



POLITECHNIKA POZNAŃSKA

WYDZIAŁ INFORMATYKI I TELEKOMUNIKACJI

Instytut Informatyki

Praca dyplomowa inżynierska

**SPRZĘTOWY GENERATOR LICZB LOSOWYCH
WYKORZYSTUJĄCY RZUTY KOŚCIĄ**

Julia Samp, 151775

Jakub Kędra, 151790

Wojciech Kot, 151879

Jakub Prusak, 151178

Promotor
dr inż. Jędrzej Potoniec

POZNAŃ 2025

Spis treści

1 Wstęp	1
1.1 Cele i założenia projektu	1
1.2 Decyzje projektowe	2
1.3 Struktura pracy oraz podział obowiązków	3
2 Obecnie stosowane generatory liczb losowych	5
2.1 Obecnie stosowane rozwiązania	5
2.2 Przegląd komercyjnych rozwiązań TRNG	5
2.2.1 Rozwiązania sprzętowe	6
2.2.2 Podobne rozwiązania	7
3 Budowa sprzętowego generatora liczb losowych	8
3.1 Projektowanie robota	8
3.1.1 Planowanie	8
3.1.2 Prototypowanie	9
3.1.3 Ostateczna wersja robota	15
Opis komponentów ostatecznej wersji robota	16
3.2 Dokumentacja techniczna – hardware	20
3.2.1 Ogólne parametry robota	20
3.2.2 Wykorzystane elementy strukturalne	21
3.2.3 Elektronika	23
3.2.4 Zasilanie	23
3.2.5 Podłączenie elektroniki	23
4 Odczytywanie losowego wyniku z kości	25
4.1 Przetwarzanie wstępne obrazów	25
4.1.1 Algorytm	25
4.1.2 Zidentyfikowane trudności i ich rozwiązania	27
4.1.3 Podsumowanie	29
4.2 Opis modelu SI	29
4.2.1 Przygotowanie danych	29
4.3 Architektura modelu	29
4.3.1 Użyte funkcje aktywacji i rodzaje warstw	29
Sieci splotowe (CNN)	29
Warstwa gęsta (Dense)	30
4.3.2 Budowa sieci neuronowej	31
4.3.3 Trenowanie modelu	32
4.3.4 Wyniki	32

4.4	Rozpoznawanie liczb	33
4.4.1	Wczytanie i przetworzenie obrazu	33
4.4.2	Klasyfikacja obrazu	33
4.4.3	Interpretacja wyniku	34
4.5	Podsumowanie	34
5	Komunikacja	35
5.1	Tryb pracy	35
5.1.1	Ciągłe generowanie danych	35
5.1.2	Wykonywanie pojedyńczych rzutów	36
6	Testy	37
6.1	Użyte testy	37
6.1.1	Test chi-kwadrat	37
6.1.2	Test monobitowy	38
6.1.3	Test serii	38
6.1.4	Test długich serii	38
6.1.5	Test pokerowy	38
6.2	Wyniki testów	39
6.2.1	Test chi-kwadrat	39
6.2.2	Test monobitowy	40
6.2.3	Test serii	40
6.2.4	Test długich serii	40
6.2.5	Test pokerowy	40
6.3	Wnioski	41
7	Zakończenie	43
Literatura		45

Rozdział 1

Wstęp

Współczesne systemy komputerowe i technologie wykorzystują liczne algorytmy generowania liczb losowych, które stanowią podstawę w takich dziedzinach jak kryptografia, symulacje czy gry komputerowe. Jednakże większość tych rozwiązań opiera się na generatorach pseudolosowych, które, mimo swojej wydajności, w istocie generują liczby deterministyczne. W praktyce może to prowadzić do problemów z bezpieczeństwem i przewidywalnością w sytuacjach, gdzie prawdziwa losowość jest kluczowa.

Inspiracją do podjęcia tematu budowy sprzętowego generatora liczb losowych (HRNG, z ang. *Hardware Random Number Generator*) było połączenie zainteresowań w dziedzinie techniki oraz pasji do gier planszowych i RPG. Gry te, korzystające od wieków z kości jako narzędzi generowania losowości, stanowią idealny przykład zastosowania fizycznych mechanizmów do uzyskiwania nieprzewidywalnych wyników. Projekt stara się potwierdzić hipotezę, że poprzez zautomatyzowanie procesu rzutu kością możliwe jest uzyskanie prawdziwej losowości mimo deterministycznego charakteru działania maszyny.

1.1 Cele i założenia projektu

Celem pracy jest zbudowanie urządzenia zdolnego do generowania liczb losowych poprzez mechaniczny rzut kością oraz analiza stopnia losowości uzyskanych wyników. Projekt ten zakłada zbudowanie urządzenia, które będzie w stanie rzucać kością w sposób powtarzalny i przewidywalny, jednak z losowym efektem wynikającym z natury procesów fizycznych, takich jak tarcie, turbulencje czy drgania. Dodatkowo praca ma na celu praktyczne zastosowanie zbudowanego generatora jako źródła entropii w systemie Linux.

Realizacja takiego urządzenia pozwoli także w prosty sposób zautomatyzować statystyczne badanie różnych kości do gry pod względem sprawdzania ich potencjalnej nieuczciwości, co było wyjściowym pomysłem, od którego dopiero powstała idea zbudowania sprzętowego generatora liczb prawdziwie losowych (HTRNG, z ang. *Hardware True Random Number Generator*) opartego na rzucie kością.

Realizacja tego projektu pozwoli nie tylko na sprawdzenie skuteczności podejścia opartego o rzuty kośćmi, ale również na pogłębienie wiedzy o granicach między deterministycznymi systemami a losością, stanowiąc wkład w rozwój technologii oraz praktycznych zastosowań w obszarach wymagających prawdziwej losowości.

Projekt robota został opracowany w oparciu o kilka kluczowych założeń, które mają na celu zapewnienie jego funkcjonalności, efektywności oraz użyteczności:

1. Założenie niewielkich rozmiarów i kompaktowości.

2. Założenie modułowości elementów robota.
3. Założenie prostej budowy.
4. Założenie możliwie szybkiego uzyskiwania wyniku.
5. Założenie możliwie niskich kosztów projektu.

Niewielkie rozmiary robota są kluczowe, ponieważ pozwalają na łatwe wykorzystanie urządzenia nawet w codziennym życiu. Kompaktowa konstrukcja oznacza mniejszą wagę i większą mobilność, przez co robota można bez problemu przenosić w różne miejsca. Dzięki temu robot jest bardziej uniwersalny w zastosowaniu i może być używany w różnych celach.

Modułowość pozwala na łatwą rozbudowę i modyfikację robota. Poszczególne komponenty mogą być wymieniane, naprawiane lub ulepszane bez konieczności ingerowania w całą konstrukcję. Dzięki temu projekt jest bardziej elastyczny, co zwiększa jego żywotność i możliwość dostosowania do nowych zastosowań. Modułowość ułatwia również składanie, rozkładanie oraz serwisowanie urządzenia.

Prosta konstrukcja zapewnia, że robot jest łatwy do złożenia, obsługi i serwisowania. Minimalizuje to ryzyko występowania błędów podczas montażu oraz potencjalne problemy eksploatacyjne. Prosta budowa sprawia, że robot jest łatwiejszy w obsłudze dla osób mniej zaawansowanych technicznie, co zwiększa dostępność urządzenia dla szerokiego grona użytkowników, co ma znaczenie w przypadku komercjalizacji rozwiązania. Jednocześnie upraszcza to proces projektowania, co pozwala na szybsze przejście od koncepcji do gotowego produktu.

Szybkość działania robota jest kluczowa dla zwiększenia jego efektywności. Urządzenie, które w szybkim czasie może uzyskać rezultat jest bardziej użyteczne niż takie, które na uzyskanie wyniku potrzebuje długiego czasu.

Optymalizacja kosztów konstrukcji i implementacji sprawia, że projekt jest bardziej dostępny dla potencjalnych użytkowników i może być wykorzystywany w praktyce a nie tylko w warunkach laboratoryjnych. Dobór odpowiednich komponentów i technologii pozwala na obniżenie ogólnych wydatków bez utraty jakości. Dzięki temu realizacja staje się możliwa nawet przy ograniczonym budżecie.

Realizacja projektu w oparciu o powyższe założenia umożliwia zaprojektowanie robota, który jest nie tylko wydajny, ale również przystępny w produkcji i użytkowaniu.

1.2 Decyzje projektowe

Jedną z najważniejszych decyzji w trakcie realizacji projektu był wybór rodzaju kości, jaką będzie odczytywać i interpretować model sztucznej inteligencji. Jako najistotniejsze kryterium wyznaczono liczbę ścian będącą potęgą liczby dwa (już na wstępny etapie projektowania odrzucając standardową kość sześciocieścienną) w celu łatwej interpretacji wyniku w notacji binarnej, stosowanej powszechnie w informatyce, w tym w kryptografii. W następnej kolejności szukano kompromisu pomiędzy łatwością w odczytce ścianek a generowaniem jak największej liczby bitów jednym rzutem, czyli kości o jak największej liczbie ścianek.

Z dostępnych powszechnie na rynku kości tylko cztero-, ósmio- i szesnastościenne spełniają pierwszy z wymienionych wyżej kryteriów. Kość czterościenną odrzucono ze względu na jej kształt ostrosłupa foremnego o podstawie trójkąta, na której liczby zapisywane są na rogach, a nie ściankach (co pokazano na rys. 1.1a). Kość czterościenną rozważono również w postaci niestandardowych kształtów kości, które pokazano na rys. 1.2, jednakże je również odrzucono z racji na bardzo niską

dostępność na rynku. Kolejną przeszkodą była trudność z interpretacją pustych (zaokrąglonych) ścianek kości typu modern, pokazanej na rys. 1.2a, lub problem z odczytem bardzo małych oznaczeń w przypadku ściętego ostrosłupa pokazanego na rys. 1.2b, które przestałyby być widoczne przy nawet niewielkim przechyleniu kości.

Kość szesnastościenną również odrzucono ze względu na mały rozmiar ścianek oraz stosunkowo niewielki kąt ich nachylenia względem siebie, który sprawia, że na zdjęciu robionym idealnie nad kością widoczne jest kilka ścianek jednocześnie, jak pokazano na rys. 1.1c. Z pozostałych opcji tylko na kości ośmiościenniej (pokazanej na rys. 1.1b) widać z góry dokładnie jedną ściankę, a do tego ma liczbę ścianek będącą potęgą liczby 2 (co jest atutem, że względem na konieczność przetwarzania wyniku na system binarny), zatem to ten rodzaj kości został wybrany do użycia w projekcie.



RYSUNEK 1.1: Zdjęcia kości z góry



RYSUNEK 1.2: Nietypowe kości czworościenne

1.3 Struktura pracy oraz podział obowiązków

Struktura pracy jest następująca:

- Rozdział 2 przedstawia przegląd obecnie dostępnych sprzętowych generatorów liczb losowych, w tym liderów na rynku oraz innych rozwiązań, zbliżonych do zaprezentowanego w poniższej pracy.
- Rozdział 3 jest poświęcony budowie sprzętowej części robota.
- Rozdział 4 zawiera dokładny opis metody odczytywania wyniku rzutu kościami wraz z opisem wykorzystanej sieci neuronowej.

- Rozdział 5 opisuje protokoł komunikacji z innym urządzeniem odbierającym wynik pracy robota.
- Rozdział 6 przedstawia wykorzystane testy statystyczne oraz ich wyniki.
- Rozdział 7 zawiera ostateczne wnioski.

Realizacja projektu została podzielona w sposób przedstawiony w tabeli 1.1.

TABELA 1.1: Podział obowiązków w projekcie

Imię i nazwisko	Zakres pracy
Jakub Kędra	Projekt i wydruk w technologii druku 3D prototypów oraz ostatecznej wersji urządzenia. Projekt układu elektronicznego. Montaż elementów strukturalnych, połączenie elementów elektrycznych i elektronicznych. Implementacja oprogramowania sterującego komponentami urządzenia. Oznaczanie przykładów uczących dla sieci neuronowej. Wykonanie testów sprawdzających poprawność działania urządzenia. Zaprojektowanie przepływu pracy. Przegląd literatury.
Wojciech Kot	Projekt i implementacja części związanej z odczytem i przetwarzaniem zdjęć z kamery. Projekt i implementacja sieci neuronowej dokonującej klasyfikacji wyniku rzutu kościa. Przygotowanie zbiorów treningowych i testowych do trenowania sieci neuronowej. Wykonanie testów sprawdzających poprawność działania urządzenia. Oznaczanie przykładów uczących dla sieci neuronowej. Zaprojektowanie przepływu pracy. Główny pomysłodawca projektu. Przegląd literatury.
Jakub Prusak	Zaprojektowanie i implementacja interfejsu komunikacji i sterowania urządzeniem Przygotowanie komunikacji maszyny z innymi urządzeniami. Zrównoleglenie przetwarzania. Wykonanie testów sprawdzających poprawność działania urządzenia. Oznaczanie przykładów uczących dla sieci neuronowej. Przegląd literatury.
Julia Samp	Opracowanie i przeprowadzenie testów statystycznych otrzymanych wyników. Implementacja testów zgodnie z standardem FIPS. Porównanie i interpretacja wyników otrzymanych dla obu modeli. Wykonanie testów sprawdzających poprawność działania urządzenia. Oznaczanie przykładów uczących dla sieci neuronowej. Korekta tekstu. Przegląd literatury.

Rozdział 2

Obecnie stosowane generatory liczb losowych

Współczesne systemy kryptograficzne oraz aplikacje wymagają generowania wysokiej jakości liczb losowych, które muszą charakteryzować się wysoką jakością i odpornością na przewidywalność. Najlepiej to założenie spełniają generatory liczb prawdziwie losowych, czyli te oparte na zjawiskach fizycznych. Jest to spowodowane tym, że kiedy w układach fizycznych bierze się pod uwagę wiele czynników, to nawet najmniejsze zmiany w którymś z tych czynników sprawiają, że wynik jest odmienny. Z tego powodu nie można przewidzieć wyniku działania takiego układu, a uzyskane wyniki są losowe [18].

Jednak powszechnie używane są generatory pseudolosowe (PRNG, z ang. *Pseudorandom Number Generator*). PRNG są całkowicie deterministyczne i zależą od zadanej im wartości początkowej zwanej ziarnem (ang. *seed*). Dlatego TRNG (z ang. *True Random Number Generator*) są szczególnie istotne w kontekście aplikacji wymagających silnej ochrony danych, takich jak systemy kryptograficzne, podpisy cyfrowe oraz w generowaniu kluczy szyfrujących, gdyż nie da się przewidzieć w skuteczny sposób generowanych przez nie wartości.

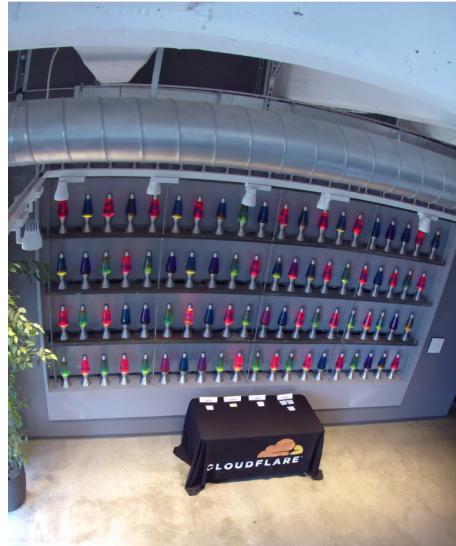
2.1 Obecnie stosowane rozwiązania

Obecnie komputery osobiste korzystają z generatorów liczb losowych (RNG, z ang. *Random Number Generator*) wykorzystujących dostępne im źródła entropii dostarczane przez użytkownika, takie jak ruchy myszy lub naciśnięcia przycisków. W nowoczesnych procesorach stosuje się też liczby generowane przez dedykowane moduły takie jak Intel DRNG za pomocą *Intel RDSEED* i *RDRAND* [11]. System Linux dostarcza również interfejsy `/dev/random` oraz `/dev/urandom` zapewniające liczby losowe.

2.2 Przegląd komercyjnych rozwiązań TRNG

Komercyjne rozwiązania bazują na przeróżnych źródłach entropii, najczęściej nie na dostarczanych przez użytkownika, aby wykluczyć możliwość ingerencji i ograniczyć przewidywalność, a na całkowicie niezależnych. Najpopularniejszym medialnie RNG jest LavaRand [9], który inspirowany był podobnym systemem, zbudowanym i opatentowanym przez Silicon Graphics [31].

Cloudflare oferuje użytkownikom dostęp do generatora liczb losowych w chmurze, który jest wykorzystywany m.in. do tworzenia kluczy kryptograficznych oraz w innych zastosowaniach wymagających silnych zabezpieczeń. Dzięki wykorzystaniu globalnej infrastruktury, generowane liczby losowe są szeroko dostępne i charakteryzują się dużą niezawodnością oraz odpornością na ataki.



RYSUNEK 2.1: Widok z kamery w biurze Cloudflare [9].

W ostatnich latach pojawiły się także inne rozwiązania chmurowe, które umożliwiają generowanie liczb losowych w czasie rzeczywistym bez potrzeby posiadania własnego sprzętu, za to na przykład za pośrednictwem API takiego jak *drand* [30]. Przykładem takiego rozwiązania jest Cloudflare – firma specjalizująca się w dostarczaniu usług związanych z bezpieczeństwem sieciowym. Na swoim blogu Cloudflare przedstawia zaawansowane metody generowania liczb losowych, które są wykorzystywane w ich systemach [10]. Dodatkowo wspomniana firma korzysta również z technologii opartych na zjawiskach fizycznych, takich jak rozpad radioaktywnego uranu, trójstopniowego wahadła czy „Entropy Wall”, która pobiera entropię z kamery ustawionej na ścianę lamp lawowych, pokazanej na rys. 2.1. Te prawdziwie losowe wartości zostają potem użyte jako ziarno dla generatorów pseudolosowych. Tego typu rozwiązania pozwalają na szybkie i bezpieczne generowanie liczb losowych w skali globalnej, zapewniając jednocześnie wysoki poziom ochrony przed atakami. LavaRand jest dzisiaj jedynie dodatkowym zabezpieczeniem firmy Cloudflare, na wypadek gdyby ich konwencjonalne źródła entropii okazały się niewystarczające. Wciąż jednak pozostaje inspirującym przykładem do szukania losowości w zjawiskach otaczających ludzi w codziennym życiu.

