

**WYDZIAŁ  
ELEKTROTECHNIKI  
I INFORMATYKI  
POLITECHNIKI RZESZOWSKIEJ**

**Rafał Ileczko**

Zastosowanie rzeczywistości rozszerzonej do wspomagania  
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	13
2.3. Zastosowanie AR w przemyśle	14
2.4. Sposoby zastosowań AR	15
2.5. Algorytmy detekcji i deskrypcji	17
2.5.1. FAST	17
2.5.2. BRIEF	18
2.5.3. ORB	18
2.5.4. SIFT i SURF	20
2.6. Dopasowanie cech	21
2.6.1. Brute-Force Matcher	21
2.6.2. FLANN based Matcher	22
2.7. Model kamery otworkowej	23
2.8. Homografia	25
2.9. Zastosowane technologie	26
2.9.1. OpenCV	26
2.9.2. Arduino	27
2.9.3. MQTT	28
2.9.4. Języki programowania	28
<b>3. Realizacja projektu</b>	<b>30</b>
3.1. Przyjęte założenia	30
3.2. Architektura rozwiązania	31
3.3. Urządzenie	32
3.4. System AR	36
3.5. Komunikacja	48
3.6. Tryby działania	48
<b>4. Testy</b>	<b>52</b>
4.1. Test odległości	53

4.2. Test kąta patrzenia . . . . .	54
4.3. Test różnych warunków oświetleniowych . . . . .	55
4.4. Test różnego natężenia oświetlenia . . . . .	56
4.5. Test obrotu kamery względem urządzenia . . . . .	56
4.6. Test częściowo widocznego urządzenia . . . . .	57
4.7. Test liczby punktów kluczowych . . . . .	58
4.8. Podsumowanie i wnioski z testów . . . . .	59
<b>5. Podsumowanie i wnioski końcowe . . . . .</b>	<b>61</b>
<b>Literatura . . . . .</b>	<b>63</b>



## Wykaz symboli, oznaczeń i skrótów

**AI** Artificial Intelligence

**AR** Augmented Reality

**DL** Deep Learning

**FLANN** Fast Library for Approximate Nearest Neighbors

**FPS** Frames per second

**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 24 godziny 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 ma za zadanie podnieść efektywność pracow-

ników obsługujących maszyny w zakładzie poprzez szybsze dostarczenie im informacji mających wpływ na przebieg danego procesu, oraz zasugerować konkretne działania prowadzące do wprowadzenia obiektu w odpowiedni stan. Jest to możliwe dzięki nałożeniu informacji o urządzeniu oraz sugestii wykonania konkretnych działań na obraz widziany przez operatora.

Odpowiednim rozwiązańem powyższych problemów wydaje się być Rozszerzona Rzeczywistość (ang. Augmented Reality - AR). Technologia 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 wąsy, bądź pogrubiając twarz. Innym częstym zastosowaniem są gry AR, gdzie najpopularniejszym przykładem jest Pokemon GO. Aplikacja wykorzystuje lokalizację użytkownika. Udostępniana jest mu mapa rzeczywistego świata, a w miejscu gdzie serwer aplikacji uzna, że występują pokemony pokazywana jest ikona. Użytkownik będący w tym odpowiedniej lokalizacji może uruchomić aplikację, i zobaczyć 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 wzroku 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.

Jednym z założeń niniejszej pracy jest połączenie urządzenia z internetem oraz możliwość wysyłania danych, ponieważ konieczna jest wiedza na temat parametrów urządzenia. Komunikacja ta powinna odbywać się w czasie rzeczywistym.

Autor za wkład własny uważa:

- Stworzenie oprogramowania pozwalającego na przeprowadzenie detekcji urządzenia na obrazie z kamery i określenie jego pozycji w przestrzeni rzeczywistej.
- Stworzenie komunikacji pomiędzy urządzeniem a systemem AR pozwalającej na przesyłanie parametrów.
- Wykonanie urządzenia elektronicznego opartego na płytce Arduino. Jego celem jest wysyłanie wartości parametrów urządzenia, co jest to niezbędne do poprawnego działania systemu. Do parametrów zalicza się: odchylenie potencjometru, oraz wcisnięcie przycisku. Dzięki temu system AR będzie wyświetlał użytkownikowi aktualne dane.

## **2. Wprowadzenie teoretyczne**

Rozdział ten zawiera wszystkie informacje potrzebne do zrozumienia zasady działania systemu. Na początku przedstawione zostały różne podejścia do modyfikacji rzeczywistości, ich założenia, oraz argumenty dla których to rozszerzenie zostało wybrane jako temat niniejszej pracy. Następnie wymienione zostały oraz opisane różne istniejące narzędzia AR. Podrozdział 2.3 opisuje przykłady zastosowań tych technologii w przemyśle na podstawie rzeczywistych wdrożeń w przedsiębiorstwach. Kolejna sekcja została poświęcona opisowi metod pozwalających na stworzenie systemu AR oraz argumentację przemawiającą za jedną z opcji. Reszta rozdziału poświęcona jest opisowi wykorzystanych metod oraz ich alternatyw i technologii, które zastosowano do tworzenia projektu.

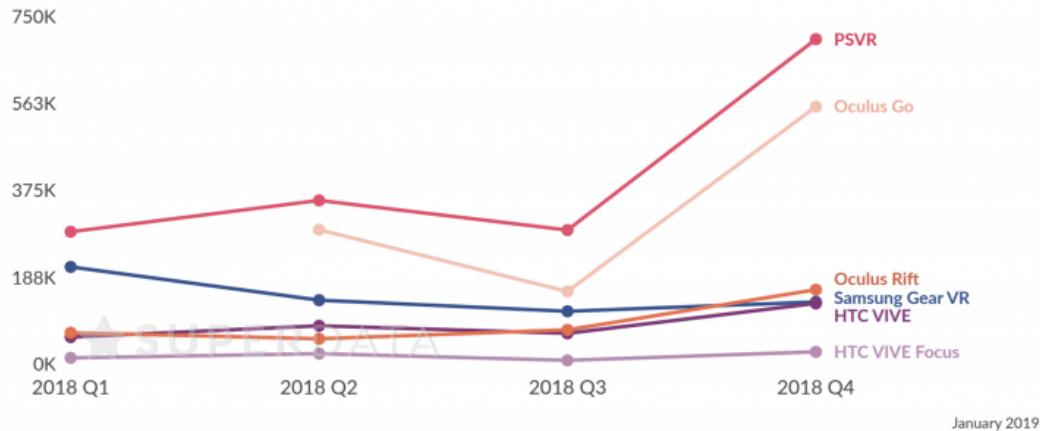
### **2.1. Wirtualna i Rozszerzona Rzeczywistość**

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 rozwiązaniem wydaje się więc wyjście ze środowiska płaskich ekranów na rzecz bardziej naturalnych doznań.

Pierwszym znaczącym krokiem w tym kierunku była publiczna zbiórka pieniędzy 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

## Virtual Reality Headsets

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



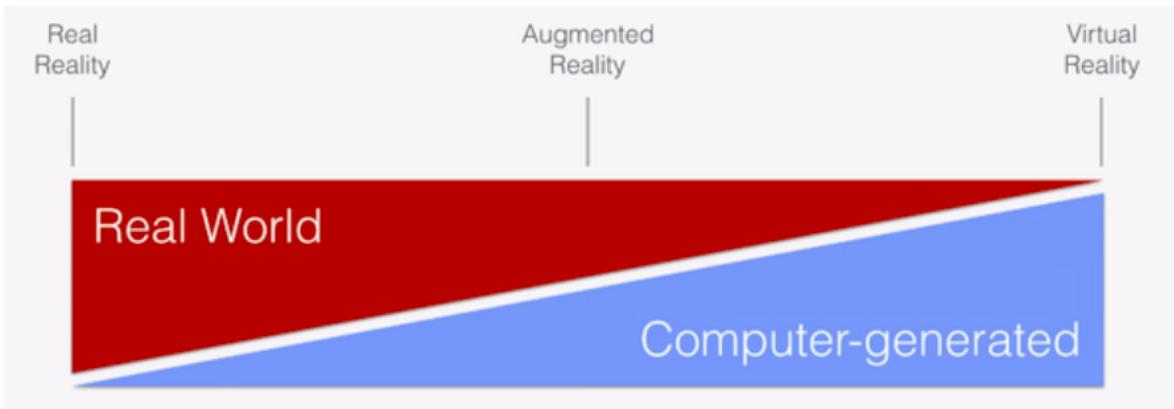
Rysunek 2.1: Zyski ze sprzedazy sprzetu VR w 2018 roku [8]

z otoczeniem niж jak ma to miejsce w symulacjach komputerowych wyświetlanych na typowym ekranie. W roku 2018 zyski firm z tytułu samego sprzetu VR przekroczyły 3,6 miliarda dolarów amerykańskich, co jest trzydziestoprocentowym wzrostem względem roku poprzedniego [7]. Zyski ze sprzedazy sprzetu 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).

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

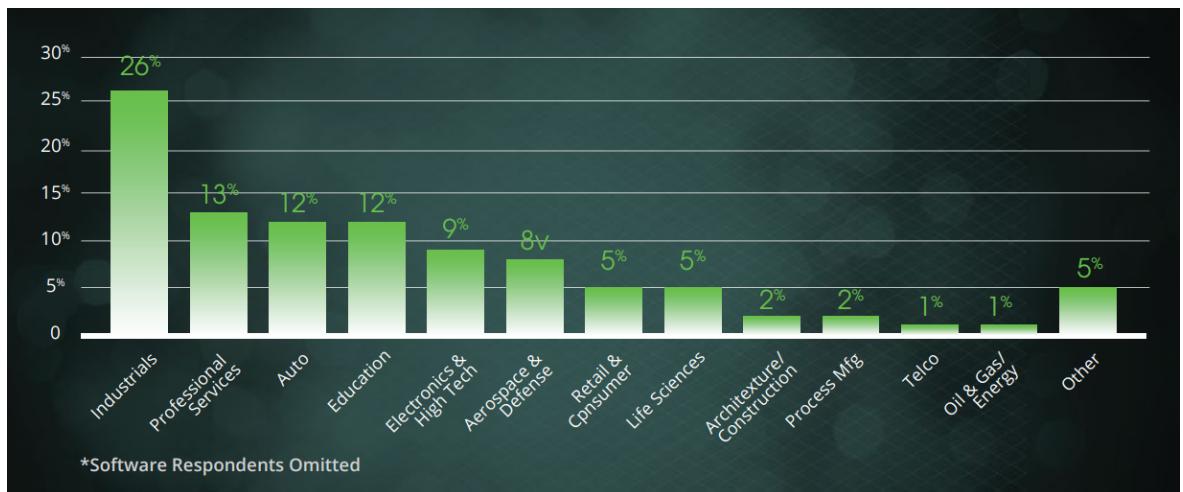


Rysunek 2.2: Spektrum rzeczywistości [9]

- Rejestruje świat w trzech wymiarach

W przeciwieństwie do wirtualnej rzeczywistości, rozszerzenie może objawiać się na wielu typach urządzeń. Najbardziej powszechnym jest smartfon oraz tablet, jednak najwygodniejszym rozwiązaniem są okulary, dzięki braku konieczności trzymania urządzenia przed oczami. 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 IKEA na smartfony. Pozwala ona, wykorzystując kamerę telefonu, wyznaczyć płaszczyznę podłogi pokoju, a następnie dzięki odpowiednim obliczeniom wyrenderować model 3D produktu we właściwie 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. Ich przykłady przedstawiono w rozdziale 2.3.



