

POLITECHNIKA BIAŁOSTOCKA

WYDZIAŁ MECHANICZNY

PRACA DYPLOMOWA

INŻYNIERSKA

TEMAT: Koncepcja robota obdarzonego cechami sztucznej inteligencji rozwiązującego układankę z losowo zmieszanyimi puzzlami

WYKONAWCA: Magdalena Falkowska

imiona i nazwisko

PODPIS:

PROMOTOR: prof. dr hab. inż. Michał Kuciej

imiona i nazwisko

PODPIS:

BIAŁYSTOK 2021 ROK

Białystok, dnia.....

Magdalena Falkowska

Imiona i nazwisko studenta

100027

Nr albumu

Automatyka i Robotyka, stacjonarne

Kierunek i forma studiów

prof. dr hab. inż. Michał Kuciej

Promotor pracy dyplomowej

OŚWIADCZENIE

Przedkładając w roku akademickim 2020/2021 prof. dr hab. inż. Michałowi Kuciejowi pracę dyplomową pt.: Koncepcja robota obdarzonego cechami sztucznej inteligencji rozwiązującego układankę z losowo zmieszany puzzlami, dalej zwaną pracą dyplomową,

oświadczam, że:

- 1) praca dyplomowa stanowi wynik samodzielnej pracy twórczej/zespołowej pracy twórczej*;
- 2) wykorzystując w pracy dyplomowej materiały źródłowe, w tym w szczególności: monografie, artykuły naukowe, zestawienia zawierające wyniki badań (opublikowane, jak i nieopublikowane), materiały ze stron internetowych, w przypisach wskazywałem/am ich autora, tytuł, miejsce i rok publikacji oraz stronę, z której pochodzą powoływane fragmenty, ponadto w pracy dyplomowej zamieściłem/am bibliografię;
- 3) praca dyplomowa nie zawiera żadnych danych, informacji i materiałów, których publikacja nie jest prawnie dozwolona;
- 4) praca dyplomowa dotychczas nie stanowiła podstawy nadania tytułu zawodowego, stopnia naukowego, tytułu naukowego oraz uzyskania innych kwalifikacji;
- 5) treść pracy dyplomowej przekazanej do dziekanatu Wydziału Mechanicznego jest jednakowa w wersji drukowanej oraz w formie elektronicznej;
- 6) jestem świadomy/a, że naruszenie praw autorskich podlega odpowiedzialności na podstawie przepisów ustawy z dnia 4 lutego 1994 r. o prawie autorskim i prawach pokrewnych (Dz. U. z 2019 r. poz. 1231, późn. zm.), jednocześnie na podstawie przepisów ustawy z dnia 20 lipca 2018 roku Prawo o szkolnictwie wyższym i nauce (Dz. U. poz. 1668, z późn. zm.) stanowi przesłankę wszczęcia postępowania dyscyplinarnego oraz stwierdzenia nieważności postępowania w sprawie nadania tytułu zawodowego;
- 7) udzielam Politechnice Białostockiej nieodpłatnej, nieograniczonej terytorialnie i czasowo licencji wyłącznej na umieszczenie i przechowywanie elektronicznej wersji pracy dyplomowej w zbiorach systemu Archiwum Prac Dyplomowych Politechniki Białostockiej oraz jej zwielokrotniania i udostępniania w formie elektronicznej w zakresie koniecznym do weryfikacji autorstwa tej pracy i ochrony przed przywłaszczeniem jej autorstwa.

.....
czytelny podpis studenta

*w przypadku pracy zespołowej

Thesis: The concept of a robot equipped with artificial intelligence features solving a puzzle with randomly mixed elements

SUMMARY

This paper describes the design of an intelligent robot capable of solving a mixed 3x3 sliding puzzle game. It is an example of a device, running on a minicomputer Raspberry Pi 4B, which uses image analysis and machine learning tools. The implementation of tasks related to the location of elements was done using the OpenCV library, while the recognition of digits is the result of a combined application of libraries Tensorflow, Keras and Teachable Machine tool from Google, allowing to train any model based on images. The puzzle is solved by using a heuristic A* algorithm based on a cost function. In addition to the Raspberry Pi, the device consists of a webcam, a PWM controller, and a manipulator that moves automatically using modeling servos. Elements from robotics theory were included in the paper to successfully move the puzzle. The design meets the requirements substituted in this thesis and provides a complete mechanism to solve the puzzle.

STRESZCZENIE

W niniejszej pracy opisany został projekt inteligentnego robota, zdolnego rozwiązywać zmieszane puzzle przesuwne o wymiarach 3x3. Jest to przykład urządzenia, działającego na minikomputerze Raspberry Pi 4B, w którym zastosowano narzędzia analizy obrazu oraz uczenia maszynowego. Realizacja zadań związanych z lokalizacją elementów została wykonana za pomocą biblioteki OpenCV, natomiast rozpoznawanie cyfr jest to efekt wspólnego zastosowania bibliotek Tensorflow, Keras oraz narzędzia Teachable Machine od Google, pozwalającego wytrenować dowolny model na podstawie obrazów. Rozwiązanie łamigłówki odbywa dzięki wykorzystaniu heurystycznego algorytmu A*, opartego o funkcję kosztu. Urządzenie oprócz Raspberry Pi składa się z kamery internetowej, sterownika PWM oraz manipulatora, który porusza się automatycznie za pomocą serwomechanizmów modelarskich. W pracy zostały zawarte elementy z teorii robotyki, które umożliwiły skuteczne przesuwanie puzzli. Projekt spełnia wymogi podstawione w niniejszej pracy i stanowi kompletny mechanizm do rozwiązywania układanki.

SPIS TREŚCI

Wstęp	5
Cel i zakres	6
1. Wprowadzenie do tematyki inteligentnych robotów	7
1.1 Podstawowe definicje	7
1.2 Kinematyka manipulatorów.....	10
1.2.1 Notacja Denavita-Hartenberga	12
1.3 Sztuczna inteligencja	14
1.3.1 Heurystyka.....	16
1.4 Przykładowe konstrukcje do rozwiązywania gier logicznych.....	19
2. Zalożenia projektowe.....	21
2.1 Dobór elementów.....	23
2.2 Narzędzia i biblioteki programistyczne	30
3. Projekt Robota intelligentnego.....	32
3.1 Budowa i kinematyka manipulatora	32
3.2 Schemat połączeń części elektronicznej	40
4. Wykonanie Robota Intelligentnego	43
4.1 Przygotowanie środowiska Raspberry Pi	44
4.2 Montaż elementów robota oraz transformacja układu współrzędnych.....	51
4.3 Analiza obrazu z kamery i rozpoznanie konfiguracji	57
4.4 Ruch manipulatora.....	60
4.5 Struktura oraz działanie programu.....	62
4.6 Podsumowanie rezultatów	75
Wnioski.....	78
Bibliografia.....	80
Załączniki	83

Koncepcja robota obdarzonego cechami sztucznej inteligencji rozwiązującego układankę z losowo zmieszanyimi puzzlami.

WSTĘP

Tematyka inteligentnych robotów w ostatnich latach zyskuje coraz większą popularność, dzięki czemu w naszym życiu codziennym pojawiają się różne udogodnienia, takie jak automatyczne odkurzacze czy też całe systemy zdolne obsługiwać wybrane funkcje w domu. Zadanie postawione w niniejszej pracy może wydawać się czysto rozrywkowe, jednak proces tworzenia takiego urządzenia może być również łącznikiem do innych zastosowań oraz stanowić bazę do dalszej eksploracji tematu sztucznej inteligencji.

Stworzony robot inteligentny stanowi namiastkę praktycznego zastosowania teorii z dziedziny robotyki. Zawiera w sobie system rozpoznawania obrazu oparty o bibliotekę OpenCV oraz korzysta z możliwości Tensorflow, dzięki któremu można tworzyć zaawansowane modele uczenia maszynowego.

W kolejnych rozdziałach zostaną opisane poszczególne etapy projektowania, a następnie wykonania intelligentnego urządzenia:

- W rozdziale 1 „Wprowadzenie do tematyki inteligentnych robotów” opisano podstawy teoretyczne wprowadzające w wykorzystane w powyższej pracy metody. W powyższym rozdziale zawarto również omówienie części kinematycznej robota, jak i podrozdział poświęcony tematowi sztucznej inteligencji, w kontekście do zastosowanego w pracy algorytmu.
- W rozdziale 2 „Założenia projektowe” przedstawiono koncepcję tworzonego robota oraz opisano elementy konstrukcyjne i programistyczne potrzebne do jego działania.
- W rozdziale 3 „Projekt Robota intelligentnego” zaprezentowano zastosowanie teorii przedstawionej w rozdziale pierwszym do projektowanego ramienia manipulatora, oraz ukazano sposób podłączenia części elektronicznej.
- W rozdziale 4 „Wykonanie Robota intelligentnego” opisano poszczególne etapy wykonania robota. Powyższy rozdział zawiera instrukcje dotyczące instalacji środowiska programistycznego oraz montażu poszczególnych części zgodnie z zaproponowanym rozwiązaniem kinematycznym z rozdziału trzeciego.

Koncepcja robota obdarzonego cechami sztucznej inteligencji rozwiązującego układankę z losowo zmieszanymi puzzlami.

CEL I ZAKRES

Celem niniejszej pracy jest stworzenie robota, który będzie w stanie samodzielnie rozpoznać konfigurację cyfr znajdujących się na układance przesuwnej, a następnie ją rozwiązać poprzez ustawnienie każdej z nich na właściwe miejsce. Wykonanie przez robota wymienionych czynności pozwoli na stwierdzenie, że obdarzony został cechami sztucznej inteligencji.

Zakres pracy obejmuje:

- Przegląd podstawowych wiadomości dotyczących tematu pracy,
- Dobór elementów i wykonanie robota z gotowych podzespołów,
- Analiza rozwiązań i opracowanie odpowiedniego algorytmu do rozwiązyania układanki,
- Zaprogramowanie robota,
- Podsumowanie i wnioski.

Koncepcja robota obdarzonego cechami sztucznej inteligencji rozwiązującego układankę z losowo zmieszanyimi puzzlami.

1 WPROWADZENIE DO TEMATYKI INTELIGENTNYCH ROBOTÓW

W tym rozdziale zostały opisane podstawowe pojęcia związane z robotyką oraz sztuczną inteligencją, nakreślając obszar związany z tematem pracy. W podrozdziale 1.4 ujęto również przegląd konstrukcji oraz rozwiązań, jakie można znaleźć w literaturze, prezentując różne podejścia do tematu sztucznej inteligencji w robotyce.

1.1 Podstawowe definicje

W niniejszej pracy wielokrotnie będą powtarzały się pewne terminy, których przytoczenie pozwoli na lepsze zrozumienie metod wykorzystanych w tym projekcie. Najbardziej podstawowymi pojęciami w robotyce są *robot* oraz *manipulator* [1]:

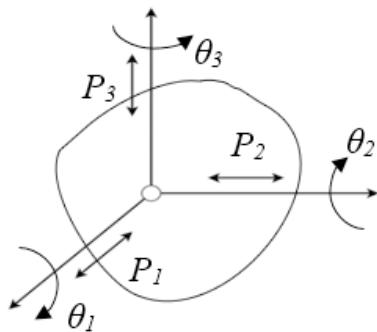
Robot – urządzenie umożliwiające realizację niektórych czynności manipulacyjnych i lokomocyjnych człowieka, posiadające pewien poziom inteligencji maszynowej.

Manipulator – część mechaniczna, urządzenie umożliwiające realizację niektórych funkcji kończyn górnych człowieka.

Powyższe definicje różnią się przede wszystkim stopniem samodzielności oraz sterowaniem. Robot jest postrzegany jako maszyna, która do działania nie wymaga ciągłej obecności człowieka, manipulatorem natomiast należy na bieżąco sterować.

Manipulatory składają się z członów połączonych ze sobą przy pomocy *par kinematycznych*, które tworzą razem *łańcuch kinematyczny* [1]. W robotyce stosuje się różne rodzaje połączeń, w celu ich podziału bierze się pod uwagę rodzaj ruchu oraz liczbę stopni swobody. Dla bryły sztywnej w przestrzeni trójwymiarowej, nieograniczonej żadnymi więzami jest 6 stopni swobody: 3 ruchy obrotowe oraz 3 postępowe, co zostało przedstawione na Rys. 1.1.

Koncepcja robota obdarzonego cechami sztucznej inteligencji rozwiązującego układankę z losowo zmieszanyymi puzzlami.



Rys. 1.1 Liczba stopni swobody bryły sztywnej [2]

Zastosowanie połączenia pomiędzy dwoma członami sprawia, że traci ono stopnie swobody. W oparciu o tą liczbę utworzono podział na V klas kinematycznych, gdzie numer klasy określa się za pomocą zależności (1.1) [3]:

$$i = 6 - s \quad (1.1)$$

gdzie:

i – numer klasy

s – liczba pozostawionych stopni swobody

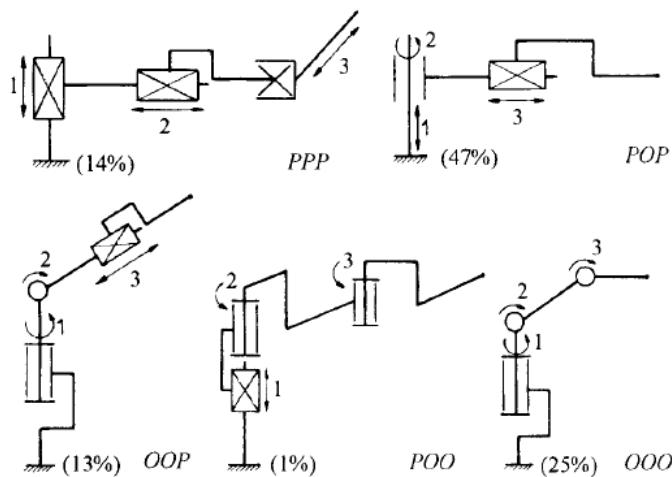
Najbardziej podstawowymi połączeniami stosowanymi w robotyce są pary kinematyczne V-tej klasy, które posiadają tylko jeden stopień swobody. Poniżej, w Tab. 1.1 przedstawiono ich charakterystykę:

Tab. 1.1. Pary kinematyczne V-tej klasy (materiał własny na podstawie [2], [3])

Lp.	Rodzaj połączenia	Rodzaje ruchu	Schemat kinematyczny
1	obrotowe (O)	1 obrót 0 postępów	
2	postępowe (P)	0 obrotów 1 postęp	

Koncepcja robota obdarzonego cechami sztucznej inteligencji rozwiązującego układankę
z losowo zmieszanyymi puzzlami.

Według tych rodzajów połączeń można wyróżnić podstawowe konfiguracje łańcuchów kinematycznych manipulatorów. Na przykładzie robota o 6-ściu stopniach swobody należy przyjąć, że pierwsze 3 człony odpowiadają za pozycjonowanie końcówki roboczej, a pozostałe 3 odpowiadają za jej orientację. Na Rys. 1.2 przedstawiono przykładowe konfiguracje manipulatorów z trzema członami, z wykorzystaniem połączeń obrotowych (O) i postępowych (P).



Rys. 1.2 Przykłady konfiguracji łańcuchów kinematycznych, wraz z ich procentową popularnością w przemyśle [2]

Pojęcie liczby stopni swobody odnosi się również do całych łańcuchów manipulatorów. Jest to liczba niezależnych zmiennych pozycji, które należy podać w celu określenia położenia wszystkich członów [4]. W prostych łańcuchach otwartych, takich jak na Rys. 1.2, liczba stopni swobody odpowiada liczbie członów ruchomych [4].

Koncepcja robota obdarzonego cechami sztucznej inteligencji rozwiązującego układankę z losowo zmieszanyymi puzzlami.

1.2 Kinematyka manipulatorów

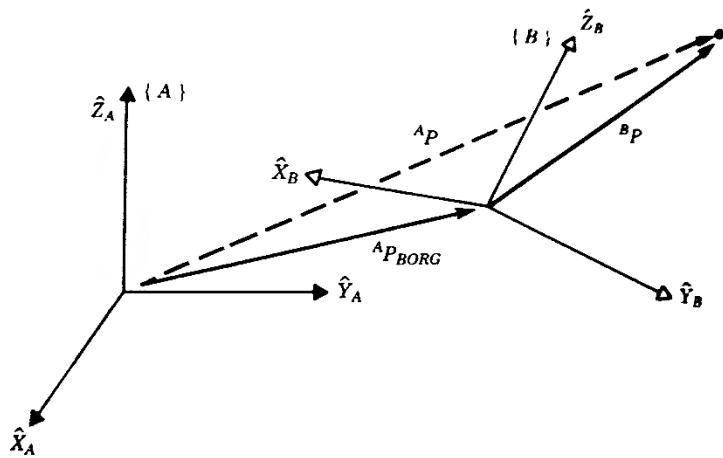
W niniejszej pracy opisanie pozycji oraz orientacji członów manipulatora w przestrzeni trójwymiarowej odbywać się będzie w układzie kartezjańskim. Pozycję dowolnego punktu w przestrzeni w danym układzie $\{A\}$ można określić poprzez wektor ${}^A P$ (1.2) orientację natomiast, poprzez związanie układu współrzędnych z danym obiektem w układzie $\{B\}$, oraz opisanie jego obrotu w stosunku do układu bazowego $\{A\}$, za pomocą macierzy obrotu oznaczonej symbolem ${}^A_B R$ (1.3) [4].

$${}^A P = \begin{bmatrix} P_x \\ P_y \\ P_z \end{bmatrix} \quad (1.2)$$

Macierz obrotu 3×3 w równaniu (1.3), składa się z trzech wektorów: ${}^A X_B$, ${}^A Y_B$, ${}^A Z_B$, których składowe opisują rzuty tych wektorów na osie współrzędnych układu $\{A\}$ oraz mogą być zapisane w postaci iloczynu skalarnego dwóch wersorów [4].

$${}^A_B R = [{}^A X_B \quad {}^A Y_B \quad {}^A Z_B] = \begin{bmatrix} {}^A X_B * X_A & {}^A Y_B * X_A & {}^A Z_B * X_A \\ {}^A X_B * Y_A & {}^A Y_B * Y_A & {}^A Z_B * Y_A \\ {}^A X_B * Z_A & {}^A Y_B * Z_A & {}^A Z_B * Z_A \end{bmatrix} \quad (1.3)$$

Do dalszego opisu posłużono się układami widocznymi na Rys. 1.3, na którym oznaczono wektor ${}^B P$ oraz szukany wektor ${}^A P$.



Rys. 1.3 Ogólne przekształcenie wektora P [4]

Koncepcja robota obdarzonego cechami sztucznej inteligencji rozwiązującego układankę z losowo zmieszanyimi puzzlami.

Opis położenia wektora P w układzie {A} można uzyskać poprzez iloczyn macierzy obrotu ${}_B^A R$ oraz wektora ${}_B^P$, a następnie dodając wektor początku układu {B} ${}_A^P_{BORG}$, co odpowiada równaniu (1.4) [4]:

$${}_A^P = {}_B^A R {}_B^P + {}_A^P_{BORG} \quad (1.4)$$

Odwzorowanie układu końcówki roboczej manipulatora w układzie bazowym wymaga opisu zarówno pozycji, jak i obrotu wszystkich poprzedzających członów. Innym sposobem przedstawienia tego odwzorowania jest równanie macierzowe (1.5). Macierz 4x4 powstała z połączenia obrotu i przesunięcia nazywana jest przekształceniem jednorodnym [4]:

$$\begin{bmatrix} {}_A^P \\ - \\ - \\ 1 \end{bmatrix} = \begin{bmatrix} {}_B^A R & | & {}_A^P_{BORG} \\ \hline 0 & 0 & 0 & | & 1 \end{bmatrix} \begin{bmatrix} {}_B^P \\ - \\ - \\ 1 \end{bmatrix} \quad (1.5)$$

Po podstawieniu wartości do powyższego równania można utworzyć zależności, dzięki którym możliwe jest obliczenie pozycji danego członu w postaci trzech parametrów x,y,z (1.6):

$$\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} c_{11} & c_{12} & c_{13} & r_1 \\ c_{21} & c_{22} & c_{23} & r_2 \\ c_{31} & c_{32} & c_{33} & r_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} \quad (1.6)$$

$$\begin{aligned} x &= c_{11}x' + c_{12}y' + c_{13}z' + r_1 \\ y &= c_{21}x' + c_{22}y' + c_{23}z' + r_2 \\ z &= c_{31}x' + c_{32}y' + c_{33}z' + r_3 \end{aligned}$$

Koncepcja robota obdarzonego cechami sztucznej inteligencji rozwiązującego układankę z losowo zmieszanyymi puzzlami.

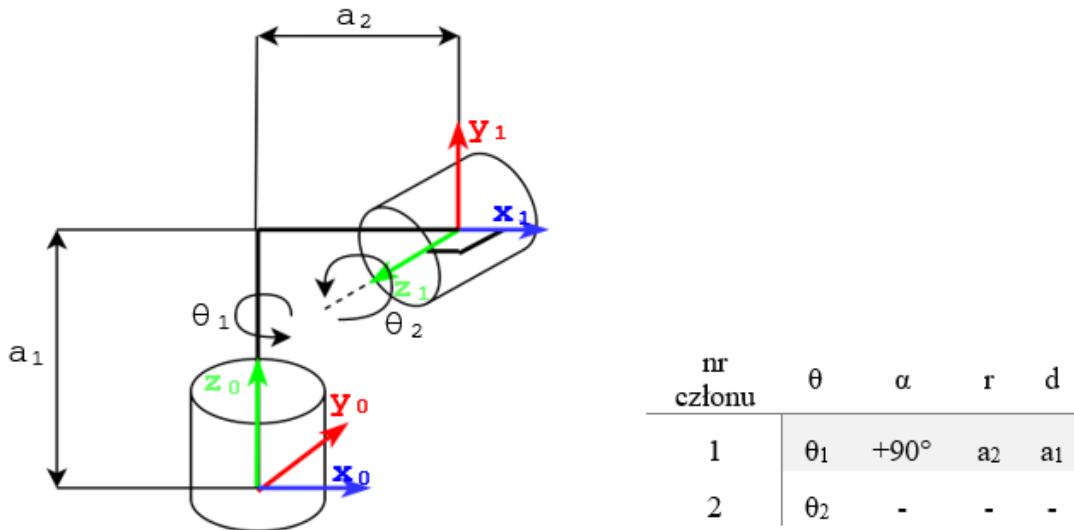
1.2.1 Notacja Denavita-Hartenberga

Notacja Denavita-Hartenberga [5], opisuje współrzędne dowolnego członu poprzez obrót i przesunięcie wzdłuż osi „z” oraz „x”. Była ona przez wiele lat modyfikowana przez różnych autorów [6] [7] [4], a ich podsumowanie można znaleźć w pozycji [2]. Na podstawie notacji z pracy [2] zostanie wyjaśniona kinematyka projektowanego w pracy dyplomowej manipulatora. Poniżej, w Tab. 1.2 opisano cztery podstawowe parametry DH oraz ich symbole:

Tab. 1.2. Opis parametrów DH (materiał własny na podstawie [2])

<i>symbol parametru</i>	<i>opis</i>
θ_n	obrót wokół osi z_{n-1}
d_n	przesunięcie wzdłuż osi z_{n-1}
a_n	obrót wokół osi x_n
r_n	przesunięcie wzdłuż osi x_n

Na schematach kinematycznych symbolem a_n będą oznaczane długości wszystkich członów, które następnie będą klasyfikowane jako r_n lub d_n . Na Rys. 1.4 ukazano przydzielanie parametrów DH na przykładzie dwóch członów obrotowych, zgodnie z opisem w Tab. 1.2.



Rys. 1.4 Przedstawienie parametrów DH wraz z tabelą (materiał własny)

Koncepcja robota obdarzonego cechami sztucznej inteligencji rozwiązującego układankę z losowo zmieszanyimi puzzlami.

Przekształcenia jednorodne omówione w poprzednim podrozdziale przyjmą następującą postać zgodnie z równaniem (1.7), gdzie H_n^{n-1} odpowiada macierzy przejścia z układu $n-1$ do n [2]. Symbolem R_n^{n-1} będzie oznaczana macierz obrotu, natomiast p_n^{n-1} będzie oznaczał wektor przesunięć.

$$H_n^{n-1} = \begin{bmatrix} \cos \theta_n & -\sin \theta_n \cos \alpha_n & \sin \theta_n \sin \alpha_n & r_n \cos \theta_n \\ \sin \theta_n & \cos \theta_n \cos \alpha_n & -\cos \theta_n \sin \alpha_n & r_n \sin \theta_n \\ 0 & \sin \alpha_n & \cos \alpha_n & d_n \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (1.7)$$

$$H_n^{n-1} = \begin{bmatrix} R_n^{n-1} & | & p_n^{n-1} \\ \hline - & - & - & - \\ 0 & 0 & 0 & | & 1 \end{bmatrix}$$

Parametry dowolnego członu n w odniesieniu do układu bazowego będzie można uzyskać dzięki iloczynowi macierzy przejść jego poprzednich członów (1.8):

$$H_n^0 = H_1^0 H_2^1 \dots H_n^{n-1} \quad (1.8)$$

Koncepcja robota obdarzonego cechami sztucznej inteligencji rozwiązującego układankę z losowo zmieszanyimi puzzlami.

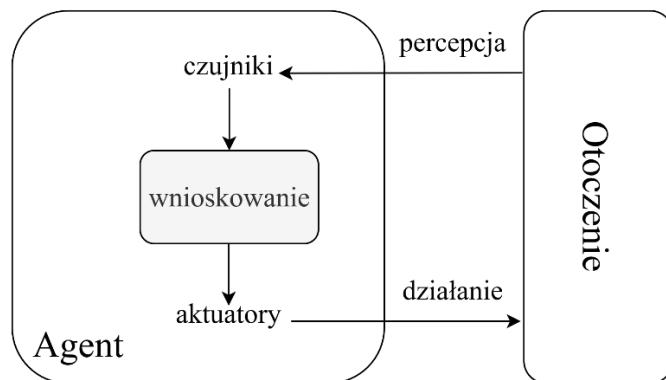
1.3 Sztuczna inteligencja

Sztuczna inteligencja jest dziedziną, która obejmuje szerokie spektrum zastosowań, a jej definicje zmieniają się zależnie od problematyki [8]. SI w robotyce łączy branże informatyki, mechaniki oraz elektroniki, dając wspólnie do stworzenia autonomicznych maszyn, które są zdolne do wykonywania czynności bez ingerencji człowieka.

Znaczenie inteligencji można rozpatrywać jako wierną ludzkiemu działaniu, lub racjonalną – bliższą matematycznej perfekcji. Te dwie kategorie łączy się z przedmiotem badań, jakim jest rozróżnienie myślenia oraz działania. W związku z różną interpretacją tworzą się cztery możliwe kombinacje: *myślienie jak człowiek, działanie jak człowiek, myślenie racjonalne* oraz *działanie racjonalne* [9]. Problem, którego dotyczy temat niniejszej pracy należy rozpatrywać w podejściu *działania racjonalnego*, inaczej zwanego metodą *inteligentnych agentów*.

Agentem w SI nazywa się urządzenie, które pobiera informacje z otoczenia i na nie oddziałuje, co może się odbywać za pomocą czujników oraz napędów. Definicja intelligentnego (racjonalnego) agenta przedstawia go jako *agenta, który z każdej możliwej sekwencji percepacji wybiera działanie o jak największej wydajności, mając na uwadze wbudowaną wiedzę oraz dane z sekwencji percepji* [9].

Mianem agenta przypada robotowi, natomiast układanka stanowi jego otoczenie. Cały układ charakteryzuje się trzema głównymi elementami: percepcją, wnioskowaniem oraz działaniem i może być przedstawiony jak na Rys. 1.5.



Rys. 1.5 Schemat agenta wzajemnie oddziałowującego z otoczeniem
(materiał własny na podstawie [9])

Koncepcja robota obdarzonego cechami sztucznej inteligencji rozwiązującego układankę z losowo zmieszanyimi puzzlami.

Poniżej, w Tab. 1.3 przedstawiono przykładowe typy agentów wraz z opisem wspomagającym skonstruowanie intelligentnego systemu. W zależności od zadań jakie ma pełnić dany agent, określa się jego odpowiednią charakterystykę poprzez miarę wydajności, otoczenie, aktuatorów oraz sensory.

Tab. 1.3. Typy agentów oraz ich charakterystyka (materiał własny na podstawie [9])

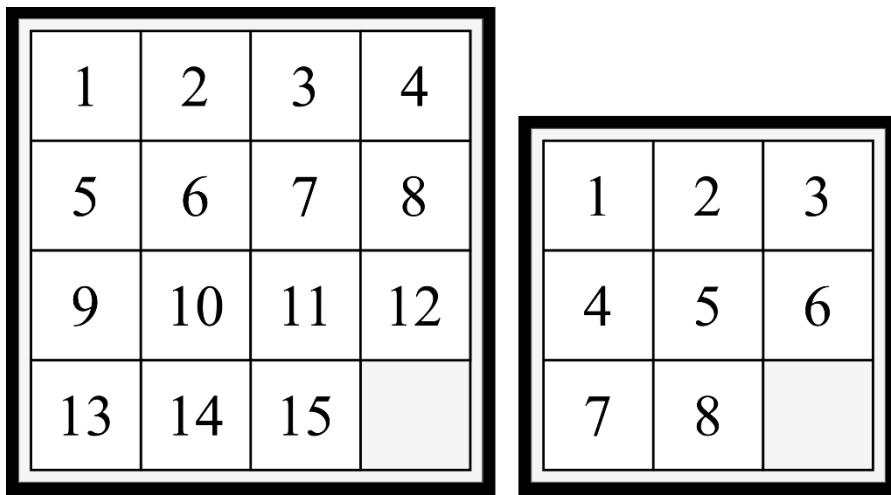
Typ agenta	Miara wydajności	Otoczenie	Aktuatorów	Sensory
Manipulator „podnieś – odlóż”	Procent poprawnie ułożonych elementów	Przenośnik taśmowy, palety	Wyświetlacz, napędy członów manipulatora	Kamera, enkodery
Autonomiczny samochód	Bezpieczeństwo, dokładność sterowania, prawidłowość identyfikacji obiektów	Infrastruktura komunikacyjna miasta, tereny niezabudowane	Wyświetlacz, głośniki, układ jazdy	Kamera, czujniki odległości, nacisku (np. na fotel), pogodowe
Łazik marsjański	Prawidłowość identyfikacji obiektów oraz wykonywanych na nich operacji	Nieregularne podłożę, warunki pogodowe	Transfer danych, układ jazdy, ramię manipulatora	Kamera, czujniki pogodowe, identyfikujące skład

Wnioskowanie stanowi obszar działań, w którym maszyna intelligentna opracowuje zestaw czynności prowadzących do rozwiązana danego zagadnienia. Zostanie on wyjaśniony na przykładzie układanki przesuwnej.

Koncepcja robota obdarzonego cechami sztucznej inteligencji rozwiązującego układankę z losowo zmieszanyimi puzzlami.

1.3.1 Heurystyka

Układanki składające się z **n** płytka stanowią klasyczny problem modelowania algorytmów z użyciem heurystyki. Najbardziej rozpowszechnione są warianty 3x3 oraz 4x4 jak na Rys. 1.6. Płytki można przesuwać dzięki jednemu pustemu miejscu w pionie oraz w poziomie, celem uporządkowania ich w kolejności rosnącej.