2.2.1 Rozwiązania sprzętowe

Wiodącymi producentami komercyjnych rozwiązań sprzętowych są firmy takie jak ID Quantique [22], Microchip Technology [28] i Semtech [39], które oferują zaawansowane urządzenia bazujące na TRNG. Produkty te zapewniają wysoki poziom bezpieczeństwa i są stosowane w wymagających tego aplikacjach, takich jak bankowość elektroniczna czy systemy wojskowe.

ID Quantique jest jednym z liderów w dziedzinie generatorów liczb losowych opartych na technologii fotoniki i mechanizmach kwantowych. Firma oferuje urządzenia, które wykorzystują detekcję fotonów w celu generowania liczb losowych. Dzięki temu rozwiązania ID Quantique charakteryzują się bardzo wysoką jakością losowości, a jednocześnie są odporne na ataki związane z analizą i przewidywaniem generowanych liczb.

Microchip Technology w swojej ofercie posiada różne moduły TRNG, w tym układy scalone, które generują liczby losowe na podstawie fluktuacji szumów termicznych. Produkty te znajdują zastosowanie w szerokim zakresie aplikacji, od urządzeń mobilnych po systemy wbudowane.

Semtech natomiast oferuje rozwiązania, które wykorzystują zjawiska losowe zachodzące w ob-

wodach analogowych do generowania liczb losowych. Firma ta jest jednym z głównych dostawców układów IoT, które wykorzystują własne TRNG do generowania wartości losowych w systemach komunikacji bezprzewodowej.

2.2.2 Podobne rozwiązania

Szukając dostępnych na rynku TRNG, można znaleźć wiele interesujących projektów. Są to zarówno rozwiązania komercyjne, najczęściej produkowane przez firmy wymienione powyżej, jak i otwartoźródłowe. Niewiele z nich jednak jest zbliżonych do realizowanego projektu. Wśród znalezionych rozwiązań zaledwie dwa inspirowały się fizycznym rzutem kością.

Pierwszym ze znalezionych rozwiązań był projekt *The Smart Dice Cup*, zrealizowany przez Carstena Magerkurth, Timo'a Engelke i Carstena Röcker [8]. Jest on inspirowany tradycyjnymi kubkami do gry w kości, przypomina je w swojej konstrukcji i sposobie działania. Urządzeniem należy potrząsnąć, aby uzyskać losową liczbę. Wynik takiego „rzutu” pokazywany jest na wyświetlaczu LED, zamontowanym na „pokrywce” urządzenia. Podstawową różnicą między realizowanym projektem a *The Smart Dice Cup* jest brak wykorzystania prawdziwych kości w drugim rozwiązaniu. Zamiast tego wynik jest generowany na podstawie zmiany prędkości, z jaką potrząsane jest urządzenie.

Drugim rozwiązaniem jest urządzenie wykorzystane w eksperymencie opisany w artykule *Dice thrown by cup and machine in PK tests* autorstwa J. B. Rhine [37]. Jest on najbardziej zbliżony do przedstawionego w poniższej pracy projektu, ponieważ również wprawia kość w ruch za pomocą obracającego się mechanicznie pojemnika. Z uwagi na technologie dostępne w 1943 roku, z którymi pracował J. B. Rhine, odczytywanie wyników musiało odbywać się ręcznie, a nie za pomocą małej kamery i sztucznej inteligencji. Jednakże, konstrukcja urządzenia jest bardzo podobna do zaprezentowanego w sekcji 3.1.2 pierwszego wariantu robota.

Rozdział 3

Budowa sprzętowego generatora liczb losowych

3.1 Projektowanie robota

3.1.1 Planowanie

Przy całym procesie budowy robota wykorzystano technologię FDM (ang. *Fused Deposition Modeling*) druku 3D. Pomimo tego, że proces druku 3D, zwłaszcza w przypadku dużych i skomplikowanych elementów, może być czasochłonny, jest to najlepsza dostępna technologia do realizacji tego rodzaju projektów. Druk 3D umożliwia tworzenie niestandardowych, precyzyjnie dopasowanych komponentów, które można szybko zmodyfikować w fazie projektowania i łatwo wydrukować ponownie w przypadku wprowadzenia zmian. Dzięki temu prototypowanie w projektach takich jak budowa robotów czy urządzeń mechanicznych jest znacznie bardziej elastyczne i tańsze w porównaniu do tradycyjnych metod, takich jak obróbka mechaniczna czy formowanie wtryskowe, które często wymagają kosztownego sprzętu i specjalistycznej wiedzy.

Drukarki 3D są także stosunkowo przystępne cenowo. Pozwalają one na wykorzystanie różnorodnych materiałów, takich jak tworzywa sztuczne, które są lekkie i trwałe, czy bardziej zaawansowane filamenty kompozytowe. Ta wszechstronność materiałowa umożliwia dostosowanie właściwości mechanicznych części, takich jak wytrzymałość, elastyczność lub odporność na wysokie temperatury, w zależności od wymagań projektu.

Co więcej, technologia druku 3D pozwala tworzyć skomplikowane kształty, których wykonanie innymi metodami mogłoby być niemożliwe lub wymagać kosztownych narzędzi. Możliwość drukowania prototypów bezpośrednio na miejscu skraca czas realizacji koncepcji, której efektem jest gotowy produkt, co jest szczególnie cenne w projektach inżynierskich, takich jak konstrukcja robotów. To czyni druk 3D idealnym narzędziem w projektach, które wymagają wielokrotnego testowania i wprowadzania ulepszeń.

Głównym założeniem podczas projektowania i budowy robota było założenie modułowości. Oznacza to, że każdy element jest wymienny i łatwo dostępny. Dzięki takiemu podejściu wymienianie elementów w przypadku awarii czy też małe modyfikacje wynikające z udoskonalania działania robota są znacznie prostsze i przede wszystkim szybsze, niż gdyby cały robot był jednolitą bryłą.

Proces projektowania robota rozpoczęto od przeanalizowania różnych metod wykonywania rzutu kościami. Rozważano tradycyjne rozwiązania takie jak kubki do gry w kości oraz baffle-box - specjalny pojemnik wykorzystywany do losowego mieszania kości. Innym podejściem byłoby skonstruowanie robota symulującego rzut kościami za pomocą mechanicznego ramienia, jednak to rozwiązanie odrzucono ze względu na skomplikowaną mechanikę urządzenia. Opcje wykorzystania

baffle-boxa odrzucono również ze względu na potrzebę transportu kości pomiędzy górami a dołem urządzenia. To wymagałoby wykorzystania kolejnych mechanicznych części takich jak np. taśma transportowa. Po przeanalizowaniu wszystkich pomysłów, zdecydowano się na rozwiązanie wykorzystujące kubek do gry w kości.

Kolejnym etapem była decyzja, w jaki sposób będzie wykonywany rzut kośćią. Pierwszym pomysłem było skonstruowanie mechanizmu, który będzie podrzucał kość w górę. Rozważano, żeby w tym celu wykorzystać tłoki, które unosiłyby cały kubek wraz z kośćią znajdująca się w środku w taki sposób, żeby kość została podrzucona, a w momencie spadania uderzała o dno oraz ścianki kubka. Uznano jednak, że takie rozwiązanie mogłoby się okazać trudne pod względem wykonania z kilku powodów.

Po pierwsze, tłoki generujące odpowiednią siłę do podrzucenia kubka musiałyby działać bardzo precyzyjnie, aby zapewnić powtarzalność i właściwą wysokość rzutu. Jednakże wielokrotne uderzenia w mechanizm mogą prowadzić do jego szybkiego zużycia, co w konsekwencji mogłoby spowodować awarię. Dodatkowo, konstrukcja takiego systemu tłokowego wymagałaby solidnego montażu i zastosowania materiałów odpornych na obciążenia, takie jak wibracje czy przeciążenia powstałe podczas pracy. Bez odpowiedniej sztywności konstrukcji, częste uderzenia i ruchy mogłyby powodować luzy w mechanizmie, a w efekcie całkowite uszkodzenie się robota. Co więcej, projektowanie i wykonanie tłoków wraz z precyzyjnym układem sterowania wymagałoby znacznych nakładów czasu i kosztów. W przypadku pracy ciągłej, dodatkowym problemem mogłaby być konieczność regularnej konserwacji i napraw mechanizmu, aby zapewnić jego długotrwałą sprawność. Wszystko to czyni to rozwiązanie niepraktycznym dla tego projektu, szczególnie gdy możliwe są prostsze i bardziej niezawodne alternatywy.

Po rozważeniu możliwości zdecydowano, że najlepszym rozwiązaniem będzie obrotowy kubek, wewnątrz którego kość porusza się i odbija od ścianek. Wariant ten roboczo nazwano *betoniarką* od podobnej zasady działania. W czasie przeglądu literatury w temacie rzutów kością znaleziono artykuł, w którym opisywany jest eksperyment, do którego wykorzystano właśnie taki mechanizm, ponieważ zapewnia on efekt rzutu zbliżony do takiego wykonanego przez człowieka [37]. Obrotowy kubek został zaprojektowany w taki sposób, aby umożliwić łatwą kontrolę nad jego ruchem. Dzięki temu możliwe jest ustalenie częstotliwości obrotu oraz określenie, jak długo ma się obracać. Wszystkie te ustawienia można zmieniać za pomocą programu napisanego w języku Python. W praktyce oznacza to, że kubek jest połączony z silnikiem, który jest sterowany przez komputer Raspberry Pi. Dzięki temu za pomocą komend w kodzie, sterujących czasem działania silnika i napięciem zasilającym silnik, możliwe jest dostosowanie, jak szybko i jak długo kubek ma się obracać. W celach testowych został skonstruowany prototypowy model robota.

3.1.2 Prototypowanie

Pierwszy prototyp robota składał się z metalowych prętów służących za stelaż oraz elementów wydrukowanych na drukarce 3D. Tymi elementami były: kubek, ramię służące do montażu kubka, uchwyty do prętów oraz płytka mocująca do kamery. Dodatkowo wykorzystano silnik prądu stałego z przekładnią 48:1 napędzający kubek [4] oraz sterownik służący do zasilania i sterowania ruchem silnika [42]. Pierwszy prototyp po złożeniu przedstawiono na rys. 3.1.

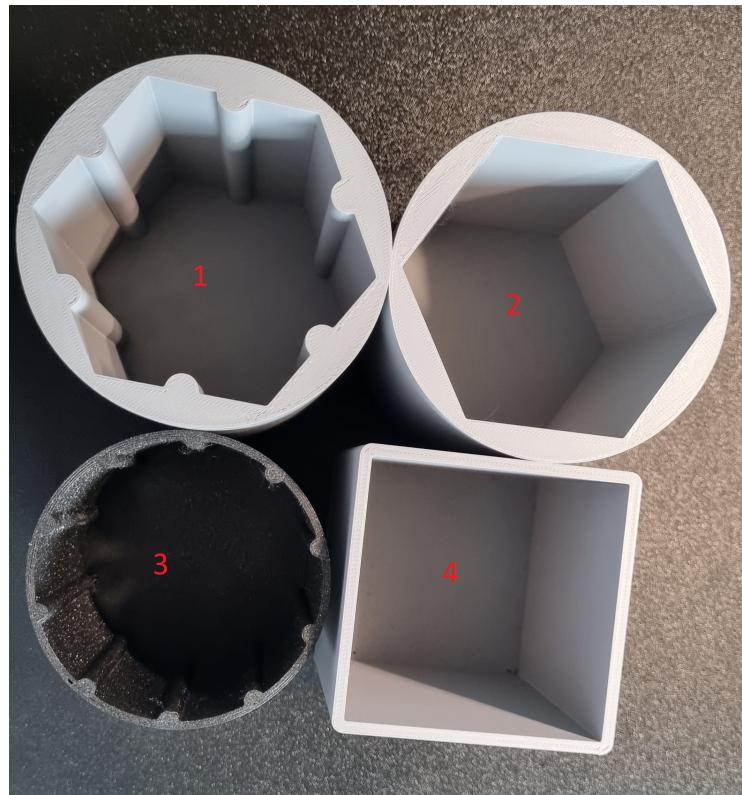
Dokładne dane techniczne komponentów robota wykorzystanych w ostatecznej wersji robota znajdują się w sekcji 3.2.



RYSUNEK 3.1: Pierwszy prototyp.

Do pierwszych testów robota zaprojektowano cztery warianty kształtów kubków. Przy ich projektowaniu wymiary wzorowano na dostępnych tradycyjnych kubkach do gry w kości. Średnice kubków do gry z reguły są w przedziale 70-90 mm [20], z tego powodu założono, że kubek powinien mieć średnicę około 80 mm. Przyjęto, że odpowiedni do tego zadania kubek powinien zawierać pewnego rodzaju nierówności na ściankach. Takie samo założenie przyjęto przy wcześniej wspomnianym eksperymencie [37]. Dzięki temu kość nie będzie się ślizgać po ściance, a zacznie się odbijać od tych nierówności, co będzie lepiej imitowało rzut kością wykonany przez człowieka. Z tego powodu odrzucono tradycyjny model kubka do gry w cylindrycznym kształcie o gładkich ściankach. Zaprojektowano cztery wersje kubka (patrz rys. 3.2): kubek kwadratowy (nr 4), kubek sześcienny (nr 2), kubek sześcienny z dodatkowymi żebrami (nr 1) oraz kubek cylindryczny z dodatkowymi żebrami (nr 3).

Następnie je przetestowano, umieszczając w środku kość do gry i obracając kubek wokół osi przechodzącej przez środek kubka. W czasie testów najgorzej sprawdziły się kubki sześcienne, ponieważ kości ośmiościennne, które wykorzystywano do testów, utykały w narożnikach dociskane przez siłę odśrodkową. Podobny problem pojawił się przy testach kubka kwadratowego. Najlepszym wariantem okazał się być cylindryczny kubek z dodatkowymi pionowymi żebrami (nr 3 na rys. 3.2), o które kość się odbijała podczas kręcenia. Jego zaletą jest to, że nie posiada on narożników, w których kość mogłaby utknąć. Dodatkowym atutem jest niewielka masa w porównaniu z resztą testowanych wariantów, co ma duże znaczenie przy obracaniu całego kubka ze względu na moment bezwładności [45]. Kubek o mniejszej masie jest łatwiej wprawić w ruch, co jest ważne ze względu na wybór silnika.



RYSUNEK 3.2: Testowane warianty kubków.

Po pierwszych testach okazało się, że niezbędny do uzyskania zamierzonego efektu będzie mechanizm, który będzie wychylał cały kubek wraz z silnikiem, który odpowiada za jego obrót. Spowodowane to było faktem, że w momencie kiedy kubek stał prostopadle do podłoża, kość zwykle ślimała się po jego dnie, ponieważ nie działała na nią żadna siła, która mogłaby wprowadzić ją w ruch, który spowodowałby odbijanie się od ścian kubka w sposób podobny jak podczas tradycyjnego rzutu kością. W tym celu postanowiono skonstruować mechanizm wychylający cały kubek, przez co siła ciężkości zaczęły dociskać ją do bocznych ścian kubka. W takim ułożeniu po wprowadzeniu kubka w ruch obrotowy, przy odpowiedniej prędkości obrotu kość zaczynała odbijać się o ścianki kubka. Z początku planowano wykorzystanie serwomechanizmu, jednak to rozwiązanie odrzucono, ponieważ większość dostępnych serwomechanizmów ma ograniczony obrót do 180° lub 360° , a to limitowało możliwości mechanizmu służącego do wychylania kubka. Ostatecznie w tym celu wybrano mały silnik krokowy 28BYJ-48, którego moment obrotowy 34,3 mNm jest w pełni wystarczający do wychylenia kubka. Silnik ten obraca układem dwóch kół zębatych przedstawionych na rys. 3.3.



RYSUNEK 3.3: Koła zębata.

Podczas testów pierwszej wersji robota wykorzystującej obrotowy kubek powstał pomysł alternatywnego rozwiązania. Rozwiązanie to implementuje inne podejście do rzutu kością. Zamiast obracać cały kubek, a dodatkowo wychylać go, wykorzystany został trwale zamontowany kubek, na którego dnie znajduje się śmiegle, które podcina leżącą na dnie kościę. Takie rozwiązanie znacznie upraszcza cały mechanizm robota oraz bardzo przyspiesza proces losowania liczby. Ten wariant nazwano roboczo *blenderem* – podobnie jak wcześniej opisany wariant *betoniarki* – od podobnej zasady działania mechanizmu.

Przy projektowaniu drugiego wariantu robota został wykorzystany ten sam stelaż złożony z metalowych prętów co w pierwszym wariantie. Na drukarce 3D wydrukowano dodatkowe części, niezbędne do realizacji tego wariantu. Zaprojektowano i wydrukowano nowy kubek, śmiegle oraz mocowanie dla silnika. Kubek został przystosowany do montażu silnika prądu stałego oraz śmieglą.

W trakcie testów zauważono, że procesor robota nagrzewa się do wysokich temperatur podczas intensywnej pracy, co mogło negatywnie wpływać na jego wydajność i żywotność. Aby temu zapobiec, w projekcie zdecydowano się na zastosowanie radiatorów (rys. 3.4), które miały pomóc w rozproszeniu nadmiaru ciepła, oraz wentylatora, który wspomagał cyrkulację powietrza wokół procesora. Jest to działanie zalecane w oficjalnej dokumentacji na stronie Raspberry Pi, mające przeciwdziałać ograniczaniu wydajności w celu ochrony przed przegrzaniem (ang. *thermal throttling*) [34]. Dodatkowym problemem rozgrzewania się procesora do wysokich temperatur jest jego bliski kontakt w robocie z materiałem PLA, z którego wykonywane były komponenty robota oraz prototypów. Materiał ten zaczyna się deformować przy temperaturze 60°C [6] co mogłoby powodować problemy. Dzięki wykorzystaniu radiatorów oraz wentylatora udało się obniżyć temperaturę pracy procesora. Praca wentylatora jest sterowana przy pomocy tranzystorów w układzie Darlingtona, złącza GPIO Raspberry Pi oraz programu w języku Python, który implementuje prostą metodę progową. Oznacza to, że program sprawdza temperaturę procesora za pomocą wbudowanych funkcji dostępnych na Raspberry Pi. Jeśli temperatura procesora przekracza 55°C, to program wysyła sygnał do uruchomienia wentylatora. Jeśli temperatura spadnie poniżej 50°C, program wyłączy wentylator. Program ten jest dodany do usługi *systemd*, dzięki czemu jest uruchamiany od razu po włączeniu systemu i stale monitoruje temperaturę procesora.