Rysunek 2.3: Procentowy udział rynku w adaptacji projektów AR

## 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.

### Google ARCore

Kolejna biblioteka stworzona przez giganta branży IT. Jej wykorzystanie jest możliwe w systemach Android oraz iOS i w silnikach Unity oraz Unreal. Doskonale nadaje się do tworzenia aplikacji AR na urządzenia mobilne, jednak wymagane jest uruchomienie aplikacji na urządzeniu mobilnym z systemem Android bądź iOS.

### 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 kilka innych, mniej rozbudowanych rozwiązań, jak EasyAR, ARToolKit czy Zapworks.

W momencie rozpoczęcie prac nad projektem dostępne narzędzia były albo ograniczone do konkretnych typów urządzeń (ARKit, ARCore), albo płatne (Vuforia, Wikitude). Z tego też powodu autor podjął decyzję o stworzeniu własnego systemu. Dopiero od niedawna Wikitude oferuje bezpłatny pakiet.

### **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. Wymagają również szeregu dokumentów raportujących co zaszło, aby można było przeprowadzić stosowne analizy.

Najbardziej prymitywnym sposobem na usprawnienie pracy dzięki technologii jest zaopatrzenie pracownika w różnego rodzaju elektronarzędzia lub tablet do sporządzania raportów i przeglądania instrukcji. 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 [13]. 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.

Firma Ingloba stworzyła dla Huawei Technologies narzędzie AR do wspomagania 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 najlepszym rozwiązaniem. Następcą prostych metod uczenia maszynowego jest uczenie głębokie

(ang. Deep Learning - DL). Te metody są już szeroko wykorzystywane w projektach wizji komputerowej. Dzięki fakty, że mogą wydobywać z obrazu unikalne cechy, zakres możliwości rośnie. Najpopularniejszym wykorzystaniem nauczonych modeli jest rozpoznawanie wielu klas obiektów na obrazie. [23]

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 jest czasochłonne. 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 głębockiego 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). Przewagą tej metody nad uczeniem głębokim jest fakt, że nie wymaga dużej mocy obliczeniowej, a dzięki możliwości dostosowania parametrów wykrywania cech, elastyczność rozwiązania jest większa. Dopasowanie cech wymaga wydobycia z obrazu unikalnych cech. Mogą być to: krawędzie, narożniki, tekstury, koła i okręgi. 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.

Na rysunku 2.4 przedstawiono przykładowy obraz z wyszczególnionymi sześcioma obszarami podpisanymi kolejnymi literami. Fragmenty A oraz B nie są unikatowe - bezchmurne niebo wygląda niemal identycznie w każdym miejscu, a wzór na ścianie budynku jest powtarzalny, więc ciężko dokładnie określić położenie tego fragmentu. C i D są bardziej szczegółowe, jednak krawędzie dachu w każdym miejscu wyglądają podobnie. Przykłady E oraz F natomiast przedstawiają narożniki, których położenie na obrazie można łatwo określić - to są przykłady cech.

Wyszukiwanie unikalnych fragmentów 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,



Rysunek 2.4: Potencjalne cechy [15]

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.

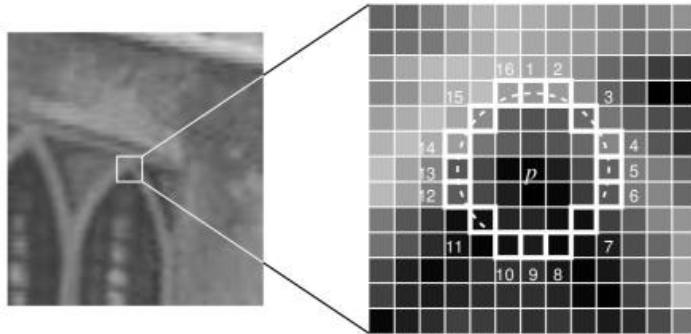
## 2.5. Algorytmy detekcji i deskrypcji

### 2.5.1. FAST

FAST jest detektorem cech zaproponowanym przez Edwarda Rostena oraz Toma Drummonda w pracy pod tytułem *Machine learning for high-speed corner detection*. Jego zasadę działania opisać można w kilku punktach:

- 1) Wybierz piksel  $p$ , który będzie sprawdzany, czy nadaje się na bycie punktem charakterystycznym. Jego jasność określona jest przez  $I_p$ .
- 2) Wybierz próg  $t$ .
- 3) Wybierz okrąg 16 pikseli wokół piksela testowanego.
- 4) Piksel  $p$  jest narożnikiem, jeśli istnieje  $n$  sąsiadujących pikseli w okręgu, których jasność jest większa niż  $I_p + t$  lub mniejsza niż  $I_p - t$ .  $n$  zwykle wynosi 12.
- 5) Zaproponowano również szybki test, aby wykluczyć dużą liczbę punktów nie będących narożnikami - testowane są jedynie 4 piksele: 1, 9, 5 i 13 (rys 2.5).

Najpierw sprawdzane jest czy 1 i 9 są odpowiednio jasne lub ciemne, następnie 5 i 13. Jeśli  $p$  ma szansę być narożnikiem, to co najmniej 3 z nich powinna być wystarczająco jasne lub ciemna. Jeśli ten warunek jest spełniony, testowane są wszystkie piksele w okręgu.



Rysunek 2.5: Przykład działania FAST [18].

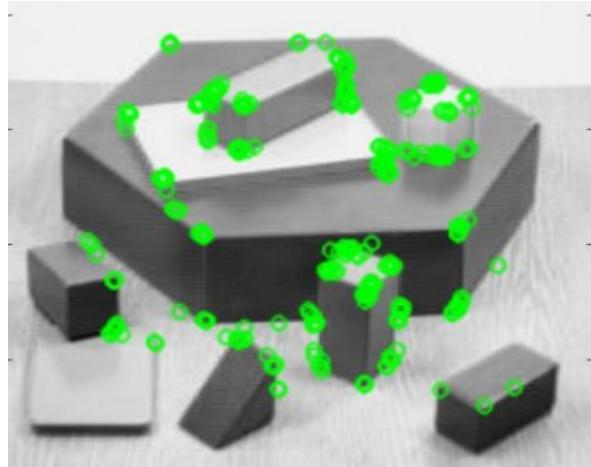
### 2.5.2. BRIEF

Jest to deskryptor zaproponowany w 2010 roku w pracy pod tytułem *BRIEF: Binary Robust Independent Elementary Features* [19]. Dzięki temu, że wykorzystuje wektory z wartościami binarnymi, zamiast zmiennoprzecinkowymi, jest szybszy niż SIFT czy SURF.

Punkty charakterystyczne są tak naprawdę małymi obszarami pikseli. BRIEF konwertuje je na wektory binarne, czyli deskryptory. Każdy deskryptor może mieć długość 128, 256 lub 512. Jednak ze względu na fakt, że operuje bezpośrednio na pojedynczych pikselach, jest podatny na szum. Dlatego też wykorzystuje się filtr Gaussa, aby wygładzić piksele i zredukować szумy. Następnie algorytm wybiera zestaw par współrzędnych  $n_d$ . Porównywana jest jasność pikseli w parach. Jeśli  $I(x) < I(y)$ , to zwróć 1, w przeciwnym razie 0. Porównanie to jest aplikowane do każdej pary, tworząc wektor wartości binarnych.

### 2.5.3. 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 [16]. Obie te techniki wyróżniają się niskimi wymaganiami mocy obliczeniowej. Autorzy tworząc to rozwiązanie podjęli się optymalizacji oraz zwiększenia odporności na zakłócenia.



Rysunek 2.6: Wykryte przez ORB cechy na przykładowym obrazie [21]

ORB na początku wykorzystuje FAST do znalezienia punktów, następnie najmniej przydatne z nich są filtrowane za pomocą metody detekcji narożników Harrisa. Orientacja narożnika jest opisywana poprzez wyznaczenie wektora przesunięcia "intensywnego centroidu", czyli momentów jasności pikseli w obszarze, od środka obszaru. Deskrypcja odbywa się za pomocą algorytmu BRIEF. Ponieważ jednak nie radzi sobie on z rotacjami, wykorzystywane są te już obliczone dla punktów charakterystycznych. Za pomocą orientacji cechy  $\theta$  obliczona zostaje macierz rotacji, która po zastosowaniu obraca  $S$ , tworząc  $S_\theta$ .

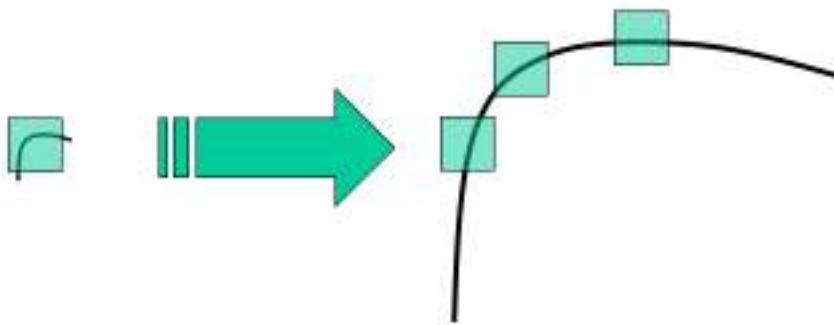
ORB dzieli cały obrót na obszary po  $2\pi/30$  (12 stopni) i tworzy tabelę dla wzorców BRIEF. Dopóki orientacja punktu  $\theta$  jest spójna pomiędzy obrazami, poprawny zestaw punktów  $S_\theta$  zostanie użyty do wyliczenia deskryptora.

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 pożądaną 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. W pracy *A Comparative Analysis of SIFT, SURF, KAZE, AKAZE, ORB, and BRISK* [20] porównano prędkości wielu algorytmów. ORB został uznany za najszybszy z nich, dlatego też został wykorzystany do realizacji projektu.

#### 2.5.4. 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 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 (rys. 2.7).



Rysunek 2.7: Porównanie narożników w różnej skali [22]

Rozwiązanie tego problemu opisał w 2004 roku D. Lowe z uniwersytetu Kolumbii Brytyjskiej, w artykule „Distinctive Image Features from Scale-Invariant Keypoints” [24], który poza wyznaczeniem punktów, oblicza również deskryptory. Na początku rozmiar obrazu jest podwajany za pomocą metod interpolacji. Następnie następuje iteracja rozmyjając rozmęcia Gaussa, z czego każdy kolejny obraz jest generowany ze zwiększoną odchyleniem standardowym. Następnie rozmiar obrazu jest zmniejszany dwukrotnie i występuje kolejna iteracja. Każda sekwencja iteracji pomiędzy pomniejszeniami obrazu nazywa się oktawą. 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 znajdują się za pomocą metody zwanej różnicą funkcji Gaussa. Dla każdego odpowiadającego sobie w ramach danej oktawy piksela wylicza się różnicę, a następnie wyznacza ekstremum.

