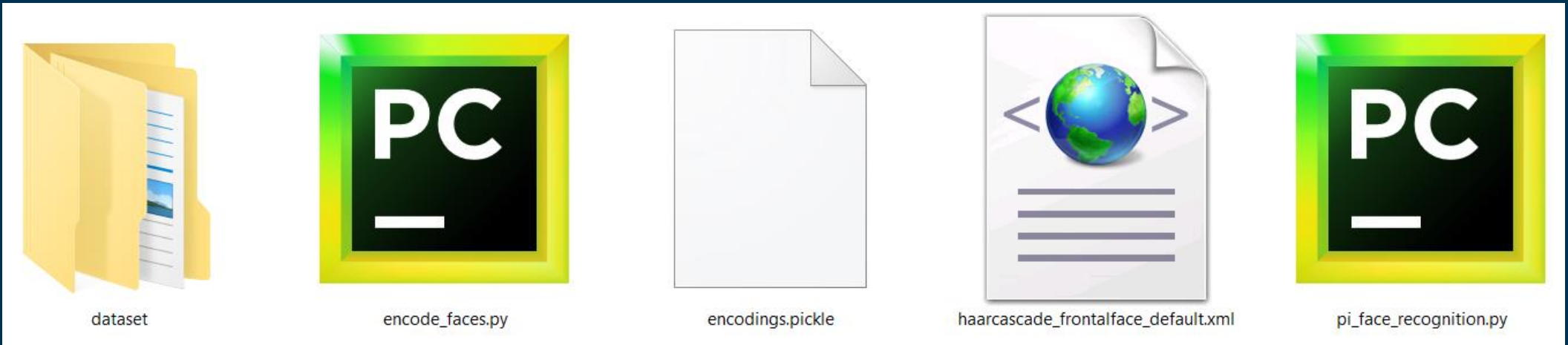
Limitations

- Limited Memory

Possible Improvements

- Larger memory storage
- Faster processor
- Improved facial detection

Logical Details (File Structure)



Logical Details (encode_faces.py)

```python
# encode_faces.py
# Used for learning faces from dataset folder
# Command below is used to learn the dataset folder,
# outputs facial encodings to --encodings (PATH) as a .pickle file.
# python encode_faces.py --dataset dataset
# --encodings encodings.pickle --detection-method hog

# import the necessary packages
from imutils import paths
import face_recognition
import argparse
import pickle
import cv2
import os

# construct the argument parser and parse the arguments
ap = argparse.ArgumentParser()
ap.add_argument("-i", "--dataset", required=True,
    help="path to input directory of faces + images")
ap.add_argument("-e", "--encodings", required=True,
    help="path to serialized db of facial encodings")
ap.add_argument("-d", "--detection-method", type=str, default="cnn",
    help="face detection model to use: either `hog` or `cnn`")
args = vars(ap.parse_args())

# grab the paths to the input images in our dataset
print("[INFO] quantifying faces...")
imagePaths = list(paths.list_images(args["dataset"]))

# initialize the list of known encodings and known names
knownEncodings = []
knownNames = []

# loop over the image paths
for (i, imagePath) in enumerate(imagePaths):
    # extract the person name from the image path
    print("[INFO] processing image {}/{}".format(i + 1,
        len(imagePaths)))
    name = imagePath.split(os.path.sep)[-2]

    # load the input image and convert it from BGR (OpenCV ordering)
    # to dlib ordering (RGB)
    image = cv2.imread(imagePath)
    rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

    # detect the (x, y)-coordinates of the bounding boxes
    # corresponding to each face in the input image
    boxes = face_recognition.face_locations(rgb,
        model=args["detection_method"])

    # compute the facial embedding for the face
    encodings = face_recognition.face_encodings(rgb, boxes)

    # loop over the encodings
    for encoding in encodings:
        # add each encoding + name to our set of known names and
        # encodings
        knownEncodings.append(encoding)
        knownNames.append(name)

# dump the facial encodings + names to disk
print("[INFO] serializing encodings...")
data = {"encodings": knownEncodings, "names": knownNames}
f = open(args["encodings"], "wb")
f.write(pickle.dumps(data))
f.close()
```