RYSUNEK 3.4: Dodane radiatory.

W obu wariantach dużym problemem był zlej jakości obraz z kamery spowodowany przez ciemne wnętrze kubka. W celu poprawy jakości zdjęć zaprojektowano system oświetlenia składający się z diod LED sterowanych przy pomocy tranzystorów w układzie Darlingtona. Dzięki temu wnętrze kubka stało się dużo jaśniejsze, co pozwala kamerze na robienie zdjęć lepszej jakości. Zdjęcie oświetlonego kubka przedstawiono na rys. 3.5.

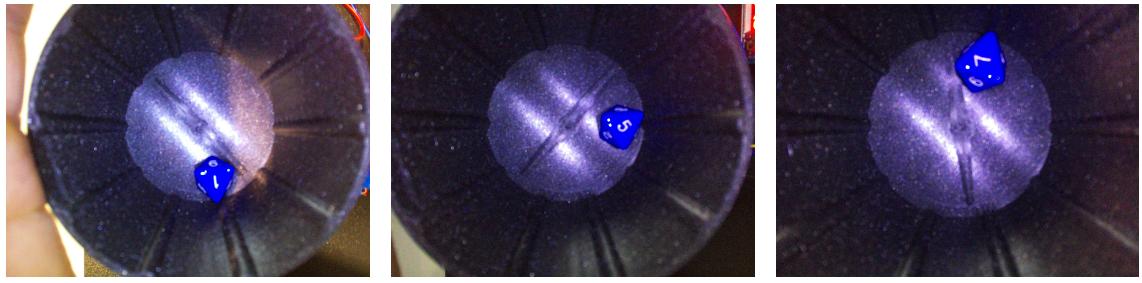
Dodatkowo rozświetlenie wnętrza kubka na tyle poprawiło jakość zdjęć, że pozwoliło to na obniżenie kamery względem kubka. Dzięki temu wysokość całego urządzenia zmniejszyła się, co miało duże znaczenie, ponieważ jednym założeniem postawionych na początku budowy było stworzenie urządzenia o niewielkich rozmiarach. Dzięki tym zabiegom otrzymywane zdjęcia stały się dużo bardziej wyraźne oraz pole widzenia kamery było ograniczone tylko do dna kubka. Diody LED służące do oświetlenia wnętrza kubka połączono szeregowo. Dzięki temu nie trzeba było wykorzystywać dodatkowych rezystorów, a liczba połączeń jest minimalna, ponieważ wystarczają dwa przewody do podłączenia całego układu diod.



RYSUNEK 3.5: Oświetlone wnętrze kubka.

Podczas testów prototypów konieczne było również określenie wysokości, na której może znajdować się kamera nad kubkiem. W tym celu wykonano zdjęcia z różną odlegością kamery od dna kubka (patrz rys. 3.6). Głównym celem było znalezienie minimalnej odległości kamery od dna kubka, które nie powodowałoby znacznego pogorszenia się jakości zdjęć, co utrudniałoby rozpoznanie wyników na kości. Dodatkową zaletą obniżenia kamery względem kubka był fakt, że kamera

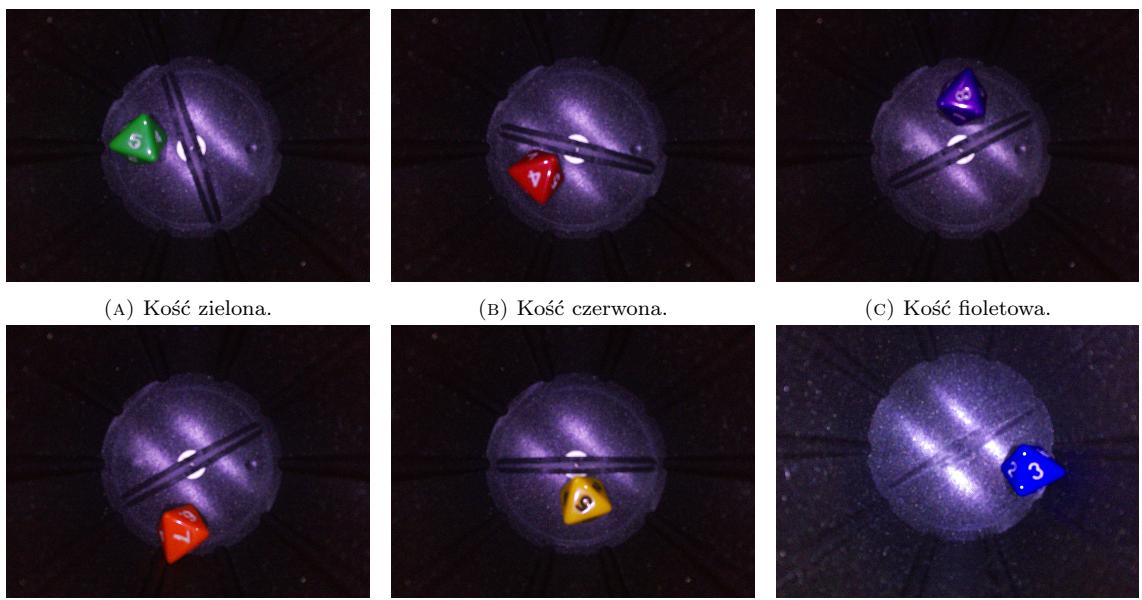
znajdująca się niżej miała w swoim zasięgu widoku tylko dno kubka, co również miało za zadanie ułatwić rozpoznawanie kości oraz jej wartości. W trakcie testowania odległość tą wyznaczono na 120 mm.



(A) Pierwsze ustawienie. (B) Drugie ustawienie. (C) Trzecie ustawienie.

RYSUNEK 3.6: Ustawienia kamery na różnych wysokościach.

Duże znaczenie ma również wykorzystywana kość. Od jej koloru i tekstury zależy jakość zdjęć zrobionych przez zamontowaną kamerę. Na rys. 3.7 przedstawiono przykładowe zdjęcia kości w różnych wariancji kolorystycznych. Na podstawie tych zdjęć wybrano kość niebieską, ponieważ numer oznaczający wynik rzutu najlepiej kontrastuje ze ścianami kości, przez co jest on najlepiej widoczny właśnie na tej kości.

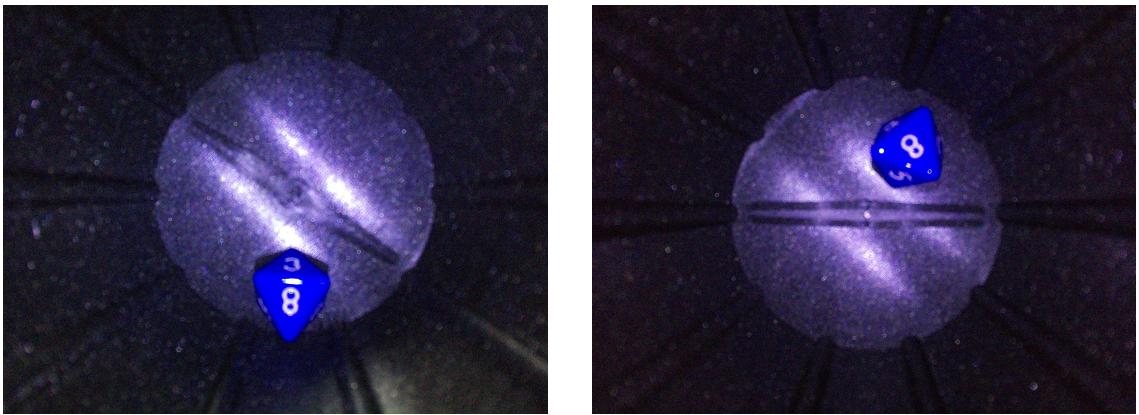


(A) Kość zielona. (B) Kość czerwona. (C) Kość fioletowa.

(D) Kość pomarańczowa. (E) Kość żółta. (F) Kość niebieska.

RYSUNEK 3.7: Przetestowane kości w różnych kolorach.

W trakcie testów konieczne okazało się również skorygowanie jasności świecenia diod LED. Wykorzystany układ czterech diod LED połączonych szeregowo dawał zbyt jasne światło, co objawiało się poprzez duże odblaski na powierzchni kości, które stanowiły jeden z głównych problemów dla modelu sieci neuronowej odczytującego wyniki rzutów, o czym jest mowa w sekcji 4.1.2. Z tego powodu eksperymentalnie wybrano rezystory, których zadaniem było ograniczenie napięcia zasilającego układ diod LED. Porównanie jasności świecenia układu diod przedstawiono na rys. 3.8.



(A) Oświetlone wnętrze kubka bez rezystorów ograniczających napięcia zasilającego diody LED.

(B) Oświetlone wnętrze kubka z rezystorami ograniczającymi napięcie zasilające diody LED.

RYSUNEK 3.8: Oświetlone wnętrze kubka.

Po skonstruowaniu obu wersji robota i wstępny przetestowaniu ich okazało się, że wariant *blendera* znacznie szybciej (około 4-krotnie) wykonuje rzut kością. Pojedynczy rzut kością za pomocą *blendera* wraz ze zrobieniem zdjęcia trwa około 1,7 sekundy. Ponadto wersja *blendera* jest znacznie stabilniejszą konstrukcją, ponieważ nie wymaga poruszania dużymi komponentami robota. Największą zaletą wariantu *blendera* jest prostsza – w porównaniu z wariantem *betoniarki* – budowa spowodowana mniejszą liczbą ruchomych elementów. Jest to ważne z punktu widzenia długotrwałej eksploatacji, podczas której bardziej skomplikowane mechanizmy szybciej się używają. Z tych powodów do docelowego robota wybrano wariant *blendera*. Dzięki temu projekt stał się mniej skomplikowany mechanicznie, a jednocześnie jego użyteczność wzrosła, ponieważ głównym zadaniem tego robota jest generowanie liczb losowych, a to zadanie szybciej był w stanie wykonywać właśnie ten wariant.

3.1.3 Ostateczna wersja robota

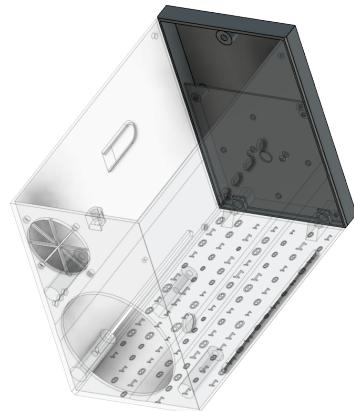
Projektowanie ostatecznej wersji robota rozpoczęto od przeanalizowania wad, zalet oraz ogólnych cech prototypów. Postanowiono, że komputer oraz układy sterujące będą umieszczone z boku kubka, a nie nad nim, tak jak w prototypach, ponieważ sprawiało to, że robot byłby zbyt wysoki względem założenia kompaktowego urządzenia. Na podstawie testów prototypów wywnioskowano, że najlepszym rozwiązaniem będzie zaprojektowanie obudowy umożliwiającej łatwy dostęp do wszystkich komponentów. Postanowiono, że finalna wersja robota będzie wykorzystywała kubek o tej samej średnicy co w prototypie. Oznaczało to, że kubek będzie miał zewnętrzną średnicę 80mm i ścianki o grubości 2 mm, na których tak jak w prototypie będą pionowe żebra (patrz nr 3 na rys. 3.2). Dookoła kubka, którego ostateczne wymiary wyniosły 126×80 mm, zaprojektowano resztę konstrukcji. Obudowę zaprojektowano w taki sposób, żeby była w stanie pomieścić kubek oraz kamerę umieszczoną na wysokości 120 mm nad dnem kubka. Dodatkowo w tylnej oraz dolnej części obudowy pozostawiono przestrzeń na resztę elementów składowych robota. Wielkość tej przestrzeni wyznaczono poprzez zwymiarowanie pozostałych elementów takich jak silnik, sterownik [42], wentylator czy Raspberry Pi [36]. Te wymiary posłużyły do określenia minimalnej potrzebnej przestrzeni, którą wyznaczono na około 30 mm z tyłu, 35 mm od dołu oraz 10 mm od góry kubka. To minimum zwiększo w taki sposób, żeby we wnętrzu pomieściły się również przewody i śruby oraz żeby po całkowitym złożeniu pozostała przestrzeń do swobodnego manipulowania elementami składowymi. Dokładne wymiary ostatecznej wersji robota przedstawiono na rys. 3.17 w sekcji 3.2.

W celu spełnienia założenia modułowości, każdy element zaprojektowano w taki sposób, żeby

posiadał specjalne miejsca na inserty lub śruby. Inserty to mosiężne elementy, które za pomocą lutownicy wgrzewa się w wydruk 3D. Posiadają one wewnętrzny gwint, dzięki czemu można dołączenia wydruków wykorzystywać śruby, nie uszkadzając samego wydruku przy wielokrotnym skręcaniu i rozkręcaniu robota. Dzięki temu założenie modułowości zostało spełnione, ponieważ dzięki śrubom i insertom każdy element robota mógł zostać wydrukowany jako osobna część, którą następnie połączono z innymi częściami w prosty do rozłożenia sposób.

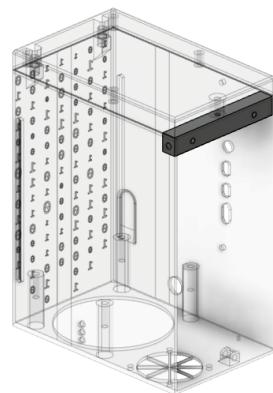
Opis komponentów ostatecznej wersji robota

Górna część obudowy – pokrywa – przedstawiona na rys. 3.9, mieszcząca kamerę oraz diody LED, została zaprojektowana w taki sposób, żeby dostęp do kubka pozostał łatwy. Osiągnięto to poprzez wykorzystanie prostego mocowania pokrywy do obudowy z wykorzystaniem pojedynczej śruby oraz magnesów neodymowych. Dzięki temu w przypadku kiedy konieczny jest dostęp do kamery lub diod LED, wystarczy zdjąć tylną ścianę robota oraz odkręcić pojedynczą śrubę. Dodatkowo magnesy neodymowe zapewniają dobre przyleganie pokrywy do reszty obudowy. Ich dodatkową zaletą jest blokowanie pokrywy w ustalonej pozycji podczas umieszczania kości wewnętrz kubka.



RYSUNEK 3.9: Pokrywa obudowy.

Dla zapewnienia sztywności całej konstrukcji oraz punktu mocowania dla tylnej ściany robota zaprojektowano belkę przykręcana do obu ścian. W tą belkę – przedstawioną na rys. 3.10 – wkręcana jest również wcześniej wspominana śruba mocująca pokrywę obudowy.

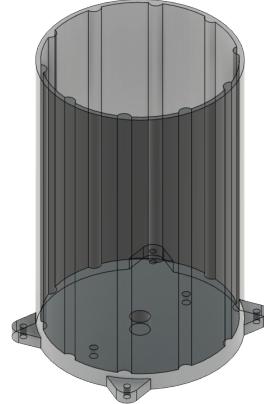


RYSUNEK 3.10: Belka.

Kamera wraz z diodami LED służącymi do oświetlenia wnętrza kubka znajduje się w pokrywie. Kamerę zamontowano przy użyciu dwóch śrub M2, natomiast diody LED umieszczone w zaprojektowanych w wydruku otworach.

W pokrywie znajdują się również dwa sześciokątne otwory na magnesy neodymowe. Dzięki temu, że otwory są sześciokątne, to walcowe magnesy idealnie się w nie wpasowują i po wciśnięciu nie wypadają. W drugiej części obudowy znajdują się takie same otwory na drugą parę magnesów. Dla pewności podczas umieszczania magnesów wykorzystano klej cyjanoakrylowy.

Kubek, w którym dokonywane są rzuty kością, zaprojektowano na podstawie kubka z prototypowej wersji *blendera*. Zachowano jego średnicę oraz kształt i rozmieszczenie wewnętrznych żeber. Zmieniona została zasada mocowania kubka w taki sposób, żeby był on przystosowany do zamocowania w obudowie. W tym celu zaprojektowano cztery mocowania widoczne na rys. 3.11, znajdujące się u dołu kubka, za pomocą których kubek jest przykręcany do obudowy. Dodatkowo pogrubiono dno kubka tak, żeby można było w nim umieścić inserty służące do przykręcenia uchwytu silnika. Ostatnią modyfikacją było podwyższenie kubka w taki sposób, żeby wysokość sięgała on aż do mocowania kamery – górnej pokrywy. Dzięki temu podczas rzutów zniknął problem z wypadającą kośćią, co było dość częstym zjawiskiem podczas testów prototypu. Niestety takie rozwiązanie spowodowało, że wnętrze kubka przestało być widoczne z zewnątrz. Jednak uznano, że widok z zewnątrz na wnętrze kubka nie jest konieczny do osiągnięcia celów projektu.



RYSUNEK 3.11: Kubek.

Projekt pierwszego uchwytu mocującego silnik składał się z miejsca do umieszczenia silnika oraz dwóch cylindrycznych słupków służących za prowadnice śrub mocujących cały uchwyt do dna kubka. Jednak, ponieważ podczas testów pojawiły się problemy z pierwszym silnikiem i wymieniono go na inny, konieczne było zaprojektowanie drugiego uchwytu. Podczas projektowania drugiego uchwytu wzorowano się na projekcie pierwszym, jednak dodano dodatkowy trzeci cylindryczny słupek na śrubę, która miała służyć do kontrolowania wychylenia całego uchwytu w osi przód-tył. Zwiększyło to możliwość regulacji i w ten sposób ograniczono tarcie śmigła o dno kubka.

Śmigło zaprojektowano w taki sposób, żeby obracając się przy samym dnie kubka, po uderzeniu w kość wybijało ją w góre. Ten efekt uzyskano dzięki niskiemu profilowi śmigła oraz bocznych ścian śmigła nachylonych pod kątem 45°.

