

**WYDZIAŁ  
ELEKTROTECHNIKI  
I INFORMATYKI  
POLITECHNIKI RZESZOWSKIEJ**

**Rafał Ileczko**

Rozszerzona rzeczywistość we wspomaganiu obsługi urządzeń

**Praca dyplomowa magisterska**

Opiekun pracy:  
dr inż. Mariusz Oszust

Rzeszów, 2019



# Spis treści

<b>Wykaz symboli, oznaczeń i skrótów</b>	<b>6</b>
<b>1. Wstęp</b>	<b>7</b>
<b>2. Wprowadzenie teoretyczne</b>	<b>10</b>
2.1. Wirtualna i Rozszerzona Rzeczywistość	10
2.2. Dostępne narzędzia dedykowane AR	12
2.3. Zastosowanie AR w przemyśle	13
2.4. Sposoby zastosowań AR	15
2.5. Algorytmy detekcji i deskrypcji	17
2.5.1. ORB	17
2.5.2. SIFT i SURF	19
2.6. Dopasowanie cech	21
2.6.1. Brute-Force Matcher	21
2.6.2. FLANN based Matcher	21
2.7. Model kamery otworkowej	22
2.8. Homografia	24
2.9. Zastosowane technologie	26
2.9.1. OpenCV	26
2.9.2. Arduino	26
2.9.3. MQTT	27
2.9.4. Języki programowania	27
<b>3. Realizacja projektu</b>	<b>29</b>
3.1. Przyjęte założenia	29
3.2. Architektura rozwiązania	30
3.3. Maszyna	31
3.4. System AR	35
3.5. Komunikacja	46
3.6. Tryby działania	47
<b>4. Testy</b>	<b>48</b>
4.1. Test odległości	50
4.2. Test kąta patrzenia	50
4.3. Test różnych warunków oświetleniowych	51

4.4.	Test różnego natężenia oświetlenia . . . . .	52
4.5.	Test obrotu kamery względem urządzenia . . . . .	52
4.6.	Test częściowo widocznego urządzenia . . . . .	53
4.7.	Podsumowanie i wnioski z testów . . . . .	54
<b>5.</b>	<b>Podsumowanie i wnioski końcowe . . . . .</b>	<b>56</b>
	<b>Literatura . . . . .</b>	<b>57</b>



## Wykaz symboli, oznaczeń i skrótów

**AI** Artificial Intelligence

**AR** Augmented Reality

**DL** Deep Learning

**FLANN** Fast Library for Approximate Nearest Neighbors

**IoT** Internet of Things

**JSON** JavaScript Object Notation

**MQTT** Message Queue Telemetry Transport

**ORB** Oriented FAST and Rotated BRIEF

**SIFT** Scale Invariant Feature Transform

**SURF** Speeded-Up Robust Features

**VR** Virtual Reality

## 1. Wstęp

Panującym obecnie trendem jest automatyzacja - słowo to występuje wielokrotnie na każdej konferencji. Dotyczy każdego rodzaju przemysłu i rynku usług. Jej efekty już od dawna odczuwamy także w życiu codziennym. Niczym dziwnym jest widok osoby mówiącej do telefonu, aby ten podał drogę do miejsca docelowego, wyszukał wykwintną restaurację, czy nawet opowiedział dowcip. Przedsiębiorstwa całego świata inwestują w działy automatyzacji produkcji zakupując kolejne manipulatory, czy też rozwoju, próbujące wymyślić nowe sposoby na wyeliminowania człowieka z procesu. W chwili obecnej to właśnie homo sapiens najsłabszym ogniwem. Maszyny, czy też oprogramowanie jest w stanie szybciej działać, wykonywać obliczenia będąc przy tym nieporównywalnie dokładniejszym. Jednocześnie urządzenia nie biorą urlopów, ich wydajność nie spada, nie wymagają motywacji, a także - co najważniejsze mogą pracować 24h na dobę. Coraz częściej pojawiają się gniazda produkcyjne oraz magazyny w pełni autonomiczne, wymagające jedynie konserwacji.

Taka sytuacja ma miejsce nie tylko na obszarach produkcyjnych. W niektórych zawodach specjalizowanych coraz częściej modele sztucznej inteligencji mają lepsze osiągi niż specjaliści z dużym doświadczeniem. Tak jest na przykład w przypadku diagnozy niektórych schorzeń na podstawie danych o pacjencie oraz statystykach chorób [1]. W Estonii uruchomiono pierwszy na świecie sąd, gdzie wyroki dotyczą drobnych przestępstw wydaje model uczenia maszynowego [2]. W chwili obecnej takie narzędzia służą jako doradcy dla ludzi. Natomiast poprzez rozwój działu sztucznej inteligencji na świecie, będą pełniły coraz większą rolę. Można więc przewidywać, że w przeciągu kilku lat AI wyprze także specjalistów.

Jeśli obecne trendy się nie zmienią, w nowoczesnych procesach produkcyjnych będzie popyt na dwa typy pracowników: inżynierów tworzących nowe rozwiązania, oraz konserwatorów obecnych. Tutaj pole do automatyzacji jest mniejsze. Nie stworzyliśmy jeszcze sztucznej inteligencji tworzącej koncepcje nowych maszyn w sposób pragmatyczny, to jest uwzględniającej dokładną specyfikację. Podobnie ciężko zastąpić konserwatorów, którzy do pracy potrzebują zarówno zwinnych rąk, szeregu narzędzi jak i pomysłów w diagnozie usterek. Mają oni jednak pewne zasadnicze ograniczenia. Posiadają zaledwie dwie ręce, oraz parę oczu mogącą patrzeć się w jedną stronę. Projekt wykonany w ramach niniejszej pracy stara się w pewnym stopniu znieść te ogranicze-

nia, aby przyszli pracownicy mając do wykonania pewną pracę mogli skupić się na swoim zadaniu znacznie redukując liczbę obiektów potencjalnie go rozpraszających i ograniczających.

Odpowiednim rozwiązańem powyższych problemów wydaje się być Rozszerzona Rzeczywistość (ang. Augmented Reality - AR). Metodologia ta polega na nałożeniu na obraz rzeczywisty widziany przez użytkownika elementów wygenerowanych przez program. Często spotykane zastosowania AR to filtry w aplikacjach typu Messenger czy Instagram. Pozwalają one wykonywać zdjęcia czy filmy ze zmodyfikowanym obrazem. Odpowiednie algorytmy poszukują ludzkiej twarzy na kadrze, a następnie modyfikują, dodając na przykład włosy, bądź pogrubiając twarz. Innym częstym zastosowaniem są gry AR, gdzie najpopularniejszym przykładem jest Pokemon GO. Aplikacja wykorzystuje lokalizację użytkownika; będąc w miejscu gdzie występują pokemony można uruchomić aplikację, a na ekranie gracz zobaczy model 3D stworka z którym można nawiązać interakcję - walczyć, bądź go złapać. Rozszerzona rzeczywistość znajduje zastosowanie również w innych dziedzinach - firma Ikea udostępniła aplikację, która pozwala wizualizować jak będzie prezentował się dany produkt w otoczeniu użytkownika. Na potrzeby AR technologii powstały również specjalnie okulary, które pokazują wygenerowany obraz bezpośrednio przed oczami użytkownika, zamiast na ekranie.

Okulary AR są narzędziem nadającym się idealnie do celów konserwacji. Poprzez nałożenie na widoczny, rzeczywisty, obraz dodatkowych elementów możliwe jest bez oderwania wzorku od obiektu zainteresowania wykonać wiele czynności. Przykładowo dzięki będącemu zawsze w polu widzenia wskazaniu miernika, można testować obwody o wiele szybciej. Dzięki kamerze przebieg procesów można nagrywać, bądź współpracować zdalnie z inną osobą, gdzie obie widzą ten sam obraz. Przykłady można mnożyć, a dodatkową zaletą jest fakt, że nikt nic w ten sposób nie traci - wciąż możliwe jest wykonywanie wszystkich czynności w sposób tradycyjny.

Autor za wkład własny uważa:

- Stworzenie oprogramowania pozwalające na spełnienie wizji opisanej w powyższym akapicie. Docelowo ma ono wspomagać obsługę i konserwację urządzeń dzięki metodom rozszerzonej rzeczywistości. Zakłada się, że dostępne maszyny, które mają być obsługiwane posiadają moduły umożliwiające połączenie się z lokalną siecią LAN, oraz mogą za jej pomocą wysyłać swoje parametry protokołem MQTT

- Stworzenie na potrzeby prezentacji pracy prostego urządzenia imitującego prawdziwe, dużo bardziej skomplikowane. A którego celem jest przesyłanie prostych wartości w postaci ciągu znaków.

## **2. Wprowadzenie teoretyczne**

### **2.1. Wirtualna i Rozszerzona Rzeczywistość**

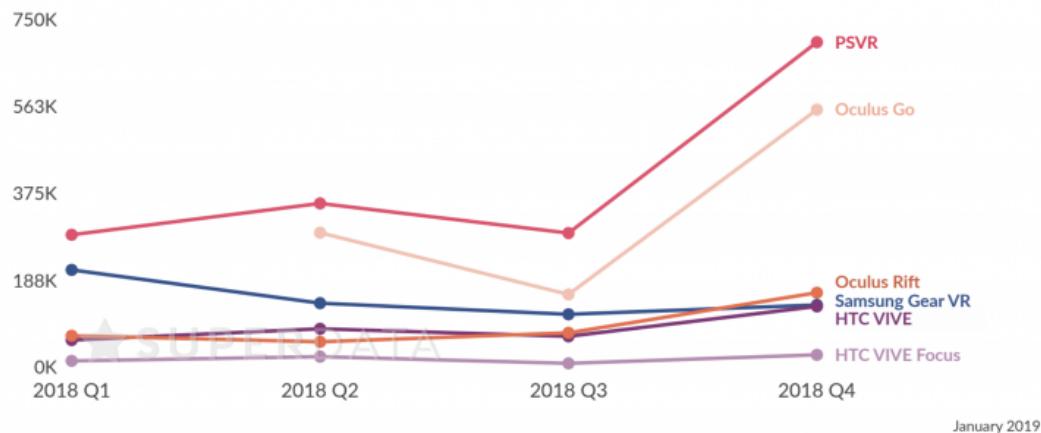
W roku 2019 fotorealistyczna grafika komputerowa nie robi już na nikim większego wrażenia. Dzięki technikom Motion Capture generowane komputerowo postacie poruszają się zupełnie naturalnie. Specjalne algorytmy są w stanie symulować setki tysięcy włosów w czasie rzeczywistym [3]. Pierwsze gry wykorzystujące RayTracing są już na rynku gier komputerowych [4], a badania nad dedykowanym sprzętem już się rozpoczęły [5]. Oznacza to, że obecna technologia zbliża się do granicy fotorealizmu, jaki jesteśmy w stanie osiągnąć. Naturalnym rozwiążaniem wydaje się więc wyjście ze środowiska płaskich ekranów na rzecz bardziej naturalnych doznań.

Pierwszym znaczącym krokiem w tym kierunku była publiczna zbiórka pieńczy na serwisie Kickstarter w 2012 roku na projekt Oculus Rift. Twórcy zebraли w ten sposób prawie 2,5 miliona dolarów amerykańskich na rozwój. Efektem ubocznym tego wydarzenia była znaczna popularyzacja terminu Wirtualna Rzeczywistość (ang. Virtual Reality - VR) wśród ludzi na świecie. Obecnie, a więc 7 lat od czasu zbiórki, do dyspozycji graczy dostępne jest 8 zestawów gogli VR, a liczba ta jeszcze się powiększy [6]. Wirtualna rzeczywistość jest technologią pozwalającą na prezentowanie użytkownikowi sztucznej, generowanej rzeczywistości. Dzięki specjalnym goglom, kontrolerom oraz technologii ich śledzenia, interakcja z otoczeniem dokładnie imituje rzeczywistą. Oznacza to, że ruch głowy użytkownika wywołuje identyczny ruch kamery w symulacji. Takie rozwiązanie pozwala na dużo większą interakcję użytkownika z otoczeniem niż jak ma to miejsce w symulacjach komputerowych wyświetlanych na typowym ekranie. W roku 2018 zyski firm z tytułu samego sprzętu VR przekroczyły 3,6 miliarda dolarów amerykańskich, co jest trzydziestoprocentowym wzrostem względem roku poprzedniego [7]. Zyski ze sprzedaży sprzętu VR w samym 2018 roku, z podziałem na kwartały prezentuje rysunek 2.1.

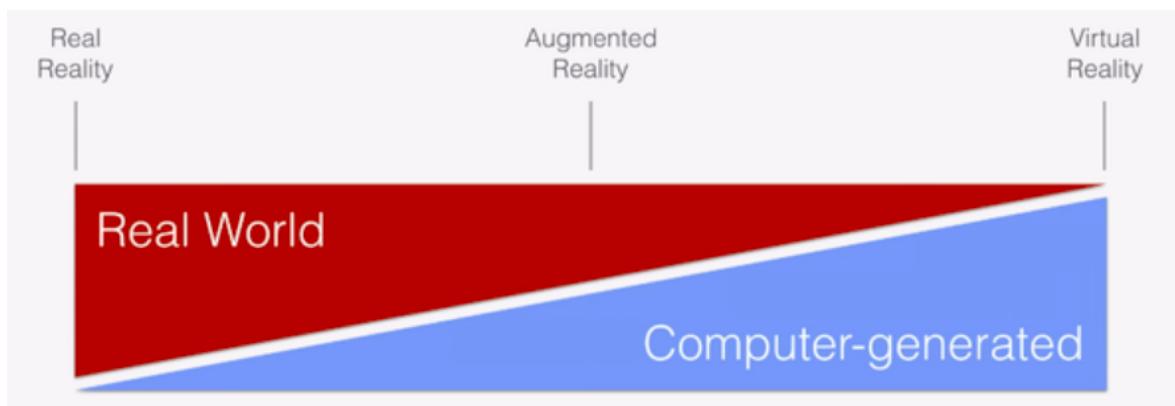
Innym podejściem do tematu modyfikacji rzeczywistości jest rzeczywistość rozszerzona (ang. Augmented Reality). Technologia ta kompromisem pomiędzy rzeczywistością faktyczną, a wirtualną. Leży na środku osi, którą nazwać można „spektrum rzeczywistości”. Przedstawia ono jaka część danego rozwiązania jest rzeczywista, a jaka wirtualna (rys. 2.2).

## Virtual Reality Headsets

Sell-through shipments: Q1 2018-Q4 2018  
Thousands, worldwide



Rysunek 2.1: Zyski ze sprzedaży sprzętu VR w 2018 roku [8]



Rysunek 2.2: Spektrum rzeczywistości [9]

Jak widać na rysunku, AR znajduje się pomiędzy światem prawdziwym, a generowanym komputerowo. Jego celem nie jest wygenerowanie zupełnie nowej rzeczywistości jak ma to miejsce w przypadku VR, a jedynie stworzenie "wartości dodanej" do otaczającego świata. Przyjętą definicję AR zaprezentował Ronald Azuma. Rozszerzona rzeczywistość powinna objawiać się na trzech płaszczyznach [11]:

- Łączy świat rzeczywisty z wirtualnym
- Interakcja występuje w czasie rzeczywistym
- Rejestruje świat w trzech wymiarach

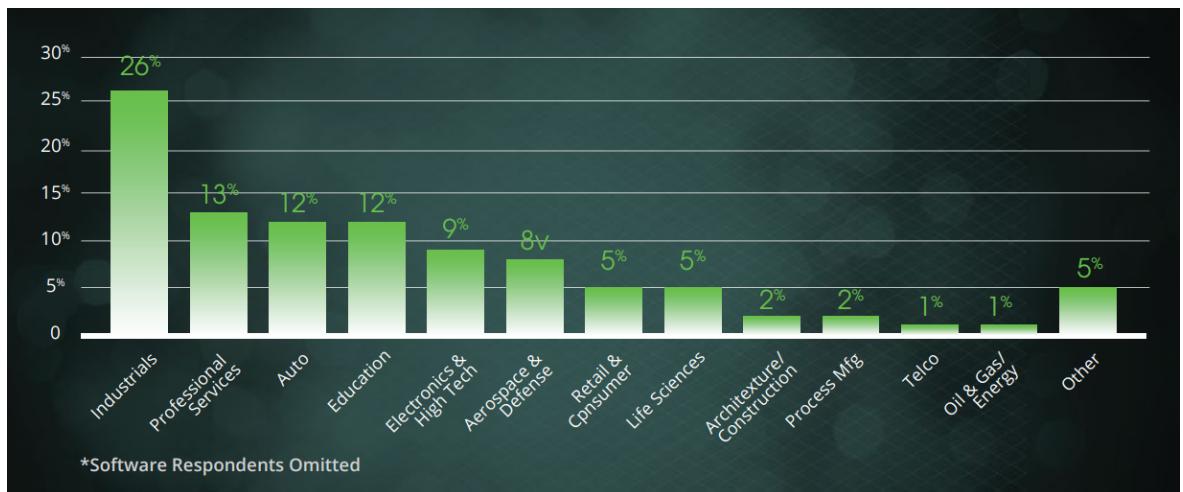
W przeciwnieństwie do wirtualnej rzeczywistości, rozszerzenie może objawiać się na wielu typach urządzeń. Najbardziej powszechnym jest smartfon, jednak odpowiedniość zapewniają okulary. Pozwalają one potencjalnie na zupełnie bezobsługowe ulepszenie rzeczywistości. Jednak zasadniczą zaletą technologii AR jest wykorzystanie obrazu do tworzenia modyfikacji. Dzięki wykorzystaniu kamery, oprogramowanie jest w stanie przetworzyć obraz, wydobyć z niego pewne informacje, a następnie wykorzystać do stworzenia rozszerzeń obrazu. Dobrym przykładem takiej funkcjonalności jest wspomniana we wstępnie aplikacja IKEI na smartfony. Pozwala ona, wykorzystując kamerę telefonu, wyznaczyć płaszczyznę podłogi pokoju, a następnie dzięki odpowiednim obliczeniom wyrenderować model 3D produktu we właściwej dla pomieszczenia skali i orientacji oraz zapamiętać położenie względem pomieszczenia. Jest to przydatne przy projektowaniu wnętrza pokoju, a możliwość poruszania się w takiej przestrzeni dopełnia wrażenia.

