Architektura systemów brzegowych - Projekt

Rozpoznawanie tablic rejestracyjnych na bramce parkingowej

Paweł Marczewski 148099

1. Założenia projektu

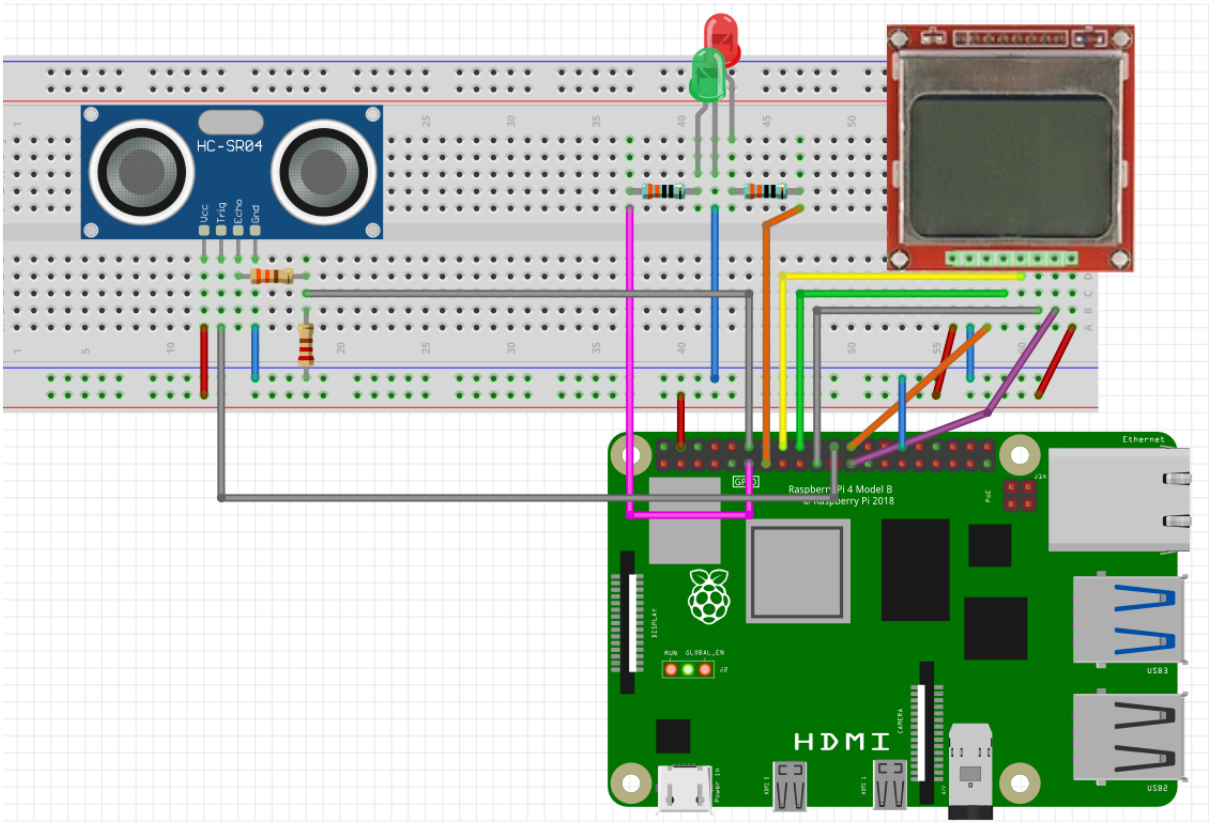
Celem projektu jest stworzenie systemu rozpoznawania numerów rejestracyjnych pojazdów, który będzie w stanie:

1. Wykrywać obecność pojazdów za pomocą czujnika odległości.
2. Przechwytywać obraz pojazdu za pomocą kamery.
3. Wykrywać i rozpoznawać numery rejestracyjne pojazdów z przechwyconego obrazu.
4. Sprawdzać, czy rozpoznany numer rejestracyjny znajduje się w bazie danych dozwolonych numerów.
5. Sygnalizować stan (dozwolony/niedozwolony) za pomocą diod LED.
6. Wyświetlać informacje o numerze rejestracyjnym na wyświetlaczu LCD.
7. Zapisywać dane dotyczące rozpoznanych numerów rejestracyjnych w lokalnej i zdalnej bazie danych
8. Synchronizować dane między bazami danych

