Systemy wbudowane 2021/22
Automatyczne rozpoznawanie tablic rejestracyjnych przy użyciu platformy Raspberry Pi Zero W v1

Marcin Hawryluk, Maksymilian Wojnar

1 Cel projektu

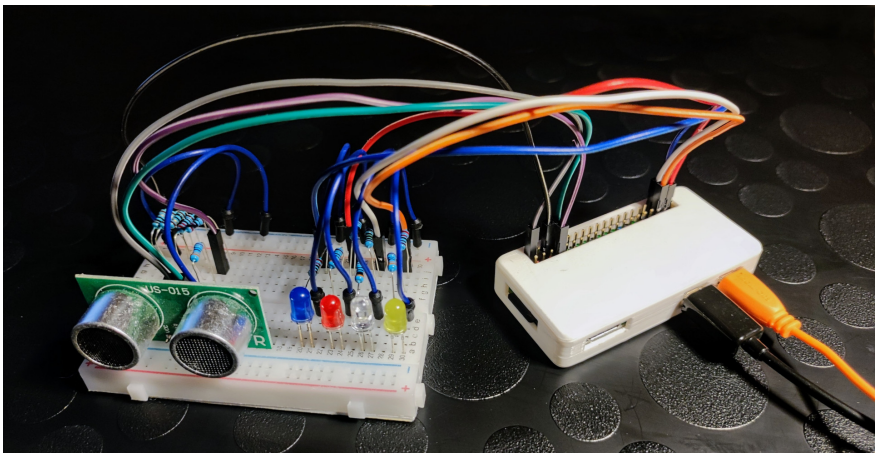
Projekt polegał na realizacji systemu detekcji tablic rejestracyjnych na obrazach zarejestrowanych przez kamerę wizyjną oraz rozpoznaniu numeru rejestracyjnego. Przykładowy obraz zarejestrowany przez kamerę znajduje się na Rysunku 1. Produkt mógłby zostać użyty jako część systemu kontroli pojazdów na drogach lub automatycznego szlabanu przy wjazdach na autostradę albo parking.

Rysunek 1: Przykładowy zarejestrowany obraz

2 Platforma sprzętowa

System zrealizowany został na platformie Raspberry Pi Zero W v1 [8] wyposażonej w jednordzeniowy procesor Broadcom BCM2835, taktowany częstotliwością 1GHz, oraz pamięć RAM o pojemności 512 MB. Do rejestracji obrazu posłużyła kamera internetowa, podłączona do płytki przez złącze microUSB. Płytka posiada również piny GPIO, które wykorzystaliśmy do podłączenia zestawu diod LED oraz czujnika odległości HC-SR04 [4]. Zdjęcie naszej płytki z podłączonymi urządzeniami znajduje się na Rysunku 2.

Platforma została podłączona do lokalnej sieci bezprzewodowej przez wbudowany moduł Wi-Fi, a następnie do komputera za pomocą tunelu SSH. Oprogramowanie rozwijaliśmy na komputerze, a następnie synchronizowaliśmy pliki za pomocą zdalnego repozytorium na GitHubie.



Rysunek 2: Wykorzystana platforma sprzętowa

3 Pierwsze podejścia do problemu

Rozpoczęliśmy pracę nad projektem od sprawdzenia różnych podejść do wykrywania tablic rejestracyjnych na obrazie. Kluczowymi aspektami naszego rozwiązania było dostosowanie rozwiązania do platformy wbudowanej o niskiej mocy, a jednocześnie wysoka skuteczność w rozpoznawaniu tablic rejestracyjnych.

Pierwszym podejściem było rozpoznawanie na obrazie kartki białego papieru na ciemnym tle. Nasze rozwiązanie wykorzystywało wstępne przetwarzanie obrazu, wyszukiwanie konturów za pomocą biblioteki OpenCV oraz wycinanie znalezionego obszaru z uwzględnieniem przechylenia kartki. Nasz program działał sprawnie, jednak posiadał wiele ograniczeń – wykrywał jedynie białe obiekty na ciemnym tle, jakiekolwiek odblaski lub białe fragmenty tła powodowały zakłócenia w wyszukiwaniu kartki, a ponadto wykorzystywany przez nas OCR miał problemy z odczytaniem tekstu z niedokładnie wyciętego obrazu.

Próbowaliśmy również samodzielnie zaimplementować segmentację obrazu metodą podziału i łączenia. Nasz kod dawał dobrej jakości wyniki, jednak przetwarzanie obrazu trwało dość długo, nawet na komputerze stacjonarnym. Dlatego zrezygnowaliśmy z tego podejścia na rzecz prostszego, opartego na progowaniu globalnym.

