iLock - Smart Facial Recognition Door Lock

A Raspberry Pi-based smart door lock system that uses facial recognition to automatically unlock doors for authorized people. Built with Python, OpenCV, and a servo motor.

What Does It Do?

iLock is a simple security system that:

- Recognizes people's faces using a camera
- Unlocks the door (via servo motor) when it sees someone it knows
- Locks the door when it sees a stranger
- All powered by your Raspberry Pi!

Hardware Requirements

- **Raspberry Pi** (3, 4, or 5)
- **Raspberry Pi Camera Module**
- **Servo Motor** (connected to GPIO pin 18)
- **Power supply** for Raspberry Pi
- **Optional**: Door lock mechanism simulated by the servo

Software Requirements

System Packages

```
sudo apt-get update
sudo apt-get install -y python3-picamera2 python3-opencv
sudo apt-get install -y build-essential cmake
sudo apt-get install -y libopenblas-dev liblapack-dev
```

Python Packages

```
pip3 install face_recognition imutils gpiozero opencv-python
```

Quick Start

1. Install iLock

```
# Clone the repository
git clone https://github.com/AKhadhraoui47/i_lock.git
cd i_lock

# Make the wrapper script executable
chmod +x ilock
```

2. Add People to the System

```
ilock picture
```

- Enter a person's name (e.g., "John")
- Press **SPACE** to capture photos (take 20-30 from different angles)
- Press **ESC** when done with that person
- Repeat for other people
- Type `done` when finished

3. Train the Model

```
ilock train
```

This processes all the photos and creates a face recognition model. Takes a few minutes.

4. Start the Security System

```
ilock security
```

The camera will now recognize faces and control the door lock automatically!

The iLock Command

The `ilock` wrapper script makes everything easy. Here are all the commands:

Command	What it does
`ilock picture`	Capture photos of people
`ilock train`	Train the face recognition model
`ilock security`	Start the door lock system
`ilock clean`	Delete all data and start over
`ilock help`	Show help information

How It Works (The Technical Details)

studio.py - Photo Capture System

Purpose: Captures training photos of people's faces for the recognition system.

Technical Implementation:

Camera Initialization

```
from picamera2 import Picamera2
import cv2
import os
import sys

# Get person's name from command line or prompt
name = sys.argv[1] if len(sys.argv) > 1 else input("Enter name: ")

# Create directory structure
os.makedirs(f"dataset/{name}", exist_ok=True)
```

The script uses `picamera2`, which is the modern camera interface for Raspberry Pi that works with the `libcamera` stack. The older `picamera` library relied on legacy drivers and generally causes the `libbcm_host.so` errors.

Camera Configuration

```
picam2 = Picamera2()
config = picam2.create_preview_configuration(
    main={"size": (512, 304), "format": "RGB888"}
)
picam2.configure(config)
picam2.start()
```

Why these settings?

- **512x304 resolution:** Smaller images process faster. Face recognition doesn't need high resolution.
- **RGB888 format:** 8 bits per color channel (Red, Green, Blue). Standard format for color images.

Image Capture Loop

```
while True:
    image = picam2.capture_array()
    cv2.imshow(f"Press Space - {name}", image)

    k = cv2.waitKey(1)

    if k % 256 == 27:
        break
    elif k % 256 == 32:
        img_name = f"dataset/{name}/image_{img_counter}.jpg"
        cv2.imwrite(img_name, image)
        img_counter += 1
```

Best practices for photo capture:

- Take 20-30 photos per person
- Vary head angles: straight, left, right, up, down
- Different expressions: neutral, smiling
- Various lighting conditions
- With/without glasses if applicable
- Different distances from camera

---

gym.py - Face Encoding Training System

Purpose: Processes all captured photos and creates a machine learning model that can recognize faces.

Technical Implementation:

Loading Images

```
from imutils import paths
import face_recognition
import pickle
import cv2

# Get all image paths
imagePaths = list(paths.list_images("dataset"))

# Extract person names from directory structure
# dataset/John/image_0.jpg → name = "John"
name = imagePath.split(os.path.sep)[-2]
```

The `imutils.paths.list_images()` function recursively finds all images (`.jpg`, `.png`, etc.) in the dataset folder.