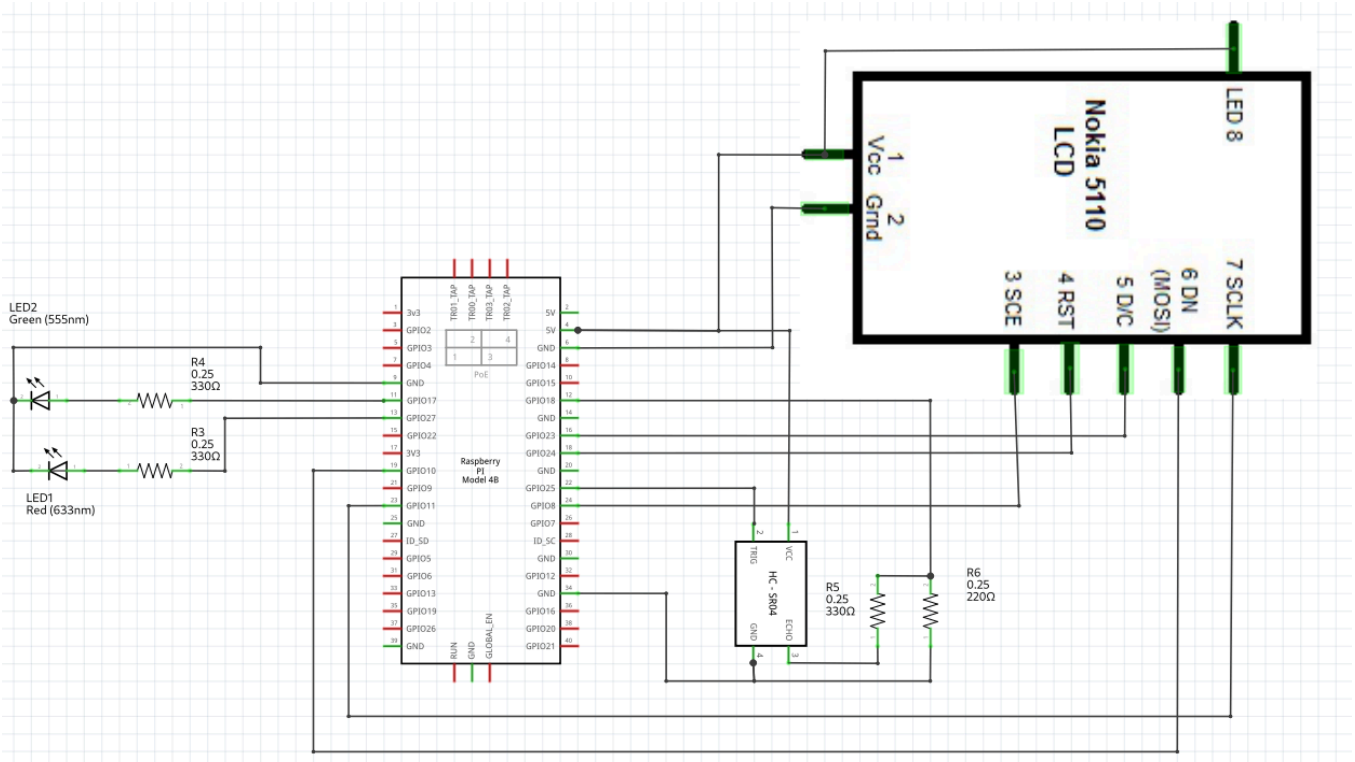
2. Warstwa sprzętowa

Wykorzytsane komponenty:

- **Raspberry Pi 3B**: Główna jednostka sterująca
- **Kamera internetowa na USB**: Do przechwytywania obrazu
- **Wyświetlacz Nokia 5110 LCD**: Wyświetlacz do pokazywania informacji
- **Czujnik odległości HC-SR04**: Do wykrywania obecności pojazdów
- **Diody LED (czerwona i zielona)**: Do sygnalizacji stanu
- **Przewody połączeniowe**: Do połączeń między komponentami
- **Rezystory** - 3x330Ω i 220Ω.
- **Płytka prototypowa:** Do ułożenia i połączenia komponentów



Rys. 1. Schemat podgladowy połączeń komponentów



Rys. 2. Schemat logiczny połączeń komponentów

Schemat przedstawia połączenia między Raspberry Pi, wyświetlaczem LCD, diodami LED oraz czujnikiem odległości. Wyświetlacz LCD jest podłączony do Raspberry Pi przez interfejs SPI. Dioda czerwona jest podłączona do pinu GPIO17,

a dioda zielona do GPIO27. Czujnik odległości jest podłączony do pinów GPIO25 (trigger) i GPIO18 (echo).

3. Instalacja bibliotek i konfiguracja

Najpierw należy zainstalować system na Raspberry Pi, w tym celu najlepiej podążać według oficjalnej instrukcji: [Getting started - Raspberry Pi Documentation]. Gdy system będzie już uruchomiony wejdź do terminala i kontynuuj podane instrukcję

Edytuj plik /boot/config.txt, aby włączyć kamerę.

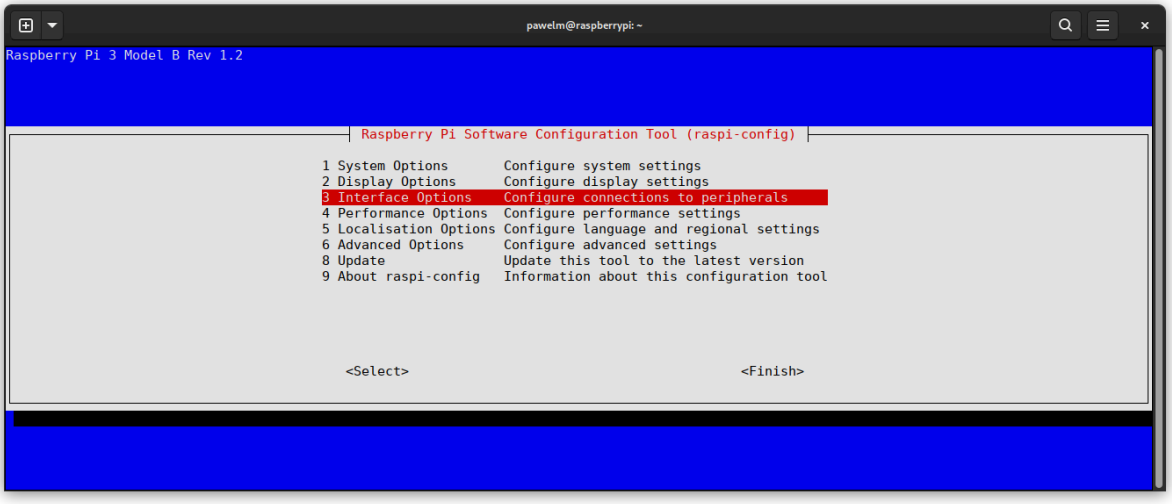
```
sudo nano /boot/config.txt
```

Ustaw zmienne tak jak pokazano poniżej:

```
# Włącza rozszerzone funkcje, takie jak aparat.
start_x=1
# Musi to być co najmniej 128 MB dla przetwarzania przez kamerę, jeśli jest większe, możesz po prostu zostawić je bez zmian.
gpu_mem=128
# Musisz zakomentować/usunąć istniejącą linię Camera_auto_detect, ponieważ powoduje to problemy z przechwytywaniem OpenCV/V4L2.
#camera_auto_detect=1
```