AR ze względu na swoją specyfikę dobrze odnajduje się w przemyśle. Raport firmy PTC „State of Industrial Augmented Reality 2019” [12] twierdzi, że przemysł adaptuje najczęściej projektów AR, dzięki czemu wydajność pracowników gwałtownie wzrasta. Firmy oszczędzają również na szkoleniach, dzięki interaktywności rozwiązań.

## 2.2. Dostępne narzędzia dedykowane AR

### Apple ARKit

Narzędzie to stworzone przez Apple pozwala tworzyć aplikacje AR na urządzenia iPhone oraz iPad. Z tego też powodu miejsce ich zastosowań jest bardzo ograniczone - nie ma możliwości zaimplementowania swojego narzędzia na żadne z dostępnych okularów AR.



Rysunek 2.3: Procentowy udział rynku w adaptacji projektów AR

### Google ARCore

Kolejna biblioteka stworzona przez giganta branży IT. Jej wykorzystanie jest możliwe w systemach Android oraz iOS i w silnikach Unity oraz Unreal. Z tego też powodu doskonale nadaje się do tworzenia aplikacji AR na urządzenia mobilne.

### Vuforia

Jest rozbudowanym rozszerzeniem do silnika Unity. Pozwala rozpoznawać i śledzić płaskie oraz proste obiekty 3D w czasie rzeczywistym. Określa ich pozycję oraz orientację w przestrzeni, pozwala w prosty sposób nakładać na obraz obiekty 3D.

### Wikitude

Jest kompletnym narzędziem do tworzenia rozwiązań AR. Zawiera w sobie rozpoznawanie i śledzenie obiektów, geolokalizację, SDK pozwala tworzyć na urządzeniach mobilnych, oraz w Unity, Cordova, Titanium i Xamarin.

Istnieje również wiele innych mniej rozbudowanych rozwiązań. Autor, uznając jednak, że żadne ze wspomnianych nie spełnia jego oczekiwania, postanowił stworzyć własne.

## 2.3. Zastosowanie AR w przemyśle

Konserwacja jest procesem periodycznym, zajmującym mniej więcej tyle samo czasu, przy czym czas ten jest indywidualny dla każdej maszyny. Diagnostyka i naprawy występują rzadko, natomiast są czasochłonne, wymagają wykonania szeregu testów i

pomiarów, aby diagnoza była możliwa. Wymagaj również szeregu dokumentów dotyczących co zaszło, aby można było przeprowadzić stosowne analizy. Oba przypadki, choć skrajnie różne, mają cechę wspólną - poprzez ograniczenia ludzkie zajmują więcej czasu niż by mogły.

Najbardziej prymitywnym sposobem na usprawnienie pracy dzięki technologii jest zaopatrzenie pracownika w różnego rodzaju elektronarzędzia, tablet do sporządzania raportów i przeglądania instrukcji, czy krótkofałówkę do komunikacji. Są to rzeczy na pewno ulepszające proces, natomiast pewne na pewne rzeczy nie mają wpływu - człowiek wciąż musi samodzielnie wypełnić dokumenty, a podczas pracy może mieć szereg wskaźników, które musi śledzić na bieżąco, bądź podążać za procedurami, które nie zawsze da się dokładnie zapamiętać.

Dzięki technikom AR można część tych czynności ułatwić bądź zniwelować. Raport z operacji może zostać wygenerowany automatycznie dzięki nagraniu, bądź danym pobranym w jej trakcie. Testowanie obwodów może się odbywać bez odrywania od nich wzroku, dzięki pokazaniu pracownikowi bezpośrednio przed oczami wskazania miernika, bądź schemat. Nauka obsługi może się odbywać dzięki przedstawieniu zainteresowanemu każdego elementu urządzenia, z dodatkowymi informacjami po wybraniu. Dzięki takim rozwiązaniom wydajność oraz dokładność wzrośnie, ponieważ nie człowiek będzie odpowiadał za część procesu, a dużo wydajniejszy program. Nie może zostać pominięty również czynnik ludzki, czyli zwykła pomyłka. Jej częstotliwość zależy od wielu czynników, takich jak stan psychiczny człowieka, zmęczenie, czas pracy, czy zbliżające się terminy. W przypadku programów jest zgoła odwrotnie. Dobrze napisany będzie działał stabilnie, nie popełniając błędów. Dlatego też wspomaganie pracy programami komputerowymi wydaje się być szczególnie pożąданie w newralgicznych momentach łańcucha produkcyjnego.

Jedną z firm, która wdrożyła rozwiązanie AR jest Avatar Partners. Celem stworzonego przez firmę rozwiązania było przyspieszenie nauki oraz konserwacji myśliwców armii Stanów Zjednoczonych. Stosując bibliotekę Vuforia stworzyli narzędzie, które było w stanie wizualizować położenie przewodów, systemu hydraulicznego i innym elementów. Według twórców czas potrzebny na wykonanie konserwacji został zmniejszony, naprawy przyspieszony oraz zredukowano liczbę popełnianych błędów. Dokładne liczby nie zostały jednak opublikowane. [13]

Firma Ingloba stworzyła dla Huawei Technologies narzędzie AR do wspomożenia

instalacji i konserwacji paneli słonecznych Huawei. Stworzone narzędzie prezentowało kolejne kroki, które pracownik powinien poczynić, aby wykonać dane zadanie. Wykorzystanym urządzeniem jest tablet, więc istnieje możliwość sterowania programem poprzez ekran dotykowy. Poza wizualizacjami nakładanymi na rzeczywiste elementy, do dyspozycji pracownika są również tekstowe oraz filmowe instrukcje. Stworzono także listę kolejnych kroków procesu. Cała operacja jest rejestrowana i wysyłana na serwer w formie filmu. Według Huawei wdrożone rozwiązanie pozwoliło operatorom lepiej zrozumieć wykonywane przez nich procesy, a same prace zaczęły wykonywać szybciej i dokładniej. [14]

## 2.4. Sposoby zastosowań AR

Rozszerzona rzeczywistość zakłada wykorzystanie kontekstu obrazu, który jest dostarczony do programu. Jednym z podejść do wykorzystania kontekstu jest stosowanie metod sztucznej inteligencji. Metody te służą do stworzenia modelu, który na podstawie dostarczonych danych jest w stanie nauczyć się wykonywać pewne zadanie. W przypadku problemu, który przedstawia poniższa praca, interesująca może być sekcja uczenia maszynowego, nazywana *uczeniem nadzorowanym*. Metody te polegają na przekazaniu do modelu sztucznej inteligencji zbioru danych wraz z etykietami zawierającymi poprawną reakcję. Specjalne algorytmy wykonują swoje obliczenia na każdym obiekcie zbioru, a następnie sprawdzają, czy obliczenie jest poprawne. W zależności od odpowiedzi, modyfikują swoje parametry, aby wynik pokrywał się z dostarczonymi etykietami. Podstawowe algorytmy uczenia nadzorowanego mają jednak pewną zasadniczą wadę - są prymitywne i dla celów wizji komputerowej nie były najlepszych rozwiązań. Następcą prostych metod uczenia maszynowego jest uczenie głębokie (ang. Deep Learning - DL). Te metody są już szeroko wykorzystywane w projektach wizji komputerowej. Dzięki faktowi, że mogą wydobywać z obrazu unikalne cechy, zakres możliwości rośnie. Najpopularniejszym wykorzystaniem nauczonych modeli jest rozpoznawanie wielu klas obiektów na obrazie.

DL ma jednak kilka ważnych wad. Aby skutecznie nauczyć model wykonywać swoje zadanie, konieczny jest duży zestaw danych. Należy je odpowiednio przygotować, więc poza samymi zasobami, uczenie kosztuje również czas. Przyjmuje się, że dla każdej klasy obiektów, konieczne jest przygotowanie około 1000 zdjęć. Drugą wadą jest moc obliczeniowa, której zarówno do procesu uczenia, jak i późniejszego działania potrzeba

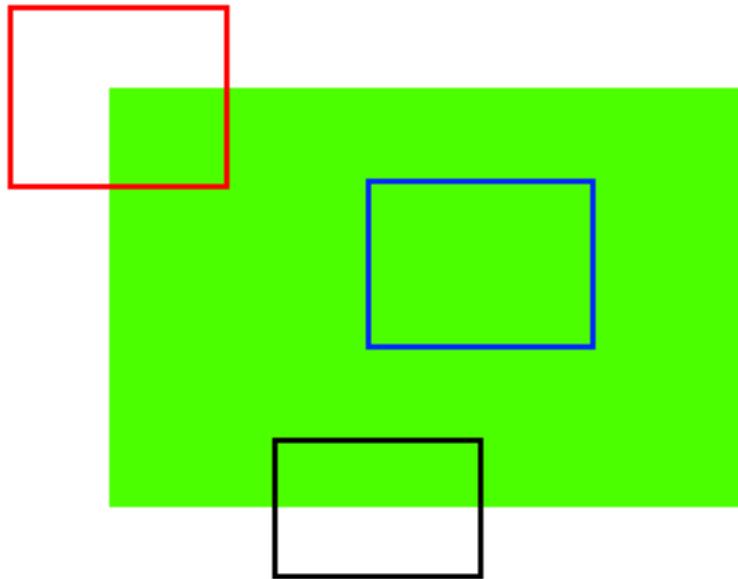
dużo. Ostatnim minusem tego rozwiązania jest nieelastyczność. Modele uczenia głębo-kiego są typu „czarna skrzynka”, a więc zawierają parametry dla człowieka zupełnie niezrozumiałe i zakres działań, które można podjąć jest dosyć wąski - ogranicza się w zasadzie do douczenia modelu.

Innym podejściem do analizy kontekstowej obrazu jest „dopasowanie cech” (ang. feature matching). Polega ono na wydobyciu z obrazu specjalnych cech. Pobierając je z obrazu referencyjnego (obiektu, który chcemy wykryć) oraz obrazu z kamery (na którym jest poszukiwany obiekt), można je porównać i wyszukać czy i gdzie znajduje się obiekt referencyjny. Przewagą tej metody nad uczeniem głębokim jest fakt, że:

- a) wymaga mniej mocy obliczeniowej,
- b) nie wymaga uczenia,
- c) dzięki możliwości dostosowania parametrów wykrywania cech, elastyczność rozwiązania jest większa.

Feature matching jest metodą wykorzystywana nie tylko do rozpoznawania obiektów. Dzięki temu, że część wykrytych cech będzie się pokrywać na dwóch obrazach mających wspólny obszar, możliwe jest łączenie zdjęć w panoramy. Inne popularne zastosowanie do stabilizacji obrazu. Wiedząc, że kamera drga w osiach prostopadłych do osi głównej kamery, obraz końcowy można delikatnie przesuwać, dzięki czemu będzie on stabilny. Feature, jest unikalną cechą danego fragmentu obrazu. Mogą być to: krawędzie, narożniki, tekstury, koła i okręgi.

Na rysunku 2.4 wyodrębnione zostały trzy obszary: niebieski, czarny i czerwony. Pierwszy obszar nie wyróżnia się, nie jest więc unikalny. Podobnie sytuacja ma się w przypadku obszaru czarnego - tak samo wygląda cały obraz wzduż dolnej krawędzi zielonego prostokąta. Natomiast czerwony obszar jest unikalny, czyli nie ma drugiego takiego samego na rysunku. Wyszukiwanie unikalnych obszarów obrazu nazywa się detekcją cech (ang. Feature detection), a same cechy punktami charakterystycznymi (ang. keypoints). Kolejnym krokiem jest przetworzenie punktów, aby można było je porównać z tymi z innego rysunku. Taki proces nazywa się opisywaniem cech (ang. Feature description), a wynikiem tej operacji są deskryptory (ang. descriptors). Z punktu widzenia programu, zarówno punkty kluczowe jak i deskryptory są wektorami bądź tablicami asocjacyjnymi zawierającymi liczby. W dalszej części pracy opisane zostaną najpopularniejsze algorytmy obsługujące proces zarówno detekcji jak i deskrypcji.



Rysunek 2.4: Potencjalne cechy

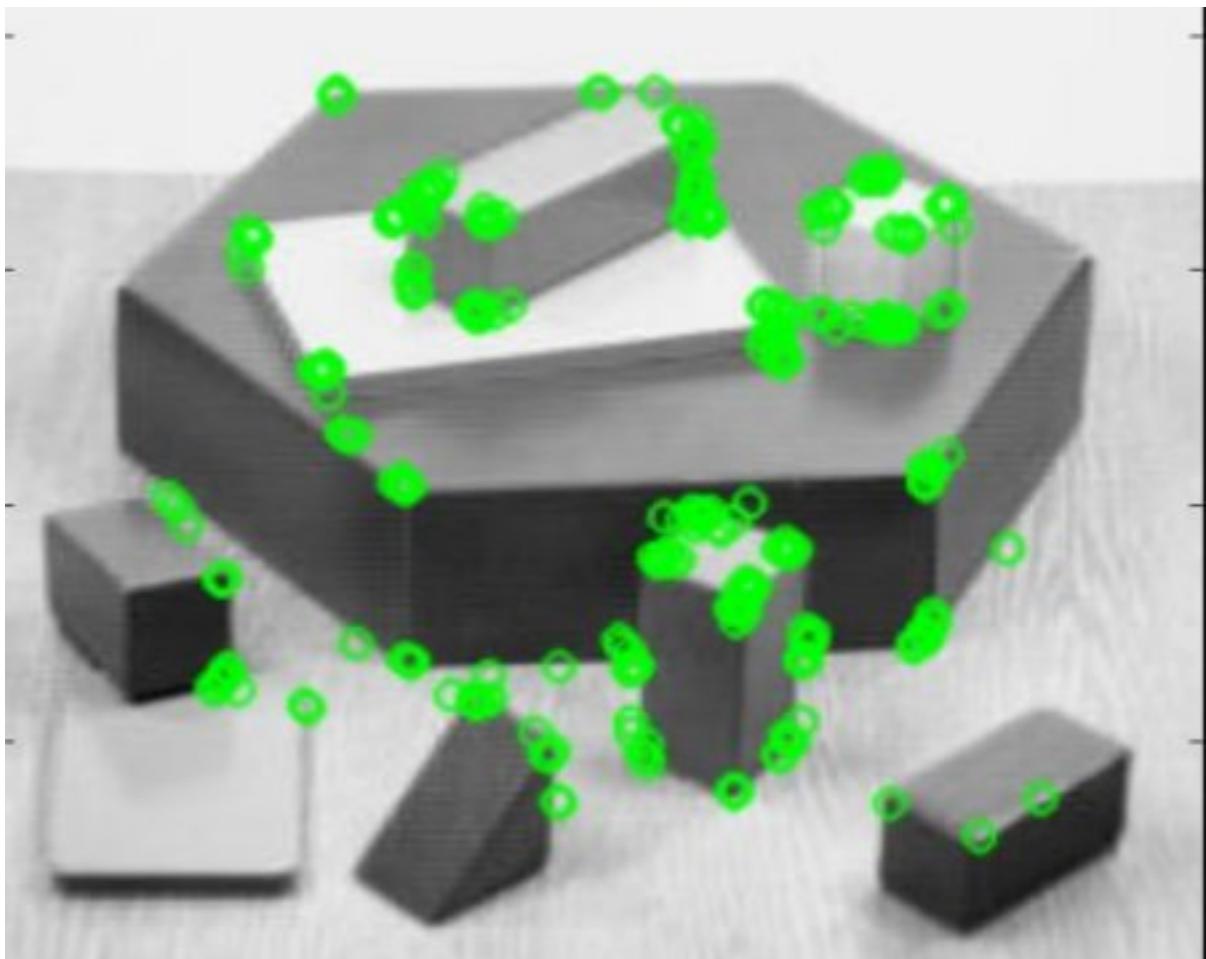
## 2.5. Algorytmy detekcji i deskrypcji

### 2.5.1. ORB - Oriented FAST and Rotated BRIEF

ORB jest metodą detekcji oraz deskrypcji punktów charakterystycznych wykorzystaną w poniższej pracy. Spaja ona detektor FAST oraz deskryptor BRIEF. Obie te techniki wyróżniają się niskimi wymaganiem mocą obliczeniowej. Autorzy tworząc to rozwiązanie podjęli się optymalizacji oraz zwiększenia odporności na zakłócenia.