Rys. 1.6 Układanki przesuwne 4x4 oraz 3x3 (materiał własny)

Badania nad rozwiązywaniem zagadnień sztucznej inteligencji doprowadziły do powstania *systemu produkcji*, który jest zbiorem cech wspólnych dla różnych klas problemów [10]. Jego trzy główne elementy to:

- Zbiór stanów

Każdy *stan* (ang. *state*) odzwierciedla możliwą sytuację danego problemu. Można wśród nich wyróżnić stan startowy, docelowy oraz pośrednie. Zbiór stanów układanki przesuwnej będzie przedstawiał możliwe konfiguracje możliwe do osiągnięcia ze stanu początkowego. Droga od stanu startowego do docelowego nazywana jest *ścieżką* (ang. *path*).

- Zbiór reguł

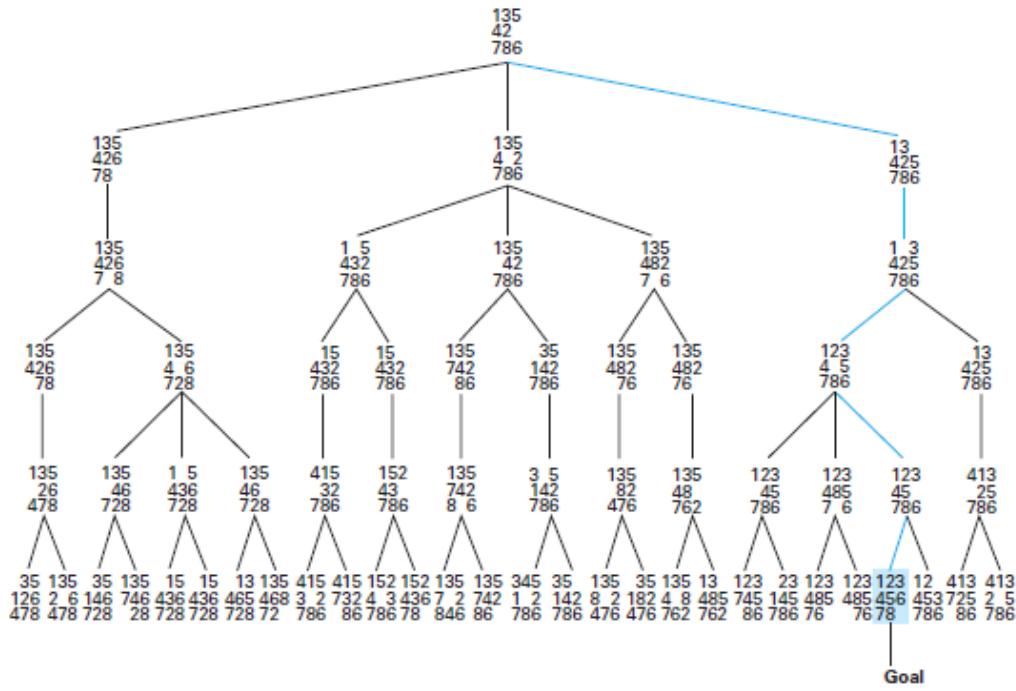
Reguły oznaczają warunki przejść z jednego stanu do drugiego. Dany element układanki można przesunąć tylko wtedy, gdy obok niego znajduje się puste miejsce oraz nie może również wyjść poza jej granice.

Koncepcja robota obdarzonego cechami sztucznej inteligencji rozwiązującego układankę z losowo zmieszanyymi puzzlami.

- System sterujący

System sterujący określa zasady wyboru następnego stanu, które doprowadzą go do stanu docelowego. Przykładowo, dla stanu o konfiguracji „1234_5786” na puste pole można przesunąć aż cztery elementy.

Do reprezentacji stanów można posłużyć się grafem stanów, lecz jako że ich liczba dla układanki 3x3 wynosi $9!/2$ [9], korzysta się jedynie z jego fragmentów dla lepszej wizualizacji. Najczęstszą metodą przedstawienia problemu jest drzewo przeszukiwania, które jest wynikiem analizy systemu sterującego i obejmuje jedynie część całościowego grafu stanów [10]. Każde drzewo rozpoczyna się od stanu początkowego, rozwidlając się do następnych osiągalnych stanów. Często stosowanym nazewnictwem przy przechodzeniu do kolejnych węzłów są *rodzic* (ang. *parent*) oraz *dziecko* (ang. *child*). Na Rys. 1.7 ukazano przykładowe drzewo o stanie początkowym „13642_786”.



Rys. 1.7 Przykładowe drzewo przeszukiwania dla konfiguracji „13542_786” [10]

Koncepcja robota obdarzonego cechami sztucznej inteligencji rozwiązującego układankę z losowo zmieszanyimi puzzlami.

- **Algorytm A***

Heurystyka pozwala na określenie miary, jak daleko program znajduje się od rozwiązania, co znacznie optymalizuje przeszukiwanie. Algorytm A* jest oparty o funkcję heurystyczną $h(x)$ oraz funkcję kosztu $g(x)$, zgodnie z równaniem (1.9) [9]:

$$f(x) = g(x) + h(x) \quad (1.9)$$

Funkcja heurystyczna w przypadku układanki opisuje dystans wszystkich pól od ich docelowej pozycji. Obliczenie tych odległości odbywa się za pomocą metryki miejskiej (Manhattan), która oznacza sumę wartości bezwzględnych różnic we współrzędnych dwóch punktów, co opisuje równanie (1.10). Funkcję kosztu natomiast, można określić jako długość ścieżki od stanu początkowego do celu. Algorytm A* podczas przeszukiwania grafu dobiera zawsze najmniejszą wartość funkcji $f(x)$.

$$d_m(p, q) = \sum_{i=1}^n |x_i - y_i| \quad (1.10)$$

Koncepcja robota obdarzonego cechami sztucznej inteligencji rozwiązującego układankę z losowo zmieszanyimi puzzlami.

1.4 Przykładowe konstrukcje do rozwiązywania gier logicznych

- **Robot do gry Jenga: „Robot Jenga: Autonomous and Strategic Block Extraction”**

(Rys. 1.8) [11]

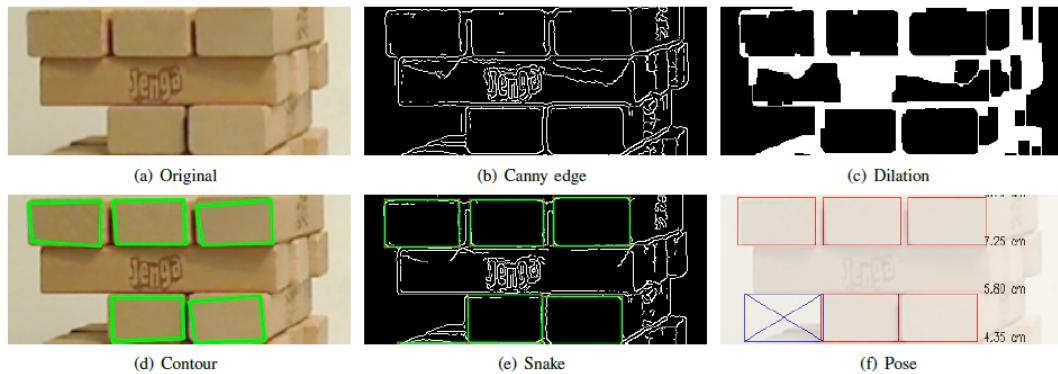


Rys. 1.8 Robot A Pioneer 3-AT z manipulatorem o pięciu stopniach swobody przy standardowej wieży Jenga [11]

Jest to przykład robota, który oprócz części logicznej musi cechować się również odpowiednią dynamiką. Gra Jenga stanowi wyzwanie dla precyzyjnej kontroli siły. Polega ona na kolejnym wyciąganiu elementów z bloku oraz ustawianiu ich na szczycie, bez utraty jego stabilności. Robot musi posiadać również odpowiedni system rozpoznawania obrazu oraz reagować na nieprzewidziane sytuacje. Strategia opisana w artykule obejmuje wybór zewnętrznych klocków, rozpoznanie oraz symulację ruchu, według której robot określa czy dane działanie może zakończyć się sukcesem. Jeżeli podczas ruchu zostanie wykryta niestabilność operacja jest przerwywana.

Robot sterowany jest poprzez komputer stacjonarny o parametrach 2.0 GHz, 2 GB RAM, a do wykonywania swojego zadania wykorzystuje jedynie dwie kamery. Na Rys. 1.9 przedstawiono rodzaje obróbki obrazu w celu estymacji położenia elementów, gdzie kluczową rolę odgrywa znalezienie konturów.

Koncepcja robota obdarzonego cechami sztucznej inteligencji rozwiązującego układankę z losowo zmieszanyimi puzzlami.



Rys. 1.9 Przetwarzanie obrazu prowadzące do lokalizacji klocków [11]

- **Robot do układania puzzli „JigsawGo: A Jigsaw Puzzle Restoration Game With A Robotic Arm”** (Rys. 1.10) [12]



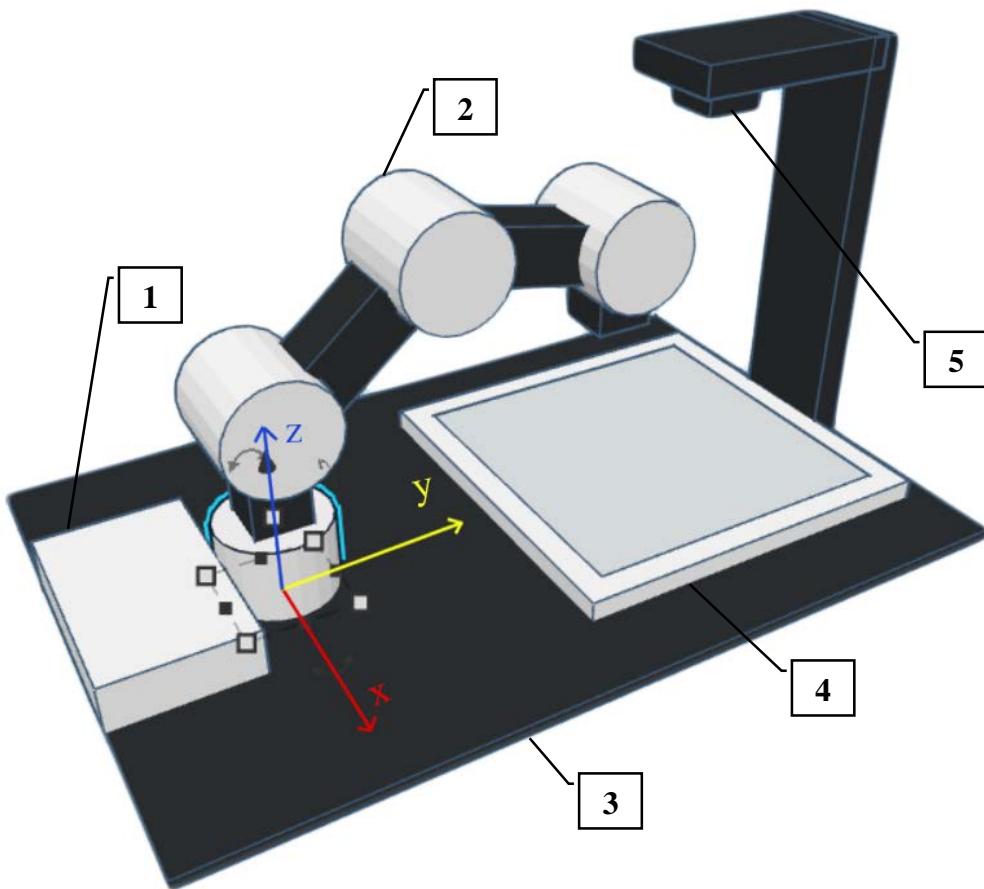
Rys. 1.10 Robot do układania puzzli za pomocą ramienia [12]

Publikacja opisuje automatycznego robota, który potrafi rywalizować z człowiekiem w układaniu tradycyjnych puzzli. Jedną z motywacji stworzenia takiego systemu może być zapotrzebowanie na automatyzację ciężkich i czasochłonnych zadań wykonywanych przykładowo w kryminalistyce, gdzie łączone są w całość pocięte dokumenty. Robot w pierwszej kolejności lokalizuje elementy, a następnie odnajduje punkty charakterystyczne na obrazie wzorowym. Dopasowanie elementów dokonuje się za pomocą algorytmu SURF.

Koncepcja robota obdarzonego cechami sztucznej inteligencji rozwiązującego układankę z losowo zmieszanyymi puzzlami.

2 ZAŁOŻENIA PROJEKTOWE

Realizacja wcześniej określonego celu projektu wymaga, oprócz zbudowania samej konstrukcji robota, również zaprojektowanie jego otoczenia. Zaproponowano rozwiązanie, w którym manipulator zostaje przytwierdzony do aluminiowej płyty, tworząc jednocześnie miejsce na swobodne umieszczenie układanki. System wizyjny robota będzie zamontowany bezpośrednio nad nią, umożliwiając skuteczną analizę pól oraz ich lokalizację w płaszczyźnie „xy”. Na Rys. 2.1 przedstawiono grafikę poglądową, na której widoczny jest planowany układ elementów.



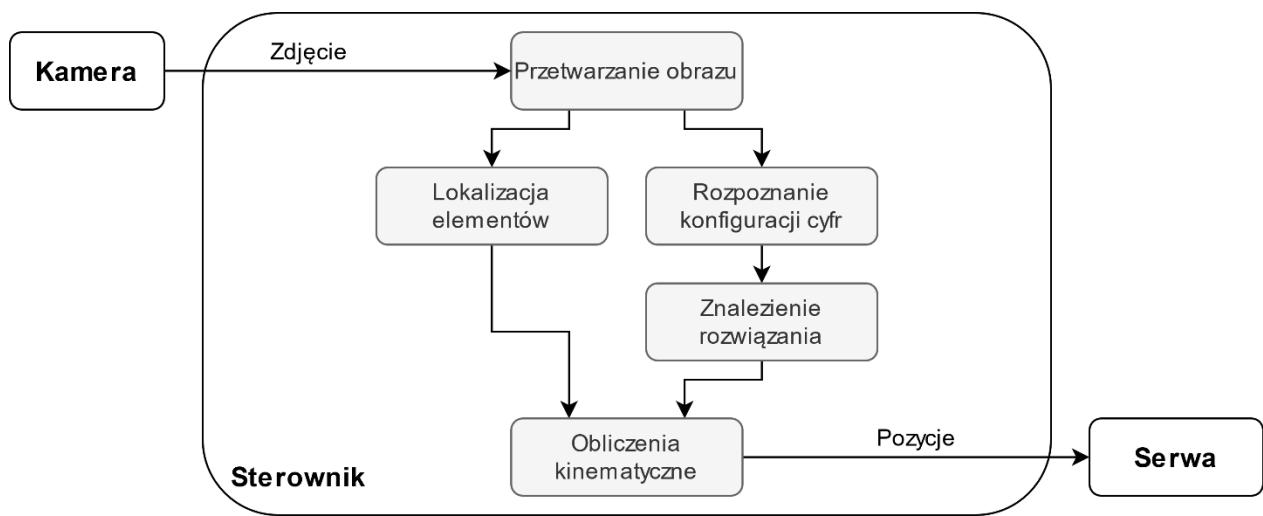
Rys. 2.1 Planowany układ elementów konstrukcji (materiał własny w Tinkercad [13]):
1 – sterownik, 2 – manipulator, 3 – płyta aluminiowa, 4 – układanka przesuwna,
5 – kamera cyfrowa

Koncepcja robota obdarzonego cechami sztucznej inteligencji rozwiązującego układankę z losowo zmieszanyimi puzzlami.

Podsumowując, planowana konstrukcja będzie składać się z następujących elementów:

- arkusz płyty aluminiowej o rozmiarze A4,
- układanka przesuwna o wymiarach 3x3,
- manipulator przegubowy oparty o serwomechanizmy modelarskie typu standard,
- układ sterujący wraz zasilaniem,
- system wizyjny – kamera cyfrowa.

Robot będzie wykonywał zdjęcie zmieszanych puzzli, a następnie je przetwarzał celem identyfikacji cyfr oraz określenia ich lokalizacji na planszy. Rozpoznawanie obrazu odbędzie się przy pomocy dostępnych bibliotek oraz narzędzi uczenia maszynowego. Korzystając z danych o konfiguracji cyfr, algorytm rozpoczęnie szukanie optymalnego rozwiązania, czego efektem będzie sekwencja ruchów, jaką ma wykonać manipulator. Ruch ramienia w konkretne pozycje będzie możliwy dzięki wykonanym obliczeniom kinematycznym dla każdego serwomechanizmu. Schemat blokowy omówionego działania przedstawia Rys. 2.2.



Rys. 2.2 Schemat działania programu (materiał własny)

Koncepcja robota obdarzonego cechami sztucznej inteligencji rozwiązującego układankę z losowo zmieszanyimi puzzlami.

2.1 Dobór elementów

- **Otoczenie**

Koncepcja robota nie zakłada sprawdzania poprawności wykonanych ruchów, a założona konstrukcja wraz z serwomechanizmami modelarskimi może posiadać luzy powodujące niedokładności w pozycji całego ramienia. Przy wyborze odpowiedniej układanki kierowano się swobodą ruchów oraz idealnym dopasowaniem elementów, który uniemożliwi przypadkowe zablokowanie ruchu. Układanka uzyta w projekcie przedstawiona na Rys. 2.3 posiada również magnesy, dzięki którym elementy są dodatkowo naprowadzane na właściwe pole.



Rys. 2.3 Układanka przesuwna liczbowa 3x3 z magnesami pozycjonującymi (materiał własny)

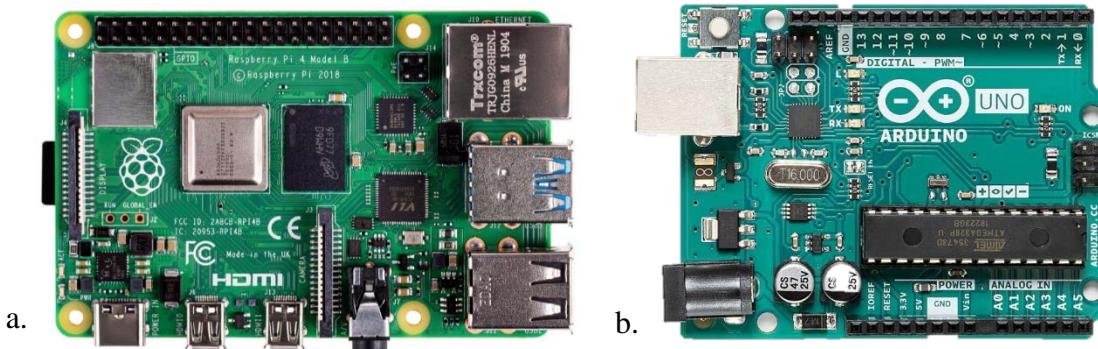
Cała konstrukcja będzie opierać się na aluminiowej planszy, na której będzie można zamontować zarówno manipulator jak i kamerę, zapewniając ich stałe położenie względem siebie. Kamera przytwierdzona bezpośrednio nad układanką zminimalizuje błąd obliczanych pozycji.

Koncepcja robota obdarzonego cechami sztucznej inteligencji rozwiązującego układankę z losowo zmieszanyymi puzzlami.

- **Raspberry Pi 4B, 2GB RAM**

Raspberry Pi, w odróżnieniu do popularnego zestawu uruchomieniowego jakim jest Arduino, nie jest oparte o mikrokontroler, których zazwyczaj używa się do sterowania robotów. Popularne są obecnie gotowe rozwiązania, takie jak płytki rozwojowe Nucleo z STM32, dzięki którym nauka programowania robotów stała się prostsza. Wybór odpowiedniej platformy związany jest z zapotrzebowaniem związanym z budową konkretnego urządzenia.

Płytki Arduino jest odpowiednia, jeżeli w planowanym projekcie zamierza się wykorzystać proste czujniki bez zaawansowanych funkcji, a program nie wymaga dużej mocy obliczeniowej. Raspberry Pi natomiast, świetnie się sprawdzi w dziedzinie analizy obrazu oraz dostępności narzędzi programistycznych. Jego przewagą jest również obecność systemu operacyjnego Linux, na którym można zainstalować różne biblioteki do uczenia maszynowego takie jak Tensorflow czy Keras. Raspberry Pi posiada także możliwość uruchamiania przetrenowanych modeli, jednak nie należy go stosować do ich trenowania, w tym celu stosuje się bardziej zaawansowane platformy takie jak Nvidia Jetson Nano, o dużo większych możliwościach. Poniżej, na Rys. 2.4 przedstawiono zdjęcia modeli Raspberry Pi oraz Arduino:



Rys. 2.4 Platformy wykorzystywane w robotyce:
a. Raspberry Pi 4B [14],
b. Arduino Uno [15]

Koncepcja robota obdarzonego cechami sztucznej inteligencji rozwiązującego układankę z losowo zmieszonymi puzzlami.

Poniżej w Tab. 1.4. przedstawiono najważniejsze cechy Raspberry Pi modelu B oraz porównano ze sobą poszczególne generacje.

Tab. 1.4. Porównanie Raspberry Pi 4B z poprzednimi generacjami [16]

			
Model	Raspberry Pi 3 model B	Raspberry Pi 3 model B+	Raspberry Pi 4 model B
Wydanie	3 - 2016 r.	3 - 2018 r.	6 - 2019 r.
Procesor	Broadcom BCM2837 64-bit	Broadcom BCM2837B0 64-bit	Broadcom BCM2711 64-bit
Rdzeń	Quad-Core ARM Cortex A53		Quad-Core ARM Cortex-A72
Pamięć RAM	1 GB		2, 4 lub 8 GB
Taktowanie	1,2 GHz	1,4 GHz	1,5 GHz
Architektura	ARMv8-A		
Systemy operacyjne	Raspberry Pi OS, Windows 10 IoT		
Pamięć	karta microSD		
Zasilanie	5,1 V / 2,5 A		5,0 V / 3 A
	port microUSB		port USB C
Interfejs USB	4x USB 2.0		2x USB 2.0 2x USB 3.0
Komunikacja	UART, SPI, I2C, GPIO		

Raspberry Pi model B cechuje się większą pamięcią RAM oraz procesorem z taktowaniem 1,5GHz, co sprzyja tworzeniu bardziej zaawansowanych aplikacji. Komunikacja z urządzeniem może odbywać się za pomocą portów UART, SPI, I2C, GPIO oraz USB.

Koncepcja robota obdarzonego cechami sztucznej inteligencji rozwiązującego układankę z losowo zmieszanymi puzzlami.

- **Kamera internetowa A4Tech Anti-glare Webcam (PK-810G)**

Jest to kamera USB o rozdzielczości 640x480 pikseli pozwalająca na swobodny montaż. W specyfikacji urządzenia ujęto kompatybilność z systemem operacyjnym Linux, co umożliwia jego pracę również w systemie Raspberry Pi. Kamera posiada dodatkowy przycisk, który może zostać wykorzystany jako wyzwalacz. Poniżej w Tab. 1.5 zostały opisane jej podstawowe parametry:

Tab. 1.5. Kamera A4Tech Anti-glare Webcam (PK-810G) [17]

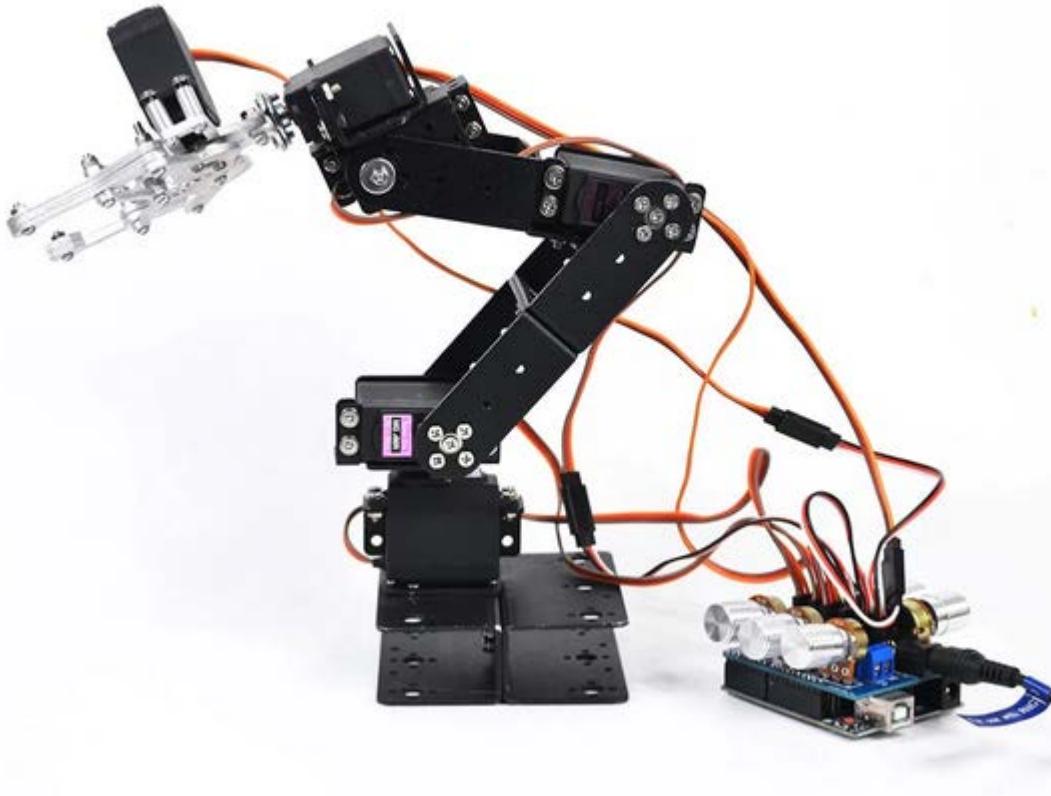
Rozdzielczość	480P, 640*480 Px
Soczewki	Antyrefleksywne
Kąt widzenia	52°
Rodzaj skupienia	Fixed focus
Kompresja	MJPEG
Liczba klatek/s	30fps
USB	2.0



- **Manipulator**

Obecnie na rynku dostępnych jest wiele elementów montażowych, dzięki którym można złożyć własną konstrukcję manipulatora, używając do tego celu serwomechanizmów modelarskich. Zdecydowano się na zakup części mechanicznej robota edukacyjnego oferowanego przez firmę Small Hammer, widocznego na Rys. 2.5. Z wybranych elementów zestawu zostanie złożony manipulator odpowiadający potrzebom zadania, jakim jest przesuwanie elementów układanki.

Koncepcja robota obdarzonego cechami sztucznej inteligencji rozwiązującego układankę z losowo zmieszanyymi puzzlami.



Rys. 2.5 Robot edukacyjny firmy Small Hammer z platformą Arduino, z możliwością ręcznego sterowania poprzez potencjometry [18]

Ważnym elementem budowy manipulatora są również odpowiednie serwomechanizmy, które będą w stanie utrzymać wyprostowane ramię oraz je swobodnie unosić. W zestawie znajdują się serwa modelu Tower Pro MG996R, którego specyfikacja znajduje się w Tab. 1.6. Najbardziej kluczowym parametrem jest moment obrotowy, który przy zasilaniu napięciem 4.8V wynosi 9,4 kg*cm. Robot tej firmy został zaprojektowany do podnoszenia małych przedmiotów, więc będzie wystarczający do zadań związanych z przesuwaniem układanki. Przykładowo, jeśli ramię manipulatora będzie miało długość 15cm, to przy napięciu 4,8V serwo udźwignie około 0,63kg.

Koncepcja robota obdarzonego cechami sztucznej inteligencji rozwiązującego układankę z losowo zmieszanymi puzzlami.

Tab. 1.6. Specyfikacja Tower Pro MG996R [19]

Napięcie zasilania:	od 4,8 V do 6,0 V	
Zakres ruchu:	od 0 ° do 180 °	
Wymiary:	40,7 x 19,7 x 42,9 mm	
Zakres temperatur:	od 0 °C do 55 °C	
Długość przewodu:	32 cm	
Masa:	55 g	
	Napięcie 4,8V	Napięcie 6V
Moment obrotowy:	9,4 kg*cm	11 kg*cm
Prędkość:	0,19 s/60°	0,15 s/60°

Jako że końcówka robocza manipulatora nie będzie odpowiedzialna za podnoszenie przedmiotów, do jej orientacji użyty zostanie serwonapęd o mniejszym momencie obrotowym - Serwo PowerHD HD-3001HB. Jego podstawowe parametry to prędkość: 0,12 s/60° oraz moment: 4,4 kg (6 V). Na Rys. 2.6 przedstawiono wykorzystane serwomechanizmy:



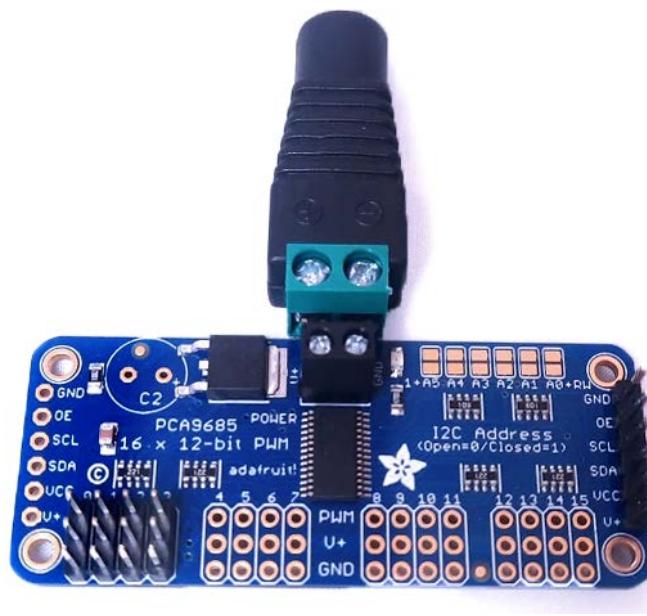
Rys. 2.6 Serwomechanizmy modelarskie TowerPro oraz PowerHD (materiał własny)

Koncepcja robota obdarzonego cechami sztucznej inteligencji rozwiązującego układankę z losowo zmieszanyymi puzzlami.

- **Adafruit PCA9685 16-Channel Servo Driver - sterownik serwomechanizmów**

Moduł przedstawiony na Rys. 2.7 umożliwia sterowanie serwomechanizmami poprzez połączenie I2C oraz zapewnia oddzielny układ do ich zasilania. Za część logiczną odpowiada sterownik PCA9685, który posiada 16 kanałów do sterowania 12-bitowym sygnałem PWM. Jego wewnętrzny oscylator umożliwia dostosowanie częstotliwości sygnału od 24 Hz do 1526 Hz.

Producenci tego typu układów zapewniają podstawowe biblioteki umożliwiające odpowiednie skonfigurowanie PCA9685 do pracy z serwomechanizmami. Do sterownika zostały przylutowane cztery piny PWM na obsługę serwonapędów oraz komunikacyjne wraz zasilaniem.



Rys. 2.7 Sterownik PCA9685 firmy Adafruit z przylutowanymi pinami (materiał własny)

Koncepcja robota obdarzonego cechami sztucznej inteligencji rozwiązującego układankę z losowo zmieszanyimi puzzlami.

