Automatic Parking Entry System - Full Development Report

Introduction

A Word from Gabriel Baziramwabo, CEO of Benax Technologies:

As part of our commitment to empower learners through real-world engineering experiences, I'm proud to share with you the story behind the creation of our Automatic Parking Entry System. This system is more than just a project — it's the foundation of the ***Automatic Gate Demo Kit****, a learning tool designed to bridge classroom theory with hands-on, industry-grade innovation.*

This journey reflects the challenges we faced, the solutions we engineered, and the technologies we brought together to build a reliable, smart gate control system. I hope it inspires and guides you as you build your own automated systems.

From Detection to Decision: A Seamless Integration of Sensors, AI, and Control

Designing an Automatic Parking Entry System wasn't just about raising a barrier - it was about orchestrating precision, speed, and smart decision-making in real-time. Our goal was to create a fully autonomous solution that detects incoming vehicles, identifies their number plates, verifies their validity, and grants access - all without human intervention.

Stage 1: Ultrasonic Sensor - The System's First Respondent

The process began with an ultrasonic sensor connected to a microcontroller, acting as the system's initial line of detection. It continuously scanned the area in front of the gate, and when a vehicle approached within 50 centimeters, it triggered the next phase.

Why this matters:
- *Performance Boost:* Prevents the AI pipeline from processing unnecessary frames.
- *Real-Time Optimization:* Ensures plate detection happens only when a car is truly present.

By making the ultrasonic sensor the gatekeeper, we built an intelligent, event-driven system from the start.

Stage 2: YOLOv8 — Accurate Plate Detection with AI

Next, the system activated **YOLOv8**, a state-of-the-art object detection model that played a pivotal role in reliably identifying license plates. Unlike generic pre-trained models that struggle with regional nuances, our YOLOv8 model was **custom-trained specifically on Rwandan license plate formats** — in this case, demo plates from our toy car prototype.

To achieve this, we began by capturing **about 100 images** of a toy car plate under a wide variety of real-world conditions:

- *Close-up and distant shots*
- *Angled (perspective) distortions*
- *Different lighting environments*
- *Various backgrounds (floor, table, textured surfaces)*

Each image was then carefully *annotated using LabelImg*, assigning the bounding box to a *single class: license_plate*. These annotated images were divided into two folders:
- ***`dataset/train/`*** *– containing 80% of the images for training*

• ***dataset/val/*** *– containing 20% for validation*

This 80-20 split was critical to help the model generalize well and avoid overfitting, especially when dealing with slight differences in how the plate is presented. Overfitting is when a machine learning model performs exceptionally well on its training data but fails to generalize to new, unseen data.

In other words, the model has memorized patterns, noise, or even irrelevant details from the training set, rather than learning the underlying features that apply broadly. As a result, it makes poor predictions when faced with validation or real-world inputs.

For example, if our YOLOv8 model had been trained on all 100 images without a proper validation set:
• *It might perfectly detect plates in those specific images.*
• *But it could struggle to detect plates in slightly different lighting, angles, or backgrounds not seen during training.*

That's why using an 80-20 split between training and validation is essential — it helps verify that the model is learning generalizable features and not just memorizing examples.

Why all this mattered:
• *If YOLOv8 failed to detect or slightly misaligned the plate, OCR would completely break down.*
• *A high-quality bounding box was essential for successful image cropping and clean text extraction.*

As a result, YOLOv8 became t*he system's crucial component*, enabling high-speed, accurate plate localization — no matter the distance, angle, or lighting.

Stage 3: OpenCV + Tesseract - Extracting the Plate Number

Once the plate was located, OpenCV took over to crop and enhance the image for text recognition.

The pipeline included:
- *Grayscale conversion: Simplifies the image for analysis.*
- *Gaussian blur: Smooths out visual noise.*
- *Otsu's thresholding: Produces a clean binary image.*

These preprocessing steps prepared the image for Tesseract OCR, which then extracted the alphanumeric plate number - for example, "RAH972U".

Stage 4: Validation Rules - Making Sure the Plate is Legit

Extracting text wasn't enough - it had to be accurate and in the correct format.