ORB na początku wykorzystuje FAST do znalezienia punktów, następnie najmniej przydatne z nich są filtrowane za pomocą metody detekcji narożników Harrisa. Orientacja obszaru jest opisywana poprzez położenie wartości punktu nazwanego „intensywny centroid”. Zakłada się, że intensywnością jest przesunięcie narożnika od centroidu daje wektor jego orientacji. Sam centroid jest punktem geometrycznego środka danej figury. Deskrypcja odbywa się za pomocą algorytmu BRIEF. Ponieważ jednak nie radzi sobie on z rotacjami, wykorzystywane są te już obliczone dla punktów charakterystycznych. Każdy zestaw cech posiada  $n$  binarnych testów w miejscu  $(xi, yi)$ . Tworzona jest dla nich macierz  $S$  o rozmiarze  $2 \times n$ , zawierającą te współrzędne pikseli. Za pomocą orientacji cechy  $\theta$  obliczona zostaje macierz rotacji, która po zastosowaniu obraca  $S$ , tworząc  $S_\theta$ .

ORB dzieli cały obrót na obszary po  $2\pi/30$  (12 stopni) i tworzy tabelę dla



Rysunek 2.5: Wykryte przez ORB cechy na przykładowym obrazie [15]

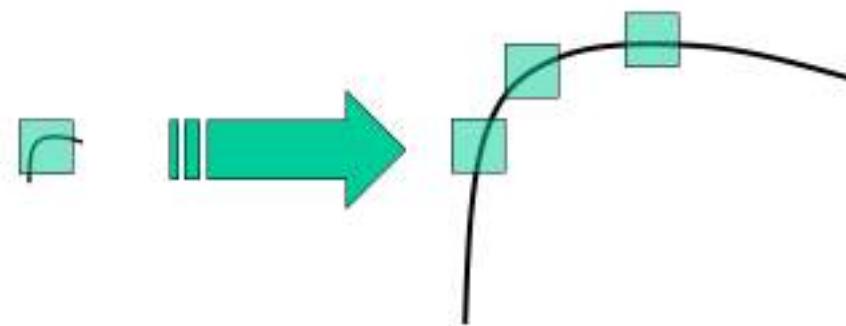
wzorców BRIEF. Dopóki orientacja punktu  $\theta$  jest spójna pomiędzy obrazami, poprawny zestaw punktów  $S_\theta$  zostanie użyty do wyliczenia deskryptora.

Każda cecha BRIEF posiada dużą wariancję i średnią bliską 0.5. Jednak gdy zostanie zorientowana wzdłuż wektora orientacji punktu, wartości te stają się bardziej rozproszone. Kolejną pożądana właściwością jest brak korelacji pomiędzy testami, aby nie wpływać na własne wyniki. W tym celu ORB przeprowadza przeszukiwanie zaczątkowe wśród wszystkich testów aby znaleźć te z dużą wariancją i średnią bliską 0.5, oraz brakiem korelacji. Metoda, ze względu na swoją szybkość i dokładność, została wykorzystana do stworzenia projektu realizowanego w ramach niniejszej pracy.

### 2.5.2. SIFT - Scale Invariant Feature Transform oraz SURF - Speeded-Up Robust Features

Inne popularne detektory i deskryptory to SIFT oraz SURF. Są to algorytmy opatentowane i płatne do komercyjnego zastosowania. SURF bazuje na SIFT i jest szybszą opcją.

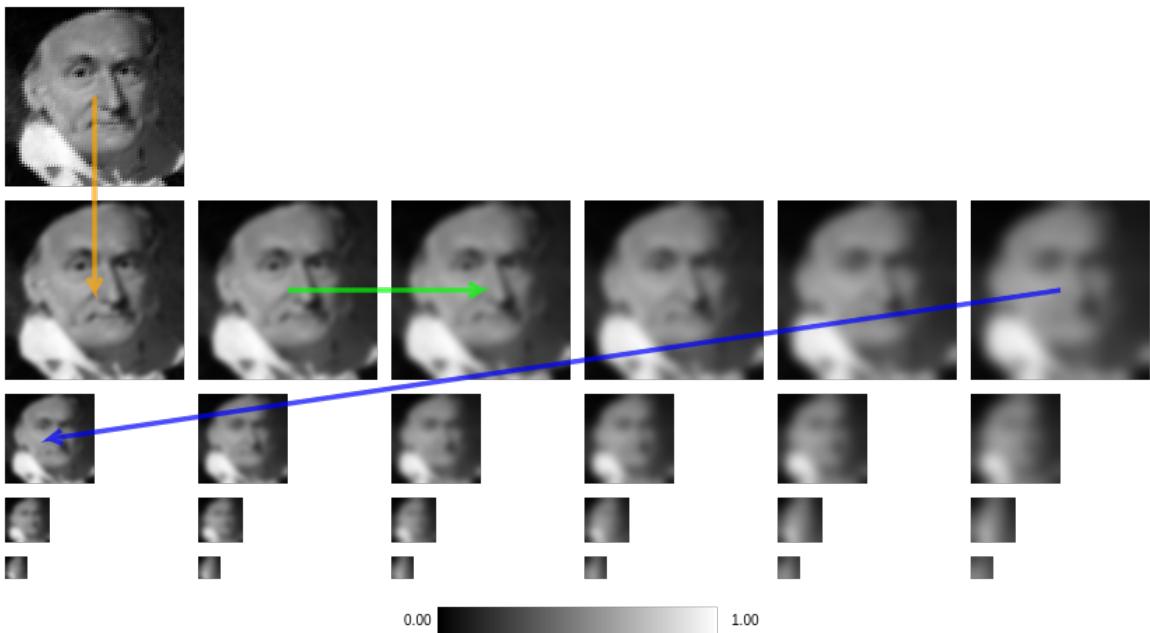
Detektor narożników Harrisa jest w stanie wykryć narożniki niezależnie ich obrotu, jednak w przypadku zmiany skali sytuacja ma się inaczej. To co na małym obrazie jest narożnikiem, na dużym (dla badanych obszarów o jednakowej wielkości) jest tylko krzywą.



Rysunek 2.6: Porównanie narożników w różnej skali [?]

Rozwiązanie tego problemu opisał w 2004 roku D. Lowe z uniwersytetu Kolumbii Brytyjskiej, w artykule „Distinctive Image Features from Scale-Invariant Keypoints” [16], który poza wyznaczeniem punktów, oblicza również deskryptory. Na początku rozmiar obrazu jest podwajany za pomocą metod interpolacji. Następnie następuje iteracja rozmyć używając rozmycia Gaussa, z czego każdy kolejny obraz jest ge-

nerowany ze zwiększym odchyleniem standardowym. Następnie rozmiar obrazu jest zmniejszany dwukrotnie i występuje kolejna iteracja. Każda sekwencja iteracji pomiędzy pomniejszeniami obrazu nazywa się oktawą. Zjawisko to odbywa się do momentu w którym obraz jest zupełnie nieczytelny. Tworzona jest w ten sposób przestrzeń skali (ang. scale space). Jej celem jest symulowanie różnych skali obserwowanego obrazu. Każdy ze stworzonych rysunków jest normalizowany. Następnie zakłada się, że przestrzeń skali ma trzy wymiary: koordynaty x, i y oraz odchylenie standardowe. Punkty kluczowe znajduje się za pomocą metody zwanej różnicą funkcji Gaussa. Dla każdego odpowiadającego sobie w ramach danej oktawy piksela wylicza się różnicę, a następnie wyznacza ekstrema. Wizualizacja tego działania przedstawiona jest na rysunku ??.



Rysunek 2.7: Schemat działania ekstrakcji cech [?]

Wyznaczone punkty są następnie filtrowane. Konieczne jest wykluczenie krawędzi wśród wykrytych cech oraz punktów o niskim kontraste, aby zwiększyć stabilność. Dzieje się to za pomocą macierzy  $H$ , o rozmiarze  $2 \times 2$ , której wartości są proporcjonalne do krzywizny danego obszaru. Następnie do każdego punktu przypisywana jest orientacja. Tworzony jest histogram według orientacji gradientów wokół określonej cechy. Ma on 36 przedziałów, każdy odpowiada 10 stopniom. Każda wartość przedziału to wielkość gradientu i  $1,5 * \theta$  (okno gaussowskie). Wybierane jest maksimum.

Kolejnym krokiem jest stworzenie deskryptora. Wybierane jest  $16 \times 16$  pikseli wokół punktu. Obszar zostaje podzielony na 16 mniejszych o rozmiarze  $4 \times 4$ . Dla każdego małego obszaru tworzony jest histogram orientacji, łącznie 128 przedziałów. Wartości te finalnie przedstawione są jako wektor.

SURF jest bardziej wydajnym odpowiednikiem SIFT. Punkty kluczowe wyznaczone są za pomocą funkcji Hessa, a deskryptory zawierają informacji o ich położeniu i rozmieszczeniu.

## 2.6. Dopasowanie cech

Dopasowanie cech wykorzystuje się do rozpoznawania obiektu z obrazu referencyjnego na drugim obrazie. Ze względu na dużą liczbę obliczeń w tym obszarze, wybranie optymalnego rozwiązania rzutuje w dużej mierze na szybkość całego systemu.

### 2.6.1. Brute-Force Matcher

Najbardziej prymitywnym sposobem na dopasowanie jest „dopasowanie brutalne” (ang. Brute-Force). Realizuje się to poprzez porównanie deskryptora punktu do wszystkich deskryptorów z drugiego. Liczony jest dystans pomiędzy parami cech. Im jest on mniejszy, tym dopasowanie większe. Zwracana jest para o najmniejszym dystansie. W zależności od sposobu obliczania odległości, istnieją różne wersje algorytmu. Najpopularniejsze to [20, s.573]:

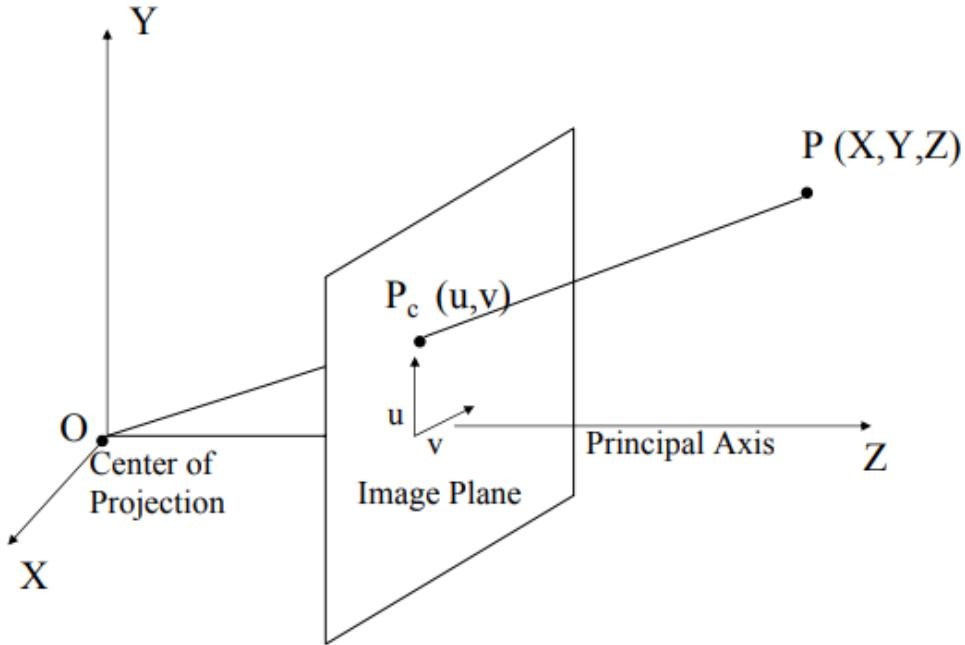
- a) NORM\_L2, który wylicza odległość euklidesową pomiędzy cechami; stosowany dla algorytmów SIFT i SURF
- b) HAMMING, wylicza odległość Hamminga (wykonanie XOR na ciągach binarnych i ich sumowanie); stosowany dla binarnych rozwiązań jak ORB, BRIEF, BRISK.

### 2.6.2. FLANN based Matcher

Inną metodą jest FLANN, będący akronimem od „Fast Library for Approximate Nearest Neighbors”. Zawiera w sobie zestaw algorytmów zoptymalizowanych pod proces poszukiwania najbliższych sąsiadów w dużych zbiorach danych oraz z wielowymiarowymi cechami. Metoda ta działa szybciej niż Brute-Force na dużych zbiorach i została wykorzystana w niniejszej pracy [20, s.575].

## 2.7. Model kamery otworkowej (Pinhole camera model)

Podstawowy model kamery, zakładający, że wszystkie promienie światła zbiegają do jednego punktu - otworu kamery. Za jej pomocą wykonano pierwszą фотографię, przedstawiającą budynki rolnicze i niebo. Poniższy rysunek prezentuje zasadę działania takiej kamery.



Rysunek 2.8: Geometryczny model kamery otworkowej [18]

Punkt zbiegu fal świetlnych jest w punkcie  $O$ . Główną osią jest  $Z$ . Płaszczyzna obrazu jest przesunięte od punktu  $O$  o długość ogniskowej  $f$ . Punkt rzeczywisty  $P$  o współrzędnych  $(X, Y, Z)$  rzutowany jest na płaszczyznę obrazu kamery na punkt  $P_c = (u, v)$ . Znając długość ogniskową oraz współrzędne punktu rzeczywistego, można wyznaczyć współrzędne punktu na obrazie.

$$\frac{f}{Z} = \frac{u}{X} = \frac{v}{Y}$$

co po przekształceniach daje

$$u = \frac{fX}{Z}$$

$$v = \frac{fY}{Z}$$

Najczęściej tego typu obliczenia wykonuje się za pomocą macierzy, więc można powyższe równania zapisać jako

$$\begin{pmatrix} u \\ v \\ w \end{pmatrix} = \begin{pmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} \quad (2.1)$$

Jeśli jednak oś  $Z$  nie przechodzi przez centrum obrazu z kamery, konieczne jest wykonanie translacji. Niech środek kadru reprezentuje  $(c_x, c_y)$ , wtedy

$$u = \frac{fX}{Z} + c_x$$

$$v = \frac{fY}{Z} + c_y$$

modyfikując równanie 2.1 otrzymamy

$$\begin{pmatrix} u \\ v \\ w \end{pmatrix} = \begin{pmatrix} f & 0 & c_x \\ 0 & f & c_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \end{pmatrix}$$

Ponieważ parametry rzeczywiste kamery podaje się w calach, konieczne jest przeskalowanie wartości do pikseli. Niech stosunek piksel/cal dla każdej z osi obrazu opisuje się wektorem  $(m_x, m_y)$ , wtedy

$$\begin{pmatrix} u \\ v \\ w \end{pmatrix} = \begin{pmatrix} m_x * f & 0 & m_y * c_x \\ 0 & m_y * f & m_y * c_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} = \begin{pmatrix} \alpha_x & 0 & u_0 \\ 0 & \alpha_y & v_0 \\ 0 & 0 & 1 \end{pmatrix} * P = KP$$

Macierz  $\begin{pmatrix} \alpha_x & 0 & u_0 \\ 0 & \alpha_y & v_0 \\ 0 & 0 & 1 \end{pmatrix}$  nazywa się macierzą parametrów wewnętrznych kamery i oznacza się literą  $K$ .

Jeśli punkt  $O$  według danego układu współrzędnych nie leży w punkcie  $(0, 0, 0)$ , a oś  $Z$  nie jest prostopadła do płaszczyzny obrazu, konieczne jest wykonanie translacji oraz rotacji. Macierz przesunięcia kamery do punktu  $0$  to  $T(T_x, T_y, T_z)$ , a macierz rotacji obracająca kamerę do pozycji jak na rysunku ?? będzie nazwana macierzą rotacji  $R$ , o rozmiarze  $3 \times 3$ . Tworzona jest macierz

$$E = (R | RT)$$

nazwaną macierzą parametrów zewnętrznych, więc kompletnym równaniem transformacji jest

$$K(R|RT) = (KR|KRT) = KR(I|T)$$

na punkt  $P_c$  jest wyliczany przez

$$P_c = KR(U|T)P = CP$$

gdzie  $C$  jest jest macierzą o rozmiarze  $3 \times 4$  nazwaną macierzą kamery.

## 2.8. Homografia

Jeśli dwie kamery są zwrócone w kierunku jednego punktu, powiązanie pomiędzy nimi może być w prosty sposób obliczone. Rysunek ?? ilustruje sytuację, gdzie dla punktu  $P\pi$ , kamera 1 rzutuje go na obraz w punkcie  $P_\pi^1$ , kamera 2 analogicznie na  $P_\pi^2$ . Wiemy też, że punkt  $P_\pi$  leży na płaszczyźnie  $\pi$ . Niech wektor normalny  $N$  płaszczyzny będzie zdefiniowany przez  $N = (a, b, c)$ . Wtedy równanie płaszczyzny będzie równe:

$$(N, 1) \cdot P = 0$$

dla każdego punktu w świecie 3D.