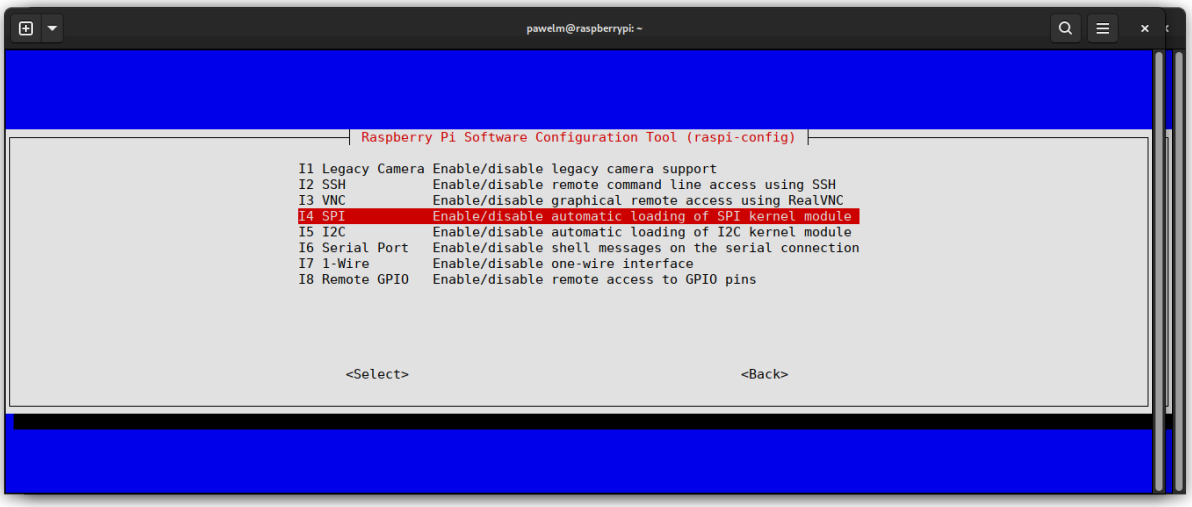
W celu komunikacji z wyświetlaczem na Raspberry Pi musi być włączony prootkół SPI, aby to zrobić wpisz w terminalu:

```
sudo raspi-config
```



Rys. 3. Raspi-config w terminalu po otworzeniu

Wejdź w 3 Interface Options i włącz SPI.



Rys. 4. Raspi-config - ustawienia interfejsów

Następnie zainstaluj wymagane biblioteki:

```
sudo apt-get install python-pip python-dev build-essential
sudo pip install RPi.GPIO
sudo apt-get install python-imaging opencv-python
sudo apt-get install git
git clone https://github.com/adafruit/Adafruit_Nokia_LCD.git
cd Adafruit_Nokia_LCD
sudo python setup.py install
```

```
sudo apt-get update
sudo apt-get install python3-opencv libleptonica-dev tesseract-ocr
tesseract-ocr-dev libtesseract-dev python3-pil tesseract-ocr-eng
tesseract-ocr-script-latn
```

```
pip install tesseract
pip install tesseract-ocr
```

Utwórz plik requirements.txt w którym umieścisz:

```
numpy>=1.18.5
Pillow>=7.1.2
requests>=2.23.0
```

Zainstaluj wymagane paczki:

```
pip install -r requirements.txt
```

4. Sterowanie diodami led

Kod sterujący diodami LED jest prosty. Diody są włączane i wyłączane w zależności od wyniku analizy numeru rejestracyjnego:

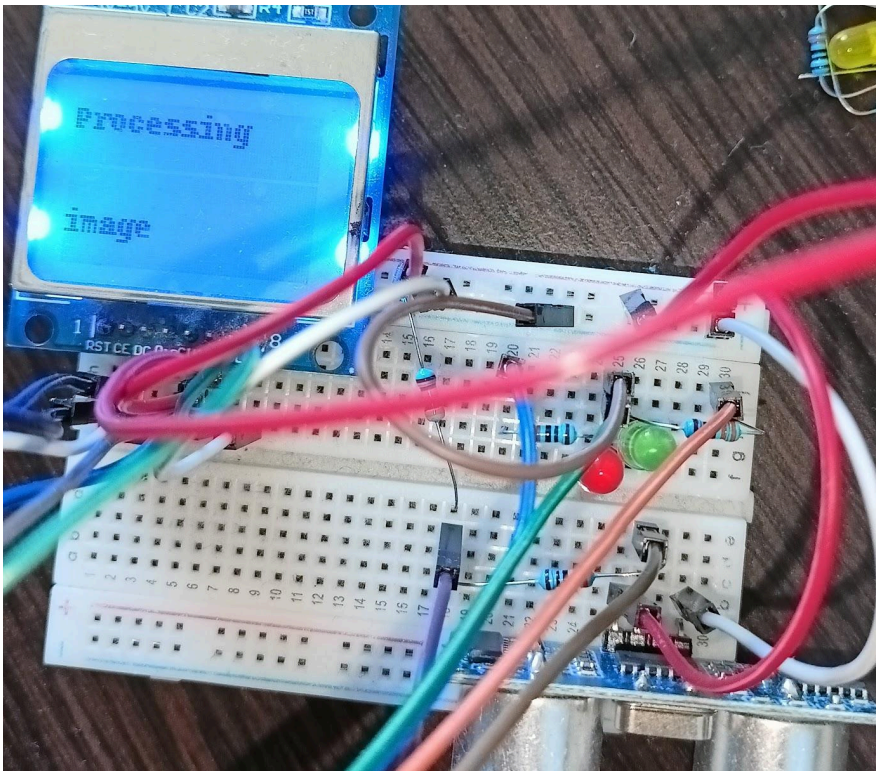
```
led_red=LED(17)
led_green=LED(27)

def turn_on_green_led():
    led_red.off()
    led_green.on()

def turn_on_red_led():
    led_red.on()
    led_green.off()

def turn_off_leds():
    led_red.off()
    led_green.off()
```