We enforced rigorous validation rules:
1. Start with "RA" - discarding any characters before it.
2. Total length of 7 characters (excluding spaces).
3. First 3 characters = uppercase letters (e.g., RAH, RAG).
4. Next 3 characters = digits.
5. Final character = uppercase letter.

This ensured the OCR output matched official Rwandan plate formats and weeded out any misreads.

Stage 5: Voting System - Improving Confidence Through Redundancy

Tesseract isn't perfect, especially under motion or glare. So we introduced a majority voting system:
- Buffer 3 consecutive OCR results.
- Accept only if 2 or more match.

Example:
Extracted -> RAH972U, RAH973U, RAH972U
Final vote -> RAH972U (wins by majority)

This added an extra layer of confidence and prevented errors from causing incorrect gate activations.

Stage 6: Logging and Controlling the Gate

Once a valid plate number was confirmed:
- It was logged to a CSV file with a timestamp.
- A check ensured that the same plate wasn't re-logged within 5 minutes, avoiding duplicates.
- A '1' was sent over serial communication to the Arduino to trigger the servo motor.
- The barrier lifted for 15 seconds, after which a '0' closed the gate.

This handshake between software and hardware completed the entry cycle - smooth, automated, and logged.

Final Thoughts: Engineering Trust Into Every Step

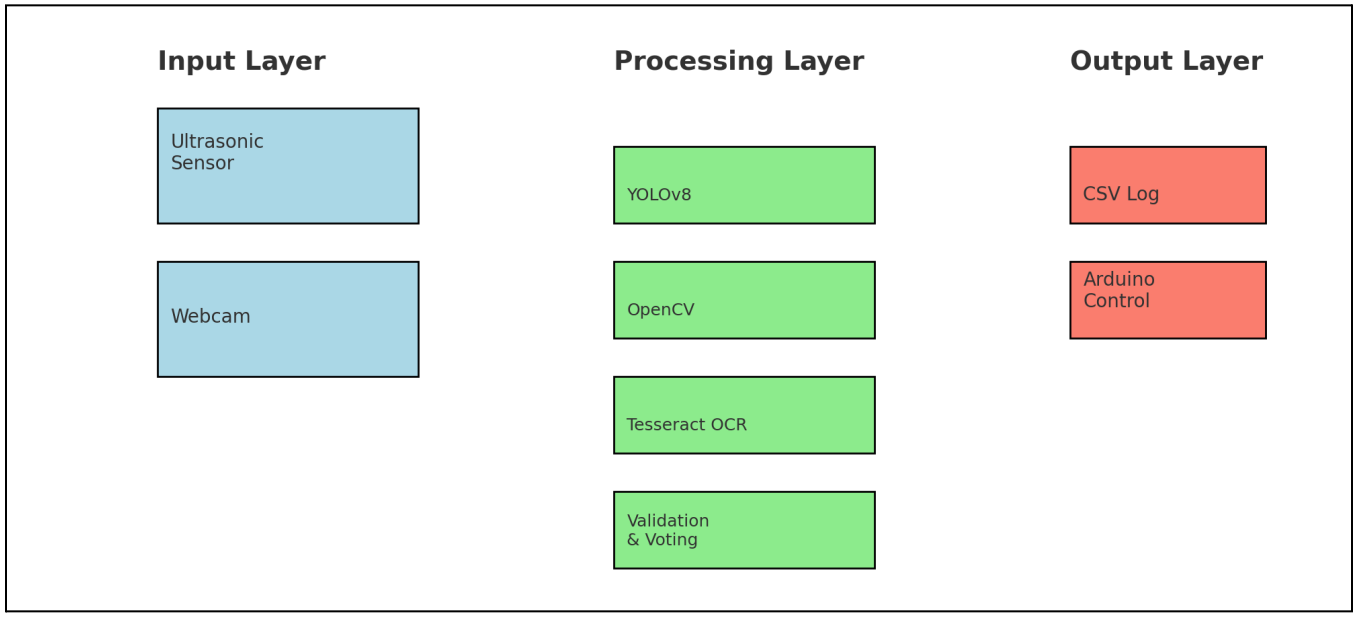
This system showcases the power of combining embedded systems, AI, and classical computer vision:
- Sensor detects the car.
- YOLOv8 finds the plate.
- OpenCV + Tesseract extract the number.
- Logic and validation ensure accuracy.
- Arduino hardware enforces control.