Wyznaczone punkty są następnie filtrowane. Konieczne jest wykluczenie krawędzi-

dzi wśród wykrytych cech oraz punktów o niskim kontraście, 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 [28, 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. Ważną rzeczą w implementacji FLANN są jego parametry, które pozwalają skonfigurować zachowanie algorytmu. Pierwszy z nich to *index\_params*, będący strukturą informującą o tym jaka odbywać się będzie indeksowanie. Dostępne są:

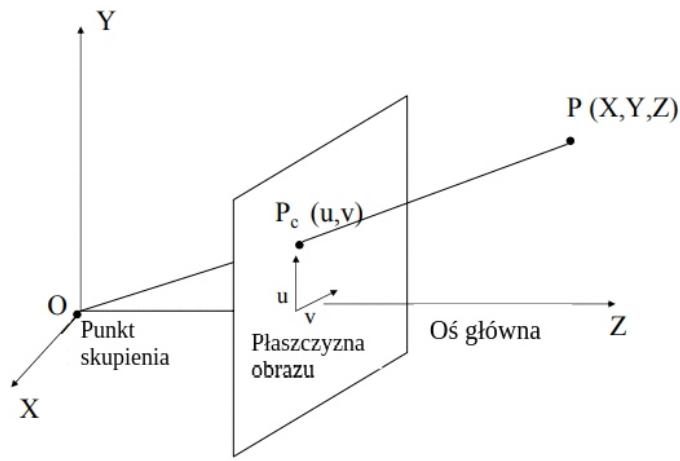
- 1) KD-tree - drzewa k-wymiarowe, które są wariantem drzew binarnych. Wykorzystuje się je do określenia najbliższych sąsiadów. Jest to domyślny algorytm używany we FLANN; można zdefiniować liczbę drzew.
- 2) Linear - wybranie tego algorytmu spowoduje działanie jak w przypadku Brute-Force, czyli porównywane są odległości pomiędzy poszczególnymi punktami.
- 3) K-means - inaczej algorytm k-średnich. Grupuje skupiska punktów, następnie dzieli je na podgrupy i tak dalej. Parametrami są: *branching*, czyli na ile grup podzielić dostępne punkty; *iterations* mówi ile iteracji podziału punktów na grupy należy wykonać; *cb\_index* kontroluje w jaki sposób środki grup są inicjowane.
- 4) Composite - łączy algorytmy KD-tree i K-means, jego parametrami są te ze wspomnianych algorytmów.
- 5) Locality-sensitive hash - wykorzystuje funkcje hashujące do opisania punktów. Parametry: *table\_number* mówi o rozmiarze tabeli hashującej; *key\_size* determinuje rozmiar klucza oraz *multi\_probe\_level* kontrolujący o tym jak odbywa się poszukiwanie sąsiadów.

Drugim, opcjonalnym argumentem FLANN jest *search\_params* mówiący o tym ile razy powiązania powinny zostać sprawdzone. Daje to większą precyzję, ale jest bardziej czasochłonne.

Metoda ta działa szybciej niż Brute-Force na dużych zbiorach i została wykorzystana w niniejszej pracy [28, 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 fotografię, przedstawiającą budynki rolnicze i niebo. Założenia modelu są stosowane do określenie położenia punktu w przestrzeni. Jest to kluczowe dla niniejszej pracy, ponieważ wymagane jest ustalenie położenia obiektu w przestrzeni względem kamery. Rysunek 2.8 prezentuje zasadę działania takiej kamery.



Rysunek 2.8: Geometryczny model kamery otworkowej [26]

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 powyż-

sze 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  $(c_u, c_y)$ , wtedy

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

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

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

$$\begin{pmatrix} u \\ v \\ w \end{pmatrix} = \begin{pmatrix} m_x * f & 0 & m_u * c_x \\ 0 & m_y * f & m_y * c_y \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} = \begin{pmatrix} \alpha_x & 0 & u0 \\ 0 & \alpha_y & v0 \\ 0 & 0 & 1 \end{pmatrix} * P = KP$$

Macierz  $\begin{pmatrix} \alpha_x & 0 & u0 \\ 0 & \alpha_y & v0 \\ 0 & 0 & 1 \end{pmatrix}$  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  $T(Tx, Ty, Tz)$ , a macierz rotacji obracająca kamerę do pozycji jak na rysunku 2.8 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

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

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.

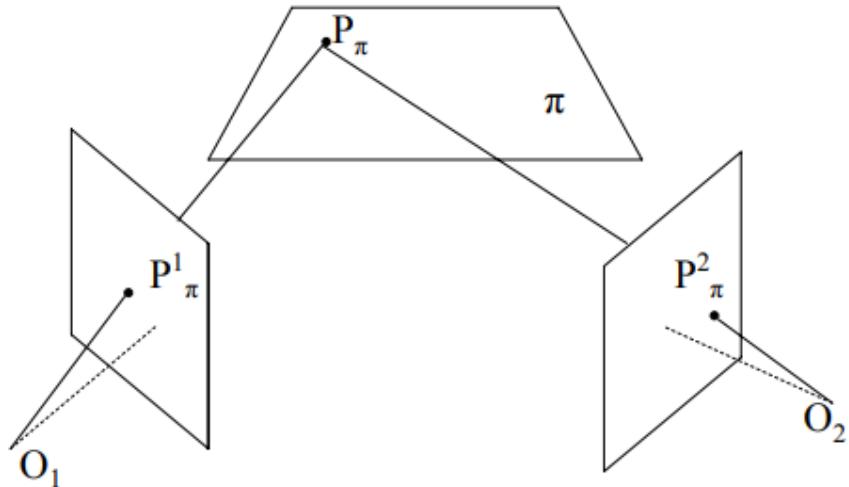
## 2.8. Homografia

Homografia jest koncepcją wykorzystaną w niniejszej pracy do określenia położenia oraz orientacji urządzenia w przestrzeni.

Jeśli dwie kamery są zwrócone w kierunku jednego punktu, powiązanie pomiędzy nimi może być w prosty sposób obliczone. Rysunek 2.9 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.



Rysunek 2.9: Przykład homografii [26]

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

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

więc

$$P_\pi = \begin{pmatrix} u_1 \\ v_1 \\ w_1 \\ \tau \end{pmatrix} = \begin{pmatrix} I \\ -N \end{pmatrix} P_\pi^1, \text{ gdzie}$$

$I$  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,

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

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

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

Macierz homografii pokazuje relację pomiędzy obrazem z dwóch kamer. Używając jej, możliwa jest transformacja widoku, aby odpowiadała położeniu drugiej [28, 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 włamywaczy w Izraelu, jak i monitoringu sprzętu górnictwa w Chinach.

Biblioteka ma interfejs 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 trwała. OpenCV zostało napisane w C++ [27].

Zawarte w bibliotece funkcje zostały wykorzystane do stworzenia systemu AR. Skorzystano zarówno z implementacji algorytmu ORB do detekcji i deskrypcji punktów, jak i FLANN do ich dopasowania, a także poszukiwanie homografii odbywało się za pomocą udostępnionych przez bibliotekę metod. Funkcjami nie związanymi bezpośrednio z rozszerzoną rzeczywistością, ale wykorzystanymi w projekcie są: wyświetlanie obrazu, rysowanie na obrazie (tekstu i kształtów), a także interfejs do pobierania strumienia wideo.

### 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, jak 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 do stworzenia urządzenia pokazowego. Zdecydowano się

na to konkretne rozwiązanie ze względu na jego prostotę użytkowania.

Domyślnym środowiskiem do tworzenia programów na Arduino oraz komunikacji 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 [29].

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.

Zdecydowano się na to konkretne rozwiązanie ze względu niskie zapotrzebowanie na transfer danych, szybkość i komunikację w czasie rzeczywistym oraz możliwość podziału wysyłanych danych na niezależne tematy.

### **2.9.4. Języki programowania**

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

Pierwszy z nich jest językiem prezentującym paradymat 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 zapro-

gramowano 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 paradigmaty obiektowy, imperatywny i strukturalny. Jest językiem interpretowanym i posiada system dynamicznych typów. Dzięki swojej prostej do zrozumienia składni znalazły zwolenników ze wielu dziedzin. Jest to obecnie czwarty najbardziej popularny język na świecie [25]. Jest on również bardzo wygodny przy korzystaniu z OpenCV i pozwala na bardzo szybkie tworzenie nowych rozwiązań, 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**

Aby projekt mógł zostać zrealizowany, poczyniono pewne uproszczenia i założenia.

Wysoka cena okularów rozszerzonej rzeczywistości uniemożliwiła ich wykorzystanie w projekcie, dodatkowo trudności w prototypowaniu na urządzeniach mobilnych spowodowały również odrzucenie tego typu medium. Autor zdecydował się więc na realizowanie systemu rozszerzonej rzeczywistości na 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 serworem 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 prostotę frontu urządzenia, podjęto decyzję o naklejeniu na nie kawałek papieru z naniesionym wzorem, aby zwiększyć dokładność detekcji. Problemy związane z rozpoznawaniem urządzenia w sytuacji gdy nie jest widoczny wzór prezentuje test w rozdziale 4.6.

System AR ma wspomagać użytkownika w obsługiwaniu maszyny poprzez tworzenie nakładki na obraz. Odbywa się to na dwóch płaszczyznach. Użytkownik jest informowany o obecnym stanie obiektu dzięki wyświetlaniu parametrów na kadrze, oraz, w zależności od wybranego trybu, może zobaczyć wyszczególniony każdy element maszyny z opisem, bądź ikony sugerujące konkretne działanie.

Obecne rozwiązanie wspiera rozpoznawanie jednego obiektu, jednak możliwy jest scenariusz, gdy będzie ich więcej. Zakładając, że każdy z obiektów jest skomunikowany z brokerem, różnice będą polegać na innym pliku konfiguracyjnym. Modyfikacja będzie się ograniczać do dodania kolejnej maszyny razem z jej obszarami, oraz na-

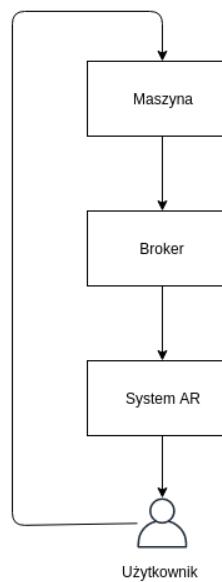
stępnych tematów do subskrybowania. Po stronie systemu AR modyfikacji wymaga jedynie przekazywanie maszyn do funkcji. Konieczne jest sprawienie, by miała ona jedynie dostęp do rozpoznanej na obrazie. Inaczej ma się sytuacja, gdy na jednym kadrze pojawią więcej niż jedno rozpoznawane urządzenie, jednak taki scenariusz nie mieści się w założeniach.

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 i zmiany konfiguracji, 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 trzech elementów: Maszyny, brokera, oraz systemu AR. Obrazuje to rysunek 3.10.



Rysunek 3.10: Architektura rozwiązania.

Komunikacja idzie w kierunku: maszyna => broker => system AR => użytkownik. Czynności, które użytkownik wykona na maszynie powodują wysłanie wi-

domości przez brokera do systemu AR, który wyświetla nowe dane użytkownikowi. Teoretycznie brokera w architekturze można się pozbyć a komunikację poprowadzić bezpośrednio maszyna => AR, na przykład w ramach systemu AR udostępniając API obsługujące zapytania HTTP. Wtedy maszyna wysyłałaby wiadomości bezpośrednio do AR. Jednak w przypadku, gdy pojawią się kolejne maszyny i systemy AR, komunikacja musiałaby się odbywać na zasadzie „każdy z każdym”. Bardziej skalowalnym rozwiązaniem jest więc połączenie każdego urządzenia jak i systemu AR z jednym brokerem. Dodatkowo jednym z założeń była możliwie największa elastyczność rozwiązania. Wiąże się to z tym, że powinna istnieć możliwość zamienienia brokera na 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.

### 3.3. Urządzenie

Jak wspomniano wcześniej, pełni ono 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, trzy potencjometry, dwustanowy przycisk (On/Off), oraz dwa przyciski typu pushbutton. Urządzenie zasilane jest poprzez przewód USB. Dokładny schemat połączeń przedstawia rysunek 3.12.

#### 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.



Rysunek 3.11: Zdjęcie urządzenia.