2.2 Narzędzia i biblioteki programistyczne

- **Język Python 3** (Rys. 2.8) – wysokopoziomowy język, który wykorzystuje się często w aplikacjach sztucznej inteligencji ze względu na jego prostotę oraz klarowność. Używa się go również do programowania Raspberry Pi.



Rys. 2.8 Logo języka Python [20]

- **Biblioteka OpenCV 4** (Open Source Computer Vision Library, Rys. 2.9) – otwarta biblioteka stosowana w wizji komputerowej oraz uczeniu maszynowym. Algorytmy zawarte w zbiorze mogą przykładowo być używane do wykrywania i rozpoznawania twarzy oraz identyfikacji i śledzenia obiektów. Biblioteka ta jest szeroko wykorzystywana w firmach, grupach badawczych oraz przez instytucje rządowe. Obsługuje systemy Windows, Linux, Android i Mac OS. OpenCV jest napisana w C++ i posiada szablonowy interfejs. Biblioteka ta pozwoli na skuteczne rozpoznanie układanki oraz określenie jej lokalizacji, dzięki czemu użytkownik będzie musiał jedynie ustawić ją w odpowiedniej orientacji – prostopadle do manipulatora.



Rys. 2.9 Logo biblioteki OpenCV [21]

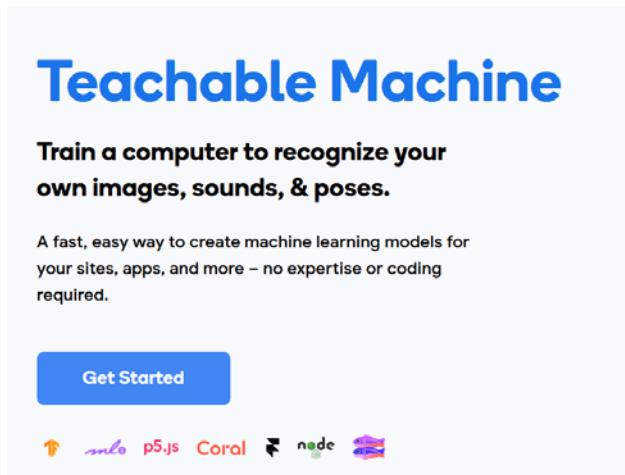
Koncepcja robota obdarzonego cechami sztucznej inteligencji rozwiązującego układankę z losowo zmieszanyimi puzzlami.

- **Biblioteka Tensorflow 2 oraz Keras** (Rys. 2.10) – Tensorflow jest to biblioteka stworzona na potrzeby uczenia maszynowego, która szczególnie skupia się na trenowaniu i wnioskowaniu głębokich sieci neuronowych. Keras natomiast, jest to wysokopoziomowe API TensorFlow 2: interfejs do rozwiązywania problemów związanych z uczeniem maszynowym, z naciskiem na nowoczesne głębokie uczenie. Keras umożliwia uruchamianie modeli na różnych urządzeniach, w tym na minikomputerze Raspberry Pi.



Rys. 2.10 Logo bibliotek Tensorflow i Keras [22] [23]

- **Teachable Machine od Google** (Rys. 2.11) – jest to narzędzie przeglądarkowe, dzięki któremu można wytrenować różnego rodzaju modele związane z obrazami lub dźwiękiem. Jest to dobre rozwiązanie w przypadku zaawansowanych zadań takich jak rozpoznawanie cyfr. Dodatkowym atutem jest fakt, że w celu wytrenowania nie trzeba wykorzystywać zasobów sprzętowych. Raspberry Pi po zainstalowaniu bibliotek Tensorflow i Keras jest w stanie z powodzeniem uruchamiać tego typu modele.



Rys. 2.11 Witryna Teachable Machine [24]

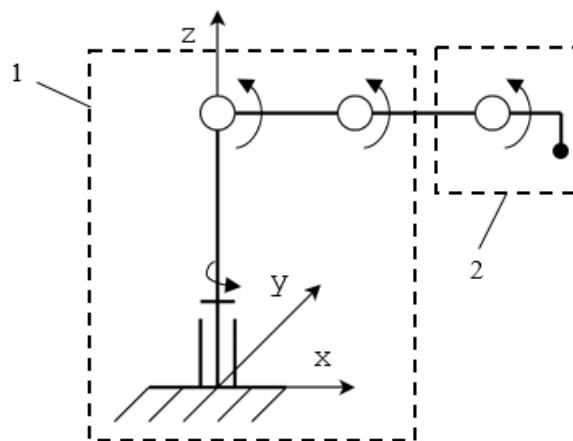
Koncepcja robota obdarzonego cechami sztucznej inteligencji rozwiązującego układankę z losowo zmieszanyymi puzzlami.

3 PROJEKT ROBOTA INTELIGENTNEGO

W tym rozdziale opisano budowę i kinematykę manipulatora, schematy połączeń części elektronicznej oraz opisano algorytm, który zostanie wykorzystany w programie.

3.1 Budowa i kinematyka manipulatora

Konstrukcje manipulatorów różnią się zależnie od postawionego celu, projektując odpowiednio układ członów można uniknąć zbędnych komplikacji związanych z szukaniem rozwiązań kinematycznych. Do wykonania zadania zaproponowano konfigurację na bazie typu antropomorficznego (OOO), co jest widoczne na Rys. 3.1:



Rys. 3.1 Schemat kinematyczny manipulatora (materiał własny): 1– pozycjonowanie,
2– orientowanie końcówki

W tym przypadku wymaga się trzech stopni swobody do pozycjonowania członu roboczego, oraz jednego dodatkowego pozwalającego zorientować końcówkę roboczą prostopadle do elementu układanki.

W celu wykonania poprawnego schematu kinematycznego zmontowano oraz przetestowano zakresy ramienia manipulatora bez podłączania części elektronicznej (Rys. 3.2). Końcówkę roboczą zbudowano ze zmatowanej gumowej podkładki przytwierdzonej do obudowy czwartego serwomechanizmu (Rys. 3.3).

Koncepcja robota obdarzonego cechami sztucznej inteligencji rozwiązującego układankę z losowo zmieszanyymi puzzlami.



Rys. 3.2 Złożone ramię manipulatora (materiał własny)

Ramię na Rys. 3.2 znajduje się w położeniu $x_0 = 0$, jednak jego bazowym położeniem będzie $y_0 = 0$, czyli wzdłuż osi x_0 . Zostanie to przedstawione w dalszej części.

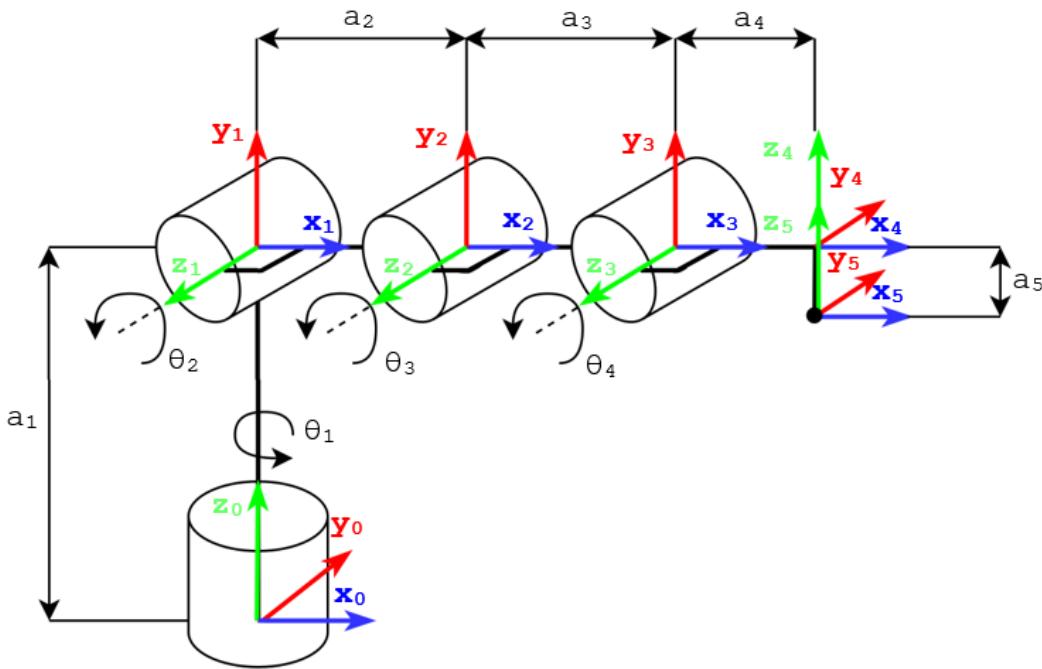


Rys. 3.3 Umiejscowienie końcówki roboczej (materiał własny)

- **Zadanie proste kinematyki**

Zadanie proste kinematyki odnosi się do obliczeń pozycji i orientacji członu roboczego manipulatora [4]. W praktyce oznacza zamianę kątów konfiguracyjnych członów obrotowych (θ) na współrzędne kartezjańskie końcówki roboczej (x, y, z). Na Rys. 3.4 przedstawiono schemat projektowanego manipulatora składający się z naniesionych odpowiednio układów współrzędnych.

Koncepcja robota obdarzonego cechami sztucznej inteligencji rozwiązującego układankę z losowo zmieszanyymi puzzlami.



Rys. 3.4 Schemat kinematyczny manipulatora z oznaczeniami według notacji DH (materiał własny w diagrams [25])

Na podstawie schematu (Rys. 3.4) wyznaczono parametry DH zamieszczone w Tab. 1.7. Kąty obrotu wokół osi „z” oznaczone są symbolem θ oraz definiują obrót serwomechanizmu. Zmiany kąta α obserwuje się przy przejściu układu H_1^0 oraz H_4^3 , co można wyobrazić sobie poprzez obrót wokół osi „x”. Kierunki +/- wyznacza się przy zastosowaniu zasady prawej dłoni, układając kciuk

w stronę osi wokół której występuje obrót. Parametry r i d wyznaczono zgodnie z przesunięciami po osiach „z” i „x” wstawiając odległości a w odpowiednie miejsca w tabeli.

Tab. 1.7. Tabela parametrów DH (materiał własny)

	θ	a	r	d
1	θ_1	90	–	a_1
2	θ_2	–	a_2	–
3	θ_3	–	a_3	–
4	θ_4	-90	a_4	–
5	–	–	–	a_5

Koncepcja robota obdarzonego cechami sztucznej inteligencji rozwiązującego układankę z losowo zmieszanyimi puzzlami.

Mając wypełnioną tabelę DH można wyznaczyć macierze przekształceń dla każdego członu zgodnie z wzorem (1.9):

$$H_n^{n-1} = \begin{bmatrix} \cos \theta_n & -\sin \theta_n \cos \alpha_n & \sin \theta_n \sin \alpha_n & r_n \cos \theta_n \\ \sin \theta_n & \cos \theta_n \cos \alpha_n & -\cos \theta_n \sin \alpha_n & r_n \sin \theta_n \\ 0 & \sin \alpha_n & \cos \alpha_n & d_n \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (1.11)$$

Powstałe macierze (1.10) należy podstawić do równania (1.11), a następnie z macierzy H_5^0 odczytać ostatnią kolumnę odpowiadającą położeniu ostatniego układu współrzędnych p_5^0 .

$$\begin{aligned} H_1^0 &= \begin{bmatrix} C\theta_1 & 0 & S\theta_1 & 0 \\ S\theta_1 & 0 & -C\theta_1 & 0 \\ 0 & 1 & 0 & a_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} H_2^1 = \begin{bmatrix} C\theta_2 & -S\theta_2 & 0 & a_2 C\theta_2 \\ S\theta_2 & C\theta_2 & 0 & a_2 S\theta_2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \\ H_3^2 &= \begin{bmatrix} C\theta_3 & -S\theta_3 & 0 & a_3 C\theta_3 \\ S\theta_3 & C\theta_3 & 0 & a_3 S\theta_3 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} H_4^3 = \begin{bmatrix} C\theta_4 & -S\theta_4 & 0 & a_4 C\theta_4 \\ S\theta_4 & C\theta_4 & 0 & a_4 S\theta_4 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \\ H_5^4 &= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & a_5 \\ 0 & 0 & 0 & 1 \end{bmatrix} \end{aligned} \quad (1.12)$$

$$H_5^0 = H_1^0 H_2^1 H_3^2 H_4^3 H_5^4 \quad (1.13)$$

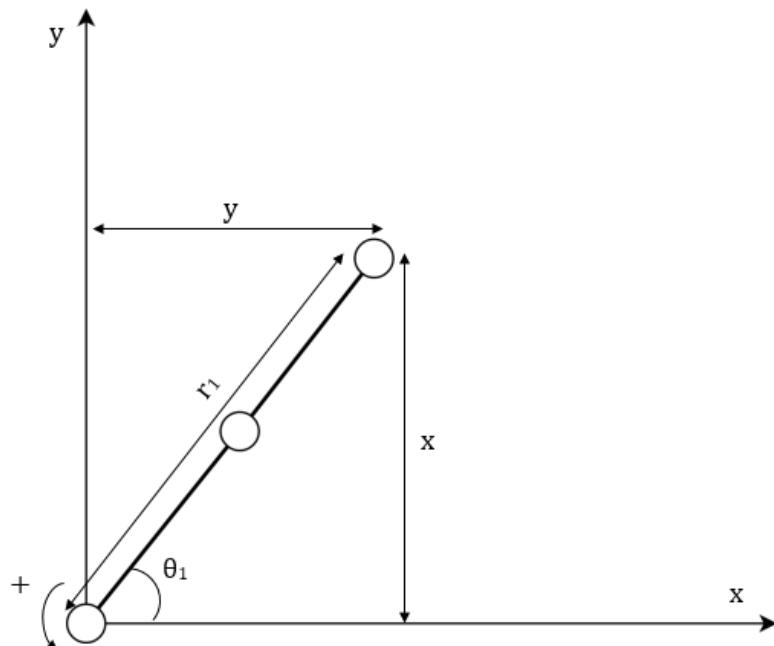
• Zadanie odwrotne kinematyki

Zadanie odwrotne kinematyki polega na wyznaczeniu kątów konfiguracyjnych poszczególnych członów, na podstawie współrzędnych kartezjańskich końcówki roboczej. Ze względu na nieliniowość równań kinematycznych, zadanie odwrotne jest trudniejsze do rozwiązania [4].

Koncepcja robota obdarzonego cechami sztucznej inteligencji rozwiązującego układankę
z losowo zmieszanyymi puzzlami.

Daną konfigurację można rozwiązać przy pomocy metody graficznej, znajdując zależności pomiędzy kątami. Kierując się podejściem, że pierwsze trzy człony stanowią pozycjonowanie końcówki, równania kątów obrotu w oparciu o docelowe współrzędne zostaną wyprowadzone tylko dla nich.

Na Rys. 3.5 przedstawiono widok z góry manipulatora, gdzie wartości x i y oznaczają pozycję docelową członu trzeciego:

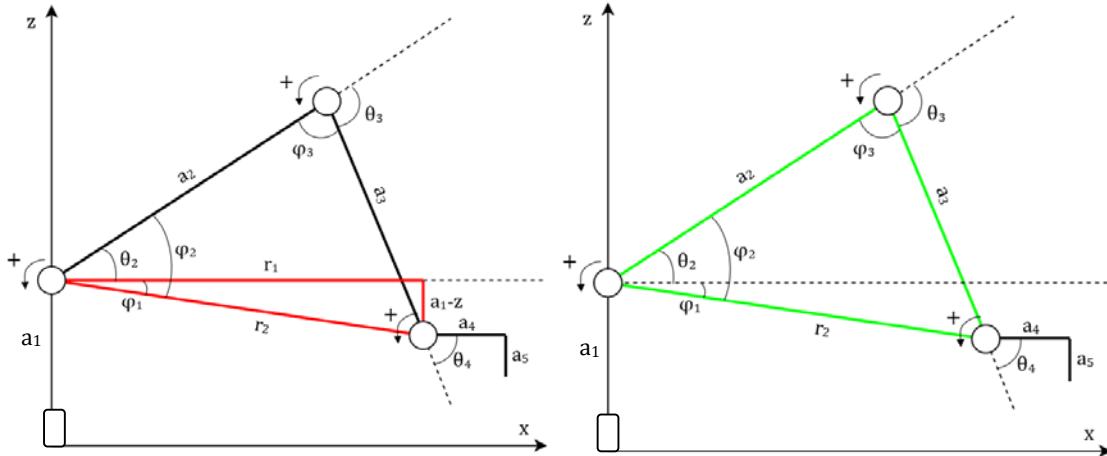


Rys. 3.5 Schemat widoku z góry manipulatora (materiał własny)

Na podstawie pierwszego schematu można obliczyć długość ramienia z perspektywy bazowego układu współrzędnych, która zmienia się wraz z promieniem zataczającego półkola oraz zależność kąta θ_1 (1.14):

$$\begin{aligned} r_1 &= \sqrt{x^2 + y^2} \\ \theta_1 &= \tan^{-1} \left(\frac{y}{x} \right) \end{aligned} \tag{1.14}$$

Koncepcja robota obdarzonego cechami sztucznej inteligencji rozwiązującego układankę z losowo zmieszanyymi puzzlami.



Rys. 3.6 Schemat widoku z boku manipulatora (materiał własny)

Na Rys. 3.6 ukazano schemat widoku z boku manipulatora z zależnościami wyprowadzonymi na podstawie dwóch trójkątów. Najpierw określono wzór na długość boku r_2 oraz kąt φ_1 , a następnie znaleziono kąty φ_2 , θ_2 , oraz θ_3 :

$$r_2 = \sqrt{(a_1 - z)^2 + r_1^2}$$

$$\varphi_1 = \tan^{-1} \left(\frac{a_1 - z}{r_1} \right)$$

$$\varphi_2 = \cos^{-1} \left(\frac{a_2 + r_2 - a_3}{2a_2 r_2} \right) \quad (1.15)$$

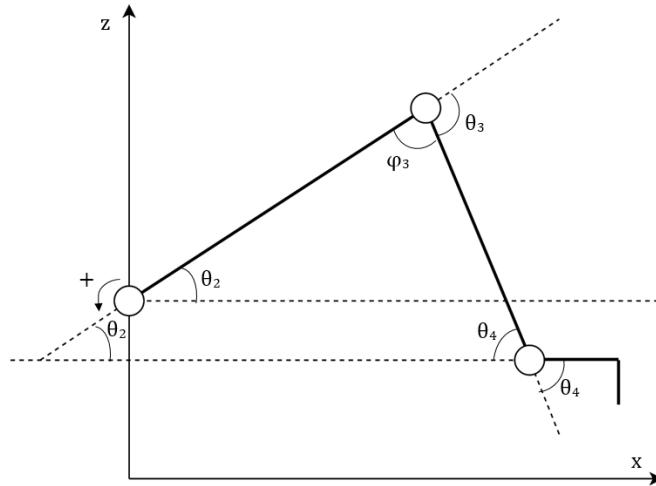
$$\theta_2 = \varphi_2 - \varphi_1$$

$$\theta_3 = 180^\circ - \varphi_3$$

Określenie orientacji końcówki nie wymaga tworzenia kolejnej macierzy ze względu na to, że jej położenie będzie zawsze prostopadłe do układanki. Wartość kąta θ_4 będzie mogła być obliczona na podstawie równania (1.16), gdzie na Rys. 3.7 pokazano te zależności.

$$\theta_4 = 180^\circ - \varphi_3 - \theta_2 \quad (1.16)$$

Koncepcja robota obdarzonego cechami sztucznej inteligencji rozwiązującego układankę z losowo zmieszanyymi puzzlami.



Rys. 3.7 Schemat widoku z boku manipulatora – zależność kąta θ_4 (materiał własny)

W programie obsługującym kinematykę zostaną ujęte również dodatkowe przypadki związane z przechodzeniem pierwszego członu przez oś y_0 , oraz kiedy kąt członu drugiego ma zerową/ujemną wartość.

- **Ruch liniowy**

Przesuwanie elementów układanki wymaga zastosowania ruchów liniowych co oznacza, że każde serwo musi obracać się z określoną prędkością. Prędkość w rozumieniu tego urządzenia została określona jako przyrost stopni obrotu danego członu, ponieważ aby móc sterować tempem obrotu serwonapędów, należy je opóźnić tworząc w pętli przyrost wypełnienia. W celu utworzenia stałej prędkości wzduł konkretnej osi można wykorzystać uproszczoną macierz jakobianową (dla prędkości liniowych) jak w zależności (1.17):

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{bmatrix} = \begin{bmatrix} j_{11} & j_{12} & j_{13} \\ j_{21} & j_{22} & j_{23} \\ j_{31} & j_{32} & j_{33} \end{bmatrix} \begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \\ \dot{\theta}_3 \end{bmatrix} \quad (1.17)$$

$$\begin{aligned} \dot{x} &= j_{11}\dot{\theta}_1 + j_{12}\dot{\theta}_2 + j_{13}\dot{\theta}_3 \\ \dot{y} &= j_{21}\dot{\theta}_1 + j_{22}\dot{\theta}_2 + j_{23}\dot{\theta}_3 \\ \dot{z} &= j_{31}\dot{\theta}_1 + j_{32}\dot{\theta}_2 + j_{33}\dot{\theta}_3 \end{aligned}$$

Koncepcja robota obdarzonego cechami sztucznej inteligencji rozwiązującego układankę z losowo zmieszanymi puzzlami.

Elementy j macierzy uzupełniono zgodnie z wzorem dla połączeń obrotowych w równaniu (1.18), gdzie i oznacza numer rozpatrywanego członu, a n całkowitą liczbę członów:

$$\begin{aligned} R_{i-1}^0 \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \times (p_n^0 - p_{i-1}^0) \\ \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{bmatrix} = \begin{bmatrix} R_0^0 \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \times (p_3^0 - p_0^0) & R_1^0 \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \times (p_3^0 - p_1^0) & R_2^0 \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \times (p_3^0 - p_2^0) \end{bmatrix} \begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \\ \dot{\theta}_3 \end{bmatrix} \quad (1.18) \end{aligned}$$

Następnie należy odwrócić równanie, żeby móc wyznaczyć wartość przyrostu dla każdego z serwonapędów, w celu uzyskania ruchu liniowego po jednej z osi.

$$\begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \\ \dot{\theta}_3 \end{bmatrix} = J^{-1} \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{bmatrix}$$

Koncepcja robota obdarzonego cechami sztucznej inteligencji rozwiązującego układankę z losowo zmieszanyymi puzzlami.

3.2 Schemat połączeń części elektronicznej

Sterownik PCA9685 komunikuje się z Raspberry Pi za pomocą szyny I²C, która wykorzystuje dwie linie sterujące: SDA- dane oraz SCL- zegar (Rys. 3.8). Część logiczną należy zasilić poprzez 3,3V (pin VCC), a serwomechanizmom należy zapewnić adekwatne w stosunku do ich poboru prądu dodatkowe zasilanie 4.8-7.2V (pin V+). Pobór prądu podczas normalnej pracy oraz zasilaniu 6V serwonapędu TowerPro wynosi ~500mA, a w przypadku swojego maksymalnego momentu obrotowego może wzrosnąć do 2,5A (na przykład podczas blokady). W związku z tym, że ramię manipulatora nie będzie narażone na wysokie obciążenia dobrano zasilacz 5V/3A.

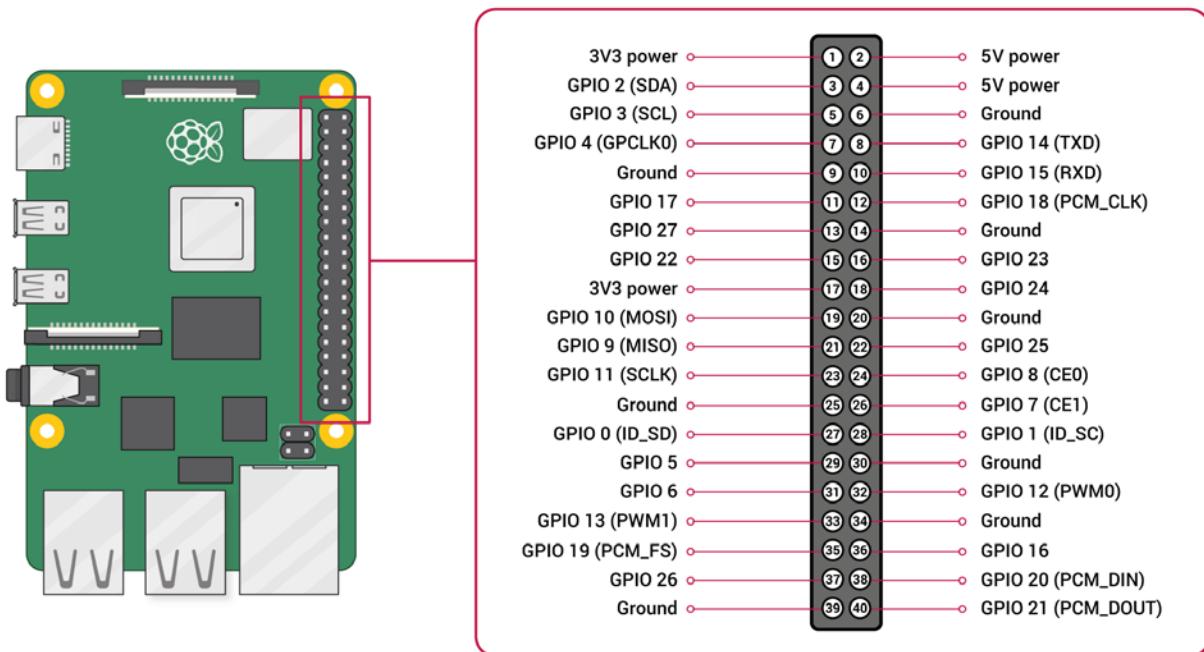


Rys. 3.8 Transmisja danych I²C [26]

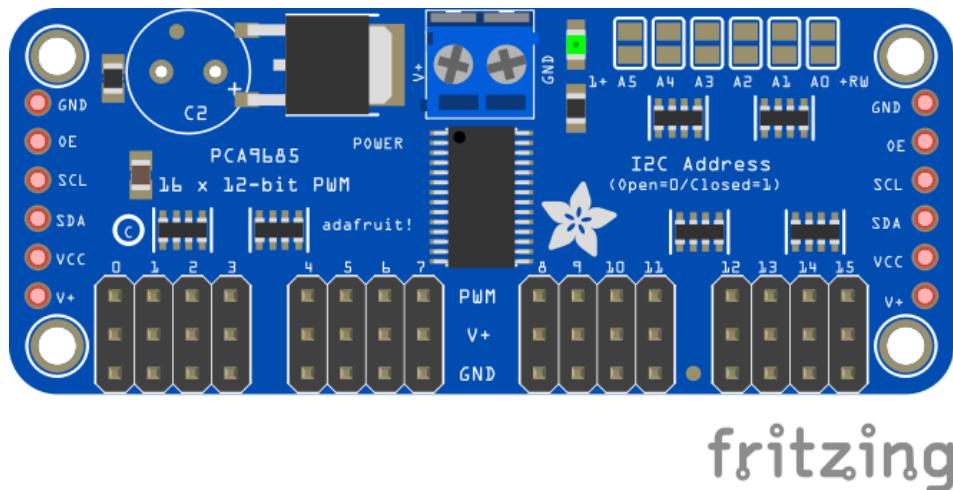
Na Rys. 3.11 przedstawiono sposób połączenia obu urządzeń wraz z wentylatorem, który będzie przymocowany w obudowie oraz jednym serwomechanizmem. Docelowo będą wykorzystane cztery kanały PWM o numerach 0-3, co na schemacie zostało pominięte w celu zachowania jego czytelności. Każdy serwomechanizm posiada wyprowadzenia V+, GND oraz sterujące PWM.

Rys. 3.9 obrazuje główne zastosowanie pinów GPIO w Raspberry Pi, według którego dobrano połączenia, a Rys. 3.10 wyprowadzenia w sterowniku Adafruit.

Koncepcja robota obdarzonego cechami sztucznej inteligencji rozwiązującego układankę z losowo zmieszanyymi puzzlami.

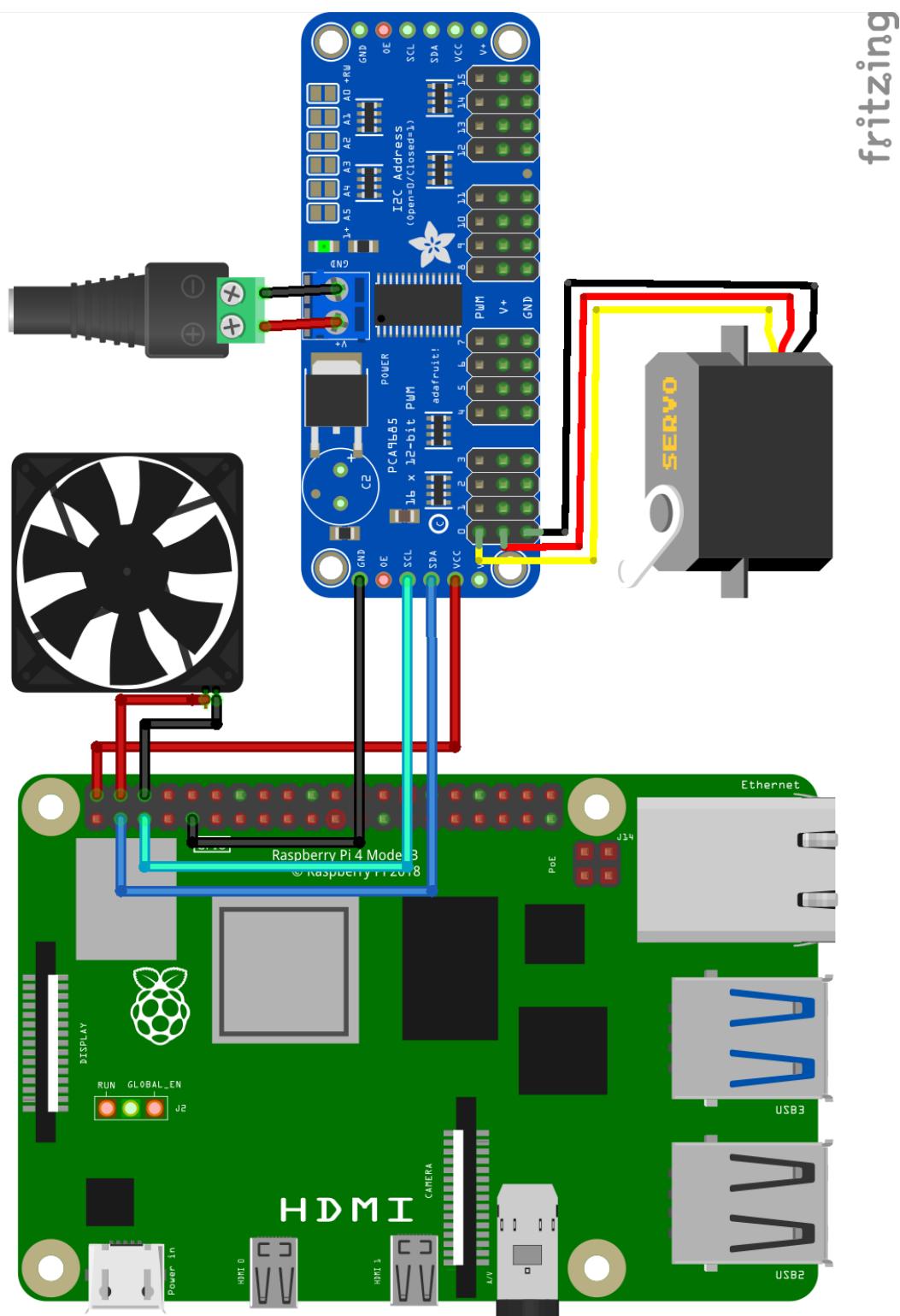


Rys. 3.9 Zastosowanie pinów GPIO w Raspberry Pi [27]



Rys. 3.10 Wyprowadzenia pinów w sterowniku Adafruit (materiał własny w Fritzing [28])

Koncepcja robota obdarzonego cechami sztucznej inteligencji rozwiązującego układankę z losowo zmieszanyimi puzzlami.



