Rozpoznawanie owoców na taśmie

Przygotowali:

Dawid Kostrzewa

Magdalena Kozłowska

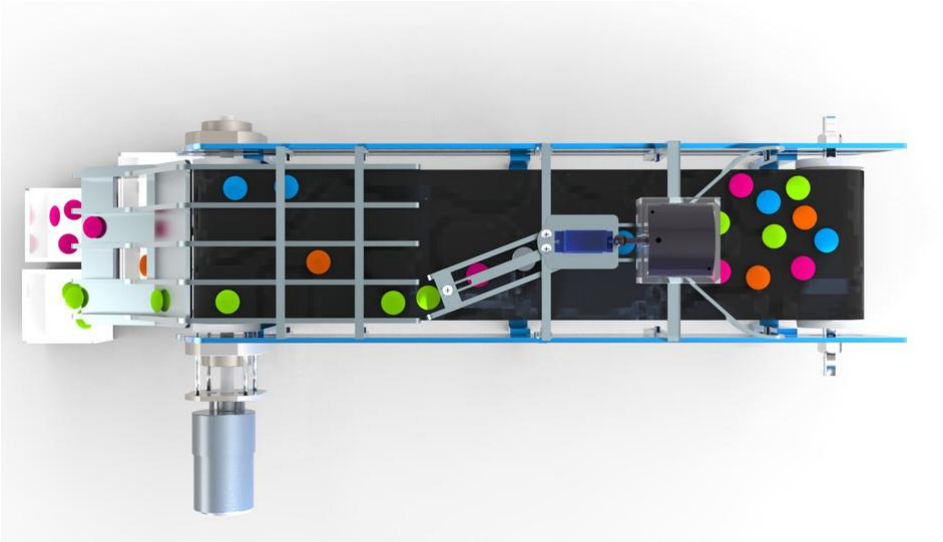
Julia Śnieg

Patryk Chodorowski

WSTĘP:

Nasz projekt ma na celu rozpoznawaniu pięciu różnych owoców: jabłek, cytryn, pomarańczy, kiwi i mango. W porównaniu do innych posiadamy w pełni przez nas zaprojektowaną i działającą manualnie bieżnie. W projekcie użyliśmy interfejsu z OpenCV do komunikacji między telefonem, a komputerem.

Inspirowaliśmy się przemysłowymi rozwiązaniami do sortowania produktów (np. Do odrzucania wadliwych owoców).



MATERIAŁY I METODY:

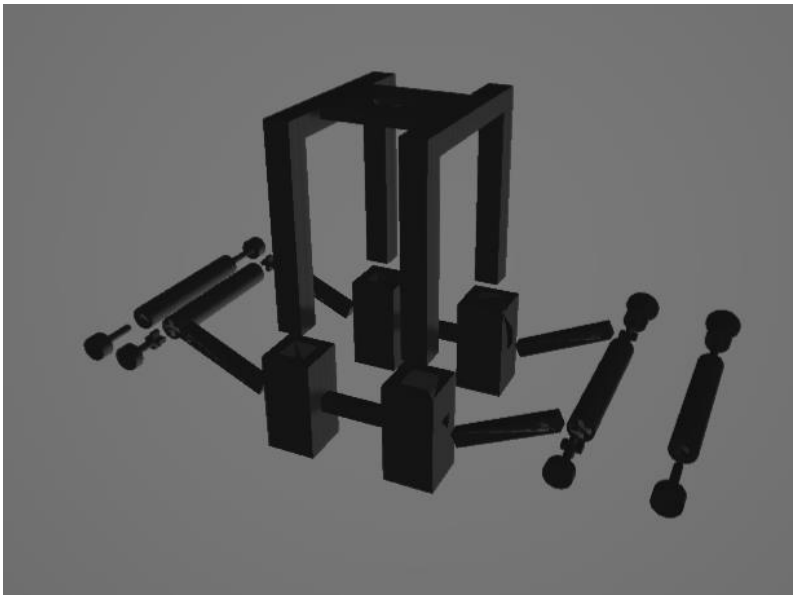
Stanowisko pomiarowe - Projekt taśmy:

Zaprezentowana przez nas taśma została zaprojektowana w programie do tworzenia projektów trójwymiarowych Blender i wydrukowana za pomocą drukarki 3D.

Taśma składa się z 2 wałków, na których rozciągnięty został materiał (matowy, jak najbardziej odróżniający się od owoców (biały lub czarny)).

Wałki zamocowane są na ramie, która pozwala na ruch taśmy w powietrzu, a na nóżkach znajduje się największy element służący do trzymania telefonu nad taśmą.

Za oświetlenie służy nam latarka z telefonu sprzężona z kamerą.