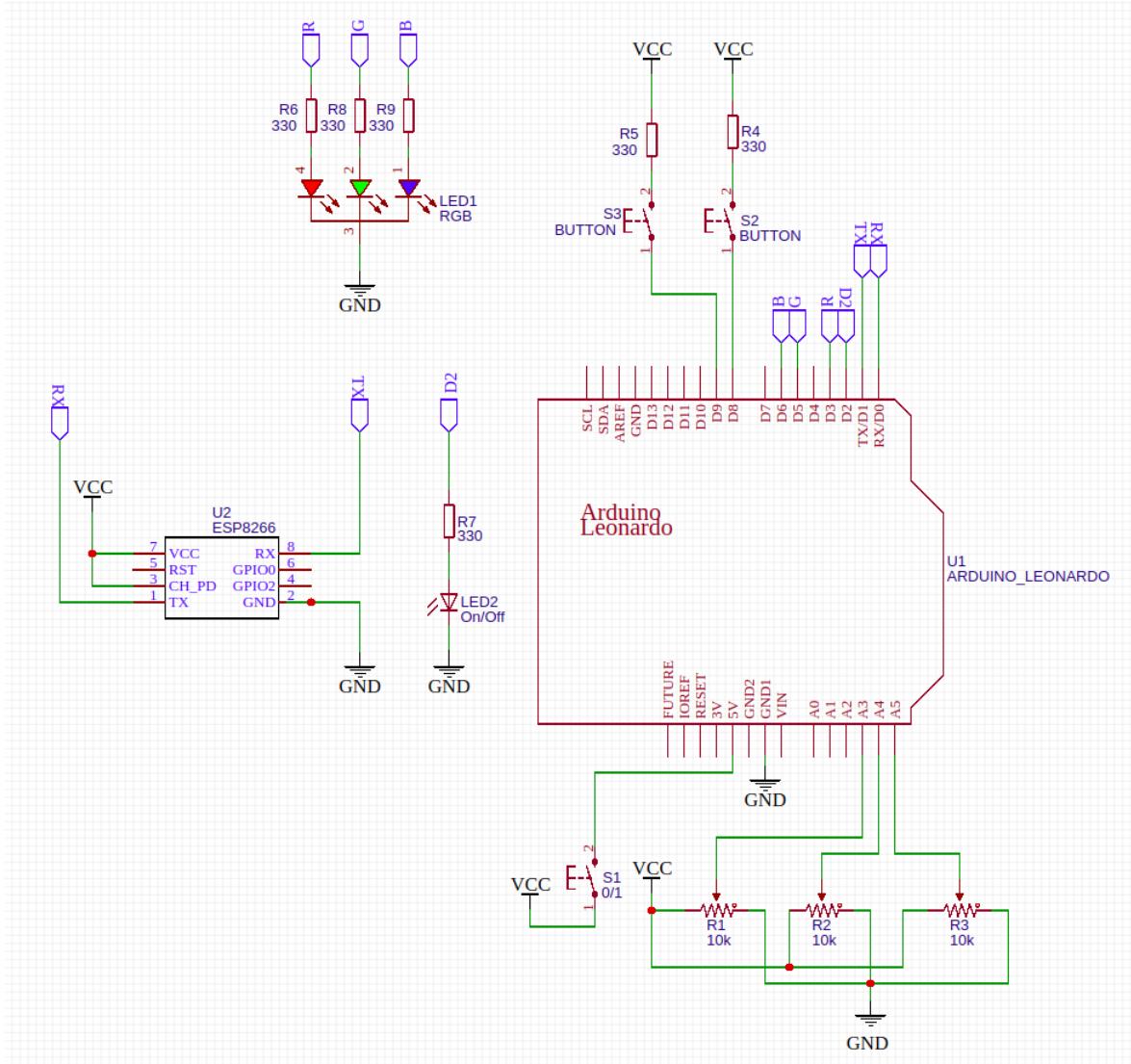
- d) Piny RX/TX zostały skrosowane i połączone z układem ESP8266.
- e) Dioda LED2 połączona do pinu D2 ma symbolizować uruchomienie układu.
- f) Przyciski połączone do pinów D8, D9 mają symulować zdarzenia , które pojawiły się w trakcie działania maszyny.
- g) Piny VCC oraz CH\_PD zostały połączone do źródła zasilania z Arduino.

### Struktura plików

Programy zawarte są w folderze arduino, w którym są dwa 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 sufiksem *\_tel* posiada dane do logowania do sieci WiFi udostępnianej przez telefon, a skrypt bez sufiksu - do sieci lokalnej.

### Omówienie programu

Każdy program pisany dla Arduino można podzielić na trzy sekcje:



Rysunek 3.12: Schemat połączeń.

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

Zawartość pierwszej sekcji dokładnie opisuje punkt a) powyższej listy. Do zrealizowania projektu wykorzystano trzy 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 typu pushbutton. Tworzony jest tutaj także klient MQTT. Stałe definiować można na dwa sposoby, poprzez



Rysunek 3.13: Struktura plików na urządzenie Arduino.

```
#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 z siecią WiFi.

Funkcja *loop* jest pętlą nieskończoną. Można z niej wyjść, natomiast spowoduje to przerwanie programu. W jej ciele odczytywane są wartości na pinach wejściowych, oraz ustawienie wartości na wyjściowych. Wysyłane są również wiadomości do brokera MQTT. Poziom napięcia na potencjometrach domyślnie mieścią się w zakresie 0-1023. Wynika to z rozdzielczości przetworników analogowo/cyfrowych na płytce, natomiast na potrzeby programu, wartości te zostały przeskalowane do 0-255. Ma to na celu ujednolicenie reprezentacji koloru z systemem szesnastkowym RGB, gdzie każda składowa ma wartości w zakresie 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. Wynika to z niedokładności przetwornika analogowo-cyfrowego, co nawet przy braku ingerencji użytkownika w wychylenie 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 wciśnięcia.

### 3.4. System AR

#### O systemie

Przez system AR rozumie się zestaw współpracujących ze sobą skryptów napisanych w języku Python, który połączeniu pobierze obraz z wybranego źródła i wyświetli zmodyfikowany obraz.

Algorytm działań realizowany od odebrania obrazu z kamery do jego wyświetlenia obrazuje rysunek 3.14. 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 dopasowanie cech z tymi z obrazu referencyjnego. Empirycznie uznano, że dobrą detekcję otrzymuje się już przy 50 dopasowanych cechach. Uznaje się, że cechy są dopasowane, jeśli istnieje 70-procentowa pewność, że tak. Jeśli odpowiednio silne dopasowanie nie zostanie znalezione, wyświetlany jest niezmieniony obraz. Jeśli jednak detekcja będzie pozytywna - poszukiwana zostaje macierz i maska homografii, oraz wywoływaną na kadrze jest funkcja rysująca. Tak zmodyfikowany kadr jest prezentowany użytkownikowi.

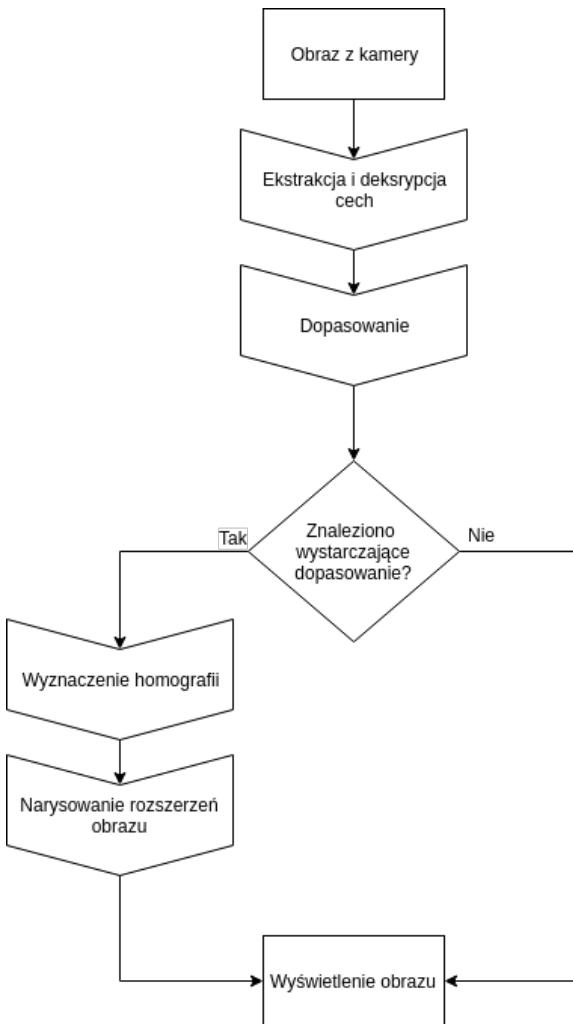
Wszystkie pliki potrzebne do uruchomienia systemu znajdują się w folderze AR (rysunek 3.15).

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

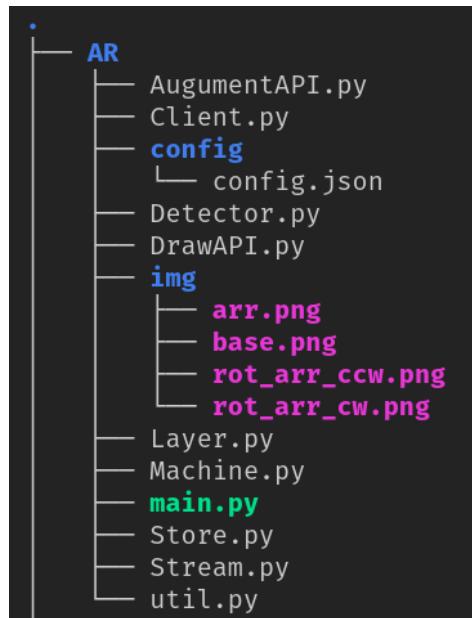
```
1  {
2      "server" : {
3          "protocol" : "mqtt",
4          "host" : "localhost",
5          "port" : 1883,
6          "topics" : [
7              {
```



Rysunek 3.14: Algorytm rozwiązania.

```

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": {
  
```



Rysunek 3.15: Struktura plików w projekcie.

---