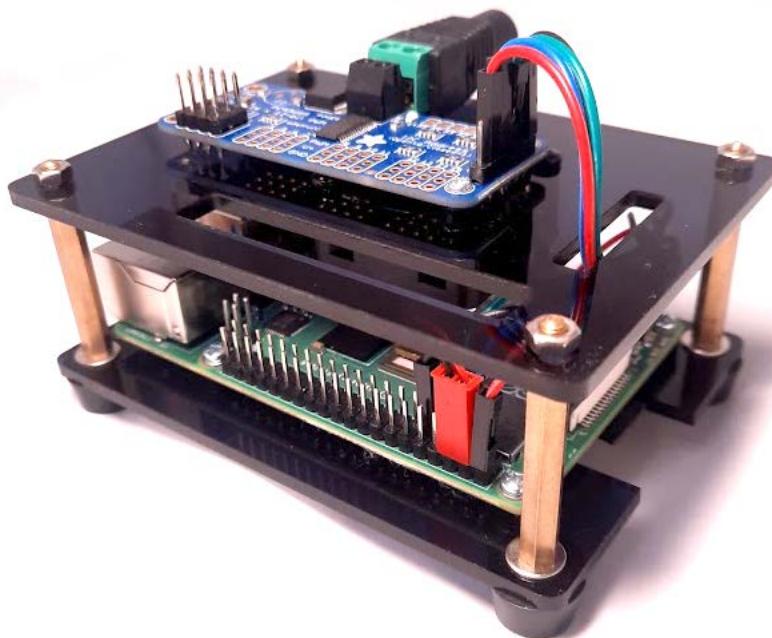
Rys. 3.11 Sposób połączenia elementów części elektronicznej na przykładzie jednego serwomechanizmu oraz wentylatora (materiał własny w fritzing [28])

Koncepcja robota obdarzonego cechami sztucznej inteligencji rozwiązującego układankę z losowo zmieszanyimi puzzlami.

4 WYKONANIE ROBOTA INTELIGENTNEGO

W tym rozdziale opisano kroki, jakie należy podjąć w celu złożenia oraz zaprogramowania konstrukcji. W pierwszej kolejności należy zainstalować środowisko do pracy, a następnie rozpocząć montaż ramienia oraz kamery. Serwonapędy posiadają zakres obrotu $\sim 190^\circ$, więc podczas ich ustawiania należy zweryfikować maksymalne oraz minimalne wychylenie przy pomocy programu testowego oraz zamontować tak, aby były zgodne ze schematem kinematycznym.

Na Rys. 4.1 przedstawiono złożony sterownik robota w obudowie chłodzącej, na którym można rozpoczęć instalację systemu oraz komponentów potrzebnych do jego zaprogramowania.



Rys. 4.1 Raspberry Pi w obudowie wraz z podłączonym sterownikiem serwonapędów
(materiał własny)

Koncepcja robota obdarzonego cechami sztucznej inteligencji rozwiązującego układankę z losowo zmieszanyimi puzzlami.

4.1 Przygotowanie środowiska Raspberry Pi

- **Instalacja systemu Raspberry Pi OS, wersja 64-bitowa**

System operacyjny oraz oprogramowanie potrzebne do jego zaimplementowania na kartę microSD należy pobrać z oficjalnej strony Raspberry Pi [29]. Za pomocą programu Raspberry Pi Imager (Rys. 4.2) należy wybrać pobrany system oraz dysk, a następnie kliknąć przycisk „Write”.

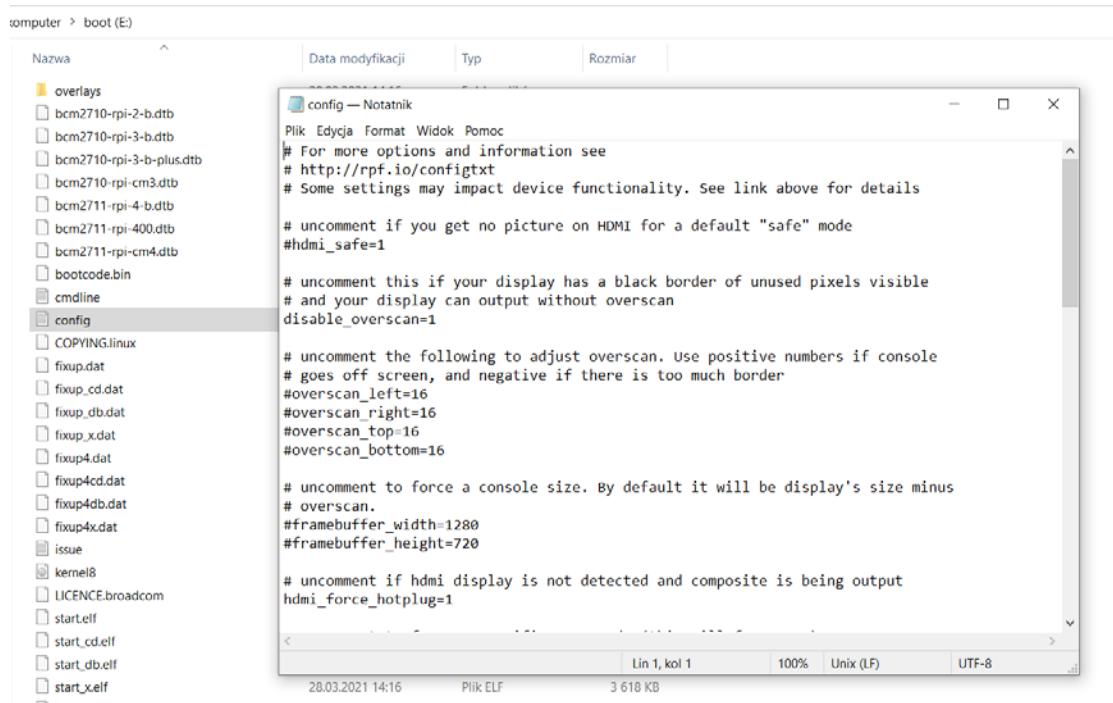


Rys. 4.2 Oprogramowanie Raspberry Pi Imager

Zanim rozpocznie się pracę w systemie, należy go odpowiednio skonfigurować poprzez modyfikację pliku „config.txt”, znajdujący się w partycji „boot” (Rys. 4.3). Raspberry Pi OS jest oparty o architekturę Linux oraz jest podzielony na dwie partycje. Partycja „boot” jest sformatowana przy pomocy systemu plików FAT, dzięki czemu ma się do niej dostęp również z poziomu systemu Windows. Zawiera ona wszelkie pliki wymagane do uruchomienia systemu.

W pliku „config.txt” znajdują się między innymi parametry potrzebne do właściwego wyświetlenia obrazu na danym monitorze, jednakże systemem Raspberry Pi można zarządzać również zdalnie z innego komputera poprzez połączenie SSH.

Koncepcja robota obdarzonego cechami sztucznej inteligencji rozwiązującego układankę z losowo zmieszonymi puzzlami.



Rys. 4.3 Widok na zawartość partycji „boot” wraz z plikiem „config.txt”

Do konfiguracji monitora o rozdzielczości 1080p, częstotliwości 60Hz oraz proporcji ekranu 16:9 poprzez połączenie HDMI użyto następujących komend :

```
hdmi_group=1  
hdmi_mode=16  
hdmi_force_hotplug=1  
hdmi_safe=1
```

Listing 1.1 Konfiguracja monitora w pliku „config.txt” (materiał własny)

Pozycje `hdmi_group` oraz `hdmi_mode` dotyczą parametrów monitora na którym ma wyświetlać się system, natomiast `hdmi_force_hotplug` oraz `hdmi_safe` razem zapewniają maksymalną kompatybilność połączenia HDMI. Dokładny opis konfiguracji wyświetlania znajduje się na stronie producenta [30].

Po włożeniu karty microSD do Raspberry Pi i jego uruchomieniu należy sprawdzić zainstalowaną wersję systemu oraz kompilatora C++, który również musi być 64-bitowy. Można to wykonać poprzez polecenia: `uname -a` oraz `gcc -v`.

Koncepcja robota obdarzonego cechami sztucznej inteligencji rozwiązującego układankę z losowo zmieszanyimi puzzlami.

Komunikację SSH należy włączyć za pomocą komendy `raspi-config`, a następnie w terminalu użytkowanego komputera wpisać polecenie `ssh pi@<adres IP urządzenia>`. Na Rys. 4.4 przedstawiono poprawne połaczenie się z systemem Raspberry Pi.

```
pi@raspberrypi: ~
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

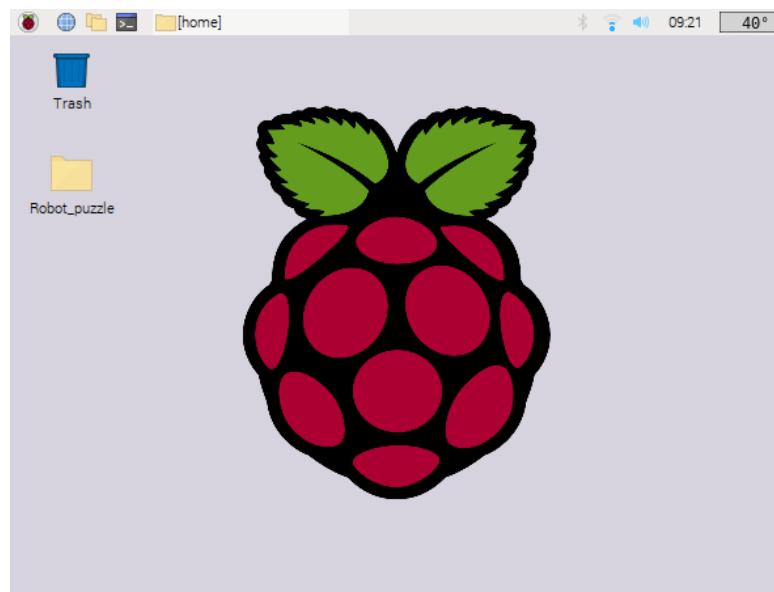
PS C:\Users\falma> ssh pi@192.168.0.120
pi@192.168.0.120's password:
Linux raspberrypi 5.10.17-v8+ #1403 SMP PREEMPT Mon Feb 22 11:37:54 GMT 2021 aarch64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Sun May 23 09:27:48 2021 from 192.168.0.108
pi@raspberrypi:~ $
```

Rys. 4.4 Połaczenie SSH - widok konsoli PowerShell (materiał własny)

W przypadku poprawnej konfiguracji monitora oraz połączeniu klawiatury i myszki można korzystać z wersji desktopowej systemu jak na Rys. 4.5.



Rys. 4.5 Widok na pulpit systemu Raspberry Pi (materiał własny)

Koncepcja robota obdarzonego cechami sztucznej inteligencji rozwiązującego układankę z losowo zmieszonymi puzzlami.

- **Partycja wymiany (SWAP)**

Kolejnym krokiem jest zwiększenie miejsca w pamięci, która może zostać użyta jako tymczasowa pamięć RAM. Jest to wymagane do prawidłowej instalacji bibliotek OpenCV i Tensorflow oraz dotyczy wersji Raspberry Pi z 2 i 4GB RAM. Partycja SWAP może przechować część danych w przypadku gdy ich ilość przekracza dostępne zasoby. Standardowy zapis na karcie SD jest procesem wolnym oraz może szybko doprowadzić do jej zużycia ze względu na limit jednocześnie operacji zapisu [31]. Aby temu zapobiec skorzystano z modułu zram, który kompresuje dane do pliku .zip i zapisuje je z powrotem do pamięci RAM. Proces instalacji modułu został przedstawiony na Listing 1.2, natomiast widok na zawartość pliku „rc.local” wraz z dodaną linią na Listing 1.3.

```
# usuniecie poprzedniej wersji dphys
$ sudo /etc/init.d/dphys-swapfile stop
$ sudo apt-get remove --purge dphys-swapfile
# instalacja zram
$ sudo wget -O /usr/bin/zram.sh
https://raw.githubusercontent.com/novaspirit/rpi_zram/ \
master/zram.sh
# ustawienie autoladowania
$ sudo nano /etc/rc.local
# dodanie linii przed exit 0
/usr/bin/zram.sh &
# zapis pliku <Ctrl+X>, <Y> oraz <Enter>
```

Listing 1.2 Instalacja modułu zram [31]

```
#!/bin/sh -e
# rc.local
# Print the IP address
_IP=$(hostname -I) || true
if [ "$_IP" ]; then
    printf "My IP address is %s\n" "$_IP"
fi

/usr/bin/zram.sh &

exit 0
```

Listing 1.3 Zawartość pliku „rc.local” (materiał własny)

Koncepcja robota obdarzonego cechami sztucznej inteligencji rozwiązującego układankę z losowo zmieszonymi puzzlami.

Następnie, należy zwiększyć domyślny rozmiar partycji SWAP oraz uruchomić ponownie urządzenie jak na Listing 1.4. Widok zmodyfikowanego pliku „zram.sh” przedstawiono na Listing 1.5.

```
$ sudo chmod +x /usr/bin/zram.sh
$ sudo nano /usr/bin/zram.sh
# pomnożyć limit * 2
mem=$(( ($totalmem / $cores)* 1024 * 2 ))
# zapis <Ctrl+X>, <Y> oraz <Enter>
$ sudo reboot
```

Listing 1.4 Zwiększenie rozmiaru SWAP [31]

```
#!/bin/bash
cores=$(nproc --all)
modprobe zram num_devices=$cores

swapoff -a

totalmem=`free | grep -e "Mem:" | awk '{print $2}'` 
mem=$(( ($totalmem / $cores)* 1024 * 2 ))

core=0
while [ $core -lt $cores ]; do
    echo $mem > /sys/block/zram$core/disksize
    mkswap /dev/zram$core
    swapon -p 5 /dev/zram$core
    let core=core+1
done
```

Listing 1.5 Zawartość pliku „zram.sh” (materiał własny)

Po restarcie urządzenia za pomocą komendy `free -m` można sprawdzić efekty widoczne na Rys. 4.6. Po wprowadzeniu zmian urządzenie ma do dyspozycji 5,4GB RAM, co jest wystarczające do instalacji OpenCV wersji 4.5 oraz Tensorflow.

```
pi@raspberrypi:~ $ free -m
              total        used        free      shared  buff/cache   available
Mem:       1798         193       1068          51        537       1486
Swap:      3596           0       3596
```

Rys. 4.6 Dostępna pamięć RAM (Mem) oraz SWAP (materiał własny)

Koncepcja robota obdarzonego cechami sztucznej inteligencji rozwiązującego układankę z losowo zmieszanymi puzzlami.

- **Przetaktowywanie procesora (ang. Overclocking CPU)**

Przetaktowywanie procesora nie jest procesem koniecznym, jednak w przypadku uruchamiania modułów sztucznej inteligencji może znacznie przyspieszyć pracę. W zależności od możliwości chłodzenia płytka można przetestować ustawienia `over_voltage` (przepięcie) oraz `arm_freq` (taktowanie). Wraz z ich zwiększeniem rośnie napięcie rdzenia CPU, moc oraz wydajność urządzenia, jednak spada maksymalna bezpieczna temperatura pracy. W Tab. 1.8 przedstawiono możliwe efekty zastosowania przetaktowania CPU, natomiast ustawienia należy dobierać rozważnie ze względu na możliwość awarii urządzenia, a do każdego rozwiązania zapewnić odpowiednie chłodzenie. W niniejszym projekcie zastosowano minimalne zwiększenie taktowania, co daje wzrost wydajności na poziomie 6,6%. Można to zrobić poprzez modyfikację wcześniejszej opisanego pliku „config.txt” (Rys. 4.3) dodając linie (Listing 1.6):

```
over_voltage =1  
arm_freq=1600
```

Listing 1.6 Konfiguracja przetaktowania w pliku „config.txt” (materiał własny)

Tab. 1.8. Możliwe efekty po przetaktowaniu procesora [31]

Zegar (MHz)	Przepięcie	Napięcie rdzenia Vcore	Maks. temp. (°C)	Moc (Wat)	Wzrost wydajności	Uwagi
0	0	0.8625		1.5		RPi 4 wyłączone
600	0	0.8625		2.8		RPi 4 bezczynne
1500	0	0.8625	82	7		Domyślnie
1600	1	0.8875	80	7.6	6.6 %	
1700	2	0.9125	78	8.3	13.3 %	
1800	3	0.9375	77	8.9	20 %	
1900	4	0.9625	75	9.5	26.6 %	
2000	6	1.0125	72	11	33.3 %	
2100	6	1.0125	72	11	40 %	
	7	1.0375	56	11.7		brak poprawy
	8	1.0625	50	12.3		brak poprawy

Koncepcja robota obdarzonego cechami sztucznej inteligencji rozwiązującego układankę z losowo zmieszonymi puzzlami.

- **Instalacja OpenCV 4.5, Tensorflow 2.3.1 oraz Tensorflow Keras 2.4.0**

Przed każdą instalacją należy sprawdzić aktualizacje za pomocą komend `sudo apt-get update` oraz `sudo apt-get upgrade`. Obecnie nie ma gotowych plików instalacyjnych dla systemu 64-bit, dlatego komplikacja OpenCV oraz Tensorflow musi odbyć się ze źródła, co zawarto w załącznikach 8 i 9. Instalacja pakietu Keras natomiast, może już odbyć się za pomocą komendy:

```
$ pip3 install keras
```

Na koniec można sprawdzić poprawność instalacji uruchamiając moduł `python` w wierszu poleceń oraz importując wszystkie pakiety. Wersje programu ukazuje komenda `__version__`, przedstawia to Listing 1.7.

```
pi@raspberrypi:~ $ python
Python 3.7.3 (default, Jan 22 2021, 20:04:44)
[GCC 8.3.0] on linux
Type "help", "copyright", "credits" or "license" for more
information.

>>> import cv2
>>> cv2.__version__
'4.5.0'
>>> import tensorflow as tf
tf.__version__
>>> tf.__version__
'2.3.1'
>>> import tensorflow.keras as keras
>>> keras.__version__
'2.4.0'
```

Listing 1.7 Sprawdzenie poprawności instalacji bibliotek(materiał własny)

- **Konfiguracja sterownika do serw**

Producent elektroniki firmy Waveshare udostępnił na swojej stronie [32] plik konfigurujący moduł PCA9685 do sterowania serwomechanizmami, który został umieszczony w pliku o nazwie „PCA9685.py” (załącznik nr). Znajdują się w nim gotowe funkcje do obsługi sygnału PWM dla każdego z dostępnych kanałów.

Koncepcja robota obdarzonego cechami sztucznej inteligencji rozwiązującego układankę z losowo zmieszanyimi puzzlami.

4.2 Montaż elementów robota oraz transformacja układu współrzędnych

Pierwszym krokiem jaki należy wykonać jest montaż oraz konfiguracja ramienia manipulatora wraz z kamerą. Na Rys. 4.7 przedstawiono sposób umiejscowienia kamery przy pomocy łącznika, rozmontowując uprzednio jej podstawę, oraz przymocowane ramię manipulatora. W dalszych krokach wyjaśniono w jaki sposób powiązać ze sobą oba układy współrzędnych.

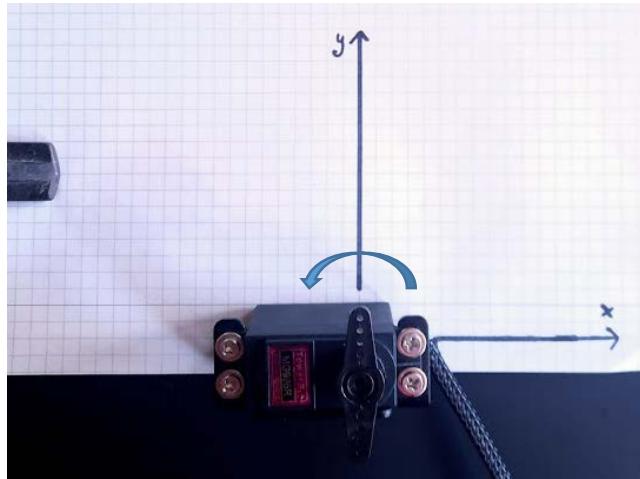


Rys. 4.7 Kamera wraz z ramieniem zamontowane na płycie (materiał własny)

Przygotowania do prawidłowego ustawienia poszczególnych członów manipulatora według schematu kinematycznego należy rozpocząć od rozrysowania osi pomocniczych, oraz ustalenia krańcowych położen pierwszego serwonapędu. Na Rys. 4.8 przedstawiono przytwierdzony pierwszy człon obrotowy manipulatora równolegle do krawędzi płyty.

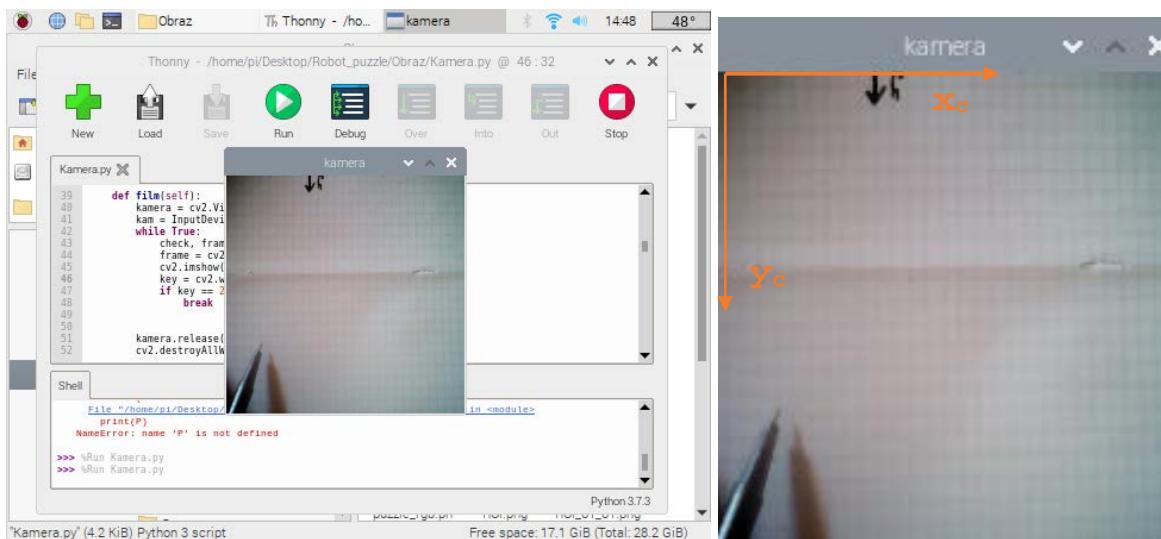
Koncepcja robota obdarzonego cechami sztucznej inteligencji rozwiązującego układankę z losowo zmieszanyymi puzzlami.

Orczyk wskazuje pozycję 90° , a strzałką oznaczono kierunek obrotu serwonapędu. Przy użyciu programu testującego serwomechanizm oraz osi pomocniczych można ustalić wstępne zakresy wypełnienia.



Rys. 4.8 Przymocowanie pierwszego serwonapędu oraz test (materiał własny)

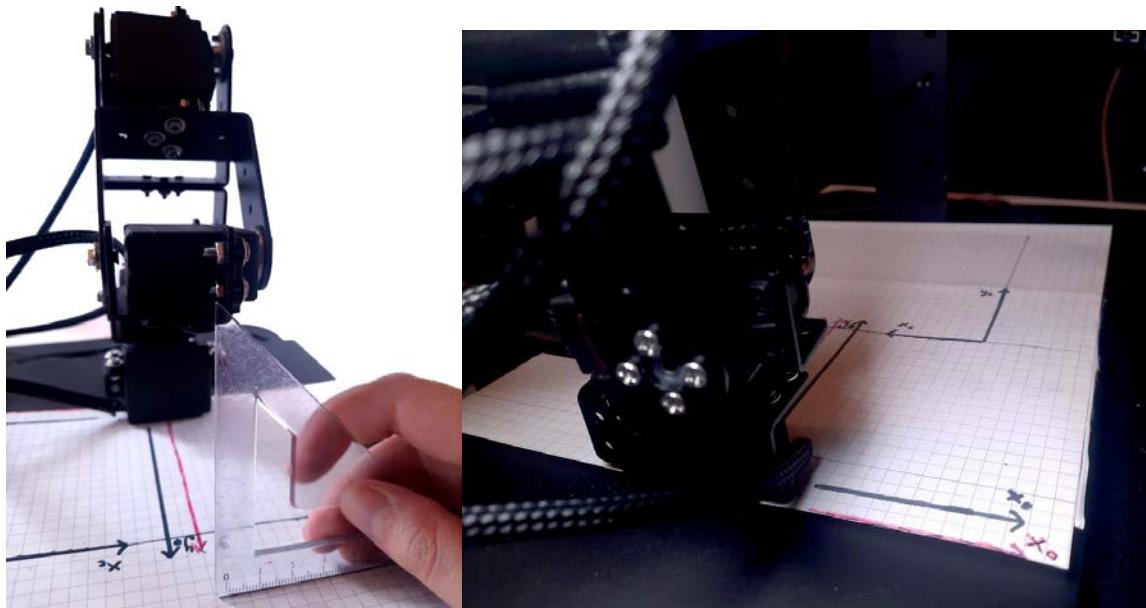
Następnie należy uruchomić program testujący kamerę oraz ustawić ją w żądanej pozycji. Następnie, jednocześnie trzymając marker oraz obserwując obraz należy odnaleźć początek układu współrzędnych kamery i oznaczyć go na kartce. Na Rys. 4.9 pokazano fragment uruchomionego programu z pomniejszonym oknem podglądu. Układ współrzędnych kamery, co wynika z właściwości obrazu, rozpoczyna się w lewym górnym rogu.



Rys. 4.9 Widok na uruchomiony program testowy kamery (materiał własny)

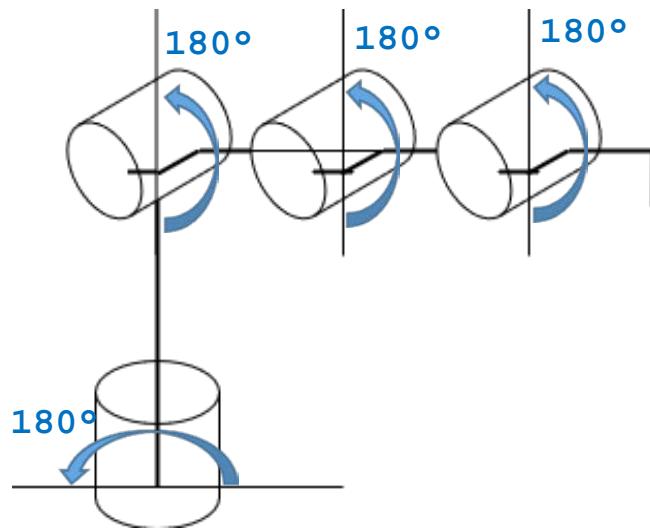
Koncepcja robota obdarzonego cechami sztucznej inteligencji rozwiązującego układankę z losowo zmieszanyymi puzzlami.

Po oznaczeniu osi współrzędnych oraz orientacyjnego zakresu widoczności kamery została przymocowana pozostała część ramienia. Kolorem różowym oznaczono ostateczne położenie układu bazowego, co jest widoczne na Rys. 4.10.



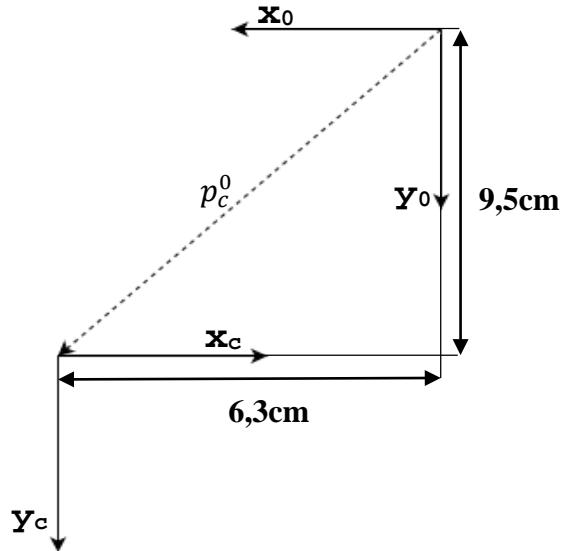
Rys. 4.10 Rysunek układów współrzędnych (materiał własny)

Każdy z pozostałych serwonapędów został zamocowany w podobny sposób co pierwszy, ustalając graniczne wartości wypełnienia. Na Rys. 4.11 przedstawiono zakres każdego z nich.



Rys. 4.11 Zakresy przymocowanych serwonapędów (materiał własny)

Koncepcja robota obdarzonego cechami sztucznej inteligencji rozwiązującego układankę z losowo zmieszanyymi puzzlami.



Rys. 4.12 Odległości układów współrzędnych (materiał własny)

Powiązanie ze sobą dwóch układów współrzędnych opisano na podstawie Rys. 4.12, gdzie wektor p_c^0 obrazuje przesunięcie z układu bazowego (x_0, y_0) do układu kamery (x_c, y_c) . Macierz jednorodna H_c^0 opisująca zarówno obrót jak i przesunięcie przyjmie następującą postać (1.19):

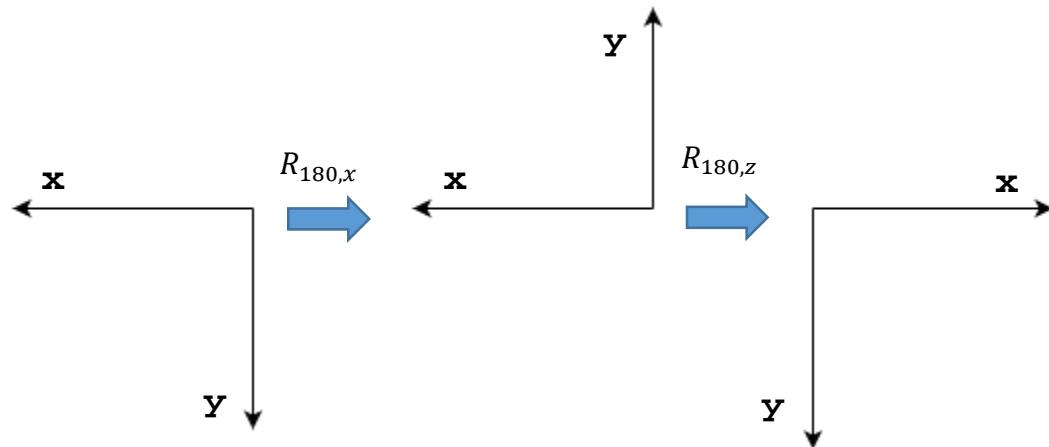
$$H_c^0 = \begin{bmatrix} R_c^0 & | & p_c^0 \\ - & - & - \\ 0 & 0 & 0 & | & 1 \end{bmatrix}$$

$$R_c^0 = R_{180,x} R_{180,z} \quad (1.19)$$

$$p_c^0 = \begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix} = \begin{bmatrix} 6,3 \text{ cm} \\ 9,5 \text{ cm} \\ 0 \text{ cm} \end{bmatrix}$$

Opis przejścia R_c^0 wymaga dwóch operacji obrotu. Zgodnie z regułą prawej dłoni do prawoskrętnego układu współrzędnych, widać że os „z” układu kamery jest zwrocona w przeciwną stronę, co układu bazowego. W związku z tym wymagany jest obrót wokół osi „x” o 180° . Następnie, aby osie „x” oraz „y” pokrywały się ze sobą, należy obrócić układ wokół osi „z”, również o 180° . Na Rys. 4.13 zwizualizowano ten proces.