5. Wyświetlanie tekstu na wyświetlaczu

Wyświetlanie tekstu na wyświetlaczu LCD Nokia 5110 odbywa się przy użyciu biblioteki Adafruit_Nokia_LCD. Kod inicjalizuje wyświetlacz i wyświetla na nim tekst:

```
# Raspberry Pi hardware SPI config:
DC = 23
RST = 24
SPI_PORT = 0
SPI_DEVICE = 0

# Hardware SPI usage:
disp = LCD.PCD8544(DC, RST, spi=SPI.SpiDev(SPI_PORT, SPI_DEVICE,
max_speed_hz=4000000))

# Initialize library.
disp.begin(contrast=60)
disp.clear()
disp.display()

font = ImageFont.load_default()
```

```
    image_to_display = Image.new('1', (LCD.LCDWIDTH, LCD.LCDHEIGHT))
    draw = ImageDraw.Draw(image_to_display)
    draw.rectangle((0,0,LCD.LCDWIDTH,LCD.LCDHEIGHT), outline=255,
fill=255)

    def display_text(text):
        disp.clear()
        draw.rectangle((0,0,LCD.LCDWIDTH,LCD.LCDHEIGHT), outline=255,
fill=255)
        draw.text((0, 0), text, font=font, fill=0)
        disp.image(image_to_display)
        disp.display()

    def clear_display():
        disp.clear()
        disp.display()
```



Rys. 4. Fizyczne połączenie komponentów oraz działający ekran

Aby wyświetlić tekst należy najpierw wyczyścić to co znajduje się na wyświetlaczu, następnie wyczyścić bufor poprzez narysowanie pustego prostokąta, potem dopiero możemy narysować tekst.

6. Wykrycie samochodu przy pomocy czujnika odległości

Czujnik odległości HC-SR04 wykrywa obecność samochodu, mierząc odległość:

```
from gpiozero import DistanceSensor
sensor = DistanceSensor(trigger=25, echo=18)
```

W pętli głównej kodu sprawdzamy odległość:

```
while True:
       print(args.distance,sensor.distance)
       if (not args.distance) or sensor.distance<1.0:
           ret, image = cam.read()
           if not ret:
               break
           process_image(image, local=args.local)
       else:
           current_time = datetime.now()
           if last_update is None or current_time - last_update >
timedelta(hours=1):
               if is_remote_database_available():
                   print("Databases\nsync")
                   display_text("Databases\nsync")
                   update_local_database_with_allowed_plates()
                   update_remote_database_with_local_plates()
                   last_update = current_time
           time.sleep(1)
```

Zmienna argos.distance to nazwa zmiennej przechowującej informację czy program został uruchomiony z daną flagą w tym przypadku -D (distance meter) - uruchomienie wykrycia w przypadku gdy sensor wykryje jakiś obiekt przed sobą. W innym przypadku sprawdzamy ile czasu minęło od ostatniej aktualizacji baz danych, jeżeli jest to przynajmniej godzina to są one aktualizowane.

7. Pobranie zdjęcia z kamery

Najpierw warto sprawdzić na którym strumieniu wideo nadaje nasza kamera, w tym celu skorzystamy z komendy v4l2-ctl –list-devices, najpierw należy zainstalować narzędzie:

```
sudo apt-get install v4l-utils
```

```
pawelm@raspberrypi:~/Documents/ASK $ v4l2-ctl --list-devices
bcm2835-codec-decode (platform:bcm2835-codec):
	/dev/video10
	/dev/video11
	/dev/video12
	/dev/video18
	/dev/video31
	/dev/media2

bcm2835-isp (platform:bcm2835-isp):
	/dev/video13
	/dev/video14
	/dev/video15
	/dev/video16
	/dev/video20
	/dev/video21
	/dev/video22
	/dev/video23
	/dev/media0
	/dev/media1

Webcam C110: Webcam C110 (usb-3f980000.usb-1.2):
	/dev/video0
	/dev/video1
	/dev/media3
```

W naszym przypadku obraz uzyskamy na strumieniu zerowym:

```
  camera_index = 0


        cam = cv2.VideoCapture(camera_index)
        last_update = None
        while True:
            if (not args.distance) or sensor.distance<1.0:
                ret, image = cam.read()
                if not ret:
                    break
                process_image(image, local=args.local)
```