Face Detection and Encoding

```
# Load image
image = cv2.imread(imagePath)
rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

# Detect face locations
boxes = face_recognition.face_locations(rgb, model="hog")

# Generate 128-dimensional face encodings
encodings = face_recognition.face_encodings(rgb, boxes)
```

What's happening here?

1. **HOG (Histogram of Oriented Gradients):** A fast face detection method that looks for facial features like eyes, nose, mouth
2. **Face boxes:** Returns coordinates `(top, right, bottom, left)` for each face
3. **Face encodings:** Creates a 128-number "fingerprint" of each face using deep learning

The 128-dimensional encoding:

- Each number represents a facial measurement
- Things like: eye distance, nose width, jawline shape, etc.
- These 128 numbers uniquely identify a person's face

Storing the Model

```
data = {
    "encodings": knownEncodings,
    "names": knownNames
}

# Save to disk using pickle
f = open("encodings.pickle", "wb")
f.write(pickle.dumps(data))
```

Why pickle?

- Pickle serializes Python objects to binary format
- Fast to save and load
- Preserves the exact data structure

Model structure:

```
{
    "encodings": [
        [0.123, -0.456, 0.789, ...],
        [0.124, -0.455, 0.791, ...],
        [0.987, 0.654, -0.321, ...]
    ],
    "names": ["John", "John", "Jane"]
}
```

---

security.py - Real-Time Face Recognition & Door Control

Purpose: The main security system that recognizes faces in real-time and controls the door lock.

Technical Implementation:

System Initialization

```
from picamera2 import Picamera2
import face_recognition
import pickle
from gpiozero import AngularServo

# Initialize servo motor (GPIO pin 18)
servo = AngularServo(
    18,
    initial_angle=0,
    min_pulse_width=0.0006,
    max_pulse_width=0.0023
)

# Load trained model
data = pickle.loads(open("encodings.pickle", "rb").read())

# State tracking
currentname = "Unknown"
```

Servo control explained:

- **GPIO pin 18:** Physical pin where servo is connected
- **Pulse width:** Controls servo angle (calibrated for specific servo model)
- **0° = locked, 90° = unlocked** (adjust based on your lock mechanism)

Main Recognition Loop

```
while True:

    frame = picam2.capture_array()
    frame = cv2.cvtColor(frame, cv2.COLOR_RGB2BGR)
    frame = imutils.resize(frame, width=500)



    boxes = face_recognition.face_locations(frame)
    encodings = face_recognition.face_encodings(frame, boxes)
```

Frame resizing: Smaller frames = faster processing. 500px width is enough for face recognition.

Face Matching Algorithm

```
for encoding in encodings:

    matches = face_recognition.compare_faces(
        data["encodings"],
        encoding
    )

    name = "Unknown"

    if True in matches:

        matchedIdxs = [i for (i, b) in enumerate(matches) if b]
        counts = {}



        for i in matchedIdxs:
            name = data["names"][i]
            counts[name] = counts.get(name, 0) + 1



        name = max(counts, key=counts.get)
```

The voting system explained:

If John has 25 training photos and 20 match, while noise causes 2 false matches:

```
counts = {"John": 20, "Jane": 2}
name = max(counts) = "John"  ✓ Correct!
```

This makes recognition more robust against false positives.