Wiemy, że

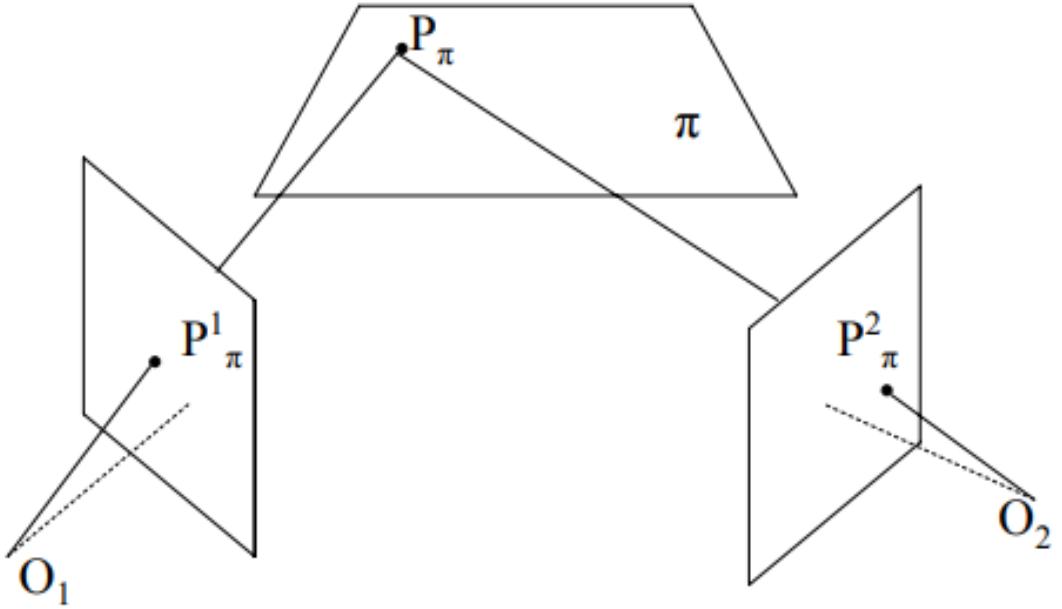
$$P_\pi^1 = \begin{pmatrix} u_1 \\ v_1 \\ w_1 \end{pmatrix} = C_1 \cdot P_\pi$$

Oznacza to, że punkt  $P_\pi$  leży na promieniu  $(u_1, v_1, w_1, 0)^T$ , jednak parametr skali jest nieznany. Nazwiemy go  $\tau$ , otrzymując

$$Ppi = \begin{pmatrix} u_1 \\ v_1 \\ w_1 \\ \tau \end{pmatrix} = \begin{pmatrix} P_\pi^1 \\ \tau \end{pmatrix}$$

Następnie, wiedząc, że  $P_\pi$  spełnia równanie płaszczyzny otrzymujemy

$$\tau u = -N \cdot P_\pi^1$$



Rysunek 2.9: Przykład homografii [18]

, więc

$$P_\pi = \begin{pmatrix} u1 \\ v1 \\ w1 \\ \tau \end{pmatrix} = \begin{pmatrix} I \\ -N \end{pmatrix} P_\pi^1$$

$I$  jest macierzą  $3 \times 3$ ,  $N$   $1 \times 3$ , więc otrzymana jest macierz  $4 \times 3$ . Niech  $C_2 = \begin{pmatrix} A_2 & a_2 \end{pmatrix}$ , gdzie  $A_2$  to macierz  $3 \times 3$ , a  $a_2$  wektor  $3 \times 1$ . Wtedy,

$$P_\pi^2 = C_2 \cdot P_\pi = \begin{pmatrix} A_2 & a_2 \end{pmatrix} \begin{pmatrix} I \\ -N \end{pmatrix} P_\pi^1$$

Macierz  $C_2$  ma rozmiar  $3 \times 4$ , kolejna  $4 \times 3$ , więc wynikiem jest macierz  $3 \times 3$ , nazywaną macierzą homografii. Ostatecznie

$$P_\pi^2 = (A_2 - a_2 N) P_\pi^1 = H P_\pi^1$$

Macierz homografii pokazuje relację pomiędzy obrazem z dwóch kamer. Używając jej, możliwa jest transformacja widoku, aby odpowiadał położeniu drugiej [20, s.660].

## **2.9. Zastosowane technologie**

### **2.9.1. OpenCV**

Najważniejszą technologią zastosowaną w niniejszej pracy jest OpenCV. Jest to popularna otwarta biblioteka o otwartych źródłach służąca do przetwarzania obrazu, stworzona przez firmę Intel. Oficjalnie projekt rozpoczął się w 1999 roku. Obecnie zawiera w sobie ponad 2500 algorytmów rozwiązywających zarówno typowe problemy wizji komputerowej, jak i algorytmy uczenia maszynowego.

Obecnie społeczność OpenCV liczy ponad 47 tysięcy ludzi, a liczba pobrań przekracza 18 milionów. Rozwijają ją głównie specjaliści od optymalizacji oraz zespół optymalizacji Intel'a. Korzystają z niej zarówno firmy komercyjne, grupy badawcze jak i urzędy. Biblioteka jest wykorzystywana zarówno do łączenia zdjęć w Google StreetView, detekcji włamywacza w Izrealu jak i monitoringu sprzętu górnictwa w Chinach.

Biblioteka ma interfejsu dla C++, Pythona, Javy i MATLABa, wspiera systemy Windows, Linux, Android i MacOS. Współpracuje z bibliotekami Tensorflow, PyTorch i innymi. Praca nad współpracą z technologiami CUDA oraz OpenCL trwają. OpenCV zostało napisane w C++ [19].

### **2.9.2. Arduino**

Arduino jest otwartą platformą dla systemów wbudowanych w ramach jednego obwodu drukowanego ze standaryzowaną biblioteką. Język, w którym programuje się urządzenia Arduino jest rozszerzeniem języka C. Motywacją do stworzenia i rozwijania projektu było przygotowanie tanich, prostych w obsłudze i powszechnie dostępnych narzędzi do programowania mikrokontrolerów, aby uprościć, a w niektórych przypadkach nawet umożliwić pracę z systemami wbudowanymi. Arduino zawiera wbudowany programator, UART bądź USB do komunikacji z komputerem, cyfrowe i analogowe wejścia/wyjścia. Dzięki standaryzacji, możliwe było stworzenie szeregu rozszerzeń do urządzenia, jaki np. moduł WiFi, Bluetooth, GSM, szereg czujników, driverów do silników itd. Dodatkowo użytkownicy mogą tworzyć własne biblioteki i je udostępniać innym.

Do wykonania poniższej pracy wykorzystano model Arduino Leonardo ze sprzętową obsługą protokołu USB.

Domyślnym środowiskiem do tworzenia programów na Arduino oraz komunika-

cji z urządzeniem jest program Arduino IDE. Pozwala w prosty sposób zaprogramować płytę, monitorować port szeregowy, a także pisać programy i pobierać biblioteki.

### **2.9.3. MQTT - Message Queue Telemetry Transport**

Jest to protokół komunikacyjny stworzony przez Andy'ego Stanford-Clarka z IBM oraz Arlena Nippera z Eurotech, oparty o wzorzec Publish-Subscribe (ang. publikacja-subskrypcja). Został stworzony do transmisji danych przez urządzenia bez dużej przepustowości oraz mocy obliczeniowej. MQ w nazwie ma swoją genezę od produktów IBM. Dzięki swojej lekkości i niezawodności wykorzystywany jest w urządzeniach mobilnych oraz internecie rzeczy (ang. Internet of Things - IoT). Wykorzystywany jest m. in. w Facebook Messenger czy AWS IoT, a także w rozrusznikach serca [21].

Architektura Publish-Subscribe zakłada, że wiadomości wysyłane przez nadawcę (publisher) trafiają do serwera (broker), a ten następnie przesyła wiadomość do zainteresowanych odbiorców (subscriber). Wiadomości przesyłane są w kanałach nazywanych tematami (topic). Odbiorca, musi subskrybować dany temat, aby otrzymać wiadomości od brokera. Nadawca nie ma informacji kto otrzyma wiadomość, ani czy ktokolwiek jest nią zainteresowany. Dzięki temu nie występuje tutaj periodyczne odpytywanie serwera aby zaktualizować dane (ang. pooling). Nadawca wysyłając wiadomość, musi także przekazać jakiego tematu ona dotyczy.

Istnieje wiele bibliotek umożliwiających stworzenie brokera, jak np. Mosquitto, HiveMQ czy RabbitMQ, jednak do celów realizacji projektu wykorzystano Mosquitto. Jest to broker o otwartych źródłach od Eclipse.

### **2.9.4. Języki programowania**

Do realizacji projektu wykorzystano 2 języki programowania: C oraz Python.

Pierwszy z nich jest językiem prezentującym paradygmat imperatywny. Jest wysokopoziomowy i kompilowany. Pojawił się w 1972. Dzięki swojej prostocie i dostępie do niskopoziomowych narzędzi, jak np. zarządzanie pamięcią, jest często wykorzystywany w systemach wbudowanych. Tak też jest w tym przypadku - Z jego pomocą zaprogramowano urządzenie do komunikacji z brokerem. Wykorzystany język był w wersji zmodyfikowanej przez fundację Arduino, dzięki czemu dostęp do kanałów wejścia/wyjścia był łatwiejszy.

Python jest językiem wysokopoziomowym ogólnego przeznaczenia. Realizuje paradygmaty obiektowy, imperatywny i strukturalny. Jest językiem interpretowanym i

posiada system dynamicznych typów. Dzięki swojej prostej do zrozumienia składni znalezł zwolenników ze wielu dziedzin. Jest to obecnie czwarty najbardziej popularny język na świecie [17]. Jest on również bardzo wygodny przy korzystaniu z OpenCV, dlatego został on wybrany do realizowania projektu rozszerzonej rzeczywistości.

Operacje na obrazach z pomocą OpenCV w Pythonie realizuje się z pomocą biblioteki NumPy, dlatego również została ona użyta w pracy.

### **3. Realizacja projektu**

#### **3.1. Przyjęte założenia**

Ze względu na ograniczenia czasowe i finansowe autora, musiały zostać popełnione pewne założenia i uproszczenia, aby projekt mógł zostać zrealizowany.

Wysoka cena okularów rozszerzonej rzeczywistości skutecznie odstraszyła autora przed ich zakupem. Dodatkowo trudności w prototypowaniu na urządzeniach mobilnych spowodowały również odrzucenie tego typu medium. Autor zdecydował się więc na realizowanie systemu rozszerzonej rzeczywistości na prywatnym laptopie Lenovo Thinkpad T460s. Sam program został napisany w języku Python, ze względu na jego prostotę i elastyczność. Wykorzystane zostały jedynie darmowe narzędzia, z których autor miał prawo korzystać. Rezygnacja z zewnętrznego urządzenia spowodowała pewne ograniczenie, ponieważ laptop posiada kamerę skierowaną jedynie w kierunku użytkownika, dlatego też zdecydowano się na wykorzystanie zewnętrznej kamery USB.

Ze względu na brak rzeczywistych urządzeń (zwanych dalej maszynami) stosowanych w przemyśle, autor podjął decyzję o wykonaniu prostego urządzenia mającego symulować te rzeczywiste i bardziej skomplikowane. Sterowanie i komunikację z serwerem realizuje się dzięki płytce Arduino Leonardo. Urządzenie składa się z 3 rezystorów, przycisku dwustanowego włącz/wyłącz, oraz dwóch przycisków typu pushbutton. Urządzenie posiada czarną obudowę. Ze względu na prostą frontu urządzenia, podjęto decyzję o naklejeniu na nie kawałek papieru z naniesionym wzorem, aby zwiększyć dokładność detekcji.

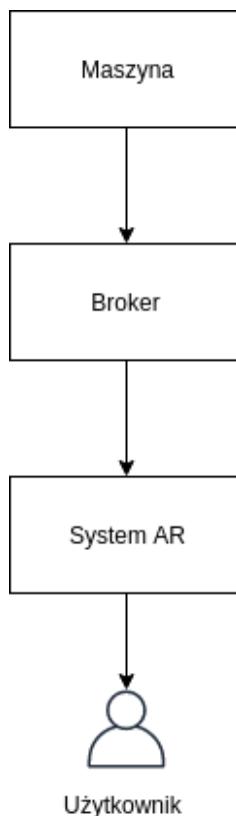
Obecne rozwiązanie wspiera rozpoznawanie jednego obiektu.

Ważnym założeniem przy tworzeniu narzędzia było to, aby nie wpływało ono w żaden sposób na działanie potencjalnej rzeczywistej maszyny. Dodatkowo wdrożenie do działającego już urządzenia powinno ograniczać się do dodania kanału komunikacji, bądź w przypadku jego istnienia tylko i wyłącznie do stworzenia pliku konfiguracyjnego.

Dodatkowo, aby wszystko działało poprawnie, zarówno maszyna jak i system AR powinny mieć połączenie z brokerem. W prezentowanej pracy połączenie jest bezprzewodowe.

### 3.2. Architektura rozwiązania

Cały system składa się z 3 bytów: Maszyny, brokera, oraz systemu AR. Zjawisko to obrazuje rysunek 3.10.

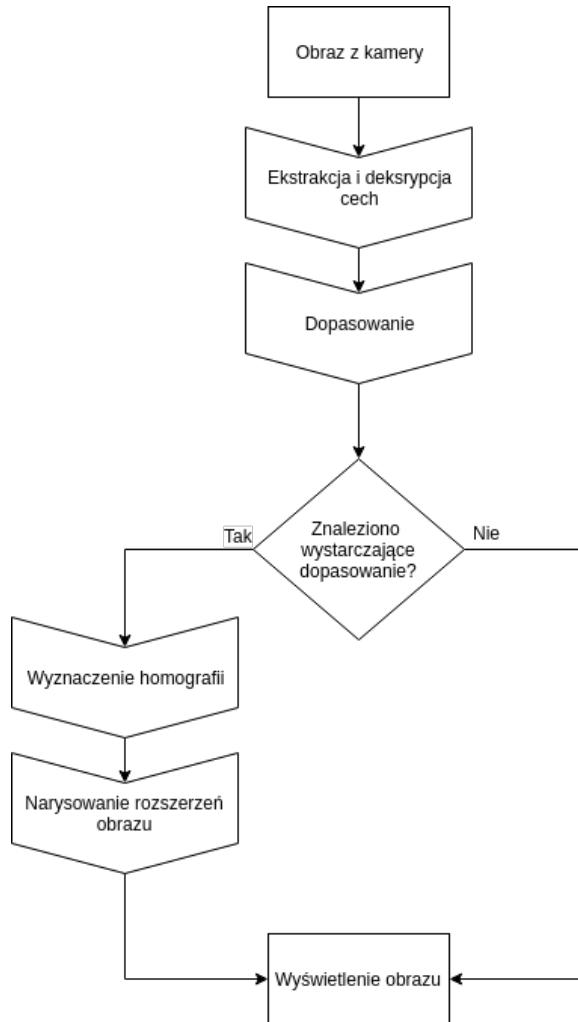


Rysunek 3.10: Architektura rozwiązania. Źródło: własne

Komunikacja jest jednostronna: maszyna => broker => system AR. Można wywnioskować, że środkowy element jest niepotrzebny, a komunikację można poprowadzić bezpośrednio maszyna => AR. Jest to myślenie poprawne, jednak jednym z założeń była możliwie największa elastyczność rozwiązania. Wiąże się to z tym, że powinna istnieć możliwość zamianienia środkowego elementu na dowolnie inny: REST Api, WebSocket, ERP itd., aby, w razie gdy istniała już jakaś infrastruktura komunikacji pomiędzy maszyną a dowolnym tworem agregującym, było możliwe dołączenia modułu AR bez konieczności przebudowywania istniejącej architektury.

Algorytm działań realizowany od odebrania obrazu z kamery do jego wyświetlenia obrazuje rysunek 3.11. Zgodnie z nim pierwszym krokiem jest ekstrakcja i detekcja cech. Zostało to połączone, ponieważ również w kodzie odbywa się to za pomocą wywołania jednej funkcji. Następnie realizowane jest *feature matching* z tymi z obrazu

referencyjnego. Jeśli nieodpowiednio silne dopasowanie nie zostanie znalezione, wyświetlany jest niezmieniony obraz. Jeśli jednak detekcja będzie pozytywna - poszukiwana zostaje macierz i maska homografii, oraz wywoływana na kadrze jest funkcja rysująca. Tak zmodyfikowany kadr jest prezentowany użytkownikowi.



