

# **Politechnika Wrocławskiego**

## **Wydział Elektroniki, Fotoniki i Mikrosystemów**

---

**KIERUNEK:** Automatyka i Robotyka (AIR)

## **PRACA DYPLOMOWA INŻYNIERSKA**

**TYTUŁ PRACY:**  
Sterowanie manipulatorem poprzez śledzenie  
ruchów ręki

**AUTOR:**  
Paweł Warlewski

**PROMOTOR:**  
Dr inż. Bogdan Kreczmer,  
Katedra Cybernetyki i Robotyki



*Niniejszą pracę dedykuję mojej rodzinie oraz przyjaciołom, którzy wspierali mnie w jej realizacji. Szczególne podziękowania kieruję również w stronę mojego promotora, który nie tylko nadzorował jej przebieg, ale także dostarczał cenne wskazówki i inspiracje.*



# Spis treści

<b>1 Wstęp</b>	<b>3</b>
<b>2 Cel i założenia</b>	<b>5</b>
<b>3 Przegląd wybranych rozwiązań i czujników</b>	<b>7</b>
<b>4 Analiza problemu</b>	<b>9</b>
<b>5 Projekt stanowiska sterowania manipulatora poprzez śledzenie ręki</b>	<b>11</b>
5.1 Projekt manipulatora . . . . .	12
5.2 Projekt sterownika manipulatora . . . . .	14
5.3 Projekt systemu wizyjnego . . . . .	17
5.4 Projekt oprogramowania stanowiska . . . . .	18
5.5 Ramka komunikacyjna . . . . .	21
<b>6 Realizacja stanowiska sterowania manipulatora poprzez śledzenie ręki</b>	<b>23</b>
6.1 Realizacja manipulatora . . . . .	23
6.2 Oprogramowanie mikrokontrolera . . . . .	27
6.3 Realizacja systemu wizyjnego . . . . .	30
6.4 Oprogramowanie do wyznaczania konfiguracji ręki . . . . .	32
<b>7 Testy i eksperymenty</b>	<b>39</b>
7.1 Testy manipulatora . . . . .	39
7.2 Testy wykrywania konfiguracji ręki . . . . .	41
7.3 Testy całości . . . . .	44
<b>8 Podsumowanie i wnioski</b>	<b>49</b>
<b>Literatura</b>	<b>51</b>
<b>Spis rysunków</b>	<b>53</b>



# Rozdział 1

## Wstęp

W XXI wieku coraz częściej spotykać się można z sytuacjami, w których korzystna byłaby praca zdalna. Lekarz wykonujący operację nie musiałby ryzykować swojego zdrowia podczas pandemii, gdyby miał on dostęp do odpowiedniej zdalnej stacji operacyjnej. Nikt nie musiałby narażać swojego zdrowia podczas zbierania materiałów promieniotwórczych, jeżeli dostęp do urządzeń do tego przeznaczonych byłby ogólnodostępny. W przypadku braku specjalisty w ośrodku, mógłby on dokonać interwencji zdalnie. Przykładów zastosowania interfejsów umożliwiających zdalne sterowanie wyspecjalizowanymi robotami jest dużo, tak dużo, że wymienienie ich zajęłoby więcej stron niż posiada ta praca.

Z biegiem lat można zauważać odchodzenie od normy, w której osoba wykonująca zadanie musi być obecna na stanowisku pracy. Z roku na rok postęp technologiczny coraz bardziej umożliwia nam stosowanie wygodniejszych rozwiązań problemów cywilizacyjnych oraz technologicznych. Kiedyś do obsługi ramienia robota należało wykorzystać terminal komputerowy. Z czasem sterowanie te stało się wygodniejsze i takim samym manipulatorem sterować mogliśmy za pomocą myszki i klawiatury. Alternatywą do tego był także joystick, chociaż wybór sterowania zależał głównie od problemu, z którym borykało się stanowisko.

Pracując na stanowisku takim jak *da Vinci Surgical System* [Int], wykorzystywanie myszki i klawiatury byłoby karkołomnym doświadczeniem. Lekarz dokonujący operacji w czasie rzeczywistym musi wykonywać każdy ruch z ogromną precyzją, a sterowanie musi być tak intuicyjny, jak to tylko możliwe. Ciężko wyobrazić sobie bardziej intuicyjne sterowanie niż po prostu poruszanie ręką. Jest to jeden z podstawowych ruchów, który ludzie posiadają od urodzenia. Nie trzeba się go dodatkowo uczyć, nie sprawia nam problemów i jest wygodny.

Takie sterowanie ma dużo zalet, ale ma także swoje wady. Główną wadą, przez którą takie rozwiązanie nie jest tak ogólnodostępne, jest jego złożoność. Stworzenie stanowiska śledzącego położenie i ruchy ręki jest naturalnie bardziej skomplikowane, im większą dokładność chcemy uzyskać. Pomijając fakt wymagań sprzętowych stanowiska, musimy mieć do dyspozycji niezawodne oprogramowanie. Podczas sytuacji, w której ważą się losy pacjenta, nie możemy polegać na programie zawierającym błędy.

W tej pracy wykorzystany został jeden z możliwych algorytmów sterowania manipulatorem naśladowującym ruchy ludzkiej ręki. Stanowisko wyznaczające konfigurację ręki może znajdować się w innym miejscu niż ramię robota powtarzające

zadane ruchy. Umożliwia to pracę zdalną przy wykonywaniu zadania. W projekcie skonstruowano manipulator umożliwiający naśladowanie ruchów oraz zbadano algorytm sterujący nim. Rejestrowanie zadawanych ruchów odbyło się z wykorzystaniem niewyspecjalizowanych do tego typu zadań urządzeń. Profesjonalne urządzenia w podobnych projektach są kosztowne, a interesujący jest faktor różnicy w dokładności zastosowanych metod. Dokładność ta została zbadana poprzez testy stanowiska i sprawdzenie wykonywanych ruchów.

Motywacją tej pracy było przede wszystkim chęć przeprowadzenia badań nad takim sterowaniem. Sam zamysł projektu jest ciekawy i pozwala na zdobycie cennego doświadczenia. Zdalne sterowanie oraz robotyka cały czas idą na przód, dlatego okazja taka jak praca dyplomowa jest idealna do poszerzenia swojej wiedzy w tych dziedzinach.

# Rozdział 2

## Cel i założenia

Sterowanie ruchem ramienia robota poprzez śledzenie ruchu ręki operatora jest bardzo wygodnym sposobem pracy z robotem. Daje dużo swobody operatorowi i przyśpiesza jego pracę. Celem tego projektu jest realizacja tego typu sterowania. Zakłada się, że śledzone będą kluczowe stawy dłoni i staw łokcia. Ich pozycja i ruch będą odwzorowane przez manipulator. Śledzenie wspomnianych stawów powinno być realizowane możliwie bezinwazyjnie, tzn. bez potrzeby dodawania do ręki operatora dodatkowych elementów, np. znaczników. Zakłada się też stworzenie podstawowej wizualizacji konfiguracji śledzonej ręki oraz konfiguracji samego manipulatora. Gotowy projekt powinien z odpowiednią dokładnością poruszać manipulatorem, tak jak porusza się naśladowana ręka. Manipulator ten powinien z odpowiednią poprawnością i dokładnością powtarzać ruchy. Zakłada się, że sprawdzenie tych parametrów odbędzie się poprzez test ramienia, w którym będzie ono miało złapać i przenieść klocek.



## Rozdział 3

# Przegląd wybranych rozwiązań i czujników

W celu śledzenia ruchów ludzkiej ręki istnieje szeroka gama dostępnych czujników wizyjnych, przykładami takich kamer są kamery RGB, głębi, stereoskopowe, termowizyjne lub na podczerwień. Jednak najprostszym i najbardziej ekonomicznym rozwiązaniem jest wykorzystanie zwykłej kamery internetowej. Taka kamera jest łatwa do zamontowania, podłączenia do komputera, i zbierania danych. Co więcej, koszt takiej kamery nie przekracza zazwyczaj 20 zł za sztukę, co czyni ją atrakcyjnym wyborem dla projektów o ograniczonym budżecie. Pomimo jej niskiej ceny, kamera internetowa może skutecznie spełniać rolę źródła danych do śledzenia ruchów ręki. Nie jest wymagana wysoka rozdzielcość obrazu, 720 pikseli jest wystarczające. Ważne jest natomiast, aby kamera miała odpowiednie pole widzenia, aby umożliwić śledzenie zgięć i ruchów ręki oraz odpowiednie oświetlenie, które zapewni czytelność obrazu. W przypadku stosowania dwóch takich kamer, jedną z nich można umieścić w taki sposób, aby zbierała dane z osi OX i OY, a drugą z osi OY i OZ. W ten sposób uzyskujemy przestrzeń trójwymiarową, w której stawy ręki są widoczne na przynajmniej dwóch współrzędnych.

Po zebraniu danych z kamer, konieczne jest zidentyfikowanie ludzkiej ręki oraz niezbędnych stawów. W tym celu można wykorzystać techniki uczenia maszynowego do identyfikacji ludzkich kończyn lub specyficznych obiektów, które można przyczepić do ręki i łatwo wykryć na obrazie z kamer. Jednym z podejść jest wykorzystanie technologii Motion Capture, która jest stosunkowo prostym rozwiązaniem z perspektywy programisty, ale wymaga zastosowania rękawa z przyczepionymi markerami lub kulkami o unikalnych symbolach. Alternatywnym podejściem jest korzystanie z dostępnych bibliotek języka Python do wykrywania obiektów, zwierząt i kończyn, które często są udostępnione za darmo przez twórców. Do śledzenia ruchów ludzkiej ręki można wykorzystać takie biblioteki jak MediaPipe [[Goob](#)], OpenCV [[Ope](#)], YOLov3 [[Jos](#)], czy TensorFlow [[Gooa](#)].

W podobnych zadaniach często spotyka się wykorzystanie jednej kamery RGB lub kamery stereoskopowej. W takim przypadku dobrze określane są dwie przestrzenne współrzędne ręki, ale trzecia współrzędna jest mniej dokładna. Aby poprawić dokładność w wyznaczaniu trzeciej współrzędnej, czasem rezygnuje się z kamer RGB na rzecz modeli kamer wyspecjalizowanych w jednoczesnym wyznaczaniu wszystkich trzech współrzędnych. Jednak taki wybór zwykle zwiększa koszty

projektu. Przy użyciu kamer stereoskopowych można skorzystać z rozwiązań zaprezentowanych w pracy [Wei05], które wykorzystują metody zarówno znacznikowe, jak i bezznacznikowe. Opierają się one na analizie różnicy między kolorem skóry, a kolorem tła. Natomiast w przypadku podejścia z wykorzystaniem kamery RGB warto zapoznać się z artykułem [Mü]. Bada się w nim wykrywanie dloni oraz jej gestów przy użyciu biblioteki OpenCV. Inne podejście przy wykorzystaniu kamer RGB jest zaprezentowane w artykule [Kof]. Przedstawione jest w nim wykrywanie stawów ręki tylko za pomocą operacji przetwarzania obrazów. W tym artykule przedstawione także są rezultaty widoczne na ramieniu robota oraz badania zachowań i dokładności algorytmów. Zastosowanie czujnika Kinect w podobnym problemie zaprezentowane jest w artykule [MZAF]. Zaprezentowane są w nim wady i zalety korzystania z takiego czujnika.

Konstrukcja manipulatora powinna umożliwiać odwzorowywanie odpowiednich ruchów ręki. W takim projekcie powinien on zawierać co najmniej 5 stopni swobody. Podczas projektowania ramienia robota można posłużyć się projektem [6DO]. Dostępny w serwisie internetowym GitHub projekt ten omawia teorię, sterowanie i potencjalne problemy związane z tworzeniem ramienia robota od podstaw.

# Rozdział 4

## Analiza problemu

Chcąc zrealizować główne cele projektu, prace można podzielić na trzy etapy: akwizycję danych, przetwarzanie tych danych oraz wykonywanie zadanego ruchu. W fazie akwizycji najlepiej jest wykorzystać czujniki, o których mowa w rozdziale 3. Uzgłędniając zalety i wady poszczególnych czujników najlepiej wykorzystać w tym celu kamery RGB lub głębi. Są one ekonomicznym rozwiązaniem i pozwalają na rejestrację obrazów w dobrej jakości. Jednakże w celu większej dokładności powinniśmy wykorzystać dwie sztuki takich czujników.

Przetwarzanie obrazów z czujników głębi jest zwykle prostsze niż z kamer RGB, choć ta różnica może zostać zniwelowana przy użyciu odpowiednich bibliotek. Konieczne jest także zbudowanie odpowiedniego stelaża, na którym zamontowane zostaną kamery. Stelaż ten nie tylko umożliwiłby zamocowanie kamer, ale także określenie przestrzeni, w której można wykonywać ruchy. Musi on być wystarczająco wysoki i długi, aby zapewnić kamerom większe pole widzenia, co przekłada się na obszar, w którym można wykonywać ruchy. Jednak szerokość stelaża powinna być wystarczająca tylko do zapewnienia stabilności, uwzględniając wcześniejsze parametry.

Do przetwarzania danych otrzymanych z czujników, spośród wymienionych wcześniej bibliotek w celu wykrywania dloni najważniejsza jest biblioteka MediaPipe. Zapewnia ona precyzyjne śledzenie ruchów dloni, o ile dłoń jest dobrze widoczna na obrazie, a jej główne stawy są rozpoznawalne. Jednakże dłoń może znajdować się pod niewłaściwym kątem, co może sprawić trudności w jej identyfikacji. W takich sytuacjach można rozważyć śledzenie dloni na podstawie poprzedniej pozycji lub estymację jej pozycji na podstawie obrazu z drugiej kamery.

Biblioteka MediaPipe nie oferuje natomiast wykrywania pozycji łokcia na obrazach, które nie zawierają znacznej części górnej części ciała. W takich przypadkach konieczne jest użycie innych metod. Jednym z podejść jest przyczepienie znacznika do łokcia. Znalezienie pojedynczego znacznika jest prostsze niż wielu oraz można do tego można wykorzystać gotowe biblioteki do wykrywania obiektów. Jednak wadą tego rozwiązania jest ograniczenie obszaru, w którym można swobodnie poruszać ręką. Zawsze na obrazie musi być widoczny znacznik oraz końcówki palców. Innym sposobem na rozwiązanie problemu łokcia jest określenie, w którą stronę skierowane jest przedramię. Wykorzystując taką podejście, na obrazie możemy mieć tylko część przedramienia, ale mając informacje o położeniu nadgarstka, można poprzez przetwarzanie obrazów określić jego kierunek. Biblioteka OpenCV, oferująca meto-

dy takie jak wykrywanie krawędzi, może być pomocna w tego rodzaju operacjach.

Wizualizacja wyników poprzez aplikacje graficzną nie jest niezbędną, ponieważ będą one widoczne w ostatnim etapie procesu śledzenia. Jednak praca na takim stanowisku może być ułatwiona, jeśli użytkownik otrzymuje informacje zwrotne dotyczące wykrywania ręki, poprawności odbierania obrazów z kamer, i poprawności przesyłania przetworzonych danych. Te informacje to ważne detale, które program powinien dostarczać. Dodatkowym rozszerzeniem może być wyświetlanie obrazów z kamer po ich przetworzeniu, co pozwoli na wizualizację wyników i sprawdzenie ich poprawności.

Manipulator odpowiedni do odwzorowywania ruchów ręki powinien posiadać sześć stopni swobody. Ludzkie ramię ma tylko przeguby rotacyjne. Wykorzystując dwa stopnie swobody do ruchów barkiem, jeden do ruchu łokcia, dwa do nadgarstka, i jeden symulujący zamykanie dłoni możliwe jest odwzorowanie wszystkich głównych ruchów ręki. Przy takim rozwiążaniu należy skupić się na śledzeniu łokcia lub jego kierunku od nadgarstka, samego nadgarstka oraz początków i końców każdego z pięciu palców. Nie jest konieczne śledzenie pozycji barku, ponieważ jest to punkt początkowy, który pozostaje stały względem ręki. Do wykonywania ruchów rotacyjnych w takim manipulatorze można wykorzystać serwomechanizmy, które są ekonomicznym i prostym rozwiążaniem, szczególnie dla małego manipulatora. Sterowanie nimi można najłatwiej zrealizować poprzez podłączenie ich do mikrokontrolera za pomocą sygnałów PWM.