Door Lock Control Logic

```
if currentname != name:
    currentname = name

    if name != "Unknown":

        print(f"[RECOGNIZED] {name}")
        servo.angle = 90
        print("[SERVO] Door unlocked")
    else:

        print("[WARNING] Unknown person")
        servo.angle = 0
        print("[SERVO] Door locked")
```

State tracking prevents spam:

- Without `currentname != name` check, servo would move every frame
- Door would constantly lock/unlock (bad for servo motor)
- With check: only triggers on state changes

Visual Feedback

```
for ((top, right, bottom, left), name) in zip(boxes, names):

    cv2.rectangle(frame, (left, top), (right, bottom), (0, 255, 225), 2)



    y = top - 15 if top - 15 > 15 else top + 15
    cv2.putText(frame, name, (left, y),
                cv2.FONT_HERSHEY_SIMPLEX, 0.8, (0, 255, 255), 2)

cv2.imshow("Facial Recognition is Running", frame)
```

Color codes (BGR format):

- `(0, 255, 225)` = Cyan rectangle
- `(0, 255, 255)` = Yellow text

Performance Monitoring

```
frame_count = 0
start_time = time.time()

# ... in loop
frame_count += 1

# At end
elapsed = time.time() - start_time
fps = frame_count / elapsed
print(f"FPS: {fps:.2f}")
```

The iLock Wrapper Script

The `ilock` bash script ties everything together with a clean interface.

Features:

- Color-coded output for readability
- Error checking and validation
- Helpful prompts and messages
- Safe cleanup operations
- Confirmation for destructive actions

Architecture:

```
ilock (wrapper)
├── ilock picture → studio.py
├── ilock train   → gym.py
├── ilock security → security.py
└── ilock clean   → rm -rf dataset/ encodings.pickle
```

Project Structure

```
i_lock/
├── ilock
├── studio.py
├── gym.py
├── security.py
├── dataset/
│   ├── John/
│   │   ├── image_0.jpg
│   │   └── image_1.jpg
│   └── Jane/
│       └── image_0.jpg
└── encodings.pickle
```

Usage Tips

Initial Setup Workflow

Step 1: Capture Training Photos

```
ilock picture
```

When the prompt appears:

1. **Enter the person's name** (e.g., "John")
2. **Position yourself in front of the camera**
3. **Press SPACE** to capture a photo - you'll hear the shutter sound
4. **Move slightly** and take another photo
5. **Repeat 20-30 times** with variations:
    - Face straight ahead
    - Turn head slightly left (~15-20°)
    - Turn head slightly right (~15-20°)
    - Look slightly up
    - Look slightly down
    - Different facial expressions (neutral, smile)
    - With glasses on/off (if applicable)
    - Different lighting (if possible)
6. **Press ESC** when done with this person
7. **Repeat for additional people** or type `done` to finish

Pro tips:

- More photos = better recognition
- Quality over quantity - ensure face is clearly visible
- Take photos in the same location where the system will be used
- Capture photos at different times of day for varying lighting
- Distance: sit/stand 1-2 meters from camera

Step 2: Train the Recognition Model

```
ilock train
```

What happens during training:

- The script scans your `dataset/` folder
- Finds all images for all people
- Detects faces in each image
- Extracts 128 facial features per face
- Creates the `encodings.pickle` model file

Expected output:

```
[INFO] quantifying faces...
[INFO] processing image 1/50
[INFO] processing image 2/50
...
[INFO] serializing encodings...
Done!
```

Training time:

- Raspberry Pi 4: ~2-5 seconds per image
- 50 total images: ~3-5 minutes
- Raspberry Pi 5: About 50% faster

Step 3: Run the Security System

```
ilock security
```

The system is now active! Here's what you'll see:

- **Live camera feed** with face detection
- **Green boxes** around detected faces
- **Names displayed** above recognized faces
- **"Unknown"** for unrecognized faces
- **Console messages** when lock state changes

What to expect:

- Recognition happens in real-time (~5-10 FPS)
- Door unlocks automatically when known face detected
- Door locks when unknown face appears
- Multiple faces can be detected simultaneously

To exit: Press `Q` or `ESC`

Adding New People

You don't need to start over! Just add more photos and retrain:

```
# Add photos for new person
ilock picture
# Enter "Sarah"
# Take 20-30 photos
# Type "done"

# Retrain with new data
ilock train

# Run system with updated model
ilock security
```

The new person will be recognized alongside existing people.

Starting Fresh

If you want to completely reset the system:

```
ilock clean
```

This will:

- Delete all photos in `dataset/`
- Delete the trained model (`encodings.pickle`)