Rysunek 3.11: Algorytm rozwiązania. Źródło: własne

### 3.3. Maszyna

#### O urządzeniu

Maszyna, jak wspomniano wcześniej, pełni jedynie rolę symulacji prawdziwego urządzenia. Oparta jest o układ Arduino z programem napisanym w języku C z nakładką przygotowaną przez fundację Arduino. Do połączenia się z siecią WiFi użyto dedykowany do tego celu moduł ESP8266. Wykorzystano również diodę LED czerwoną, diodę RGB, 3 potencjometry, dwustanowy przycisk (On/Off), oraz 2 przyciski

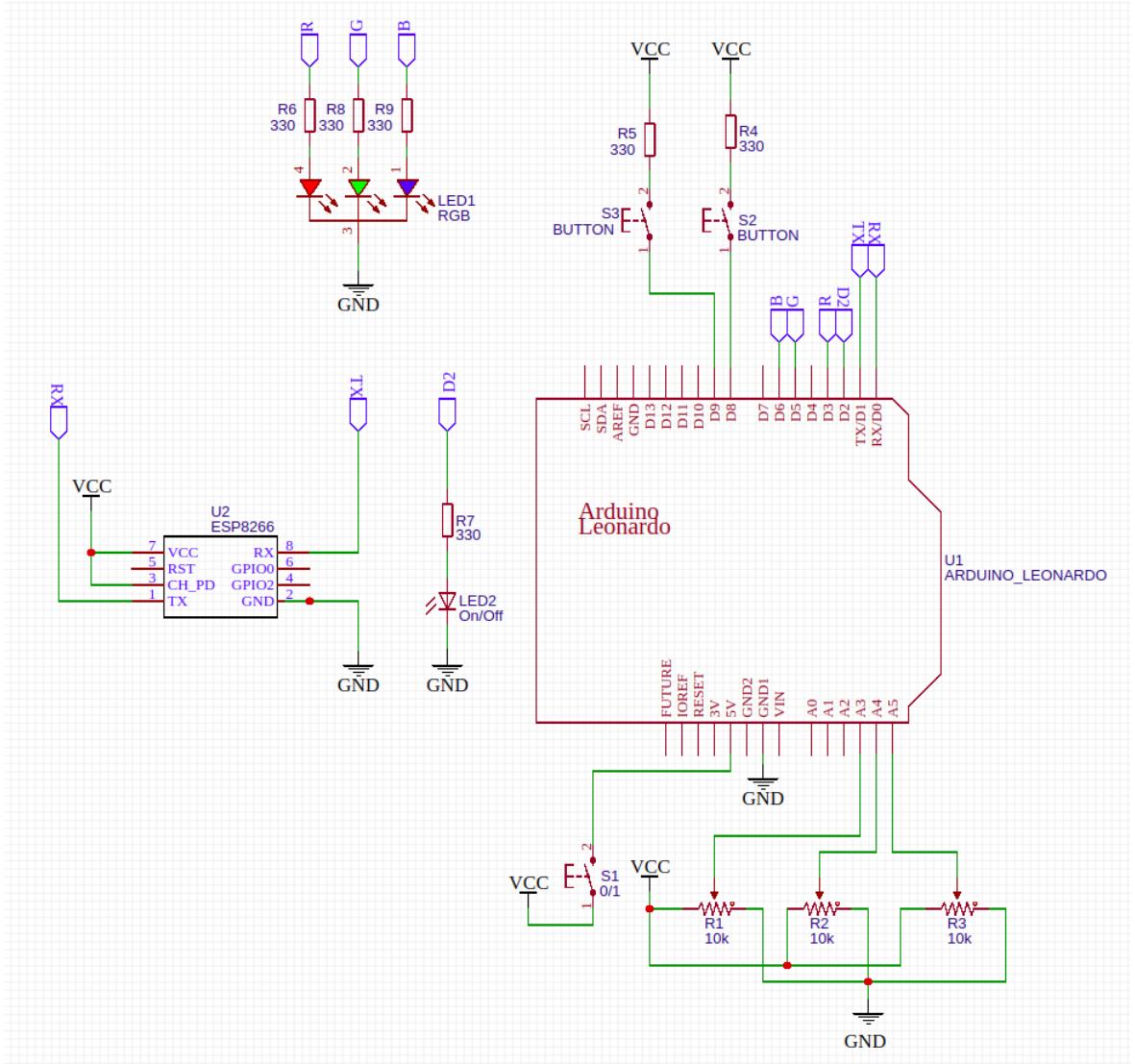
typu pushbutton. Urządzenie zasilane jest poprzez przewód USB. Dokładny schemat połączeń przedstawia rysunek 3.13.



Rysunek 3.12: Zdjęcie urządzenia. Źródło własne

### Schemat połączeń

- a) A3, A4, A5 służą jako przetworniki analogowo/cyfrowe z rozdzielczością 1024. Ich zadaniem jest przetworzenie aktualnego stanu napięcia (rezystancji) na potencjometrach do wartości całkowitej w programie. Każdy z potencjometrów odpowiada za jedną składową koloru diody RGB. Kolejno R, G i B.
- b) D3, D5, D6 generują sygnał PWM doprowadzany do pinów diody RGB, kolejno R, G, B. Wartość wypełnienia zależy od stanu napięcia na potencjometrach.
- c) Przycisk na pinie 5V miał symulować przycisk włączenia/wyłączenia urządzenia, jednak ze względu na problemy w działaniu płytki Arduino po zmianie stanu, nie pełni on żadnej roli.
- d) Piny RX/TX zostały skrosowane i połączone z układem ESP8266.
- e) Dioda LED2 połączona do pinu D2 ma symbolizować uruchomienie układu.

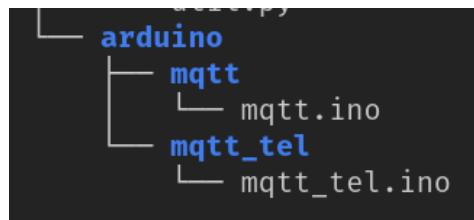


Rysunek 3.13: Schemat połączeń. Źródło własne

- f) Przyciski połączone do pinów D8,D9 mają symulować zdarzenia , które pojawiły się w trakcie działania maszyny.
- g) Piny VCC oraz CH\_PD zostały połączone do źródła zasilania z Arduino.

### Struktura plików

Programy zawarte są w folderze arduino, w którym są 2 podfoldery: mqtt oraz mqtt\_tel, a każdy z nich zawiera po jednym pliku .ino. Wynika to z architektury Arduino, gdzie każdy skrypt powinien znajdować się w folderze o takiej samej nazwie. Oba pliki .ino zawierają analogiczny kod do mikrokontrolera, przy czym wariant z sufikssem \_tel posiada dane do logowania do sieci WiFi udostępnianej przez telefon autora, a skrypt bez sufiksu - do sieci lokalnej w domu autora.



Rysunek 3.14: Struktura plików na urządzenie Arduino. Źródło własne

## Omówienie programu

Każdy program pisany dla Arduino można podzielić na trzy sekcje:

- importowanie bibliotek, definiowanie stałych i zmiennych,
- funkcja setup,
- funkcja loop.

Zawartość pierwszej sekcji dokładnie opisuje punkt a) powyższej listy. Do zrealizowania projektu wykorzystano 3 biblioteki: WiFiESP - umożliwiające połączenie się z siecią WIFI poprzez układ ESP8266; PubSubClient - pozwala na publikowanie wiadomości protokołem MQTT; Bounce2 odpowiada za zredukowanie wpływu drgań styków w momencie wcisnięcia przycisku pushbutton. Tworzony jest tutaj także klient MQTT. Stałe definiować można na 2 sposoby, poprzez

```
#define <nazwa> <wartosc>
```

co pozwala na bezpośrednie podmienienie nazwy w programie na wartość, bądź

```
const <typ> <nazwa> = <wartosc>
```

Drugi sposób łączy nazwę zmiennej z komórką w pamięci podręcznej urządzenia, do której później program będzie się odwoływał gdy natrafi na daną nazwę. W programie występują również zmienne, definiowane w następujący sposób (fragment w nawiasach kwadratowych jest opcjonalny)

```
<typ> nazwa [= <domyslna wartosc>];
```

Funkcja setup ma za zadanie przygotować program do późniejszego zadania. W tym miejscu ustawia się, czy pin ma pełnić funkcję wejścia czy wyjścia, konfiguruje się porty szeregowe, a także inicjuje połączenie wifi.

Funkcja loop jest pętlą nieskończoną. Można z niej wyjść, natomiast spowoduje to przerwanie programu. W jej ciele wykonują się sczytania z pinów i ustawienie na nich wartości; wysyłane są również wiadomości do brokera MQTT. Wartości z potencjometrów domyślnie mieszczą się w zakresie 0-1023, natomiast na potrzeby programu, wartości te zostały przeskalowane do 0-255. Dodatkowo, w celu zwiększenia stabilności wartości, odczytane i przeskalowane wskazania są obniżane o 10, przy czym nie mogą być mniejsze od 0. Program zakłada, że broker jest informowany o zmianie wartości napięcia na potencjometrze tylko, jeśli obecna różni się o więcej niż 4 od ostatnio wysłanej. Publikowane jest wskazanie potencjometru przeskalowane do wartości 0-255. Wynika to z niedokładności przetwornika analogowo-cyfrowego, co nawet przy braku ingerencji we wskazaniu potencjometrów może doprowadzić do kaskady publikowanych wiadomości, co jest szumem komunikacyjnym. W celu redukcji wpływu drgań styków, odczytanie ostatecznej wartości na przycisku dokonuje się z opóźnieniem 50ms względem momentu wcisnięcia. Broker informowany jest jedynie, że przycisk został wysłany.

### 3.4. System AR

#### O systemie

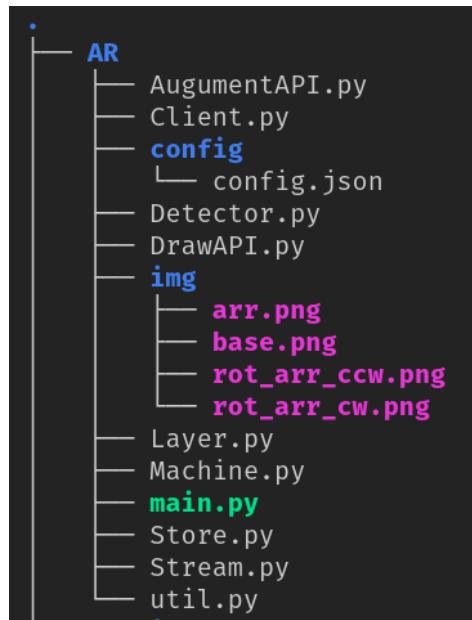
Przez system AR rozumie się zestaw współpracujących ze sobą skryptów napisanych w języku Python, który po włączeniu pobierze obraz z wybranego źródła i wyświetli zmodyfikowany obraz. Wszystkie pliki potrzebne do uruchomienia systemu znajdują się w folderze AR (rysunek 3.15).

Poza skryptami, których dokładny opis i wyjaśnienie działania przedstawione zostaną w kolejnym podrozdziale, w tej samej lokalizacji znajduje się folder z plikiem konfiguracyjnym, oraz obrazkami. Konfiguracja napisana została w pliku JSON, jej uproszczoną zawartość prezentuje Listing 1.

---

Listing 1: Plik konfiguracyjny użyty w projekcie

```
1  {
2      "server" : {
3          "protocol" : "mqtt",
4          "host" : "localhost",
5          "port" : 1883,
6          "topics" : [
```



Rysunek 3.15: Struktura plików w projekcie. Źródło własne

```

7     {
8         "topic": "arduino/red",
9         "name": "red",
10        "default": 0,
11        "type": "int"
12    },
13    { ... }
14 },
15 "machines": [
16     {
17         "name": "Test Machine",
18         "ref": "img/base.png",
19         "areas": {
20             "PB": [92, 144, 178, 233],
21             (...),
22         }
23     }
24 ]

```

---

Wydzielone zostały 2 główne sekcje:

- a) server - zawiera informacje wymagane do połączenia z brokerem MQTT, takie jak adres hosta, port, oraz listę tematów, które powinny być nasłuchiwanie. Każdy temat reprezentuje jedna tablica asocjacyjna, zawierające klucze dotyczące nazwy tematu po stronie mqtt, nazwy tematu po stronie aplikacji, oraz opcjonalnie domyślnej wartości i typu, na który powinna zostać przekonwertowana zmienna po odbiorze.
- b) machines - lista maszyn, które algorytm będzie próbował znaleźć. W chwili pisania niniejszej pracy algorytm jest dostosowany do rozpoznawania jedynie jednej maszyny, ale pozostawiono furtkę do późniejszego rozwinięcia. Każda maszyna w konfiguracji to tablica asocjacyjna, która zawiera nazwę maszyny, ścieżkę do obrazka referencyjnego, oraz listę par klucz-wartość, reprezentującą potencjalnie interesujące obszary na panelu maszyny. Para ma strukturę: <nazwa\_obszaru>: [x1, y1, x2, y2], gdzie wartości są parami współrzędnych kartezjańskich wyznaczonych z obrazu referencyjnego.

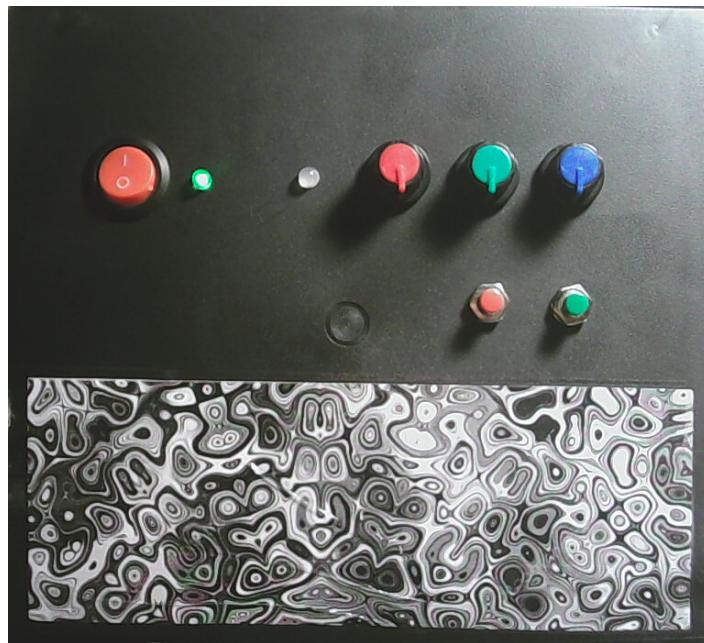
### Obraz referencyjny

Obrazem referencyjnym użytym w projekcie jest zdjęcie panelu urządzenia testowego, zrobionego za pomocą kamery webcam użytej do późniejszego testowania systemu. Rysunek 3.16 przedstawia tenże obraz, a 3.17 wyznaczone z niego punkty charakterystyczne. Zauważać można, że elementy użytkowe panelu nie pozwalają na wyznaczenie dużej liczby punktów, z tego też powodu do dolnej części panelu przyklejono wzór, który pozwala zauważalnie zwiększyć stabilność rozpoznawania.

### Stworzone skrypty

**main.py** - Główny skrypt systemu, w celu jest uruchomienia powinien zostać uruchomiony przez użytkownika. Dostępnymi parametrami są:

- t/-type - typ źródła strumienia wideo. Dostępne są: remote (po adresie IP), file (z pliku), cam (kamera połączona bezpośrednio z komputerem), image (jedyncza klatka). Domyslną wartością jest cam.
- s/-source - źródło strumienia, zależna od typu. Więc może to być kolejno: adres IP, ścieżka do pliku, id kamery, ścieżka do obrazka.



Rysunek 3.16: Obraz referencyjny. Źródło własne

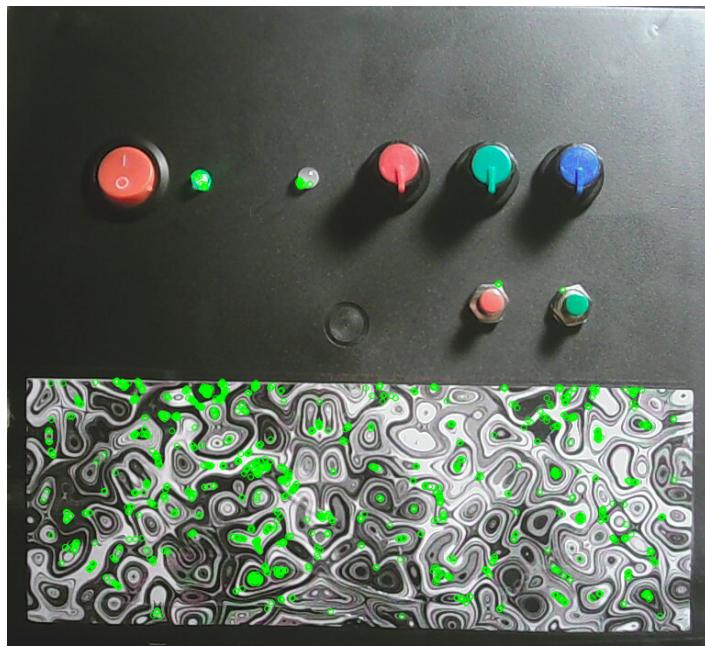
- -c/-config - ścieżka do pliku konfiguracyjnego. Domyślnie ..//config/config.json.

W samym pliku znajduje się również skrypt inicjujący działanie systemu. Odbywa się to w kilku krokach. Na początku tworzony jest obiekt strumienia video cap. Następnie tworzona jest instancja systemu, przyjmująca konfigurację JSON jako argument. Opcjonalnymi krokami są: obrysowanie rozpoznanego obiektu, oraz wyświetlanie liczby klatek na sekundę (FPS). Kolejnym krokiem jest wywołanie metody add\_layer, w której parametrem jest funkcja modyfikująca obraz z kamery. Ostatnim krokiem jest wywołanie metody run, która zaczyna pobierać obraz ze strumienia i go odpowiednio przetwarzać. Schemat tego działania przedstawia Listing 2.

Listing 2: Uruchomienie systemu

---

```
1 # Parse CLI arguments
2 args = prepare_arguments()
3
4 # Setup VideoCapture
5 cap = Stream(args.source, args.type)
6
7 # Init AugumentAPI instance based on config
8 augment = AugumentAPI(args.config)
```



Rysunek 3.17: Obraz referencyjny. Źródło własne

```
9 augment.outline_detection()  
10 augment.show_fps()  
11 augment.add_layer(draw_stats)  
12  
13 # Run  
14 augment.run(cap)
```

---

Funkcja rysująca przyjmuje trzy parametry. Pierwszym z nich jest frame, czyli obiekt klasy Frame, machine - tablica asocjacyjna z pliku konfiguracyjnego, oraz params - parametry z tematów MQTT. Funkcja ta musi zwrócić frame. Poniższy fragment (listing 3) rzeczywistej funkcji użytej w projekcie, ma za zadanie narysować biały prostokąt, oraz tekst "Urządzenie wylaczone" w odpowiednim miejscu na kadrze jeśli parametr `connected` ma wartość `OFF`. Docelowo funkcja rysująca jest jedynym miejscem, poza plikiem konfiguracyjnym, który późniejszy użytkownik powinien modyfikować. Funkcja rysująca wywoływana jest dla każdego kadru.