Podczas testów ostatecznej wersji robota napotkano wcześniej wspominane problemy z pierwotnie wykorzystywanym silnikiem DC 6 V. Silnik ten miał okrągły wał, przez co śmigło musiało być bardzo dokładnie spasowane, aby uniknąć ślizgania się wału wewnątrz otworu śmigła. To sprawiało, że montowanie śmigła i jego demontaż był bardzo trudny. Dodatkowo po wielokrotnych

rzutach kością śmigło wbijało się coraz niżej na wał silnika i w ostateczności tarło o dno kubka tak mocno, że silnik nie był w stanie się obracać. Żeby temu zaradzić, umieszczono pomiędzy śmigłem a dnem kubka metalową podkładkę, po której śmigło mogło się ślizgać łatwiej niż po dnie kubka. To jednak nie rozwiązało problemu, ponieważ po kolejnych kilku tysiącach rzutów śmigło zaczęło się blokować. Z tego powodu postanowiono wymienić silnik na mocniejszy 12 V silnik z przekładnią, który ma mniejszą częstotliwość obrotu (około 2000 RPM zamiast 4000 RPM), ale ma większy moment obrotowy. Zaletą nowego silnika jest jego wał w kształcie litery „D”. Pozwala to na znacznie luźniejsze spasowanie otworu śmigła z wałem, przez co demontaż śmigła jest znacznie łatwiejszy. Ponadto nowy silnik jest znacznie cichszy niż poprzedni. Obie wersje uchwytów silnika przedstawiono na rys. 3.12.



(A) Pierwsza wersja uchwytu silnika oraz śmigło względem kubka.



(B) Druga wersja uchwytu silnika oraz śmigło względem kubka.

RYSUNEK 3.12: Zaprojektowane uchwyty silnika oraz śmigła.

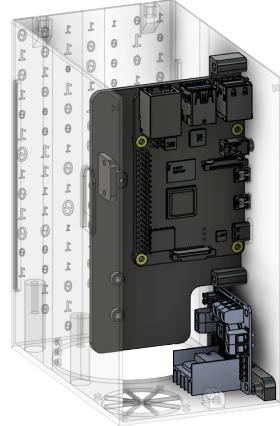
Do mocowania Raspberry Pi zaprojektowano specjalną płytę przykręcana do boku obudowy, którą przedstawiono na rys. 3.13. W projekcie tej płytki uwzględniono otwory do przymocowania Raspberry Pi oraz układu ULN2803A Darlington. Dodatkowo przygotowano specjalne miejsce do mocowania przycisku. Na płytce pozostało również miejsce na zamontowanie szyny zasilania. Płytkę tą umieściłyśmy w obudowie w taki sposób, żeby znajdowała się ona bezpośrednio nad wentylatorem. Dzięki temu strumień powietrza bezpośrednio chłodzi najważniejsze elementy elektroniczne robota. Przycisk zamocowano na tej samej płytce, z wykorzystaniem dodatkowego elementu wydrukowanego na drukarce 3D. Dzięki temu znalazły się on bezpośrednio przy ścianie obudowy, a dodatkowo jego mocowanie nadal spełnia założenie modułowości poprzez mocowanie za pomocą śrub i insertów.



RYSUNEK 3.13: Płytki do montażu elektroniki.

Układ L298N sterujący silnikiem przykręcono do obudowy pośrednio, poprzez specjalnie zaprojektowane i wydrukowane mocowanie. Dzięki temu wykorzystano gotowe otwory na śruby znaj-

dujące się w płytce układu L298N. Na rys. 3.14, na którym przedstawiono płytka do mocowania elektroniki z zamontowaną Raspberry Pi 4b oraz układ L298N, widoczne jest wspominane mocowanie służące do przykręcenia układu L298N do dna obudowy robota.

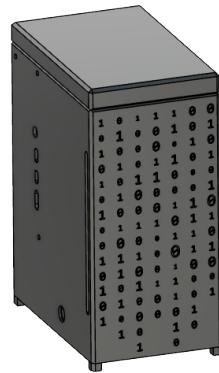


RYSUNEK 3.14: Zamocowany sterownik L298N [38] z widoczną płytka, na której zamocowana jest Raspberry Pi 4b [40].

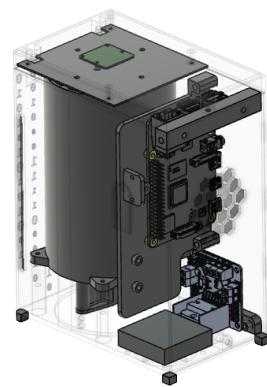
Do dna obudowy robota przymocowano, za pomocą śrub, również wentylator.

Obudowę zaprojektowano w taki sposób, żeby pomieściła wszystkie powyższe elementy. W jej ścianach zaprojektowano otwory na wyjścia Raspberry Pi, diody LED oraz gniazdo zasilania. W ścianie obudowy na wysokości miejsca, w którym znajduje się wewnętrzny przycisk, zaprojektowano specjalne wycięcie. Dzięki dodatkowemu zmniejszeniu grubości ściany obudowy w tym miejscu jest bardziej elastyczna, co pozwala na kliknięcie przycisku znajdującego się po wewnętrznej stronie ściany obudowy. Na dnie obudowy zaprojektowano również specjalne słupki służące za podpórki dla kubka. W słupkach tych zaprojektowano otwory na inserty, dzięki którym kubek można przykręcić do obudowy gwarantując tym jego stabilność.

Podczas projektowania obudowy przewidziano także takie elementy jak wycięcia od spodu bezpośrednio pod wentylatorem, służące za wlot powietrza oraz wycięcia w tylnej ścianie, służące za wylot powietrza. Dodatkowo w dnie umieszczono duży otwór umożliwiający swobodne mocowanie oraz dostęp do silnika, diod LED i gniazd zasilania. Na bocznych ścianach zaprojektowano przerwy, które następnie zaślepiono kontrastującym kolorystycznie filamentem. Przerwy te tak samo jak cyfry na przedniej ścianie obudowy są tylko i wyłącznie elementami estetycznymi. Ostatnim elementem robota są zaprojektowane nóżki, które przyklejono do dna robota. Zapewniają one przepływ powietrza pod robotem gdzie znajduje się wlot powietrza do wentylatora. Gotowy model robota przedstawiono na rys. 3.15 oraz rys. 3.16.



RYSUNEK 3.15: Projekt gotowego robota od frontu.



RYSUNEK 3.16: Projekt gotowego robota z pokazanymi wewnętrzny komponentami [40, 38, 21].

3.2 Dokumentacja techniczna – hardware

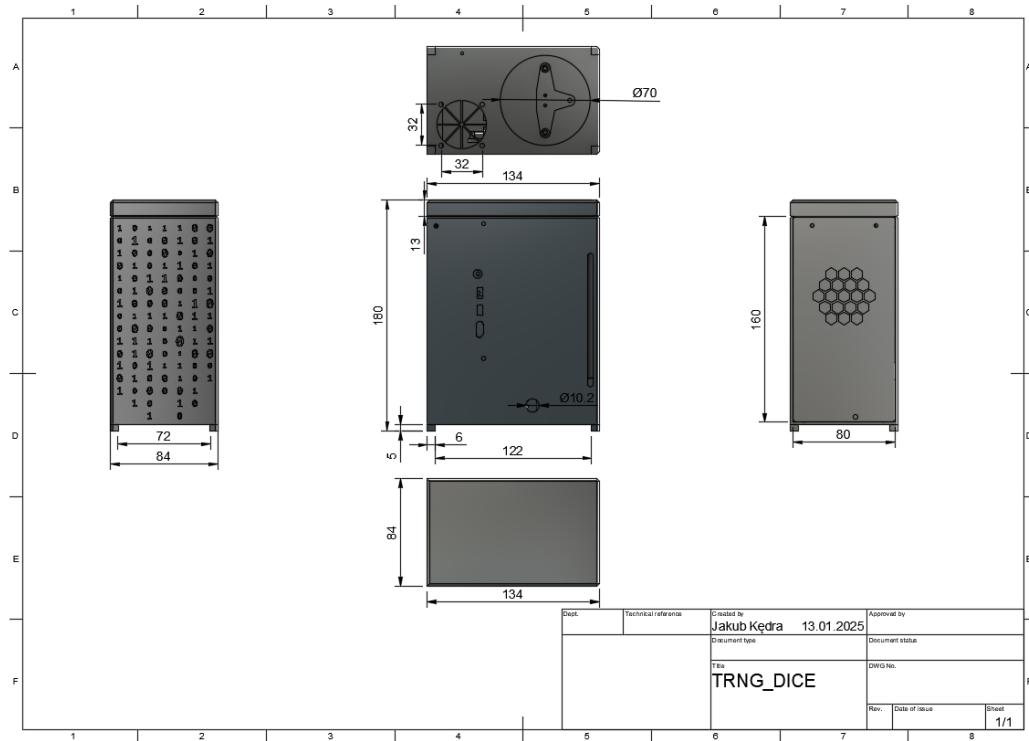
3.2.1 Ogólne parametry robota

Model 3D całego robota znajduje się w repozytorium projektu pod linkiem: <https://github.com/KotZPolibudy/Kosteczkator/tree/main/model%203d>.

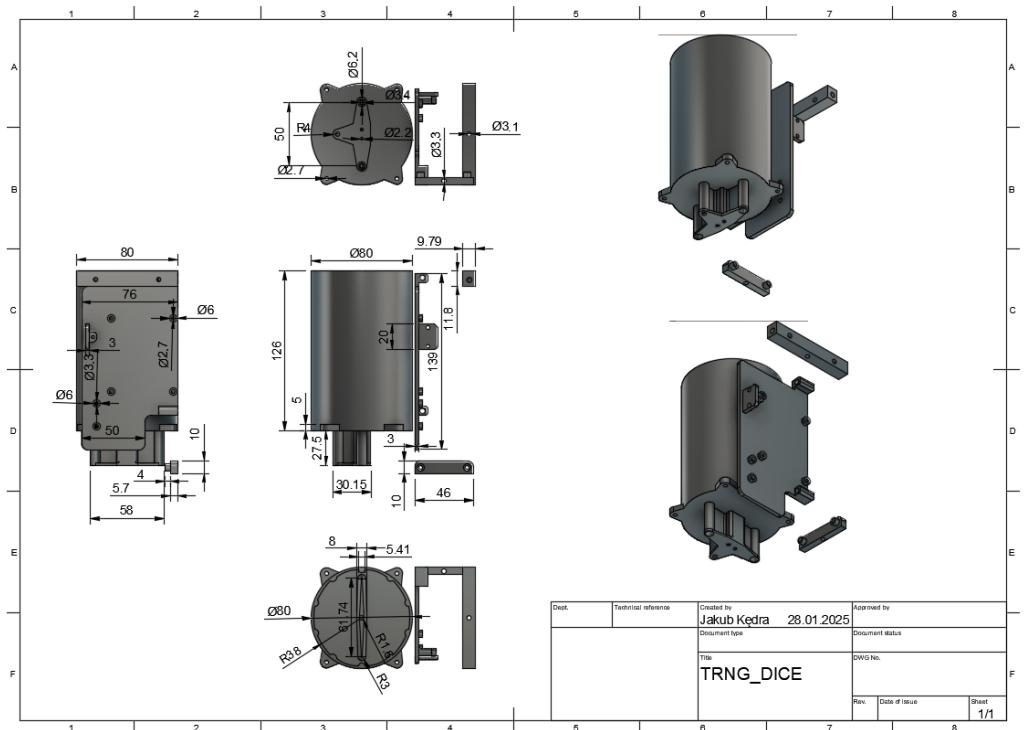
Wymiary zewnętrzne: $180 \times 134 \times 84$ mm

Masa: 0,5 kg

Pozostałe najważniejsze wymiary przedstawiono na rys. 3.17 oraz na rys. 3.18.



RYSUNEK 3.17: Wymiary zewnętrzne.



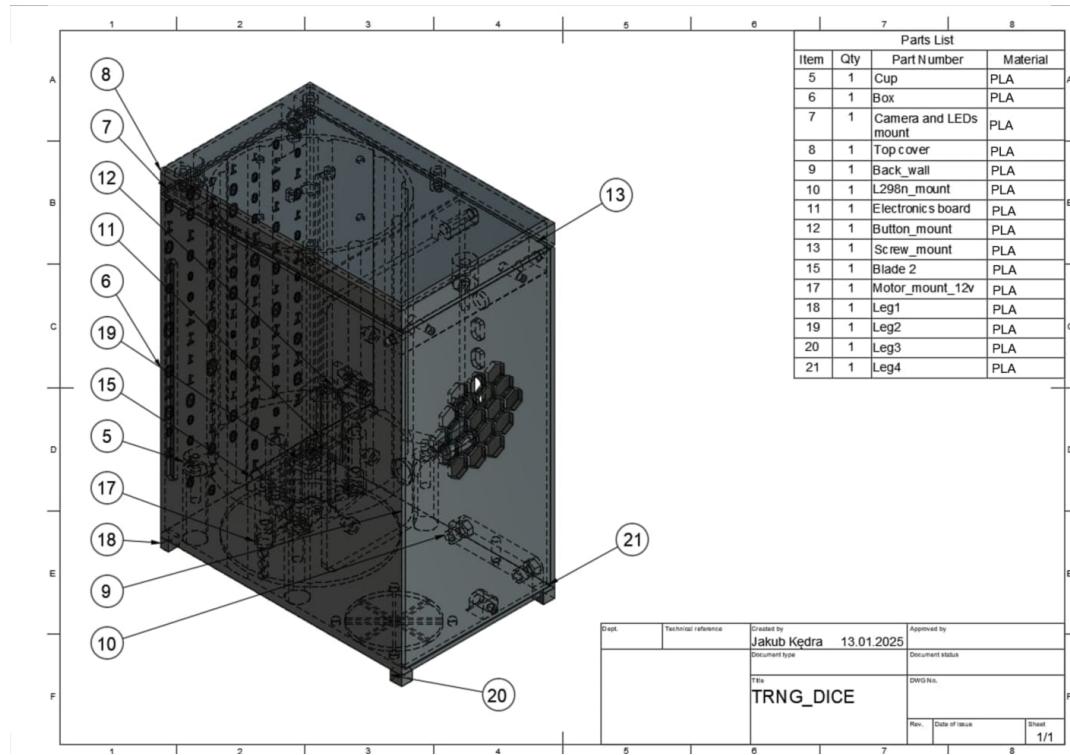
RYSUNEK 3.18: Wymiary komponentów wewnętrznych.

3.2.2 Wykorzystane elementy strukturalne

Wszystkie elementy strukturalne wymienione poniżej są zaznaczone na rys. 3.19. Zostały one wydrukowane w technologii FDM na drukarce 3D. Jako materiał do druku części wybrano PLA (ang. *Polylactic Acid*) [6]. Wyboru dokonano po porównaniu właściwości fizycznych, cen i łatwości

wydruku różnych materiałów. Wzięto pod uwagę PLA, PETG (ang. *Polyethylene Terephthalate Glycol-modified*) [5] oraz ABS (ang. *Acrylonitrile Butadiene Styrene*) [7]. Ostatecznie wybrano PLA ze względu na jego sztywność, dużą popularność, niską cenę oraz prostotę procesu druku. ABS odrzucono ze względu na szkodliwe opary podczas druku i potrzebę zamkniętej komory drukarki. PETG najprawdopodobniej byłby również odpowiednim materiałem do wydrukowania potrzebnych w tym projekcie części, jednak jest on podatny na nitkowanie w czasie druku. Ta cecha obniżała by jakość wydruku komponentów robota, które zawierają wiele otworów drukowanych w płaszczyźnie pionowej co sprzyjałoby nitkowaniu [2, 27].

- obudowa (nr 6 na rys. 3.19),
- tylna ściana obudowy (nr 9 na rys. 3.19),
- górná pokrywa (nr 8 na rys. 3.19),
- kubek (nr 5 na rys. 3.19),
- płytka do montażu kamery i diod LED (nr 7 na rys. 3.19),
- uchwyt do silnika (nr 17 na rys. 3.19),
- płytka do montażu elektroniki (nr 11 na rys. 3.19),
- mocowanie przycisku (nr 12 na rys. 3.19),
- mocowanie sterownika L298 (nr 10 na rys. 3.19),
- belka (nr 13 na rys. 3.19),
- śmigło (nr 15 na rys. 3.19),
- nóżki (nr 18-21 na rys. 3.19).



RYSUNEK 3.19: Komponenty robota

3.2.3 Elektronika

- Raspberry Pi 4B – Komputer wyposażony w czterordzeniowy procesor ARM Cortex-72. Posiada on możliwość łączenia Wi-Fi 802.11ac oraz Bluetooth 5.0 [36].
- Raspberry Pi Camera V2 – Kamera przystosowana do łączenia z Raspberry Pi za pomocą taśmy Raspberry Pi. Posiada ona 8-megapikselowy sensor Sony IMX219 [36].
- L298N – Układ zaprojektowany do przyjmowania standardowych sygnałów TTL wykorzystywany jako sterownik silników [42].
- ULN2803A – Układ ten zawiera osiem tranzystorów w układzie Darlingtona. Tranzystory są w układzie ze wspólnym emiterem. Układ ma dodatkowo wbudowane diody tłumiące dla obciążzeń indukcyjnych [41].
- Silnik N20 DC 12 V z metalową przekładnią 2000RPM.
- Trzy diody LED 3 mm białe.
- Trzy diody LED 3 mm: czerwona, niebieska, zielona.
- Przycisk monostabilny THT.
- Gniazdo wtykowe DC w formacie 5,5 × 2,1 mm.

3.2.4 Zasilanie

- Zasilacz DC 12 V – wejście Power Jack 5,5 x 2,1 mm. Wykorzystano zasilacz o mocy 60 W, jednak w zupełności do zasilania robota wystarczyłby zasilacz o mocy 8,5 W zakładając około 50% zapasu mocy.