8. Wykrycie rejestracji

Aby usprawnić proces wykrycia rejestracji w postaci tekstu z obrazu skorzystamy z biblioteki opencv do wykrycia największego prostokąta na zdjęciu (czym

najprawdopodobniej jest nasza rejestracja), polepszy to nie tylko jakość ale i czas potrzebny do przekonwertowania obrazu na tekst.

```
def process_image(image, image_name=None,local=False):

    # image to text
    text=""
    Cropped = extract_plate(image)
    ...

def extract_plate(image):
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)  # Convert to gray
scale
    gray = cv2.bilateralFilter(gray, 13, 15, 15)
    edged = cv2.Canny(gray, 30, 200)  # Perform Edge detection

    contours = cv2.findContours(edged.copy(), cv2.RETR_TREE,
cv2.CHAIN_APPROX_SIMPLE)
    contours = imutils.grab_contours(contours)
    contours = sorted(contours, key=cv2.contourArea, reverse=True)[:10]
```

Obraz konwertuje się najpierw do skali szarości, przetwarzanie go jest wtedy bardziej efektywne i szybsze, ponieważ nie musimy pracować na trzech kanałach kolorów (B, G, R). Następnie filtr bilateralny jest używany do wygładzania obrazu przy zachowaniu krawędzi. Parametry (13, 15, 15) określają rozmiar filtra, oraz wartości dla przestrzeni kolorów i współrzędnych przestrzennych. Algorytm Canny'ego jest używany do wykrywania krawędzi w obrazie. Dwa parametry (30, 200) są progami dolnym i górnym dla wykrywania krawędzi., cv2.findContours znajduje wszystkie kontury w binarnym obrazie krawędzi, imutils.grab_contours upraszcza sposób uzyskiwania konturów z różnych wersji OpenCV. Kontury są sortowane według ich powierzchni, a następnie wybierane jest dziesięć największych.

```
 for c in contours:
       peri = cv2.arcLength(c, True)
       approx = cv2.approxPolyDP(c, 0.018 * peri, True)
       if len(approx) == 4:
           mask = np.zeros(gray.shape, np.uint8)
           new_image = cv2.drawContours(mask, [approx], 0, 255, -1)
           new_image = cv2.bitwise_and(image, image, mask=mask)
           (x, y) = np.where(mask ==
25/home/pawelm/Documents/ASK/cropped_images/test2_PY14291.jpg5)
           (topx, topy) = (np.min(x), np.min(y))
           (bottomx, bottomy) = (np.max(x), np.max(y))
```

```
            Cropped = gray[topx:bottomx+1, topy:bottomy+1]
            return Cropped

    return None
```

Obliczana jest długość obwodu konturu. Funkcja approxPolyDP upraszcza kontur przy użyciu współczynnika dokładności (0.018 * peri). Tablice rejestracyjne są zazwyczaj prostokątne, więc szukamy konturów z czterema wierzchołkami. Tworzymy maskę na podstawie wykrytego konturu. Używamy maski, aby przyciąć oryginalny obraz do obszaru tablicy rejestracyjnej, a następnie wyodrębniamy prostokąt, który obejmuje tablicę. Jeśli znaleziono kontur o czterech wierzchołkach, zwracamy wyodrębniony obraz tablicy rejestracyjnej.

Rys. 5. Obraz w skali szarości

Rys. 6. Obraz w skali szarości z nałożonym filtrem bilateralnym

Rys. 7. Obraz po wykryciu krawędzi

Rys. 8. Maska

Rys. 9. Obraz po nałożeniu maski

Rys. 10. Wycięta rejestracja

9. Optyczne rozpoznawanie znaków

```
if Cropped is not None:
    display_text(f"Processing\n\nimage")
```

```
        text = pytesseract.image_to_string(Cropped,
            config='--oem  3 --psm 6 -c
tessedit_char_whitelist=ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789')
        text = text.strip()
```

Sprawdzamy, czy zmienna Cropped zawiera obraz (czy wyodrębnienie tablicy rejestracyjnej zakończyło się sukcesem). Jeśli Cropped jest None, oznacza to, że tablica rejestracyjna nie została znaleziona, a dalsze przetwarzanie nie jest możliwe. Wywołujemy funkcję display_text, która wyświetla komunikat "Processing\n\nimage" na wyświetlaczu LCD, informując użytkownika o trwającym procesie przetwarzania obrazu. Używana jest biblioteka Tesseract OCR do rozpoznawania tekstu z wyodrębnionego obrazu tablicy rejestracyjnej:

- pytesseract.image_to_string(Cropped, config='...') wywołuje funkcję rozpoznawania tekstu na obrazie Cropped.
- --oem 3 oznacza użycie trybu OCR Engine Mode 3, który łączy mechanizmy rozpoznawania z Tesseract Legacy oraz LSTM.
- --psm 6 oznacza użycie trybu Page Segmentation Mode 6, który traktuje obraz jako jednolinijkowy blok tekstu.
- -c tessedit_char_whitelist=ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789 ogranicza zestaw znaków, które Tesseract ma rozpoznawać, do wielkich liter alfabetu angielskiego oraz cyfr, co zwiększa dokładność rozpoznawania numerów rejestracyjnych.

Ostatnia linia kodu usuwa wszelkie białe znaki (np. spacje, nowe linie) z początku i końca rozpoznanego tekstu. Funkcja strip() jest używana, aby uzyskać czysty tekst, który można porównać lub zapisać w bazie danych.