Logical Details (pi_face_recognition.py)

```python
from imutils.video import VideoStream
from imutils.video import FPS
import face_recognition
import argparse
import imutils
import pickle
import time
import cv2
from pushbullet import Pushbullet
import RPi.GPIO as GPIO

GPIO.setmode(GPIO.BCM)
inPin = 23
GPIO.setup(inPin, GPIO.IN)

# Builds argument line to avoid console
args = {}
args["cascade"] = "haarcascade_frontalface_default.xml"
args["encodings"] = "encodings.pickle"

# Load the known faces and embeddings along with OpenCV's Haar
# cascade for face detection
print("")
print("=-=-=-=-=-=-=-=-= INITIALIZATION =-=-=-=-=-=-=-=-=")
print("[Status] Loading Encodings and Face Detector...")
data = pickle.loads(open(args["encodings"], "rb").read())
detector = cv2.CascadeClassifier(args["cascade"])
print("[Status] Encodings and Face Detector successfully loaded.")
print("=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=")
print("")
print("[Status] Listening for door...")

# While the program is running (Raspberry Pi is turned on)
while True:

    # If voltage is read from pin (Door is opened), turn on camera
    if GPIO.input(inPin) == 1:
        print("[Status] Door opened, turning on camera...")

        # Initialize the video stream and allow the camera sensor to warm up
        print("[Status] Starting video stream...")
        # Use for USB camera
        vs = VideoStream(src=0).start()
        # Use for PiCamera
        # vs = VideoStream(usePiCamera=True).start()
        time.sleep(2.0)

        # Start the FPS counter and timer
        fps = FPS().start()
```

```python
        starttime = time.time()
        # Initializes the list of names from video stream
        namesExt = []

        # Loop over frames from the video file stream
        while True:
            # Grab the frame from the threaded video stream and resize it
            # to 300px (to speedup processing)
            frame = vs.read()
            frame = imutils.resize(frame, width=300)

            # Convert the input frame from (1) BGR to grayscale (for face
            # detection) and (2) from BGR to RGB (for face recognition)
            gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
            rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)

            # Detect faces in the grayscale frame
            rects = detector.detectMultiScale(gray, scaleFactor=1.1,
                minNeighbors=5, minSize=(30, 30))

            # OpenCV returns bounding box coordinates in (x, y, w, h) order
            # but we need them in (top, right, bottom, left) order, so we
            # need to do a bit of reordering
            boxes = [(y, x + w, y + h, x) for (x, y, w, h) in rects]

            # Compute the facial embeddings for each face bounding box
            encodings = face_recognition.face_encodings(rgb, boxes)
            names = []

            # Loop over the facial embeddings
            for encoding in encodings:
                # Attempt to match each face in the input image to our known
                # encodings
                matches = face_recognition.compare_faces(data["encodings"],
                    encoding)
                name = "Unknown"

                # Check to see if we have found a match
                if True in matches:
                    # Find the indexes of all matched faces then initialize a
                    # dictionary to count the total number of times each face
                    # was matched
                    matchedIdxs = [i for (i, b) in enumerate(matches) if b]
                    counts = {}

                    # Loop over the matched indexes and maintain a count for
                    # each recognized face face
                    for i in matchedIdxs:
                        name = data["names"][i]
```

Logical Details (pi_face_recognition.py)

```
                    counts[name] = counts.get(name, 0) + 1

                # Determine the recognized face with the largest number
                # of votes (note: in the event of an unlikely tie Python
                # will select first entry in the dictionary)
                name = max(counts, key=counts.get)

            # Update the list of names and list holding all names
            names.append(name)
            namesExt.append(name)

        # Loop over recognized faces
        for ((top, right, bottom, left), name) in zip(boxes, names):
            # Draw a rectangle over recognized faces
            cv2.rectangle(frame, (left, top), (right, bottom),
                (0, 255, 0), 2)
            y = top - 15 if top - 15 > 15 else top + 15
            cv2.putText(frame, name, (left, y), cv2.FONT_HERSHEY_SIMPLEX,
                0.75, (0, 255, 0), 2)

        # Display the video stream
        cv2.imshow("Video Stream", frame)
        key = cv2.waitKey(1) & 0xFF

        # Checks time, turns camera off after 20 seconds have passed
        checktime = time.time()
        endtime = starttime - checktime
        # print(endtime) # Just for testing
        if endtime <= -20:
            break

        # If "q" is pressed, break from video stream loop
        if key == ord("q"):
            break

        # Ping FPS
        fps.update()

    # Display information from video stream
    fps.stop()
    print("[Status] Elasped Time: {:.2f}".format(fps.elapsed()))
    print("[Status] Approximate FPS: {:.2f}".format(fps.fps()))

    # Initiate notification system
    pb = Pushbullet("o.u5SrykF33g3v58rFKNMN7OgJbla94g8t")
    device = pb.get_device("Samsung SM-G781U")

    # Video stream will have 4 scenarios:
    # 1) (Don't Notify) - Unknown face(s) detected, but a known face was also detected
    # 2) (NOTIFY) - Unknown face(s) detected, no known face(s) detected
    # 3) (Don't Notify) - Known face(s) detected, no unknown
    # 4) (NOTIFY) - No faces were detected
    flag = False
    if "Unknown" in namesExt:
        print("[Status] Unknown person detected, checking" +
              " list for known before sending notification...")
        flag = True
        # Checks if there was known face(s) with the unknown face
        for name in namesExt:
            if name != "Unknown":
                # The unknown was with a known face, does not notify
                flag = False
                print("[Status] Unknown is with Known, not sending notification.")
                break

        # We don't know anyone in the stream, notifies
        if flag:
            print("[ALERT] Only Unknown in list, sending notification to " + str(device))
            push = device.push_note("Facial Recognition Camera",
                                    "Your camera detected an unknown face.")
    # No unknowns
    else:
        # Creates a set of the names, the name is in the set only once
        nameSet = set(namesExt)

        # Checks if there were no faces, notifies
        if len(nameSet) == 0:
            print("[ALERT] No face detected. Sending notification to " + str(device))
            push = device.push_note("Facial Recognition Camera", "Your camera" +
                                    " was turned on and no faces were detected.")

        # Faces were detected, but no unknowns, prints detected faces
        else:
            for name in nameSet:
                print("[Status] Detected " + name)
            print("[Status] No Unknowns, not sending notification.")

    # Cleans up
    cv2.destroyAllWindows()
    vs.stop()
    vs.stream.release()

    # Gives update on status
    print("")
    print("[Status] Listening for door...")
```