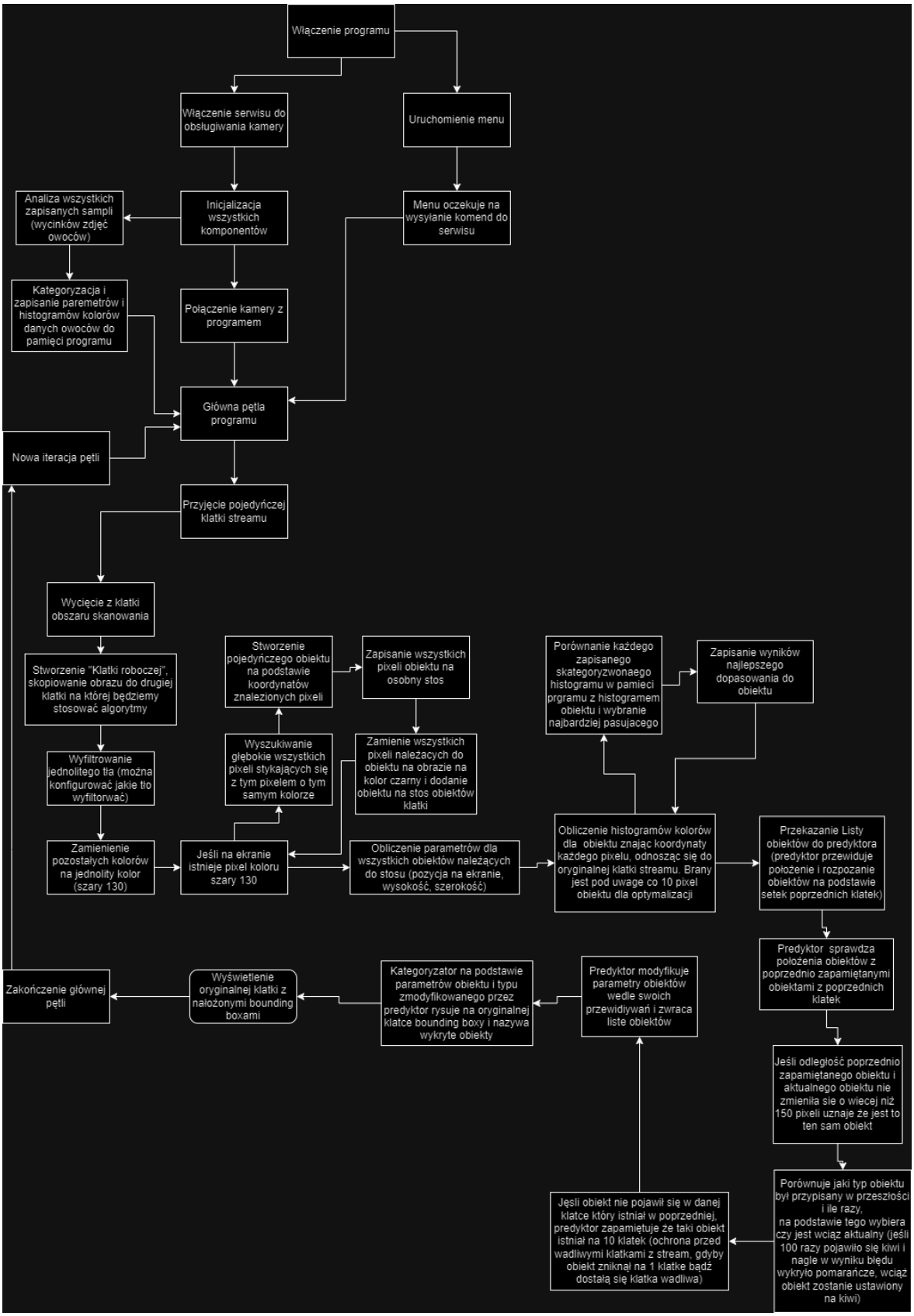
Aplikacja:

Do przekazywania obrazu jest użyta kamera telefonu o ustawionej rozdzielczości przesyłania 1280x720 w 30 FPS w programie IP WebCam.

Program odbiera kolejne klatki streamu za pomocą interfejsu OpenCV skompilowanego do C++.

Wszystkie zastosowane metody i algorytmy są w pełni napisane przez nas (niskopoziomowe działanie na pojedynczych pikselach), w niektórych miejscach implementowaliśmy własnoręcznie algorytmy typu erozja, wyfiltrowanie tła, wyszukiwanie głębokie, tworzenie histogramów na podstawie danych.