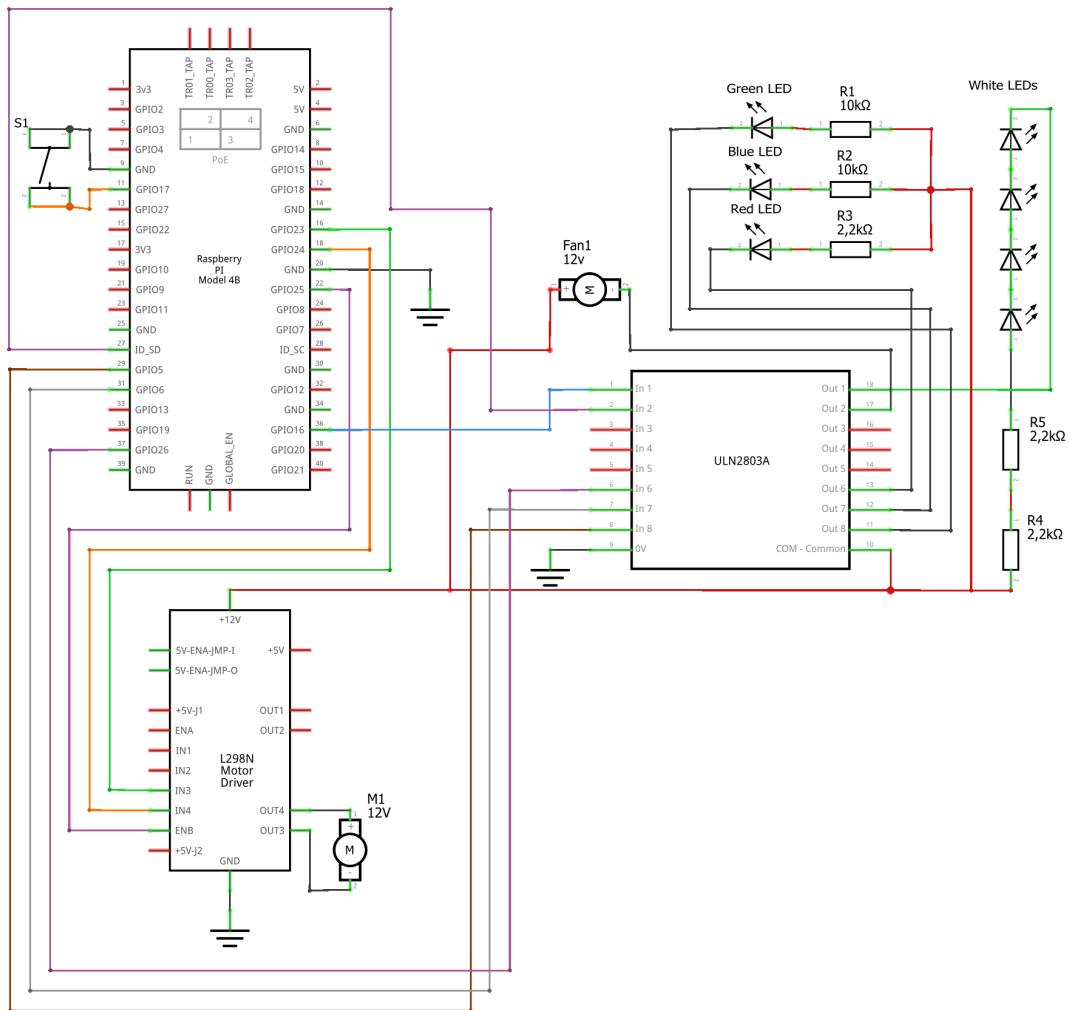
3.2.5 Podłączenie elektroniki

Schemat na rys. 3.20 przedstawia wszystkie elementy układu elektronicznego wraz z połączeniami, które zostały wykorzystane w robocie. Złącza GPIO mikrokontrolera Raspberry Pi 4b zostały użyte do kontrolowania poszczególnych elementów, takich jak silnik prądu stałego, wentylator oraz diody LED. Elementy te są sterowane za pośrednictwem układów L298N oraz ULN2803A. Raspberry Pi jest zasilane przez port USB-C, a wszystkie podzespoły dzielą wspólną masę. Silnik prądu stałego (M1) jest sterowany za pomocą układu L298N, który umożliwia kontrolę prędkości z jaką obraca się silnik, poprzez kontrolę współczynnika wypełnienia (ang. *duty cycle*) za pomocą sygnału PWM generowanego przez Raspberry Pi i dostarczanego na pin ENB poprzez złącze GPIO. Układ L298N pozwala również na kontrolę kierunku, w której obraca się silnik poprzez odpowiednie ustawienie stanów wysokiego i niskiego na pinach IN3 oraz IN4.

Zewnętrzny zasilacz dostarcza napięcie 12V, które zasila zarówno silnik, jak i oba sterowniki – L298N oraz ULN2803A. Do podłączenia zestawu trzech diod LED (czerwonej, niebieskiej oraz zielonej umieszczonych w gotowym robocie w ściance obudowy robota) wykorzystano rezystory ograniczające napięcie, które podłączono do anod poszczególnych diod. Wartości rezystancji dobrano na podstawie zalecanych wartości napięcia dla tych diod oraz w taki sposób, żeby ich światło nie było zbyt jasne, ponieważ mają to być diody służące tylko do sygnalizowania procesów wykonywanych przez robota. Z tego powodu rezystory, które wybrano mają wyższe wartości rezystancji niż te, które są zalecane, jednak było to działanie celowe. Dla diod zielonej i niebieskiej wybrano

rezystor $10\text{ k}\Omega$ dla każdej osobno, natomiast dla czerwonej $2,2\text{ k}\Omega$. Piny GPIO Raspberry Pi odpowiadają za kontrolę włączania poszczególnych diod poprzez sterowanie układem ULN2803A. Białe diody LED są połączone szeregowo i sterowane za pomocą tranzystorowego układu ULN2803A. W układzie zastosowano rezystory ograniczające napięcie. Każda z tych diod powinna być zasilana napięciem od 3 V do 3,2 V, co oznacza, że po połączeniu szeregowym ich układ powinien być zasilany napięciem 12 V, które dostarcza im układ ULN2803A. Zdecydowano się jednak na zastosowanie rezystorów ograniczających napięcie ze względu na zbyt dużą jasność świecenia tych diod, co powodowało powstawanie odblasków na kości. Wartości rezystancji tych rezystorów dobrano eksperymentalnie, o czym jest mowa w sekcji 3.1.2 i ostatecznie wybrano dwa rezystory połączone szeregowo o rezystancji $2,2\text{ k}\Omega$, dające łącznie $4,4\text{ k}\Omega$. Dodatkowo układ zawiera mały wentylator (Fan1) zasilany napięciem 12V.

W robocie wykorzystano szynę zasilania, z której zasilane są wszystkie wspomniane komponenty. Schemat zawiera także przycisk (S1), który jest wykorzystywany jako prosty przełącznik do wywołania sygnału wejściowego na pinie GPIO Raspberry Pi.

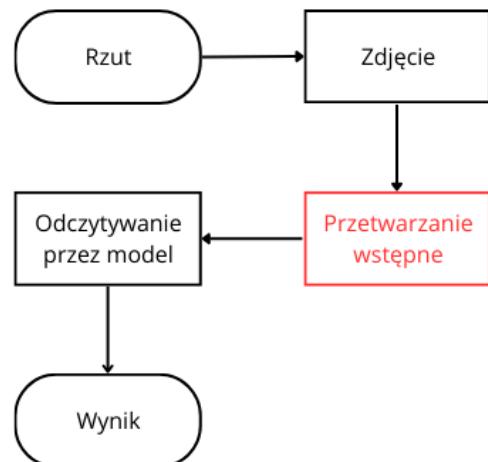


RYSUNEK 3.20: Schemat elektryczny.

Rozdział 4

Odczytywanie losowego wyniku z kości

Kolejnym krokiem przetwarzania – po wykonaniu rzutu i zdjęcia – jest odczyt wylosowanej wartości z kości. Jak pokazano na rys. 4.1, najpierw zdjęcie musi zostać poddane odpowiedniemu przetwarzaniu, które zostało opisane w tym rozdziale.



RYSUNEK 4.1: Schemat pracy robota

4.1 Przetwarzanie wstępne obrazów

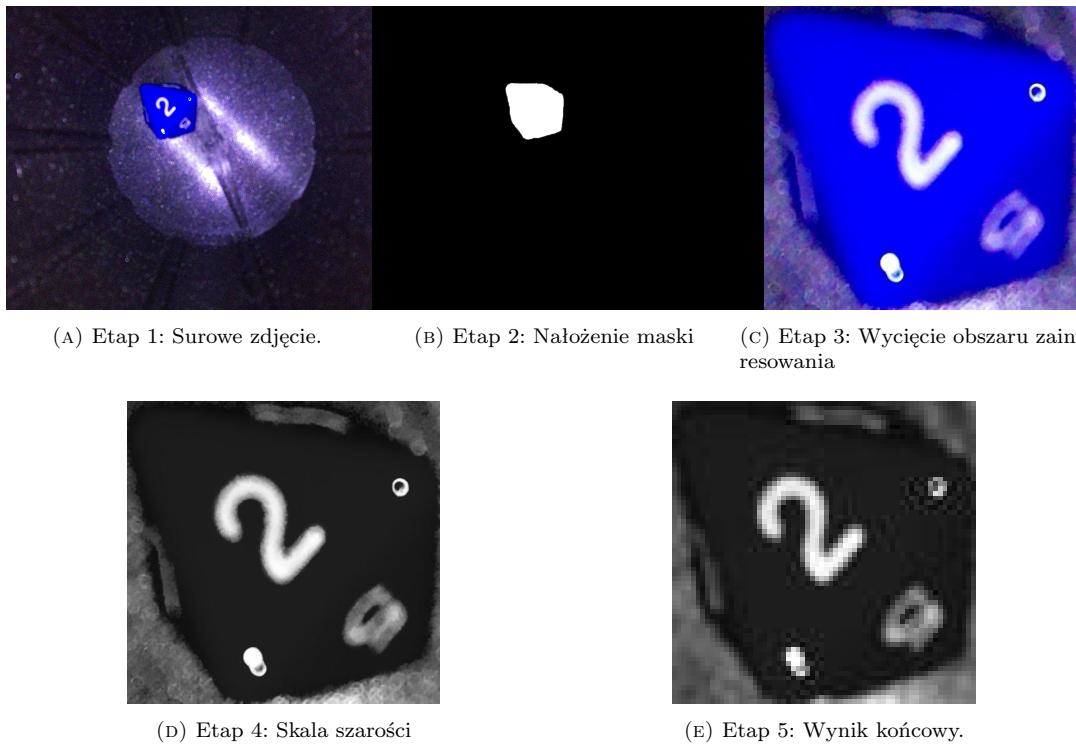
W niniejszym rozdziale omówiono proces przetwarzania wstępnego obrazów kości, który przekształca dane pochodzące z fizycznego komponentu robota (kamery dostarczającej zdjęcia kości) na dane wejściowe dla modelu sztucznej inteligencji. Przez dane wejściowe dla modelu SI rozumie się tutaj odpowiednio sformatowane obrazy, a więc takie w skali szarości, o rozmiarach 64×64 piksele, zawierające jedynie kość wyciętą ze zdjęcia (nie zawierające w tle całego kubka).

4.1.1 Algorytm

1. Wczytanie obrazu wejściowego.
2. Odnalezienie kości za pomocą maski na komponencie nasycenia.
3. Stworzenie i wycięcie ramki ograniczającej (ang. *bounding box*) wokół maski.
4. Przeskalowanie do odpowiedniego rozmiaru.

5. Konwersja do skali szarości.
6. Zapisanie gotowego obrazu.

Na rys. 4.2 przedstawiono przykładowe zdjęcie surowe 4.2a oraz kolejne etapy przetwarzania, aż do finalnego etapu 4.2e.



RYSUNEK 4.2: Kolejne etapy przetwarzania obrazu. Wszystkie obrazy mają równą wysokość.

Przedstawiony algorytm został zaimplementowany w języku Python, a jego zadaniem jest identyfikacja, wycięcie i przeskalowanie obszarów zawierających obiekty zainteresowania na zdjęciach.

Zdjęcia w formacie JPEG są wczytywane za pomocą modułu Pillow [17], który umożliwia konwersję obrazów do przestrzeni barw RGB, zapewniając jednolitość formatów danych wejściowych. Następnie obrazy są przekształcane do przestrzeni barw HSV, co pozwala oddzielić komponenty odpowiadające za barwę (H), nasycenie (S) oraz jasność (V).

Komponent nasycenia (S) jest wygładzany za pomocą filtra Gaussa [12], również dostępnego w ramach tego samego modułu, co redukuje szумy i pozwala na bardziej precyzyjną analizę. Wykorzystano parametry filtru z maską o wymiarach 5×5 pikseli oraz odchyleniem standardowym równym 1. Takie ustawienia zapewniają balans między wygładzeniem a zachowaniem szczegółów obrazu.

Na podstawie wygładzonego komponentu nasycenia za pomocą progowania tworzona jest maska binarna, która identyfikuje obszary o wysokim nasyceniu, odpowiadające obiektom zainteresowania (kości). Próg został dobrany eksperymentalnie, kończąc na wartości 224, tak aby w kontrolowanym środowisku (opisanym w poprzednim rozdziale) maska obejmowała obszar zainteresowania, ale nie obejmowała tła (czarnego kubka). Korzysta to z faktu, że kubek wykonany jest z materiału o niskim nasyceniu, a używana kość ma jednolity, jasny kolor, a więc i wysokie nasycenie.

W celu usunięcia niewielkich luk w masce binarnej stosowana jest operacja zamknięcia morfologicznego [1]. Operacja ta ujednolica maskę, co jest szczególnie istotne w przypadku obiektów o niejednorodnej strukturze nasycenia, gdzie maska mogłaby zawierać rozłączne fragmenty obiektu.

Po zastosowaniu zamknięcia morfologicznego maska jest analizowana w celu zlokalizowania największego konturu obejmującego obszar zainteresowania. W tym celu wykorzystano funkcje biblioteki OpenCV [16], które umożliwiają zarówno zastosowanie domknięcia morfologicznego, jak i wykrycie konturu, i obliczenie otaczającego go prostokąta. Rozmiar prostokąta jest dynamicznie dopasowywany do obszaru maski.

Na tej podstawie oryginalny obraz jest kadrowany w kształt prostokąta wokół obszaru zainteresowania, a następnie przeskalowywany do wymiarów 64×64 pikseli. Ostatecznie obraz jest konwertowany do skali szarości, co zmniejsza wymiarowość danych i pozwala sieci neuronowej skupić się na strukturze obrazu. Konwersja do skali szarości dodatkowo minimalizuje negatywny wpływ odblasków, powstających gdy kość odbija światło wprost do kamery, co poprawia niezawodność analizy.

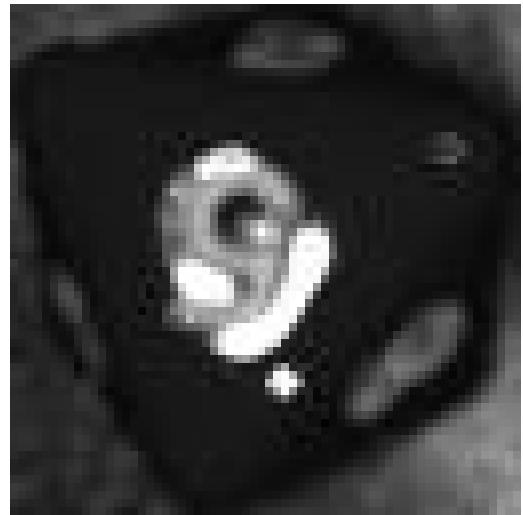
4.1.2 Zidentyfikowane trudności i ich rozwiązania

Wspomniane wcześniej odblaski znacznie pogarszają skuteczność odczytywania wyniku z kości. Jednak zastosowanie skali szarości pozwoliło w znacznym stopniu pozbyć się tego problemu, uwidaczniając widoczną na kości cyfrę, co pokazuje rys. 4.3.

Dzięki zastosowaniu skali szarości łatwiej jest oddzielić jasne punkty, będące wynikiem odblasku światła diody od ściany kości, od nieco ciemniejszych, lecz wciąż jasnych punktów oznaczających cyfrę na kości.



(A) Odblask na przeskalowanym zdjęciu.



(B) Odblask po zmianie na skalę szarości.

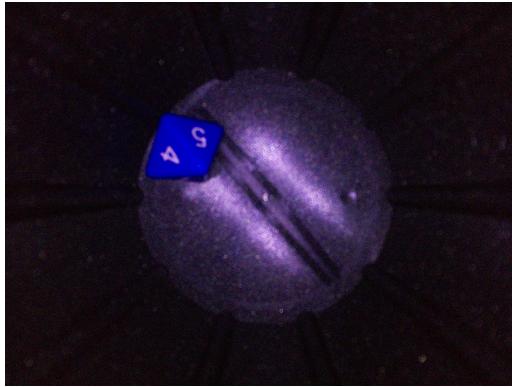
RYSUNEK 4.3: Porównanie odblasku przed i po przetworzeniu.

Inną trudnością, która objawiała się w początkowych fazach pracy, były zupełnie czarne obrazy, ale problem ten wynikał z usterki zastosowanej diody i został zażegnany poprzez fizyczną wymianę oświetlenia. Obecnie taki problem jest wykrywany oraz zgłaszanym jest wyjątek oznaczający potrzebę naprawy usterki oświetlenia.

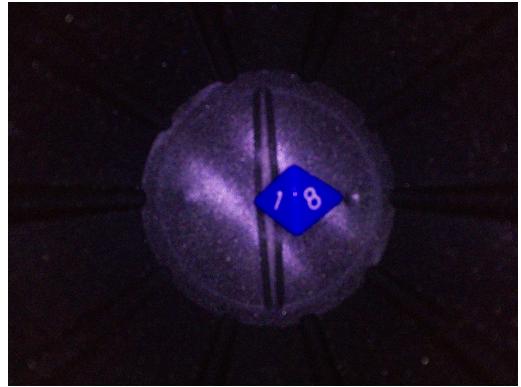
Nie był to jedyny problem powodowany przede wszystkim częścią sprzętową – drugim takim było zatrzymanie śmiegi w miejscu, przez co do ewaluacji przez model kilkukrotnie trafiało identyczne lub niemal identyczne zdjęcie. To również zostało rozwiązane fizycznie, poprzez dodanie metalowej podkładki pod śmigłem, ale mimo zażegnania problemu w taki sposób, zapewniono również obsługę wyjątku, który nadarzy się, gdy wielokrotnie zostanie odczytana identyczna wartość. W związku z ryzykiem fałszywego rozpoznania zatarcia śmiegi przez losowo wypadającą wielokrotnie identyczną wartość, ustalono próg wykrywania takiej sytuacji i zgłaszania wyjątku

na 13 identycznych odczytów z rzędu. Statystyczna szansa na takie zdarzenie przy prawidłowej pracy maszyny wynosi $\frac{1}{8^{12}} \approx 1,455 \cdot 10^{-11}$

Kolejną, znacznie częściej spotykaną trudnością w obecnej architekturze urządzenia są niejednoznaczne wyniki, a więc moment, gdy kość zatrzyma się w pozycji, w której widać więcej niż jedną ściankę. Najczęściej jest to spowodowane tym, że kość zatrzymała się na śmigle napędowym, przez co wynik może być niejednoznaczny, co pokazano na rys. 4.4.



(A) Kość zatrzymana na śmigle (ujęcie 1).