W pierwszej fazie projektu korzystaliśmy również z oprogramowania Motion [5] do udostępniania obrazu z kamery w sieci lokalnej oraz zapisywania zrzutów w określonym interwale czasowym. Odczytywaliśmy następnie te zrzuty z pamięci i traktowaliśmy je jako wejście naszego programu. Później okazało się jednak, iż nie potrzebujemy specjalnego oprogramowania do obsługi kamery, gdyż niosło to za sobą pewne konsekwencje: proces Motion mógł spowalniać naszą platformę sprzętową, dodatkowo zrzuty generowane przez oprogramowanie szybko zaśmiecały pamięć urządzenia i musieliśmy utrzymywać osobny skrypt do automatycznego czyszczenia folderów ze starych obrazów. Zdecydowaliśmy się zatem przejść na obsługę kamery z poziomu biblioteki OpenCV oraz napisać własny serwer, który oprócz obrazu na żywo, pozwolił nam na udostępnianie wielu obrazów z naniesionymi przekształceniami, co okazało się bardzo pomocne w debugowaniu naszego programu i posiada walory edukacyjne.

Kod poprzednich podejść, które nie weszły do użycia w finalnej wersji produktu, można znaleźć w folderze `experimental` w naszym repozytorium. Najnowsze, opracowane przez nas rozwiązanie, było inspirowane artykułem [6], jednak wymagało ono gruntownej modyfikacji podejścia. Artykuł ten został bowiem stworzony z myślą o tablicach amerykańskich, które różnią się od europejskich brakiem większej przerwy pomiędzy dwoma częściami numeru.

4 Zastosowane technologie

- **Python3** [12]
- **OpenCV** [9] + NumPy [10]
- **Tesseract-OCR** [11] + Pytesseract [7]
- **Flask** [2] + HTML, JavaScript, Bootstrap [1]
- **gpiozero** [3]

5 Przetwarzanie obrazu i detekcja tablic

5.1 Wstępne przetwarzanie i indeksacja

Oryginalny obraz (Rysunek 1) zamieniamy w pierwszej kolejności na czarno-biały (Rysunek 3), a następnie stosujemy morfologiczne domknięcie (Rysunek 4), aby rozmyć i połączyć małe elementy.

Rysunek 3: Konwersja do czerni i bieli

Rysunek 4: Morfologiczne domknięcie

Wykorzystujemy fakt, iż tablica rejestracyjna to jasny obszar z ciemnym obramowaniem. Stosujemy binaryzację algorytmem Otsu (Rysunek 5). Kolejnym krokiem jest zastosowanie morfologicznego otwarcia (Rysunek 6) w celu usunięcia szumów i małych fragmentów z obrazu. Jest to ważny krok naszego algorytmu, gdyż pozwala na znaczne zmniejszenie liczby komponentów po zastosowaniu indeksacji, a tym samym na znacznie przyspieszenie działania programu.

Rysunek 5: Binaryzacja

Rysunek 6: Morfologiczne otwarcie

Za pomocą operacji domknięcia morfologicznego łączymy białe grupy pikseli (Rysunek 7). Na koniec stosujemy segmentację obrazu (Rysunek 8) korzystając funkcji z biblioteki OpenCV *connectedComponentsWithStats*.

Rysunek 7: Morfologiczne domknięcie

Rysunek 8: Indeksacja

5.2 Kryterium detekcji obiektów

Ze zbioru otrzymanych elementów wybieramy te, które mogą być uznane za tablice rejestracyjne. Jako kryterium przyjęliśmy odpowiednią proporcję szerokości do wysokości obiektu (od 4:1 do 5:1) oraz odpowiednie pokrycie powierzchni obrazu (od 0.3% do 10%). Dobranie właściwych wartości było bardzo istotnym elementem projektu naszego systemu ze względu na konieczność minimalizacji liczby przetwarzanych obiektów, gdyż przetwarzanie każdego dodatkowego obiektu na płytce o małej mocy może zajmować dużo czasu. W celu wycięcia oraz określenia wymiarów obiektu przeprowadzamy proces detekcji konturów, wybieramy najdłuższy spośród odnalezionych, obliczamy współrzędne najmniejszego prostokąta obejmującego tenże kontur, a na koniec wykorzystujemy te współrzędne do wycięcia oraz obrócenia analizowanego obiektu.

