



POLITECHNIKA POZNANSKA

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
2	Fragment teoretyczny	3
2.1	Przegląd komercyjnych rozwiązań TRNG	3
2.1.1	Rozwiązania sprzętowe	3
2.1.2	Generatory TRNG w chmurze	4
2.2	Wady dostępnych rozwiązań TRNG	5
2.2.1	Wydajność i szybkość generowania liczb losowych	5
2.2.2	Koszty implementacji	5
2.2.3	Stabilność i jakość generowanych liczb losowych	5
2.2.4	Skomplikowana kalibracja i konserwacja	5
2.2.5	Złożoność integracji z istniejącymi systemami	5
2.2.6	Ograniczona dostępność i skalowalność	6
2.2.7	Zagadnienia związane z bezpieczeństwem	6
2.2.8	Podsumowanie wad TRNG	6
2.2.9	Podsumowanie dostępnych na rynku TRNG	6
2.3	Rozwiązania podobne do naszego	6
3	Budowa sprzętowego generatora liczb losowych	7
3.1	Projektowanie robota	7
3.1.1	Prototypowanie	7
3.1.2	Ostateczna wersja robota	12
	Opis komponentów ostatecznej wersji robota	13
3.2	Dokumentacja techniczna	17
3.2.1	Hardware	17
	Ogólne parametry robota:	17
	Komponenty:	17
	Diagramy i schematy:	18
3.2.2	Software	19
4	Odczytywanie losowego wyniku z kości	20
4.1	Przetwarzanie wstępne obrazów	21
4.1.1	Algorytm	21
4.1.2	Zidentyfikowane trudności i ich rozwiązania	23
4.1.3	Podsumowanie	25
4.2	Opis Modelu SI	25
4.2.1	Przygotowanie danych	25
4.2.2	Architektura modelu	26

4.2.3	Użyte funkcje aktywacji	26
4.2.4	Trenowanie modelu	26
4.2.5	Wyniki	26
4.2.6	Podsumowanie	27
4.3	Funkcja predict	28
4.3.1	Wczytanie i przetworzenie obrazu	28
4.3.2	Predykcja klasy	28
4.3.3	Interpretacja wyniku	28
5	Przetwarzanie wyniku	29
6	Testy	30
6.1	Użyte testy	30
6.1.1	Test chi-kwadrat	30
6.1.2	Test monobitowy	31
6.1.3	Test serii	31
6.1.4	Test długich serii	31
6.1.5	Test pokerowy	31
6.2	Wyniki testów	32
6.2.1	Test chi-kwadrat	32
6.2.2	Test monobitowy	32
6.2.3	Test serii	32
6.2.4	Test długich serii	33
6.2.5	Test pokerowy	33
6.2.6	Wnioski	33
7	Zakończenie	34
Literatura		35

Rozdział 1

Wstęp

Współczesne systemy komputerowe i technologie wykorzystują liczne algorytmy generowania liczb losowych, które stanowią podstawę w takich dziedzinach jak kryptografia, symulacje komputerowe 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 nieprzewidywalnością w sytuacjach, gdzie prawdziwa losowość jest kluczowa.

Inspiracją do podjęcia tematu budowy sprzętowego generatora liczb losowych było połączenie zainteresowań w dziedzinie techniki oraz pasji do gier RPG i planszowych. Gry te, od wieków korzystające z kostek jako narzędzia generowania losowości, stanowią idealny przykład zastosowania fizycznych mechanizmów do uzyskiwania nieprzewidywalnych wyników. Projekt bazuje na hipotezie, że poprzez zautomatyzowanie procesu rzutu kością możliwe jest uzyskanie prawdziwej losowości, mimo deterministycznego charakteru działania maszyny.

Celem pracy jest zbudowanie urządzenia zdolnego do generowania liczb losowych poprzez mechaniczny rzut kością oraz analiza stopnia losowości uzyskanych wyników. Projekt ten zakłada stworzenie 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 ocenę możliwości praktycznego zastosowania takiego generatora jako źródła entropii w systemie Linux.

Realizacja takiego urządzenia pozwoli nam także w prosty sposób zautomatyzować statystyczne badanie różnych kości do gry pod względem sprawdzania ich potencjalnej nieuczciwości.