Koncepcja robota obdarzonego cechami sztucznej inteligencji rozwiązującego układankę z losowo zmieszanyymi puzzlami.



Rys. 4.13 Wizualizacja obrotów układu współrzędnych (materiał własny)

Zależność (1.13) przedstawia macierze obrotu według obu osi, a po ich wymnożeniu otrzymuje się macierz R_c^0 , zgodnie z zależnością (1.12).

$$R_{180,x} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & \cos(180^\circ) & -\sin(180^\circ) \\ 0 & \sin(180^\circ) & \cos(180^\circ) \end{bmatrix} \quad R_{180,x} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \end{bmatrix}$$

$$R_{180,z} = \begin{bmatrix} \cos(180^\circ) & -\sin(180^\circ) & 0 \\ \sin(180^\circ) & \cos(180^\circ) & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad R_{180,z} = \begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$
(1.20)

Dzięki powyższemu opisowi, korzystając z równania (1.14) można odczytać współrzędne (x_0, y_0) , na podstawie współrzędnych (x_c, y_c) kamery. Współrzędna „ z_c ” nie jest brana pod uwagę, więc jej wartość wynosi 0.

$$\begin{bmatrix} x_0 \\ y_0 \\ z_0 \\ 1 \end{bmatrix} = H_c^0 \begin{bmatrix} x_c \\ y_c \\ z_c \\ 1 \end{bmatrix}$$
(1.21)

Koncepcja robota obdarzonego cechami sztucznej inteligencji rozwiązującego układankę z losowo zmieszanyimi puzzlami.

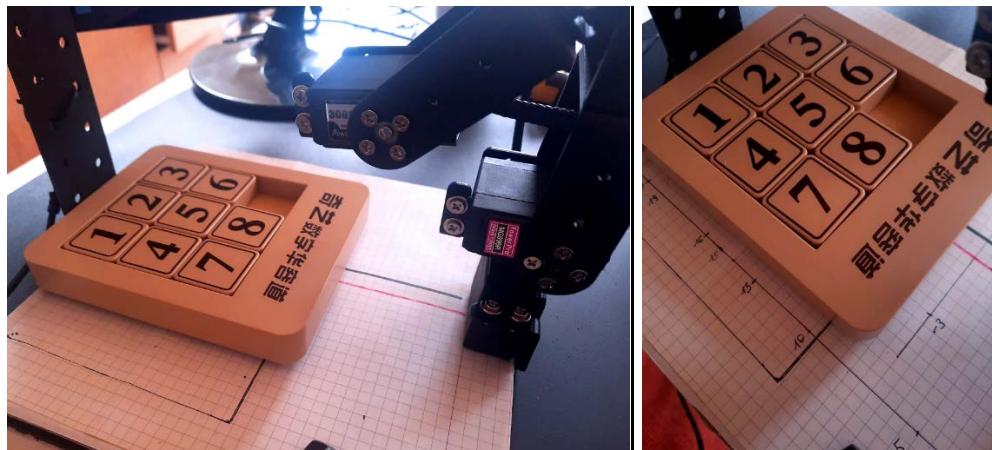
- **Sprawdzenie poprawności działania**

Za pomocą funkcji `Kamera().szukaj` zawartej w pliku Kamera.py można zweryfikować poprawność wykonanych działań. Dodając komendy `print()` oraz odwołując się do każdej z pozycji wygenerowano następujące wartości (Listing 1.8):

```
Pozycje puzzli:  
(-3.62, 15.59, 8)  
(-0.23, 15.59, 8)  
(3.16, 15.59, 8)  
(-3.62, 12.24, 8)  
(-0.23, 12.24, 8)  
(3.16, 12.24, 8)  
(-3.62, 8.89, 8)  
(-0.23, 8.89, 8)  
(3.16, 8.89, 8)
```

Listing 1.8 Wyniki działania programu (materiał własny)

Porównując widoczną na Rys. 4.14 lokalizację układanki z uzyskanymi wynikami programu widać, że jest on poprawny. Należy zaznaczyć, że pozycja „y” jest wyznaczona dla 4 członu manipulatora, a nie końcówki roboczej. Po dodaniu 3.4cm do każdej pozycji „y” końcówka znajdzie się w położeniach: 18.99, 15.64 oraz 12.29cm.



Rys. 4.14 Sprawdzenie wygenerowanych lokalizacji (materiał własny)

Koncepcja robota obdarzonego cechami sztucznej inteligencji rozwiązującego układankę z losowo zmieszanyimi puzzlami.

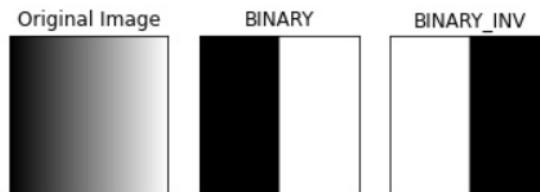
4.3 Analiza obrazu z kamery i rozpoznanie konfiguracji

Analiza obrazu odbywa się dwuetapowo:

- 1) Znalezienie układanki na zdjęciu oraz określenie jej lokalizacji
- 2) Rozpoznanie aktualnej konfiguracji cyfr

Biblioteka OpenCV dostarcza zaawansowanych narzędzi do obróbki obrazu. Najważniejszymi funkcjami oraz ich parametrami użytymi w tym procesie są [33]:

- `cv2.VideoCapture()` otwiera plik wideo lub przechwytuje go z urządzenia nadającego,
- `cv2.imwrite()` oraz `cv2.imread()` służą do odczytu oraz zapisu obrazu,
- `cv2.cvtColor()` zmienia przestrzeń kolorów obrazu na podany w argumencie, `cv2.COLOR_RGB2GRAY` dokonuje konwersji z typu RGB na skalę szarości,
- `cv2.adaptiveThreshold()` umożliwia progowanie obrazu według podanych w argumentów algorytmów, z których zastosowano metodę `cv2.BINARY_INV` (Rys. 4.15) oraz algorytm `cv2.ADAPTIVE_THRESH_GAUSSIAN_C`, który uśrednia wartości sąsiednich pikseli z możliwością dostosowania parametrów,

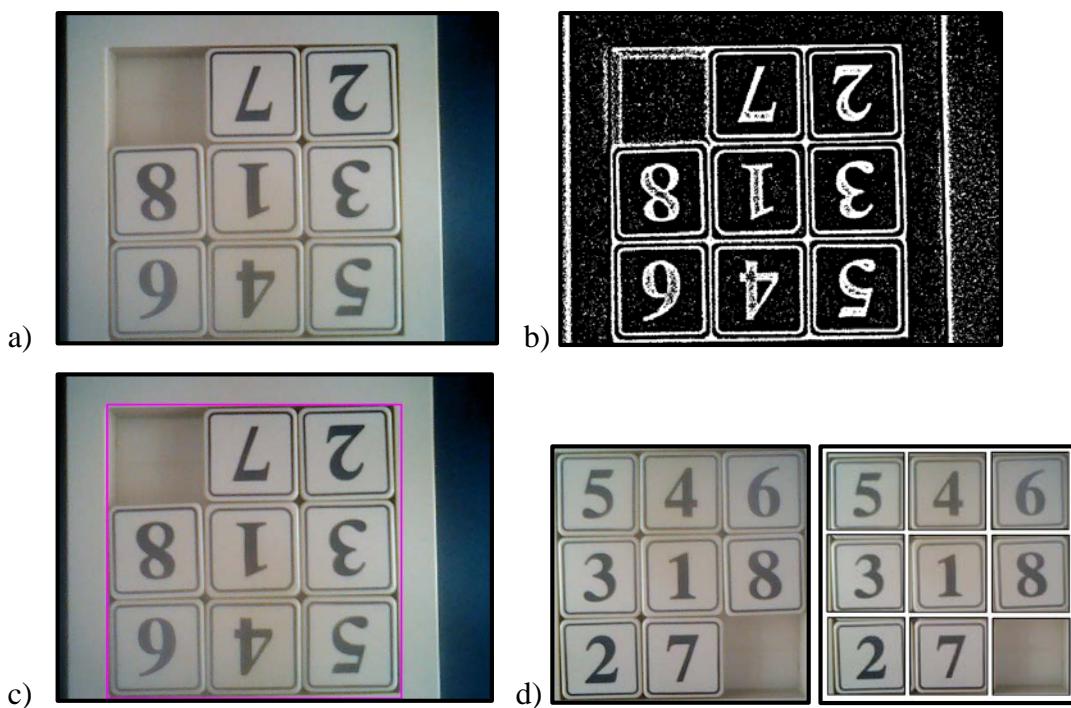


Rys. 4.15 Efekt progowania obrazu BINARY oraz BINARY_INV [33]

- `cv2.findContours()` znajduje kontury w obrazie binarnym, `cv2.RETR_EXTERNAL` znajduje jedynie najbardziej zewnętrzne kontury, a `cv2.CHAIN_APPROX_SIMPLE` odpowiada za kodowanie konturu za pomocą 4 punktów,
- `cv2.boundingRect()` odpowiada za pobranie lokalizacji znalezionej kontury, a `cv2.rectangle()` go rysuje,
- `cv2.rotate()` obraca dany obraz.

Koncepcja robota obdarzonego cechami sztucznej inteligencji rozwiązującego układankę z losowo zmieszanyimi puzzlami.

Na Rys. 4.16 przedstawiono obrazy z kamery po każdym etapie obróbki. Rys. 4.16b ukazuje efekt odwróconego progowania obrazu, który zastosowany przed szukaniem konturu zwiększa jego szanse powodzenia. Przed wyrysowaniem konturu na oryginalnym zdjęciu dodano warunek ograniczający na jego wysokość i szerokość tak, aby nie generować niepożądanych ramek. Obszar ROI (z ang. obszar zainteresowań) powstaje z zapisania tej części obrazu, którą definiuje kontur oraz jego obrócenia do dalszych operacji.

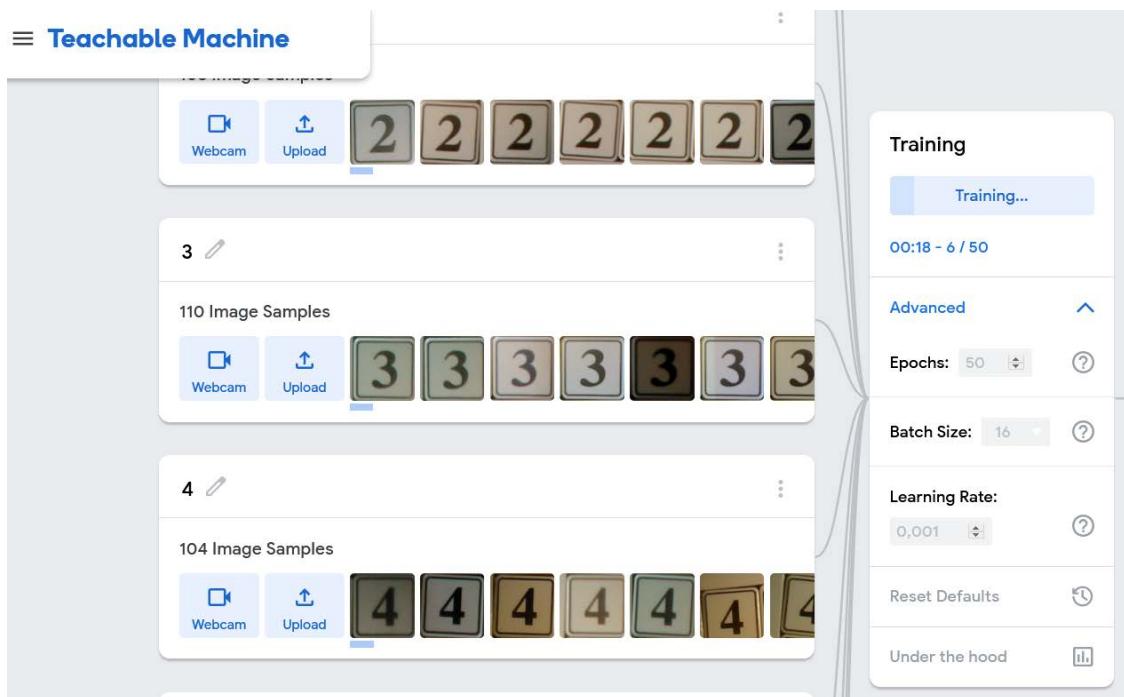


Rys. 4.16 Etapy obróbki zdjęcia: a) wykonanie b) progowanie c) nałożenie znalezionej ramki na pierwotne zdjęcie d) przycięcie do obszaru kluczowego ROI oraz podział na fragmenty (materiał własny)

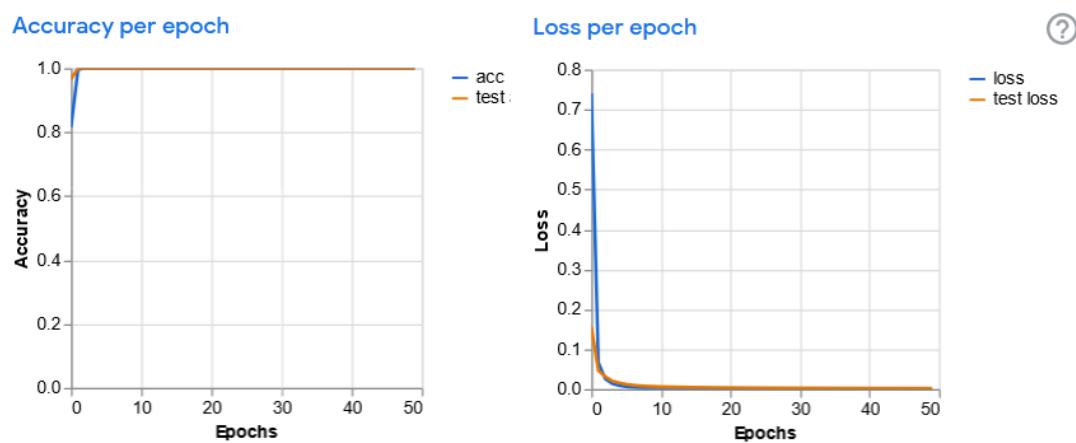
Przygotowanie zdjęcia do rozpoznawania cyfr w module Tensorflow wymaga aby były one w wymiarze kwadratu. Proces wytrenowania modelu rozpoczyna się od wykonania zdjęć w różnym oświetleniu oraz nieidealnej orientacji cyfr, co widoczne jest na Rys. 4.17. Materiał do wytrenowania zgromadzono z podzielonych zdjęć układanki jak na Rys. 4.16d.

Koncepcja robota obdarzonego cechami sztucznej inteligencji rozwiązującego układankę z losowo zmieszanyimi puzzlami.

Za pomocą narzędzia Teachable Machine (Rys. 4.17) utworzono model, w którym umieszczone zostały zdjęcia poszczególnych cyfr oraz nadano im identyfikatory. Na Rys. 4.18 przedstawiono wyniki treningu modelu za pomocą domyślnej konfiguracji, świadczące o jego poprawności.



Rys. 4.17 Trenowanie modelu za pomocą Teachable machine [34]

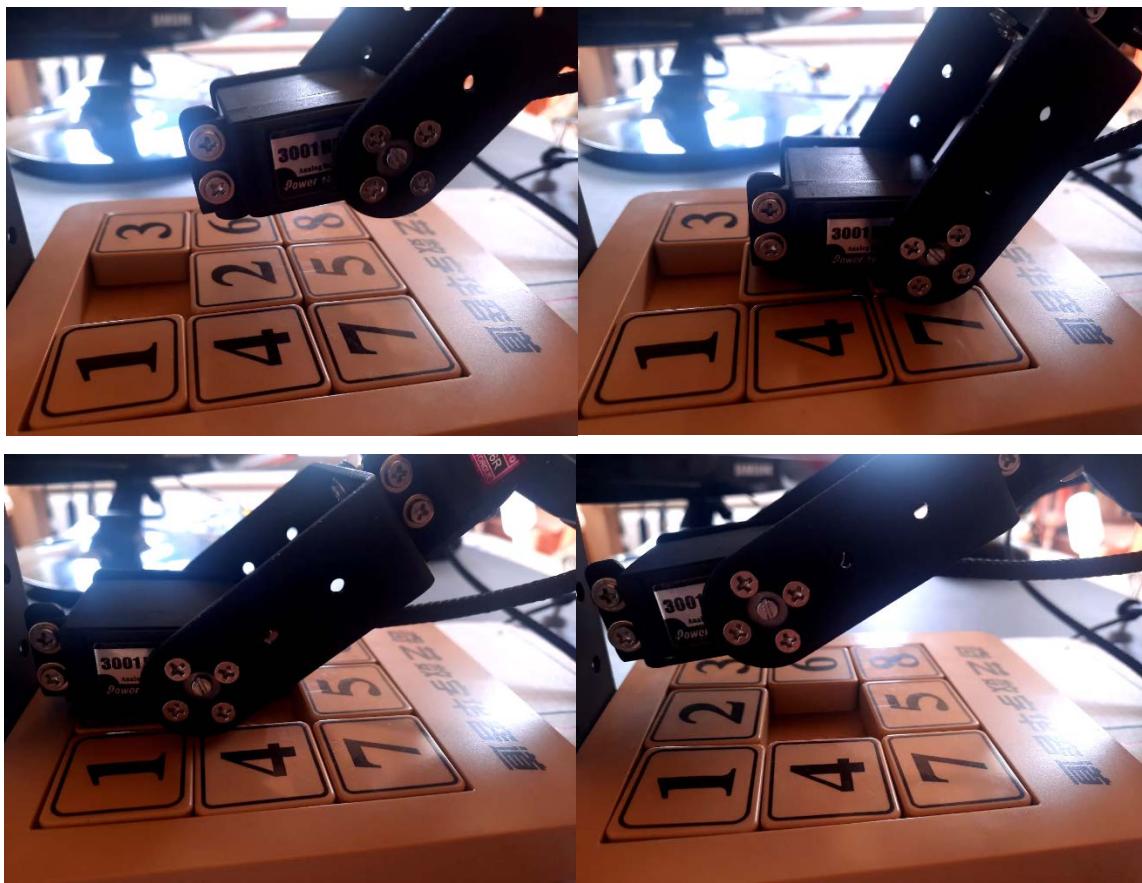


Rys. 4.18 Parametry wytrenowanego modelu: Accuracy - procent poprawnych predykcji, oraz Loss – różnice w wartościach predykcji oraz rzeczywistych, wraz ze wzrostem liczby epok [34]

Koncepcja robota obdarzonego cechami sztucznej inteligencji rozwiązującego układankę z losowo zmieszanyimi puzzlami.

4.4 Ruch manipulatora

Po zaimplementowaniu programu odpowiedzialnego za sterowanie manipulator jest w stanie wykonywać przesunięcia w oparciu o pozyskane dane z kamery. Na Rys. 4.19 przedstawiono ruch ramienia w górę do pustego miejsca który obejmuje następujące kroki: zatrzymanie się nad elementem, na elemencie, przesunięcie oraz ponowne zatrzymanie nad nim.

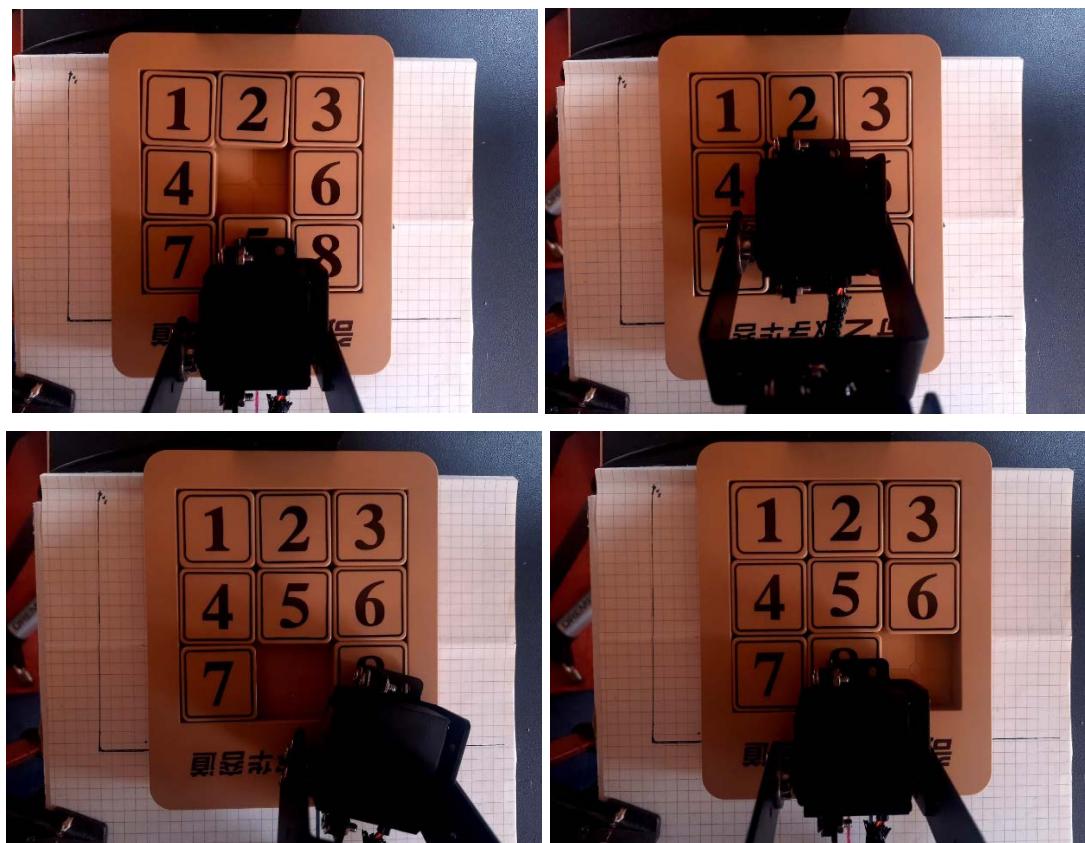


Rys. 4.19 Etapy wykonanego ruchu w górę do pustego miejsca (materiał własny)

Ruch serwomechanizmów odbywa się poprzez przyrost wypełnienia, który reguluje ich prędkość. Manipulator najpierw ustawia się w pozycji początkowej poprzez ustawienie sygnału PWM, co następuje z maksymalną prędkością serw. Po tym może rozpocząć płynne, wolniejsze ruchy, które odbywają się w pętli od poprzedniej pozycji do następnej, o określony przyrost w stopniach.

Koncepcja robota obdarzonego cechami sztucznej inteligencji rozwiązującego układankę z losowo zmieszanyimi puzzlami.

Dzięki zaimplementowanej funkcji obliczającej prędkości na podstawie macierzy jacobianowej, zasymulowano ruch liniowy. Ze względu na specyfikę działania serw nie jest on idealny, ponieważ ze względu na opóźnienia w sygnale PWM nie zawsze poruszają się one z tą samą prędkością. Po niewielkich korektach dodanych do prędkości kalibrujących ich wspólne działanie uzyskano zadowalający wynik, co przedstawiono na Rys. 4.20:



Rys. 4.20 Ruch manipulatora po osiach x i y, widok z góry (materiał własny)

Koncepcja robota obdarzonego cechami sztucznej inteligencji rozwiązującego układankę z losowo zmieszanyimi puzzlami.

4.5 Struktura oraz działanie programu

Cały program został podzielony na obszar związany z obrazem (folder „Obraz”) oraz z ruchem (folder „Ruch”). W folderze głównym „Robot_puzzle” znajdują się również pliki „main.py”, który stanowi plik uruchomieniowy oraz „Puzzle.py” z algorytmem A*. Na Rys. 4.22 uwidoczniono fragment edytora Atom z otwartym projektem. Listing całego programu ujęto w załącznikach 1-7.

The screenshot shows the Atom code editor interface. On the left, the 'Project' sidebar displays a file tree for a project named 'Robot_puzzle'. The tree includes a 'Robot_puzzle' folder containing 'Obraz' and 'Ruch' subfolders, each with various Python files like 'model.py', 'Kamera.py', etc. Below the tree are files 'main.py' and 'Puzzle.py'. The right panel shows the content of 'main.py'. The code is color-coded, with syntax highlighting for keywords, comments, and variable names. The code itself is a Python script that imports cv2, evdev, and other modules from the 'Ruch' and 'Obraz' packages. It defines a 'Start' class with methods for initializing a servomotor ('serwo'), performing期待 ('oczekiwanie'), taking a camera image ('kamera'), and solving a puzzle ('puzzle'). A detailed comment explains the initialization of the 'Serwo' class from the 'Pozycje' module. The status bar at the bottom indicates the file is 'main.py', it's 4:30, and there are 11 changes.

```
Project
Robot_puzzle
  Obraz
    model
      keras_model.h5
      labels.txt
    Kamera.py
    Rozpoznawanie.py
  Ruch
    Kinematyka.py
    PCA9685.py
    Serwo.py
  main.py
  Puzzle.py

main.py
1  #!/usr/bin/python3
2  import cv2
3  import evdev
4  from evdev import InputDevice, categorize, ecodes
5  from Ruch.Serwo import Serwo, Pozycje
6  from Ruch.Kinematyka import Prosta, Odwrotna, Predkosc
7  from Obraz.Kamera import Kamera
8  import Obraz.Rozpoznawanie as rozpoznawanie
9  from Puzzle.Puzzle import Puzzle
10 from time import sleep
11 import numpy as np
12
13 class Start:
14
15     def __init__(self,model,startowa="start"):
16         self.serwo(startowa)
17         self.model = model
18         self.oczekiwanie()
19         self.kamera()
20         self.puzzle()
21
22     def serwo(self,startowa):
23         ...
24         Funkcja inicjuje klasę Serwo od pozycji początkowej
25         zapisanej w klasie Pozycje.
26         ...
27         # ----- Inicjalizacja klas ----- ##
28         pozycje = Pozycje()
29         odwrotna = Odwrotna()
30
31         # ----- obliczenie kątów dla startowej pozycji ----- ##
32         T = odwrotna.oblicz(pozycje.poz[startowa])
33         p = 0.3 #predkosc
34         self.p = p
35         self.serwo = Serwo/T[n][n][n][n]
```

Rys. 4.22 Struktura programu – widok edytora Atom (materiał własny)

Poniżej opisano poszczególne moduły programu oraz klasy, które zostały użyte do poszczególnych zadań.

Koncepcja robota obdarzonego cechami sztucznej inteligencji rozwiązującego układankę z losowo zmieszanymi puzzlami.

- **Obraz**

Moduł składa się z dwóch plików: „Kamera.py” oraz „Rozpoznawanie.py” oraz folderu zawierającego wytrenowany model do rozpoznawania cyfr.

Klasa „Kamera()” w „Kamera.py” odpowiada za wykonanie zdjęcia oraz jego obróbkę, celem znalezienia lokalizacji elementów na układance. Główna funkcja „szukaj” zwraca już obliczone współrzędne w układzie bazy.

W pliku „Rozpoznawanie.py” znajduje się obsługa wytrenowanego modelu dostarczona przez Teachable Machine. Funkcja pobiera kolejno zdjęcia z liczbami do predykcji oraz zwraca najbardziej prawdopodobny wynik.

- **Ruch**

Moduł składa się z trzech plików: „PCA9685.py”, „Kinematyka.py” oraz „Serwo.py”.

Plik „PCA9685.py” stanowi konfigurację sterownika PWM. Zawiera on gotowe funkcje pozwalające na ustawienie danego serwonapędu przy pomocy wypełnienia, którego zakres znajduje się pomiędzy wartościami 400 – 2500.

W pliku „Kinematyka.py” zawarto wszelkie macierze przekształceń potrzebne do obliczania pozycji oraz prędkości. Klasa „Kinematyka()” podzielona jest na podklasy „Prosta()”, „Odwrotna()” oraz „Predkosc()”.

W pliku „Serwo.py” w klasie „Serwo()” znajdują się funkcje pomocnicze do sterowania serwonapędami, czyli zakresy wypełnień, przeliczanie ich na stopnie oraz ustawianie całego manipulatora na konkretne pozycje, z daną prędkością. Druga klasa „Pozycje()” stanowi bufor do przechowywania pozycji wygenerowanych przez moduł Obrazu.

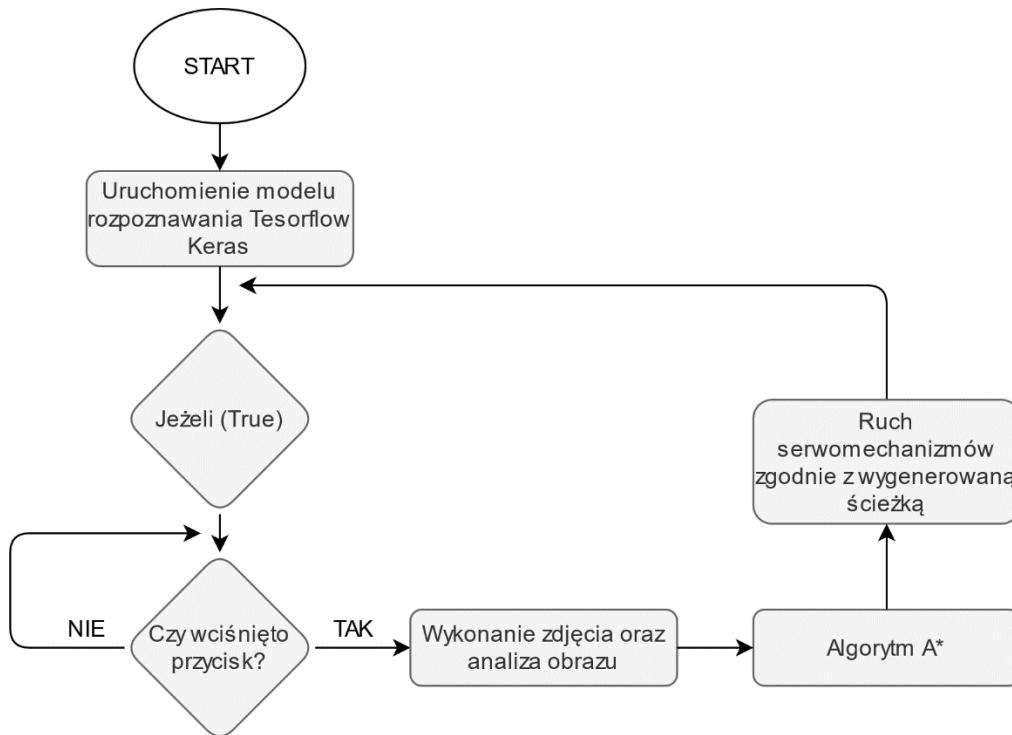
- **Algorytm A***

Algorytm znajduje się w głównym folderze w pliku „Puzzle.py” i może stanowić samodzielną część programu.