Listing 3: Uruchomienie systemu

---

```
1 if params["connected"] == "OFF":  
2     frame.draw_rectangle((10, 30), (170, 37))  
3     frame.draw_text("Urządzenie wylaczone", (20, 50))
```

---

**Stream.py** Skrypt zawiera klasę Stream, która tworzyinstancję obiektu strumienia video, który zostanie później użyty w projekcie. W zależności typu źródła podanego w parametrach głównego skryptu main.py, obiekt się delikatnie różni, jednak każdy wariant posiada metody: cap, release, read, aby cały system mógł działać poprawnie. Dodatkowo w przypadku wariantu z kamerą połączoną bezpośrednio z komputerem, skrypt ustawia odpowiednią rozdzielcość, aby ograniczyć zużycie zasobów. Klasa Stream przyjmuje 2 parametry w swojej funkcji inicjującej: source oraz source\_type, które odpowiadają parametrom –source i –type.

**Store.py** Zawiera definicję klasy Store, której zadaniem jest nadzorowanie stanu parametrów przekazywanych przez MQTT. Zawiera metody pozwalające na zmianowanie nazwy tematu na nazwę parametru w aplikacji, pobranie i ustawienie stanu. Funkcja inicjująca przyjmuje listę tematów z pliku konfiguracyjnego. Stan to tablica asocjacyjna zawierająca pary parametr-wartość. Listing 4 przedstawia proces stworzenia i ustawienie wartości dla parametru "test"

---

Listing 4: Uruchomienie systemu

```
1 topics = [{"topic": "test/topic", "name": "top", "default": 0}]
2 store = Store(topics) # {"top": 0}
3
4 store.set_state("top", 100)
5 store.get_state() # {"top": 100}
```

---

**Machine.py** Plik Machine.py zawiera klasę o tej samej nazwie. Jest to prosta klasa zawierająca w sobie informacje na temat pojedynczej maszyny: nazwę, obraz referencyjny oraz listę obszarów. Klasa zawiera pole statyczne all, pozwalające na śledzenie stanu rozpoznawanych maszyn. Zawiera publiczną metodę get\_area, która na podstawie nazwy obszaru zwróci jego współrzędne, a w razie potrzeby wywoła wyjątek. Posiada też pola name, areas i ref\_image, zawierające informacje o obiekcie. Listing 5 pokazuje proces tworzenia instancji klasy, jak i pobierania obszaru.

---

Listing 5: Stworzenie instancji Machine

```
1 config = {"name": "Maszyna", "ref": "test.jpg", "areas": {"A1": [0, 0, 100, 100]}}
```

```

2 machine = Machine(config)
3
4 machine.get_area("A1") # [0,0,100,100]
5 machine.get_area("Void")
6 # Traceback (most recent call last):
7 #   File "<stdin>", line 1, in <module>
8 #     File "/home/rafcio/Mgr/AR/Machine.py", line 28, in
9 #       get_area
10 #         raise ValueError("No area named" + label)
11 # ValueError: No area named VOID

```

---

**Detector.py** Detektor.py zawiera klasę Detector. Jej zadaniem jest znalezienie homografii pomiędzy obrazkiem referencyjnym, a kadrem z kamery. W ramach inicjującej przyjmowana jest struktura maszyny. Po pobraniu obrazka referencyjnego, tworzony jest deskryptor, domyślnie ORB, ustawiony na wykrywanie 1000 punktów kluczowych. **SPRAWDZIĆ**. Następnie za jego pomocą wyznaczane są punkty kluczowy oraz deskryptory obrazu referencyjnego, oraz przygotowywany jest matcher FLANN. Całość tego procesu przedstawia Listing ??

Listing 6: Stworzenie instancji Detector

---

```

1 class Detector(object):
2     def __init__(self, machine):
3         self.__machine = machine
4
5         # Reference image
6         ref_img = cv2.imread(machine["ref"])
7         self.__ref_image = ref_img
8
9         # Prepare descriptor
10        self.__descriptor = cv2.ORB_create(1000)
11
12        # Compute keypoints and descriptors for reference image
13        self.__kp, self.__desc = self.detectAndCompute(self.
14            __ref_image)

```

```

14
15     # Prepare matcher
16     self.__index_params = dict(algorithm = 6,                      #
17                               table_number = 12,
18                               key_size = 20,
19                               multi_probe_level = 1)
20     self.__search_params = dict(checks=300)
21     self.__matcher = cv2.FlannBasedMatcher(self.__index_params
22                                           , self.__search_params)

```

---

Klasa zawiera statyczne metody: calc\_good\_points, która filtruje znalezione dopasowania, oraz false\_detection, zwracającą krotkę charakterystyczną dla braku detekcji obiektu na obrazie. Publiczne metody to detect, przyjmująca kadr z kamery, a zwracająca wartość boolowską czy znaleziono, macierz homografii oraz maskę; get\_pts, zwracająca macierz zawierającą współrzędne narożników obrazka referencyjnego. Jest to przydatne podczas rysowania zgodnego z perspektywą urządzenia.

**Client.py** Plik zawiera klasę ClientMQTT, umożliwiającą sprawną komunikację z brokerem. Funkcja inicjująca posiada dwa opcjonalne argumenty: host oraz port. W przypadku ich braku domyślnymi wartościami są kolejno localhost"i 1883. Klasa umożliwia dodanie własnych wywołań zwrotnych (ang. callback) dla połączenia i rozłączenia z brokerem, w przypadku nowej wiadomości od brokera. Kluczową dla działania systemu jest metoda register\_handler, która przyjmuje nazwę tematu oraz funkcję zwrotną, wywoływaną, gdy przyjdzie nowa wiadomość z podanego tematu. Callback przyjmuje nazwę eventu oraz samą wiadomość. Handlery są dodawane automatycznie dla każdego topicu, dzięki czemu możliwa jest aktualizacja stanu aplikacji w czasie rzeczywistym.

Listing 7: Tworzenie instancji klienta

---

```

1 def handler(event, msg):
2     fn("New message on {} topic: {}".format(event, msg))
3
4 client = ClientMQTT(host="localhost", port=1883)
5

```

```
6 client.register_handler("topic", handler)
7 client.connect()
```

---

### DrawAPI.py Plik zawiera:

- a) Klasę T - zawiera typy możliwych transformacji na metodzie rysującej. W chwili obecnej są to: FOLLOW - nakładka na kadr nie jest transformowana geometrycznie, ale początek jej lokalnego układu współrzędnych (lewy góry narożnik) podąża za początkiem układu współrzędnych rozpoznanego obiektu; PERSPECTIVE - na nakładkę aplikowana jest homografia, przez co ma ona wspólną płaszczyznę z panelem urządzenia.
- 

```
1 class T:
2     """ Transform types """
3     FOLLOW=0
4     PERSPECTIVE=2
```

---

- b) Klasę Frame - semantycznie jest to obiekt kadru z kamery. W ramachinicjalizacji konieczne jest podanie obrazu, natomiast opcjonalnymi parametrami są: macierz homografii, maska homografii, oraz rozmiar obrazu referencyjnego.

Klasa Frame udostępnia metody:

- c) draw\_rectangle - rysuje prostokąt o zadanych: początku, rozmiarze, kolorze oraz transformacji. Domyślnie prostokąt jest biały oraz bez transformacji. Reszta parametrów jest obligatoryjna. Przykład:
- 

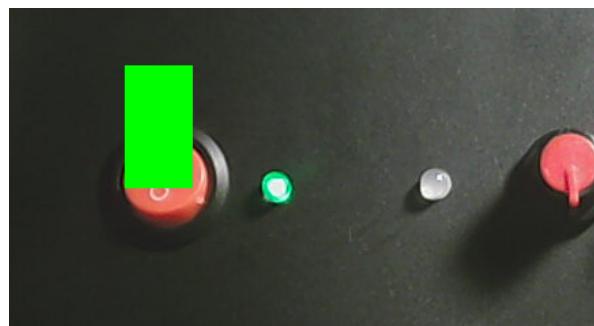
```
1 draw_rectangle((100,100), (50, 90), color=(255, 0, 0),
transform=T.PERSPECTIVE)
```

---

- d) draw\_polyline - rysuje wielokąt. Wymaganym parametrem jest lista krotek dwuelementowych, zawierająca współrzędne kolejnych wierzchołków. Opcjonalnymi parametrami są: kolor oraz grubość linii. Przykład:
- 

```
1 frame.draw_polyline([np.int32([(10,10),(30, 40), (70,
20), (90,100)])], color=(0, 255, 0))
```

---



Rysunek 3.18: Rysowanie prostokąta. Źródło własne



Rysunek 3.19: Rysowanie wielokąta. Źródło własne

- e) `draw_text` - rysuje na kadrze tekst o zadanej treści i współrzędnych. Wśród opcjonalnych parametrów są: krój fontu, rozmiar, kolor, grubość linii, sposób wygładzania linii oraz sposób transformacji. Przykład:

---

```
1 frame . draw_text ( "img / rot _ arr _ cw . png " , ( 20 , 50 ) , size = 0.5 )
```

---



Rysunek 3.20: Rysowanie tekstu. Źródło własne

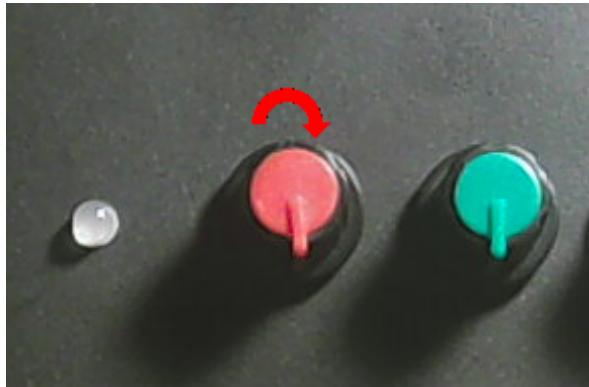
- f) `draw_image` - rysunek inny obrazek. Konieczne jest podanie drugiej obrazu, natomiast opcjonalne są: współrzędne (domyślnie punkt (0,0)), rozmiar (domyślnie

rozmiar obrazka) oraz sposób transformacji. Przykład:

---

```
1 frame.draw_image(path, (160, 400), (50, 50), transform=T.  
PERSPECTIVE)
```

---



Rysunek 3.21: Rysowanie obrazka. Źródło własne

g) get\_image - bezparametrowa funkcja zwracająca aktualny kadr, razem z ewentualnymi zmianami.

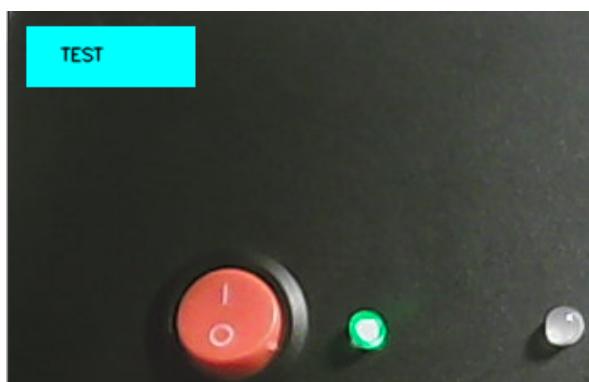
Dzięki temu, że każda z wymienionych metod zwraca całą instancję, możliwe jest wywołań i pisanie kodu w postaci:

---

```
1 frame \  
2   .draw_rectangle((10,10),(100,100), color=(255,255,0)) \  
3   .draw_text("TEST", (30, 30))
```

---

Co skutkuje narysowaniem na ekranie



Rysunek 3.22: Rysowanie prostokąta i tekstu. Źródło własne

**AugumentAPI.py** Zawarta w pliku klasa AugumentAPI jest jedną z 2 obiektów, które powinny zostać zainicjowane w skrypcie głównym main.py. Pierwszym z nich jest obiekt strumienia video, a drugim właśnie AugumentAPI, które ten właśnie strumień przyjmie w funkcji startu run. Dodatkowo podczas inicjalizacji obiektu należy podać ścieżkę do pliku konfiguracyjnego w formacie json, co przedstawia listing 8.

---

Listing 8: Tworzenie oraz uruchomienie systemu AR

---

```
1 augment = AugumentAPI(config)
2 augment.run(cap)
```

---

W ramach inicjalizacji skonfigurowane zostaną obiekty Detector, Client oraz Store, oraz zasubskrybowane będą tematy MQTT pobrane z pliku konfiguracyjnego. Klasa zawiera metodę add\_layer, w ramach której podaje się funkcję rysującą. Zostaje ona do pola \_\_layers. Wykorzystane jest ono w metodzie parse\_frame w ramach której na kadr nakładane są elementy stworzone w funkcji rysujących. Ponieważ wywoływanie podanych funkcji jest realizowane w ramach kolejki FIFO, możliwe jest tworzenie warstw jedna na drugiej. Może to być szczególnie pomocne podczas tworzenia bardziej skomplikowanych interfejsów.

Inne dostępne dla użytkownika metody to:

- a) stop() - zatrzymuje działanie systemu
- b) show\_fps() - pokazuje klatkaż w lewym górnym rogu ekranu
- c) outline\_detection() - rozpoznany obiekt zostaje obrysowany czerwoną linią

**util.py** Zawiera dodatkowe generyczne funkcje wykorzystywane w każdym z plików projektu. Zawiera między innymi metody na wyświetlanie informacji w terminalu w zależności od ich przeznaczenia: info, warning, error, success; przeskakuj obraz, aby jego rozmiar nie przekraczał maksymalnego, bez zmiany skali, odczytaj plik JSON i inne.

### 3.5. Komunikacja

Komunikacja odbywa się za pomocą protokołu MQTT, którego opis przedstawiono w jednym z poprzednich rozdziałów. Aby założenia projektowe mogły być spełnione, wyznaczono tematy do komunikacji

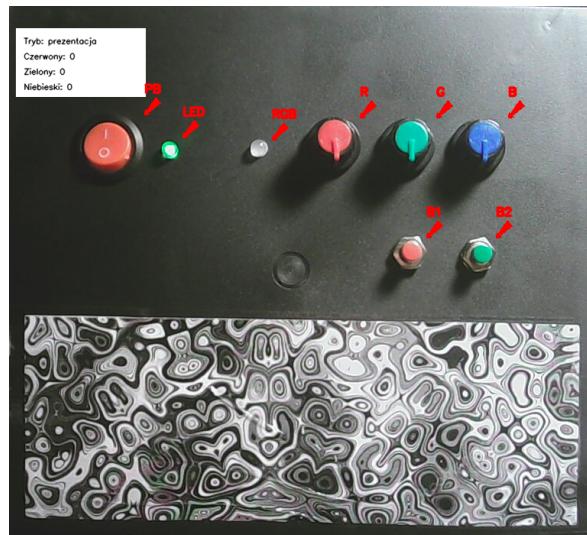
- 1) connected - temat informujący czy urządzenie ma aktywne połączenie z brokerem. Może przyjąć wartość "ON" bądź "OFF". Jest to specjalny kanał, ponieważ dzięki zastosowaniu techniki "testamentu", 5 sekund po przerwaniu połączenia z brokerem, temat przyjmie wartość "OFF".
- 2) arduino/red - zawiera składową czerwonego koloru. Przyjmuje wartości o 0 do 255 typu String. Jest to wartość potencjometru na rezystorze R1, z czerwoną gałką.
- 3) arduino/green - zawiera składową zielonego koloru. Przyjmuje wartości o 0 do 255 typu String. Jest to wartość potencjometru na rezystorze R2, z zieloną gałką.
- 4) arduino/blue - zawiera składową niebieskiego koloru. Przyjmuje wartości o 0 do 255 typu String. Jest to wartość potencjometru na rezystorze R3, z niebieską gałką.
- 5) arduino/mode - przyjmuje wartości od 0 do 2 typu String. Oznacza tryb, opisany w kolejnym podrozdziale. Wartość jest zwiększa o 1, bądź ustawiania na 0 po wciśnięciu przycisku B1.

### **3.6. Tryby działania**

W celu przeprowadzenia prezentacji, autor podjął decyzję o stworzeniu 3 trybów pokazowych, mających zademonstrować działanie systemu.