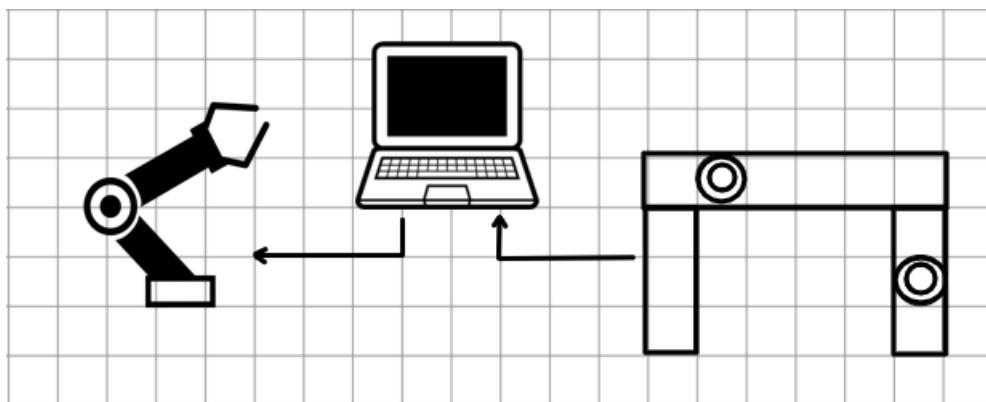
## Rozdział 5

# Projekt stanowiska sterowania manipulatora poprzez śledzenie ręki

Realizacja naśladowania ruchów ręki przez manipulator obejmuje następujące kroki:

1. Pobranie obrazów z kamer zamontowanych na stelażu;
2. Znalezienie na obrazach odpowiednich stawów;
3. Obliczenie kątów obrotu serwomechanizmów;
4. Przesłanie ramki z zadanymi kątami obrotu do mikrokontrolera;
5. Wykonanie zleconych ruchów przez serwomechanizmy.

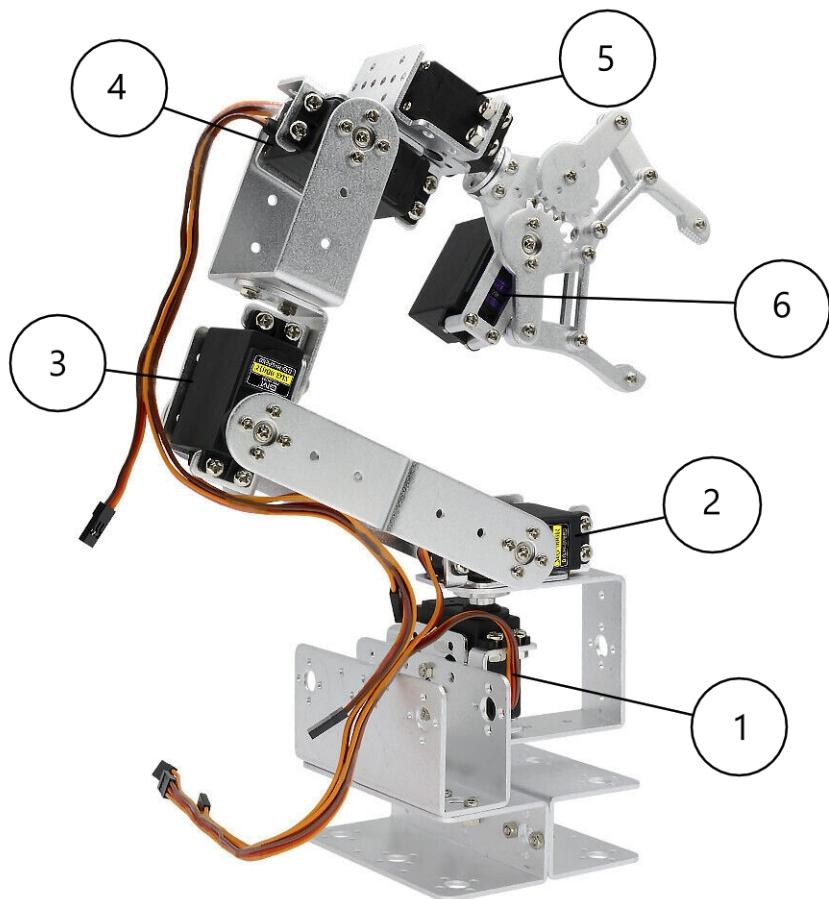
Na stelażu znajdują się dwie kamery, które odpowiedzialne są za przesyłanie obrazów do komputera. Komputer przetwarza te dane i oblicza odpowiednie wartości kątów. Następnie dane te są przesyłane dalej do mikrokontrolera za pomocą przewodowego interfejsu USB, tak jak to przedstawiono na rys. 5.1. Po tym procesie można zaobserwować wykonanie ruchów manipulatora. Podczas wykorzystywania całego stanowiska, warto podkreślić, że stelaż oraz manipulator to są osobne elementy. Ramie robota można ustawić na stelażu, ale osoba wykonująca ruchy, miałaby problemy z pełnym ich obserwowaniem.



Rysunek 5.1: Koncept projektu – komunikacja

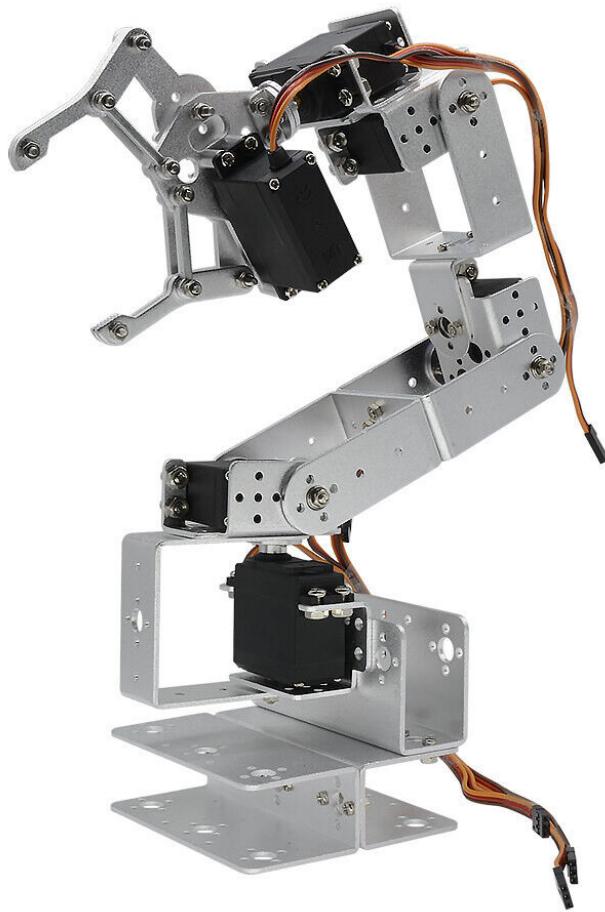
## 5.1 Projekt manipulatora

Manipulator posiada sześć stopni swobody, a za ich sterowanie odpowiadają serwomechanizmy MG-996R firmy Tower Pro. Te serwomechanizmy posiadają kąt obrotu  $180^\circ$ , mogą bezproblemowo przenosić ramie do 11 kg, pobierają takie samo napięcie jak mikrokontroler i są dostępne w przystępnej cenie. Ramie manipulatora wzorowane jest na gotowym modelu manipulatora, wykorzystując te same elementy mechaniczne i modele serwomechanizmów. Wszystkie części manipulatora można znaleźć w sklepie internetowym eBay.



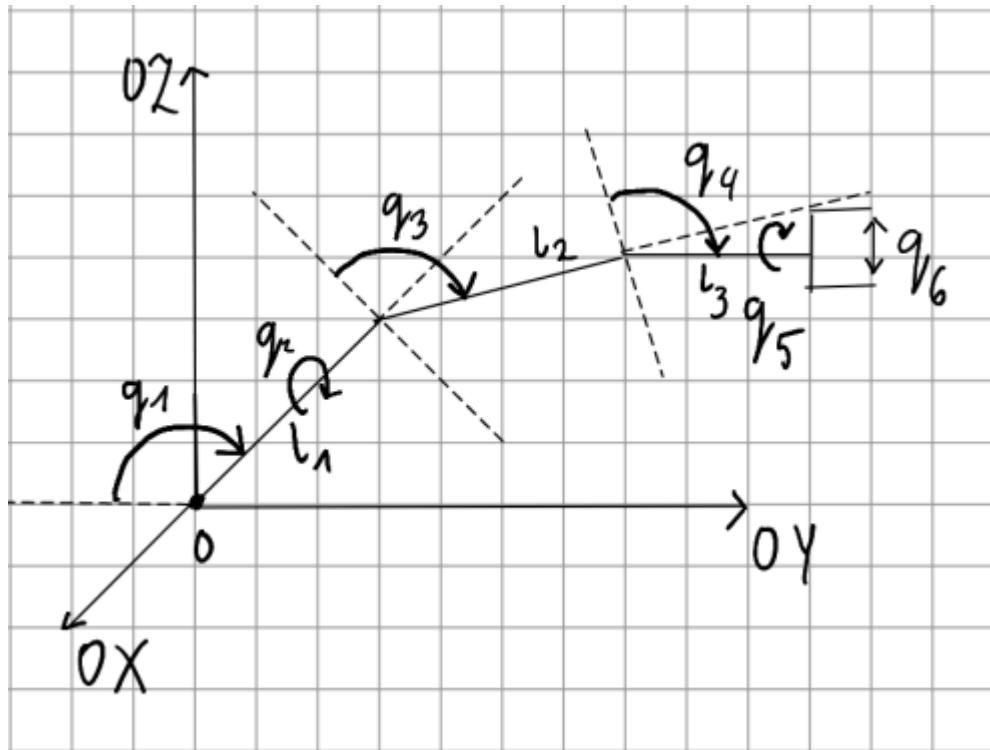
Rysunek 5.2: Ramie robota – zdjęcie konceptualne przód

Na rys. 5.2 punktami od 1 do 6 oznaczone są serwomechanizmy. W przypadku ludzkiej ręki stawy jakie nas interesują to: bark, łokieć, nadgarstek, i palce. Serwomechanizmy 1 i 2 naśladują ludzki bark, jednak w tym projekcie jest on punktem stałym w przestrzeni. Serwomechanizm 3 odpowiada za łokieć, 4 i 5 za nadgarstek. Palce naśladowane są w postaci zaciskania efektora przy zbliżeniu do siebie palca środkowego i kciuka, a odpowiada za to będzie serwomechanizm nr 6.



Rysunek 5.3: Ramie robota – zdjęcie konceptualne tył

Manipulator w pozycji wyprostowanej ma długość 45 cm i opiera się on na podstawie  $3 \times 39 \times 9,5$  cm. Zapewnia ona wystarczającą stabilność podczas wykonywania ruchów ograniczonych ustaloną prędkością. Manipulator składa się z 12 metalowych elementów połączonych śrubami M6 i nakrętkami. Serwomechanizmy są przyjmocowane do manipulatora za pomocą śrub o łbach philips. Istnieją również 4 łożyska, które zapewniają płynność ruchów. Z serwomechanizmów wychodzą 3 przewody: zasilanie, uziemienie i sygnał PWM.



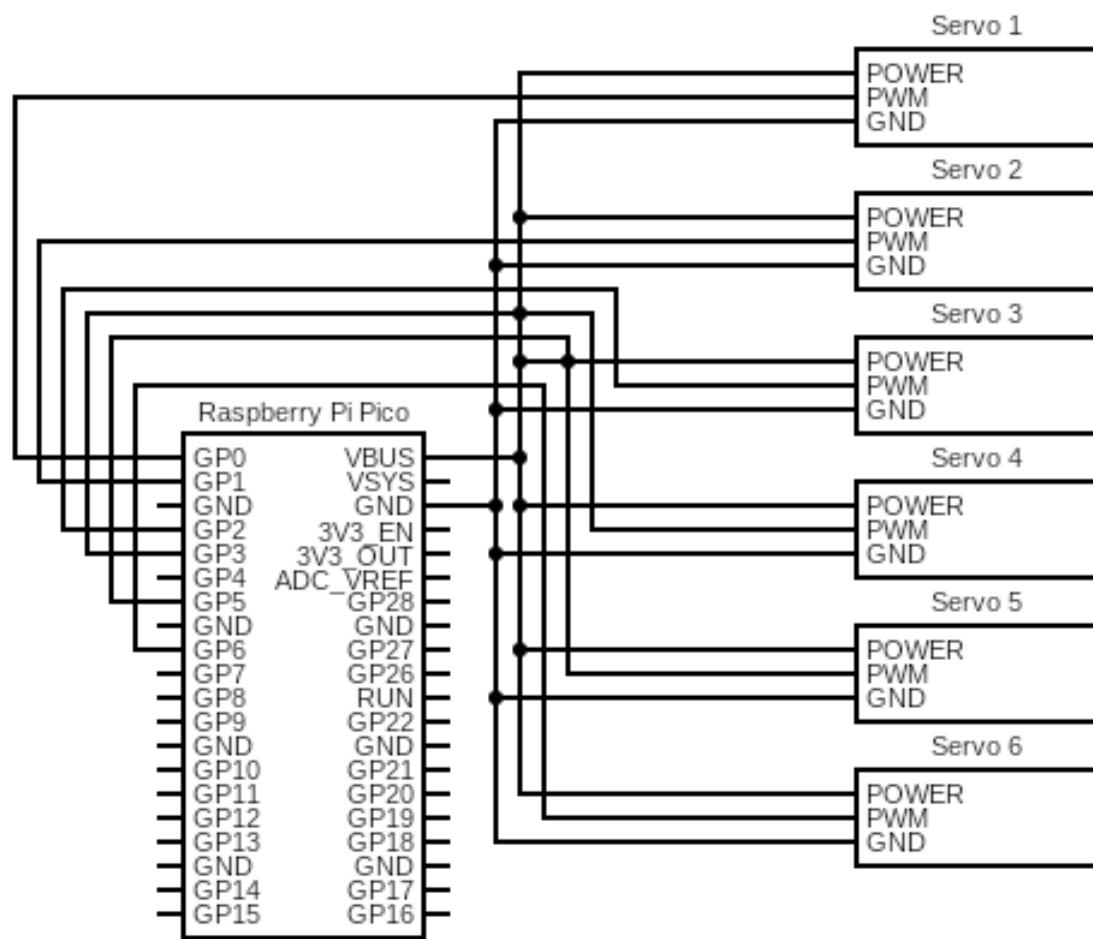
Rysunek 5.4: Rysunek techniczny kątów obrotów manipulatora