W klasie Puzzle() zawarto wszelkie operacje dotyczące przeszukiwania grafu za pomocą algorytmu A*.

Koncepcja robota obdarzonego cechami sztucznej inteligencji rozwiązującego układankę z losowo zmieszanyimi puzzlami.

Program główny zawarty w pliku „main.py” uruchamia wszystkie pozostałe moduły, poniżej na Rys. 4.23 zaprezentowano jego kolejność działań:



Rys. 4.23 Schemat głównego programu (materiał własny)

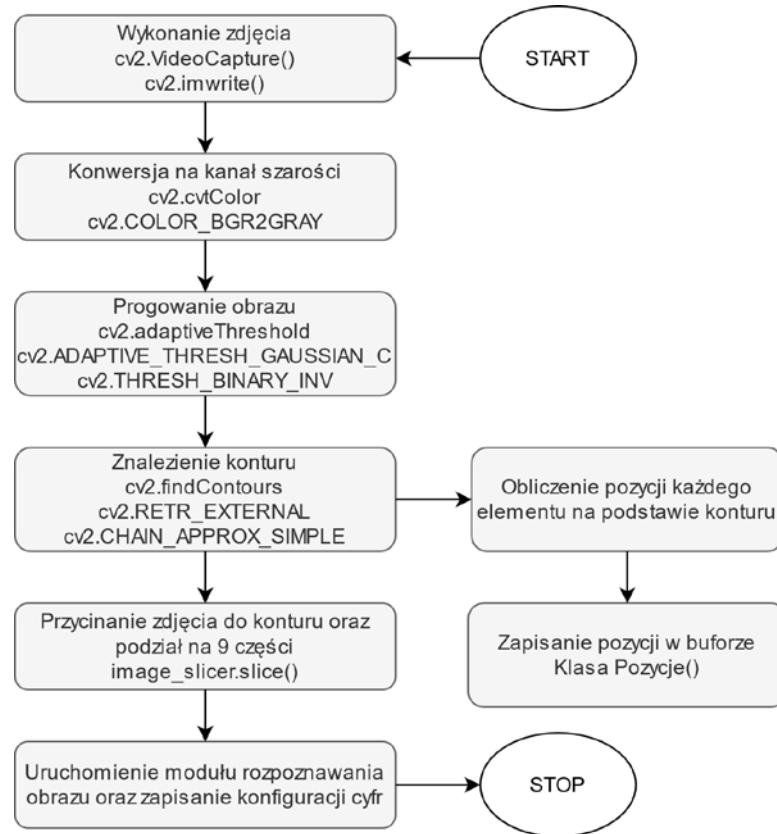
Po uruchomieniu programu, w pierwszej kolejności ładuje się biblioteka Tensorflow. Jest to proces, który może trwać do 30s dlatego zaproponowano rozwiązanie, w którym odbywa się to tylko raz. Następnie program rozpoczyna nieskończoną pętlę, w której oczekuje na działanie użytkownika, jakim jest wciśnięcie przycisku kamery. Zostało to wykonane przy pomocy biblioteki evdev, która pozwala na wykonanie działania od przerwania w wybranym urządzeniu na architekturze typu Linux. Przycisk kamery wywołuje przerwanie widoczne w systemie tylko wtedy, kiedy jest uruchomiona za pomocą `cv2.VideoCapture()`, dlatego jego obsługa znajduje się w funkcji `Kamera().film()`, gdzie obraz jest pobierany bez wyświetlania go w oknie.

Użytkownik po zmieszaniu elementów układanki może nacisnąć przycisk wtedy, gdy lampka kamery świeci się na niebiesko – oznacza to, że program uruchomił urządzenie i przeszedł w stan oczekiwania na przerwanie.

Koncepcja robota obdarzonego cechami sztucznej inteligencji rozwiązującego układankę z losowo zmieszanyimi puzzlami.

- Wykonanie zdjęcia oraz analiza obrazu

Poniżej na Rys. 4.24 ukazano schemat blokowy modułu rozpoznawania obrazu:



Rys. 4.24 Schemat blokowy modułu Obraz (materiał własny)

Po wywołaniu programu za pomocą funkcji `cv2.VideoCapture()` oraz `cv2.imwrite()` zapisywane jest zdjęcie wykonane z odpowiednim naświetleniem, przy pomocy dodatkowej funkcji opóźniającej zawartej w Listingu 1.9:

```
def zdjecie(self,nazwa):
    kamera = cv2.VideoCapture(0)
    x = 0
    while True:
        _, frame = kamera.read()
        key = cv2.waitKey(1)
        if x==20:
            cv2.imwrite( "puzzle_rgb.png", frame )
            break
        x+=1
```

Listing 1.9 Pętla opóźniająca zapis obrazu (materiał własny)

Koncepcja robota obdarzonego cechami sztucznej inteligencji rozwiązującego układankę z losowo zmieszanyimi puzzlami.

Następnie, pozostałe funkcje OpenCV opisane wcześniej w rozdziale 4.3 przygotowują obraz układanki do rozpoznawania cyfr. Na Listingach 1.10 oraz 1.11 przedstawiono fragmenty programu odpowiedziane za obróbkę obrazu oraz zapis pozycji układanki do zmiennej `Pozycja_B`. Funkcja `image_slicer.slice()` dzieli obraz na 9 części.

```
frame = cv2.imread("puzzle_rgb.png")
gray = cv2.cvtColor(frame, cv2.COLOR_RGB2GRAY)
threshold = cv2.adaptiveThreshold(gray, 255, \
cv2.ADAPTIVE_THRESH_GAUSSIAN_C,\n
cv2.THRESH_BINARY_INV, 21, 3)
cv2.imwrite( nazwa, threshold )
```

Listing 1.10 Fragment funkcji „Kamera().zdjecie()” (materiał własny)

```
def szukaj(self):
    puzzle_rgb = cv2.imread("puzzle_rgb.png")
    puzzle = cv2.imread("puzzle.png",0)
    puzzle_ROI = cv2.imread("puzzle_rgb.png")
    contours, hierarchy = cv2.findContours(puzzle,
    cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
    for contour in contours:
        [x, y, w, h] = cv2.boundingRect(contour)
        if w<400 or h<400:
            continue
        cv2.rectangle(puzzle_rgb,(x,y),(x+w, y+h),\
        (255, 0, 255), 2)
        P = self.pozycje(x, y, w, h)
        Pozycja_B = self.px_cm(P)
        ROI = puzzle_ROI[y:y+h,x:x+w]
        ROI = cv2.rotate(ROI, cv2.ROTATE_180)

        cv2.imwrite( "ROI.png", ROI )
        slice("ROI.png",9)

    cv2.imwrite( "kontury.png", puzzle_rgb )
    return Pozycja_B
```

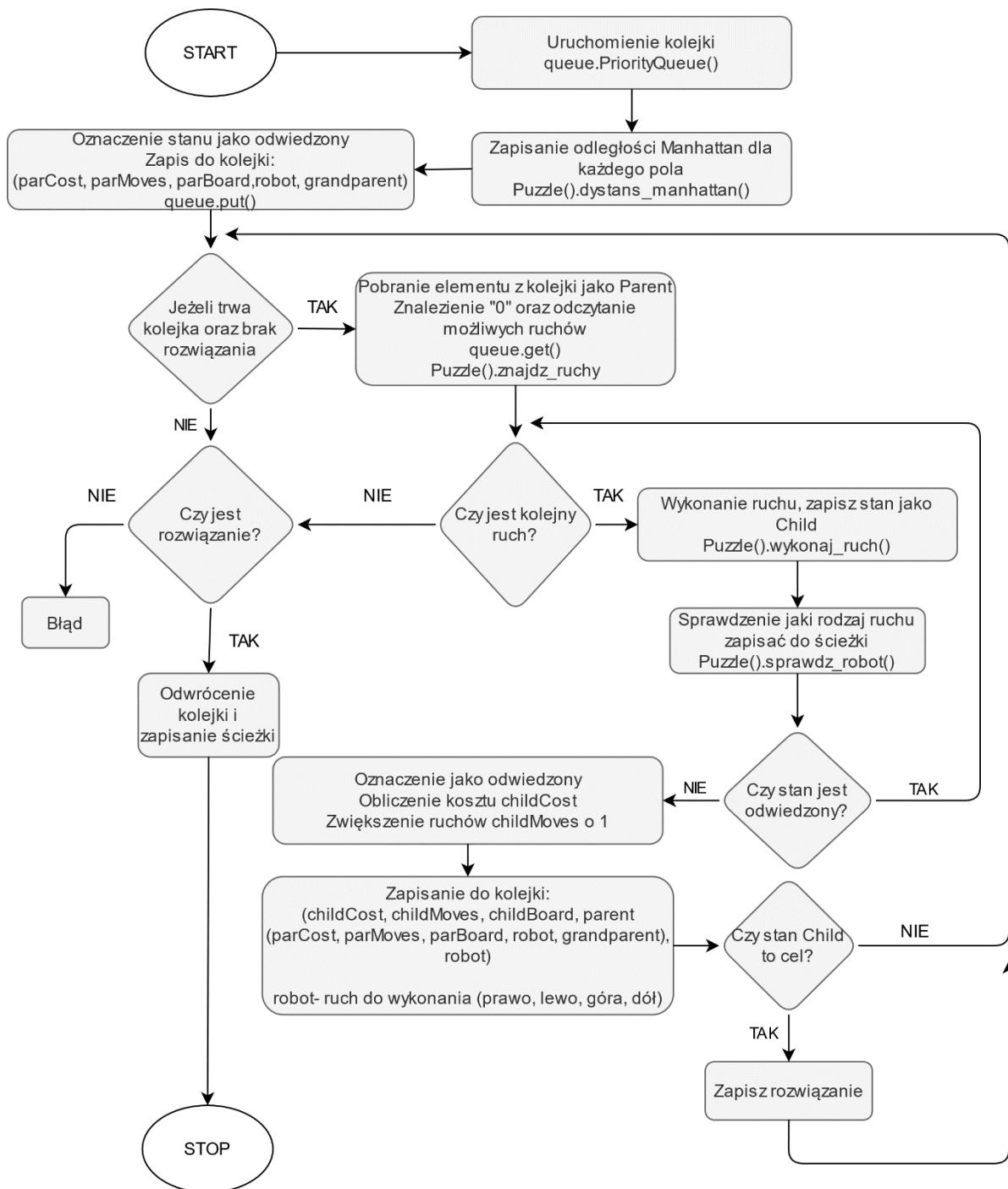
Listing 1.11 Funkcja „Kamera().szukaj()” (materiał własny)

Listing programu rozpoznawania obrazu umieszczono w załączniku nr 2. Program pobiera kolejno wycięte fragmenty, których nazewnictwo odpowiada numerom pól na układance (0-8) oraz zwraca konfigurację cyfr na układance.

Koncepcja robota obdarzonego cechami sztucznej inteligencji rozwiązującego układankę z losowo zmieszanyimi puzzlami.

- **Algorytm A***

Na Rys. 4.25 przedstawiono schemat blokowy algorytmu A*:



Rys. 4.25 Schemat blokowy algorytmu A* (materiał własny)

Koncepcja robota obdarzonego cechami sztucznej inteligencji rozwiązującego układankę z losowo zmieszanyimi puzzlami.

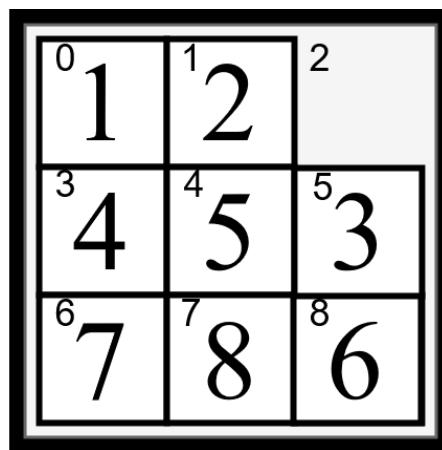
Funkcja `Puzzle().szukaj()` rozpoczyna swoje działanie od utworzenia kolejki priorytetowej `PriorityQueue()` oraz listy na odwiedzone stany. Kolejka dodając kolejne stany sortuje je od najmniejszego kosztu, który jest sumą odległości poszczególnych liczb od ich miejsca docelowego oraz liczby wykonanych ruchów. Funkcja `Puzzle().dystand_manhattan()` przy pomocy wcześniej utworzonej listy odległości dla poszczególnych pól (Listing 1.12) zwraca zsumowany koszt danego stanu.

```
def odleglosc(self):
    tabela_odl = {};
    for aa in range(9):
        for bb in range(9) :
            arow = aa//3; acol=aa%3
            brow = bb//3; bcol=bb%3
            tabela_odl[(aa,bb)] = abs(arow-\
            brow)+abs(acol-bcol)
    return tabela_odl
```

```
tabela_odl:
{
(0, 0): 0, (0, 1): 1, (0, 2): 2, (0, 3): 1, (0, 4): 2, (0, 5): 3, (0,
6): 2, (0, 7): 3, (0, 8): 4, (1, 0): 1, (1, 1): 0, (1, 2): 1, (1, 3):
2, (1, 4): 1, (1, 5): 2, (1, 6): 3, (1, 7): 2,...(8, 8): 0
}
```

Listing 1.12 Utworzenie odległości dla każdego pola układanki [35]

Poniżej na Rys. 4.26 przedstawiono sposób numeracji pól na układance, który używany jest w programie:



Rys. 4.26 Numeracja pól na układance (materiał własny)

Koncepcja robota obdarzonego cechami sztucznej inteligencji rozwiązującego układankę z losowo zmieszanyimi puzzlami.

W każdej operacji dodawania stanu do kolejki zawarty jest: koszt dziecka (childCost), suma ruchów dziecka (childMoves), stan dziecka (childBoard), informacja z nazwą wykonywanego ruchu (robot = prawo/lewo/góra/dół) oraz te same informacje dotyczące rodzica (Parent = (parCost, parMoves, parBoard, robot, grandparent)). Dzięki takiej konstrukcji tworzy się zagnieżdzona lista zawierająca drogę powrotną do stanu początkowego. Przykład takiej listy przedstawiono na Listing 1.13.

Program określa możliwe ruchy znajdując najpierw pozycję pustego miejsca – 0. Funkcja `Puzzle().znajdz_ruchy()` sprawdza dostępność ruchów z numeru tej pozycji dzięki wcześniejszemu określeniu ich przy pomocy listy `legalneRuchy` (Listing 1.14). Lista `legalneRuchy` dotyczy przemieszczenia cyfr o określone miejsce ponieważ konfiguracja zawarta jest w stringu np. „123456780”, więc aby przesunąć element w dół należy przesunąć cyfrę o 3.

Listing 1.13 Przykład końcowej zawartości listy z kolejki po znalezieniu rozwiązania (materiał własny)

```
self.legalneRuchy = {
0: (1,3      ),  # pola +1, +3 moga przesunac sie na 0
1: (-1,3,1),
2: (-1,3),
3: (-3,1,3),
4: (-3,-1,1,3),
5: (-3,-1,3),
6: (-3,1),
7: (-3,-1,1),
8: (-3,-1)}
```

Listing 1.14 Lista pomocnicza do określenia ruchów [35]

Koncepcja robota obdarzonego cechami sztucznej inteligencji rozwiązującego układankę z losowo zmieszanyimi puzzlami.

Pętla for zawarta w funkcji `Puzzle().szukaj()` wybiera kolejne wygenerowane ruchy dla danego stanu oraz zapisuje go do kolejki, gdzie dalej ulega on sortowaniu. Po każdym rozpatrywanym stanie jest on oznaczany jako odwiedzony, dzięki czemu funkcja nie zapisuje go ponownie do kolejki. Na koniec pętli stan porównywany jest ze stanem docelowym. Jeżeli program znalazł rozwiązanie przechodzi do odczytywania ścieżki według pozycji Parent, a następnie ją odwraca (Listing 1.15).

```
if wynik :  
    print(queue.get())  
    path = []  
    while wynik :  
        path.append(wynik[1:4])#dodaj pozycje 1-3 do sciezki  
        wynik = wynik[4]          #przejdz do rodzica  
        path.reverse()  
    return path  
else :  
    return []
```

Listing 1.15 Proces odczytywania ścieżki [35]

Przykład wypisanego wyniku znajduje się na Listing 1.16, gdzie liczba ruchów odpowiada długości ścieżki - `len(path)`.

Koncepcja robota obdarzonego cechami sztucznej inteligencji rozwiązującego układankę z losowo zmieszanymi puzzlami.

```
Ruchy = 28
(0, '103654872', '')
(1, '153604872', 'gora')
(2, '153640872', 'lewo')
(3, '153642870', 'gora')
(4, '153642807', 'prawo')
(5, '153602847', 'dol')
(6, '153062847', 'prawo')
(7, '153862047', 'gora')
(8, '153862407', 'lewo')
(9, '153862470', 'lewo')
(10, '153860472', 'dol')
(11, '153806472', 'prawo')
(12, '153086472', 'prawo')
(13, '153486072', 'gora')
(14, '153486702', 'lewo')
(15, '153406782', 'dol')
(16, '103456782', 'dol')
(17, '130456782', 'lewo')
(18, '136450782', 'gora')
(19, '136452780', 'gora')
(20, '136452708', 'prawo')
(21, '136402758', 'dol')
(22, '136420758', 'lewo')
(23, '130426758', 'dol')
(24, '103426758', 'prawo')
(25, '123406758', 'gora')
(26, '123456708', 'gora')
(27, '123456780', 'lewo')
```

Listing 1.16 Wygenerowana ścieżka (materiał własny)

Koncepcja robota obdarzonego cechami sztucznej inteligencji rozwiązującego układankę z losowo zmieszanymi puzzlami.

- **Ruch serwomechanizmów zgodnie ze ścieżką**

W pliku „main.py” w funkcji `Start().puzzle()` znajduje się obsługa każdego etapu w wygenerowanej ścieżce (Listing 1.17), która korzysta z funkcji `Start().ruch()` do ustawiania manipulatora na konkretne pozycje.

```
for etap in sciezka:  
    kierunek = etap[2]  
    x,y,z = pozycje.poz[zero]  
    zero = etap[1].find("0")  
  
    if kierunek == "lewo":  
        #wysokosc nad elementem,pozycja o 3cm w prawo od pustego pola  
        poz[0] = (x+3,y,8)  
        poz[1] = (x+3,y,3.8)  
        # -1cm dla pewności przesunięcia w lewo  
        poz[2] = (x-1,y,3.8)  
        poz[3] = (x-1,y,8)
```

Listing 1.17 Fragment funkcji „Start().puzzle()” – zapis pozycji ruchu w lewo (materiał własny)

Funkcja `Start().ruch()` może wygenerować ruch serwomechanizmów z tą samą prędkością, lub z obliczoną w taki sposób aby powstał ruch liniowy. Rozpoczyna swoje działanie od uruchomienia funkcji `Odwrotna().oblicz` znajdującej się w pliku „Kinematyka.py” aby zapisać pozycje w formie kątów, a następnie z pliku „Serwo.py” dzięki funkcjom `oblicz_predkosc_poz1()` oraz `Serwo().ustaw_poz()` wykonywany jest ruch serwomechanizmów.

Na Listing 1.18 oraz Listing 1.19 przedstawiono fragmenty funkcji odpowiedzialnych za obliczanie prędkości na podstawie pozycji z której rozpoczyna się ruch, oraz zamianę współrzędnych kartezjańskich dostarczonych przez moduł Obrazu na kąty konfiguracyjne.

Koncepcja robota obdarzonego cechami sztucznej inteligencji rozwiązującego układankę z losowo zmieszanyimi puzzlami.

```

def oblicz(self,polozenie):
    x, y, z = polozenie
    #obliczenie kata Theta1 dla przypadkow: x=0, x>0, x<0
    T, R = self.zaleznosci_x(polozenie)
    a = self.a
    z = z-a[1]
    Fi = {}
    #implementacja zaleznosci z trojkatow
    R[2] = sqrt(pow(z,2) + pow(R[1],2))
    Fi[1] = np.arctan(z/R[1])
    Fi[2] = np.arccos( ( pow(a[2],2) + pow(R[2],2)-\
        pow(a[3],2) ) / ( 2*a[2]*R[2] ) )
    Fi[3] = np.arccos((pow(a[2],2) + pow(a[3],2) -\
        pow(R[2],2)) / ( 2*a[2]*a[3] ) )
    #Obliczenie kata Theta2 dla przypadkow: z=0, z>0, z<0
    T = self.zaleznosci_z(z,a,R,Fi,T)
    T[3] = -(pi-Fi[3])
    T[4] = -T[3]-T[2]

    for x in range(1,5): #zamiana katow Theta na stopnie
        T[x] = (T[x]/pi)*180
    T = (T[1],T[2],T[3],T[4]) #zamiana slownika na krotke
    return T

```

Listing 1.18 Funkcja główna kinematyki odwrotnie (materiał własny)

```

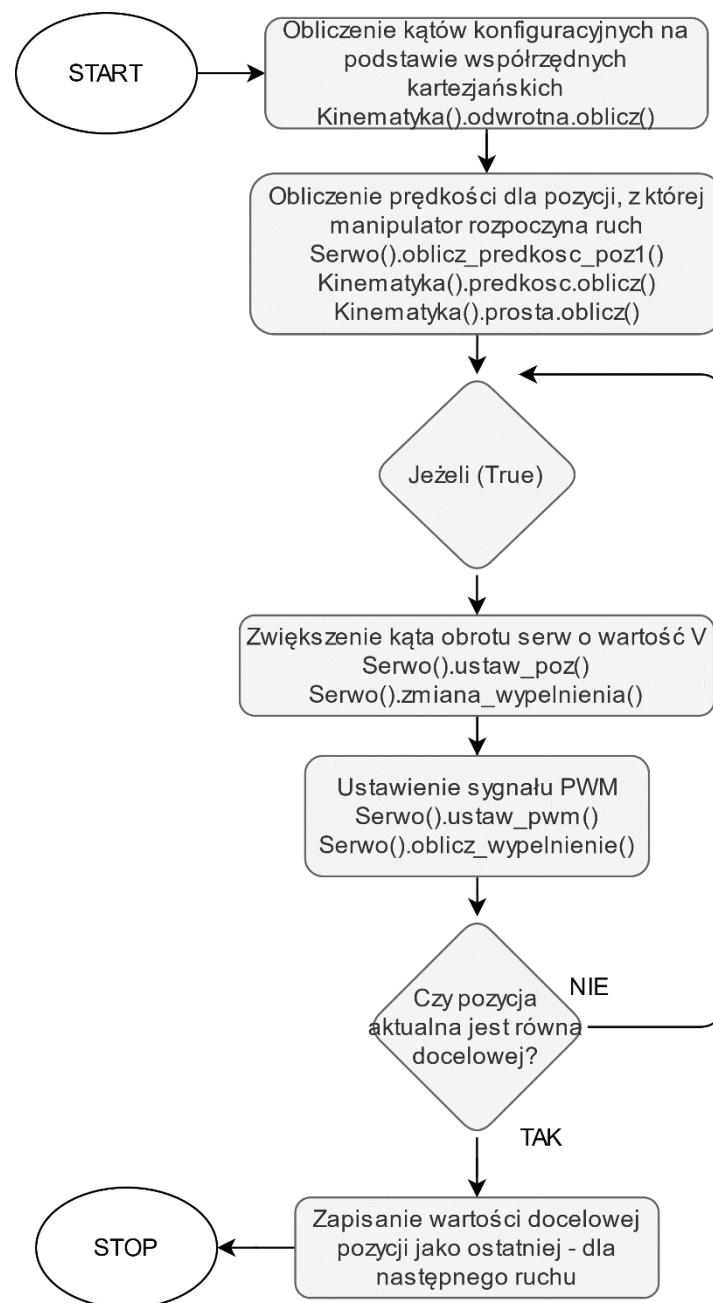
def oblicz(self,Poz_akt,P1,P2):
    P1 = list(P1) #pozycja poprzednia xyz
    P2 = list(P2) #pozycja docelowa xyz
    Poz_akt = list(Poz_akt) #pozycja aktualna w stopniach
    V_3 = self.kierunek(P1,P2)# funkcja okreslenia kierunku ruchu
    alfa = self.alfa
    alfa, T = self.stopnie_na_rad(alfa,Poz_akt)
    H = self.macierze_DH(T,alfa)
    H_mult = self.mnozenie(H)
    #utworzenie macierzy jakobianu
    J1, J2, J3 = self.jakobian(H_mult)
    Jakobian = hstack((J1,J2,J3))
    Jakobian_odw = linalg.inv(Jakobian)
    #obliczenie przyrostu dla trzech serw
    T_dot1 = round(abs(Jakobian_odw[0,0]*V_3[0]+\
        Jakobian_odw[0,1]*V_3[1]+Jakobian_odw[0,2]*V_3[2]),6)
    T_dot2 = round(abs(Jakobian_odw[1,0]*V_3[0]+\
        Jakobian_odw[1,1]*V_3[1]+Jakobian_odw[1,2]*V_3[2]),6)
    T_dot3 = round(abs(Jakobian_odw[2,0]*V_3[0]+\
        Jakobian_odw[2,1]*V_3[1]+Jakobian_odw[2,2]*V_3[2]),6)
    T_dot = [T_dot1,T_dot2,T_dot3]
    return T_dot

```

Listing 1.19 Funkcja główna obliczania prędkości (materiał własny)

Koncepcja robota obdarzonego cechami sztucznej inteligencji rozwiązującego układankę z losowo zmieszanyymi puzzlami.

Poniżej na Rys. 4.27 przedstawiono schemat blokowy ruchu serw, który rozpoczyna się w funkcji `Start().ruch()` po zapisaniu pozycji dla konkretnego kierunku (prawo, lewo, góra, dół).

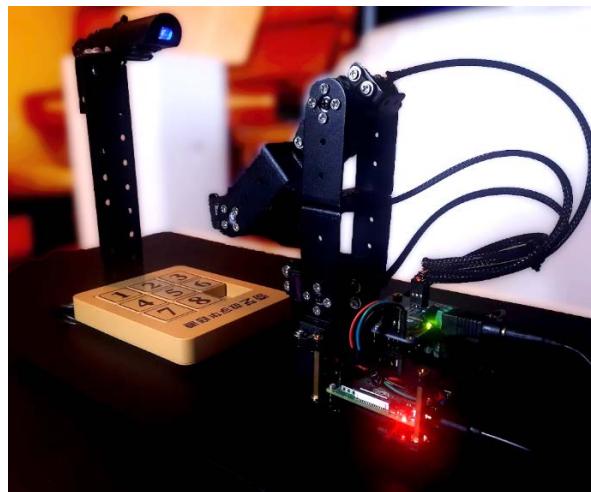


Rys. 4.27 Schemat blokowy ruchu serwomechanizmów (materiał własny)

Koncepcja robota obdarzonego cechami sztucznej inteligencji rozwiązującego układankę z losowo zmieszanyimi puzzlami.

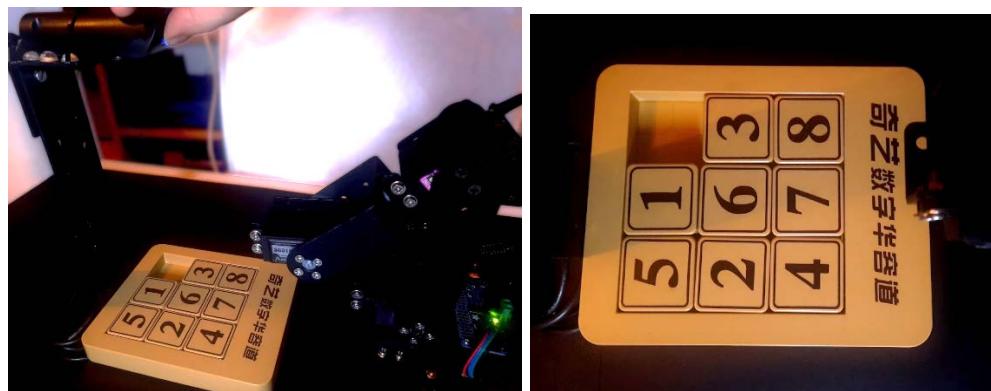
4.6 Podsumowanie rezultatów

Robot po uruchomieniu programu oczekuje na działanie użytkownika, a swoją gotowość oznacza niebieską lampką w kamerze dzięki zainicjowanemu modułowi, który oczekuje na przerwanie od przycisku (Rys. 4.28).



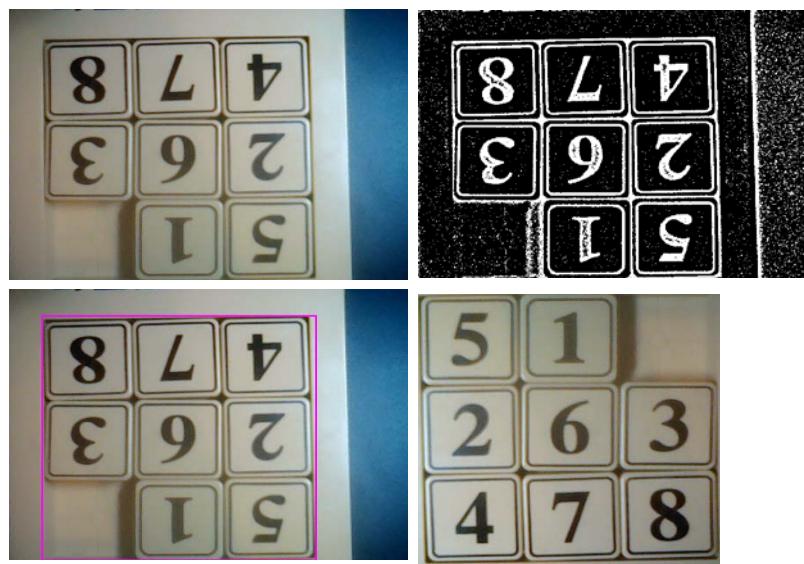
Rys. 4.28 Robot z włączonym programem w oczekiwaniu na wciśnięcie przycisku (materiał własny)

Po zmieszaniu elementów oraz naciśnięciu przycisku (Rys. 4.29), robot wykonuje zdjęcie oraz je analizuje, zwracając pozycje każdego elementu układanki wraz z rozpoznaną konfiguracją cyfr. Czas trwania od wykonania zdjęcia do obliczenia sekwencji ruchów trwa łącznie nie więcej niż 2 sekundy, po czym robot przechodzi do rozwiązywania. Na Rys. 4.30 przedstawiono wyniki analizy obrazu danej konfiguracji.



Rys. 4.29 Zmieszanie puzzli oraz wciśnięcie przycisku (materiał własny)

Koncepcja robota obdarzonego cechami sztucznej inteligencji rozwiązującego układankę z losowo zmieszanyimi puzzlami.