```

20     "PB": [92, 144, 178, 233],
21     (...),
22 }
23 ]
24 }
```

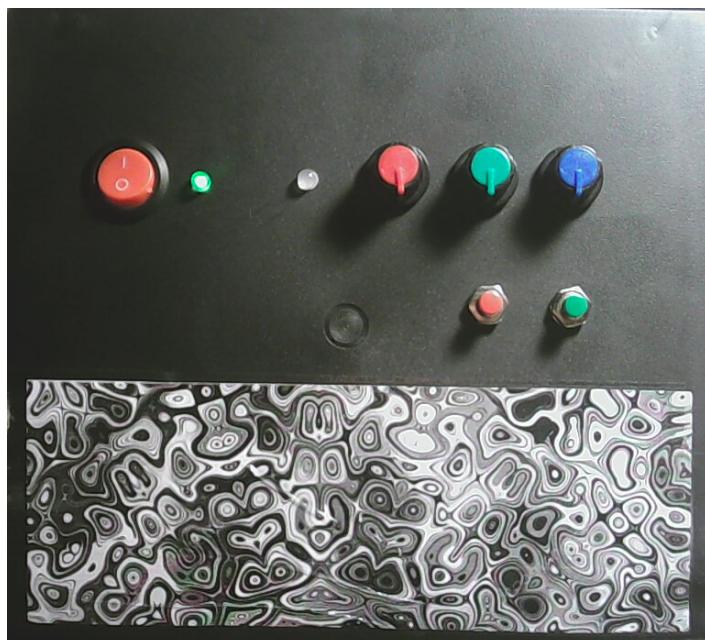
Wydzielone zostały dwie główne sekcje:

- server* - zawiera informacje wymagane do połączenia z brokerem MQTT, takie jak adres hosta, port, oraz listę tematów, które powinny być nasłuchiwane. Każdy temat reprezentuje jedna tablica asocjacyjna, zawierające klucze dotyczące nazwy tematu po stronie brokera MQTT, nazwy parametru po stronie aplikacji, oraz opcjonalnie domyślnej wartości i typu, na który powinna zostać przekonwertowana zmienność po odbiorze.
- machines* - lista maszyn, które algorytm będzie próbował znaleźć. W chwili pisania niniejszej pracy algorytm jest dostosowany do rozpoznawania jedynie jednej maszyny, ale pozostało 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 dwiema 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 - gdy uruchamiano system rozpoznawania urządzenia bez wzoru, często wyznaczona orientacja urządzenia nie pokrywała się z rzeczywistością. Może to być związane z wypukłością niektórych elementów urządzenia.

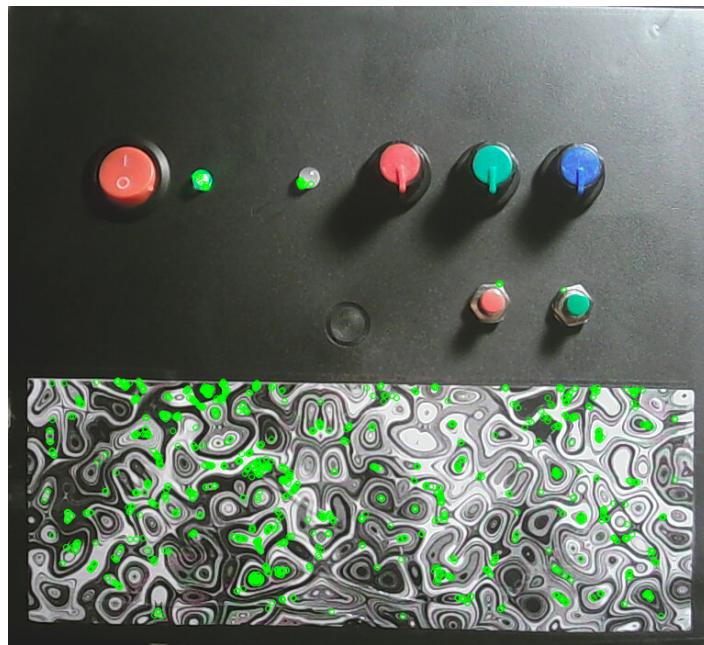


Rysunek 3.16: Obraz referencyjny.

### Stworzone skrypty

**main.py** - Główny skrypt systemu, 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* (pojedyncza klatka). Domyślną wartością jest *cam*.



Rysunek 3.17: Wyznaczone punkty kluczowe dla obrazu referencyjnego.

- *-s/-source* - źródło strumienia, zależna od typu. Więc może to być kolejno: adres IP, ścieżka do pliku, id kamery, ścieżka do obrazka.
- *-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 wideo *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*, 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)
9 augment.outline_detection()
10 augment.show_fps()
11 augment.add_layer(draw_stats)
12
13 # Run
14 augment.run(cap)

```

---

Funkcja rysująca jest stosowana do tworzenia rozszerzeń obrazu. Przyjmuje trzy parametry. Pierwszym z nich jest *frame*, czyli obiekt klasy *Frame*, *machine* - tablica asocjacyjna z pliku konfiguracyjnego, oraz *params* - parametry tematów MQTT. Funkcja ta musi zwrócić *frame*. Poniższy fragment (listing 3) rzeczywistej funkcji użytej w projekcie, ma za zadanie narysować biały prostokąt w rogu ekranu, oraz tekst „Urzadzenie wylaczone” w odpowiednim miejscu na kadrze jeśli parametr „connected” ma wartość „OFF”. 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("Urzadzenie 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 dwa parametry w swojej funkcji inicjującej: *source* oraz *source\_type*, które odpowiadają parametrom *-source* i *-type* z wywołania skryptu w terminalu.

**Store.py** Zawiera on definicję klasy *Store*, której zadaniem jest nadzorowanie stanu parametrów przekazywanych przez MQTT. Zawiera metody pozwalające na zmianę 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** Zawiera on definicję klasy o tej samej nazwie. Zawiera ona 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 też 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]
```

---

**Detector.py** Plik 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 model ORB, ustawiony na wykrywanie 4000 punktów kluczowych. Testy, które doprowadziły do wybrania tej liczby zostały opisane w rozdziale 4.7. Następnie za jego pomocą wyznaczane są punkty kluczowe oraz deskryptory obrazu referencyjnego, oraz przygotowywany jest matcher FLANN. Całość tego procesu przedstawia Listing 6.

---

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(4000)
11
12        # Compute keypoints and descriptors for reference image
13        self .__kp, self .__desc = self .detectAndCompute( self .
14                                         __ref_image )
15
16        # Prepare matcher
17        self .__index_params = dict( algorithm = 6 ,           #
18                                     FLANN_INDEX_LSH
19                                     table_number = 12 ,
20                                     key_size = 20 ,
21                                     multi_probe_level = 1 )
22
23        self .__search_params = dict( checks=300 )
24
25        self .__matcher = cv2.FlannBasedMatcher( self .__index_params
26 , self .__search_params )

```

---

Klasa zawiera statyczne metody: *calc\_good\_points*, która filzuje znalezione dopasowania, oraz *false\_detection*, zwracającą krotkę charakterystyczną dla braku detekcji obiektu na obrazie. Publiczne metody to *detect*, przyjmująca kadr z kamery, a zwracającą wartość boolowską (czy znaleziono obiekt na obrazie), macierz homografii oraz maskę; *get\_pts*, zwracającą 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 rozłączenie z brokerem, oraz 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. Funkcja zwrotna 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 ramach inicjalizacji konieczne jest podanie obrazu, natomiast opcjonalnymi parametrami są: macierz homografii, maska homografii, oraz rozmiar obrazu referencyjnego.

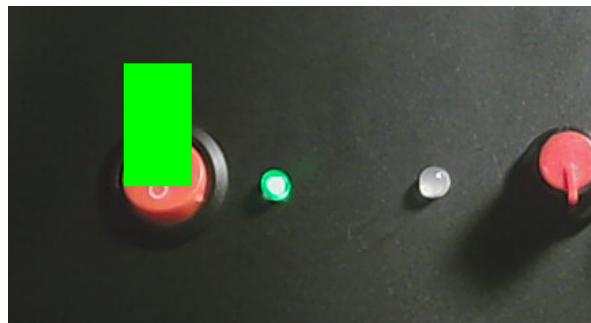
Klasa *Frame* udostępnia metody:

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

---

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

---



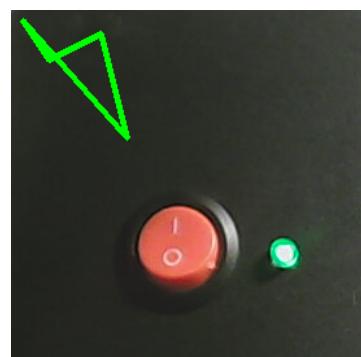
Rysunek 3.18: Rysowanie prostokąta.

d) *draw\_polyline* - rysuje wielokąt. Wymagany 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))
```

---



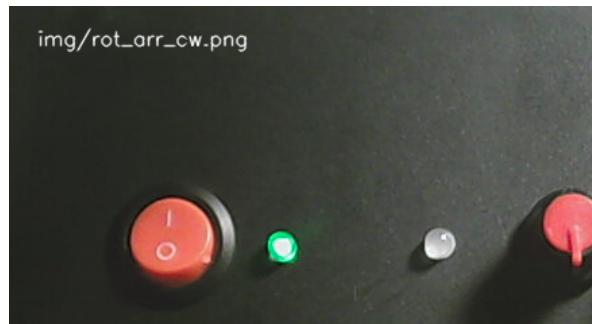
Rysunek 3.19: Rysowanie wielokąta.

e) *draw\_text* - rysuje na kadrze tekst o zadanej treści i współrzędnych. Wśród opcjonalnych parametrów są: król 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)
```

---



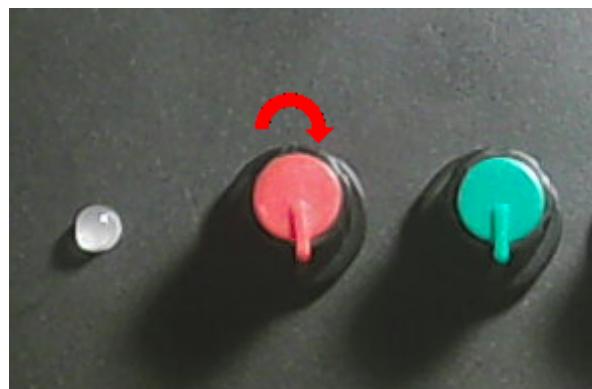
Rysunek 3.20: Rysowanie tekstu.

- 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.

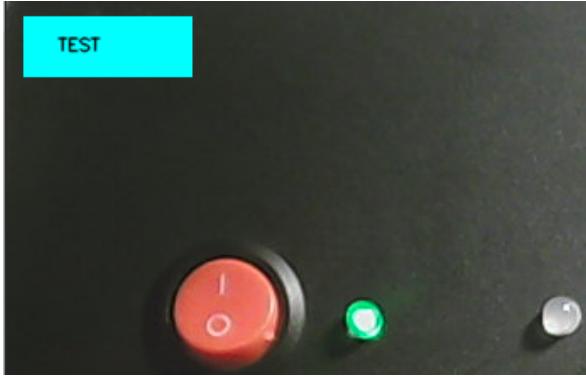
- g) *get\_image* - bezparametrowa funkcja zwracająca aktualny kadr, razem z ewentualnymi zmianami.

Możliwe jest też łączenie wywołań w łańcuchy. Taki kod znajduje się na listingu 3.4, a obraz wynikowy przedstawia rysunek 3.22.

---

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

---



Rysunek 3.22: Rysowanie prostokąta i tekstu.

**AugumentAPI.py** Zawarta w pliku klasa *AugumentAPI* jest jedną z dwóch obiektów, które powinny zostać zainicjowane w skrypcie głównym *main.py*. Pierwszym z nich jest obiekt strumienia video, a drugim właśnie *AugumentAPI*, które ten 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 subskrybowane będą tematy MQTT pobrane z pliku konfiguracyjnego. Klasa zawiera metodę *add\_layer*, w ramach której podaje się funkcję rysującą. Zostaje ona dodana 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 liczbę klatek na sekundę 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 wiadomości w terminalu w zależności od ich przeznaczenia: *info* (informacyjne), *warning* (ostrzeżenie), *error* (błąd), *success* (sukces); funkcja przeskalowania obrazu tak, aby jego rozmiar nie przekraczał maksymalnego, bez zmiany skali; odczytanie pliku JSON.