We didn't just automate the gate - we engineered a trusted, intelligent access system, ready for real-world deployment.

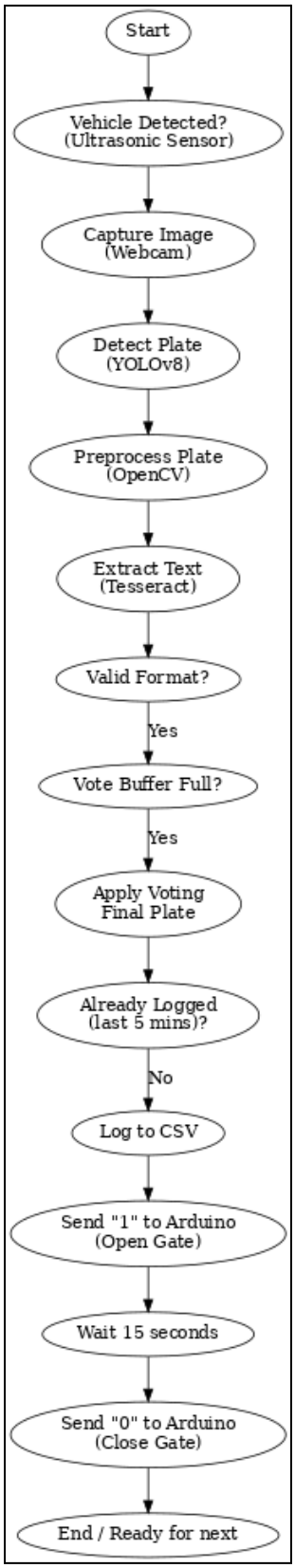
System Diagrams

The following diagrams illustrate how the system components interact, particularly the integration between the PC and the microcontroller:

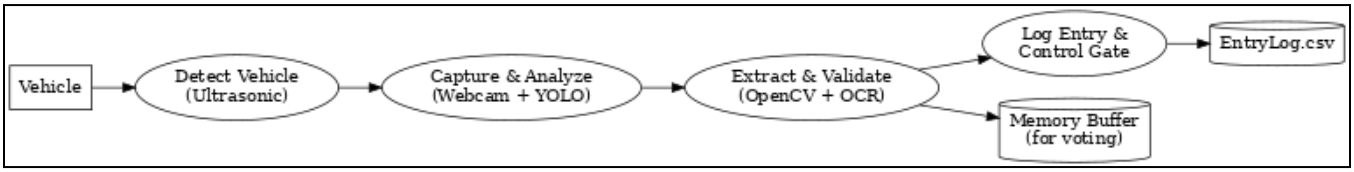
- System Architecture



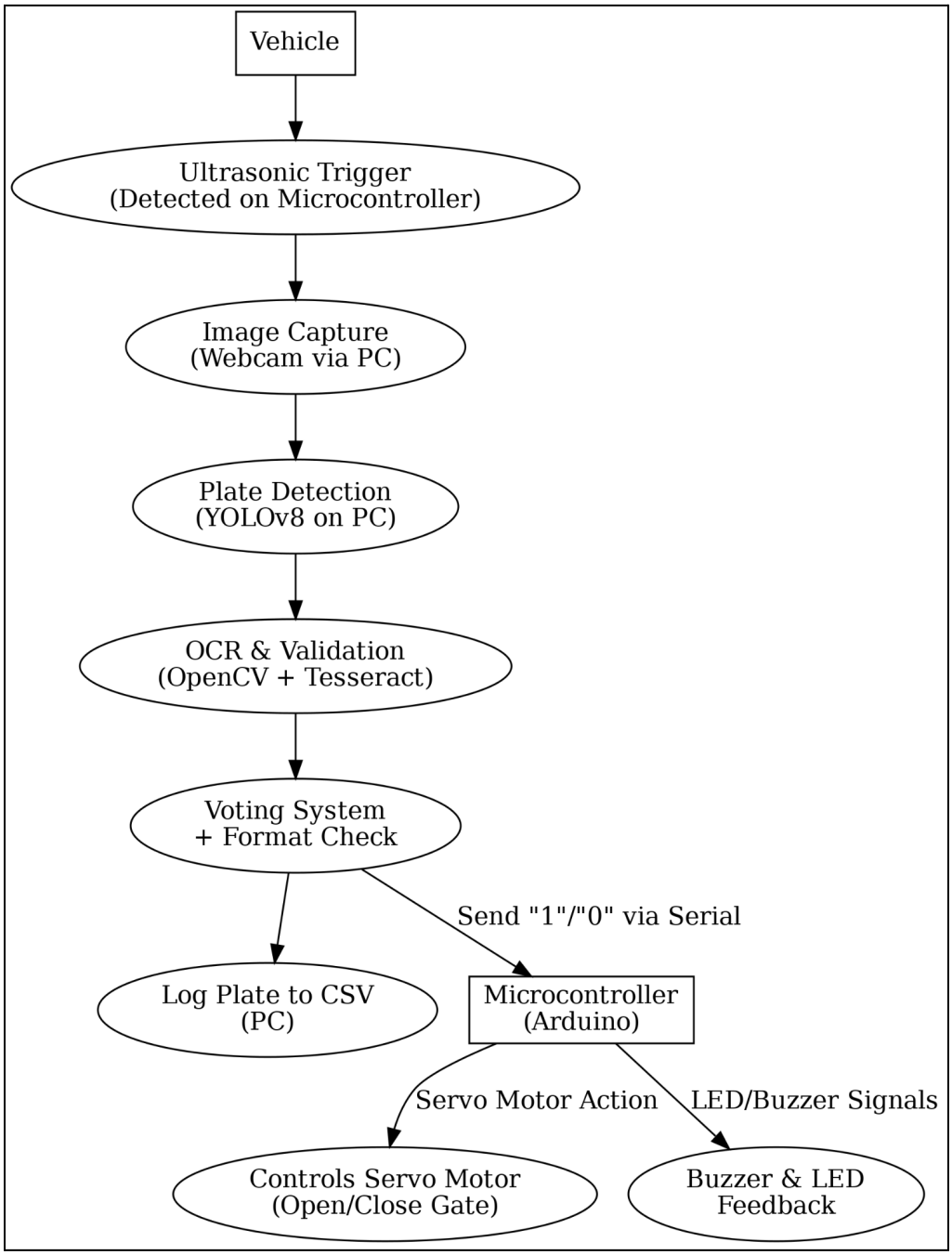
- Flowchart


Start
Vehicle Detected?
(Ultrasonic Sensor)
Capture Image
(Webcam)
Detect Plate
(YOLOv8)
Preprocess Plate
(OpenCV)
Extract Text
(Tesseract)
Valid Format?
Yes
Vote Buffer Full?
Yes
Apply Voting
Final Plate
Already Logged
(last 5 mins)?
No
Log to CSV
Send "1" to Arduino
(Open Gate)
Wait 15 seconds
Send "0" to Arduino
(Close Gate)
End / Ready for next

- Data Flow Diagram (DFD)



- PC–Microcontroller Integration Diagram



C++ Code: Gate Controller

```
#include <Servo.h>

#define TRIGGER_PIN 15  // D8 (GPI15) for Wemos D1 Mini
#define ECHO_PIN    13  // D7 (GPI13) for Wemos D1 Mini
#define SERVO_PIN    2  // D4 (GPIO2) for Wemos D1 Mini

Servo barrierServo;

void setup() {
  Serial.begin(115200);
  pinMode(TRIGGER_PIN, OUTPUT);
  pinMode(ECHO_PIN, INPUT);

  barrierServo.attach(SERVO_PIN);
  barrierServo.write(0);  // Start closed
}

void loop() {
  digitalWrite(TRIGGER_PIN, LOW);
  delayMicroseconds(2);
  digitalWrite(TRIGGER_PIN, HIGH);
  delayMicroseconds(10);
  digitalWrite(TRIGGER_PIN, LOW);

  long duration = pulseIn(ECHO_PIN, HIGH);
  float distance_cm = (duration * 0.0343) / 2.0;
  Serial.println(distance_cm);

  if (Serial.available()) {
    char cmd = Serial.read();
    if (cmd == '1') {
      barrierServo.write(180);  // Open
    } else if (cmd == '0') {
      barrierServo.write(0);    // Close
    }
  }

  delay(50);
}
```