Realizacja tego projektu pozwoli nie tylko na sprawdzenie skuteczności takiego podejścia, 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.

— TODO —

Struktura pracy jest następująca.

W rozdziale 2 przedstawiono

Rozdział 3 jest poświęcony

Rozdział 4 zawiera

— TODO —

ej musimy we wstępie gdzieś wcisnąć że einstein tez uważały że rzuty kością są losowe bo powiedział Bog nie rzuca kości mi

Jakub Kędra

Podział zadań wśród autorów pracy był następujący:

- Jakub Kędra zaprojektował i wykonał część sprzętową.
- Wojciech Kot zaprojektował i zaimplementował OCR i część związaną z siecią neuronową. Jest głównym pomysłodawcą projektu.
- Julia Samp przygotowała, przeprowadziła oraz opracowała testy statystyczne otrzymanych wyników.
- Jakub Prusak wykonał integrację maszyny z systemem Linux.

Rozdział 2

Fragment teoretyczny

— TODO —

Co i dlaczego tak w sumie budujemy?
może coś o kostkach, w tym konwencja nazywania kostek
czemu k8 zamiast tradycyjnej k6
dlaczego k8 a nie d8?

2.1 Przegląd komercyjnych rozwiązań TRNG

Współczesne systemy kryptograficzne oraz aplikacje wymagają generowania liczb losowych, które muszą charakteryzować się wysoką jakością i odpornością na przewidywalność. Najlepiej spełniają do generatory liczb losowych oparte na fizycznych zjawiskach, zwane TRNG (True Random Number Generator). TRNG są szczególnie istotne w kontekście aplikacji wymagających silnej ochrony danych, takich jak systemy kryptograficzne, podpisy cyfrowe, oraz w generowaniu kluczy szyfrujących.

2.1.1 Rozwiązania sprzętowe

Wśród komercyjnych rozwiązań sprzętowych, wiodącymi producentami są firmy takie jak **ID Quantique**, **Microchip Technology** i **Semtech**, które oferują zaawansowane urządzenia bazujące na TRNG. Produkty te zapewniają wysoki poziom bezpieczeństwa i są stosowane w wymagających aplikacjach, takich jak bankowość elektroniczna czy systemy wojskowe.

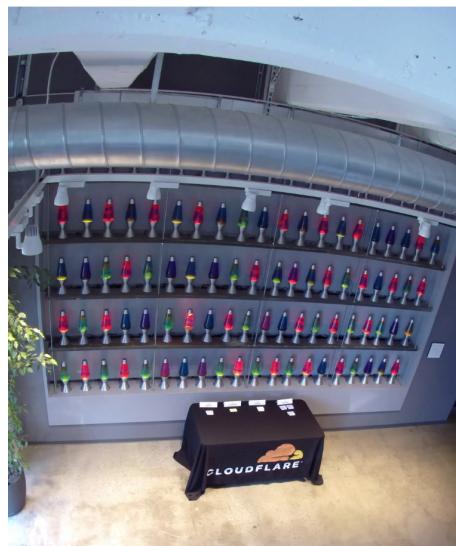
- **ID Quantique** jest jednym z liderów w dziedzinie generatorów liczb losowych opartych na technologii fotoniki. Firma oferuje urządzenia, które wykorzystują detekcję fotonów w celu generowania liczb losowych. Dzięki temu rozwiązania ID Quantique charakteryzują się bardzo wysoką jakością losowości, 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 obwodach analogowych do generowania liczb losowych. Firma ta jest jednym z głównych

dostawców układów TRNG, które znajdują szerokie zastosowanie w urządzeniach IoT oraz w systemach komunikacji bezprzewodowej.

2.1.2 Generatory TRNG w chmurze

W ostatnich latach pojawiły się także rozwiązania chmurowe, które umożliwiają generowanie liczb losowych w czasie rzeczywistym bez potrzeby posiadania własnego sprzętu. 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. Cloudflare korzysta z technologii opartych na zjawiskach fizycznych, takich jak „Entropy Wall”, czyli seed'owanie PRNG za pomocą danych pochodzących z kamery, ze zdjęć ściany lamp lawowych*, nieprzewidywalnego fizycznie trójstopniowego wahadła, rozpadu radioaktywnego Urana oraz procesów związane z tzw. „chaosem kwantowym”. Tego typu rozwiązania pozwalają na szybkie i bezpieczne generowanie liczb losowych w skali globalnej, zapewniając jednocześnie wysoki poziom ochrony przed atakami.