### 3.5. Komunikacja

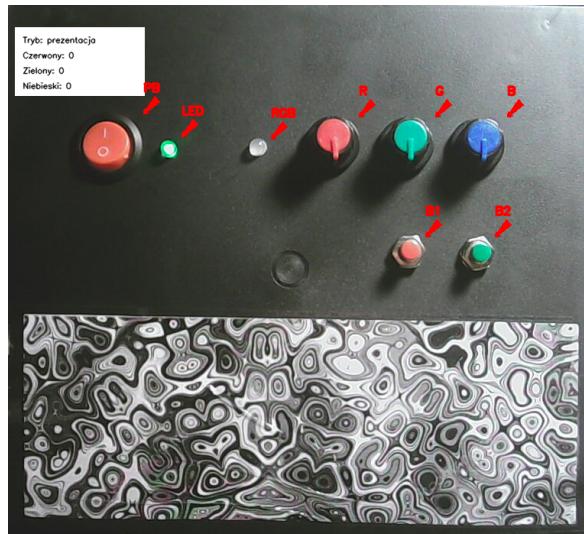
Komunikacja odbywa się za pomocą protokołu MQTT. Aby założenia projektowe mogły być spełnione, wyznaczono tematy do komunikacji:

- 1) *connected* - temat informujący czy urządzenie ma aktywne połączenie z brokerem. Może przyjąć wartość „ON” bądź „text”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 napięcia na potencjometrze na R1, z czerwoną gałką.
- 3) *arduino/green* - zawiera składową zielonego koloru. Przyjmuje wartości o 0 do 255 typu *String*. Jest to wartość napięcia na potencjometrze R2, z zieloną gałką.
- 4) *arduino/blue* - zawiera składową niebieskiego koloru. Przyjmuje wartości o 0 do 255 typu *String*. Jest to wartość napięcia na potencjometrze R3, z niebieską gałką.
- 5) *arduino/mode* - przyjmuje wartości od 0 do 2 typu *String*. Oznacza tryb, opisany w rozdziale 3.6. Wartość jest zwiększa o 1, bądź ustawiania na 0 po wcisnięciu przycisku B1.

### 3.6. Tryby działania

W celu przeprowadzenia prezentacji, autor podjął decyzję o stworzeniu trzech trybów, 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.

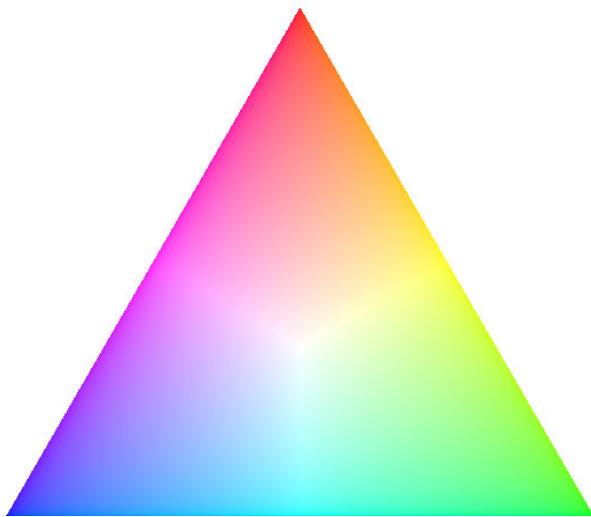


Rysunek 3.23: Tryb pokazowy.

Drugi i trzeci tryb są bardzo do siebie podobne. Ich celem jest pokierowanie użytkownika tak, aby dioda RGB świeciła na odpowiedni kolor. Dioda RGB to tak naprawdę trzy diody (czerwona, zielona i niebieska), z czego jasność każdej z nich zależy od doprowadzonego napięcia w zakresie 0-5V. Kolor wynikowy zależy natomiast od sumy składowych każdej z mniejszych diod. Zasadę działania zjawiska ilustruje rysunek 3.24. Widać na nim, że na przykład kolor czerwony i zielony daje kolor żółty, a suma wszystkich kolorów biały.

Składowe dla trybu drugiego (limonkowy) to (100, 200, 0), kolejno dla koloru czerwonego, zielonego i niebieskiego. Użytkownikowi przedstawione są przy potencjometrach strzałki skierowane w stronę, w którą powinien zakręcić. W przypadku, gdy danej składowej jest zbyt mało, prezentowana przy odpowiednim potencjometrze strzałka skierowana w prawą stronę, a jeśli zbyt mało, to skierowaną w lewą. Jeśli składowa ma odpowiednią wartość, nic nie jest rysowane. Dodatkowo rysowany jest prostokąt reprezentujący aktualny kolor na komputerze. Różne warianty prezentuje rysunek 3.25

Trzeci tryb nazwany został turkusowym, a pożądane składowe to (5, 85, 35).



Rysunek 3.24: Trójkąt kolorów RGB [30].

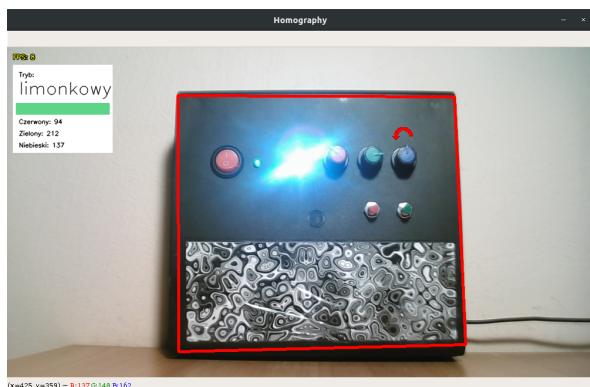
Wariant dla niepoprawnego i poprawnego stanu w tym trybie przedstawiono na rysunku 3.26.

Ze względu na niedokładność potencjometrów i przetwornika analogowo/cyfrowego Arduino, przyjęto dokładność +/- 15 wartości składowej.



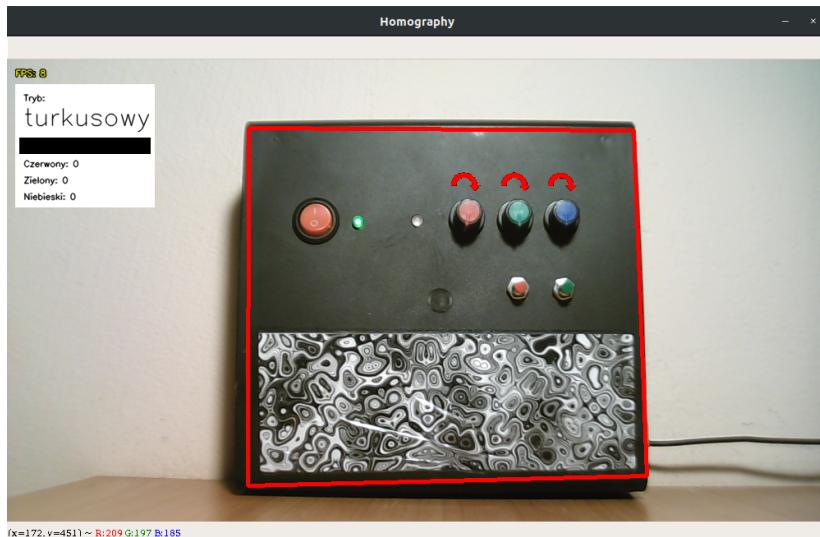
(a) Brak składowych.

(b) Poprawne składowe.

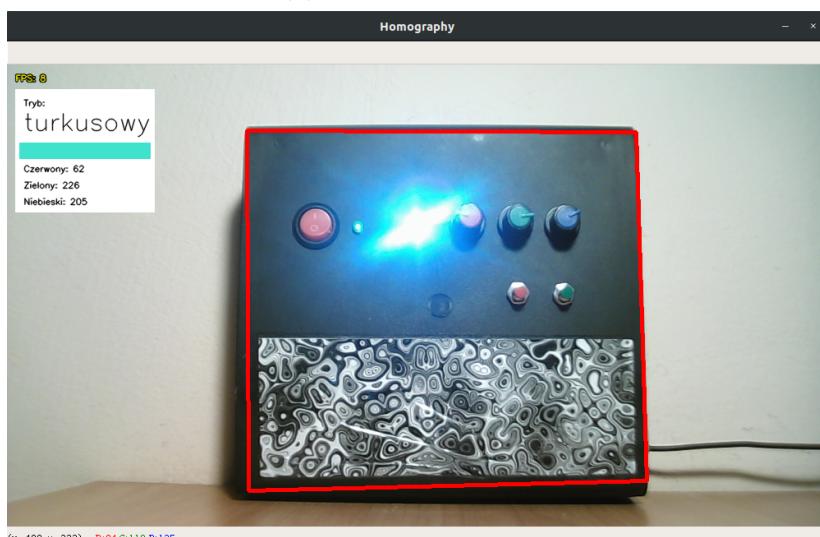


(c) Zbyt duża składowa koloru niebieskiego

Rysunek 3.25: Tryb limonkowy.



(a) Brak składowych.



(b) Poprawne składowe.

Rysunek 3.26: Tryb turkusowy.

## 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 4.27

Testy wykonano poprzez zwiększenie bądź zmniejszanie badanego parametru do momentu, w którym rozpoznawanie było stabilne. Punkty brzegowe zostały uwiecznione

i przedstawione w stosownych podrozdziałach w postaci zdjęcia sytuacji.



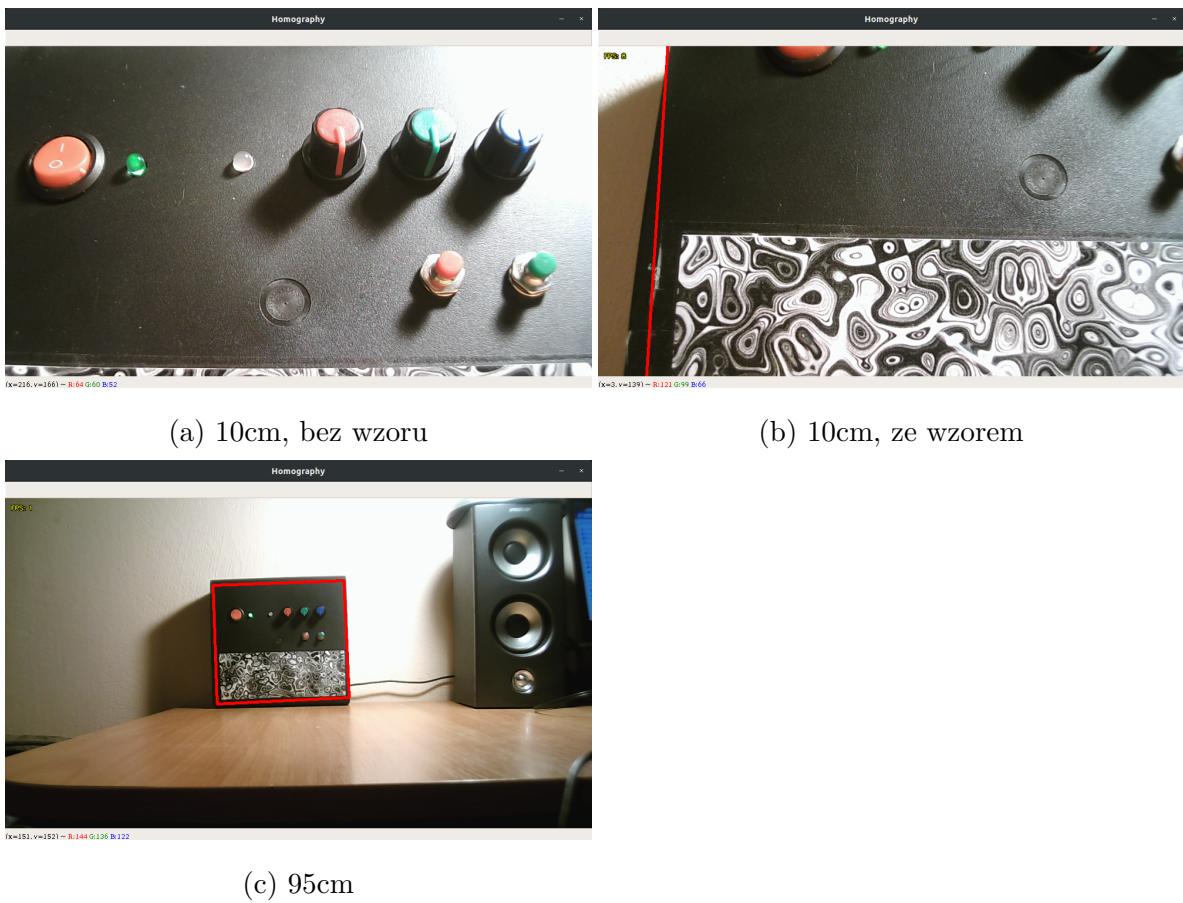
Rysunek 4.27: Stanowisko do testów.