## 5.2 Projekt sterownika manipulatora

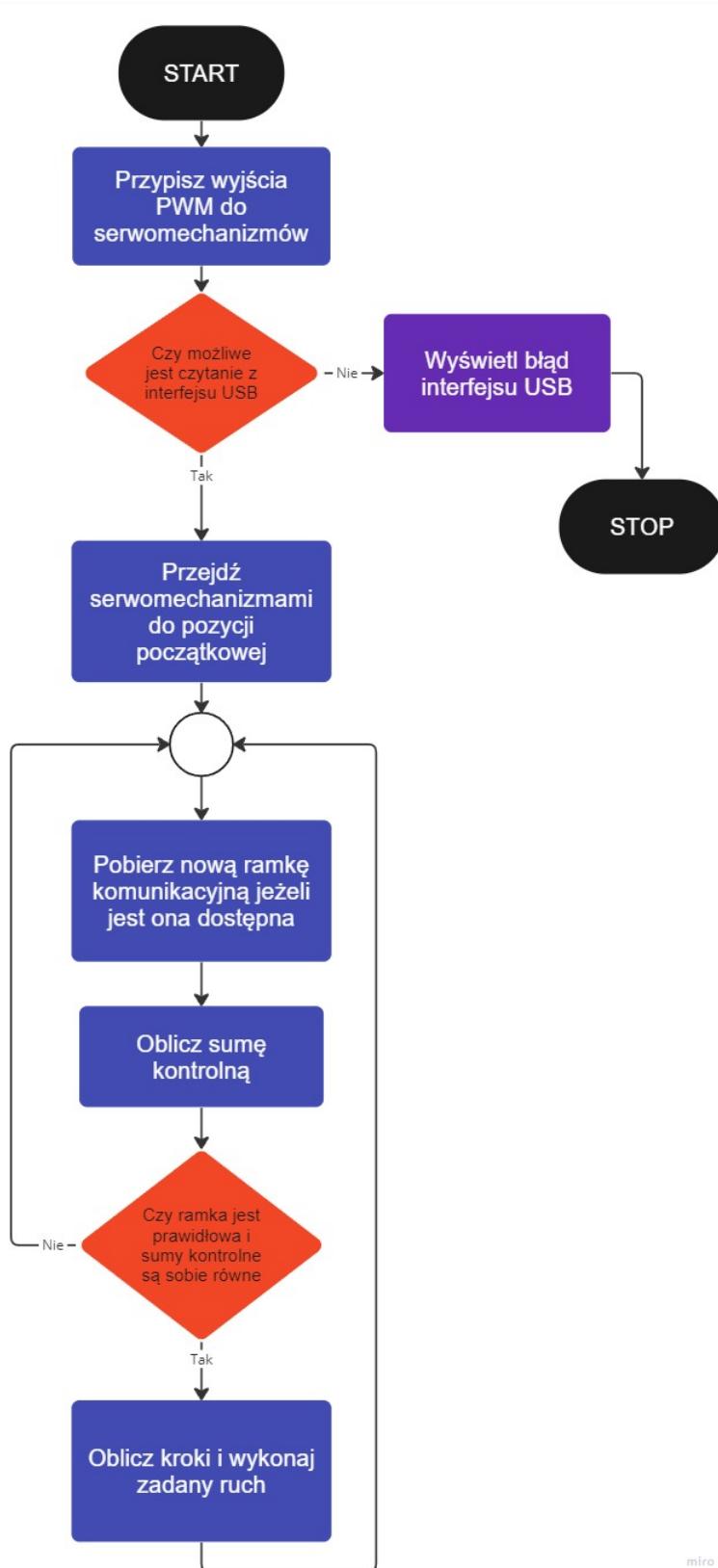
Do sterowania manipulatorem planowo wykorzystany został mikrokontroler Raspberry Pi Pico, który odpowiada za kontrolowanie sześciu serwomechanizmów. Napięcie zasilające serwomechanizmy pochodzi z komputera i mikrokontrolera poprzez przewodowy interfejs USB umożliwiający dwustronną komunikację. Piny GP0, GP1, GP2, GP3, GP5 i GP6 na rys. 5.5 przyjęto jako źródła sygnału PWM, a zasilanie serwomechanizmów dostarczane jest z VBUS o napięciu stałym 5 V.

Serwomechanizmy mają ograniczoną prędkość, aby uniknąć drgań i zagrożenia wynikającego z nieprzewidzianą kolizją. Zaczynają one zawsze w pozycji startowej, i co określony czas wykonują ruch do nowej zadanej pozycji. Nowa pozycja określana jest na podstawie otrzymywanych danych z portu USB.

Mikrokontroler otrzymuje ramki komunikacyjne z interfejsu USB (patrz rozdział 5.5). I jeżeli pobrana ramka jest prawidłowa to wykonywane jest zadany w niej ruch przez serwomechanizmy. Kolejność w jakiej realizowane są procesy mikrokontrolera widoczne są na rys. 5.6.



Rysunek 5.5: Schemat elektryczny

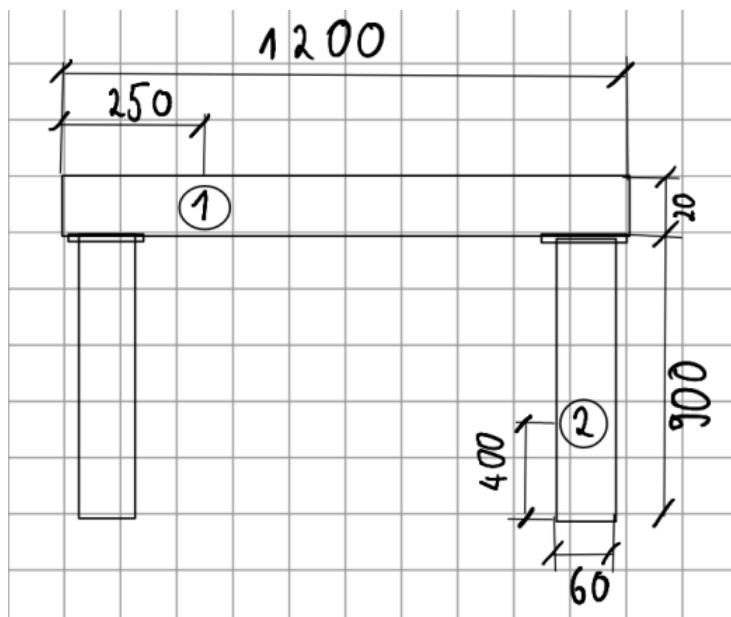


miro

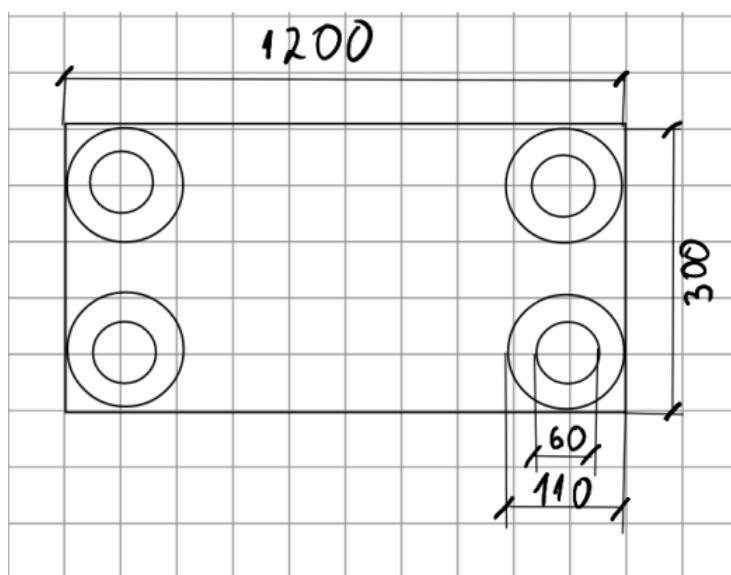
Rysunek 5.6: Schemat blokowy mikrokontrolera

### 5.3 Projekt systemu wizyjnego

Przyjęto, że kamery przymocowane są do stelaża przypominającego wyglądem stół barowy, który składa się z drewnianego blatu o wymiarach  $120 \times 30 \times 2$  cm. Do blatu planowo przymocowane są cztery nogi teleskopowe o długościach od 60 do 90 cm. Jedna taka noga ma szerokość 6 cm i posiada kołnierz o średnicy 11 cm, zapewniając stabilną całość konstrukcji. Nogi takie planowano zamocować do blatu w skrajnych jego pozycjach, rozwiążanie takie zapewni większą manewrowość ręki w przestrzeni.



Rysunek 5.7: Stelaż – widok z boku



Rysunek 5.8: Stelaż – widok z dołu

Na rys. 5.7 przedstawiony jest widok z boku stelaża, z zaznaczonymi punktami 1 i 2, które symbolizują kamery. Jedna z kamer została umieszczona na bocznej krawędzi blatu, a druga na jednej z nóg stelaża. Rys. 5.8 pokazuje rozłożenie nóg stelaża w skrajnych pozycjach blatu. Wszystkie nogi są identycznej długości, a jedna z nich zawiera kamerę. Kamera ta jest umieszczona w taki sposób, aby dwie płaszczyzny utworzone przez obrazy z obu kamer przecinały się.



Rysunek 5.9: Model kamery wykorzystanej w projekcie

W projekcie zaplanowano użycie kamery internetowej (rys. 5.9) o rozdzielczości  $1920 \times 1080$  pikseli i wymiarach  $8 \times 8 \times 3,5$  cm. Pozwala ona uzyskać obraz w dobrej jakości, za niewielką cenę, a także posiada ona uchwyt, ułatwiający jej mocowanie na stelażu. Przesłane dane z kamer poprzez porty USB zostają dalej przetworzone na komputerze.

## 5.4 Projekt oprogramowania stanowiska

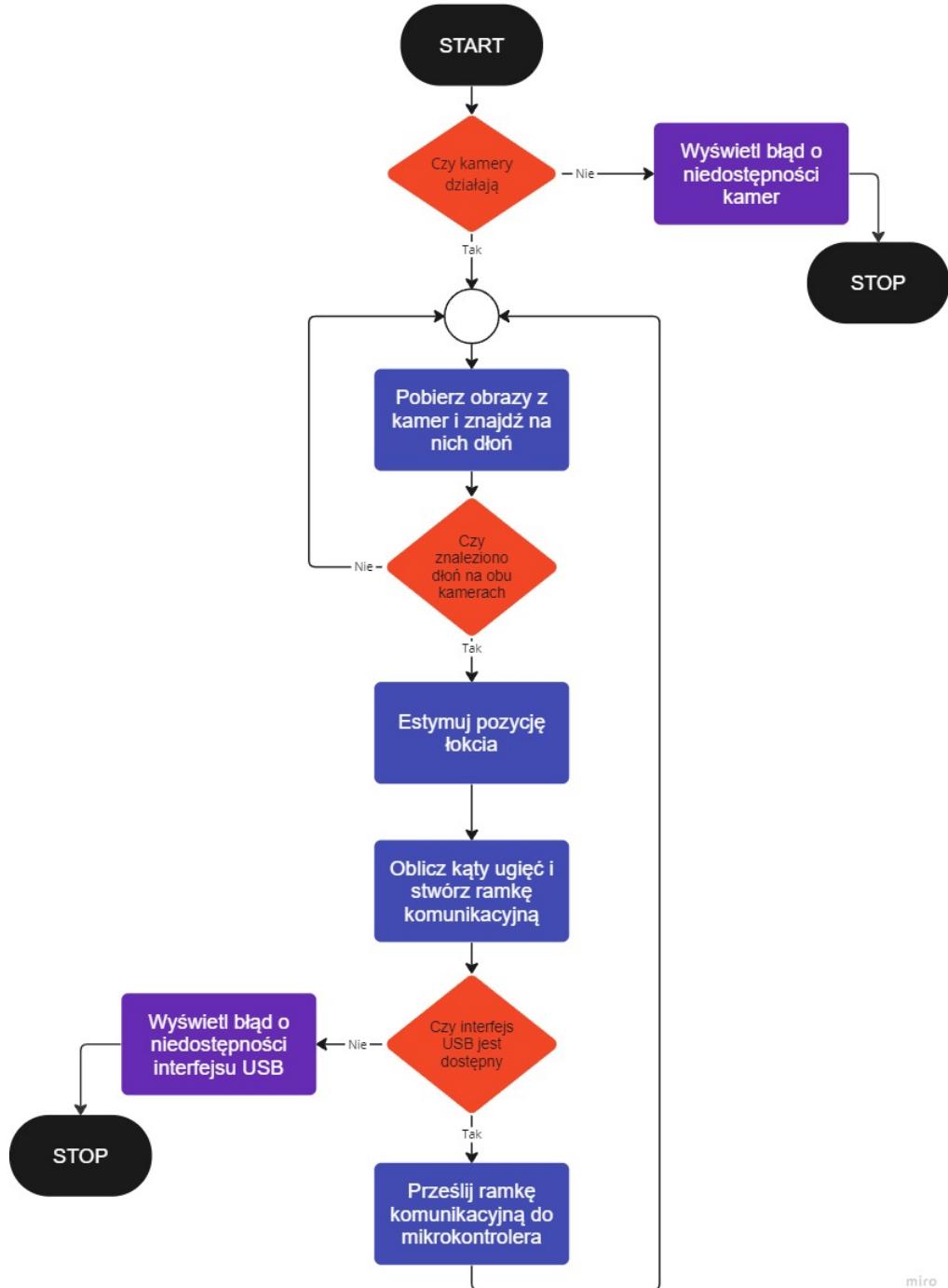
Aby wykrywać rękę, planuje się skorzystać z biblioteki MediaPipe, udostępnianej bezpłatnie przez Google. Biblioteka ta umożliwia wykrywanie dłoni i pozycji ciała, co pozwala nam śledzić główne stawy ludzkiego ciała, używając do tego technologii uczenia maszynowego. Zadaniem oprogramowania na komputerze jest przetwarzanie danych, aby mikrokontroler mógł skupić się na sterowaniu manipulatorem. Kolejność realizacji zadań przedstawiona jest na rys. 5.10. Główne funkcje oprogramowania obejmują:

- Odbieranie danych z dwóch kamer;
- Znalezienie szukanych stawów na obu obrazach;
- Wyświetlanie obrazów z kamer, wraz ze znalezionymi stawami;
- Obliczanie współrzędnych X, Y, i Z każdego szukanego stawu;
- Obliczenie o jaki kąt trzeba obrócić każdy serwomechanizm, aby uzyskać odpowiednie współrzędne;
- Zmiana kątów obrotów na sygnały PWM serwomechanizmów;
- Skonstruowanie ramki komunikacji i przesłanie jej przez USB do mikrokontrolera.

W programie przyjmujemy, że bark jest punktem startowym manipulatora, ale jest on traktowany jako punkt stały w przestrzeni. Człowiek może zmieniać jego położenie, lecz manipulator nie. W związku z tym, obliczając kąty obrotu serwomechanizmów, musimy uwzględnić ich położenie względem poprzedniego stawu. Na przykład, poruszając łokciem, patrzmy na zmianę jego położenia względem barku, a nie jego absolutne położenie w przestrzeni. To samo dotyczy nadgarstka, który jest traktowany jako punkt względny względem stawu łokciowego.

Jedną z funkcji programu jest możliwość podglądu obrazów z kamer oraz znalezionych na nich stawów. Jednak nie posiada on wizualizacji innych danych, takich jak kąty obrotu czy współrzędne w przestrzeni. Program musi zbierać, przetwarzać, i przesyłać dane w krótkich odstępach czasu, wynoszących maksymalnie sekundę. Dlatego zrezygnować należy z niepotrzebnych funkcji wizualnych, aby zachować płynność komunikacji i sterowania manipulatorem.

Rys. 5.4 obrazuje kąty  $q$  odpowiedzialne za serwomechanizmy. Kąty te w programie przybierają wartości 0 – 180 stopni. Wyjątkiem jest kąt ugięcia łokcia, który przyjmuje wartości 0 – 90 stopni. Następnie kąty te zmieniane są one na sygnały PWM ograniczone o wartości minimalne i maksymalne serwomechanizmów.