*LavaRand pozostaje dzisiaj jedynie dodatkowym zabezpieczeniem firmy, na wypadek gdyby ich konwencjonalne źródła entropii okazały się niewystarczające, jednak wciąż pozostaje inspirującym przykładem do szukania losowości wszędzie wokół nas.



RYSUNEK 2.1: Widok z kamery w biurze Cloudflare

Źródło: <https://blog.cloudflare.com/lavarand-in-production-the-nitty-gritty-technical-details/> TODO - dodać do bibtex'a ;)

Firma ta 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 Cloudflare generowane liczby losowe są szeroko dostępne i charakteryzują się dużą niezawodnością oraz odpornością na ataki.

—TODO—

Zrobić mniej poszatkowane

idk kto to zrobi najlepiej, ale w wolnej chwili zerknijcie tu proszę

I TAK ZOSTAWIAM TO W PDF, ŻE BYŚCIE PRZEGLĄDAJĄC NA TO TRAFILI!

2.2 Wady dostępnych rozwiązań TRNG

Mimo że Generatory Liczb Losowych Oparte na Zjawiskach Fizycznych (TRNG) oferują wysoki poziom bezpieczeństwa i losowości, istnieje kilka istotnych wad i ograniczeń związanych z ich implementacją i użytkowaniem. Wśród głównych problemów, które mogą wpływać na wydajność oraz niezawodność TRNG, wyróżnia się następujące aspekty:

2.2.1 Wydajność i szybkość generowania liczb losowych

Wydajność TRNG jest często niższa niż w przypadku Generujących Liczby Losowe Oparte na Algorytmach (PRNG). Generowanie liczb losowych za pomocą zjawisk fizycznych, takich jak szum termiczny czy fluktuacje kwantowe, może być procesem czasochłonnym, szczególnie w systemach wymagających dużych ilości losowych liczb w krótkim czasie. W wyniku tego, TRNG mogą nie spełniać wymagań wydajnościowych w aplikacjach o dużym zapotrzebowaniu na losowość, takich jak systemy kryptograficzne o bardzo wysokiej częstotliwości operacji.

2.2.2 Koszty implementacji

Urządzenia TRNG, zwłaszcza te oparte na zaawansowanych technologiiach, takich jak fotonika kwantowa czy detekcja szumów kwantowych, mogą wiązać się z wysokimi kosztami produkcji oraz utrzymania. Z tego powodu TRNG są często droższe w porównaniu do bardziej ekonomicznych rozwiązań opartych na algorytmach deterministycznych (PRNG), które wystarczają do wielu zastosowań, gdzie wysoka jakość losowości nie jest kluczowym wymaganiem.

2.2.3 Stabilność i jakość generowanych liczb losowych

Choć TRNG są uznawane za bezpieczne, ich jakość może być wpływana przez różne czynniki zewnętrzne, takie jak temperatura, zakłócenia elektromagnetyczne czy inne zmiany środowiskowe. W wyniku tego, generowane liczby losowe mogą wykazywać pewne niskiej jakości właściwości, co wymaga zastosowania dodatkowych mechanizmów, takich jak procesy post-przetwarzania, aby zapewnić ich odpowiednią losowość. Nawet małe zakłócenia w systemie mogą prowadzić do wzorców, które mogą zostać wykorzystane w atakach kryptograficznych.

2.2.4 Skomplikowana kalibracja i konserwacja

Urządzenia TRNG, szczególnie te, które wykorzystują skomplikowane zjawiska fizyczne, wymagają starannej kalibracji i ciągłego monitorowania, aby zapewnić ich prawidłowe funkcjonowanie. W wielu przypadkach konieczne jest stosowanie systemów nadzoru, które monitorują jakość generowanych liczb losowych w czasie rzeczywistym. Ponadto, wymogi dotyczące utrzymania odpowiednich warunków pracy, takich jak stabilna temperatura czy brak zakłóceń elektromagnetycznych, mogą stanowić dodatkową przeszkodę w ich szerokim zastosowaniu.