Python Code: Plate Recognition & Control

```
import cv2
from ultralytics import YOLO
import pytesseract
import os
import time
import serial
import serial.tools.list_ports
import csv
from collections import Counter
import random

model = YOLO('/opt/homebrew/runs/detect/train4/weights/best.pt')

save_dir = 'plates'
os.makedirs(save_dir, exist_ok=True)

csv_file = 'plates_log.csv'
if not os.path.exists(csv_file):
    with open(csv_file, 'w', newline='') as f:
        writer = csv.writer(f)
        writer.writerow(['Plate Number', 'Timestamp'])

def detect_arduino_port():
    ports = list(serial.tools.list_ports.comports())
    for port in ports:
        if "usbserial" in port.device or "wchusbserial" in port.device:
            return port.device
    return None

arduino_port = detect_arduino_port()
arduino = serial.Serial(arduino_port, 115200, timeout=1) if arduino_port else None
if arduino: time.sleep(2)

def mock_ultrasonic_distance():
    return random.choice([random.randint(10, 40)] + [random.randint(60, 150)] *
10)

cap = cv2.VideoCapture(0)
plate_buffer = []
entry_cooldown = 300
last_saved_plate = None
last_entry_time = 0

print("[SYSTEM] Ready. Press 'q' to exit.")


while True:
    ret, frame = cap.read()
    if not ret:
        break

    distance = mock_ultrasonic_distance()
    print(f"[SENSOR] Distance: {distance} cm")
```

```
    if distance <= 50:
        results = model(frame)

        for result in results:
            for box in result.boxes:
                x1, y1, x2, y2 = map(int, box.xyxy[0])
                plate_img = frame[y1:y2, x1:x2]

                gray = cv2.cvtColor(plate_img, cv2.COLOR_BGR2GRAY)
                blur = cv2.GaussianBlur(gray, (5, 5), 0)
                thresh = cv2.threshold(blur, 0, 255, cv2.THRESH_BINARY +
cv2.THRESH_OTSU)[1]

               plate_text = pytesseract.image_to_string(
                   thresh, config='--psm 8 --oem 3 -c
tessedit_char_whitelist=ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789'
               ).strip().replace(" ", "")

                if "RA" in plate_text:
                    start_idx = plate_text.find("RA")
                    plate_candidate = plate_text[start_idx:]
                  if len(plate_candidate) >= 7:
                      plate_candidate = plate_candidate[:7]
                prefix, digits, suffix = plate_candidate[:3], plate_candidate[3:6], plate_candidate[6]
                    if (prefix.isalpha() and prefix.isupper() and
                        digits.isdigit() and suffix.isalpha() and suffix.isupper()):

                          print(f"[VALID] Plate Detected: {plate_candidate}")
                          plate_buffer.append(plate_candidate)

                          if len(plate_buffer) >= 3:
                           most_common = Counter(plate_buffer).most_common(1)[0][0]
                              current_time = time.time()

                           if (most_common != last_saved_plate or
                               (current_time - last_entry_time) > entry_cooldown):

                        with open(csv_file, 'a', newline='') as f:
                            writer = csv.writer(f)
                            writer.writerow([most_common, time.strftime('%Y-%m-%d %H:%M:%S')])
                                  print(f"[SAVED] {most_common} logged to CSV.")

                                  if arduino:
                                      arduino.write(b'1')
                                      print("[GATE] Opening gate (sent '1')")
                                      time.sleep(15)
                                      arduino.write(b'0')
                                      print("[GATE] Closing gate (sent '0')")

                                  last_saved_plate = most_common
                                  last_entry_time = current_time
                              else:
                               print("[SKIPPED] Duplicate within 5 min window.")

                              plate_buffer.clear()
```

```
                cv2.imshow("Plate", plate_img)
                cv2.imshow("Processed", thresh)
                time.sleep(0.5)

    annotated_frame = results[0].plot() if distance <= 50 else frame
    cv2.imshow('Webcam Feed', annotated_frame)

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

cap.release()
if arduino:
    arduino.close()
cv2.destroyAllWindows()
```