10. Rozpoznawanie rejestracji poprzez API

Aby skorzystać z API platerecognizer należy założyć darmowe konto,uzyskamy wtedy możliwość do 2500 zapytań, następnie z zakładki products/snapshot-cloud/ należy skopiować token przy pomocy którego uzyskamy dostęp.

```
try:
            response = requests.post(
                'https://api.platerecognizer.com/v1/plate-reader/',
                data=dict(regions=['pl','ua','de'],
config=json.dumps(dict(region="strict",mode="fast"))),
                files=dict(upload=image_jpg.tobytes()),
                headers={'Authorization': 'Token
9256a71f20eec8eafa039504889d94c07cd91a58'})
```

Ten fragment kodu tworzy i wysyła żądanie POST do zewnętrznego API rozpoznawania tablic rejestracyjnych (Plate Recognizer). Parametry żądania obejmują:

- URL API: 'https://api.platerecognizer.com/v1/plate-reader/'
- Dane:

    - regions=['pl','ua','de'] - lista regionów (krajów) do rozpoznania tablic rejestracyjnych.config=json.dumps(dict(region="strict",mode="fast")) - dodatkowe ustawienia konfiguracyjne w formacie JSON, gdzie region jest ustawiony na "strict" (ścisły) tryb rozpoznawania, a mode na "fast" (szybki) tryb przetwarzania.
- Pliki:
    - upload=image_jpg.tobytes() - obraz przesyłany jako plik binarny w formacie JPEG.
- Nagłówki:
    - 'Authorization': 'Token 9256a71f20eec8eafa039504889d94c07cd91a58' - nagłówek autoryzacyjny zawierający token dostępu do API.

```
text=response.json()['results'][0]['plate'].upper()
```

Po otrzymaniu odpowiedzi z API, kod próbuje odczytać rozpoznany numer rejestracyjny z pola results[0]['plate'] w formacie JSON, a następnie konwertuje go na wielkie litery za pomocą upper().

```
    except  (requests.RequestException, KeyError, IndexError) as e:
        print(f"API error: {e}\nProcessing locally")
        display_text(f"API error\nProcessing\nlocally")
        if Cropped is not None:
            text = pytesseract.image_to_string(Cropped,
config='--oem 3 --psm 6 -c
tessedit_char_whitelist=ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789')
            text = text.strip()
```

Jeśli API nie zadziałało poprawnie, kod próbuje przetworzyć obraz lokalnie za pomocą Tesseract OCR.

11. Postawienie lokalnej bazy danych

W celu postawienia bazy lokalnej należy zainstalować sqlite oraz bibliotekę pythonową:

```
sudo apt-get install sqlite3
sudo apt-get install python3-sqlite
```

Następujący kod łączy się z bazą lub ją tworzy w przypadku braku, następnie tworzy dwie tablice do przechowywania dozwolonych tablic oraz tych wykrytych przez system:

```
```

```
def initialize_local_database():
    conn = sqlite3.connect('license_plates.db')
    c = conn.cursor()
    c.execute('''CREATE TABLE IF NOT EXISTS plates
                 (id INTEGER PRIMARY KEY AUTOINCREMENT,
                  plate_number TEXT,
                  timestamp DATETIME DEFAULT CURRENT_TIMESTAMP,
                  image_path TEXT)''')
    c.execute('''CREATE TABLE IF NOT EXISTS allowed_plates
                   (id INTEGER PRIMARY KEY AUTOINCREMENT,
                    plate_number TEXT UNIQUE)''')
    conn.commit()
    conn.close()
```

Tabela plates przechowuje informacje o rozpoznanych rejestracjach, zaś tabela allowed_plates przechowuje rejestracje które są dozwolone.

12. Postawienie zdalnej bazy danych (Docker)

Do postawienia serwera wymagny będzi docker, w celu jego instalcji postępuj według oficjalnej instrukcji dla twoejgo systemu https://docs.docker.com/engine/install/.

Aby uruchomić kontener PostgreSQL, użyj poniższego polecenia:

```
docker run --name postgres-backup -e POSTGRES_PASSWORD=mysecretpassword
-e POSTGRES_DB=license_plates -p 5432:5432 -d postgres
```

Aby zatrzymać kontener PostgreSQL, użyj poniższego polecenia:

```
docker stop postgres-backup
```

Aby ponownie uruchomić kontener, użyj poniższego polecenia:

```
docker stop postgres-backup
```

Będziesz potrzebować biblioteki psycopg2, aby móc współdziałać z bazą danych PostgreSQL za pomocą skryptu w języku Python. Zainstaluj go za pomocą pip:

```
pip install psycopg2-binary
```

```
# Initialize PostgreSQL database
```

```python
def initialize_remote_database():
    try:
        conn = get_postgres_connection()
        cur = conn.cursor()
        cur.execute('''CREATE TABLE IF NOT EXISTS plates
                       (id SERIAL PRIMARY KEY,
                        plate_number TEXT,
                        timestamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
                        image_path TEXT)''')
        cur.execute('''CREATE TABLE IF NOT EXISTS allowed_plates
                       (id SERIAL PRIMARY KEY,
                        plate_number TEXT UNIQUE)''')
        conn.commit()
        cur.close()
        conn.close()
        return True
    except Exception as e:
        print(f"Error initializing remote database: {e}")
        return False

initialize_remote_database()


def is_remote_database_available():
    try:
        conn = get_postgres_connection()
        conn.close()
        return True
    except:
        return False
```

Lokalna baza danych wygląda bliźniaczo do zdalnej, musimy jedynie za każdym razem upewnić się że mamy z połączenie w przypadku próby komunikacji z serwerem.