Rysunek 9: Analizowany obraz z wykrytą tablicą

5.3 Przetwarzanie wyciętych tablic

Wykryte i wycięte z oryginalnego obrazu tablice poddajemy dalszemu przetwarzaniu, aby ułatwić rozpoznanie tekstu wykorzystywanemu przez nas systemowi OCR. Jest to ważny krok, gdyż rozpoznanie tekstu na naszej platformie sprzętowej może trwać nawet kilkanaście sekund. Dlatego tak istotne jest, aby wybierać z obrazu dobrych kandydatów na tablice oraz umożliwić systemowi odczytanie tekstu za pierwszą próbą. Surowy obraz wycięty z Rysunku 1 widzimy na Rysunku 10. Obraz przekształcamy do czerni i bieli (Rysunek 11), a następnie rozmywamy funkcją *medianBlur* (Rysunek 12), aby usunąć szum z obrazu.

Rysunek 10: Wycięta tablica

Rysunek 11: Czarno-białe

Rysunek 12: Rozmycie

W dalszej kolejności skalujemy wartości pikseli funkcją *convertScaleAbs* (Rysunek 13), aby ostatecznie poprawić kontrast tła i znaków operacją $255 \cdot \left(\frac{image}{255}\right)^3$ (Rysunek 14).

Rysunek 13: Skalowanie

Rysunek 14: Poprawa kontrastu

6 Rozpoznawanie tekstu

Do rozpoznania tekstu ze zdjęć wykorzystaliśmy bibliotekę Pytesseract, która bazuje na silniku Tesseract-OCR od Google. Aby poprawić otrzymywane wyniki, na wywoływaną funkcję nałożyliśmy następujące ograniczenia:

- tekst może składać się wyłącznie z małych liter angielskiego alfabetu, cyfr oraz znaku spacji,
- tekst powinien mieć dokładnie jedną linijkę,
- tekst nie może zaczynać się, ani kończyć białymi znakami,
- minimalna długość tekstu to 5 znaków.

7 Serwer HTTP

W celu wygodnego korzystania z serwisu, stworzyliśmy prosty serwer oferujący stronę prezentującą analizowany obraz, rezultat wstępnej obróbki, zindeksowane segmenty, odnalezione obiekty spełniające kryteria oraz wyodrębnioną tablicę rejestracyjną, jak również rozpoznany numer. Strona automatycznie odświeża wszystkie obrazy, aby można było śledzić działanie systemu na żywo.

Rysunek 15: Oferowana strona internetowa

8 Sterowanie procesem i sygnalizacja stanu

Wykorzystaliśmy podłączone diody do sygnalizowania aktualnego etapu pracy systemu. Żółta lampka wskazuje, iż program przeprowadza aktualnie przetwarzanie obrazu i detekcję tablic. Lampka w kolorze białym rozświetla się, kiedy narzędzie OCR rozpoznaje tekst na obrazie. Gdy etap ten dobiega końca, zapala się jedna z dwóch diod – czerwona, gdy nie znaleziono żadnego tekstu na obrazie albo niebieska, gdy operacja zakończyła się sukcesem.

Czujnik odległości został użyty jako element wywołujący poszczególne procedury systemu. Gdy dany obiekt znajdzie się w ustalonej odległości od czujnika, rozpoczyna się detekcja obiektów. Przy większym zbliżeniu obiektu, następuje uruchomienie procedury rozpoznawania tekstu.

9 Kod źródłowy i uruchomienie systemu

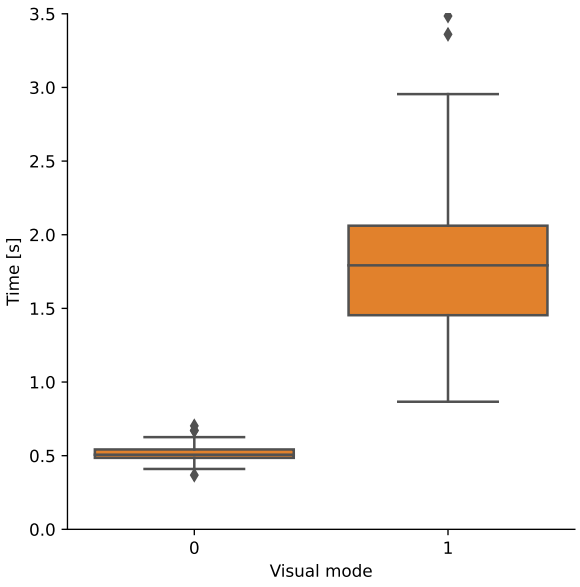
Kod naszego produktu umieściliśmy w repozytorium w serwisie GitHub: `https://github.com/m-wojnar/EmbeddedSystems`. Wraz z kodem dołączony jest prosty skrypt `run.sh` służący do łatwego uruchamiania systemu. Skrypt przyjmuje jako parametry następujące flagi:

- `--visual_mode` lub `-v` – uruchamia skrypt przetwarzający tablice i jednocześnie server HTTP do wizualizacji działania systemu,
- `--interactive_mode` lub `-i` – uruchamia system przetwarzający tablice z włączonym sterowaniem za pomocą czujnika odległości (domyślnie nasz produkt nieustannie próbuje wykrywać tablice na obrazie),
- `--background_mode` lub `-b` – uruchamia system w tle.