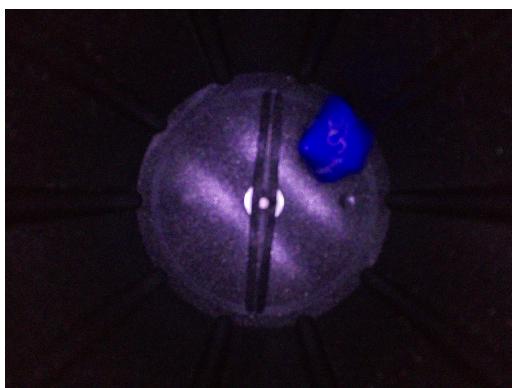


(B) Kość zatrzymana na śmigle (ujęcie 2).

RYSUNEK 4.4: Porównanie dwóch ujęć kości zatrzymanej na śmigle.

Rozwiązaniem tego problemu jest model dokonujący klasyfikacji wyniku rzutu, który odczytuje ze zdjęcia tylko jeden, najbardziej prawdopodobny wynik i wybiera ten, który jest bardziej widoczny, ponieważ podczas uczenia model miał do czynienia z takimi niejednoznacznymi sytuacjami, które zostały ręcznie oznaczone właśnie na potrzeby treningu modelu.

Zdarzają się również rzadkie przypadki, w których kość wyląduje równo na śmigle. Jednakże w takim wypadku wynik jest idealnie czytelny, identycznie jak gdy kość lądowała normalnie na podstawce kubka, więc nie jest to problemem. Ostatnim rodzajem problemu, jaki został naprawiony, był przypadek, w którym kość nie skończyła jeszcze swojej fazy lotu lub wpadła w bardzo długie wirowanie – efektywnie uniemożliwiając odczytanie wyniku. Przykłady zarejestrowanych przypadków wystąpienia tego problemu przedstawiono na rys 4.5.



(A) Wirująca kość.



(B) Kość w ruchu.

RYSUNEK 4.5: Nieczytelne ujęcia wynikające z dłuższego niż przewidywany ruchu kości

Rozwiązaniem większości wypadków, w których zachodziła ta sytuacja, okazało się spowolnienie działania maszyny. Obecnie oczekiwaniem aż kość zatrzyma się w miejscu steruje parametr czasowy, przyjmujący wartość jednej sekundy od zakończenia pracy śmigła. Jest to wystarczające rozwiązanie, gdyż obecnie sytuacje niepewne z tego powodu zdarzają się bardzo rzadko (około 1 raz na 5000 rzutów).

Istnieje możliwość udoskonalenia tego rozwiązania, poprzez stosowanie detekcji ruchu kości, jednak to rozwiązanie najpewniej okazałoby się bardziej kosztowne czasowo niż proste czekanie stosowane obecnie.

4.1.3 Podsumowanie

Przedstawiony algorytm przetwarzania wstępnego pozwala na skuteczne przygotowanie danych wejściowych dla modelu sztucznej inteligencji. Automatyzuje proces identyfikacji i kadrowania obiektów zainteresowania w obrazach, co znacząco poprawia jakość danych. Rozwiązanie zostało zaprojektowane z myślą o łatwej adaptacji do innych zastosowań wymagających podobnego przetwarzania obrazów, na przykład w przypadku zmiany kości w maszynie na inną, lub z inną liczbą ścianek.

4.2 Opis modelu SI

Model sztucznej inteligencji został zaprojektowany w celu klasyfikacji zdjęć kości ośmiościennych do jednej z ośmiu klas, odpowiadających cyfrom od 1 do 8 na każdej ze ścianek. Do implementacji modelu wykorzystano przede wszystkim moduł TensorFlow [15] oraz wchodzący obecnie w jego skład Keras [14], który umożliwia łatwe tworzenie i trenowanie sieci neuronowych.

4.2.1 Przygotowanie danych

Dane wejściowe zostały podzielone na zestawy treningowy i walidacyjny w proporcji 70:30. W celu zwiększenia różnorodności danych treningowych zastosowano techniki augmentacji obrazów dostępne w klasie `ImageDataGenerator` [13], takie jak:

- obrót o losowy kąt w zakresie do 90° ,
- przesunięcia poziome i pionowe w zakresie $\pm 20\%$ wymiarów obrazu,
- transformacje perspektywiczne (ang. *shear*) w zakresie $\pm 10\%$,
- losowe przybliżenia lub oddalenia (ang. *zoom*) w zakresie 90% – 110% oryginalnego obrazu.

Wszystkie augmentacje przedstawiono na rys. 4.6 Dodatkowo, obrazy są normalizowane do zakresu $[0, 1]$, co pozwala na skuteczniejsze działanie sieci neuronowej.

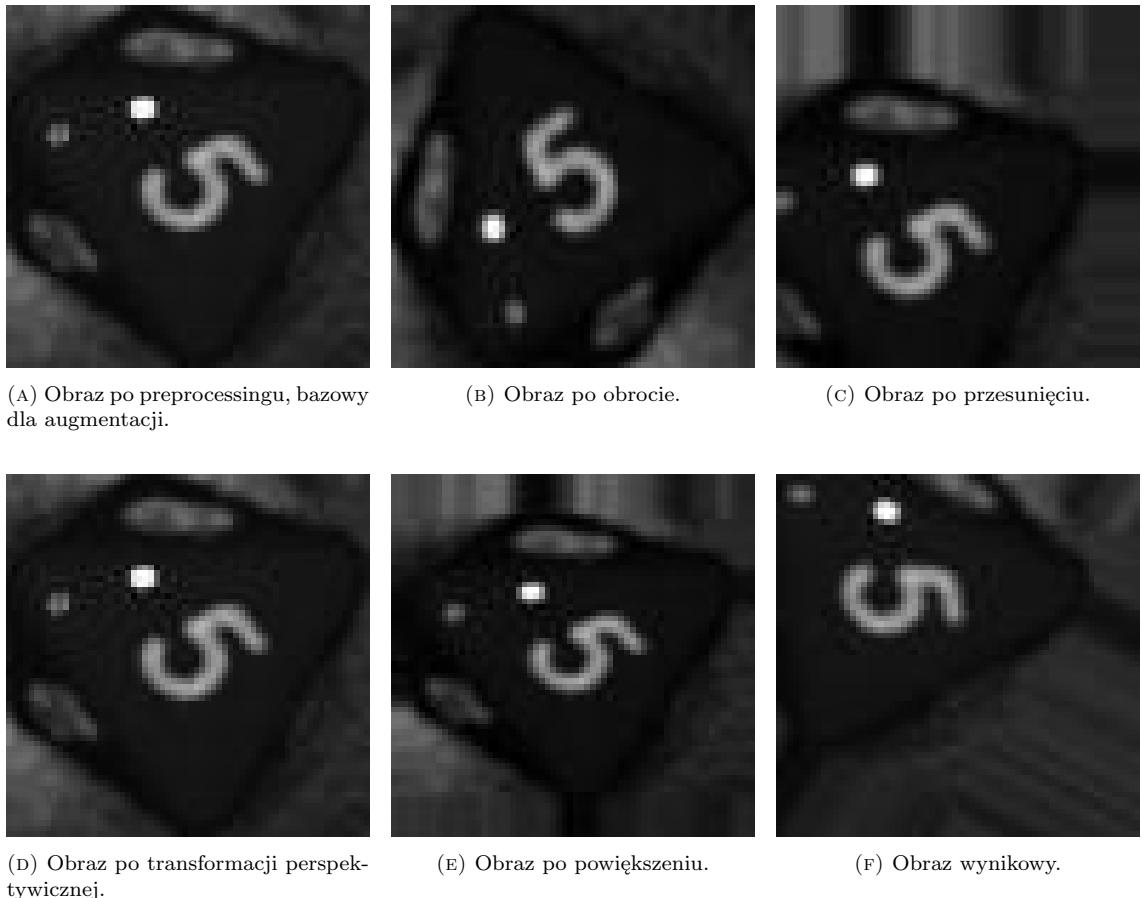
W szczególności należy zaznaczyć, że finalnie uniknięto początkowo przeoczonego błędu, jakim jest tworzenie obrazów lustrzanych w wyżej wymienionych transformacjach, gdyż obrazy lustrzane sprawiały, że ścianki 2 oraz 5 były znacznie gorzej rozróżniane przez wytrenowany model. Problem nie był początkowo łatwy do wykrycia, gdyż zdjęcia odbite zarówno w pionowej jak i poziomej osi były poprawne, a niepoprawne były te odbite tylko względem jednej osi, czego przykład można zaobserwować na rys. 4.7.

4.3 Architektura modelu

4.3.1 Użyte funkcje aktywacji i rodzaje warstw

Sieci splotowe (CNN)

Sieci splotowe (ang. *convolutional neural network, CNN*) [19] to rodzaj sieci neuronowych, które szczególnie dobrze sprawdzają się w zadaniach związanych z przetwarzaniem obrazów, takich jak



RYSUNEK 4.6: Różne składowe augmentacji obrazów uczących

rozpoznawanie obiektów, klasyfikacja czy segmentacja. Główna różnica między siecią splotową a tradycyjną siecią neuronową polega na tym, że w sieciach splotowych stosuje się operację splotu, która umożliwia wydobycie lokalnych cech z obrazu, redukując liczbę parametrów i umożliwiając efektywniejsze uczenie się reprezentacji obrazu.

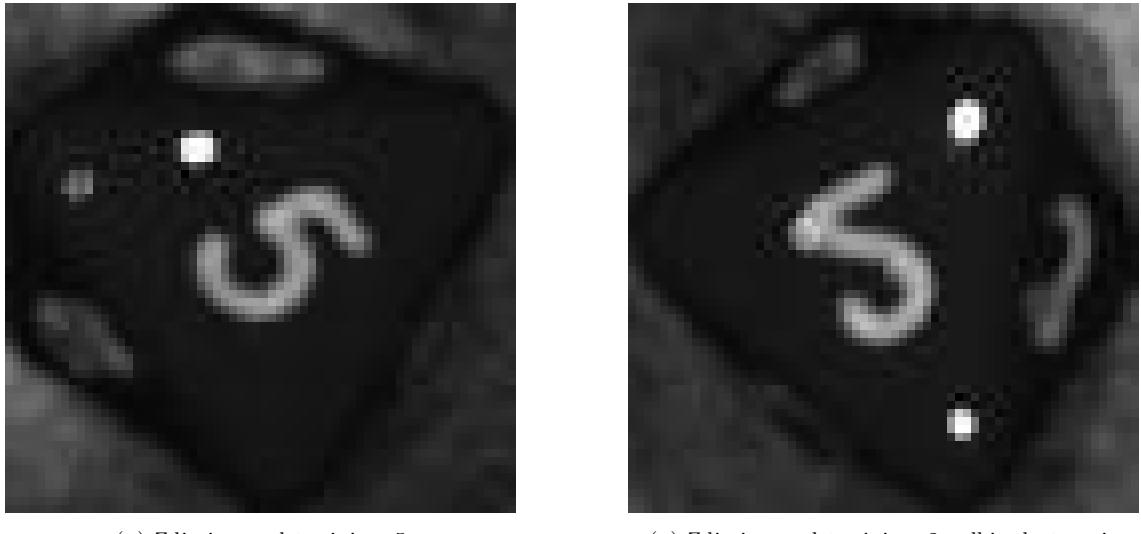
Warstwy splotowe w sieci CNN wykonują operację splotu, polegającą na zastosowaniu filtrów (jader) na obrazie wejściowym, co pozwala na wyodrębnienie różnych cech (np. krawędzi, tekstur) w różnych lokalizacjach obrazu. Kolejne warstwy splotowe uczą się coraz bardziej złożonych cech w miarę postępującej abstrakcji, dzięki czemu sieć może rozpoznać skomplikowane wzorce w danych wejściowych.

Warstwa gęsta (Dense)

Warstwa gęsta (ang. *Dense, fully connected layer*) [19], to warstwa w której każdy neuron jest połączony z każdym neuronem w poprzedniej warstwie. Tego typu warstwy zwykle znajdują się na końcu sieci, po warstwach splotowych, i służą do agregowania informacji uzyskanych z wcześniejszych warstw w celu podjęcia ostatecznej decyzji, jak w klasyfikacji. W przypadku tego modelu, warstwy gęste przekształcają dane wyjściowe z warstw splotowych w wektory, które reprezentują cechy odpowiednie do klasyfikacji.

Użyte funkcje aktywacji:

- ReLU – najczęściej stosowana funkcja aktywacji w sieciach splotowych. Jest to funkcja, która zwraca 0 dla wszystkich ujemnych wartości i zachowuje wartość dodatnią dla wszystkich liczb



RYSUNEK 4.7: Wizualne przedstawienie problemu dot. odbicia lustrzanego

dodatkowych. ReLU pozwala na szybkie i efektywne uczenie się sieci, eliminując problem zanikania gradientu, który występuje w tradycyjnych funkcjach aktywacji, takich jak sigmoidalna.

- Softmax – to funkcja aktywacji, która jest używana w zadaniach klasyfikacyjnych, szczególnie w przypadku, gdy mamy do czynienia z wieloma klasami. Softmax przekształca wyjścia z warstwy gęstej w rozkład prawdopodobieństwa, gdzie suma prawdopodobieństw dla wszystkich klas wynosi 1.

4.3.2 Budowa sieci neuronowej

Model jest wielowarstwową siecią splotową, składającą się z następujących elementów:

1. warstwa wejściowa o rozmiarze 64×64 pikseli i jednym kanale (skala szarości),
2. trzech warstw splotowych z funkcją aktywacji ReLU oraz warstw max poolingu:
 - a) pierwsza warstwa splotowa z 32 filtrami o rozmiarze 3×3 ,
 - b) warstwa max poolingu z jądrem 2×2 ,
 - c) druga warstwa splotowa z 64 filtrami o rozmiarze 3×3 ,
 - d) warstwa max poolingu z jądrem 2×2 ,
 - e) trzecia warstwa splotowa z 128 filtrami o rozmiarze 3×3 ,
 - f) warstwa max poolingu z jądrem 2×2 ,
3. warstwa spłaszczająca (Flatten),
4. dwóch w pełni połączonych warstw (Dense):
 - a) pierwsza warstwa Dense z 128 neuronami i funkcją aktywacji ReLU,
 - b) druga warstwa Dense z 8 neuronami i funkcją aktywacji softmax, przeznaczona do klasyfikacji na 8 klas.

W przypadku zmiany kości na taką z inną liczbą ścianek, np. na również rozważane w fazie koncepcyjnej kości czworościenną, sześciocząsienną, albo szesnastościenną, ostatnia warstwa musiałaby również ulec odpowiedniej zmianie, tak aby liczba neuronów odpowiadała liczbie ścianek na używanej kości.

4.3.3 Trenowanie modelu

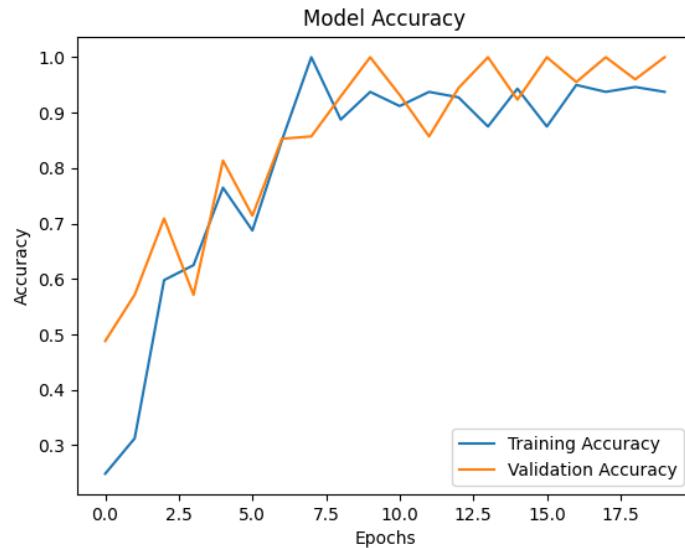
Model został nauczony z wykorzystaniem optymalizatora Adam [24, 23], funkcji straty entropii krzyżowej [29] dostępnej w module tensorflow jako sparse categorical crossentropy [44], oraz metryki dokładności (ang. *accuracy*) [43]. Proces trenowania obejmował dobraną eksperymentalnie liczbę 20 epok, gdyż dalsze zwiększenie liczby epok nie przynosiło istotnej poprawy wydajności, zwiększając jedynie czas obliczeń. Na wejściu model oczekwał obrazu przedstawiającego górną ściankę kości, przekształconego odpowiednio, a więc tak jak opisano w 4.1.

4.3.4 Wyniki

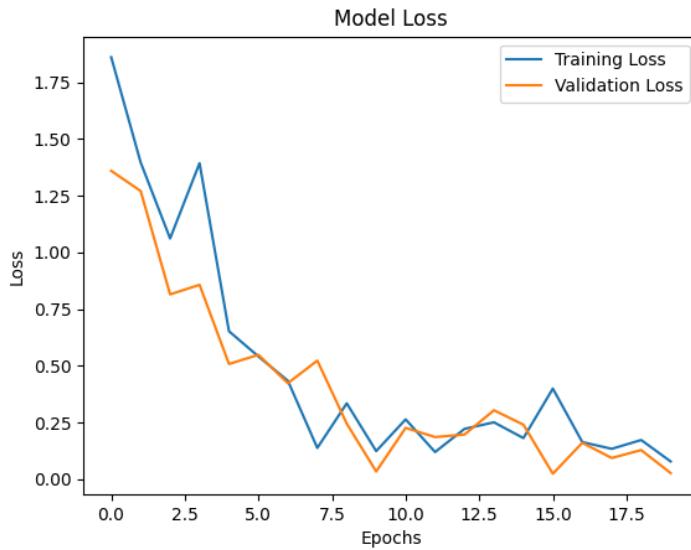
Podczas trenowania osiągnięto następujące końcowe wyniki:

- Dokładność na zbiorze treningowym: 0,9375
- Dokładność na zbiorze walidacyjnym: 1,0
- Strata na zbiorze treningowym: 0,0786
- Strata na zbiorze walidacyjnym: 0,0274

Wyniki zostały również zwizualizowane na wykresach (rys. 4.8 oraz 4.9) przedstawiających zmianę dokładności i straty w trakcie trenowania.



RYSUNEK 4.8: Wykres dokładności modelu.



RYSUNEK 4.9: Wykres straty modelu.

Model został zapisany w formacie `.keras` i jest gotowy do użycia w systemie rozpoznawania liczb na ośmiościennej kości, opisany w kolejnej sekcji.