Rysunek 5.10: Schemat blokowy oprogramowania do wyznaczania konfiguracji ręki

## 5.5 Ramka komunikacyjna

Ramka komunikacyjna przesyłana jest jednokierunkowo z komputera za pomocą interfejsu USB. Składa się ona z trzech części. Wszystkie dane w ramce odzielone są znakiem spacji. Pierwsza część to nagłówek ramki, który zawiera znak startu przedstawiany zawsze jako znak S oraz sumę kontrolną CRC-16. W kolejnej części znajduje się sześć liczb, które odpowiadają za ustawienie kątów obrotu serwomechanizmów. Pierwsza liczba odpowiada za pierwszy serwomechanizm, druga za drugi, i tak dalej. Ostatnia część ramki to znak końca przedstawiany zawsze jako znak E. Jeśli obliczona suma kontrolna nie będzie się zgadzać z odebraną lub jeśli format ramki nie będzie poprawny, to zostanie ona odrzucona. Dane przesyłane są w formacie ASCII z prędkością 115200 bitów na sekundę. Przykładowa ramka ma następujący format:

S FB6F 5000 5000 6000 5000 6000 8000 E



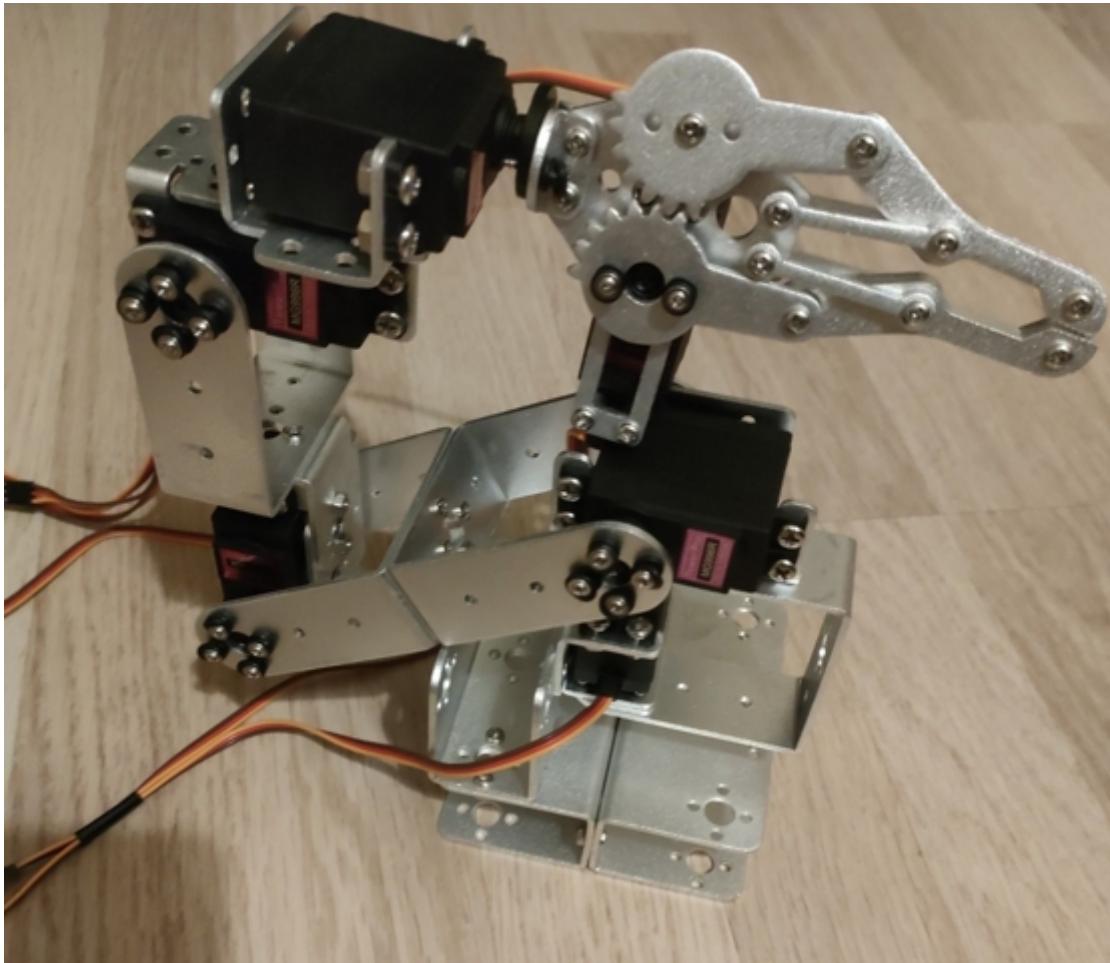
# Rozdział 6

## Realizacja stanowiska sterowania manipulatora poprzez śledzenie ręki

Rozdział ten przedstawia poszczególne etapy realizacji projektu. Podzielona ona została na osobne części, każda omawiająca inną część projektu. Podrozdziały zawierają w sobie zdjęcia oraz fragmenty kodu obrazujące finałowy wygląd wyników. Etapy realizacji przebiegały zgodnie z kolejnością podrozdziałów wymienionych poniżej. Pierwszy rozdział poświęcony jest realizacji manipulator. W kolejnym rozdziale znajduje się jego oprogramowanie. W trzecim punkcie omawiana jest realizacja systemu wizyjnego, a w ostatnim etapie oprogramowanie do wykrywania konfiguracji ręki.

### 6.1 Realizacja manipulatora

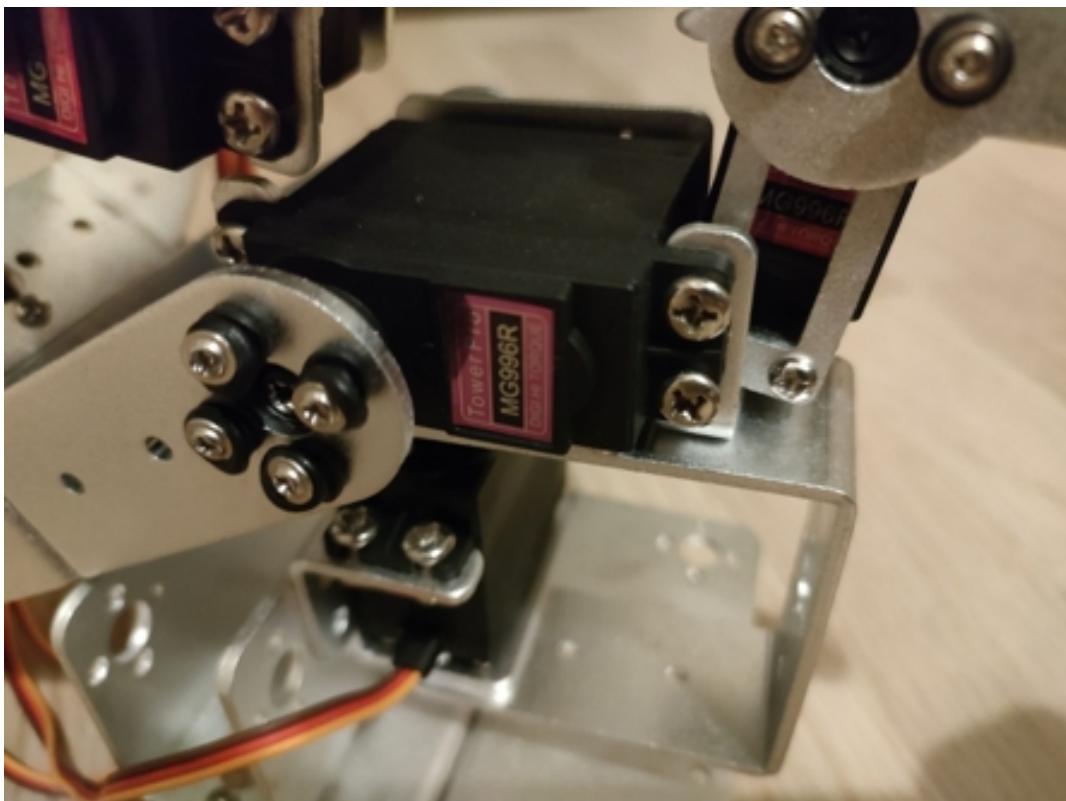
Manipulator przedstawiony na rys. 6.1 zrealizowany został według wcześniejszych założeń, podczas łączenia ze sobą części mechanicznych nie wystąpiły żadne problemy. Pojawiły się jednak one w serwomechanizmach, które miały zbyt duży kąt obrotu, wynoszący około  $270^\circ$ . Przez ten fakt, należy dla każdego serwomechanizmu obliczyć pozycję minimalną i maksymalną, w jakich może pracować. Bez ustalania ich, silniki mogłyby uderzać w inne fragmenty ramienia i uszkadzać swoje części wewnętrzne. Kolejny problem napotkany podczas montowania serwomechanizmów, dotyczył tego, że w jednym z nich przekładnia była uszkodzona. Nie nadawała się ona do pracy, i wytwarzaly hałas przekraczający dopuszczalny, zatem musiał być on wymieniony. Dwanaście mechanicznych elementów zostało połączone ze sobą śrubami i nakrętkami, takimi jak zakładano w projekcie. W efektorze jednak miały one inny wymiar niż przewidywano, a kąt obrotu serwomechanizmu jest tam znacznie mniejszy. Przy wykonywaniu gwałtownych ruchów robot traci stabilność, wynikającą ze zbyt małego obciążenia podstawy. W podstawie znajdują się miejsca na wkręty lub śruby widziane na rys. 6.3, umożliwiające zamontowanie ramienia do większej, ważcej więcej podstawy. Ewentualnie przez jej wyjątkowy kształt, można do niej włożyć dodatkowe wolne obciążenie.



Rysunek 6.1: Finalny manipulator

W tym projekcie użyte zostały orczyki z tworzywa sztucznego, które są wystarczająco trwałe do długotrwałej pracy ramienia. Orczyki montowane do metalowych części ramienia są za pomocą wkrętów. Wkręty te jednak są zbyt długie, dlatego należało użyć podkładek, tak aby te nie ocierały się o serwomechanizm, lecz nadal dysponowały odpowiednim naciskiem na ramie. Montowanie te widoczne jest na rys. 6.2.

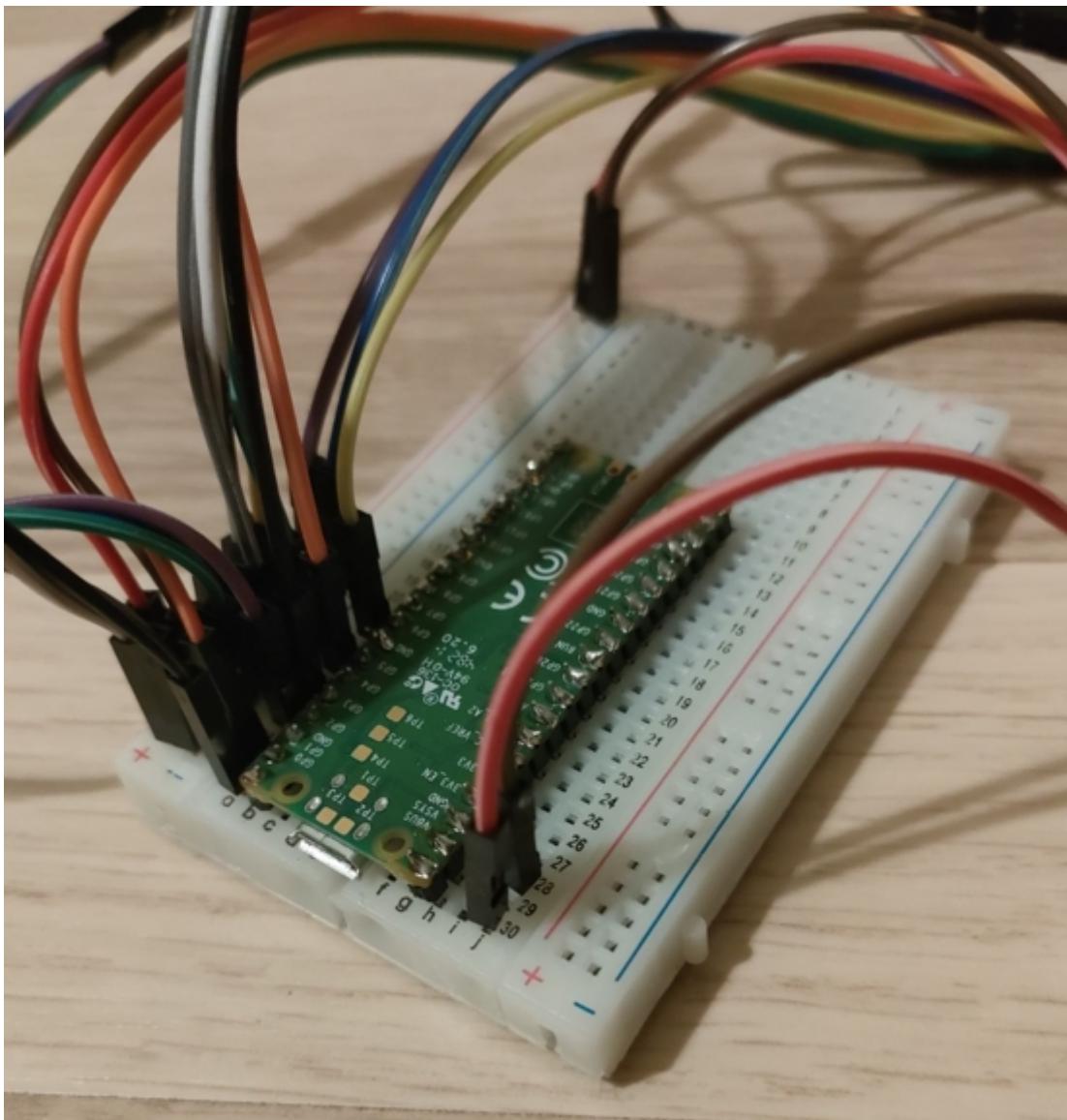
Połączenia elektryczne do mikrokontrolera zrealizowane zostały przy pomocy płytka stykowej, widocznej na rys. 6.4. Użycie takiej płytki znacznie zmniejsza koszt płytki, ułatwia połączenie, i redukuje wkładany w ten etap czas. Jest to rozwiązanie mało profesjonalne, dlatego możliwe jest, jako rozszerzenie projektu wymienienie jej na płytę PCB.



Rysunek 6.2: Zamontowany serwomechanizm – drugi serwomechanizm sterujący



Rysunek 6.3: Podstawa manipulatora



Rysunek 6.4: Połączenia mikrokontrolera

## 6.2 Oprogramowanie mikrokontrolera

Sterowanie ramieniem robota zostało przeprowadzone za pomocą mikrokontrolera Raspberry Pi Pico, który wykorzystuje język programowania Python. Serwomechanizmy przypisano do pinów 0, 1, 2, 3, 4, 5 i 6, a częstotliwość ich pracy wynosi 50 Hz. Inicjalizacja serwomechanizmów przedstawiona została na rys. 6.5, natomiast fragment kodu odpowiedzialny za sterowanie ich ruchem znajduje się na rys. 6.6.

```
# Define GPIO Pins for servos
SERVO_PINS = [0, 1, 2, 3, 5, 6]

# Initialize and create list of servos
servos = [machine.PWM(machine.Pin(pin)) for pin in SERVO_PINS]

# Set frequency for each servo
for servo in servos:
    servo.freq(50)
```

Rysunek 6.5: Inicjalizacja serwomechanizmów na mikrokontrolerze

Na początku działania programu wszystkie serwomechanizmy ustawione znajdują w pozycji startowej, w której znajdują się one w środkowym punkcie pomiędzy wartościami maksymalnymi i minimalnymi. Następnie uruchomione zostaje odbieranie danych ze standardowego wejścia mikrokontrolera, umożliwiając tym samym odbieranie ramek komunikacyjnych z portu USB. Interwał oczekiwania między danymi wynosi 1 sekundę, w czasie którego mikrokontroler pozostaje w stanie uśpienia. Jeśli odebrana ramka nie zawiera dokładnie 6 elementów, ich suma CRC-16 nie zgadza się z podaną w ramce, lub nie zawiera znaku początku lub końca, zostaje ona zignorowana.