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**

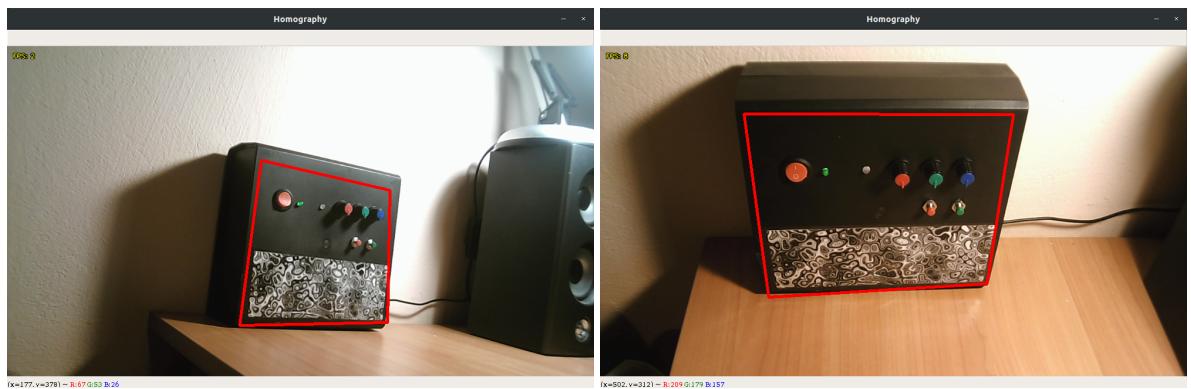
Testowano poprzez przesuwanie kamery wzduż osi głównej ustawionej prostopadle do płaszczyzny panelu urządzenia. Rozpoznawanie było stabilne w zakresie 10-95cm.



Rysunek 4.28: Testy odległości.

## 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. 4.29a, 4.29b).



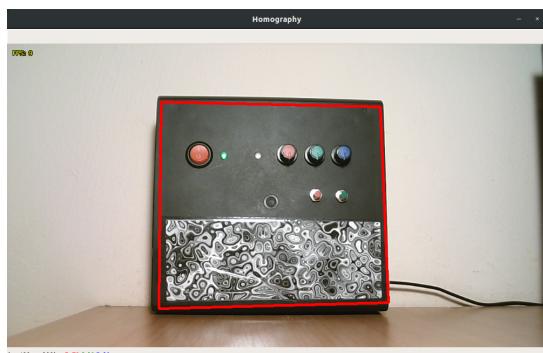
(a)  $45^\circ$ w płaszczyźnie poziomej

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

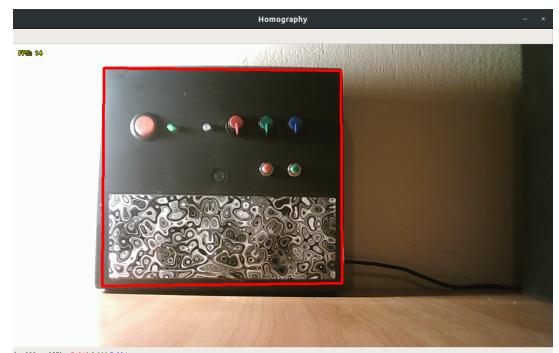
Rysunek 4.29: Test kąta patrzenia.

### 4.3. Test różnych warunków oświetleniowych

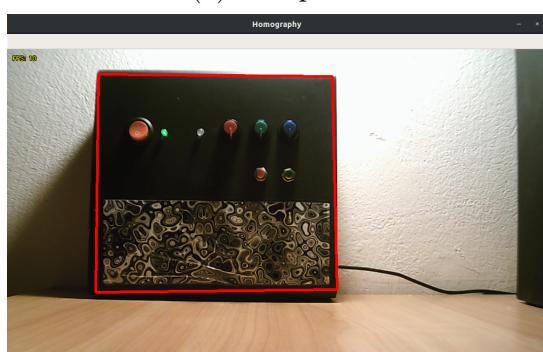
Poprzez warunki oświetleniowe rozumie się punktowe źródło światła skierowane z boku urządzenia. Celem było sprawdzenie, czy cienie rzucane przez wypukłe elementy panelu wpływają na detekcje.



(a) na wprost



(b) z boku

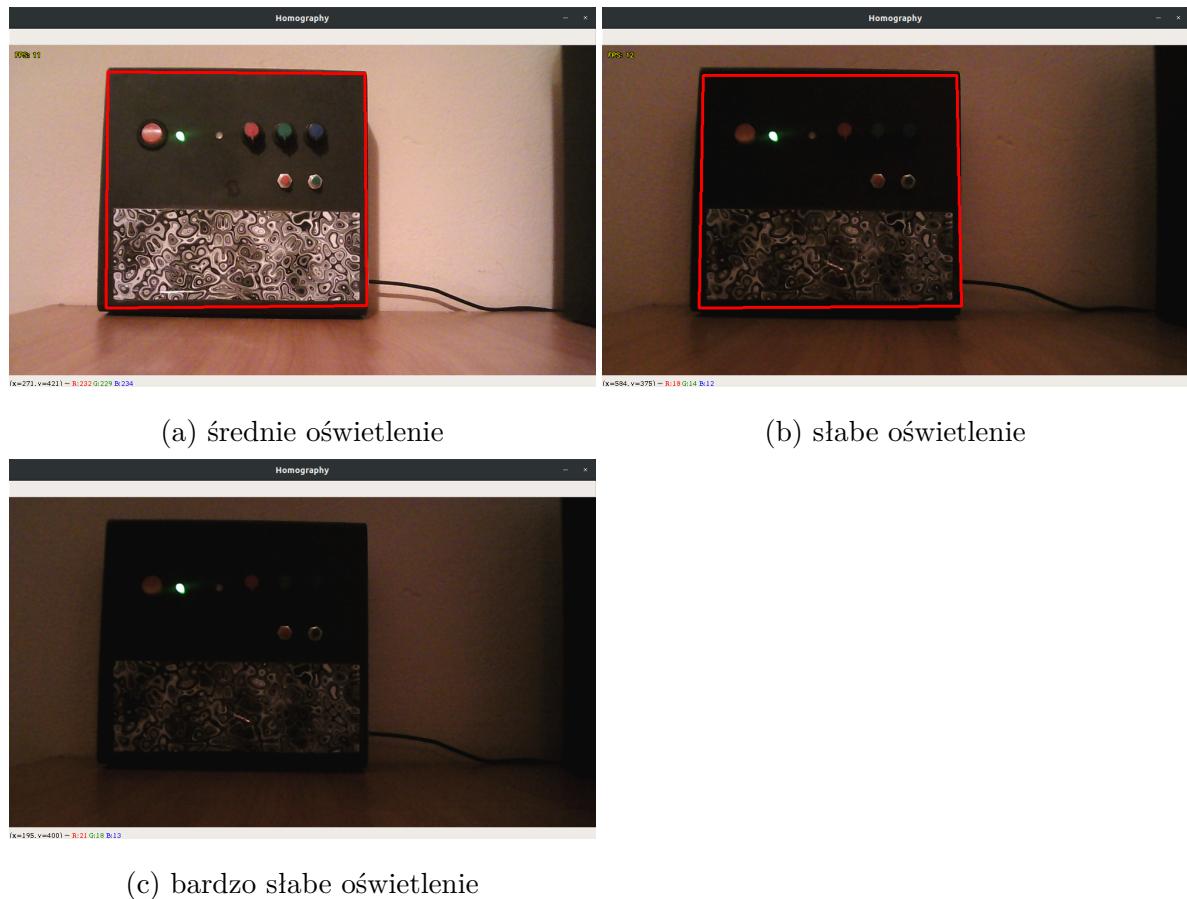


(c) od góry

Rysunek 4.30: Test różnych warunków oświetleniowych.

#### 4.4. Test różnego natężenia oświetlenia

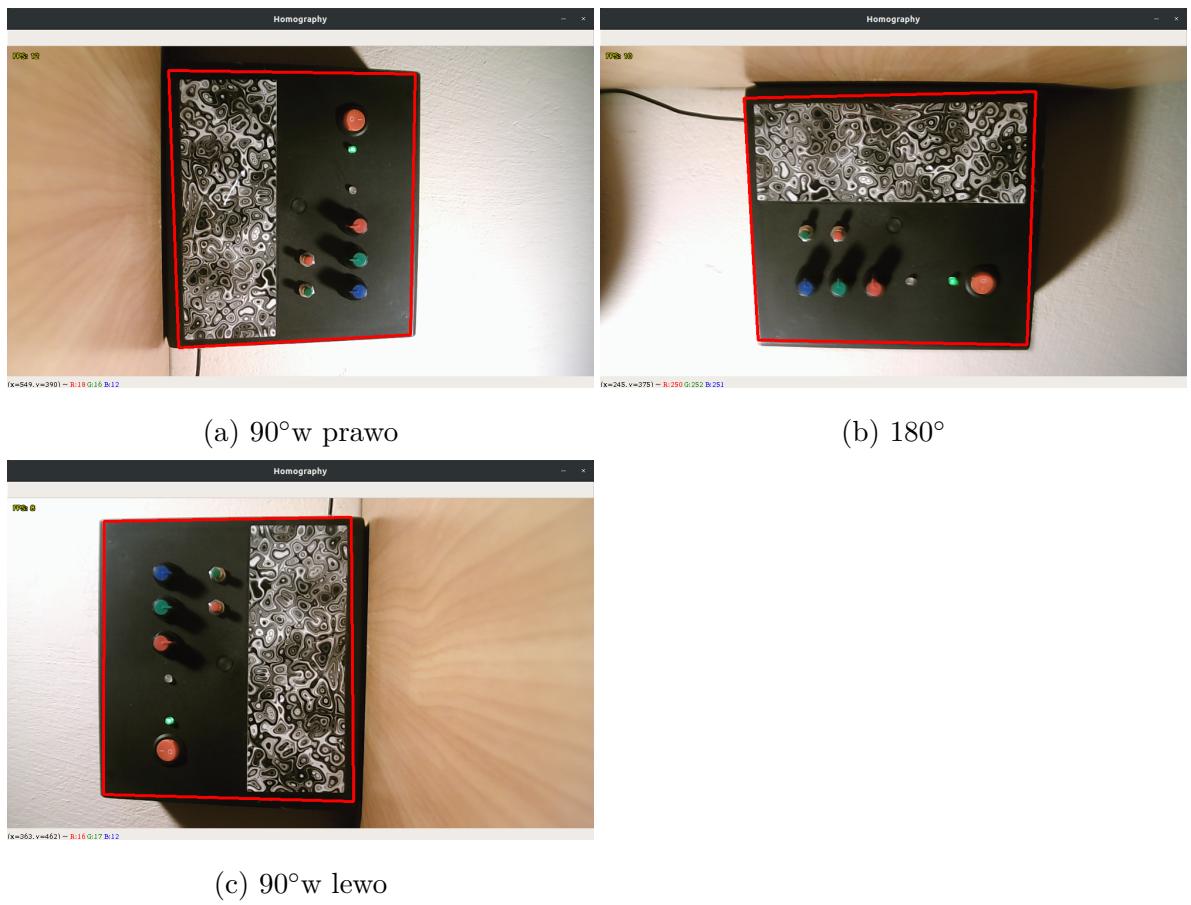
Celem testu było sprawdzenie jak system detekcji działa w warunkach słabego oświetlenia. Detekcje były poprawne średniego i słabego oświetlenia (rys. 4.31a, 4.31b). Dla oświetlenia bardzo słabego, czyli takiego dla którego elementy panelu nie były już wyraźnie, detekcja nie następowała (rys. 4.31c).