Schemat blokowy programu wykonany w programie DrawIO:



Użyte funkcje

Nasz projekt na przestrzeni czasu bardzo się zmieniał napotykając różne problemy z używanymi funkcjami. Na początku stworzyliśmy funkcje EdgeDetection(int treshold) która wykrywała róznice między pixelami a pixelami sąsiadującymi lecz zrezygnowaliśmy z niej z powodu dużej nieprzewidywalności czy konkretnych barwach, argument treshold pozwalał na wybranie jak duża musi być różnica by brać różnice miedzy pixelami jako wartą zarejestrowania i wykrycia tego jako krawędź.

Później próbowaliśmy użyć funkcji Canny z OpenCV do wykrywania krawędzi https://docs.opencv.org/3.4/da/d22/tutorial_py_canny.html

Lecz mimo lepszej pracy wciąż pozostawał problem że pewne barwy nachodzące na siebie nie były tak wyraźnie różne i przez to nie wykrywane przez funkcje wykrywania krawędzi.

Wszystkie te metody wykrywania krawędzi miały jedną bardzo ważną wadę. Trzeba było wypełnić obiekt posiadając jego obwód co okazało się bardzo kosztowne obliczeniowo lecz zdawało to egzamin, lecz wciąż problemy z niestabilnym wykrywaniem krawędzi istniały, gdzie jedna mała dziura w obwodzie obiektu nie pozwalała go wykryć jako obiekt, i też stąd obiekty często migotały w aplikacji, był niestabilne, a przy poruszaniu nimi, produkowało się rozmycie co jeszcze bardziej sprawiało trudność w wykryciu różnicy między pixelami, gdyż rozmycie je załagadza i była to po prostu nie efektywna forma wykrywania obiektu.

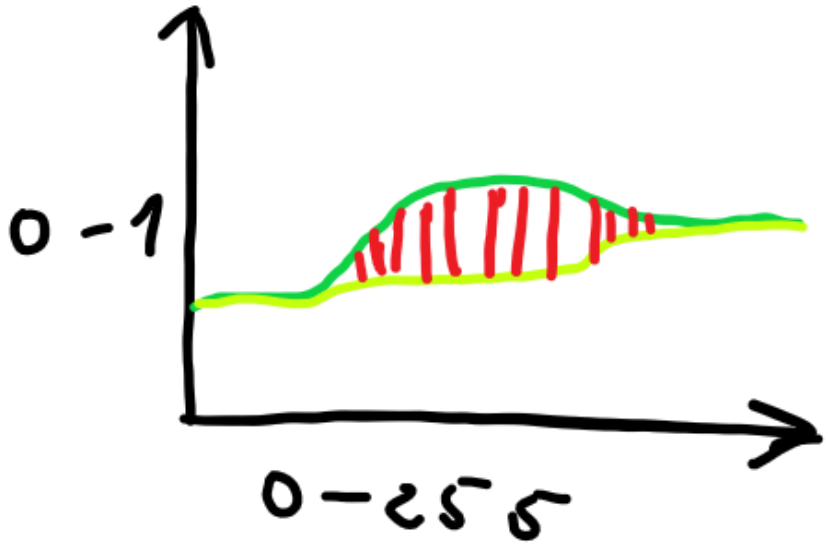
Ostatnie podejście i finalne jakie wybraliśmy do wykrywania obiektów było wyeliminowanie tła. Nasza funkcja do wyfiltrowania tła bada różnice między wartościami poszczególnych kanałów np. kolor 255 255 255 ma różnice 0, a kolor 255 254 253 ma różnice 255-254+255-253+254-253=4, jako argument przyjmuje wartość poniżej której różnica ta ma być traktowana jako tło.

Bardzo dużym plusem tego rozwiązania jest to że kolor biały i czarny nie zależnie od naświetlnia 255 255 255 czy 120 120 120 jest traktowany jako tło, więc aplikacja jest w stanie wyfiltrować tło z nierówno oświetlonego pasu gdzie krawędzie zazwyczaj są ciemniejsze. Nasz system wyfiltrowania nie ogranicza się tylko do tła białego i czarnego, bo można ustawić inne bazy kolorów od których ma się rozpoczynać różnica kanałów od siebie. Jedyny o co trzeba zadbać to, to żeby kolory tła nie nachodziły się z kolorami obiektów, ale to raczej w każdym rozwiązaniu trzeba zadbać o taki aspekt.

Kolejnym ogromnym plusem z tego rozwiązania jest to że obiekty pozostałe na ekranie już są w pełni wypełnione, co bardzo uprościło program obliczeniowo, a po zastosowaniu erozji obrazu, jesteśmy w stanie wydzielić stykające się obiekty, i według nas jest to najlepsze podejście do rozwiązania problemu wydzielenia obiektów, ma niemal 100% skuteczność w ich wydzielaniu, jedyny drobny minus jest taki że gdy obiekty odbijają światło białym blaskiem, robi małe dziury w obiektach, ale nie wpływa to na jakość kategoryzacji tego obiektu, a wręcz eliminuje odbicia od światła lampy zostawiając tylko naturalną barwę obiektu.