Aby uniknąć gwałtownych ruchów ramieniem, które mogą prowadzić do drgań i potencjalnych zagrożeń, proces poruszania ramieniem od pozycji startowej do pozycji docelowej zostaje podzielony na ruchy krokowe. Każdy serwomechanizm wykonuje ruch składający się z 40 kroków, przy czym między kolejnymi krokami występuje opóźnienie wynoszące 10 ms. W wyniku tego zastosowania powstanie płynny ruch o ograniczonej prędkości, co minimalizuje ryzyko zagrożeń.

```

servo_positions_start = [5000, 7000, 5000, 6000, 4500, 8200] # Initial position
servo_positions_end = servo_positions_start                      # Final position
servo_positions_present = servo_positions_start                  # Present position
servo_positions_iterations = [0, 0, 0, 0, 0, 0]                 # Next step position
steps = 40

utime.sleep(2)

# Get servos to starting position
for i in range(len(servos)):
    print(f"Moving servo {i} to position {servo_positions_start[i]}")
    servos[i].duty_u16(servo_positions_start[i])
    utime.sleep(0.01)

utime.sleep(2)

(...)

# Calculate steps for each servo
for i in range(len(servos)):
    servo_positions_iterations[i] = int((servo_positions_end[i] -
                                          servo_positions_start[i]) / steps)

# Iterate over every step
for j in range(steps):
    # Iterate over every servo
    for i in range(len(servos)):
        servo_positions_present[i] += servo_positions_iterations[i]
        print(f"Moving servo {i} to position {servo_positions_present[i]}")
        servos[i].duty_u16(servo_positions_present[i])
    utime.sleep(0.01)

```

Rysunek 6.6: Wykonywanie ruchów serwomechanizmów na mikrokontrolerze

Proces weryfikacji poprawności otrzymywanych ramek jest przedstawiony na fragmencie kodu 6.7. Po pojawienniu się tekstu na wejściu standardowym, jest on dzielony na 9 części, przy użyciu spacji jako separatora. W pierwszej kolejności, sprawdzane są znaki początku i końca ramki. Jeżeli nie są one obecne w ramce, zostaje ona zignorowana. Następnie pobierane jest ciało ramki i obliczana jest suma kontrolna CRC-16. Ta suma jest porównywana z analogiczną sumą występującą w ramce. Jeżeli obie sumy są identyczne, a także liczb w ciele ramki jest sześć, zostaje wykonany zadany ruch.

```
# Wait for input on stdin, with waiting time of 100ms
poll_results = poll_obj.poll(100)
if poll_results:
    # Read the data from stdin
    frame = sys.stdin.readline().strip()

    # Clear stdin buffer
    poll_results = poll_obj.poll(1)
    while poll_results:
        sys.stdin.readline()
        poll_results = poll_obj.poll(1)

    # Split the data into individual values
    data = frame.split(' ')

try:
    # Check if frame has starting and ending sign
    if data[0] == 'S' and data[8] == 'E':

        # Get body of frame
        servo_positions_end = [int(item) for item in data[2:8]]

        # Calculate CRC-16
        checksum = calculate_crc16(servo_positions_end)

        # Check if checksums are equal
        if checksum == data[1]:
            # If there are 6 values move servos
            if len(servo_positions_end) == 6:

                (...)

except ValueError:
    print(ValueError)
```

Rysunek 6.7: Sprawdzanie ramki na mikrokontrolerze

### 6.3 Realizacja systemu wizyjnego

Stelaż, na którym zamontowane są kamery do śledzenia ruchów ręki, został przedstawiony na rys. 6.8. Jest to prosta konstrukcja, która zapewnia miejsce do wykonywania ruchów ręką i śledzenia ich. Ruchy te są jednak ograniczone wymiarami stelaża, z wysokością nie przekraczającą połowy wysokości stelaża, i szerokością nie przekraczającą połowy jego długości.

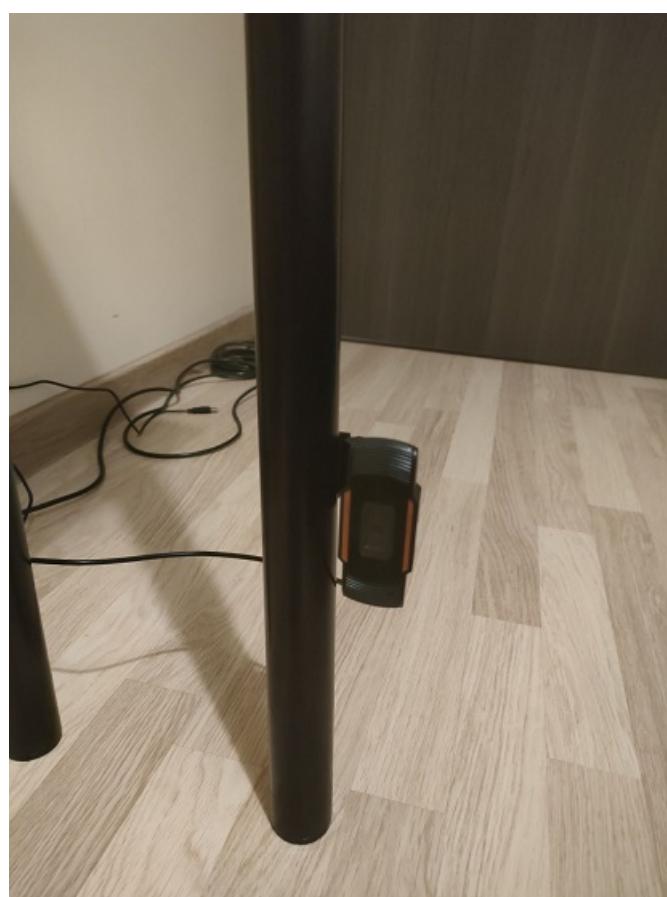


Rysunek 6.8: Złożony stelaż

Kamery, widoczne na rys. 6.9 i rys. 6.10, są zamocowane za pomocą pojedynczego wkręta, co umożliwia regulację ich pozycji w celach kalibracyjnych. Przewody USB kamery można poprowadzić wzduż nóg stelaża lub blatów, poprawiając estetyczny wygląd projektu.



Rysunek 6.9: Kamera górnego stelaża



Rysunek 6.10: Kamera dolnego stelaża

Nogi stelaża zostały przymocowane do blatu za pomocą pięciu wkrętów, co przedstawiono na rys. 6.11. Istnieje możliwość poluzowania nóg, obracając teleskopową część zgodnie z kierunkiem wskazówek zegara i dokręcenia ich, wykonując ruch przeciwny.



Rysunek 6.11: Mocowanie nogi stelaża

## 6.4 Oprogramowanie do wyznaczania konfiguracji ręki

Oprogramowanie opiera się na wykorzystaniu gotowego modelu wykrywania dloni, który dostarczony jest wraz z biblioteką MediaPipe. Natomiast przetwarzanie obrazów jest wykonywane przy użyciu biblioteki OpenCV. Kod odpowiedzialny za odnajdywanie dloni przedstawiono na rys. 6.12. Znalezione współrzędne przegubów dloni są uśredniane w celu zapewnienia większej stabilności. Rys. 6.13 przedstawia stawy wykrywane przez bibliotekę MediaPipe, z których można wybrać te, które nas interesują. W kolejnym kroku należy użyć wykrywania krawędzi za pomocą metody Canny Edge Detection. Otrzymany w ten sposób obraz zależy od tła i kontrastu między kolorem skóry, a tłem. W celu uzyskania bardziej zadowalających wyników zastosowano powierzchnie wyłożoną jednolicie czarnym materiałem. Ta- kie tło zapewnia brak artefaktów, minimalne cienie, i kontrast z kolorem skóry. Po wykryciu krawędzi możemy skoncentrować się na określaniu kierunku łokcia w stosunku do nadgarstka. W tym celu używamy transformacji Hough Line. Transformacja ta znajduje proste linie na obrazie. Można dostosowywać parametry transformacji, prowadzi to jednak do rozdrobnienia aproksymacji. Aproksymacja taka ma większą ilość linii, a co wiąże się z tym gorszą jakość uśrednienia. Ostatecznie można uśrednić współrzędne końców wszystkich znalezionych linii i ich pozycje w czasie. W ten sposób znaleziony kierunek łokcia staje się mniej zależny od zakłóceń i znacznych niedokładności. Kod implementujący tą procedurę znajduje się na rys. 6.14.

```

# Initialize MediaPipe hands model
mp_hands = mp.solutions.hands.Hands(min_detection_confidence=0.5, min_tracking_confidence=0.5, max_num_hands=1)
previous_hand = []

(...)

# Detect the hand
results_hands = mp_hands.process(image)
if results_hands.multi_hand_landmarks:
    hand_landmarks = results_hands.multi_hand_landmarks[0]
    previous_hand.append(hand_landmarks)
    if len(previous_hand) > 2:
        previous_hand.pop(0)

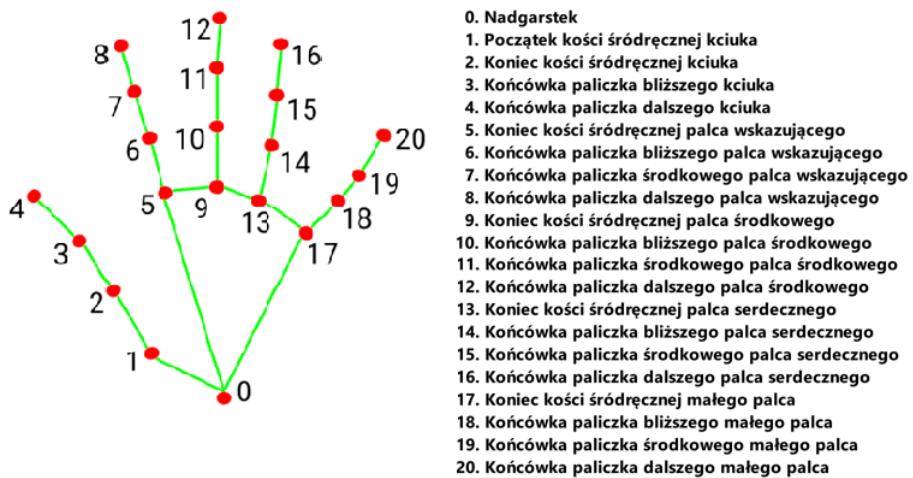
    # Iterate through the joints
    for joint_index in range(len(previous_hand[0].landmark)):
        x_sum = sum(element.landmark[joint_index].x for element in previous_hand)
        y_sum = sum(element.landmark[joint_index].y for element in previous_hand)
        z_sum = sum(element.landmark[joint_index].z for element in previous_hand)

        hand_landmarks.landmark[joint_index].x = x_sum / len(previous_hand)
        hand_landmarks.landmark[joint_index].y = y_sum / len(previous_hand)
        hand_landmarks.landmark[joint_index].z = z_sum / len(previous_hand)

    previous_hand[-1] = hand_landmarks

```

Rysunek 6.12: Inicjalizacja i wykorzystanie modelu Hand biblioteki MediaPipe



Rysunek 6.13: MediaPipe – wykrywane stawy

```

previous_elbow = []

(...)

# Canny edge detection
edges = cv2.Canny(image, 100, 100)

# Use Hough Line Transform to detect lines
lines = cv2.HoughLines(edges, 1, np.pi / 180, threshold=50)
line_endpoints = [] # Store endpoints of valid lines

if lines is not None:
    for line in lines:
        rho, theta = line[0]
        a = np.cos(theta)
        b = np.sin(theta)
        x0 = a * rho
        y0 = b * rho

        # Calculate two points to draw the line
        x1 = int(x0 + 1000 * (-b))
        y1 = int(y0 + 1000 * (a))
        x2 = int(x0 - 1000 * (-b))
        y2 = int(y0 - 1000 * (a))

        # Set y1 to be at the top
        if y1 > y2:
            x1, y1, x2, y2 = x2, y2, x1, y1

        # If line is above or below wrist ignore it
        if y1 < wrist_y and y2 < wrist_y or y1 > wrist_y and y2 > wrist_y:
            continue

        try:
            # Calculate x of the line at the fixed y of wrist
            closest_x = int(((x2 - x1) / (y1 - y2)) * (y1 - wrist_y) + x1)

            # If coords are within the radius of wrist, draw the line
            if wrist_x - radius < closest_x < wrist_x + radius:
                cv2.line(image_with_landmarks, (int(x1), int(y1)), (int(x2), int(y2)), (0, 127, 0), 1)
                line_endpoints.append((x2, y2))
        except:
            pass

    (...)

try:
    # Calculate the mean point of the detected lines
    mean_x = sum(coord[0] for coord in line_endpoints) / len(line_endpoints)
    mean_y = sum(coord[1] for coord in line_endpoints) / len(line_endpoints)

    # Add the mean point to the array
    previous_elbow.append((int(mean_x), int(mean_y)))
    if len(previous_elbow) > 10:
        previous_elbow.pop(0)
except:
    pass

if len(previous_elbow) > 0:
    # Calculate the mean point of all points in array
    elbow_x = sum(coord[0] for coord in previous_elbow) / len(previous_elbow)
    elbow_y = sum(coord[1] for coord in previous_elbow) / len(previous_elbow)

```

Rysunek 6.14: Wykrywanie kierunku łokcia – kamera górną

Kod, który został przedstawiony powyżej, służy do wykrywania kierunku, w którym znajduje się łokieć, jednak nie określa jego dokładnej pozycji. Znalezienie dokładnej pozycji łokcia ułatwia obliczanie kątów w kolejnych etapach. Warto zaznaczyć, że na przetwarzanych obrazach nie zawsze się on znajduje. Do znalezienia jego pozycji wykorzystać można proporcje ludzkiej ręki, a w szczególności długości przedramienia i dloni. Długość ludzkiego przedramienia jest proporcjonalna do długości dloni w stosunku około 1,618 : 1,0 [Mei]. Pomimo, że ta proporcja nie jest dokładna oraz zawiera pewien błąd w granicach  $\pm 0,1$ , to stosujemy ją w obliczeniach i przybliżamy ją do 1,6. Można również wykorzystać tutaj długość kości śródręcznej, która jest proporcjonalna w stosunku około 2 : 1 w porównaniu do dloni. Dłoń może być zgięta, dlatego stosowanie w obliczeniach długości niezależnej od większej ilości punktów jest łatwiejsze. Wykorzystanie tych metod pokazane jest na fragmencie kodu na rys. 6.16

Po znalezieniu wszystkich stawów, konieczne jest obliczenie kątów ugięć (patrz rys. 5.4, które zostaną przekazane serwomechanizmom. Fragment kodu na rys. 6.15 przedstawia przykładowe obliczanie  $q_5$ . W tych obliczeniach wykorzystuje się kąt nachylenia wektora poprowadzonego między punktem początkiem palca wskazującego, a początkiem małego palca i płaszczyzny OXY.

```
def angle_4(landmarks):
    hand = np.array([-landmarks[2][0] + landmarks[3][0], landmarks[2][1] - landmarks[3][1],
                     landmarks[2][2] - landmarks[3][2]])

    # Normal vector of [X, Y] plane
    vector_OZ = np.array([0.0, 0.0, 1.0])

    # Factor of hand and OZ, and their length
    factor = np.dot(hand, vector_OZ)
    length_0 = np.linalg.norm(hand)
    length_1 = np.linalg.norm(vector_OZ)

    # Calculate the angle between the line and plane
    angle_4 = np.degrees(np.arccos(factor / (length_0 * length_1)))

    return angle_4
```

Rysunek 6.15: Funkcja obliczania kąta obrotu piątego serwomechanizmu

Po dokonaniu obliczeń kątów, wartości wynikowe można przekształcić na sygnały PWM. Jest to przekształcenie liniowe, a sygnały są ograniczone wartościami minimalnymi i maksymalnymi. Kiedy już posiadamy obliczone długości sygnałów, konieczne jest obliczenie sumy kontrolnej i utworzenie ramki komunikacyjnej. Przykład przekształcania kątów można znaleźć na fragmencie kodu na rys. 6.17, a proces tworzenia ramki komunikacyjnej na rys. 6.18.

```

def new_elbow(landmarks):
    # Calculate delta between wrist and start of index finger
    delta_length = np.array([landmarks[1][0] - landmarks[2][0], landmarks[1][1] - landmarks[2][1],
                            landmarks[1][2] - landmarks[2][2]])

    # Calculate it's length
    length = np.linalg.norm(delta_length)

    # Get coordinates of elbow and wrist
    elbow_x, elbow_y, elbow_z = landmarks[0]
    wrist_x, wrist_y, wrist_z = landmarks[1]

    # Calculate the interpolation factor
    factor = length * 2.6 / math.sqrt((wrist_x - elbow_x)**2 + (wrist_y - elbow_y)**2 +
                                      (wrist_z - elbow_z)**2)

    new_x = wrist_x + factor * (elbow_x - wrist_x)
    new_y = wrist_y + factor * (elbow_y - wrist_y)
    new_z = wrist_z + factor * (elbow_z - wrist_z)

    # Cap new coordinates within the image
    new_x = max(0.0, min(1.0, new_x))
    new_y = max(0.0, min(1.2, new_y))
    new_z = max(0.0, min(1.0, new_z))

    # Interpolate the coordinates of the new point
    landmarks[0] = (new_x, new_y, new_z)

```

Rysunek 6.16: Estymowanie pozycji łokcia w przestrzeni

```

def angle_to_duty(angle, min_duty, max_duty, index):
    # Calculate the proportion of the angle within the range
    if index != 2:
        proportion = (angle) / (180.0)
    else:
        proportion = (angle) / (90.0)

    # Map the proportion to the duty range
    value = min_duty + proportion * (max_duty - min_duty)

    return value

```

Rysunek 6.17: Funkcja mapowania kąta obrotu na sygnał PWM

```
def createFrame(duties):
    # Define the CRC-16 function
    crc16 = crcmod.predefined.Crc('crc-16')

    # Convert duties to bytes and update the CRC-16
    for value in duties:
        crc16.update(value.to_bytes(2, byteorder='big'))

    # Get the CRC-16 value
    crc_value = crc16.crcValue

    frame = f"S {crc_value:04X} "
    for servo in duties:
        frame += f"{servo} "

    return frame + f"E\r\n"
```

Rysunek 6.18: Tworzenie ramki komunikacyjnej



# Rozdział 7

## Testy i eksperymenty

W celu zweryfikowania poprawności działania programu oraz manipulatora przeprowadzono kilka testów poszczególnych elementów. Pierwsza część testów dotyczyła sprawdzenia poprawności funkcjonowania manipulatora. Drugą część stanowiło oprogramowanie do wykrywania konfiguracji ręki. Na samym końcu przeprowadzono test całościowy, obejmujący procesy wykrywania ręki, obliczania kątów i wykonywania zadanego ruchu manipulatorem. W ostatnim teście użyto obiektu łatwego do chwycenia przez manipulator i zbadane zostały jego reakcje na zadawane mu dane.