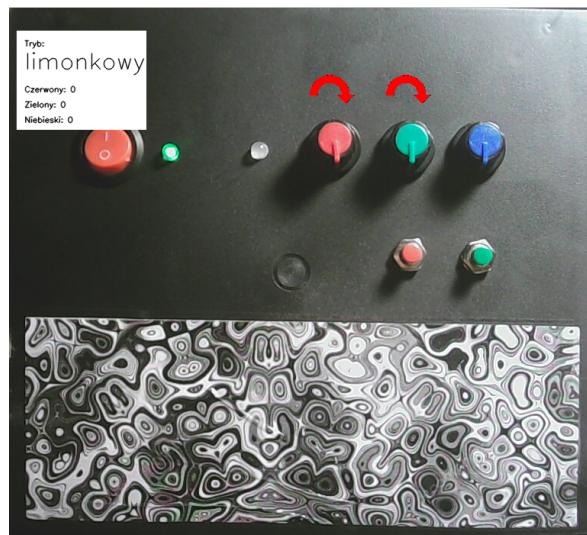
Pierwszym trybem jest tryb pokazowy. Za pomocą strzałek oznaczone są wszystkie obszaru opisane w pliku konfiguracyjnym dla każdej maszyny. Na strzałki nałożona jest homografia, wobec czego podążają one za obszarami oraz są one transformowane geometrycznie. Dodatkowo w lewym górnym narożniku ekranu rysowany jest obszar informacyjny, z wartościami liczbowymi składowych kolorów, a także aktualnym trybem.

Drugi i trzeci tryb mają analogiczne działanie. Otóż celem tego trybu jest pokierowanie użytkownika tak, aby dioda RGB świeciła na odpowiedni kolor. Składowe dla trybu drugiego (limonkowy) to (100, 200, 0), kolejno dla koloru czerwonego, zielonego i niebieskiego. Trzeci tryb nazwany został turkusowym, a pożądane składowe to (5, 85, 35). Ze względu na niedokładność potencjometrów i przetwornika analogowo/cyfrowego



Rysunek 3.23: Tryb prezentacji. Źródło własne

Arduino, przyjęto dokładność +/- 15. W przypadku, gdy dana składowa nie mieści się w pożądany obszarze, na obrazie rysowana jest strzałka skierowana w kierunku, w którym gałka potencjometru powinna zostać obrócona. Obrazuje to rysunek 3.24



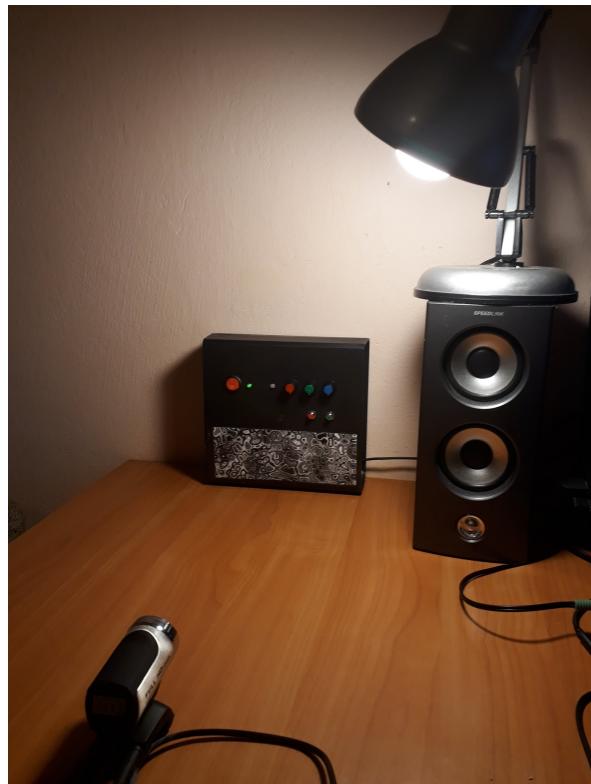
Rysunek 3.24: Tryb limonkowy. Źródło własne

## 4. Testy

W celu ustalenia sprawności i dokładności systemu rozpoznawania obiektu, autor podjął decyzję o wykonaniu testów dla różnych warunków. Testy, nie licząc testu odległości, wykonano w odległości 50cm kamery od urządzenia. Wykorzystana została

kamera internetowa Tecknet c018 FullHD oraz lampa symulująca różne warunki oświetleniowe. Stanowisko testowe zostało przedstawione na rysunku 4.25

Testy wykonano poprzez zwiększenie bądź zmniejszanie badanego parametru do momentu, w którym rozpoznawanie było stabilne. Każdy z punktów brzegowych został uwieczniony i przedstawiony w stosownym podrozdziiale w postaci zdjęcia sytuacji.



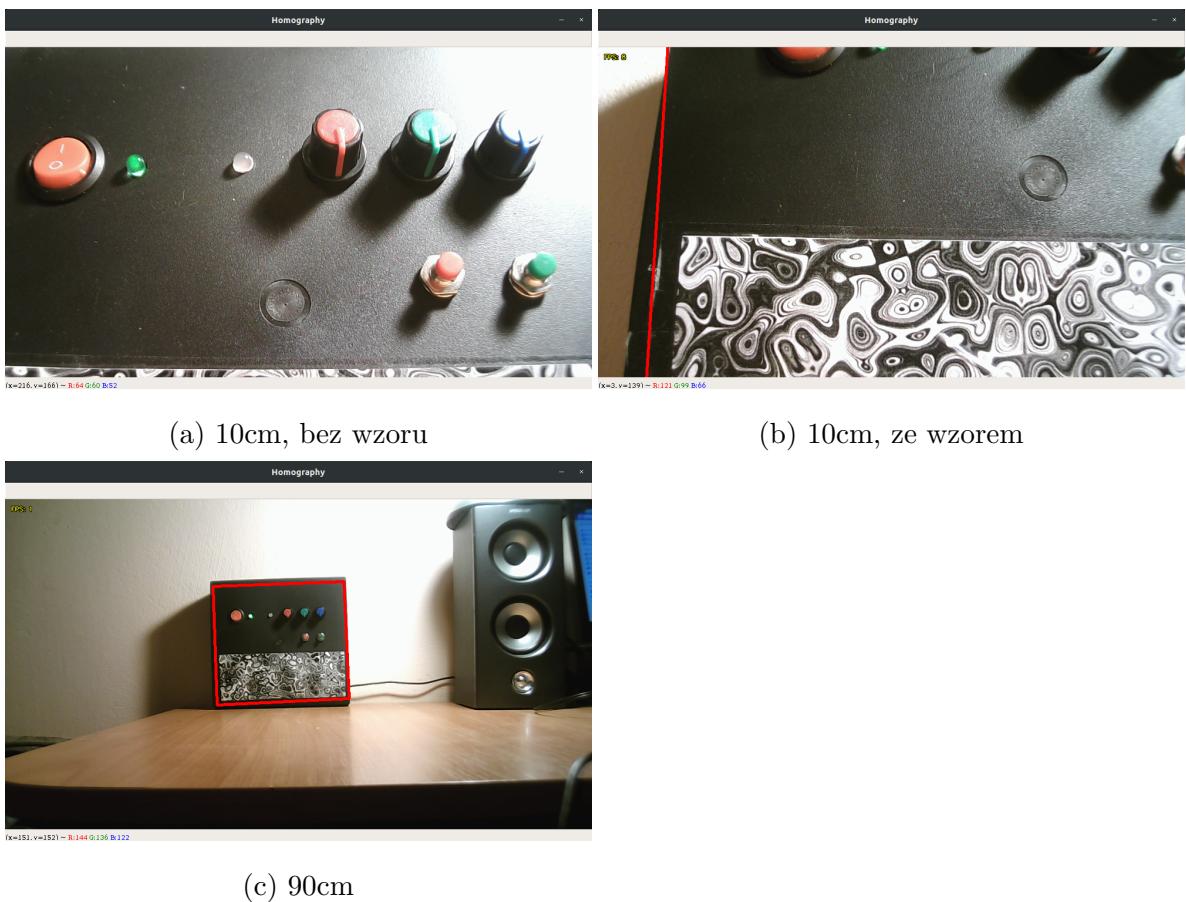
Rysunek 4.25: Stanowisko do testów. Źródło własne

Wykonane zostały:

- a) Test odległości
- b) Test kąta patrzenia
- c) Test różnych warunków oświetleniowych
- d) Test różnego natężenia oświetlenia
- e) Test obrotu kamery względem urządzenia
- f) Test częściowo widocznego urządzenia

## 4.1. Test odległości

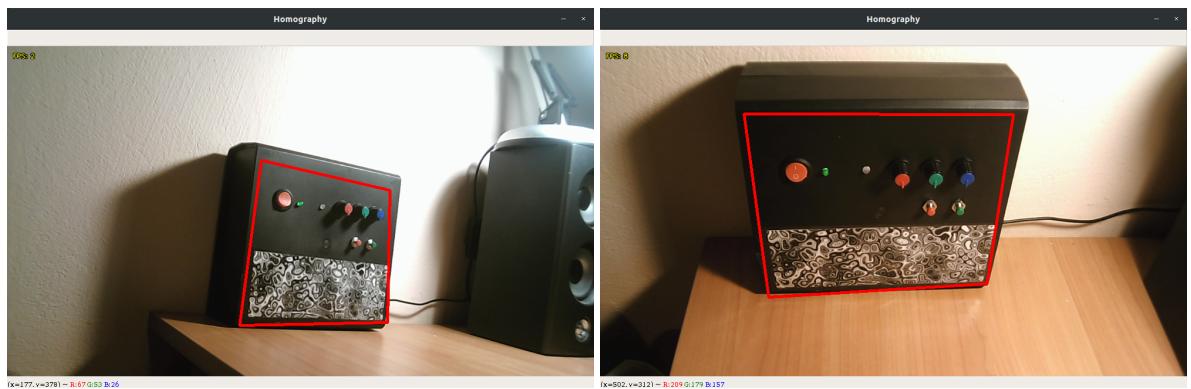
Testowano poprzez przesuwanie kamery wzdłuż osi głównej ustawionej prostopadle do płaszczyzny panelu urządzenia. Rozpoznawanie było stabilne w zakresie 10-90cm. W przypadku bliższego wariantu, fakt czy urządzenie zostało rozpoznane zależało od obecności wzoru na kadrze - jego brak powodował brak detekcji, patrz rys. 4.26a.



Rysunek 4.26: Testy odległości. Źródło własne

## 4.2. Test kąta patrzenia

Testy wykonano dla różnych katów pomiędzy osią główną kamery, a płaszczyzną panelu urządzenia, dla odległości 50cm. Wykonane próby dowiodły, że urządzenie jest poprawnie wykrywane do odchylenia około 45 stopni w dowolnej płaszczyźnie (rys. 4.27a, 4.27b).



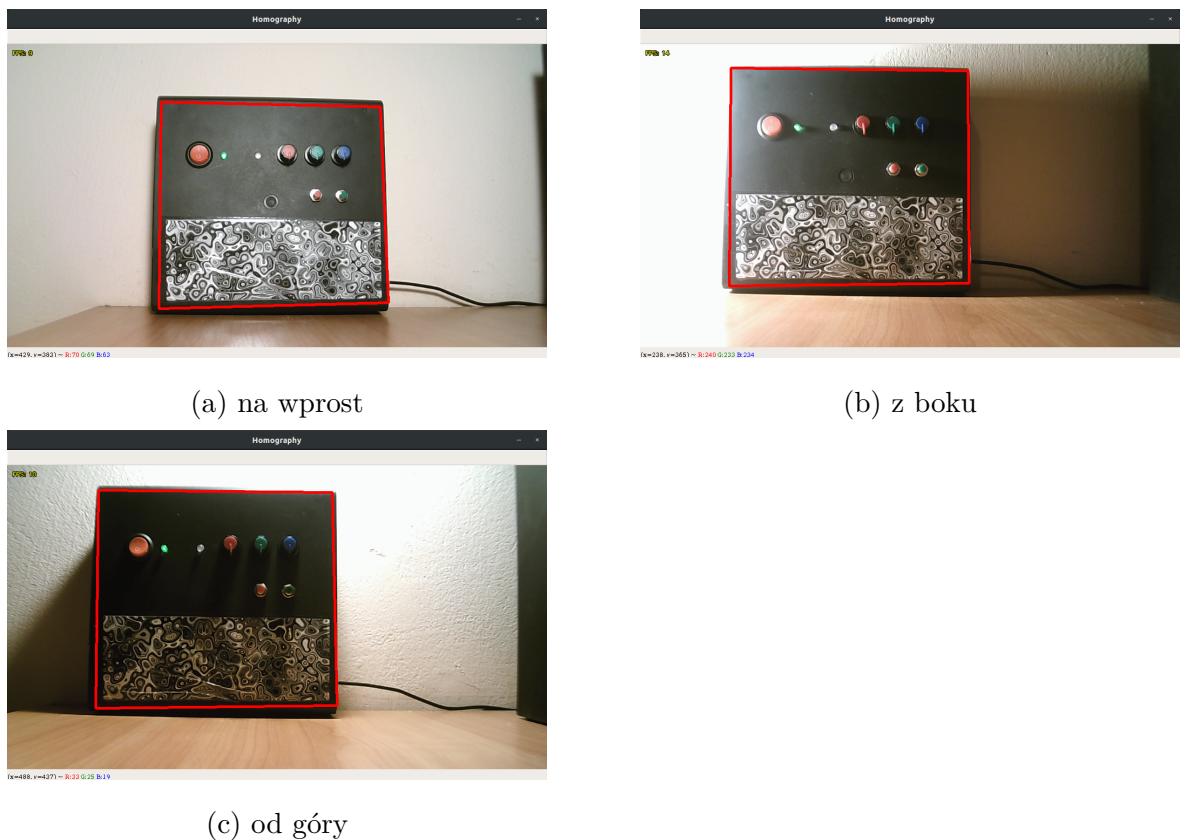
(a)  $45^\circ$  w płaszczyźnie poziomej

(b)  $45^\circ$  w płaszczyźnie pionowej

Rysunek 4.27: Test kąta patrzenia. Źródło własne

### 4.3. Test różnych warunków oświetleniowych

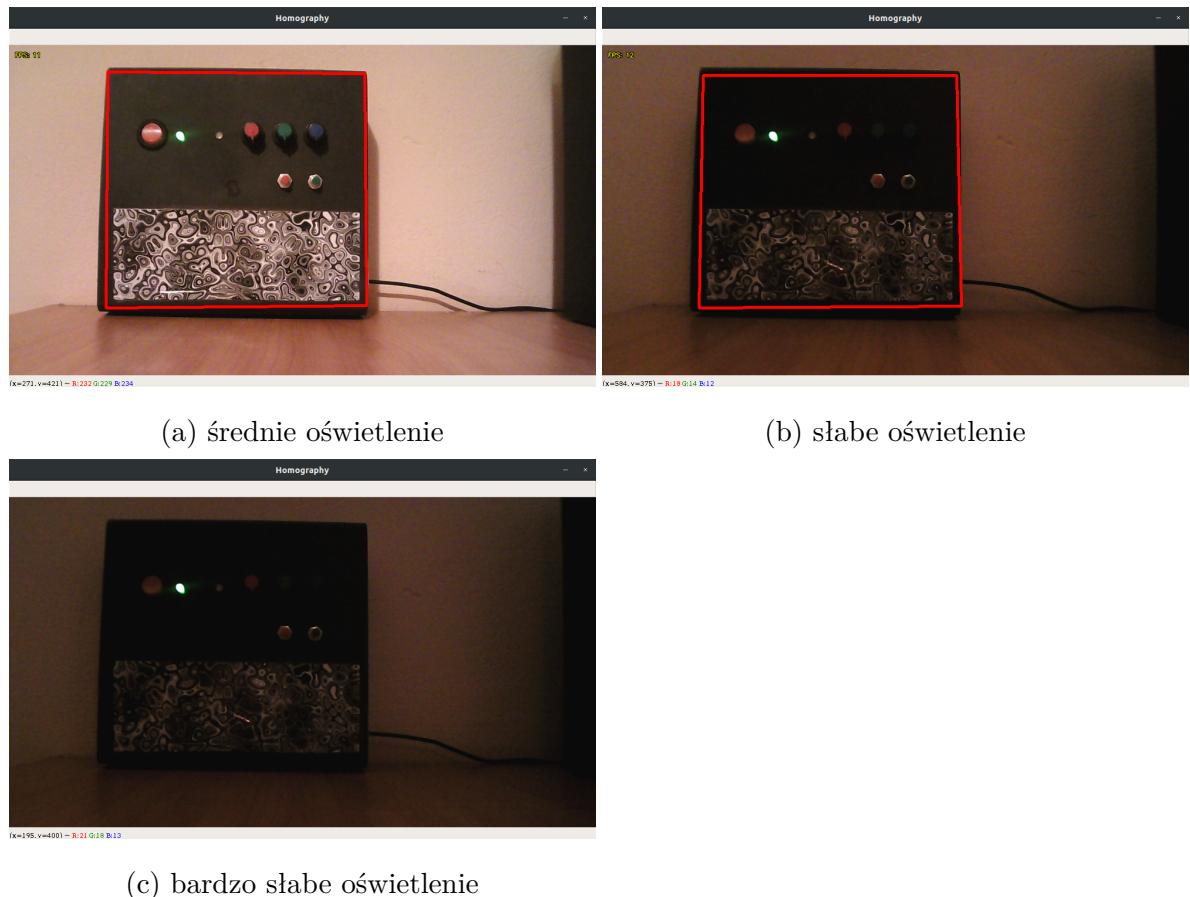
Poprzez warunki oświetleniowe rozumie się punktowe źródło światła skierowane z boku urządzenia. Celem było sprawdzenie, czy cienie rzucane przez wypukłe elementy panelu wpływają na detekcje.