13. wprowadzenie rejestracji do bazy danych

```python
# Save data to PostgreSQL database
def save_to_remote_database(plate_number, image_path):
    try:
        # print(f"Saving to remote: {plate_number} {image_path}")
```

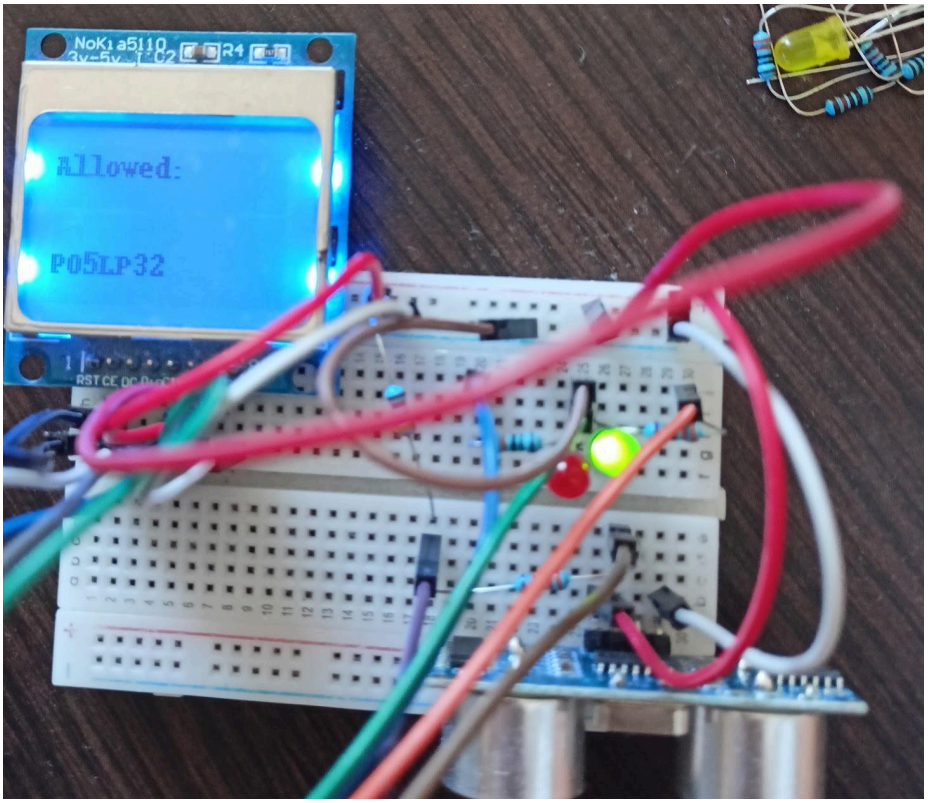
```
        conn = get_postgres_connection()
        cur = conn.cursor()
        cur.execute("INSERT INTO plates (plate_number, image_path)
VALUES (%s, %s)", (plate_number, image_path))
        conn.commit()
        cur.close()
        conn.close()
        return True
    except Exception as e:
        print(f"Error saving to remote database: {e}")
        return False
```

Wykryta rejestracja jest zapisywana do bazy danych wraz ze ścieżką do zdjęcia rejestracji w celu późniejszego pobrania i dalszej analizy w przypadku błędów. Kod dla bazy lokalnej wygląda analogicznie.

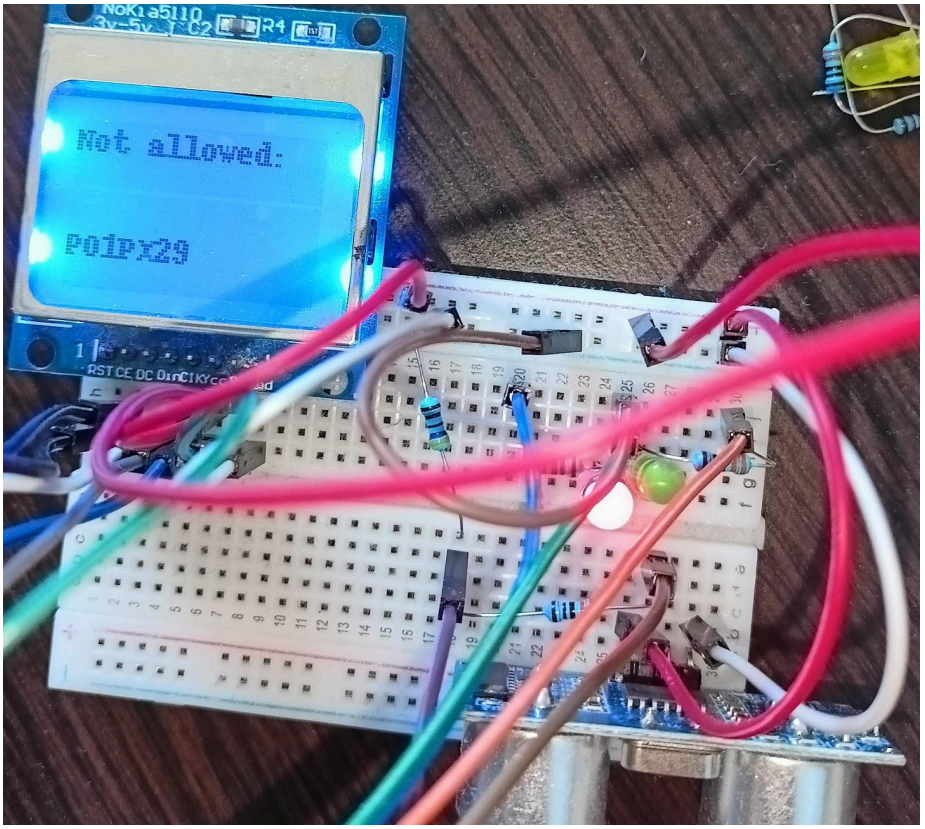
14. Weryfikacja rejestracji

```
# Check if plate is allowed
def is_plate_allowed(plate_number):
    result = None
    if is_remote_database_available():
        try:
            conn = get_postgres_connection()
            cur = conn.cursor()
            cur.execute("SELECT * FROM allowed_plates WHERE
plate_number = %s", (plate_number,))
            result = cur.fetchone()
            cur.close()
            conn.close()
        except:
            return False
    else:
        conn = sqlite3.connect('license_plates.db')
        cur = conn.cursor()
        cur.execute("SELECT * FROM allowed_plates WHERE plate_number =
?", (plate_number,))
        result = cur.fetchone()
        cur.close()
        conn.close()
    return result is not None
```

Jeżeli zdalna baza jest dostępna to pobieramy z niej dozwolone rejestracje, jeżeli nie, to robimy to z bazy lokalnej.