### 7.1 Testy manipulatora

Testy manipulatora polegały na zadaniu mu wcześniej zdefiniowanych ruchów i sprawdzeniu, czy wykonuje je prawidłowo. W tym celu zdefiniowano dwie pozycje, w których manipulator mógł się znajdować oraz jeden ciągle zmieniający się kąt. Zdjęcia dwóch skrajnych zadanych pozycji widoczne są na rys. 7.1. Ruch wykonywany przez manipulator miał przypominać kiwanie ręką wraz z obracaniem jej o kąt  $\pm 45^\circ$ . Podczas testów oceniano płynność ruchów oraz zachowanie manipulatora przy różnych czasach odstępu między ruchami i różnej liczbie kroków. W czasie trwania testu zaznaczone zostały dwa punkty krańcowe i zbadany kąt między nimi. Wyznaczony kąt różnił się od zakładanego i wynosił on łącznie  $92^\circ$ . Przy 20 krokach manipulator wykonywał gwałtowne ruchy, co powodowało drgania. Ustawiając liczbę kroków na 60, ruchy były zbyt wolne. Najlepsze rezultaty przyniosło 40 kroków na jeden zadany ruch. W tym przypadku można było także skrócić odstępy czasowe między ruchami bez wywoływania efektu szarpania. Finalna forma zadawanych ruchów była satysfakcjonująca. Ruchy te odbywały się gładko i bez szarpań, co świadczyło o dobrze wykonanej konstrukcji.

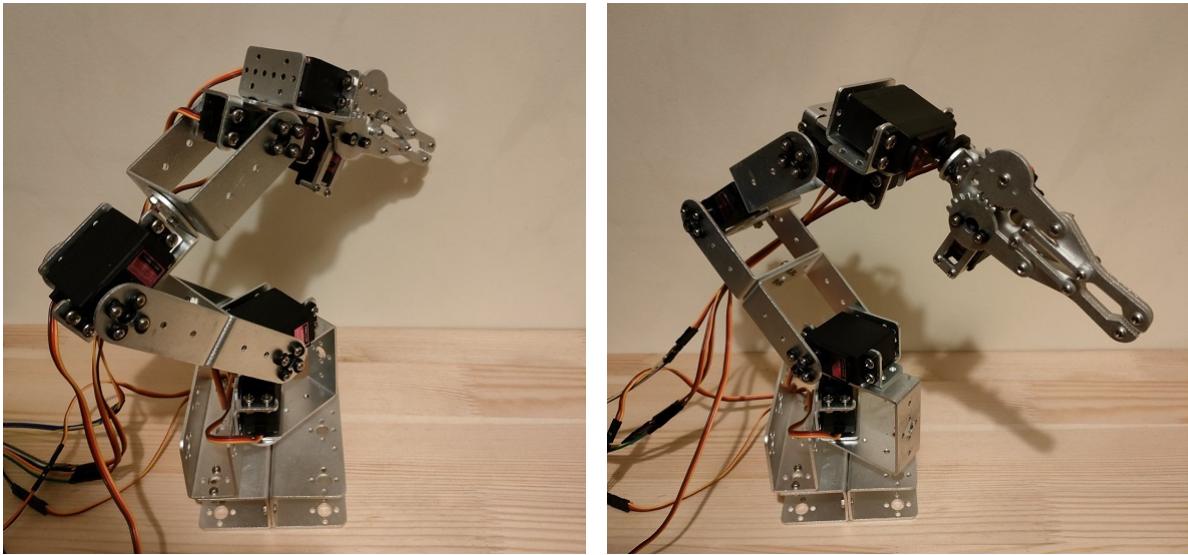
Następnie zbadana została dokładność serwomechanizmów. W manipulatorze występują dwa serwomechanizmy wyższej jakości posiadające metalowe przekładnie (serwomechanizmy nr 2 i 3, patrz rys. 5.2) oraz cztery o niższej jakości z przekładniami plastikowymi. Badaniom poddany został jeden serwomechanizm lepszej jakości (nr 2) i jeden gorszej (nr 1). Dla każdego z nich zadano 13 ruchów zmieniających ich kąt o  $15^\circ$ . Manipulator umieszczony został w polu widzenia kamer stelaża zapewniającego niezmienne pozycje kamer. Po każdym skończonym ruchu wyko-

nane zostało zdjęcie. Zdjęcia te następnie nałożono na siebie i wyznaczono jeden punkt początkowy (początek pierwszego ramienia) oraz końcowy (koniec pierwszego ramienia). Między punktami poprowadzone zostały linie, za pomocą których wyznaczono kąty obrotów. Przy ich wyznaczaniu skorzystano ze serwisu interneto-wego Ginifab [Pro], umożliwiającego łatwe ich wyznaczanie. Dla zmierzonych kątów należy uwzględnić niepewność pomiarów wynoszącą  $\pm 1^\circ$ . Test ten następnie powtórzono i porównano ze sobą wyniki. Rezultaty badania przedstawia tabela 7.1. Zauważać można, że materiał, z którego wykonano przekładnie nie ma większego znaczenia. Duże znaczenie ma jednak położenie serwomechanizmów. W przypadku drugiego serwomechanizmu niedokładności są bardziej widoczne w miejscach, w których oddziaływanie siły grawitacji jest największe.

Tabela 7.1: Tabela porównująca kąty obrotów serwomechanizmów. Pierwsza kolumna serwomechanizmu przedstawia kąty pierwszego testu, a druga drugiego.

<b>Zadane kąty</b>	<b>Pierwszy serwomechanizm</b>					<b>Drugi serwomechanizm</b>			
	$\alpha$	$\beta$	$\Delta$	$\beta$	$\Delta$	$\beta$	$\Delta$	$\beta$	$\Delta$
0°	0°	0°	0°	0°	0°	0°	0°	0°	0°
15°	13°	2°	14°	1°	15°	0°	15°	0°	0°
30°	30°	0°	30°	0°	29°	1°	29°	1°	1°
45°	46°	1°	45°	0°	46°	1°	45°	0°	0°
60°	62°	2°	61°	1°	59°	1°	59°	1°	1°
75°	76°	1°	76°	1°	75°	0°	75°	0°	0°
90°	90°	0°	90°	0°	90°	0°	91°	0°	0°
105°	104°	1°	105°	0°	102°	3°	103°	2°	2°
120°	119°	1°	119°	1°	116°	1°	116°	1°	1°
135°	136°	1°	136°	1°	131°	4°	132°	3°	3°
150°	149°	1°	149°	1°	149°	1°	149°	1°	1°
165°	165°	0°	165°	0°	162°	3°	163°	2°	2°
180°	180°	0°	180°	0°	180°	0°	180°	0°	0°

W kolejnym etapie testom podlegała warstwa komunikacyjna oraz sterowanie manipulatorem z odpowiednio dobranych poleceń mikrokontrolera sterującego manipulatorem. Ocenie zostały poddane procesy odbierania dużej ilości danych w krótkich odstępach czasu oraz sprawdzenie, czy wszystkie ruchy zostały poprawnie wykonane. Zadane ruchy były identyczne do tych występujących w pierwszym teście manipulatora. Ramki danych o nieprawidłowym formacie (patrz rozdział 5.5) zostały skutecznie odrzucone. Mikrokontroler prawidłowo odbierał, sprawdzał i przetwarzał dostarczone dane.



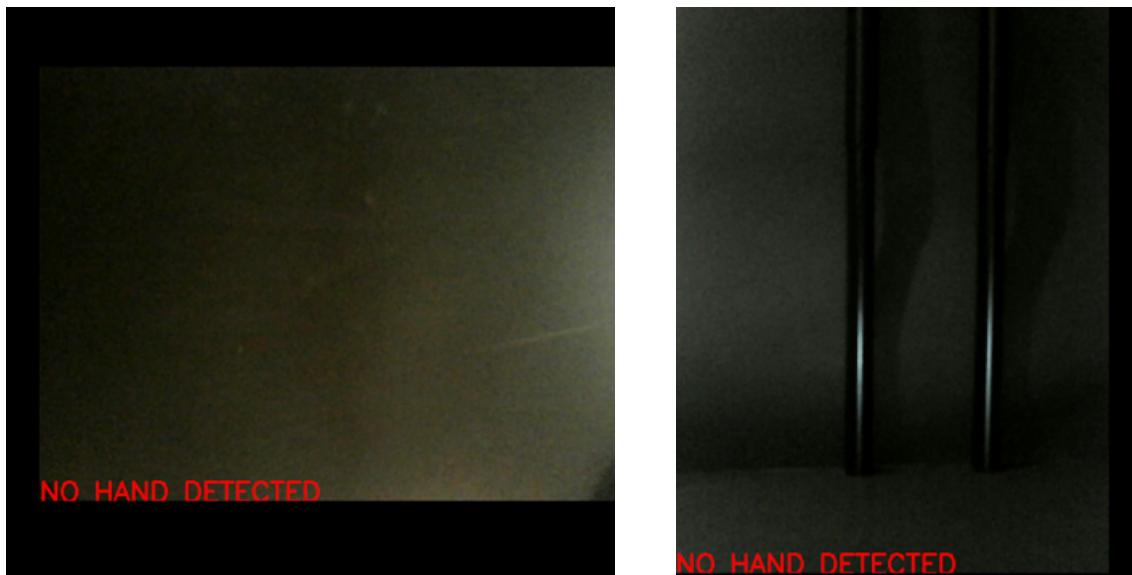
(a) Lewa pozycja manipulatora

(b) Prawa pozycja manipulatora

Rysunek 7.1: Test manipulatora – wykonywanie zadawanych ruchów

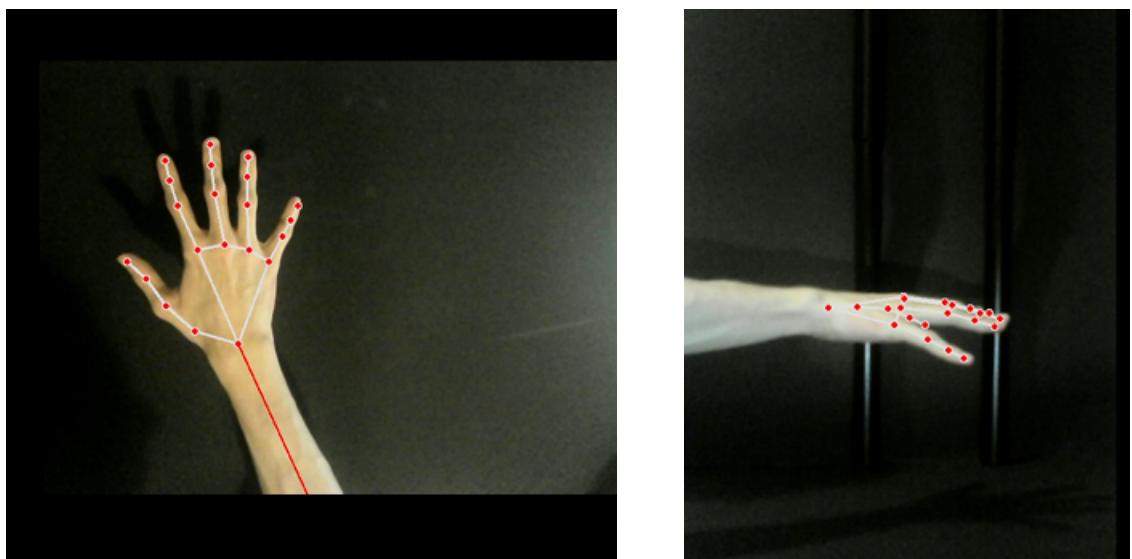
## 7.2 Testy wykrywania konfiguracji ręki

Testy wykrywania konfiguracji ręki polegały na identyfikacji ręki i sprawdzeniu poprawności obliczonych przez program kątów. Obrazy były na bieżąco pobierane ze stelaża i przetwarzane przez oprogramowanie. Na początku przeprowadzano ruchy stawami ręki, zmieniając tylko jeden kąt ugięcia. Następnie sprawdzano kąty przy ruchu całym ramieniem. Kąty  $q_1$ ,  $q_4$  i  $q_5$  (patrz rys. 5.4) były obliczane bezproblemowo, a dokładność była satysfakcjonująca. Jednakże kąt  $q_1$  może on zmieniać się szybko, gdy pozycja łokcia jest blisko barku w płaszczyźnie OXY. Kąty  $q_2$  oraz  $q_3$  były wyliczane z akceptowalną dokładnością, jednakże wspomniana dokładność była istotnie mniejsza w porównaniu z resztą kątów. Zmienna  $q_6$ , odpowiadająca za uściszkę efektora, wykazywała problemy z odczytem odległości między palcami, gdy palce były słabo widoczne na jednej z kamer lub były zasłonięte. W przypadku, gdy palce nie były widoczne na obrazie, program dokonywał estymacji ich pozycji, co prowadziło do braku pewności co do ich rzeczywistej lokalizacji. Przykład takiej sytuacji przedstawiono na rys. 7.5. Poprawnie wykryta ręka została przedstawiona na rys. 7.3, 7.4 oraz 7.5. Rys. 7.2 przedstawia obraz przed wykryciem ręki, a rys. 7.6 pokazuje obraz po usunięciu ręki z pola widzenia kamery. W takiej sytuacji kąty ugięć pozostawały takie, jakie były w ostatnim momencie, gdy ręka była jeszcze widoczna. W trakcie testów zauważono, że nie można poruszać ręką zbyt szybko, ponieważ powoduje to rozmycie obrazu otrzymanego z kamery. Wynika to z tego, że użyte kamery nie umożliwiają rejestrowania obrazu z wystarczającą częstotliwością.