Rysunek 4.28: Test różnych warunków oświetleniowych. Źródło własne

#### 4.4. Test różnego natężenia oświetlenia

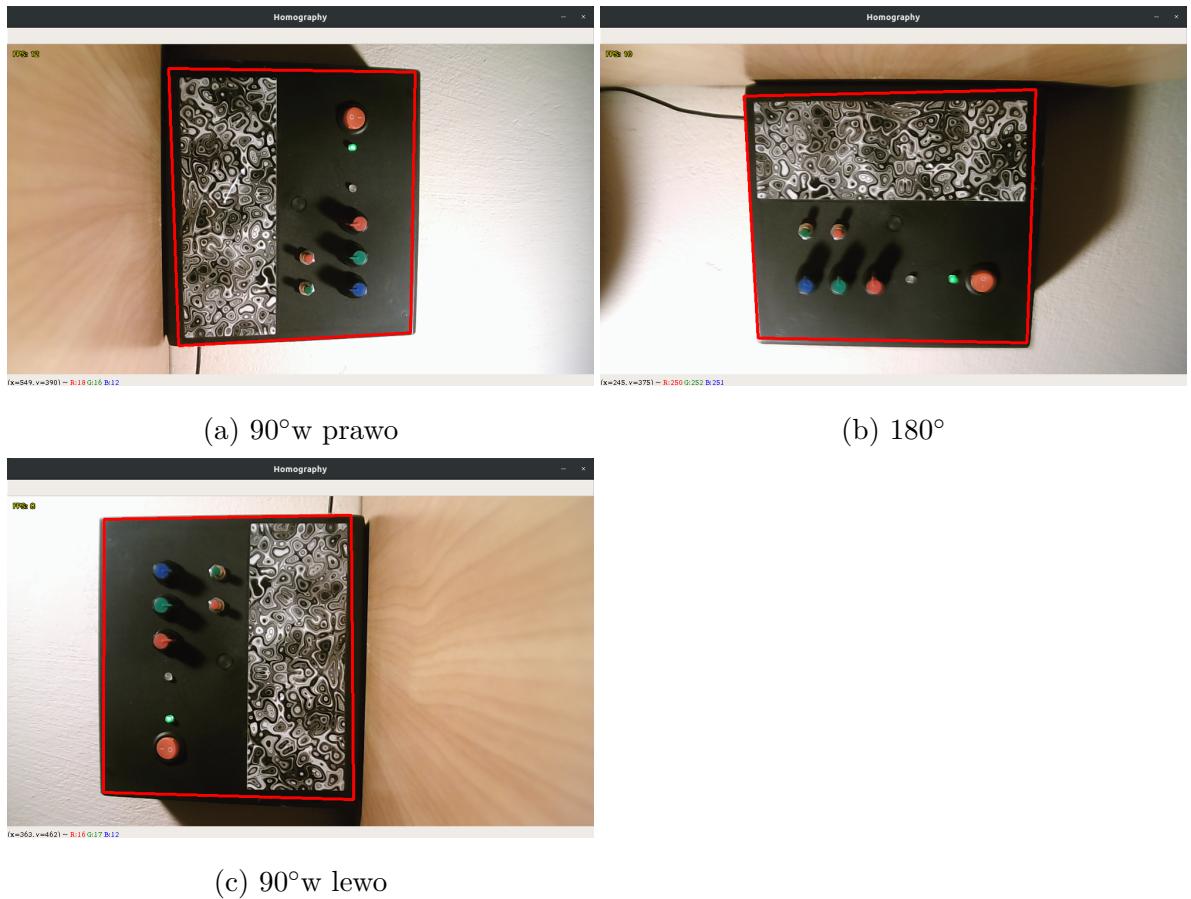
Celem testu było sprawdzenie jak system detekcji działa w warunkach słabego oświetlenia. Detekcje były poprawne średniego i słabego oświetlenia (rys. 4.29a, 4.29b). Dla oświetlenia bardzo słabego, czyli takiego dla którego elementy panelu nie były już wyraźnie, detekcja nie następowała (rys. 4.29c).



Rysunek 4.29: Test różnych natężeń oświetlenia. Źródło własne

#### 4.5. Test obrotu kamery względem urządzenia

Celem było sprawdzenie jak detekcja reaguje na obrót obrazu względem osi głównej kamery. Testowano poprzez obracanie kamery co  $90^\circ$ .

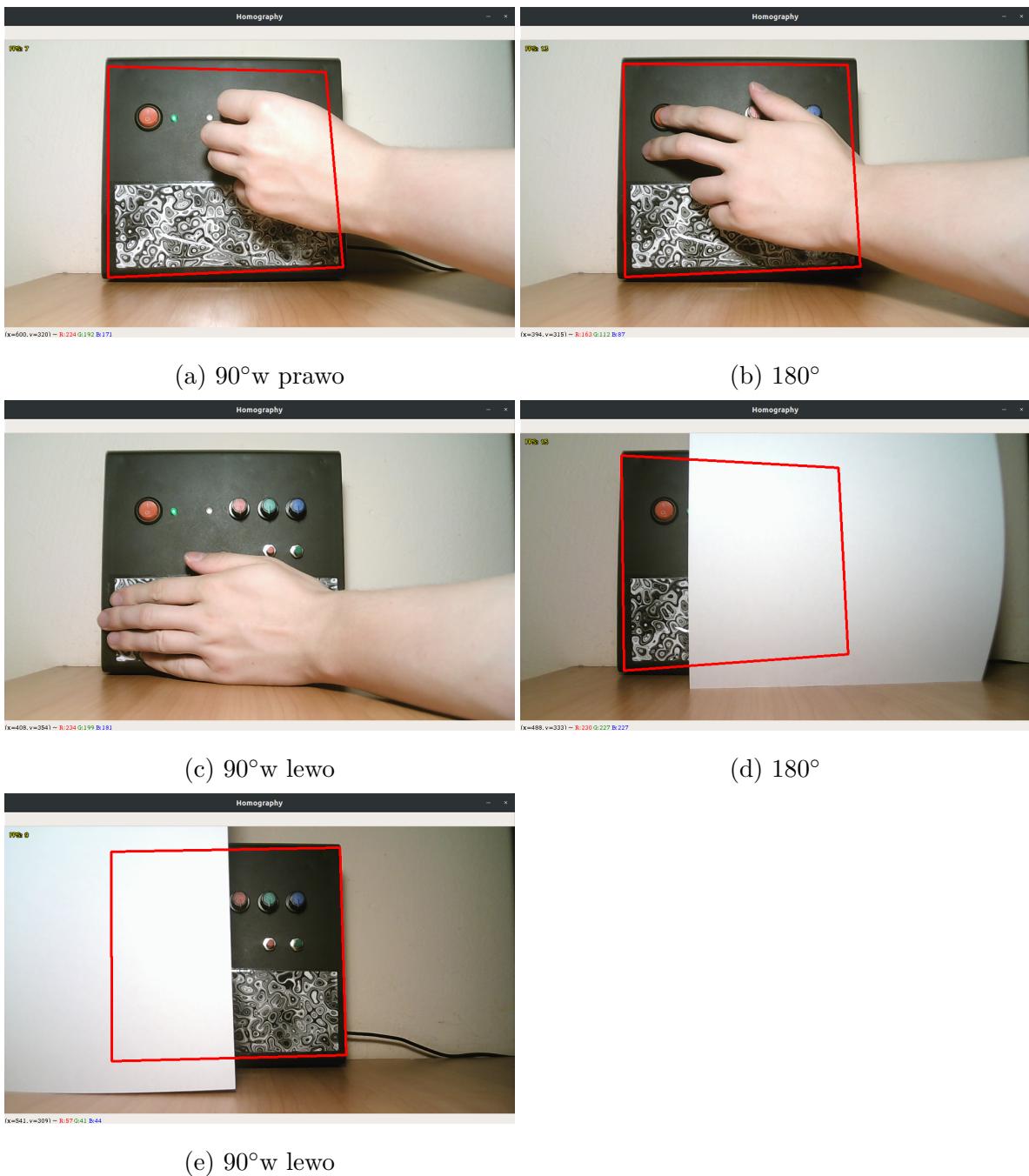


Rysunek 4.30: Test obrotu względem osi głównej kamery. Źródło własne

#### 4.6. Test częściowo widocznego urządzenia

Celem było przetestowanie zachowania algorytmu detekcji dla częściowo zasłoniętego panelu urządzenia. Wykonano łącznie pięć testów: dwa symulujące naturalne zachowania, to jest ludzka dłoń korzystająca z panelu, trzeci testował zachowanie przy niewidocznym wzorze, a dwa ostatnie sprawdzały procentowe zasłonięcie panelu.

Gdy cześć panelu została zasłonięta dlonią, detekcja była poprawna. W przypadku, gdy całość wzoru nie była widoczna, detekcja nie następowała. Dla dwóch ostatnich pomiarów, urządzenie było rozpoznawane jeśli conajmniej 25%-50% było widoczne.



Rysunek 4.31: Test częściowo widocznego urządzenia. Źródło własne

## 4.7. Podsumowanie i wnioski z testów

- W teście odległość, maksymalną wartość zwiększyć można korzystając z lepszej kamery, bądź opierając się na obrazie w większej rozdzielczości
- Wykorzystana kamera rozmazuje obraz w ruchu
- Obrót względem osi głównej nie wpływa na poprawność detekcji

- d) Urządzenie jest poprawnie rozpoznawane dla  $+/-45^\circ$ w dowolnej osi, wydaje się to być wartością wystarczającą do praktycznego wykorzystania systemu
- e) Rozpoznanie nie występuje dla bardzo słabego oświetlenia. Można to poprawić poprzez obróbkę obrazu przed detekcją, na przykład rozjaśnienie, bądź zwiększenie kontrastu, lub wykorzystanie lepszej kamery
- f) Pozycja źródła światła (a więc cienie) nie wpływa na poprawność detekcji
- g) Przy w większości zasłoniętym panelu występują problemy z rozpoznawaniem, szczególnie, gdy nie jest widoczny wzór. W praktyce, dla bardziej skomplikowanych panelów nie powinno być to aż tak uciążliwe, jednak poprawę sytuacji zapewni rozmieszczenie wzorów w różnych miejscach.

## **5. Podsumowanie i wnioski końcowe**

Wykorzystanie AR do celów użytkowania i obsługi urządzeń jest bez wątpienia przyszłością. Możliwości jakie zapewnia rozszerzenie widocznego obrazu pozwala na odciążenie pracowników, dzięki czemu będą oni popełniać mniej błędów, ponieważ niektóre decyzje podejmować może program. Dodatkowo ze względu na dostęp do danych w czasie rzeczywistym pozwala na przyspieszenie części zadań. Urządzenie stworzone na potrzeby pracy było w pełni zamknięte, nie posiadające zewnętrznych portów dostępowych, ani wyświetlacz. Jest to w niektórych przypadkach wymagane, jeśli urządzenie jest podane na warunki zewnętrzne, a pracuje w takich warunkach. Wtedy dostęp do niektórych funkcji może być utrudniony, jednak wyświetlanie odpowiednich parametrów przed oczami pracownika rozwiązuje ten problem.

Tryby, w których pracownik jest prowadzony za rękę przez specjalnie przygotowany tryb, pokazujący kolejne kroki procesu pozwoli zredukować czas uczenia, a także koszty z tym związane. Zamiast wysyłać serwisanta lub trenera na miejsce usterki czy też treningu, nawiązać można wideo rozmowę, gdzie obaj będą widzieć ten sam obraz, co pozwala na bezproblemową komunikację.

Udało się wykonać projekt, który spełnia wymagania postawione przez autora. Stworzony system poprawnie rozpoznaje obiekt podany na obrazie referencyjnym. Rozpoznawanie działa stabilnie dla typowych warunków, jakim jest światło o średnim natężeniu i kamera zwrócona na wprost panelu urządzenia. Cienie nie wypływają na poprawność detekcji, więc nie jest potrzebne specjalne przygotowanie środowiska wo-kół rzeczywistego urządzenia. Widoczność do 45 stopni wydaje się być wystarczająca do poprawnej pracy.

Polem do usprawnienia programu jest zastosowanie rzeczywistych okularów rozszerzonej rzeczywistości, dzięki czemu doświadczenie użytkownika będzie dużo lepsze niż na monitorze komputera. Kompromisem może być także aplikacja na urządzenia mobilne. Konieczne jest wykorzystanie lepszej kamery, ponieważ niedokładność zastosowanej pogarsza działanie systemu. Dodatkowa optymalizacja procesu pozwoli na zwiększenie FPS. Wykorzystane metody do tworzenia nakładek są prymitywne. Po-żądane jest wykorzystanie bibliotek typu OpenGL do tworzenia grafiki, dzięki czemu możliwe będzie komponowanie ładniejszych i czytelniejszych interfejsów, a także wykorzystanie animacji i modelów 3D.

## Literatura

- [1] F.Jiang *Artificial intelligence in healthcare: past, present and future*, 2017. Dostęp 20.06.2019. <https://svn.bmjjournals.org/content/2/4/230>
- [2] <https://www.wired.com/story/can-ai-be-fair-judge-court-estonia-thinks-so/>. Dostęp 20.06.2019.
- [3] <https://www.nvidia.pl/object/nvidia-hairworks-pl.html> Dostęp 15.06.2019.
- [4] <https://news.developer.nvidia.com/real-time-path-tracing-and-denising-in-quake-ii-rtx/>. Dostęp 15.06.2019.
- [5] T. Purcell, I. Buck, W. Mark, P. Hanrahan *Ray Tracing on Programmable Graphics Hardware* <https://graphics.stanford.edu/papers/rtongfx/rtongfx.pdf>. Dostęp 15.06.2019.
- [6] <https://store.steampowered.com/steamvr?l=polish>. Dostęp 15.06.2019.
- [7] <https://www.viar360.com/virtual-reality-market-size-2018/>. Dostęp 15.06.2019.
- [8] <https://www.viar360.com/virtual-reality-market-size-2018/>. Dostęp 15.06.2019.
- [9] <https://www.quora.com/What-is-an-umbrella-term-for-VR-AR-MR>. Dostęp 15.06.2019.
- [10] S. Chi-Yin Yuen, E. Johnson *Augmented Reality: An Overview and Five Directions for AR in Education*, 2011. <https://aquila.usm.edu/cgi/viewcontent.cgi?article=1022&context=jetde>
- [11] R. Azuma *A Survey of Augumented Reality*. Dostęp 20.06.2019. <https://www.cs.unc.edu/~azuma/ARpresence.pdf>
- [12] PTC *The State of Industrial Augmented Reality 2019*, 2019. <https://www.ptc.com/-/media/Files/PDFs/Augmented-Reality/State-of-AR-Report-2019.pdf>. Dostęp 15.06.2019.

- [13] <https://engine.vuforia.com/content/vuforia/en/case-studies/avatar-partners.html>. Dostęp 19.06.2019
- [14] <https://www.manufacturing.net/article/2016/10/case-study-augmented-reality-maintenance-technicians>. Dostęp 20.06.2019.
- [15] [https://docs.opencv.org/3.0-beta/doc/py\\_tutorials/py\\_feature2d/py orb/py\\_orb.html](https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py orb/py_orb.html). Dostęp 17.06.2019.
- [16] D.Lowe *Distinctive Image Features from Scale-Invariant Keypoints*, 2004. [https://people.eecs.berkeley.edu/~malik/cs294/lowe\\_ijcv04.pdf](https://people.eecs.berkeley.edu/~malik/cs294/lowe_ijcv04.pdf)
- [17] <https://insights.stackoverflow.com/survey/2019/#technology>. Dostęp 16.06.2019
- [18] <https://www.ics.uci.edu/~majumder/vispercep/cameracalib.pdf>. Dostęp 19.06.2019
- [19] <https://opencv.org/about/>. Dostęp 19.06.2019
- [20] A. Kaehler, G. Bradski *Distinctive Image Features from Scale-Invariant Keypoints*. O'Reilly, 2017.
- [21] <https://devenv.pl/mqtt-protokol-transmisi-danych-dla-iot/>. Dostęp 19.06.2019

POLITECHNIKA RZESZOWSKA im. I. Łukasiewicza  
Wydział Elektrotechniki i Informatyki

Rzeszów, 2019

**STRESZCZENIE PRACY DYPLOMOWEJ MAGISTERSKIEJ**  
**ROZSZERZONA RZECZYWIŚĆ WE WSPOMAGANIU**  
**OBSŁUGI URZĄDZEŃ**

Autor: Rafał Ileczko, nr albumu: EF-144087

Opiekun: dr inż. Mariusz Oszust

Słowa kluczowe: wizja komputerowa, rozszerzona rzeczywistość, internet rzeczy, opencv, ar

Celem pracy było stworzenie systemu rozszerzonej rzeczywistości, pozwalającego na komunikację z dowolnym połączonym z siecią urządzeniem w przypadku wykrycia go na obrazie z kamery. Wykonana aplikacja pozwala na komunikację w czasie rzeczywistym dzięki protokołowi MQTT.

RZESZOW UNIVERSITY OF TECHNOLOGY  
Faculty of Electrical and Computer Engineering

Rzeszow, 2019

**MSC THESIS ABSTRACT**

**AUGUMENTED REALITY IN SUPPORT OF DEVICE OPERATING**

Author: Rafał Ileczko, nr albumu: EF-144087

Supervisor: PhD Mariusz Oszust

Key words: computer vision, augmented reality, iot, opencv, ar

The purpose of this thesis was to develop an augmented reality system allowing to communicate with any external device connected to the network when detected in the camera's view. Developed application allow to real time communication via MQTT protocol.