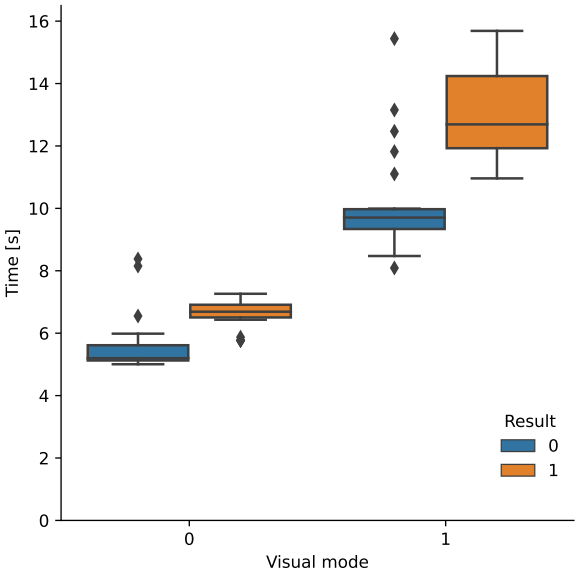
Flagi można dowolnie łączyć, aby uruchomić system wykrywania tablic w pożądanej konfiguracji.

10 Pomiary czasu

Dokonaliśmy pomiarów czasu lokalizowania tablic na obrazie oraz wykrywania tekstu (biblioteka pytesseract). Testy przeprowadziliśmy z włączonym oraz wyłączonym trybem wizualnym, gdyż ta opcja miała największy wpływ na responsywność naszego produktu. Otrzymaliśmy sumarycznie 137 pomiarów dla lokalizowania tablic (69 w trybie wizualnym i 68 bez tego trybu) i 73 pomiary dla rozpoznawania tekstu (33 w trybie wizualnym i 40 bez tego trybu). Pomiary dla rozpoznawania tekstu podzieliliśmy również ze względu na rezultat (29 dla wyniku pozytywnego i 44 dla wyniku negatywnego), gdyż miał on istotny wpływ na czas działania systemu. Wyniki pomiarów prezentujemy na Rysunkach 16 i 17.



Rysunek 16: Czas lokalizowania tablic na obrazie



Rysunek 17: Czas rozpoznawania tekstu

Nie przeprowadziliśmy niestety testów dokładności i rzetelności działania stworzonego przez nas produktu, gdyż nie posiadaliśmy odpowiednich warunków do sprawdzenia systemu w warunkach rzeczywistych. Forma zajęć oraz metodyka wytwarzania ograniczyły nas do testowania produktu jedynie na zdjęciach samochodów z widocznymi tablicami rejestracyjnymi.