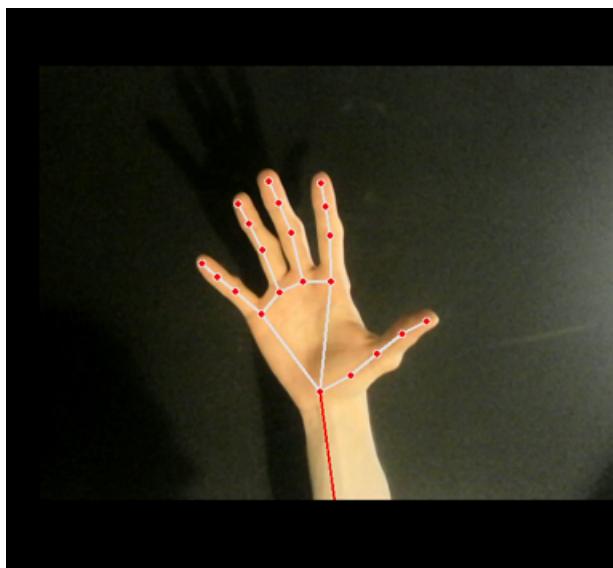
(a) Obraz otrzymanym z kamery rejestrującej widok z góry (b) Obraz otrzymanym z kamery rejestrującej widok z boku

Rysunek 7.2: Okno aplikacji z widokiem przypadku braku ręki na obrazie wraz z informacją o jej niewykryciu

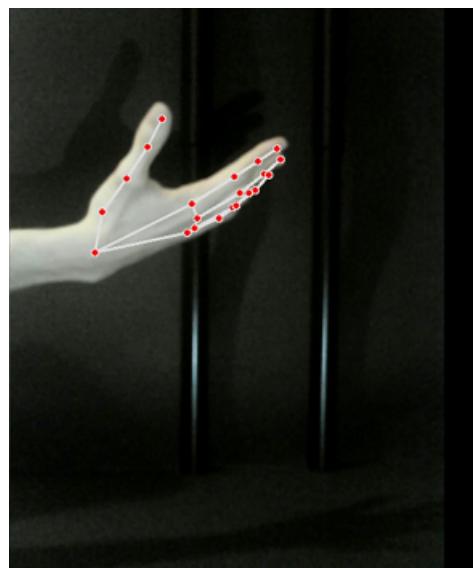


(a) Obraz otrzymanym z kamery rejestrującej widok z góry (b) Obraz otrzymanym z kamery rejestrującej widok z boku

Rysunek 7.3: Okno aplikacji z widokiem powierzchni wykrytej dłoni skierowanej w dół

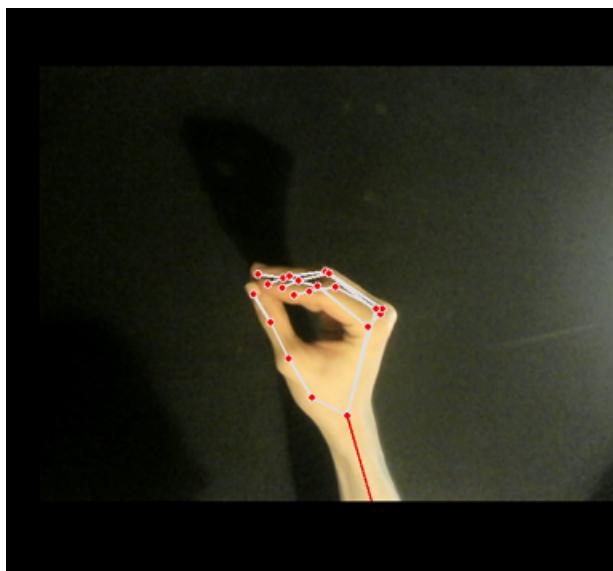


(a) Obraz otrzymanym z kamery rejestrującej widok z góry

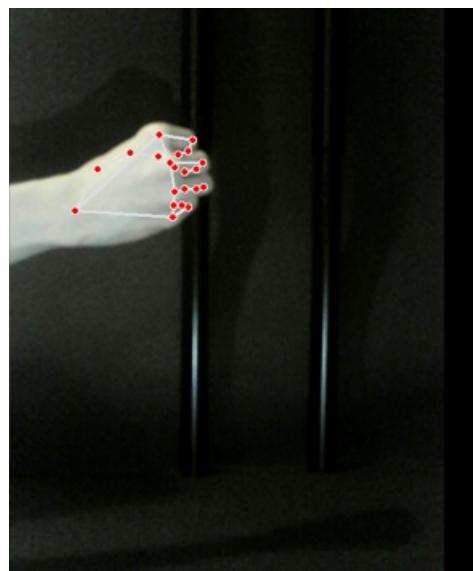


(b) Obraz otrzymanym z kamery rejestrującej widok z boku

Rysunek 7.4: Okno aplikacji z widokiem powierzchni wykrytej dłoni skierowanej w góre

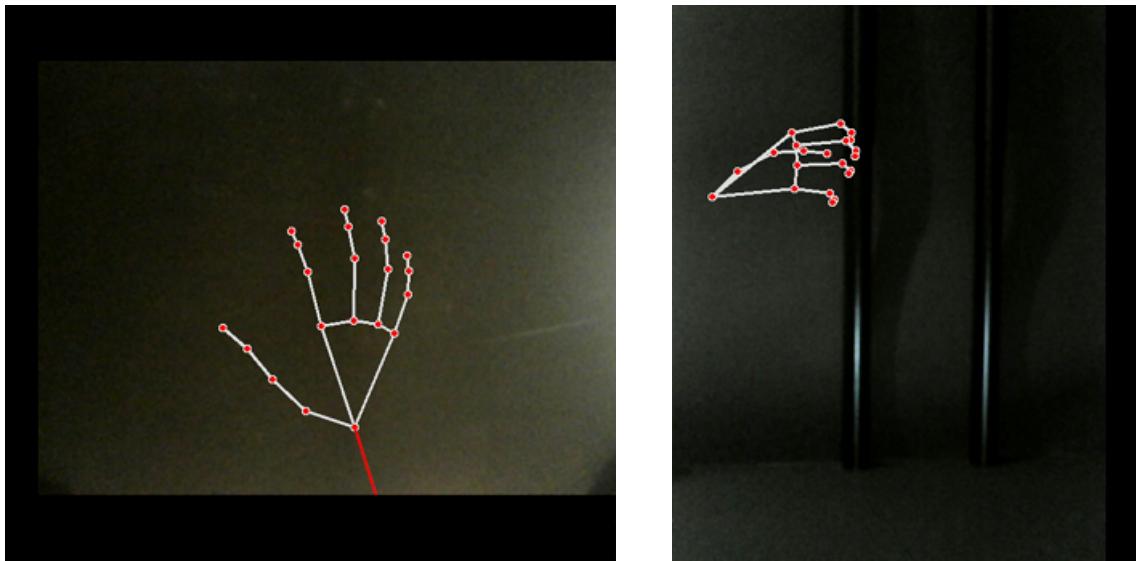


(a) Obraz otrzymanym z kamery rejestrującej widok z góry



(b) Obraz otrzymanym z kamery rejestrującej widok z boku

Rysunek 7.5: Okno aplikacji z widokiem ścisniętych palców wykrytej dłoni



(a) Obraz otrzymanym z kamery rejestrującej widok z góry (b) Obraz otrzymanym z kamery rejestrującej widok z boku

Rysunek 7.6: Okno aplikacji z widokiem zapamiętanej konfiguracji ręki po jej usunięciu z pola widzenia kamery

### 7.3 Testy całości

Finalnym testem w projekcie było sprawdzenie poziomu integracji oraz współpracy między oprogramowaniem do wyznaczania konfiguracji ręki, warstwy komunikacyjnej oraz manipulatorem. W tym celu przeprowadzono eksperyment, w którym złapano ręką mały obiekt w polu widzenia kamer i obserwowano naśladowane ruchy. Przed manipulatorem został umieszczony taki sam mały obiekt. Tak przygotowane stanowisko widoczne jest na rys. 7.7 i 7.8. Ręka wykonała ruch, który miał na celu chwycenie i podniesienie wspomnianego obiektu. Przebieg kolejnych faz eksperymentu przedstawiają rys. 7.9, 7.10, 7.11 i 7.12. Na dwóch pierwszych rysunkach widać chwytyanie obiektu, a na dwóch kolejnych jego odkładanie. Manipulatorowi udało się odwzorować zadany ruch, jednak należało wykonać go powoli. Jednym z czynników przyczyniających się do tego jest szybkość wyznaczania konfiguracji ręki. Kolejnym ważnym aspektem przyczyniającym się do zmniejszenia płynności pracy projektu jest ograniczona prędkość serwomechanizmów. Ograniczenie te jednak jest istotne, gdyż bez niego ruchy byłyby zbyt gwałtowne.

Przy pracy na tym stanowisku należy pamiętać o tym, że nie wyznaczamy pozycji barku w przestrzeni. W związku z tym należy ustawić się w adekwatnym miejscu, czyli na środku pola widzenia kamer. Ustawienie się w złym miejscu skutkować może błędymi kątami  $q_1$ ,  $q_2$  i  $q_3$  (patrz rys. 5.4). Przy zbliżaniu łokcia do przyjętej stałej pozycji barku występowały szybkie zmiany wcześniej wspomnianych kątów. Spowodowane to było małą odległością między dwoma punktami w przestrzeni. Jest to ograniczenie i wada programu, jednak jest ono nieuniknione przy braku estymacji pozycji barku.



Rysunek 7.7: Stanowisko testowe, widok początkowego stanu rozmieszczenia obiektów i położenia manipulatora

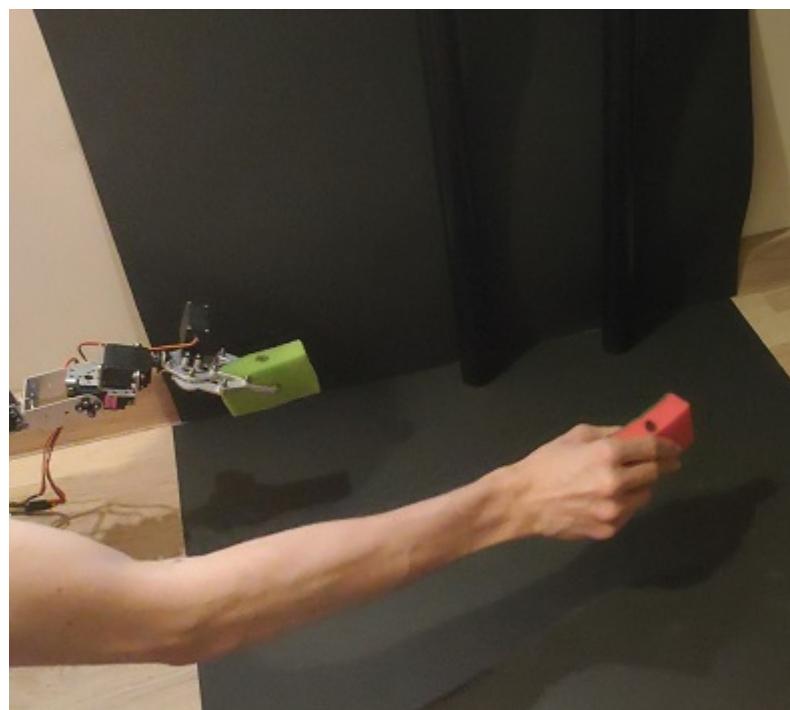


(a) Obraz otrzymanym z kamery rejestrującej widok z góry

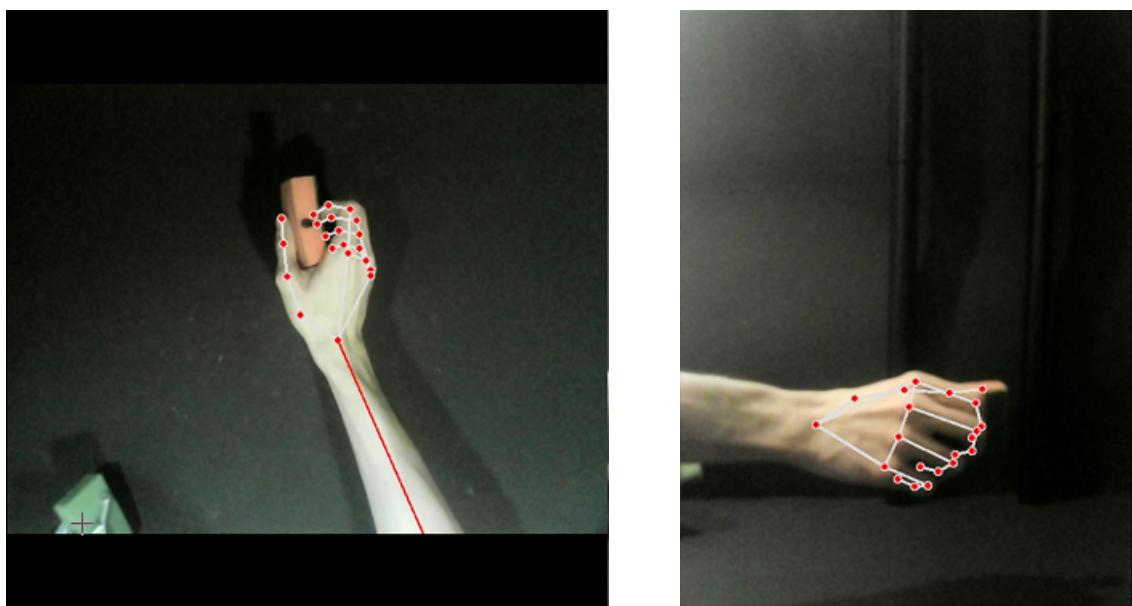


(b) Obraz otrzymanym z kamery rejestrującej widok z boku

Rysunek 7.8: Okno aplikacji z widokiem stanowiska przed rozpoczęciem testu

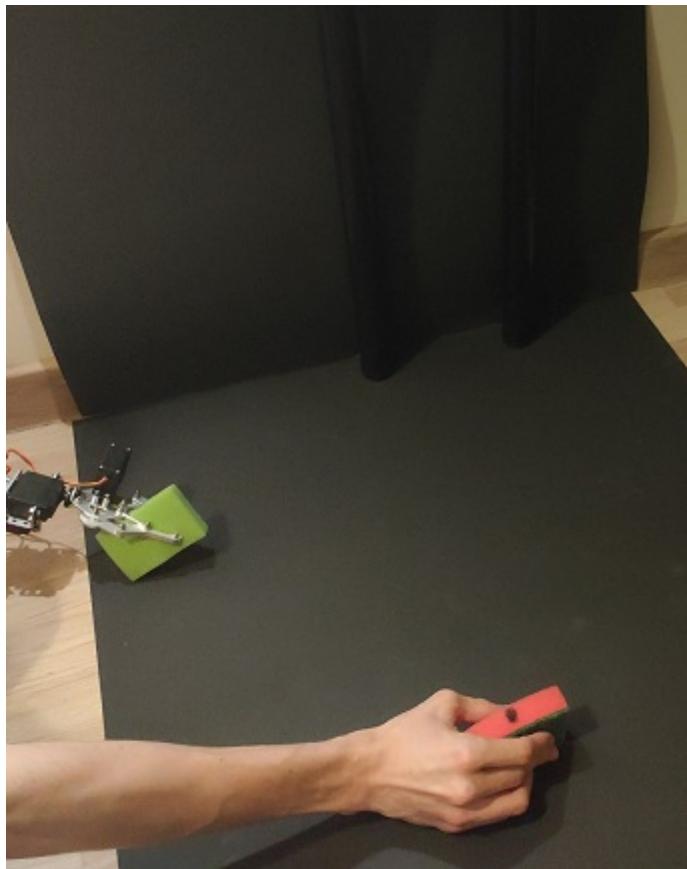


Rysunek 7.9: Stanowisko testowe, widok fazy chwytania obiektu

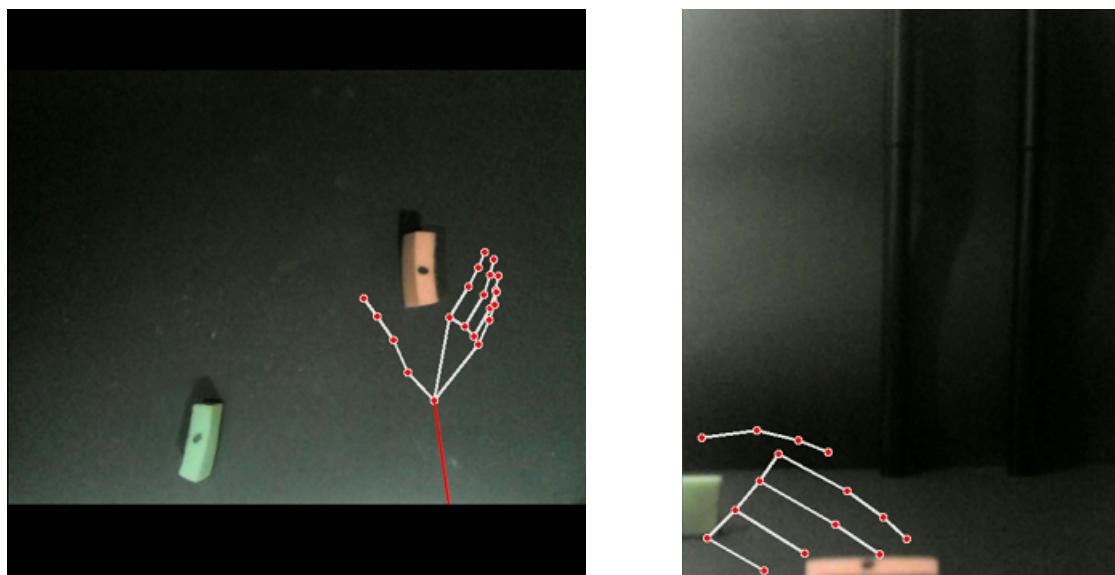


(a) Obraz otrzymany z kamery rejestrującej widok z góry (b) Obraz otrzymany z kamery rejestrującej widok z boku