Rys. 4.30 Analiza obrazu układanki (materiał własny)

Pozostałe parametry obliczone w programie mogą zostać odczytane z konsoli (Listing 1.20).

```
Zapisane pozycje: {'start': (0, 6, 10), 0: (-3.54, 15.58, 8),
1: (-0.14, 15.58, 8), 2: (3.27, 15.58, 8), 3: (-3.54, 12.23, 8),
4: (-0.14, 12.23, 8), 5: (3.27, 12.23, 8), 6: (-3.54, 8.87, 8),
7: (-0.14, 8.87, 8), 8: (3.27, 8.87, 8)}

Rozpoznane cyfry {0: 5, 1: 1, 2: 0, 3: 2, 4: 6, 5: 3, 6: 4, 7: 7, 8: 8}

Start: 510263478
Cel: 123456780
Rozwinięto 13 stanów. W kolejce znajduje się jeszcze 13 stanów.
Liczba ruchów= 11

Pozycja DOCELOWA:
Odwrótka: (75.4, 54.4, -97.9, 43.5) Prosta: (3.3, 12.6, 8.0)
Pozycja POPRZEDNIA:
Odwrótka: (90.0, 95.3, -138.1, 42.8)
Predkosc: [1.333333, 0.061848, 1.201006]
-----
Pozycja DOCELOWA:
Odwrótka: (75.4, 35.0, -97.4, 62.4) Prosta: (3.3, 12.6, 3.6)
Pozycja POPRZEDNIA:
Odwrótka: (75.4, 54.4, -97.9, 43.5)
Predkosc: [0.0, 0.52963, 0.161953]
-----
...

```

Listing 1.20 Fragment wyniku działania programu (materiał własny)

Koncepcja robota obdarzonego cechami sztucznej inteligencji rozwiązującego układankę z losowo zmieszanyimi puzzlami.

Po ułożeniu elementów robot wraca do pozycji startowej oraz oczekuje na kolejny ruch użytkownika, aż do momentu wyłączenia programu (Rys. 4.31).



Rys. 4.31 Ułożenie oraz powrót do pozycji początkowej (materiał własny)

Podsumowując, robot poprawnie wykonuje swoje działania zgodnie z założeniami niniejszej pracy dyplomowej. Dzięki poprawnie zaimplementowanym obliczeniom kinematycznym, pomimo występujących luzów konstrukcyjnych manipulator kieruje końcówkę roboczą na właściwe pozycje.

Podczas działania występują błędy położenia o wartości do 0,5cm, które niwelowane są poprzez mniejszy punkt nacisku końcówki w stosunku do pola elementu układanki. Dzięki zmatowieniu powierzchni gumowej podkładki, zwiększone zostało tarcie pomiędzy końcówką oraz układanką, co poprawiło znacznie przesuwanie elementów oraz zminimalizowało błędy.

Koncepcja robota obdarzonego cechami sztucznej inteligencji rozwiązującego układankę z losowo zmieszanyimi puzzlami.

WNIOSKI

Stworzony robot odznacza się cechami sztucznej inteligencji poprzez umiejętność rozpoznawania elementów układanki oraz jej rozwiązywania. Proces tworzenia tak złożonej konstrukcji wymagał dogłębnej analizy każdego z zagadnień oraz połączenia ich w spójną całość.

Obliczenia kinematyczne oraz schematy należało wykonywać z dużą precyzją, ponieważ nagromadzenie błędów mogło doprowadzić do niezadowalających efektów. Skuteczne sterowanie wynika z prawidłowo dobranych zakresów wypełnienia PWM osobno dla każdego serwonapędu, oraz ich poprawnego montażu. Największy wpływ na rezultat działań było właściwe przytwierdzenie kamery oraz prawidłowe oznaczenie odległości pomiędzy jej układem, a układem bazowym. Proces ten składał się z wielu etapów prób oraz poprawek w kodzie programu obsługującego.

Przyrost wypełnienia, który pełnił funkcję regulacji prędkości obrotu nie jest rozwiązaniem idealnym. Lepszym sposobem na uzyskanie ruchu liniowego mogłyby być serwa o zakresie obrotu 360° lub silniki prądu stałego. Sterownik PWM oraz sam minikomputer Raspberry Pi mogą tworzyć opóźnienia, które skutkują nieregularnym sterowaniem. W związku z tym, robot podczas przesuwania elementów niekiedy zatrzymywał się dłużej w danej pozycji. Rozwiązaniem powyżej przytoczonych problemów mogłyby być rozdzielenie funkcji i przeniesienie sterowania na dodatkową platformę, przykładowo Arduino.

Biblioteki stworzone na podstawy uczenia maszynowego oraz analizy obrazu pozwoliły na skuteczne rozpoznawanie cyfr oraz określenie ich lokalizacji. Założeniem dotyczącym pracy robota z układanką była możliwość przesuwania jej tak, aby użytkownik nie musiał umieszczać jej za każdym razem w idealnej pozycji, a za razem mógł ją swobodnie podnosić oraz odkładać. Element ten został również spełniony, o ile elementy znajdowały się w polu widzenia kamery.

Wytrenowanie modelu do rozpoznawania cyfr obejmowało wykonanie w przybliżeniu stu zdjęć każdej cyfry oraz pustego pola. Zdjęcia te musiały być wykonane w różnym oświetleniu oraz obejmować możliwe przypadki orientacji każdego elementu. Początkowe próby rozpoznawania cyfr wykonane były przy pomocy modelu wytrenowanym na mniejszej liczbie zdjęć, co

Koncepcja robota obdarzonego cechami sztucznej inteligencji rozwiązującego układankę z losowo zmieszanyimi puzzlami.

powodowało niekompletny proces identyfikacji. Im więcej próbek zastosowano w trenowanym modelu, tym rzadziej pojawiały się niepowodzenia.

Algorytm A* sprawdził się w rozwiązywaniu zmieszanych puzzli, nie powodując dodatkowych opóźnień w działaniu programu. Znajdywana jest najkrótsza możliwa ścieżka dzięki skonstruowanej funkcji kosztu, która jest sumą odległości każdego elementu do jego docelowej pozycji oraz długości sekwencji. Projekt nie zakładał uprzedniego sprawdzania rozwiązywalności układanki, dlatego puzzle muszą być zmieszane poprzez ręczne ich przesuwanie. Swobodne wyjęcie puzzli oraz umieszczenie ich w losowej konfiguracji może skutkować brakiem rozwiązania. Do działania obejmującego ten przypadek należałoby dodać funkcję sprawdzającą czy dana konfiguracja jest rozwiązywalna. Dalszym kierunkiem badań mogłoby być porównanie różnych algorytmów stosowanych przy przeszukiwaniu grafów.

Niniejszy projekt dyplomowy stanowi kompletny, autonomiczny oraz intelligentny mechanizm, który spełnia wszystkie wymogi postawione w niniejszej pracy.

Koncepcja robota obdarzonego cechami sztucznej inteligencji rozwiązującego układankę z losowo zmieszonymi puzzlami.

BIBLIOGRAFIA

- [1] J. Honczarenko, Roboty przemysłowe: Budowa i zastosowanie., Wydawnictwa Naukowo-Techniczne, 2010.
- [2] A. Morecki i T. Knapczyk, Podstawy robotyki: Teoria i elementy manipulatorów i robotów, Wydawnictwa Naukowo-Techniczne, 1999.
- [3] J. Felis, Teoria Maszyn i Mechanizmów: Cz. I, Analiza mechanizmów, Uczelniane Wydawnictwo Naukowo-Dydaktyczne AGH, 2004.
- [4] J. J. Craig, Wprowadzenie do robotyki: Mechanika i sterowanie, Wydawnictwa Naukowo-Techniczne, 1995.
- [5] J. Denavit i R. S. Hartenberg, „A kinematic notation for lower-pair mechanisms based on matrices,” *Trans.\ ASME E, Journal of Applied Mechanics*, tom 22, pp. 215-221, 1995.
- [6] K. Waldron, „A study of overconstrained linkage geometry by solution of closure equations — Part 1. Method of study,” *Mechanism and Machine Theory*, tom 8, pp. 95-104, 1973.
- [7] R. Paul, Robot Manipulators: Mathematics, Programming and Control, Cambridge: MIT Press, 1982.
- [8] D. R. Franceschetti, Principles of Robotics Artificial Intelligence, Grey House Publishing, 2018.
- [9] S. Russell i P. Norvig, Artificial Intelligence A Modern Approach, Pearson, 2020.
- [10] B. D. Brookshear G., Computer Science - An Overview, Pearson Education Limited, 2015.
- [11] J. Wang, P. Rogers, L. Parker, D. Brooks i M. Stilman, „Robot Jenga: Autonomous and Strategic Block Extraction,” w *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2009.
- [12] H.-c. Shih, J.-L. Lu i C. Wu, „JigsawGo: A jigsaw puzzle restoration game with a robotic arm,” w *Proceedings of the 2017 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the 2017 ACM International Symposium on Wearable Computers*, 2017.
- [13] „Specyfikacja Raspberry Pi 4B”,
https://www.raspberrypi.org/documentation/hardware/raspberrypi/bcm2711/rpi_DATA_2711_1p0_preliminary.pdf, [31.05.2021].

Koncepcja robota obdarzonego cechami sztucznej inteligencji rozwiązującego układankę z losowo zmieszonymi puzzlami.

- [14] „Arduino”, <https://store.arduino.cc/arduino-uno-rev3>, [31.05.2021].
- [15] "Sklep Botland", <https://botland.com.pl/moduly-i-zestawy-raspberry-pi-4b/14646-raspberry-pi-4-model-b-wifi-dualband-bluetooth-2gb-ram-15ghz-765756931175.html>, [31.05.2021].
- [16] „A4Tech”, <http://a4tech.com/product.aspx?id=127>, [31.05.2021].
- [17] „Small Hammer”, <https://smallhammer.cc/product/snam5300-4dof-aluminum-robot-arm-diy-robotic-claw-arduino-kit>, [31.05.2021].
- [18] „Sklep Botland”, <https://botland.com.pl/serva-typu-standard/4799-serwo-towerpro-mg-996r-robot-180-standard.html>, [31.05.2021].
- [19] „Python”, <https://pl.python.org/>, [31.05.2021].
- [20] „OpenCV”, <https://opencv.org/>, [31.05.2021].
- [21] „Tensorflow”, <https://www.tensorflow.org/>, [31.05.2021].
- [22] „Keras”, <https://keras.io/>, [31.05.2021].
- [23] „Teachable Machine”, <https://teachablemachine.withgoogle.com/>, [31.05.2021].
- [24] „Diagrams”, <https://www.diagrams.net/>, [31.05.2021].
- [25] P. Górecki, Mikrokontrolery dla początkujących, Wydawnictwo BTC, 2006.
- [26] „Raspberry Pi documentation”, <https://www.raspberrypi.org/documentation/usage/gpio/>, [31.05.2021].
- [27] „Fritzing”, <https://fritzing.org/>, [31.05.2021].
- [28] „Raspberry Pi”, <https://www.raspberrypi.org/software/>, [31.05.2021].
- [29] „Dokumentacja Raspberry Pi - opcje wyświetlania w config.txt”, <https://www.raspberrypi.org/documentation/configuration/config-txt/video.md>, [31.05.2021].
- [30] „QEngineering”, <https://qengineering.eu/install-raspberry-64-os.html>, [31.05.2021].
- [31] „Waveshare”, https://www.waveshare.com/w/upload/6/6c/Servo_Driver_HAT.7z, [31.05.2021].
- [32] „OpenCV - namespace cv”, <https://docs.opencv.org/4.5.0/d2/d75/namespacencv.html>, [31.05.2021].

Koncepcja robota obdarzonego cechami sztucznej inteligencji rozwiązującego układankę z losowo zmieszonymi puzzlami.

- [33] „Teachable Machine”, <https://teachablemachine.withgoogle.com/>, [31.05.2021].
- [34] „The sliding tiles puzzle”, <https://www.openbookproject.net/py4fun/tiles/tiles.html>. [31.05.2021].
- [35] A. Sodemann, <http://robogrok.com/>, [31.05.2021].
- [36] B. Siciliano i O. Khatib, Springer Handbook of Robotics, Springer-Verlag Berlin Heidelberg, 2016.
- [37] K. Dobrowolska, W. Dyczka i H. Jakuszenkow, Matematyka : dla studentów studiów technicznych 2, Helpmath, 2006.
- [38] J. Cicolani, Beginning Robotics with Raspberry Pi and Arduino, Using Python and OpenCV, Jeff Cicolani, 2018.

ZAŁĄCZNIKI

Załącznik 1. Plik Kamera.py

```
1  #!/usr/bin/python3
2  import cv2
3  import time
4  import numpy as np
5  from image_slicer import slice
6  from numpy import cos, sin, pi
7  import evdev
8  from evdev import InputDevice, categorize, ecodes
9
10 class Kamera:
11     """
12         Wykonanie i zapis zdjecia oraz jego przygotowanie do wykrycia konturu.
13     """
14     def zdjecie(self,nazwa):
15         kamera = cv2.VideoCapture(0)
16         x = 0
17         while True:
18             _, frame = kamera.read()
19             key = cv2.waitKey(1)
20             if x==20:
21                 cv2.imwrite( "puzzle_rgb.png", frame )
22                 break
23             x+=1
24         kamera.release()
25         cv2.destroyAllWindows()
26         frame = cv2.imread("puzzle_rgb.png")
27         gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
28         threshold = cv2.adaptiveThreshold(gray, 255,
29                                         cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY_INV, 21, 3)
30         cv2.imwrite( nazwa, threshold )
31
32     """
33         Funkcja testowa do ustawiania kamery.
34     """
35     def film(self):
36         kamera = cv2.VideoCapture(0)
37         while True:
38             check, frame = kamera.read()
39             frame = cv2.resize(frame, (200, 200))
40             cv2.imshow("kamera", frame)
41             key = cv2.waitKey(5)
42             if key == 27:
43                 break
44         kamera.release()
45         cv2.destroyAllWindows()
46
47     """
48         Uruchomienie kamery bez podgladu, wcisniecie przycisku w kamerze
49         wychodzi z funkcji.
50     """
51     def oczekiwanie(self):
52         kamera = cv2.VideoCapture(0)
53         kam = InputDevice('/dev/input/event5')
54         while True:
55             check, frame = kamera.read()
56             for event in kam.read_loop():
57                 if event.type == ecodes.EV_KEY:
58                     if event.value == 1:
59                         if event.code == 212:
60                             break
61             kamera.release()
62             cv2.destroyAllWindows()
63
64     """
65         Transformacja ukladu wspolrzednych, zapisanie macierzy przejsc
66         wykonanie operacji na nich.
67         Poz_B - baza, Poz_C - kamera
68     """
69     def transformacja(self,Poz):
70         R180_x = np.matrix([[1,0,0],[0,-1,0],[0,0,-1]])
71         R180_z = np.matrix([[1,0,0],[0,-1,0],[0,0,1]])
72         p0_C = np.matrix([[6.2],[9.5],[0]])
```

Koncepcja robota obdarzonego cechami sztucznej inteligencji rozwiązującego układankę
z losowo zmieszanyimi puzzlami.

```

71      R0_C = np.matrix([[-1,0,0],[0,1,0],[0,0,-1]])
72      H0_C = np.concatenate((R0_C,p0_C),1)
73      H0_C = np.concatenate((H0_C,[[0,0,0,1]]),0)
74      x = Poz[0]
75      y = Poz[1]
76      Poz_C = [[x],[y],[0],[1]]
77      Poz_B = np.dot(H0_C,Poz_C)
78      x_B = Poz_B[0]
79      y_B = Poz_B[1]
80      return(Poz_B)
81
82      Zamiana pikseli na centymetry dla kazdej z pozycji.
83
84      def px_cm(self,P):
85          przelicznik = 0.02325
86          Poz_B = {}
87          Poz = {}
88          pozycja = {}
89          for x in P:
90              P[x] = list(P[x])
91              Poz[x] = [y* przelicznik for y in P[x]]
92              print(Poz[x],"przed zamiana w cm")
93              Poz_B[x] = self.transformacja(Poz[x])
94              x_B = float(Poz_B[x][0])
95              y_B = float(Poz_B[x][1])
96              pozycja[x] = (round(x_B,2), round(y_B-3.4,2), 8)
97          return pozycja
98
99      Podzial glownej ramki na mniejsze kwadraty i obliczenie pozycji w pikselach.
100
101     def pozycje(self,x,y,w,h):
102         P = []
103         m0 = 5/6
104         m1 = 0.5
105         m2 = 1/6
106         P[0] = (x + m0 * w, y + m0 * h)
107         P[1] = (x + m1 * w, y + m0 * h)
108         P[2] = (x + m2 * w, y + m0 * h)
109         P[3] = (x + m0 * w, y + m1 * h)
110         P[4] = (x + m1 * w, y + m1 * h)
111         P[5] = (x + m2 * w, y + m1 * h)
112         P[6] = (x + m0 * w, y + m2 * h)
113         P[7] = (x + m1 * w, y + m2 * h)
114         P[8] = (x + m2 * w, y + m2 * h)
115         return P
116
117     Glogna funkcja szukania konturu i zapisu pozycji. Funkcja pobiera
118     przygotowane zdjecie po progowaniu i wyszukuje kontur. Znaleziona ramka
119     rysowana jest na oryginalnym zdjeciu, a nastepnie tworzy sie nowy obraz
120     wylacznie z obszarem zainteresowan ROI, w celu podzielenia na 9 czesci.
121
122     def szukaj(self):
123         puzzle_rgb = cv2.imread("puzzle_rgb.png")
124         puzzle = cv2.imread("puzzle.png",0)
125         puzzle_ROI = cv2.imread("puzzle_rgb.png")
126         contours, hierarchy = cv2.findContours(puzzle, cv2.RETR_EXTERNAL,
127                                               cv2.CHAIN_APPROX_SIMPLE)
128         Pozycja_B = 0
129         for contour in contours:
130             [x, y, w, h] = cv2.boundingRect(contour)
131             #zeby nie bralo pod uwage konturow o innych wymiarach
132             if w<400 or h<400:
133                 continue
134             cv2.rectangle(puzzle_rgb, (x, y), (x + w, y + h), (255, 0, 255), 2)
135             P = self.pozycje(x, y, w, h)
136             Pozycja_B = self.px_cm(P)
137             ROI = puzzle_ROI[y:y+h,x:x+w]
138             ROI = cv2.rotate(ROI, cv2.ROTATE_180)
139             cv2.imwrite( "ROI.png", ROI )
140             slice("ROI.png",9) #podzial na mniejsze obrazy
141             cv2.imwrite( "kontury.png", puzzle_rgb )
142             return Pozycja_B
143
144     #debug
145     if __name__=='__main__':
146         kam = Kamera()
147         kam.film()
148         kam.zdjecie("puzzle.png")
149         P = kam.szukaj()
150         print(P)

```

Koncepcja robota obdarzonego cechami sztucznej inteligencji rozwiązującego układankę z losowo zmieszonymi puzzlami.

Załącznik 2. Plik Rozpoznawanie.py

```
1  import tensorflow.keras
2  from PIL import Image, ImageOps
3  import numpy as np
4
5  def model():
6      # Wyłącz adnotacje naukowe
7      np.set_printoptions(suppress=True)
8      labels_path = '/home/pi/Desktop/Robot_puzzle/Obraz/model/labels.txt'
9      labelsfile = open(labels_path, 'r')
10     # Czytaj z pliku etykiety
11     classes = []
12     line = labelsfile.readline()
13     while line:
14         classes.append(line.split(' ', 1)[1].rstrip())
15         line = labelsfile.readline()
16     labelsfile.close()
17     # Załaduj model
18     model = tensorflow.keras.models.load_model(
19         ('/home/pi/Desktop/Robot_puzzle/Obraz/model/keras_model.h5'))
20     return model, classes
21
22 def start(model, classes):
23     # Utworzenie tablicy o określonym kształcie do wpasowania w model keras
24     # Pierwszy element shape oznacza liczbę obrazów
25     data = np.ndarray(shape=(9, 224, 224, 3), dtype=np.float32)
26     k= 0
27     for i in range(1,4):
28         for j in range(1,4):
29             image = Image.open('ROI_0'+str(i)+'_0'+str(j)+'.png')
30             size = (224, 224)
31             image = ImageOps.fit(image, size, Image.ANTIALIAS)
32             image_array = np.asarray(image)
33             # Normalizuj obraz
34             normalized_image_array = (image_array.astype(np.float32) / 127.0) - 1
35             # Załaduj obraz do tablicy
36             data[k] = normalized_image_array
37             k += 1
38     # Uruchom predykcje
39     predictions = model.predict(data)
40     # Prog decyzji >70%
41     conf_threshold = 70
42     confidence = []
43     threshold_class = {}
44     puzzle_cyfry = ""
45     for j in range(9):
46         confidence = []
47         # for each one of the classes
48         for i in range(0, len(classes)):
49             # scale prediction confidence to % and append to 1-D list
50             confidence.append(int(predictions[j][i]*100))
51
52             # if above confidence threshold, send to queue
53             if confidence[i] > conf_threshold:
54                 threshold_class[j]= int(classes[i])
55                 puzzle_cyfry += classes[i]
56     print(puzzle_cyfry)
57     return puzzle_cyfry
```

Koncepcja robota obdarzonego cechami sztucznej inteligencji rozwiązującego układankę z losowo zmieszanyimi puzzlami.

Załącznik 3.Plik Kinematyka.py

```

1  import time
2  from numpy import pi, cos, sin, dot, matrix, transpose, cross,\ 
3  hstack, linalg
4  import numpy as np
5  from math import sqrt
6
7  class Kinematyka:
8      def __init__(self):
9          self.Theta = {1: 0, 2: 0, 3: 0, 4: 0}
10         self.a = {1: 6, 2: 10.5, 3: 9.5, 4: 3.5, 5: -1.7}
11         self.alfa = {1: 90, 2: 0, 3: 0, 4: -90, 5:0}
12         a = self.a
13         self.r = {1: 0, 2: a[2], 3: a[3], 4: a[4], 5: 0}
14         self.d = {1: a[1], 2: 0, 3: 0, 4: 0, 5: a[5]}
15
16     def mnozenie(self,H):
17         H_0_1 = matrix(H["01"])
18         H_0_2 = dot(H["01"], H["12"])
19         H_0_3 = dot(H_0_2, H["23"])
20         H_0_4 = dot(H_0_3, H["34"])
21         H_0_5 = dot(H_0_4, H["45"])
22         H_mult = {1:H_0_1, 2:H_0_2, 3:H_0_3, 4:H_0_4, 5:H_0_5}
23         return H_mult
24
25     def macierze_DH(self,T,alfa):
26         r = self.r
27         d = self.d
28
29 #zmienna do przechowywania macierzy przekształceni
30 #w postaci słownika, gdzie kluczem jest "01", "12", "23" itd.
31         H = {}
32         T[5] = 0 #dodanie 5 miejsca w tabeli dla theta
33         for n in range (1,6): #macierze H
34             numer = str(n-1)+ str(n)
35             #wn - wiersze do wprowadzenia do macierzy H
36             w1 = [cos(T[n]), -sin(T[n])*cos(alfa[n]), \
37                   sin(T[n])*sin(alfa[n]), r[n]*cos(T[n])]
38             w2 = [sin(T[n]), cos(T[n])*cos(alfa[n]), - \
39                   cos(T[n])*sin(alfa[n]), r[n]*sin(T[n])]
40             w3 = [0, sin(alfa[n]), cos(alfa[n]), d[n]]
41             w4 = [0, 0, 0, 1]
42             H[numer] = [w1, w2, w3, w4]
43         return H
44
45     def stopnie_na_rad(self,alfa,theta):
46         T = {} #słownik dla kątów theta
47         for x in range (1,6): #zamiana kątów alfa na radiany
48             alfa[x] = (alfa[x]/180)*pi
49             for x in range (4):
50                 theta[x] = (theta[x]/180)*pi
51 #zamiana kątów theta na radiany
52             for x in range(1,5): #zamiana listy na słownik
53                 T[x] = theta[x-1]
54 #kąty theta sa teraz w słowniku od 1 do 4
55         return alfa, T
56
57 class Prosta(Kinematyka):
58
59     def oblicz(self,theta):
60         theta = list(theta)
61 #zamiana krótki na liste aby mozna było przekształcać
62         alfa = self.alfa
63         alfa, T = self.stopnie_na_rad(alfa,theta)
64         H = self.macierze_DH(T,alfa) #funkcja DH
65         H_mult = self.mnozenie(H)
66         H0_5 = H_mult[3]
67         x = round(H0_5[0,-1], 1)
68         y = round(H0_5[1,-1], 1)
69         z = round(H0_5[2,-1], 1)
70         polozenie = (x,y,z)
71         return polozenie
72
73 class Odwrotna(Kinematyka):
74
75     def zaleznosci_x(self,polozenie):
76         x,y,z = polozenie

```

Koncepcja robota obdarzonego cechami sztucznej inteligencji rozwiązującego układankę
z losowo zmieszanyimi puzzlami.

```

77         Theta = {} #słownik kątów Theta
78         R = {}
79
80         if x == 0.0:
81             Theta[1] = pi/2
82             R[1] = y
83         if x < 0:
84             Theta[1] = pi/2 + np.arctan(y/x) + pi/2
85             R[1] = sqrt( pow(x,2) + pow(y,2) )
86         if x > 0:
87             Theta[1] = np.arctan(y/x)
88             R[1] = sqrt( pow(x,2) + pow(y,2) )
89         return Theta, R
90
91     def zaleznosci_z(self,z,a,R,Fi,T):
92         if z == 0.0:
93             T[2] = np.arccos( ( pow(R[1],2) + pow(a[2],2)-\
94             pow(a[3],2) ) / (2*R[1]*a[2]) )
95         if z >0:
96             T[2] = Fi[2] + Fi[1]
97         if z <0:
98             T[2] = Fi[2] - abs(Fi[1])
99         return T
100
101    def oblicz(self,polozenie):
102        x, y, z = polozenie
103        T, R = self.zaleznosci_x(polozenie)
104        a = self.a
105        z = z-a[1]
106        Fi = {}
107        R[2] = sqrt(pow(z,2) + pow(R[1],2))
108        Fi[1] = np.arctan(z/R[1])
109        Fi[2] = np.arccos( ( pow(a[2],2) + pow(R[2],2) - \
110          pow(a[3],2) ) / ( 2*a[2]*R[2] ) )
111        Fi[3] = np.arccos( (pow(a[2],2) + pow(a[3],2) - \
112          pow(R[2],2)) / ( 2*a[2]*a[3] ) )
113        T = self.zaleznosci_z(z,a,R,Fi,T)
114        T[3] = -(pi-Fi[3])
115        T[4] = -T[3]-T[2]
116
117        for x in range(1,5): #zamiana kątów Theta na stopnie
118            T[x] = (T[x]/pi)*180
119        T = (T[1],T[2],T[3],T[4]) #zamiana słownika na krotkę
120        return T
121
122    class Predkosc(Kinematyka):
123
124        def jakobian(self, H_mult):
125            H0_1 = H_mult[1]
126            H0_2 = H_mult[2]
127            H0_3 = H_mult[3]
128
129            R0_1 = H0_1[:,3]
130            R0_2 = H0_2[:,3]
131
132            d0_1 = matrix([[H0_1[0,-1]], [H0_1[1,-1]], [H0_1[2,-1]]])
133            d0_2 = matrix([[H0_2[0,-1]], [H0_2[1,-1]], [H0_2[2,-1]]])
134            d0_3 = matrix([[H0_3[0,-1]], [H0_3[1,-1]], [H0_3[2,-1]]])
135
136            wektor001 = [0,0,1]
137            d = transpose(d0_3)
138            J1 = transpose(cross(wektor001, d))
139            J2 = transpose(cross(dot(R0_1,wektor001), \
140              transpose(d0_3-d0_1)))
141            J3 = transpose(cross(dot(R0_2,wektor001), \
142              transpose(d0_3-d0_2)))
143            return J1,J2,J3
144
145        def kierunek(self,P1,P2):
146
147            P1 = np.array(P1)
148            P2 = np.array(P2)
149            kierunek = np.subtract(P2,P1)
150            x1,y,z = P1
151            x2,y,z = P2
152            if x1 == 0:
153                if x2 >x1:
154                    x1=0.5
155                else:
156                    x1=-0.5

```

Koncepcja robota obdarzonego cechami sztucznej inteligencji rozwiązującego układankę
z losowo zmieszanyimi puzzlami.

```
157     P1 = (x1,y,z)
158     for i in range(3):
159         if np.any(kierunek[i]) != 0:
160             kierunek[i] = 8
161             if i == 0:
162                 kierunek[i] = 8
163     return kierunek
164
165 def oblicz(self,Poz_akt,P1,P2):
166     P1 = list(P1)
167     P2 = list(P2)
168     Poz_akt = list(Poz_akt)
169     V_3 = self.kierunek(P1,P2)
170
171     alfa = self.alfa
172     alfa, T = self.stopnie_na_rad(alfa,Poz_akt)
173     H = self.macierze_DH(T,alfa)
174     H_mult = self.mnozenie(H)
175
176     J1, J2, J3 = self.jakobian(H_mult)
177     Jakobian = hstack((J1,J2,J3))
178     Jakobian_odw = linalg.inv(Jakobian)
179     #Słownik dla prędkości członu 3
180
181     T_dot1 = round(abs(Jakobian_odw[0,0]*V_3[0]+\ \
182     Jakobian_odw[0,1]*V_3[1]+Jakobian_odw[0,2]*V_3[2]),6)
183     T_dot2 = round(abs(Jakobian_odw[1,0]*V_3[0]+\ \
184     Jakobian_odw[1,1]*V_3[1]+Jakobian_odw[1,2]*V_3[2]),6)
185     T_dot3 = round(abs(Jakobian_odw[2,0]*V_3[0]+\ \
186     Jakobian_odw[2,1]*V_3[1]+Jakobian_odw[2,2]*V_3[2]),6)
187     T_dot = [T_dot1,T_dot2,T_dot3]
188     return T_dot #predkosci serw
189
190 if __name__ == '__main__':
191
192
193     kin = Kinematyka()
194     predkosc = Predkosc()
195     prosta = Prosta()
196     odwrotna = Odwrotna()
197     poz = {}
198     poz[0] = (90,20,-30,0)
199
200     V = predkosc.oblicz(poz[0],"y")
201     print(V)
202
203     x,y,z = prosta.oblicz(poz[0])
204     pozycja = (x,y,z)
205     print("x,y,z: ",pozycja)
206
207     k = odwrotna.oblicz(pozycja)
208     print("katy: ",k)
```

Koncepcja robota obdarzonego cechami sztucznej inteligencji rozwiązującego układankę z losowo zmieszonymi puzzlami.