11 Wnioski

- Wykorzystanie języka Python oraz bibliotek OpenCV i pytesseract pozwoliło nam na łatwe wykonanie zaawansowanego systemu przetwarzania obrazu w czasie rzeczywistym. Używane przez nas oprogramowanie posiada ogromny potencjał i daje możliwość stworzenia szybkiej, wydajnej oraz solidnej implementacji rozbudowanych systemów.
- Największym ograniczeniem naszego produktu była moc obliczeniowa użytej platformy sprzętowej. Raspberry Pi Zero W v1 jest małym i energooszczędnym komputerem jednopłytkowym, jednak jednordzeniowy procesor oraz pamięć operacyjna o wielkości 512 MB stanowiły istotną trudność w stworzeniu szybkiego i dokładnego systemu. Ze względu na małą moc płytki, optymalizacja naszego rozwiązania na urządzenia wbudowane była dla nas priorytetowa – zdecydowaliśmy się na uproszczenie przetwarzania obrazu, minimalizowaliśmy liczbę wykrytych obiektów do przetworzenia, zastosowaliśmy czujnik odległości w celu uruchamiania systemu wykrywania tablic tylko wtedy, gdy jest to konieczne oraz zalecamy wyłączenie trybu wizualnego w środowisku produkcyjnym.
- Użycie Raspberry Pi wraz z biblioteką gpiozero pozwoliło nam na łatwe i szybkie wprowadzenie do systemu dodatkowych elementów elektronicznych – diód LED oraz ultradźwiękowego czujnika odległości.
- Ciekawą obserwacją, wynikającą z dokonanych przez nas pomiarów, jest zauważenie różnicy czasu w rozpoznawaniu tekstu w zależności od rezultatu tej operacji. Biblioteka Tesseract-OCR kończy działanie nawet 20-30% szybciej, jeśli nie znajdzie żadnego tekstu na obrazie.
- Czas rozpoznawania tablic oraz tekstu jest silnie uzależniony od włączenia trybu wizualnego. Dodatkowe zapisy do pamięci masowej oraz wielokrotne zapytania GET stanowią istotny narzut wydajnościowy dla jednordzeniowego procesora oraz pamięci – czas odpowiedzi systemu, liczony od momentu aktywacji do zwrócenia rozpoznanego tekstu, może być nawet dwukrotnie dłuższy.
- Tryb wizualny, pomimo znacznego spowolnienia działania systemu, okazał się bardzo przydatny w trakcie tworzenia oprogramowania, gdyż pozwalał na łatwe zdiagnozowanie przyczyn problemów z lokalizowaniem tablic lub rozpoznawaniem tekstu. Ma też potencjał edukacyjny, gdyż w przystępny sposób prezentuje kolejne kroki przetwarzania obrazu i umożliwia śledzenie działania systemu na żywo.

12 Dalsze kierunki rozwoju

- W pierwszej kolejności warto byłoby przetestować system w warunkach rzeczywistych i ocenić jego przydatność w środowisku produkcyjnym. Mogłoby to być pomocne w podjęciu decyzji, czy warto inwestować w poprawianie skuteczności działania systemu, optymalizację produktu pod urządzenia wbudowane, a może przejść na komputer jednopłytkowy o większej mocy.
- Nasz produkt pozostawia szerokie pole do wykorzystania go jako element większych systemów – można zastanowić się nad zastosowaniem przenośnego i energooszczędnego systemu rozpoznawania tablic w ruchu drogowym, na parkingach, autostradach, promach czy terenach, na których wymagana jest kontrola nad poruszającymi się pojazdami.
- Aby wykorzystywać projekt w warunkach rzeczywistych, warto byłoby poprawić trafność rozpoznawania tekstu poprzez ulepszenie przetwarzania obrazu – system często mylił podobne znaki, na przykład literę „I” z liczbą „1”, liczbę „5” z literą „S” czy literę „Z” z liczbą „2”. Wartościowe byłoby również dostosowanie produktu do użycia w trudnych warunkach, na przykład przy sztucznym świetle po zmroku, w deszczu, w śniegu lub w pełnym słońcu.

- Jeśli zastosowanie produkcyjne wymaga szybkiej reakcji systemu, uzasadnione byłoby przejście na platformę o większej mocy (czas rzędu kilku sekund nie zawsze może być akceptowalny w pewnych środowiskach). Należy zachować ostrożność podczas stosowania radykalnych optymalizacji w naszym systemie, aby nie pogorszyć skuteczności rozpoznawania tablic.

Literatura

[1] Bootstrap is a powerful, feature-packed frontend toolkit. `https://getbootstrap.com/docs/5.2/getting-started/introduction/`.

[2] Flask – a simple framework for building complex web applications. `https://flask.palletsprojects.com/en/2.1.x/`.

[3] gpiozero – a simple interface to GPIO devices with Raspberry Pi. `https://gpiozero.readthedocs.io/en/stable/`.

[4] HC-SR04 – User's Manual. `https://botland.com.pl/index.php?controller=attachment&id_attachment=476`.

[5] Motion is a highly configurable program that monitors video signals from many types of cameras. `https://motion-project.github.io/motion_config.html`.

[6] OpenCV: Automatic License/Number Plate Recognition (ANPR) with Python. `https://pyimagesearch.com/2020/09/21/opencv-automatic-license-number-plate-recognition-anpr-with-python/`

[7] pytesseract – a python wrapper for Google's Tesseract-OCR. `https://pypi.org/project/pytesseract/`.

[8] Raspberry Pi Zero W. `https://www.raspberrypi.com/documentation/computers/raspberry-pi.html#raspberry-pi-zero-w`.

[9] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[10] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, Sept. 2020.

[11] A. Kay. Tesseract: An Open-Source Optical Character Recognition Engine. *Linux J.*, 2007(159):2, jul 2007.

[12] Python Core Team. *Python: A dynamic, open source programming language*. Python Software Foundation, 2019.