Metodę którą nasz program stosuje do kategoryzacji wykrytych obiektów jest porównanie histogramu wszystkich 3 kanałów barw wykrytego obiektu. Histogramy są znormalizowane więc nie wpływa na tą metodę to jak duży jest obiekt.

Przykładowe zwizualizowanie działania tej metody w porównywaniu dwóch histogramów kanału zielonego.



Gdzie na osi pionowej znajduje się znormalizowana wartość pola histogramu a na osi poziomej znajdują się kolejne wartości kanału barwy.

Czerwone kreski to różnica w poszczególnych pól histogramu, dane różnice są brane do kwadratu, by duże różnice bardzo wpływały na wynik. Te różnicę się sumuje a następnie jest ona odejmowana od pewnej wartości w wzorze matematycznym, tą samą operacje wykonujemy dla pozostałych dwóch kanałów i dostajemy wynik procentowy od 0 do 1.0 jak bardzo pasują do siebie te trzy histogramy. Na czym bazujemy naszą klasyfikacje.

Wykryty obiekt jest porównywany z kilkoma zapisanymi zdjęć np jabłek, więc dodanie do systemu jabłka w innym oświetleniu czy pozycji sprawi że też będzie wykrywany obiekt jako jabłko, wszystko to dzieje się automatycznie po dodaniu kolejnego zdjęcia do odpowiedniego folderu, nie trzeba nic zmieniać w kodzie.

Finalnym krokiem jest przekazanie naszych danych do predyktora, który przewiduje na podstawie poprzednich klatek, pozycji i klasyfikacji obiektów, czym finalnie predyktor sądzi że jest obiekt, by uniknąć problemów związanych z chwilową błędną klasyfikacją czy niestabilną transmisją obrazu z kamery. Predyktor jest w stanie zapamiętać kiedyś wykryte obiekty więc nawet jeśli znikną na jakiś czas z obrazu, wciąż o nich pamięta zwiększając dokładność klasyfikacji. Predyktor też zlicza wszystkie obiekty i jego funkcje bardzo pomagają w dokładnym zliczaniu obiektów. Paremetrami jakimi możemy sterować w perdyktorze głównie są długości jak długo coś zapamiętują, bądź jak łatwo go przekonać do zmiany zdania.

Na samym końcu używamy własnych funkcji do rysowania bounding boxów które przyjmują obiekt, a funkcja korzysta tylko z pozycji obiektu i jego rozmiarów i na tej podstawie rysuje wspomniany bounding box.

Bardzo dużo kodu naszego programu jest nie używana(~30%), bo pozostała z poprzednich iteracji programu takie jak wykrywanie krawędzi, anty-aliasing, blobowanie pixeli etc.

Wszystkie funkcje programu są napisane autorsko prócz interfejsu Kamera-Komputer-Monitor który pochodzi z OpenCV

3. WYNIKI:

Nasz program pozwala na prawidłowe rozpoznanie owoców z niemal 100% dokładnością po poprawnej konfiguracji i spreparowaniu potrzebnych programowi zdjęć.

Zalety:

-Szybkość

-Łatwość pierwszego uruchomienia (program nie wymaga szkolenia sieci przez długi czas)

-Bardzo duża adaptacyjność i możliwość wprowadzenia zmian w locie (wystarczy podmienić zdjęcia i nazwy by zamiast owoców np. wykrywał warzywa czy też inne obiekty)

-Dużo narzędzi konfigurowania (tryb kalibracji, debugowania, wiele ustawień zmienianych w menu programu)

Wady:

-Bazowanie jedynie na barwach, program nie rozróżni marchewki od pomarańczy

-Potrzeba manualnego dodawania zdjęć do bazy danych programu, co bywa monotonne

-W przypadku problemów z klasyfikacją ciężko jest stwierdzić co trzeba poprawić

Zajętości pamięci komputera: Kilkanaście MB RAM, Kilkanaście MB na dysku głównie zapisanych zdjęć obiektów do których odnosi się program.

Czasu działania: ~3-15ms na przetworzenie klatki, przesył z telefonu do programu zajmuje kolejne ~15 ms

Link do github:

https://github.com/AllirWasTaken/psiProjekt

Bibliografia:

https://learnopencv.com/image-filtering-using-convolution-in-opencv/

https://docs.opencv.org/3.4/da/d22/tutorial_py_canny.html

https://docs.opencv.org/3.4/dd/d43/tutorial_py_video_display.html

https://docs.opencv.org/4.x/db/deb/tutorial_display_image.html