Rysunek 4.31: Test różnych natężeń oświetlenia.

#### 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  $90^\circ$ .

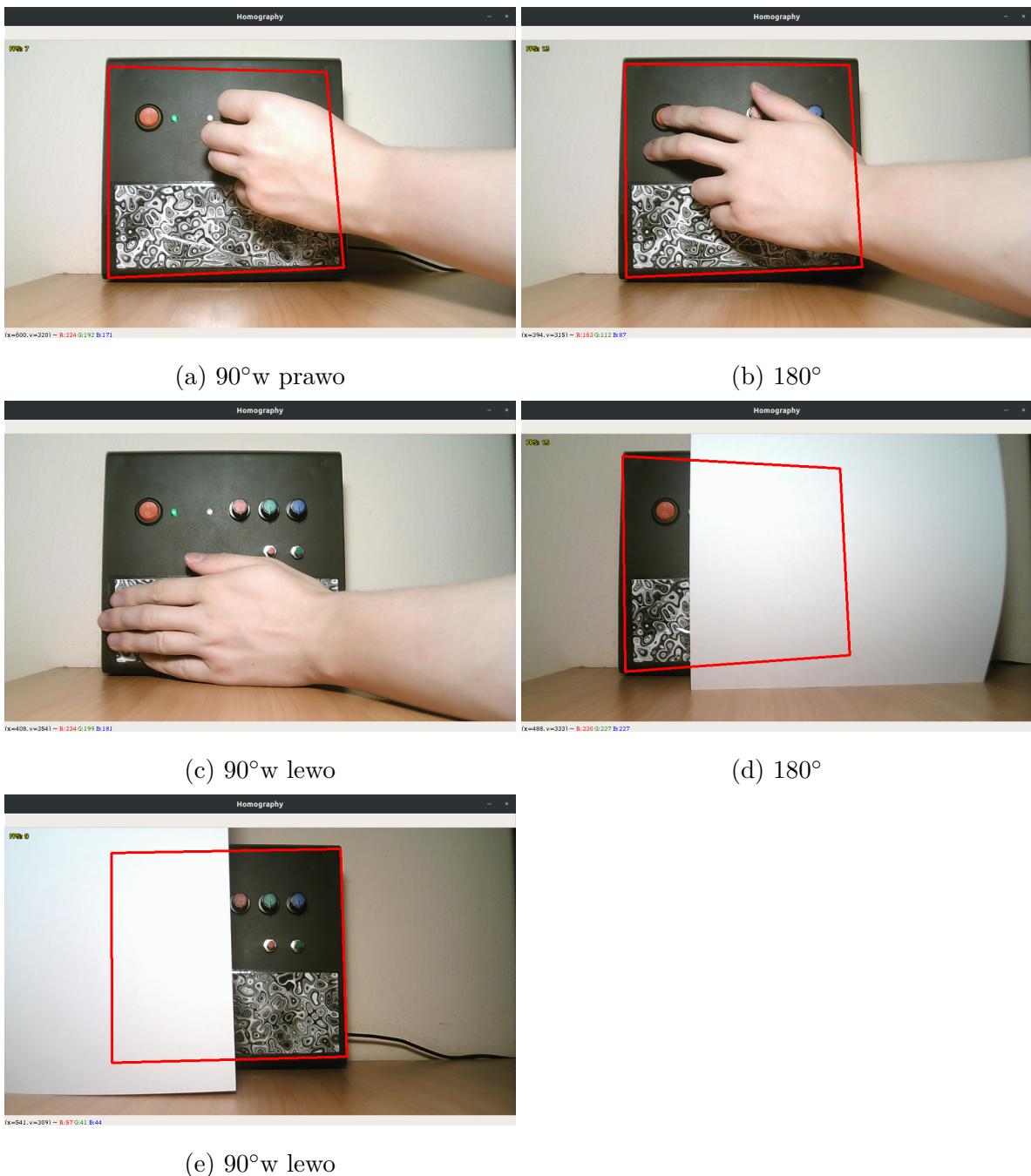


Rysunek 4.32: Test obrotu względem osi głównej kamery.

#### 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 część panelu została zasłonięta dłonią, detekcja była poprawna. W przypadku, gdy całość wzoru nie była widoczna, detekcja nie następowała. Dla dwóch ostatnich pomiarów, urządzenie było rozpoznawane jeśli co najmniej 25%-50% było widoczne.



Rysunek 4.33: Test częściowo widocznego urządzenia.

## 4.7. Test liczby punktów kluczowych

Motywacją do wykonania tego testu było określenie optymalnej liczby punktów kluczowych, które algorytm ORB powinien wyznaczyć. Testowano dwa parametry: maksymalną odległość dla której detekcje tracą stabilność, czyli otrzymywane są różne wykluczające się położenia w przeciągu kilku klatek strumienia; liczbę klatek na sekundę. Przetestowano 7 wariantów: 500 (domyślna wartość), 1000, 2000, 3000, 4000,

Tablica 4.1: Wyniki testu liczby punktów kluczowych.

liczba punktów	maksymalna odległość [cm]	FPS [kl./s]
500	45	10-16
1000	70	9-15
2000	85	9-11
3000	90	8-10
4000	95	8-10
5000	95	7-9
10000	95	5-7

5000, 10000. Wyniki testów przedstawiono w tablicy 4.1. Maksymalna odległość dla której detekcje były stabilne wynosi się w okolicy 95cm przy 4000 punktach, podczas dalszego zwiększania ich liczby malała jedynie liczba klatek na sekundę, nie zwiększa-jąc dokładności. Dlatego też liczba 4000 punktów kluczowych została wybrana podczas realizacji pracy.

## 4.8. Podsumowanie i wnioski z testów

- a) W teście odległości, maksymalną wartość zwiększyć można korzystając z lepszej kamery, bądź opierając się na obrazie w większej rozdzielczości. Dla bliższych dystansów detekcje zależały od faktu, czy wzór mieścił się w kadrze. Problem ten wynika z małej liczby wykrytych punktów nie należących do wzoru (rys. 4.28a).
- b) Wykorzystana kamera rozmazuje obraz w ruchu.
- c) 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.
- h) Maksymalna odległość dla której detekcje są stabilne przy użytej kamerze oraz wykorzystanej rozdzielczości wynosi 95cm.

## **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 wokół rzeczywistego urządzenia. Odporność na różne kąty pomiędzy kamerą a obiektem, do 45 stopni wydaje się być wystarczająca do poprawnej pracy. Stworzony projekt może zostać wykorzystywany między innymi do nauki obsługi urządzeń dzięki możliwości zlokalizowania poszczególnych ich elementów w przestrzeni, a także diagnostyki dzięki komunikacji odbywającej się w czasie rzeczywistym.

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 liczby klatek na sekundę. Wykorzystane metody do tworzenia nakładek są

prymitywne. Pożądane jest wykorzystanie bibliotek typu OpenGL do tworzenia grafiki, dzięki czemu możliwe będzie komponowanie ładniejszych i czytelniejszych interfejsów, a także wykorzystanie animacji i modelów 3D. W praktycznym zastosowaniu przydatna byłaby także obsługa więcej niż jednego urządzenia, a także możliwość określenia stanu obiektu nie tylko na podstawie przesyłanych parametrów, ale też stanu wizualnego, na przykład określenie obrotu potencjometru na podstawie jego wyglądu.

## 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\\_features\\_meaning/py\\_features\\_meaning.html#features-meaning](https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_features_meaning/py_features_meaning.html#features-meaning). Dostęp 22.06.2019.
- [16] E. Rublee *ORB: an efficient alternative to SIFT or SURF* [http://www.willowgarage.com/sites/default/files/orb\\_final.pdf](http://www.willowgarage.com/sites/default/files/orb_final.pdf)
- [17] E. Rosten, T. Drummond *Machine learning for high-speed corner detection* [https://www.edwardrosten.com/work/rosten\\_2006\\_machine.pdf](https://www.edwardrosten.com/work/rosten_2006_machine.pdf)
- [18] [https://docs.opencv.org/3.0-beta/doc/py\\_tutorials/py\\_feature2d/py\\_features\\_meaning/py\\_features\\_meaning.html#features-meaning](https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_features_meaning/py_features_meaning.html#features-meaning). Dostęp 22.06.2019.
- [19] M. Calonder, V. Lepetit, Ch. Strecha, P. Fua *BRIEF: Binary Robust Independent Elementary Features* [https://www.cs.ubc.ca/~lowe/525/papers/calonder\\_eccv10.pdf](https://www.cs.ubc.ca/~lowe/525/papers/calonder_eccv10.pdf). Dostęp 22.06.2019.
- [20] S. Tareen *A Comparative Analysis of SIFT, SURF, KAZE, AKAZE, ORB, and BRISK*, 2018. [https://www.researchgate.net/publication/323561586\\_A\\_comparative\\_analysis\\_of\\_SIFT\\_SURF\\_KAZE\\_AKAZE\\_ORB\\_and\\_BRISK](https://www.researchgate.net/publication/323561586_A_comparative_analysis_of_SIFT_SURF_KAZE_AKAZE_ORB_and_BRISK). Dostęp 22.06.2019.
- [21] [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.
- [22] [https://docs.opencv.org/3.0-beta/doc/py\\_tutorials/py\\_feature2d/py\\_sift\\_intro/py\\_sift\\_intro.html#sift-intro](https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_sift_intro/py_sift_intro.html#sift-intro)
- [23] A. Voulodimos *Deep Learning for Computer Vision: A Brief Review*, 2018. <https://www.hindawi.com/journals/cin/2018/7068349/abs/>. Dostęp 22.06.2019
- [24] D. Lowe *Distinctive Image Features from Scale-Invariant Keypoints*, 2004. <https://people.eecs.berkeley.edu/~malik/cs294/lowe-ijcv04.pdf>

- [25] <https://insights.stackoverflow.com/survey/2019/#technology>. Dostęp 16.06.2019
- [26] <https://www.ics.uci.edu/~mjamdar/vispercep/cameracalib.pdf>. Dostęp 19.06.2019
- [27] <https://opencv.org/about/>. Dostęp 19.06.2019
- [28] A. Kaehler, G. Bradski *Distinctive Image Features from Scale-Invariant Keypoints*. O'Reilly, 2017.
- [29] <https://devenv.pl/mqtt-protokol-transmisi-danych-dla-iot/>. Dostęp 19.06.2019
- [30] <http://www.allthesky.com/articles/colorpreserve.html>. Dostęp 24.06.2019.

POLITECHNIKA RZESZOWSKA im. I. Łukasiewicza  
Wydział Elektrotechniki i Informatyki

Rzeszów, 2019

## **STRESZCZENIE PRACY DYPLOMOWEJ MAGISTERSKIEJ**

### **ZASTOSOWANIE RZECZYWISTOŚCI ROZSZERZONEJ DO WSPOŁAGANIA 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 zwiększenie efektywności procesów dokonywanych w zakładach.

RZESZOW UNIVERSITY OF TECHNOLOGY  
Faculty of Electrical and Computer Engineering

Rzeszow, 2019

## **MSC THESIS ABSTRACT**

### **APPLICATION OF AUGMENTED REALITY TO SUPPORT THE OPERATION OF DEVICES**

Author: Rafał Ileczko, nr albumu: EF-144087

Supervisor: Mariusz Oszust PhD

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 allows to increase the efficiency of processes in factories.