Załącznik 4. Plik PCA9685.py

```

1  #!/usr/bin/python3
2  import time
3  import math
4  import smbus
5  # =====
6  # Raspi PCA9685 16-Channel PWM Servo Driver
7  # =====
8  class PCA9685:
9      # Registers/etc.
10     __SUBADR1          = 0x02
11     __SUBADR2          = 0x03
12     __SUBADR3          = 0x04
13     __MODE1             = 0x00
14     __PRESCALE          = 0xFE
15     __LED0_ON_L         = 0x06
16     __LED0_ON_H         = 0x07
17     __LED0_OFF_L        = 0x08
18     __LED0_OFF_H        = 0x09
19     __ALLLED_ON_L       = 0xFA
20     __ALLLED_ON_H       = 0xFB
21     __ALLLED_OFF_L      = 0xFC
22     __ALLLED_OFF_H      = 0xFD
23
24     def __init__(self, address=0x40, debug=False):
25         self.bus = smbus.SMBus(1)
26         self.address = address
27         self.debug = debug
28         self.write(self.__MODE1, 0x00)
29
30     def write(self, reg, value):
31         "Zapisuje 8-bitowa wartosc do adresu"
32         self.bus.write_byte_data(self.address, reg, value)
33
34     def read(self, reg):
35         "Odczytuje bajt od I2C"
36         result = self.bus.read_byte_data(self.address, reg)
37         return result
38
39     def setPWMFreq(self, freq):
40         "Ustawia czestotliwosc PWM"
41         prescaleval = 25000000.0          # 25MHz
42         prescaleval /= 4096.0            # 12-bit
43         prescaleval /= float(freq)
44         prescaleval -= 1.0
45         prescale = math.floor(prescaleval + 0.5)
46         oldmode = self.read(self.__MODE1);
47         newmode = (oldmode & 0x7F) | 0x10
48         self.write(self.__MODE1, newmode)
49         self.write(self.__PRESCALE, int(math.floor(prescale)))
50         self.write(self.__MODE1, oldmode)
51         time.sleep(0.005)
52         self.write(self.__MODE1, oldmode | 0x80)
53
54     def setPWM(self, channel, on, off):
55         "Wybiera kanał PWM"
56         self.write(self.__LED0_ON_L+4*channel, on & 0xFF)
57         self.write(self.__LED0_ON_H+4*channel, on >> 8)
58         self.write(self.__LED0_OFF_L+4*channel, off & 0xFF)
59         self.write(self.__LED0_OFF_H+4*channel, off >> 8)
60
61     def setServoPulse(self, channel, pulse):
62         "Ustawia wypełnienie PWM, czestotliwosc musi wynosic 50HZ"
63         pulse = pulse*4096/20000          #PWM czestotliwosc 50HZ, the okres 20ms
64         self.setPWM(channel, 0, int(pulse))
65
66     if __name__ == '__main__':
67         pwm = PCA9685(0x40, debug=False)
68         pwm.setPWMFreq(50)
69         a= [1490, 800, 600, 2500]
70         pwm.setServoPulse(0,a[0])
71         time.sleep(0.02)
72         pwm.setServoPulse(1,a[1])
73         time.sleep(0.02)
74         pwm.setServoPulse(2,a[2])
75         time.sleep(0.02)
76         pwm.setServoPulse(3,a[3])

```

Koncepcja robota obdarzonego cechami sztucznej inteligencji rozwiązującego układankę
z losowo zmieszanyimi puzzlami.

Załącznik 5. Plik Serwo.py

```
1  from Ruch.Kinematyka import Predkosc, Prosta
2  from Ruch.PCA9685 import PCA9685
3  #from Kinematyka import Predkosc, Prosta
4  #from PCA9685 import PCA9685
5  import time
6  import numpy as np
7
8
9  class Serwo:
10     """
11         Klasa zawierająca funkcje związane z ruchem manipulatora.
12     """
13     def __init__(self, poz_1, V= [0.3, 0.3, 0.3, 0.3]):
14         self.poz_1 = poz_1
15         #kolejność numer:(gora,dol)
16         self.W_zakres = { 1: (2460,580), 2: (500,2400), 3: (420,2450), \
17             4: (2420,600) }
18         self.A_180 = (180,0)
19         self.A_180_inv = (-180,0)
20         self.A_90 = (90, -90)
21         self.V = V
22         self.V2 = V
23         self.pwm_start()
24
25     def pwm_start(self):
26         """
27             Inicjalizacja klasy sterownika PCA9685
28         """
29         self.pwm = PCA9685(0x40, debug=False)
30         self.pwm.setPWMFreq(50)
31
32     def oblicz_wypełnienie(self, A):
33         """
34             Funkcja oblicza wartość wypełnienia "W" PWM
35             dla podanego kąta alfa "A" w stopniach
36         """
37         W = [0,0,0,0]
38         W_zakres = self.W_zakres
39         for i in range(0,4):
40             if i == 0:
41                 A_zakres = self.A_180
42             elif i == 2:
43                 A_zakres = self.A_180_inv
44             else:
45                 A_zakres = self.A_90
46
47             A_max = A_zakres[0]
48             A_min = A_zakres[1]
49             W_max = W_zakres[i+1][0]
50             W_min = W_zakres[i+1][1]
51
52             wyp = ((W_max - W_min) / (A_max - A_min)) * (A[i] - A_min) \
53             + W_min
54             W[i] = wyp
55         return W
56
57     def ustaw_predkosc(self, V):
58         """
59             Możliwość zmiany domyślnej prędkości serwonapędów
60         """
61         self.V = [V,V,V,V]
62
63     def oblicz_predkosc_poz1(self, poz2):
64         """
65             Obliczenie i ustawienie prędkości serwonapedów dla poprzedniej pozycji
66             za pomocą jacobiana, w celu wykonania ruchu liniowego
67         """
68         poz1= self.poz_1
69         predkosc = Predkosc()
70         prosta = Prosta()
71         P1 = prosta.oblicz(poz1)
72         prosta = Prosta()
73         P2 = prosta.oblicz(poz2)
74         V = predkosc.oblicz(poz1,P1,P2)
75         self.V = V
76         k1,k2,k3,k4 = poz1
```

Koncepcja robota obdarzonego cechami sztucznej inteligencji rozwiązującego układankę
z losowo zmieszanyimi puzzlami.

```
77         k1 = round(k1,1)
78         k2 = round(k2,1)
79         k3 = round(k3,1)
80         k4 = round(k4,1)
81         poz11 = (k1,k2,k3,k4)
82         print("Pozycja POPRZEDNIA")
83         print("Odwrotna", poz11)
84         print("Predkosc: ",V)
85         print("-----")
86         return V
87
88     def ustaw_pwm(self,P):
89         """
90             Funkcja ustawiajaca pozycje serwonapędów od wypełnienia pozycji P
91         """
92         pwm = self.pwm
93         P=self.oblicz_wypelnienie(P)
94         pwm.setServoPulse(0, P[0])
95         pwm.setServoPulse(1, P[1])
96         pwm.setServoPulse(2, P[2])
97         pwm.setServoPulse(3, P[3])
98         time.sleep(0.001)
99
100    def zmiana_wypelnienia(self,P1,P2,V):
101        """
102            Funkcja prędkości. Wykonywana jest inkrementacja lub dekrementacja
103            stopni o określona wartość prędkości V, dla każdego z serw.
104        """
105        while(1):
106            for x in range(3):
107                if P1[x] < P2[x]:
108                    if x == 2:
109                        V[x] += V[x]*0.03
110                    elif x == 1:
111                        V[x] += V[x]*0.02
112                        P1[x] += V[x]
113                        diff = P2[x] - P1[x]
114                        if abs(diff) < abs(V[x]):
115                            P1[x] += diff
116                if P1[x] > P2[x]:
117                    if x == 1:
118                        V[x] += V[x]*0.02
119                    elif x == 2:
120                        V[x] += V[x]*0.01
121                        P1[x] -= V[x]
122                        diff = P1[x] - P2[x]
123                        if abs(diff) < abs(V[x]):
124                            P1[x] -= diff
125                P1[3] = -P1[2]-P1[1]-5
126                self.ustaw_pwm(P1)
127                if np.all(P1[:3] == P2[:3]):
128                    break
129
130    def ustaw_poz(self,pozycje):
131        """
132            Funkcja ustawia manipulator na określona pozycję
133        """
134        V = self.V
135        P1 = list(self.poz_1)
136        P2 = list(pozycje)
137        self.zmiana_wypelnienia(P1,P2,V)
138
139        # -- Przypisanie ostatniej pozycji jako początkowej -- #
140        self.poz_1 = pozycje
141
142    class Pozycje:
143        """
144            Klasa do przechowywania pozycji manipulatora wraz z funkcjami do ich
145            dodawania i kasowania. Pozycje przechowywane są w słowniku "poz".
146        """
147        def __init__(self):
148            # -- Przechowywane pozycje xyz -- #
149            self.poz = {}
150            self.poz["start"] = (0,6,10)
151
152        def dodaj_puzzle(self,puzzle):
153            for x in puzzle:
154                self.poz[x] = puzzle[x]
```

Koncepcja robota obdarzonego cechami sztucznej inteligencji rozwiązującego układankę z losowo zmieszanymi puzzlami.

```
157 def puzzle(self, poz, z):
158     poz2 = (poz[0], poz[1], z)
159     return poz2
160
161     def dodaj(self, nowa_poz):
162         # -- Automatyczne przypisanie numeru według długości słownika -- #
163         numer = len(self.poz)
164         self.poz[numer] = nowa_poz
165
166     def kasuj(self, nr_poz):
167         usun = self.poz
168         # -- None - Jeżeli nie ma takiej pozycji to nie wyrzuci błędu -- #
169         usun.pop(nr_poz, None)
170
171 if __name__ == '__main__':
172     poz = Pozycje()
173     serwo = Serwo(poz.poz["start"])
174     serwo.zmiana_wypelnienia([90,55,-100,65],[0,80,-100,65],[0.5,0.5,0.5,0.5])
175     time.sleep(2)
176     serwo.zmiana_wypelnienia([0,55,-100,65],[90,80,-100,65],[0.5,0.5,0.5,0.5])
```

Koncepcja robota obdarzonego cechami sztucznej inteligencji rozwiązującego układankę
z losowo zmieszanyimi puzzlami.

Załącznik 6. Plik main.py

```
1  #!/usr/bin/python3
2  import cv2
3  from Ruch.Serwo import Serwo, Pozycje,
4  from Ruch.Kinematyka import Prosta, Odwrotna, Predkosc
5  from Obraz.Kamera import Kamera
6  import Obraz.Rozpoznawanie as rozpoznawanie
7  from Puzzle import Puzzle
8  from time import sleep
9  import numpy as np
10
11 class Start:
12     def __init__(self,model,startowa="start"):
13         self.serwo(startowa)
14         self.model = model
15         self.oczekiwanie()
16         self.kamera()
17         self.puzzle()
18
19     def serwo(self,startowa):
20         '''
21             Funkcja inicjuje klasę Serwo od pozycji początkowej
22             zapisanej w klasie Pozycje.
23         '''
24         # ----- Inicjalizacja klas ----- ##
25         pozycje = Pozycje()
26         odwrotna = Odwrotna()
27
28         # ----- Obliczenie katów dla startowej pozycji ----- ##
29         T = odwrotna.oblicz(pozycje.poz[startowa])
30         p = 0.3 #predkosc
31         self.p = p
32         self.serwo = Serwo(T,[p,p,p,p])
33
34     def oczekiwanie(self):
35         kamera = Kamera()
36         kamera.oczekiwanie()
37
38     def kamera(self):
39         '''
40             Funkcja inicjuje klasę Kamera, która wykonuje zdjęcie. Następnie
41             wykrywa pozycje poszczególnych pól i przekazuje je klasie Pozycje.
42             Na koniec uruchamiana jest biblioteka rozpoznawania cyfr i zapisywana
43             jest ich sekwencja.
44         '''
45         # ----- Inicjalizacja klas ----- ##
46         kamera = Kamera()
47         pozycje = Pozycje()
48
49         # ----- Wykonanie zdjęcia oraz zapisanie pozycji ----- ##
50         kamera.zdjecie("puzzle.png")
51         puzzle_pozycje = kamera.szukaj()
52         pozycje.dodaj_puzzle(puzzle_pozycje)
53         print("Pozycje ze słownika: ",pozycje.poz)
54         self.pozycje = pozycje
55
56         # ----- Rozpoznanie cyfr i zapisanie sekwencji ----- ##
57         model, classes = self.model
58         self.kolejnosc = rozpoznawanie.start(model,classes)
59
60     def ruch(self,poz,liniowy=0):
61         '''
62             Funkcja pobiera informację o położeniu xyz z klasy Pozycje,
63             oblicza kąty Theta za pomocą klasy Odwrotna. Umożliwia ruch domyślny
64             (Jedna prędkość dla wszystkich złącz) oraz ruch liniowy
65             (Prędkości złącz obliczone przez klasę Predkosc).
66         '''
67         # ----- Inicjalizacja klas ----- ##
68         odwrotna = Odwrotna()
69         predkosc = Predkosc()
70         prosta = Prosta()
71         serwo = self.serwo
72         # ----- Obliczenie katów dla docelowej pozycji ----- ##
73         x,y,z = poz
74         #pominiecie punktu x=0
75         if x == 0:
76             x = 0.1
```

Koncepcja robota obdarzonego cechami sztucznej inteligencji rozwiązującego układankę

z losowo zmieszanyimi puzzlami.

```
77         poz = (x,y,z)
78         T2 = odwrotna.oblicz(poz)
79         pozycja = prosta.oblicz(T2)
80         k1,k2,k3,k4 = T2
81         k1 = round(k1,1)
82         k2 = round(k2,1)
83         k3 = round(k3,1)
84         k4 = round(k4,1)
85         T22 = (k1,k2,k3,k4)
86         print("Pozycja DOCELOWA:")
87         print("Odwrotna: ",T22,"Prosta: ",pozycja)
88
89         if liniowy ==0:
90             # ----- Ustawienie serw na obliczone katy, ruch domyslny ----- ###
91             serwo.ustaw_predkosc(self.p)
92             serwo.ustaw_poz(T2)
93
94         if liniowy == 1:
95             # ----- Ustawienie serw na obliczone katy, ruch liniowy ----- ###
96             poz2 = T2
97             serwo.oblicz_predkosc_pozl(poz2)
98             serwo.ustaw_poz(T2)
99
100        def puzzle(self):
101            kolejnosc = self.kolejnosc
102            pozycje = self.pozycje
103            puzzle = Puzzle(kolejnosc)
104            zero= kolejnosc.find("0")
105
106            print('Start: ', puzzle.start)
107            print('Cel: ', puzzle.cel)
108            sciezka = puzzle.szukaj(puzzle.start)
109            print("Liczba ruchow= ", (len(sciezka)))
110            poz = [0,0,0,0]
111
112            for etap in sciezka:
113                kierunek = etap[2]
114                x,y,z = pozycje.poz[zero]
115                zero = etap[1].find("0")
116
117                if kierunek == "lewo":
118                    poz[0] = (x+3,y,8)
119                    poz[1] = (x+3,y,3.6)
120                    poz[2] = (x-1,y,3.6)
121                    poz[3] = (x-1,y,8)
122                elif kierunek == "prawo":
123                    poz[0] = (x-3,y,8)
124                    poz[1] = (x-3,y,3.6)
125                    poz[2] = (x+1,y,3.6)
126                    poz[3] = (x+1,y,8)
127                elif kierunek == "gora":
128                    poz[0] = (x,y-3,8)
129                    poz[1] = (x,y-3,3.6)
130                    poz[2] = (x,y+1,3.6)
131                    poz[3] = (x,y+1,8)
132                elif kierunek == "dol":
133                    poz[0] = (x,y+3,8)
134                    poz[1] = (x,y+3,3.6)
135                    poz[2] = (x,y-1,3.6)
136                    poz[3] = (x,y-1,8)
137                else: continue
138                #wykonanie przesuniecia
139                self.ruch(poz[0],liniowy=1)
140                sleep(0.5)
141                self.ruch(poz[1],liniowy=1)
142                sleep(0.5)
143                self.ruch(poz[2],liniowy=1)
144                sleep(0.5)
145                self.ruch(poz[3],liniowy=1)
146                sleep(0.5)
147                #powrot do pozycji startowej
148                self.ruch(pozycje.poz["start"],liniowy=0)
149        if __name__ == '__main__':
150            model = rozpoznawanie.model()
151            while True:
152                Start(model)
```

Koncepcja robota obdarzonego cechami sztucznej inteligencji rozwiązującego układankę
z losowo zmieszanyimi puzzlami.

Załącznik 7. Plik Puzzle.py

```

1   import queue as Q
2
3   class Puzzle:
4       """
5       Inicjalizacja klasy Puzzle. Klasa pobiera konfiguracje startowa
6       oraz tworzy zmienne pomocnicze.
7       """
8       def __init__(self,start = "362107548"):
9           self.start = start
10          self.cel = "123456780"
11          self.legalneRuchy = {
12              0: (1,3),
13              1: (-1,3,1),
14              2: (-1,3),
15              3: (-3,1,3),
16              4: (-3,-1,1,3),
17              5: (-3,-1,3),
18              6: (-3,1),
19              7: (-3,-1,1),
20              8: (-3,-1)}
21          self.tabla_odl = self.odleglosc()
22
23      """
24      Funkcja sprawdza jakie ruchy mozna wykonac dla aktualnej pozycji
25      pustego miejsca.
26      """
27      def znajdz_ruchy(self, st):
28          zero = st.find("0")
29          return zero, self.legalneRuchy[zero]
30
31      """
32      Tabela pomocnicza z odleglosciami wedlug metryki miejskiej.
33      """
34      def odleglosc(self):
35          tabela_odl = {};
36          for aa in range(9):
37              for bb in range(9) :
38                  arow = aa//3; acol=aa%3
39                  brow = bb//3; bcol=bb%3
40                  tabela_odl[(aa,bb)] = abs(arow-brow)+abs(acol-bcol)
41          return tabela_odl
42
43      """
44      Suma kosztu dla wszystkich pol z cyframi.
45      """
46      def dystans_manhattan(self,st):
47          koszt = 0
48          for pole in st:
49              if pole != "0":
50                  docelowe = self.cel.find(pole)
51                  aktualne = st.find(pole)
52                  koszt += self.tabla_odl[(aktualne,docelowe)]
53          return koszt
54
55      Zapis ruchu do zmiennej robot, ktory zostanie wykorzystany
56      w pliku main.py do odczytania sciezki.
57      """
58      def sprawdz_robot(self,r):
59          if r == 1:
60              robot = "lewo"
61          if r == -1:
62              robot = "prawo"
63          if r == 3:
64              robot = "gora"
65          if r == -3:
66              robot = "dol"
67          return robot
68
69      Wykonanie ruchu, czyli zamiana miejscami dwoch cyfr.
70      """
71      def wykonaj_ruch(self, st, zero, r):
72          stan = list(st)
73          stan[zero + r], stan[zero] = stan[zero], stan[zero + r]
74          return "".join(stan)
75
76      Glowna funkcja algorytmu A*. Uruchamia przeszukiwanie stanow oraz umieszcza

```

Koncepcja robota obdarzonego cechami sztucznej inteligencji rozwiązującego układankę z losowo zmieszonymi puzzlami.

```
77     je w kolejce priorytetowej, gdzie sa sortowane od najmniejszego kosztu.
78     Ostateczny koszt to funkcja f(x), ktora sumuje koszt związanego z odleglosciami
79     oraz liczbe dotychczasowych ruchow.
80     '''
81     def szukaj(self, st):
82         #zmienne pomocnicze
83         odwiedzone = {}
84         kolejka = Q.PriorityQueue()
85         ratio = 5
86         origCost = self.dystans_manhattan(st)
87         robot= ""
88         #koszt startowy
89         orig = (origCost,0,st,robot,None)
90         kolejka.put(orig)
91         odwiedzone[orig] = True
92         rozwiniete = 0
93         rozwiazanie = None
94         while kolejka and not rozwiazanie :
95             #pobierz pierwszy stan z kolejki
96             parent = kolejka.get()
97             rozwiniete += 1 #rozwin go
98             (parCost, parMoves, parBoard, robot, grandparent) = parent
99             zero, ruchy = self.znajdz_ruchy(parBoard)
100            for r in ruchy :
101                childBoard = self.wykonaj_ruch(parBoard,zero,r)
102                robot = self.sprawdz_robot(r) #zapisz jaki to ruch
103                if odwiedzone.get(childBoard) : continue
104                odwiedzone[childBoard] = True
105                childMoves = parMoves+1
106                #sumowanie kosztu
107                childCost = self.dystans_manhattan(childBoard) + childMoves
108                child = (childCost, childMoves, childBoard, robot, parent)
109                kolejka.put(child)
110                if childBoard == self.cel:
111                    rozwiazanie = child
112        if rozwiazanie :
113            print("Rozwinięto %s stanów. W kolejce znajduje się jeszcze %s \ stanów" % (rozwiniete, kolejka.qsize()))
114            sciezka = []
115            while rozwiazanie :
116                # Z informacji (childCost, childMoves, childBoard, robot, parent)
117                # dodaj pozycje 1-3 do sciezki
118                sciezka.append(rozwiazanie[1:4])
119                rozwiazanie = rozwiazanie[4]      # przejdź wyżej do rodzica
120                sciezka.reverse()
121                return sciezka # (numer kroku, 'stan', 'jaki ruch')
122            else :
123                return [] #blad
124
125 #debug pliku
126 if __name__ == '__main__':
127     puz = Puzzle()
128     print('Start: ', puzzle.start)
129     print('Cel: ', puzzle.cel)
130     sciezka = puz.szukaj(puz.start)
131     print("Liczba ruchów= ", (len(sciezka)))
132     for etap in sciezka : print(etap)
```

Koncepcja robota obdarzonego cechami sztucznej inteligencji rozwiązującego układankę z losowo zmieszonymi puzzlami.

Załącznik 8. Instalacja OpenCV 4.5.0

Instalacja bibliotek zależnych:

```
$ sudo apt-get install build-essential cmake git unzip pkg-config
$ sudo apt-get install libjpeg-dev libpng-dev
$ sudo apt-get install libavcodec-dev libavformat-dev libswscale-dev
$ sudo apt-get install libgtk2.0-dev libcanberra-gtk* libgtk-3-dev
$ sudo apt-get install libgstreamer1.0-dev libgstreamer-plugins-base1.0-dev
$ sudo apt-get install libxvidcore-dev libx264-dev
$ sudo apt-get install python3-dev python3-numpy python3-pip
$ sudo apt-get install libtbb2 libtbb-dev libdc1394-22-dev
$ sudo apt-get install libv4l-dev v4l-utils
$ sudo apt-get install libopenblas-dev libatlas-base-dev libblas-dev
$ sudo apt-get install liblapack-dev gfortran lib hdf5-dev
$ sudo apt-get install libprotobuf-dev libgoogle-glog-dev libgflags-dev
$ sudo apt-get install protobuf-compiler
```

Pobranie plików:

```
#pobierz pliki
$ cd ~
$ wget -O opencv.zip https://github.com/opencv/opencv/archive/4.5.0.zip
$ wget -O opencv_contrib.zip https://github.com/opencv/opencv_contrib/archive/4.5.0.zip
# rozpakuj
$ unzip opencv.zip
$ unzip opencv_contrib.zip
$ mv opencv-4.5.0 opencv
$ mv opencv_contrib-4.5.0 opencv_contrib
# usun archiwa
$ rm opencv.zip
$ rm opencv_contrib.zip
```

Kompilacja –wyznaczenie parametrów:

```
cmake -D CMAKE_BUILD_TYPE=RELEASE \
-D CMAKE_INSTALL_PREFIX=/usr/local \
-D OPENCV_EXTRA_MODULES_PATH=~/opencv_contrib/modules \
-D ENABLE_NEON=ON \
-D WITH_OPENMP=ON \
-D BUILD_TIFF=ON \
-D WITH_FFMPEG=ON \
-D WITH_TBB=ON \
-D BUILD_TBB=ON \
-D BUILD_TESTS=OFF \
-D WITH_EIGEN=OFF \
-D WITH_GSTREAMER=ON \
-D WITH_V4L=ON \
-D WITH_LIBV4L=ON \
```

Koncepcja robota obdarzonego cechami sztucznej inteligencji rozwiązującego układankę z losowo zmieszanymi puzzlami.

```
-D WITH_VTK=OFF \
-D WITH_QT=OFF \
-D OPENCV_ENABLE_NONFREE=ON \
-D INSTALL_C_EXAMPLES=OFF \
-D INSTALL_PYTHON_EXAMPLES=OFF \
-D BUILD_NEW_PYTHON_SUPPORT=ON \
-D BUILD_opencv_python3=TRUE \
-D OPENCV_GENERATE_PKGCONFIG=ON \
-D BUILD_EXAMPLES=OFF ..
```

Kompilacja właściwa (1,5h):

```
$ make -j4
```

Instalacja:

```
$ sudo make install
$ sudo ldconfig
$ make clean
$ sudo apt-get update
```

Koncepcja robota obdarzonego cechami sztucznej inteligencji rozwiązującego układankę z losowo zmieszonymi puzzlami.

Załącznik 9. Instalacja Tensorflow 2.4.0

```
# instalacja pip and pip3
$ sudo apt-get install python-pip python3-pip
$ sudo pip uninstall tensorflow
$ sudo pip3 uninstall tensorflow
# biblioteki zależne
$ sudo apt-get install gfortran
$ sudo apt-get install libhdf5-dev libc-ares-dev libeigen3-dev
$ sudo apt-get install libatlas-base-dev libopenblas-dev libblas-dev
$ sudo apt-get install liblapack-dev
# aktualizacja setuptools
$ sudo -H pip3 install --upgrade setuptools
$ sudo -H pip3 install pybind11
$ sudo -H pip3 install Cython==0.29.21
$ sudo -H pip3 install h5py==2.10.0
$ pip3 install gdown
$ sudo cp ~/.local/bin/gdown /usr/local/bin/gdown
# pobierz wheel
$ gdown
https://drive.google.com/uc?id=1x2nxbi8rLvF7WzYkuwt9SOuDvsOUytjE
# instalacja TensorFlow 2.4.0 (± 63 min @1950 MHz)
$ sudo -H pip3 install tensorflow-2.4.0-cp37-cp37m-linux_aarch64.whl
```

Biblioteki zależne:

```
$ sudo apt-get install wget curl libhdf5-dev libc-ares-dev libeigen3-
dev
$ sudo apt-get install libatomic1 libatlas-base-dev zip unzip
$ gdown
https://drive.google.com/uc?id=1oquDH2xcrhiXYU0eXTU9pw6CnCLreByr
$ sudo tar -C /usr/local -xzf libtensorflow_cp37_64OS_2_4_0.tar.gz
```

Instalacja Bazel:

```
$ sudo apt-get install build-essential zip unzip curl
# install Java
$ sudo apt-get install openjdk-11-jdk
$ wget
https://github.com/bazelbuild/bazel/releases/download/3.1.0/bazel-
3.1.0-dist.zip
$ unzip -d bazel bazel-3.1.0-dist.zip
$ cd bazel
$ nano scripts/bootstrap/compile.sh -c
```

Kompilacja Bazel:

```
$ env EXTRA_BAZEL_ARGS="--host_javabase=@local_jdk//:jdk" bash
./compile.sh
# copy the binary
$ sudo cp output/bazel /usr/local/bin/bazel
# clean up
```

Koncepcja robota obdarzonego cechami sztucznej inteligencji rozwiązującego układankę z losowo zmieszonymi puzzlami.

```
$ cd ~  
$ rm bazel-3.1.0-dist.zip  
# if you have a copied bazel to /usr/local/bin you may also  
# delete the whole bazel directory, freeing another 500 MByte  
$ sudo rm -rf bazel
```

Instalacja Tensorflow:

```
$ sudo apt-get install build-essential make cmake wget unzip  
$ sudo apt-get install libhdf5-dev libc-ares-dev libeigen3-dev  
$ sudo apt-get install libatlas-base-dev libopenblas-dev libblas-dev  
$ sudo apt-get install gfortran liblapack-dev  
$ sudo pip3 install keras_applications --no-deps  
$ sudo pip3 install keras_preprocessing --no-deps  
$ sudo pip3 install -U --user six wheel mock  
$ wget -O tensorflow.zip  
https://github.com/tensorflow/tensorflow/archive/v2.4.0.zip  
$ unzip tensorflow.zip  
$ mv tensorflow-2.4.0 tensorflow  
$ cd tensorflow  
$ ./configure
```

Opis konfiguracji:

```
pi@raspberrypi:~/tensorflow $ ./configure  
You have bazel 3.1.0- (non-git) installed.  
Please specify the location of python. [Default is /usr/bin/python3]: <enter>  
  
Found possible Python library paths:  
/usr/lib/python3/dist-packages  
/usr/local/lib/python3.7/dist-packages  
Please input the desired Python library path to use. Default is [/usr/lib/python3/dist-packages] <enter>  
  
Do you wish to build TensorFlow with OpenCL SYCL support? [y/N]: n  
No OpenCL SYCL support will be enabled for TensorFlow.  
  
Do you wish to build TensorFlow with ROCm support? [y/N]: n  
No ROCm support will be enabled for TensorFlow.  
  
Do you wish to build TensorFlow with CUDA support? [y/N]: n  
No CUDA support will be enabled for TensorFlow.  
  
Do you wish to download a fresh release of clang? (Experimental) [y/N]: n  
Clang will not be downloaded.  
  
Please specify optimization flags to use during compilation when bazel option "--config=opt" is specified [Default is  
-march=native -Wno-sign-compare]: <enter>  
  
Would you like to interactively configure ./WORKSPACE for Android builds? [y/N]: n  
Not configuring the WORKSPACE for Android builds.  
  
Preconfigured Bazel build configs. You can use any of the below by adding "--config=<>" to your build command. See  
.bazelrc for more details.  
--config=mkl      # Build with MKL support.  
--config=monolithic  # Config for mostly static monolithic build.  
--config=ngraph    # Build with Intel nGraph support.  
--config=numa      # Build with NUMA support.  
--config=dynamic_kernels # (Experimental) Build kernels into separate shared objects.  
--config=v2        # Build TensorFlow 2.x instead of 1.x. <-->  
Preconfigured Bazel build configs to DISABLE default on features:  
--config=noaws      # Disable AWS S3 filesystem support.  
--config=nogcp       # Disable GCP support.  
--config=nohdfs     # Disable HDFS support.  
--config=nonccl      # Disable NVIDIA NCCL support.  
Configuration finished
```

Koncepcja robota obdarzonego cechami sztucznej inteligencji rozwiązującego układankę z losowo zmieszonymi puzzlami.

Kompilacja Tensorflow ~40h:

```
$ sudo bazel clean
$ sudo bazel --host_jvm_args=-Xmx1624m build \
    --config=opt \
    --config=noaws \
    --config=nogcp \
    --config=nohdfs \
    --config=nonccl \
    --config=monolithic \
    --config=v2 \
    --local_cpu_resources=1 \
    --define=tflite_pip_with_flex=true \
    --copt=-ftree-vectorize \
    --copt=-funsafe-math-optimizations \
    --copt=-ftree-loop-vectorize \
    --copt=-fomit-frame-pointer \
    //tensorflow/tools/pip_package:build_pip_package
```

Instalacja wheel:

```
# synthesize the wheel
$ sudo bazel-bin/tensorflow/tools/pip_package/build_pip_package
/tmp/tensorflow_pkg
# install some dependencies (if not already done)
$ sudo apt-get install libatlas3-base libopenblas-dev libopenblas-base
libblas-dev
$ sudo apt-get install libgfortran5 liblapack-dev
$ sudo pip3 install pybind11
$ sudo pip3 install Cython==0.29.21
# install h5py with Cython version 0.29.21
$ sudo pip3 install h5py
# remove old versions if found
$ sudo pip uninstall tensorflow
$ sudo pip3 uninstall tensorflow
# get to the folder where the wheel is located and install tensorflow
$ cd /tmp/tensorflow_pkg
$ sudo pip3 install tensorflow-2.4.0-cp37-cp37m-linux_aarch64.whl
```

Wyczyszczenie:

```
$ sudo rm -rf ~/.cache/bazel
```