Rys. 11. Rejestracja dozwolona



Rys. 12. Rejestracja niedozwolona

15. Aktualizacja baz danych

```
# Update local database with allowed plates from remote database
def update_local_database_with_allowed_plates():
    conn = get_postgres_connection()
    cur = conn.cursor()
    cur.execute("SELECT plate_number FROM allowed_plates")
    allowed_plates = cur.fetchall()
    cur.close()
    conn.close()

    conn = sqlite3.connect('license_plates.db')
    c = conn.cursor()
    c.execute("DELETE FROM allowed_plates")  # Clear existing allowed
plates
    for plate in allowed_plates:
        c.execute("INSERT OR IGNORE INTO allowed_plates (plate_number)
VALUES (?)", (plate[0],))
    conn.commit()
    conn.close()

    # Update remote database with plates from local database
def update_remote_database_with_local_plates(last_update):
    conn_local = sqlite3.connect('license_plates.db')
    c_local = conn_local.cursor()

    # Convert last_update to a string if it's not None
    last_update_str = last_update.strftime('%Y-%m-%d %H:%M:%S') if
last_update else '1970-01-01 00:00:00'

    # Fetch plates with timestamp after the last update
    c_local.execute("SELECT plate_number, timestamp, image_path FROM
plates WHERE timestamp > ?", (last_update_str,))
    local_plates = c_local.fetchall()
    conn_local.close()

    conn_remote = get_postgres_connection()
    cur_remote = conn_remote.cursor()
    for plate in local_plates:
        plate_number, timestamp, image_path = plate
```

```
        # Convert timestamp string to datetime object
        timestamp = datetime.strptime(timestamp, '%Y-%m-%d %H:%M:%S')
        cur_remote.execute(
            "INSERT INTO plates (plate_number, timestamp, image_path)
VALUES (%s, %s, %s) ON CONFLICT DO NOTHING",
            (plate_number, timestamp, image_path)
        )
    conn_remote.commit()
    cur_remote.close()
    conn_remote.close()
...

    current_time = datetime.now()
    if last_update is None or current_time - last_update >
timedelta(hours=1):
        if is_remote_database_available():
            print("Databases\nsync")
            display_text("Databases\nsync")
            update_local_database_with_allowed_plates()
            update_remote_database_with_local_plates(last_update)
            last_update = current_time
    time.sleep(1)
```

W przypadku gdy aktualnie nie wykrywamy żadnego auta bazy danych są aktualizowane co godzinę. Wykryte rejestracje są wysyłane do zdalnej bazy, zaś rejestracje dozwolone są na nowo pobierane ze zdalnej bazy.

16. Jakość rozwiązań

Trafność rozwiązania lokalnego dla 20 zdjęć testowych to jedynie 27.14%, zaś dla API jest to 100%.

Lokalne rozwiązanie nie działa najlepiej, wynika to z dwóch rzeczy: Przedstawiona metoda nie zawsze poprawnie wykrywa rejestrację na obrazku gdyż nie każdy największy prostokąt jest rejestracją (czasem wykryty obiekt nie jest nawet prostokątnym blokiem), po drugie tesseract-ocr jest przystosowany do rozpoznawania tekstu na prostej nie zakrzywionej płaszczyźnie, czego nie można powiedzieć o rejestracjach zawartych w zbiorze testowym.

W przypadku implementacji projektu w świecie rzeczywistym kamera powinna być skierowana do auta pod stałym kątem a rejestracja powinna się znajdować mniej więcej w tym samym miejscu, dzięki czemu można by przyciąć i przeskalować obraz optymalnie dla modelu OCR. Innym podejściem jest użycie dotrenowanego modelu z rodziny np. YOLO oraz stworzenie własnego modelu do rozpoznawania tablic, niestety posiadana płytka Raspberry Pi 3B nie pozwoliła na instalację Pytorcha (oficjalnie wspierane są tylko architektury arm64). W przypadku najnowszego sprzętu warto zwrócić uwagę także na

Raspberry Pi AI Kit kompatybilny z Raspberry Pi 5, połączenie to pewnie uzyskałoby wynik w ułamku sekundy.

Warto by także zapisywać dozwolone rejestracje w programie a nie pobierać je z a każdym razem, przyspieszyłoby to proces i oszczędziło by to na transferze danych.

17. Źródła

- How to Stream USB Cameras in Python: A Beginner's Guide to OpenCV - e-con Systems
- Nokia 5110 LCD Display Setup For Raspberry Pi Guide - bluetin.io
- Nokia5110 Display Interfacing with Raspberry Pi
- GitHub - adafruit/Adafruit_Nokia_LCD: Python library for the using the Nokia 5110/3310 monochrome graphic LCD with a Raspberry Pi or Beaglebone Black.
- Use a USB Camera with Raspberry Pi for Beginners | by Robotamateur | Medium
- ANPR on Raspberry Pi | Plate Recognizer ALPR
- GitHub - madmaze/pytesseract: A Python wrapper for Google Tesseract
- License Plate Recognition using OpenCV Python | by ABHISHEK KUMAR GUPTA
- License Plate Recognition with OpenCV and Tesseract OCR - GeeksforGeeks
- Pomiar odległości z wykorzystaniem Raspberry Pi i Sklep Botland