4.4 Rozpoznawanie liczb

W celu rozpoznawania liczb na zdjęciach przetworzonych przez model napisano funkcję `predict_number`, która przekształca surowy obraz pobrany wcześniej z kamery do odpowiedniego formatu i zwraca przewidywaną klasę. Funkcja działa w następujących krokach, opisanych w kolejnych podrozdziałach:

1. Wczytanie i przetworzenie obrazu.
2. Klasyfikacja obrazu (odczytanie wyniku).
3. Interpretacja wyniku.

4.4.1 Wczytanie i przetworzenie obrazu

W przypadku funkcji dokonującej klasyfikacji na pliku, obraz należy najpierw wczytać. Krok ten jest pomijany, gdy do funkcji zostanie przekazany jako parametr obraz wczytany wcześniej. Obraz wejściowy następnie jest poddawany takim samym transformacjom, jak przy obrazach, na których trenowany był model, a więc skalowany jest na rozmiar zgodny z wymaganiami modelu (64×64 pikseli), zmieniany w skale szarości, oraz normalizowany.

4.4.2 Klasyfikacja obrazu

Przygotowany obraz jest przekazywany do modelu w celu uzyskania wyników klasyfikacji:

```
p = model.predict(obraz_wejsciowy)
klasyfikacja = np.argmax(p, axis=1)
```

Funkcja `np.argmax` zwraca indeks klasy o najwyższym prawdopodobieństwie.

4.4.3 Interpretacja wyniku

Na podstawie indeksu przewidywanej klasy przypisywana jest odpowiadająca mu etykieta (od 1 do 8). Wynikiem funkcji jest etykieta odpowiadająca numerowi na kości, co umożliwia dalsze wykorzystanie tego w systemie.

W dalszym etapie przetwarzania, aby przejść z cyfr w systemie dziesiętnym na ich zapis binarny, wykonywana jest specyficzna transformacja. Zakres [1, 8] da się zapisać za pomocą trzech bitów, co w pełni wykorzystuje dostępny zakres (ta właściwość była również kluczową przesłanką w wyborze kości o dokładnie ośmiu ściankach). Zdecydowano się interpretować ściankę oznaczoną numerem 8 jako cyfrę 0, co oznacza, że w transformacji na zapis binarny, liczba 8 jest interpretowane jako 000, a nie 1000 jak mogłoby się intuicyjnie wydawać. Finalnie, zaetykietowany wynik przetwarzany jest na ciąg trzech bitów, odpowiadających klasie.

4.5 Podsumowanie

Podczas realizacji projektu wytrenowane zostało wiele modeli o różnej architekturze, jednak inne architektury niż przedstawiona w tym rozdziale nie dawały zadowalających wyników. Dopiero opisana architektura oraz modele przyniosły oczekiwane rezultaty. W rozdziale 6 zostały opisane testy na dwóch wyuczonych modelach. Pierwszy z nich nie korzysta w pełni z opisanych metod przetwarzania wstępnego 4.1 i dopiero jego niezadowalające wyniki sprawiły, że do przetwarzania wstępnego dodano krok kadrujący zdjęcie do samej kości, zamiast przekazywać zdjęcie wraz z tłem. Zmiana ta pozwoliła wyodrębnić obszar zainteresowań, co znacznie polepszyło jakość klasyfikacji przez sztuczną inteligencję. Co więcej, drugi z modeli miał do dyspozycji znacznie zasobniejszy zbiór uczący i walidacyjny, co pozwoliło osiągnąć już niemal perfekcyjne wyniki.

Rozdział 5

Komunikacja

Aby móc korzystać z robota, opracowany został protokół komunikacyjny, według którego odbywa się komunikacja, poprzez wysyłanie jednabajtowych znaków poprzez emulowany port szeregowy. Po stronie robota Raspberry Pi wykorzystuje sterownik `g_serial` [3], który odpowiada za tę emulację. Z robotem może komunikować się każde urządzenie z systemem Linux lub Windows.

Komunikacja między robotem i drugim urządzeniem odbywa się za pośrednictwem przewodu USB. Do obsługi komunikacji wykorzystywany jest skrypt wykonany w języku Python, który jest uruchamiany przez `systemd` [26] po uruchomieniu robota. Po uruchomieniu robot czeka na komendę, która wybierze jego tryb pracy. Obecnie do wyboru pracy osługiwane są znaki 'e' i 'd', w przypadku otrzymania innych znaków są one ignorowane, a robot czeka aż otrzyma jeden z obsługiwanych znaków. Po rozpoczęciu jednego z trybów pracy możliwe jest zakończenie tego trybu, wysyłając do robota znak 'q'. Sprawi to, że robot wróci do stanu, w którym czeka na wybranie trybu pracy. Zważywszy na to, że do wykonywania rzutów używana jest kość ośmiościenna, każdy rzut generuje 3 bity, tak jak zostało opisane w rozdziale 4.4.3.

5.1 Tryb pracy

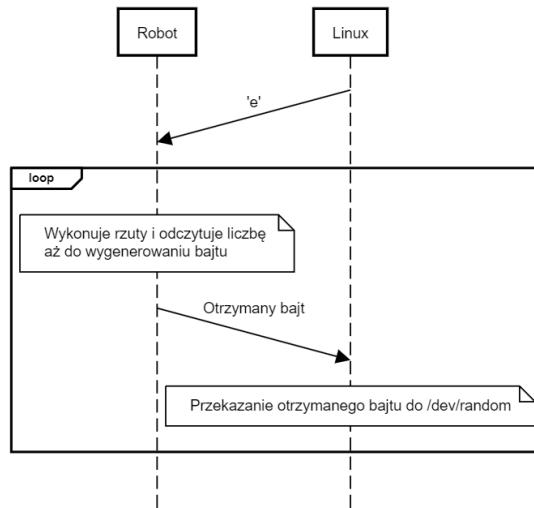
5.1.1 Ciągłe generowanie danych

Aby wykorzystać ten tryb pracy, należy wysłać do robota znak 'e'. W tym trybie zadaniem jest ciągłe generowanie kolejnych bajtów danych. Odbiera się to przez wykonanie następującego algorytmu:

1. Utworzenie zmiennej *num* i przypisanie wartości początkowej 0.
2. Wykonanie rzutu kością.
3. Wykorzystanie modelu opisanego w rozdziale 4 do odczytania liczby i dodanie jej do zmiennej *num*.
4. Przesunięcie bitowe zmiennej *num* w lewo o 3 bity, ponieważ każdy rzut generuje 3 bity.
5. Powtórzenie kroku 1, 2 i 3 aż do wygenerowania co najmniej 8 bitów, ponieważ generując bajty, należy wykonać kolejno 3 rzuty (zostaje 1 bit), 3 rzuty (zostają 2 bity), 2 rzuty (wykorzystujemy bity pozostałe z poprzednich rzutów).
6. Wydobycie bajtu ze zmiennej *num*, wykonując koniunkcję binarną z liczbą 255 i wysłanie go do drugiego urządzenia.

7. Przesunięcie bitowe zmiennej *num* w prawo o 8 bitów, przez co zostanie usunięta część, która została już wysłana.

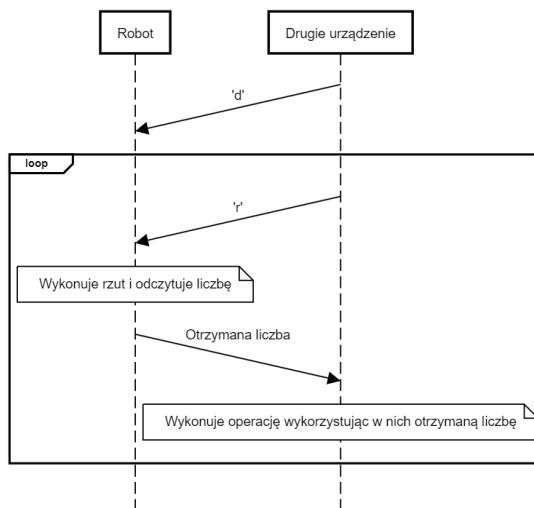
W powyższym algorytmie dane wysyłane są po bajcie danych, ponieważ jest to najmniejsza jednostka, którą można przekazać na raz wykorzystując `pySerial`. Głównym założeniem tego trybu pracy jest wykorzystanie go jako dodatkowe źródło entropii w systemie Linux poprzez wypełnienie `/dev/random`, dla zobrazowania tego wykonany został rys. 5.1. Ten tryb pracy może również zostać wykorzystany w celach kryptograficznych na przykład w trakcie generowania kluczy.



RYSUNEK 5.1: Diagram sekwencji dla ciągłego generowania danych

5.1.2 Wykonywanie pojedyńczych rzutów

Aby wykorzystać ten tryb pracy, należy wysłać do robota przez port szeregowy znak 'd'. W tym trybie zadaniem jest oczekiwanie na polecenie wykonania rzutu, czyli znak 'r'. Innym obsłużonym znakiem jest 'q', który spowoduje powrót do wyboru trybu pracy, pozostałe znaki są ignorowane. Rzut można wykonać również poprzez naciśnięcie przycisku na robocie. Po wykonanym rzucie liczba jest odczytywana i przekazywana do drugiego urządzenia. Następnie otrzymaną liczbę można wykorzystać w swoim programie. Działanie to zobrazowano na rys. 5.2.



RYSUNEK 5.2: Diagram sekwencji dla wykonania rzutu na żądanie

Rozdział 6

Testy

6.1 Użyte testy

Do przetestowania ciągów odczytanych przez model zaprezentowane w rozdziale 4.2 użyto pięć testów statystycznych. Pierwszy z nich to statystyka rozkładu chi-kwadrat, aby zbadać, czy rozkład otrzymywanych wyników jest równomierny. Pozostałe cztery to zestaw podany przez amerykańską normę FIPS-140-2 [32]. Każdy z nich jest przygotowany dla ciągów o długości 20 000 bitów. Poziom istotności tych testów jest równy 0,0001 [25].

W sekcjach 6.1.2 – 6.1.5 zostały opisane testy wskazane w owej normie. Przedstawione w nich stałe są zgodnie z normami przedstawionymi w FIPS 140-2 [32]. Zostały one zaimplementowane w języku Python.

Do przeprowadzenia testów rozważano również wykorzystanie biblioteki *TestU01* [35]. Niestety, do zastosowania nawet najmniejszego z zaproponowanych w niej testów potrzeba przynajmniej 10^6 bitów. Przez wzgląd na ograniczenia czasowe oraz ryzyko nadmiernej eksploatacji robota przy generowaniu tak dużej ilości danych, zrezygnowano z wykorzystania tej biblioteki.

6.1.1 Test chi-kwadrat

Sformuowano następujące hipotezy:

H_0 : Rozkład prawdopodobieństwa wyników jest równomierny.

H_1 : Rozkład prawdopodobieństwa wyników nie jest równomierny.

Aby zweryfikować hipotezę zerową, dla wszystkich n rzutów odczytano, ile razy wypadła każda z k ścian kości (O_i). Wyniki te porównano z oczekiwana liczbą wyrzucenia każdej ze ścianek $E_i = \frac{n}{k}$.

$$\chi^2 = \sum_{i=1}^k \left(\frac{O_i - E_i}{E_i} \right)^2$$

Ponieważ do generowania liczb losowych użyto kości ósmiościennej, to k jest równe 8, co daje wzór:

$$\chi^2 = \sum_{i=1}^8 \left(\frac{O_i - E_i}{E_i} \right)^2$$

Stopień swobody przy $k = 8$ równa się:

$$k - 1 = 8 - 1 = 7$$

Ponieważ w tabeli wartości krytycznych dla rozkładu chi-kwadrat zaprezentowanej przez NIST [33] nie uwzględniono poziomów istotności z rozwinięciem do czterech cyfr po przecinku, przyjęto poziom istotności dla najmniejszego z dostępnych poziomów istotności, czyli równego 0,001. Od-

czytana z tabeli wartość krytyczna wynosi 24,322. Jeśli otrzymana wartość będzie mniejsza od wartości krytycznej, nie ma podstaw do odrzucenia hipotezy zerowej.

6.1.2 Test monobitowy

Test monobitowy bada proporcję między liczbą zer a liczbą jedynek w otrzymanym ciągu bitów. Dla wszystkich wygenerowanych bitów zliczono liczbę jedynek w ciągu (X). Oczekiwana liczba jedynek w ciągu wynosi:

$$9725 < X < 10275$$

Sformuowano następujące hipotezy:

H_0 : Proporcja zer i jedynek jest zgodna z oczekiwana.

H_1 : Proporcja zer i jedynek nie jest zgodna z oczekiwana.

6.1.3 Test serii

Celem testu serii jest zliczenie tak zwanych *serii*, czyli nieprzerwanych ciągów takich samych bitów. Test serii sprawdza, czy ilość serii każdej długości jest zgodna z oczekiwany wartościami oraz bada, czy zmiany między wartościami bitów nie są zbyt częste bądź zbyt rzadkie. Oczekiwane rozkłady wystąpień poszczególnych serii przedstawiono w tabeli 6.1. Sformuowano hipotezy:

H_0 : Rozkład wystąpień serii bitów jest zgodny z przyjętym rozkładem.

H_1 : Rozkład wystąpień serii bitów nie jest zgodny z przyjętym rozkładem.

TABELA 6.1: Oczekiwane liczby wystąpień serii

Długość serii	Przedział
1	2343 - 2657
2	1135 - 1365
3	542 - 708
4	251 - 373
5	111 - 201
6 i więcej	111 - 201

6.1.4 Test długich serii

Test długich serii polega na sprawdzeniu, czy w testowanym ciągu bitów nie ma zbyt wielu występujących pod rząd takich samych bitów. Ciągi 20 000 bitów nie powinny zawierać serii dłuższych niż 25 bitów. Sformuowano następujące hipotezy:

H_0 : Wygenerowany ciąg nie zawiera długiej serii.

H_1 : Wygenerowany ciąg zawiera długą serię.

6.1.5 Test pokerowy

Test pokerowy wykorzystuje statystykę rozkładu chi-kwadrat. Polega na podziale badanego ciągu na segmenty 4-bitowe i zliczeniu liczby wystąpień każdej możliwej z szesnastu kombinacji s_i . Następnie oblicza się statystykę testową:

$$X = \frac{16}{5000} \sum_{i=0}^{15} s_i^2 - 5000$$

Rozkład sekwencji w ciągu jest zgodny z oczekiwany, gdy:

$$2,17 < X < 46,17$$

Sformuowano hipotezy:

H_0 : Rozkład 4-bitowych segmentów w ciągu jest zgodny z oczekiwany.

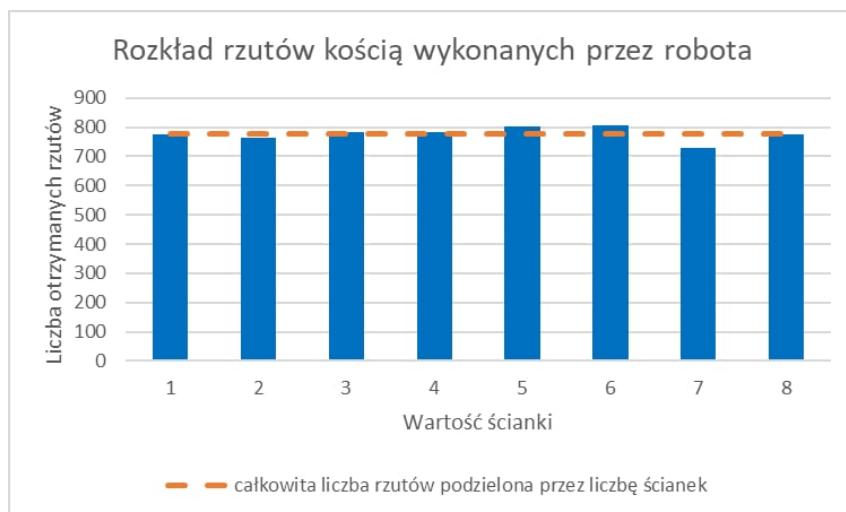
H_1 : Rozkład 4-bitowych segmentów w ciągu nie jest zgodny z oczekiwany.

6.2 Wyniki testów

Przedstawione w sekcji 6.1 testy zostały wykorzystane do sprawdzenia losowości ciągu wygenerowanego przez robota i odczytywane przez oba modele sztucznej inteligencji. Ich wyniki przedstawiono w poniższej sekcji.

Jednakże, przed rozpoczęciem testów statystycznych wyników odczytanych przez model sztucznej inteligencji, postanowiono zbadać rozkład wyników rzutów wykonanych przez robota (przedstawionych na wykresie na rys. 6.1), które zostały ręcznie oznaczone do przykładów uczących dla sieci neuronowej. Miało to na celu sprawdzenie, czy ewentualny nierównomierny rozkład wyników końcowych jest spowodowany wadami maszyny lub nieprawidłowości w bryle kości, czy może problemem jest sam model sztucznej inteligencji.

Wartość statystyki testowej chi-kwadrat wyniosła 5,488, czyli mniej od przyjętej wartości krytycznej. Nie ma zatem podstaw do odrzucenia hipotezy zerowej: rozkład prawdopodobieństwa wyników dla rzutów wykonywanych przez robota jest równomierny.



RYSUNEK 6.1: Wykres straty modelu.

6.2.1 Test chi-kwadrat

Dla odczytanych wyników rzutów kością przez pierwszy model otrzymano wartość statystyki testowej chi-kwadrat równą 246,079, która znacznie przekroczyła przyjętą wartość krytyczną równą 24,322. Odrzucono hipotezę H_0 i przyjęto hipotezę H_1 : rozkład prawdopodobieństwa wyników nie jest równomierny.

Natomiast dla drugiego modelu wartość statystyki testowej chi-kwadrat jest równa 20,072. Jest ona mniejsza od przyjętej wartości krytycznej, zatem nie ma podstaw do odrzucenia hipotezy zerowej: rozkład prawdopodobieństwa wyników jest równomierny.

Otrzymany rozkład dla obu modeli przedstawiono w tabeli 6.2.

TABELA 6.2: Rozkład otrzymanych wyników rzutu kością

Wynik	Liczba otrzymanych rzutów	
	Model 1	Model 2
1	849	870
2	682	841
3	952	838
4	831	819
5	866	836
6	1124	737
7	816	818
8	547	908

6.2.2 Test monobitowy

Dla modelu pierwszego dla otrzymanego ciągu zliczono 10694 bitów o wartości 1. Jest ona większa od oczekiwanej liczby jedynek. Hipotezę zerową odrzucono. Natomiast dla modelu drugiego zliczono 9805 bitów o wartości jeden. Jest to liczba zgodna z oczekiwaniem. Brak podstaw do odrzucenia hipotezy zerowej dla modelu drugiego.