Rysunek 7.10: Okno aplikacji z widokiem stanowiska po chwyceniu obiektu



Rysunek 7.11: Stanowisko testowe, widok fazy odkładania obiektu



(a) Obraz otrzymany z kamery rejestrującej widok z góry (b) Obraz otrzymany z kamery rejestrującej widok z boku

Rysunek 7.12: Okno aplikacji z widokiem stanowiska po odłożeniu obiektu

Po testach zbadano kąt jaki przyjęła ręka i manipulator, a następnie porównano wyniki. Przeprowadzono trzy takie testy. Celem tego badania było sprawdzenie dokładności z jaką manipulator naśladował ruch z punktu do punktu podczas przemieszczania obiektów. Do zbadania jej użyta została długość ramienia manipulatora oraz ręki wykonującej eksperiment i długości wektorów przemieszczenia obiektów. Do określenia kątów użyty został wzór:

$$\alpha = \arcsin\left(\frac{0,5 \cdot L}{R}\right) \quad (7.1)$$

gdzie:

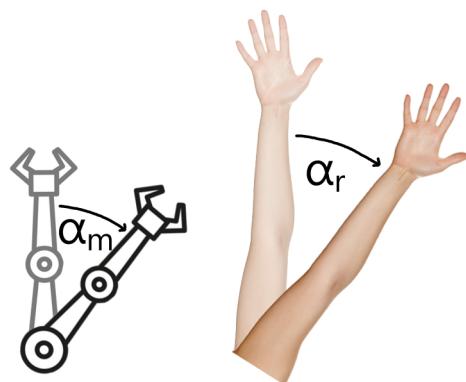
$L$  – długość wektora przemieszczenia obiektu,

$R$  – długość ramienia wykonującego przemieszczenie.

Podczas mierzenia odległości należy uwzględnić niepewność jej pomiarów. Wysokość ona  $\pm 0,1$  cm. Wynika ona z niskiej klasy urządzeń pomiarowych. W projekcie tym zakłada się, że satysfakcyjająca dokładność wynosi co najmniej  $\pm 5^\circ$ . Ze względu na ten warunek nie są konieczne bardziej dokładne urządzenia pomiarowe. Wartości wyliczonych kątów zostały zaokrąglane są do dwóch miejsc po przecinku, a stosunki do trzech (patrz tab. 7.2). W obliczeniach przyjęto długość ramienia manipulatora jako 39 cm, naśladowanego ramienia jako 75 cm. Rys. 7.13 pokazuje mierzone kąty oraz ich oznaczenia przyjęte w tabeli. W trzech wykonanych testach średnia stosunków miar kątów (manipulatora do ręki) wynosiła 0,821. Kąt przemieszczenia manipulatora był średnio mniejszy o  $2,13^\circ$ , niż kąt przemieszczenia naśladowanego ramienia.

Tabela 7.2: Tabela porównującą przemieszczenia obiektów

Przemeszczenie manipulatorem	Kąt $\alpha_m$	Przemeszczenie ręką	Kąt $\alpha_r$	Stosunek kątów
$14,9 \pm 0,1$ cm	$11,75^\circ$	$24,0 \pm 0,1$ cm	$9,20^\circ$	$1,000 : 0,783$
$14,8 \pm 0,1$ cm	$10,94^\circ$	$24,4 \pm 0,1$ cm	$9,35^\circ$	$1,000 : 0,855$
$17,4 \pm 0,1$ cm	$12,89^\circ$	$27,7 \pm 0,1$ cm	$10,63^\circ$	$1,000 : 0,825$



Rysunek 7.13: Mierzone kąty podczas eksperimentu z przemieszczaniem obiektu

# Rozdział 8

## Podsumowanie i wnioski

Głównym założeniem projektu było skonstruowanie manipulatora odwzorowującego ruchy ręki i stanowiska obserwującego tę rękę. Aby zrealizować to założenie prace podzielono na trzy główne etapy. Podczas prac przy każdym z nich spisano wnioski i spostrzeżenia, które spisano poniżej. Zakładano, że efektem finalnym pracy powinno być odwzorowywanie zadawanych ruchów z odpowiednią dokładnością za pomocą manipulatora. Takie ramieno robota mogłyby być stosowane podczas pracy zdalnej w zadaniach wymagających precyzji działania, dlatego dokładność ta nie może być niska. Sterowanie poprzez naśladowanie ruchów ręki jest wygodnym i intuicyjnym rozwiązańem. Jest to także rozwiązanie zapewniające bezpieczeństwo pracy na odległość. W dalszej części przedstawiono podsumowania etapów prac nad projektem wraz z wnioskami na ich temat.

**Manipulator** posiadający 6 stopni swobody jest w stanie odwzorować wszystkie główny ruchy ludzkiej ręki (pomijając poszczególne palce) oprócz jednego. W projekcie naśladowanie nadgarstka realizowane jest poprzez jeden serwomechanizm rotacyjny. Realistyczny nadgarstek jednak jest stawem o przegubie kulowym, tak jak bark. Ograniczenia wynikające z braku tego jednego stopnia swobody nie są jednak znaczące. Przy przeprowadzonych testach największe niedokładności, które się pojawiały spowodowane były oddziaływaniami siłami grawitacji. W pozycjach, gdzie były one największe występowały różnice między zmierzonymi, a zadanymi kątami wynoszącymi do  $4^\circ$  (patrz tab. 7.1). Średni błąd między kątami był mniejszy niż  $1^\circ$ . Odwzorowywany ruch wykonywany został płynnie i bez szarpań. Manipulator mógłby posiadać dodatkowo sprzążenie zwrotne, umożliwiające wykrywanie siły uścisku lub sprawdzania czy nastąpiła kolizja. Zastosowanie takich rozwiązań poprawiłoby bezpieczeństwo stanowiska, lecz zwiększyłoby koszty i wymagany nakład pracy przy konstrukcji. Mikrokontroler użyty w tym zadaniu jest wystarczający, nie powodował on żadnych opóźnień ani błędów oprogramowania. Możliwe byłoby zmniejszenie wymiarów stanowiska, gdyby wykorzystany został mikrokontroler będący w stanie odpowiednio szybko wykonywać operacje przetwarzania obrazów. Przy takim rozwiązańiu etap przesyłania danych do komputera stacjonarnego zostałaby pominięty.

**Wykrywanie konfiguracji ręki** jest etapem obarczonym największymi błędami. Błędy te są trudne do zmierzenia, co wynikać może z czynników takich jak zbyt

małe oświetlenie czy zasłonięte szukane części ciała. Największe rozbieżności w otrzymywanych wynikach powodowane są przez to, że pozycja barku może różnić się od pozycji zakładanej. Ustawienie się w nieadekwatnej pozycji skutkuje nagłymi zmianami kątów  $q_1$  i/lub  $q_2$  (patrz rys. 5.4). Wykrywanie palców także wiązało się z problemami podczas, gdy palce te były niewidoczne przez jedną z kamer. W sytuacji takiej następowała estymacja ich pozycji w przestrzeni co skutkowało niedokładnościami. Problem ten jednak jest nieunikniony przy wykrywaniu ręki za pomocą systemu wizyjnego. W rozdziale 7.2 omawiającym testy wykrywania konfiguracji ręki widoczne jest, że szukane stawy ręki są poprawnie wykrywane. Na wizualizacji nie widać jednak obliczonych kątów, a one zawierają mogą największe błędy, gdyż estymowane pozycje z obrazu są bardziej niedokładne. Projekt ten pokazuje, że ekonomiczne rozwiązywanie takie jak dwie kamery RGB umożliwia wykrywanie konfiguracji ręki na takim samym poziomie jak inne podobne systemy wizyjne. W celu uniknięcia problemów zawartych w projekcie należałoby zastosować inną metodę wyznaczania pozycji stawów ręki. Użyć w tym celu można na przykład rękawicy wirtualnej rzeczywistości.

**Efekt finalny** projektu jest zgodny z założeniami, jednak praca przy tym stanie wisku może być mniej intuicyjna niż zakładano. Pozycję serwomechanizmów obliczamy na podstawie kątów ugięć stawów, a nie ich pozycji w przestrzeni. Skutkować to może innym zachowaniem niż zakładano. Podczas testów różnica między zakładanym, a rzeczywistym kątem przemieszczenia obiektu wynosiła średnio  $2,13^\circ$ . W początkowych fazach projektu zakładano, że różnica ta może wynosić maksymalnie  $5^\circ$ , aby można uznać ten wynik za satysfakcyjny. W celu zwiększenia intuicyjności projektu należałoby zastosować inną metodę naśladowania ruchów ręki. Większą precyzję można otrzymać przez naśladowanie pozycji dloni w przestrzeni i na jej podstawie obliczania kinematyki odwrotnej manipulatora. Wykorzystanie takiej metody rozwiązałoby problemy z estymacją łokcia i kątami  $q_1$  oraz  $q_2$  poruszonymi w poprzednim punkcie. Takie podejście sprawiałoby jednak dodatkowe problemy sterowania manipulatorem takie jak na przykład osobliwości.

# Literatura

- [6DO] 6 DOF robot arm controlled trough a raspberry pi. <https://github.com/stawo/robot-arm>.
- [Gooa] Google Brain team. — strona dewelopera. <https://www.tensorflow.org>.
- [Goob] Google ML. — strona dewelopera. <https://developers.google.com/mediapipe>.
- [Int] Intuitive. — intuitive da vinci. <https://www.intuitive.com/en-us/products-and-services/da-vinci>.
- [Jos] Joseph Redmon. — strona dewelopera. <https://pjreddie.com/darknet/yolo/>.
- [Kof] Jonathan Kofman. Robot-Manipulator Teleoperation by Markerless Vision-Based Hand-Arm Tracking. <https://www.tandfonline.com/doi/full/10.1080/15599610701580467>.
- [Mei] Gary Meisner. Human Hand and Foot. <https://www.goldennumber.net/human-hand-foot/#:~:text=The%20ratio%20of%20the%20forearm,also%201.618%20the%20Divine%20Proportion./>.
- [MZAF] Ahmed F. Shanta Mohammed Z. Al-Faiz. Kinect-Based Humanoid Robotic Manipulator for Human Upper Limbs Movements Tracking. [https://www.scirp.org/pdf/ICA\\_2015011611313511.pdf](https://www.scirp.org/pdf/ICA_2015011611313511.pdf).
- [Mü] Vincent Mühler. Simple Hand Gesture Recognition using OpenCV and JavaScript. <https://medium.com/@muehler.v/simple-hand-gesture-recognition-using-opencv-and-javascript-eb3d6ced28a0>.
- [Ope] OpenCV team. — strona dewelopera. <https://opencv.org>.
- [Pro] Ginifab — online protractor. [https://www.ginifab.com/feeds/angle\\_measurement/](https://www.ginifab.com/feeds/angle_measurement/).
- [Wei05] Thibaut Weise. Hand tracking for virtual object manipulation. <http://www.doc.ic.ac.uk/~eedwards/StudentProjects/tw201/report.pdf>, 2005.



# Spis rysunków

5.1	Koncept projektu – komunikacja . . . . .	11
5.2	Ramie robota – zdjęcie konceptualne przód . . . . .	12
5.3	Ramie robota – zdjęcie konceptualne tył . . . . .	13
5.4	Rysunek techniczny kątów obrotów manipulatora . . . . .	14
5.5	Schemat elektryczny . . . . .	15
5.6	Schemat blokowy mikrokontrolera . . . . .	16
5.7	Stelaż – widok z boku . . . . .	17
5.8	Stelaż – widok z dołu . . . . .	17
5.9	Model kamery wykorzystanej w projekcie . . . . .	18
5.10	Schemat blokowy oprogramowania do wyznaczania konfiguracji ręki . . . . .	20
6.1	Finalny manipulator . . . . .	24
6.2	Zamontowany serwomechanizm – drugi serwomechanizm sterujący . . . . .	25
6.3	Podstawa manipulatora . . . . .	25
6.4	Połączenia mikrokontrolera . . . . .	26
6.5	Inicjalizacja serwomechanizmów na mikrokontrolerze . . . . .	27
6.6	Wykonywanie ruchów serwomechanizmów na mikrokontrolerze . . . . .	28
6.7	Sprawdzanie ramki na mikrokontrolerze . . . . .	29
6.8	Złożony stelaż . . . . .	30
6.9	Kamera góra stelaża . . . . .	31
6.10	Kamera dolna stelaża . . . . .	31
6.11	Mocowanie nogi stelaża . . . . .	32
6.12	Inicjalizacja i wykorzystanie modelu Hand biblioteki MediaPipe . . . . .	33
6.13	MediaPipe – wykrywane stawy . . . . .	33
6.14	Wykrywanie kierunku łokcia – kamera góra . . . . .	34
6.15	Funkcja obliczania kąta obrotu piątego serwomechanizmu . . . . .	35
6.16	Estymowanie pozycji łokcia w przestrzeni . . . . .	36
6.17	Funkcja mapowania kąta obrotu na sygnał PWM . . . . .	36
6.18	Tworzenie ramki komunikacyjnej . . . . .	37
7.1	Test manipulatora – wykonywanie zadawanych ruchów . . . . .	41
7.2	Okno aplikacji z widokiem przypadku braku ręki na obrazie wraz z informacją o jej niewykryciu . . . . .	42
7.3	Okno aplikacji z widokiem powierzchni wykrytej dłoni skierowanej w dół . . . . .	42
7.4	Okno aplikacji z widokiem powierzchni wykrytej dłoni skierowanej w górę . . . . .	43

7.5 Okno aplikacji z widokiemściśniętych palców wykrytej dłoni . . . . .	43
7.6 Okno aplikacji z widokiem zapamiętanej konfiguracji ręki po jej usunięciu z pola widzenia kamery . . . . .	44
7.7 Stanowisko testowe, widok początkowego stanu rozmieszczenia obiektów i położenia manipulatora . . . . .	45
7.8 Okno aplikacji z widokiem stanowiska przed rozpoczęciem testu . . . . .	45
7.9 Stanowisko testowe, widok fazy chwytania obiektu . . . . .	46
7.10 Okno aplikacji z widokiem stanowiska po chwyceniu obiektu . . . . .	46
7.11 Stanowisko testowe, widok fazy odkładania obiektu . . . . .	47
7.12 Okno aplikacji z widokiem stanowiska po odłożeniu obiektu . . . . .	47
7.13 Mierzone kąty podczas eksperymentu z przemieszczaniem obiektu . .	48