Closed Loop Automation

* continuously monitors it’s own output
* compares the output with desired target
* Uses the real-time feedback
* Adjust the behaviour based on the feedback

**Models Used In CLA:**

* **PID Controller** (Proportional–Integral–Derivative) proportional to current error, based on sum of past errors, based on prediction of future error.
* **State Space Model (**Describes the system using **mathematical equations of states**.)
* **Fuzzy Logic Controller(**Uses human-like reasoning)
* **Bang-Bang Controller** (On-Off Control)

import face\_recognition

import cv2

import os

import pyttsx3

# Initialize text-to-speech engine

engine = pyttsx3.init()

engine.setProperty('rate', 150)

# Load known faces

KNOWN\_FACES\_DIR = r"C:\Users\BHAVAGNA\OneDrive\Desktop\ezzample\images"

known\_face\_encodings = []

known\_face\_names = []

for person\_name in os.listdir(KNOWN\_FACES\_DIR):

    person\_folder = os.path.join(KNOWN\_FACES\_DIR, person\_name)

    if not os.path.isdir(person\_folder):

        continue

    for image\_name in os.listdir(person\_folder):

        image\_path = os.path.join(person\_folder, image\_name)

        image = face\_recognition.load\_image\_file(image\_path)

        encodings = face\_recognition.face\_encodings(image, num\_jitters=3)

        if len(encodings) == 0:

            print(f"[WARNING] No face found in {image\_path}, skipping.")

            continue

        known\_face\_encodings.append(encodings[0])

        known\_face\_names.append(person\_name)

video\_capture = cv2.VideoCapture(0)

print("Press 'q' to quit")

already\_announced = set()

while True:

    ret, frame = video\_capture.read()

    if not ret:

        print("[ERROR] Failed to grab frame.")

        break

    small\_frame = cv2.resize(frame, (0, 0), fx=0.5, fy=0.5)

    rgb\_small\_frame = cv2.cvtColor(small\_frame, cv2.COLOR\_BGR2RGB)

    face\_locations = face\_recognition.face\_locations(rgb\_small\_frame, model='hog')

    face\_encodings = face\_recognition.face\_encodings(rgb\_small\_frame, face\_locations)

    face\_names = []

    for face\_encoding in face\_encodings:

        matches = face\_recognition.compare\_faces(known\_face\_encodings, face\_encoding, tolerance=0.45)

        name = "Unknown"

        face\_distances = face\_recognition.face\_distance(known\_face\_encodings, face\_encoding)

        if len(face\_distances) > 0:

            best\_match\_index = face\_distances.argmin()

            if matches[best\_match\_index]:

                name = known\_face\_names[best\_match\_index]

        face\_names.append(name)

    # Draw and announce

    for (top, right, bottom, left), name in zip(face\_locations, face\_names):

        top \*= 2

        right \*= 2

        bottom \*= 2

        left \*= 2

        color = (0, 0, 255) if name == "Unknown" else (0, 255, 0)

        cv2.rectangle(frame, (left, top), (right, bottom), color, 2)

        cv2.rectangle(frame, (left, bottom - 35), (right, bottom), color, cv2.FILLED)

        cv2.putText(frame, name, (left + 6, bottom - 6), cv2.FONT\_HERSHEY\_DUPLEX, 0.9, (255, 255, 255), 1)

        # Only speak for known and not already announced

        if name != "Unknown" and name not in already\_announced:

            print(f"{name} recognized.")

            engine.say(f"{name} detected")

            engine.runAndWait()

            already\_announced.add(name)

    cv2.imshow("Face Recognition", frame)

    if cv2.waitKey(1) & 0xFF == ord('q'):

        break

video\_capture.release()

cv2.destroyAllWindows()

**output:**