6.2.3 Test serii

Wyniki testów serii dla obu modeli przedstawiono w tabeli 6.3. Rozkład serii w badanym ciągu odczytanym przez pierwszy model nie jest zgodny z oczekiwany dla obu serii o długości 1, obu serii o długości 2, serii jedynek o długości 4 i serii zer o długości 6 lub więcej. Hipotezę zerową dla modelu pierwszego odrzucono.

Dla modelu drugiego rozkłady wszystkich serii są zgodne z oczekiwany. Brak podstaw do odrzucenia hipotezy zerowej.

TABELA 6.3: Liczby wystąpień serii w ciągu

Długość serii	Przedział	Model 1		Model 2	
		Serie zer	Serie jedynek	Serie zer	Serie jedynek
1	2343 - 2657	2717	2303	2405	2477
2	1135 - 1365	1367	1141	1212	1250
3	542 - 708	559	678	612	593
4	251 - 373	263	374	338	290
5	111 - 201	113	158	162	155
6 i więcej	111 - 201	82	177	194	159

6.2.4 Test długich serii

W wygenerowanym ciągu odczytanym przez model pierwszy zarówno dla zer, jak i dla jedynek, najdłuższa znaleziona seria zawierała 13 bitów. Jest to liczba mniejsza od maksymalnej przyjętej długości najdłuższego ciągu bitów wynoszącej 25. Nie ma podstaw do odrzucenia hipotezy zerowej dla modelu pierwszego.

Natomiast dla modelu drugiego najdłuższy nieprzerwany ciąg zer miał długość 17, a dla jedynek 16. To również są mniejsze wartości od maksymalnych przyjętych, zatem nie ma podstaw do odrzucenia hipotezy zerowej dla modelu drugiego.

6.2.5 Test pokerowy

Dla testu pokerowego dla modelu pierwszego otrzymano statystykę testową równą 120,026. Jest ona większa od oczekiwanej, zatem odrzucono hipotezę zerową. Dla modelu drugiego staty-

styka testowa jest równa 28,794. Jest ona zgodna z wartością oczekiwana, zatem nie ma podstaw do odrzucenia hipotezy zerowej. Rozkład segmentów w ciągach odczytanych przez oba modele przedstawiono w tabeli 6.4.

TABELA 6.4: Liczby wystąpień kombinacji bitów

Kombinacja bitów	Liczba wystąpień	
	Model 1	Model 2
0000	197	361
0001	249	310
0010	277	347
0011	312	317
0100	279	246
0101	309	280
0110	363	283
0111	338	294
1000	255	321
1001	355	276
1010	311	298
1011	355	322
1100	322	320
1101	335	297
1110	366	318
1111	377	310

6.3 Wnioski

Dla pierwszego modelu odrzucono hipotezę zerową dla czterech z pięciu testów. W wynikach testu statystyki rozkładu chi-kwadrat (sekcja 6.2.1) można zauważać, że liczba 6 jest odczytywana ponad dwa razy częściej niż liczba 8. Ma to bardzo duży wpływ na otrzymany ciąg bitów, ponieważ liczba 6 jest interpretowana jako 110, a liczba 8 jako 000, co można zaobserwować w wynikach pozostałych testów. Sprawia to, że jedynki pojawiają się częściej w ciągu niż zera. Ukaże to już test monobitowy (sekcja 6.2.1), gdzie liczba jedynek w ciągu jest większa niż oczekiwana. Podobny problem można zaobserwować w wynikach testu pokerowego (sekcja 6.2.5), gdzie segmenty zawierające trzy lub cztery bity o wartości 1 występują znacznie częściej niż segmenty zawierające więcej bitów o wartości 0.

Przez wzgląd na niezadowalające wyniki modelu pierwszego, dokonano jego poprawy (sekcja 4.5) i utworzono drugi model sztucznej inteligencji. Dla wszystkich przeprowadzonych testów nie było podstaw do odrzucenia hipotezy zerowej. W przeciwieństwie do pierwszego modelu, najczęściej odczytywanym wynikiem jest 8, a najrzadziej – 6. Różnica w liczbie odczytów tych wyników nie jest aż tak znacząca (737 dla 6 i 908 dla 8) jak dla pierwszego modelu (1124 dla 6 i 547 dla 8). Częstsze wystąpienia wyniku 8 widać w teście monobitowym, gdzie zliczonych zer jest więcej niż jedynek. Dysproporcja między nimi jest niewielka, wynosi zaledwie niecałe 400 bitów, kiedy dla pierwszego modelu różnica wynosiła prawie 1400 bitów. Podobnie dla testu serii liczby poszczególnych serii dla zer i jedynek są dużo bardziej zrównoważone dla drugiego modelu niż pierwszego.

Dla testów długich serii dla obu modeli nie było podstaw do odrzucenia hipotezy zerowej. Dla pierwszego z nich najdłuższe ciągi dla obu wartości bitów miały długość 13, a dla drugiego 17 (zera) i 16 (jedynki). Ponieważ bity są generowane trójkami przy każdym rzucie kością, oznacza to, że dla wartości 7 (kodowanej jako 111) i 8 (kodowanej jako 000) odczytano maksymalnie po

cztery razy pod rząd dla modelu pierwszego i maksymalnie po pięć razy pod rząd dla modelu drugiego.

Wyniki dla drugiego modelu są zadowalające, ponieważ nie ma dla nich przesłanek do odrzucenia żadnej z hipotez zerowych. Jednakże w przyszłości możliwa jest implementacja lepszego modelu, który potencjalnie mógłby otrzymywać jeszcze lepsze wyniki (rozdział 7), ponieważ wartość statystyki testowej chi-kwadrat dla modelu drugiego jest niemal czterokrotnie większa, niż dla wartości statystyki testowej chi-kwadrat otrzymanej dla ręcznie oznaczone do przykładów uczących dla sieci neuronowej. Prowadzi to do wniosku, że uzyskanie prawdziwej losowości jest możliwe, mimo deterministycznego charakteru działania maszyny.

Rozdział 7

Zakończenie

W ramach realizacji projektu udało się spełnić jego główny cel, jakim było stworzenie sprzętowego generatora liczb losowych. Wykorzystuje on rzut kością, a wynik jest odczytywany i interpretowany przez model sztucznej inteligencji. Przeprowadzono testy statystyczne, badające losowość otrzymanych wyników. Pokazały one, że możliwe jest uzyskanie prawdziwej losowości zautomatyzowanego rzutu kością, mimo deterministycznego działania wykonującego maszyny. Sumarycznie, w czasie realizacji projektu, robot wykonał ponad 30 000 rzutów kością.

Mimo osiągnięcia celów projektu, możliwe jest wprowadzenie wielu usprawnień, które poprawią jego jakość i wygodę użytkownika. Korzystnym ulepszeniem byłoby zintegrowanie maszyny nie tylko z systemem Linux, ale również z popularnym systemem Windows.

Rozwiązaniem, które na pewno zachęciłoby wielu fanów gier planszowych i RPG, byłoby rozszerzenie modelu sztucznej inteligencji o odczytywanie kości w innych wzorach i kolorach, a także o innych ilościach ścianek. Wprowadziwszy te ulepszenia, możliwe będzie zbieranie statystyk używanej kości, co pozwoli na wykrycie potencjalnych nieprawidłowości w jej budowie, które mogą zaburzać rozkład generowanych przez nią wyników.

Co więcej, można również usprawnić wykorzystywane metody przetwarzania zdjęć oraz uzyskać „czystsze” zdjęcia wejściowe dla sieci neuronowej. Wśród rozważanych sposobów na uzyskanie tego efektu znajdują się dokładniejsze wycięcie tła czy poprawienie hardware poprzez np. wprowadzenie kamery o małej ogniskowej lub lepsze oświetlenie dna kubka.

Innym ulepszeniem, które zwiększyłoby możliwości robota jest nauczenie modelu sztucznej inteligencji odczytywania wyników kilku kości jednocześnie lub zaimplementowanie przetwarzania, które pozwoliłoby na odczytywanie wyników z dwóch kości. Pozwoliłoby to na generowanie kilkukrotnie większej liczby bitów w tym samym czasie. Dzięki temu udałoby się chociaż częściowo zmniejszyć największą wadę zrealizowanego projektu, jaką jest wolne generowanie nowych liczb. Jednakże to usprawnienie wiąże się z trudnością, wykonywania samego rzutu kilkoma kości mi czy innymi problemami takimi jak kości potencjalnie upadające na siebie, uniemożliwiając efektywny odczyt.

Wśród innych planów na najbliższą przyszłość znajduje się również dodanie zastosowania dla przycisku robota. Wśród rozważanych funkcjonalności znajduje się przełączanie między trybami robota (opisanymi w rozdziale 5.1) bądź wykonywanie pojedynczych rzutów kością.

W przypadku komercjalizacji produktu należy rozważyć również wymianę komputera Raspberry Pi na tańszy model lub stworzenie dedykowanego rozwiązania. Taka zmiana pozwoliłaby obniżyć cenę komercjalizowanego produktu, a sama maszyna mogłaby znaleźć zastosowanie nie tylko w kryptografii, ale również wśród fanów gier planszowych oraz RPG, gdzie mogłaby znaleźć zastosowanie jako urządzenie generujące sprawiedliwe rzuty kością dla graczy chcących grać zdalnie.

za pośrednictwem sieci.

To jedynie część z pomysłów na dalsze rozwijanie projektu, które narodziły się w trakcie kolejnych etapów jego realizacji. Wykonała go czteroosobowa grupa, dzięki owocnej współpracy i jasnemu podziałowi zadań. Finalnie, projekt został ukończony z pełnym sukcesem, a wszystkie jego założenia zostały zrealizowane.

W pracy zostały wykorzystane generatywne modele SI, przy sprawdzaniu poprawności składowej i językowej zdań oraz generowaniu drobnych fragmentów wykorzystanego kodu.

Literatura

- [1] Katarzyna Stąpor Adam Świtowski. Operacje morfologiczne na obrazach w odcieniach szarości – zastosowanie na potrzeby wizji komputerowej. *Studia Informatica*, 25(2), 2004.
- [2] Juan Claver Adrián Rodríguez-Panes and Ana María Camacho. The influence of manufacturing parameters on the mechanical behaviour of pla and abs pieces manufactured by fdm: A comparative analysis. *MDPI Materials*, 2018.
- [3] Al Borchers. Linux Gadget Serial Driver v2.0. [on-line]
https://docs.kernel.org/usb/gadget_serial.html, 2008. Dostęp: 24 stycznia 2025.
- [4] Botland. silnik 65x26mm 5v z przekładnią 48:1. [on-line]
<https://botland.com.pl/silniki-dc-katowe-z-przekladnia/3696-kolo-silnik-65x26mm-5v-z-przekladnia-48-1-5903351248129.html>. Dostęp: 20 stycznia 2025.
- [5] Prusa Research by Joseph Prusa. Petg. [on-line]
https://help.prusa3d.com/pl/article/petg_2059, 2022. Dostęp: 20 stycznia 2025.
- [6] Prusa Research by Joseph Prusa. Pla. [on-line] https://help.prusa3d.com/pl/article/pla_2062, 2022. Dostęp: 20 stycznia 2025.
- [7] Prusa Research by Joseph Prusa. Abs. [on-line] https://help.prusa3d.com/pl/article/abs_2058, 2023. Dostęp: 20 stycznia 2025.
- [8] Carsten Röcker Carsten Magerkurth, Timo Engelke. The smart dice cup: A radio controlled sentient interaction device. [on-line] <https://dl.ifip.org/db/conf/iwec/icec2006/MagerkurthER06.pdf>, 2006. Dostęp: 26 stycznia 2025.
- [9] Cloudflare. Lavarand in production – the nitty-gritty technical details, 2017. <https://blog.cloudflare.com/lavarand-in-production-the-nitty-gritty-technical-details/>
Dostęp: 20 stycznia 2025.
- [10] Cloudflare. League of entropy: Not all heroes wear capes, 2019.
<https://blog.cloudflare.com/league-of-entropy/> Dostęp: 2025-01-22.
- [11] Intel Corporation. Intel® digital random number generator (drng). [on-line]
<https://www.intel.com/content/www/us/en/developer/articles/guide/intel-digital-random-number-generator-drng-software-implementation-guide.html>, 2023.
Dostęp: 24 stycznia 2025.
- [12] G. Deng and L.W. Cahill. An adaptive gaussian filter for noise reduction and edge detection. In *1993 IEEE Conference Record Nuclear Science Symposium and Medical Imaging Conference*, pages 1615–1619 vol.3, 1993.
- [13] Keras Developers. Imagedatagenerator: Keras preprocessing layers. [on-line]
<https://keras.io/api/preprocessing/image/>, 2025. Dostęp: 20 stycznia 2025.
- [14] Keras Developers. Keras documentation. [on-line] <https://keras.io/>, 2025. Dostęp: 20 stycznia 2025.

- [15] TensorFlow Developers. Tensorflow documentation. [on-line] <https://www.tensorflow.org/>, 2025. Dostęp: 20 stycznia 2025.
- [16] OpenCV Documentation. Opencv: Open source computer vision library, 2025. <https://docs.opencv.org/> Dostęp: 20 stycznia 2025.
- [17] Pillow Documentation. Pillow — python imaging library (pil fork), 2025. Dostęp: 20 stycznia 2025.
- [18] James Gleick. *Chaos. Making a New Science*. Zysk i S-ka, 1987.
- [19] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. <http://www.deeplearningbook.org>.
- [20] gryplanszowe.pl. [on-line] <https://gryplanszowe.pl/search.php?text=kubek+do+gry>. Dostęp: 24 stycznia 2025.
- [21] Peter Höjerslev. Raspberry pi camera. [on-line] <https://grabcad.com/library/raspberry-pi-camera-4>, 2019. Dostęp: 24 stycznia 2025.
- [22] ID Quantique. Id quantique: Quantum random number generators. [on-line] <https://www.idquantique.com/random-number-generation/overview/>, 2025. Dostęp: 24 stycznia 2025.
- [23] Keras. Adam optimizer. [on-line] https://www.tensorflow.org/api_docs/python/tf/keras/optimizers/Adam, 2025. Dostęp: 20 stycznia 2025.
- [24] D.P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.
- [25] Zbigniew Kotulski. *Matematyka Stosowana*. Polskie Towarzystwo Matematyczne, 2001.
- [26] Społeczność open source Lennart Poettering. Systemd documentation. [on-line] <https://systemd.io>, 2024. Dostęp: 25 stycznia 2025.
- [27] Wenjie Liu. Fabrication of pla filaments and its printable performance. *IOP Conference Series: Materials Science and Engineering*, 275(012033), 2017.
- [28] Microchip Technology. Microchip technology: Fips 140-3 random number generator esvts ready. [on-line] <https://www.microchip.com/en-us/product/rng90#Documentation>, 2025. Dostęp: 24 stycznia 2025.
- [29] K.P. Murphy. *Machine Learning: A Probabilistic Perspective*. Adaptive Computation and Machine Learning series. MIT Press, 2012.
- [30] Distributed Randomness Beacon Network. drand documentation, 2025. <https://drand.love> Dostęp: 2025-01-22.
- [31] Landon Curt Noll, Robert G. Mende, and Sanjeev Sisodiya. Method for seeding a pseudo-random number generator with a cryptographic hash of a digitization of a chaotic system, 1998. <https://patents.google.com/patent/US5732138> Dostęp: 2025-01-22.
- [32] National Institute of Standards and Technology. Security requirements for cryptographic modules. [on-line] <https://csrc.nist.gov/pubs/fips/140-2/upd2/final>, 2001. Dostęp: 20 grudnia 2024.
- [33] National Institute of Standards and Technology. Critical values of the chi-square distribution. [on-line] <https://www.itl.nist.gov/div898/handbook/eda/section3/eda3674.htm>, 2023. Dostęp: 20 stycznia 2025.
- [34] Raspberry Pi. Add heat sinks. [on-line] <https://www.raspberrypi.com/documentation/computers/raspberry-pi.html#add-heat-sinks>. Dostęp: 24 stycznia 2025.

- [35] Richard Simard Pierre L'Ecuyer. *TestU01. A Software Library in ANSI C for Empirical Testing of Random Number Generators*. D'epartement d'Informatique et de Recherche Op'erationnelle Universit'e de Montr'eal, 2013.
- [36] Raspberry Pi. Raspberry pi hardware. [on-line]
<https://www.raspberrypi.com/documentation/computers/raspberry-pi.html#flagship-series>. Dost  p: 20 stycznia 2025.
- [37] J. B. Rhine. Dice thrown by cup and machine in pk tests. *Journal of Parapsychology*, 7(3), 1943.
- [38] David Scoggins. L298n stepper driver. [on-line]
<https://grabcad.com/library/l298n-stepper-driver-1>, 2024. Dost  p: 24 stycznia 2025.
- [39] Semtech Corporation. Semtech: Random numbers and lora. [on-line] <https://learn.rakwireless.com/hc/en-us/articles/26580705507607-Random-Numbers-and-LoRa>, 2025. Dost  p: 24 stycznia 2025.
- [40] Hasanain Shuja. Raspberry pi 4 model b. [on-line]
<https://grabcad.com/library/raspberry-pi-4-model-b-1>, 2019. Dost  p: 24 stycznia 2025.
- [41] STMicroelectronics. Eight darlington arrays. [on-line]
<https://www.st.com/en/interfaces-and-transceivers/uln2803a.html>, 2018. Dost  p: 22 stycznia 2025.
- [42] STMicroelectronics. Dual full bridge driver. [on-line]
<https://www.st.com/en/motor-drivers/l298.html>, 2023. Dost  p: 22 stycznia 2025.
- [43] TensorFlow. Accuracy metric. [on-line]
https://www.tensorflow.org/api_docs/python/tf/keras/metrics/Accuracy, 2025. Dost  p: 20 stycznia 2025.
- [44] TensorFlow. Sparse categorical crossentropy loss. [on-line] https://www.tensorflow.org/api_docs/python/tf/keras/losses/SparseCategoricalCrossentropy, 2025. Dost  p: 20 stycznia 2025.
- [45] Witold Zawadzki. Momenty bezw  adno  ci bez ca  ek. *Foton*, (107), 2009.



© 2025 Julia Samp, Jakub Kędra, Wojciech Kot, Jakub Prusak

Instytut Informatyki, Wydział Informatyki i Telekomunikacji
Politechnika Poznańska

Skład przy użyciu systemu L^AT_EX – TeX Live oraz workflow za pomocą Github Actions.