2.2.5 Złożoność integracji z istniejącymi systemami

Integracja TRNG z już działającymi systemami, zwłaszcza w kontekście urządzeń wbudowanych lub systemów o dużych wymaganiach obliczeniowych, może wiązać się z wieloma trudnościami. Często konieczne jest dostosowanie sprzętu lub oprogramowania w celu zapewnienia kompatybilności i pełnej funkcjonalności. Dodatkowo, ze względu na fizyczną naturę tych urządzeń, integracja z innymi komponentami może prowadzić do wzrostu kosztów oraz złożoności całego systemu.

2.2.6 Ograniczona dostępność i skalowalność

Choć rynek TRNG rośnie, nadal jest on stosunkowo niszowy w porównaniu do bardziej po-wszechnych rozwiązań opartych na PRNG. Ograniczona dostępność wyspecjalizowanych urządzeń TRNG, szczególnie tych, które oferują wysoką jakość generowanych liczb losowych, sprawia, że ich wdrożenie jest trudniejsze, zwłaszcza w przypadkach wymagających masowej produkcji. Ponadto, nie wszystkie rozwiązania są skalowalne, co może być problemem w przypadku aplikacji wymagających elastyczności i łatwego dostosowywania wydajności do rosnących potrzeb.

2.2.7 Zagadnienia związane z bezpieczeństwem

Chociaż TRNG zapewniają wyższy poziom bezpieczeństwa niż PRNG, nie są one całkowicie odporne na ataki. Ataki fizyczne, takie jak manipulacje w obrębie urządzenia lub przechwytywanie sygnałów z jego elementów, mogą prowadzić do kompromitacji jakości liczb losowych. Ponadto, w przypadku rozwiązań opartych na technologii chmurowej, takich jak te oferowane przez Cloudflare, istnieje ryzyko ataków związanych z przechwytywaniem lub manipulowaniem danymi w trakcie transmisji, co może wpływać na bezpieczeństwo generowanych liczb losowych.

2.2.8 Podsumowanie wad TRNG

Chociaż TRNG oferują niezrównaną jakość losowości w porównaniu do rozwiązań opartych na algorytmach, posiadają również szereg wad, które mogą ograniczać ich szerokie zastosowanie. Należy do nich niska wydajność, wysokie koszty implementacji, problemy ze stabilnością generowanych liczb losowych, złożoność integracji z systemami oraz ryzyko związane z bezpieczeństwem. W miarę jak technologia będzie się rozwijać, możliwe jest, że te ograniczenia zostaną przezwyciężone, jednak obecnie stanowią one istotne wyzwanie dla szerokiego przyjęcia TRNG w różnych aplikacjach.

2.2.9 Podsumowanie dostępnych na rynku TRNG

2.3 Rozwiązania podobne do naszego

— TODO —

Tutaj rozpiszemy to co dopiero niedawno trafiło na issues, czyli to jak ktoś zrobił elektroniczny smart-dice, oraz to jak ktoś zrobił to co nazwaliśm y wariantem "betoniarka" ale z 60 lat temu ;)

Rozdział 3

Budowa sprzętowego generatora liczb losowych

3.1 Projektowanie robota

3.1.1 Prototypowanie

Przy całym procesie budowy robota wykorzystano technologię druku 3D. Pozwala na szybkie modyfikacje przy jednoczesnym zachowaniu bardzo wysokiej dokładności i jakości wykonania elementów. Dzięki temu podczas budowy prototypów a później ostatecznej wersji robota można było w prosty sposób modyfikować, projektować i zmieniać każdy element składowy robota. Pomimo tego, że proces druku 3D w szczególności dużych skomplikowanych elementów jest stosunkowo czasochłonny to jest to najlepsza dostępna nam technologia do wykorzystania w tego rodzaju projektach.

Główym 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 znaczco prostsze i przede wszystkim szybsze niż gdyby cały robot był jednolitą bryłą.

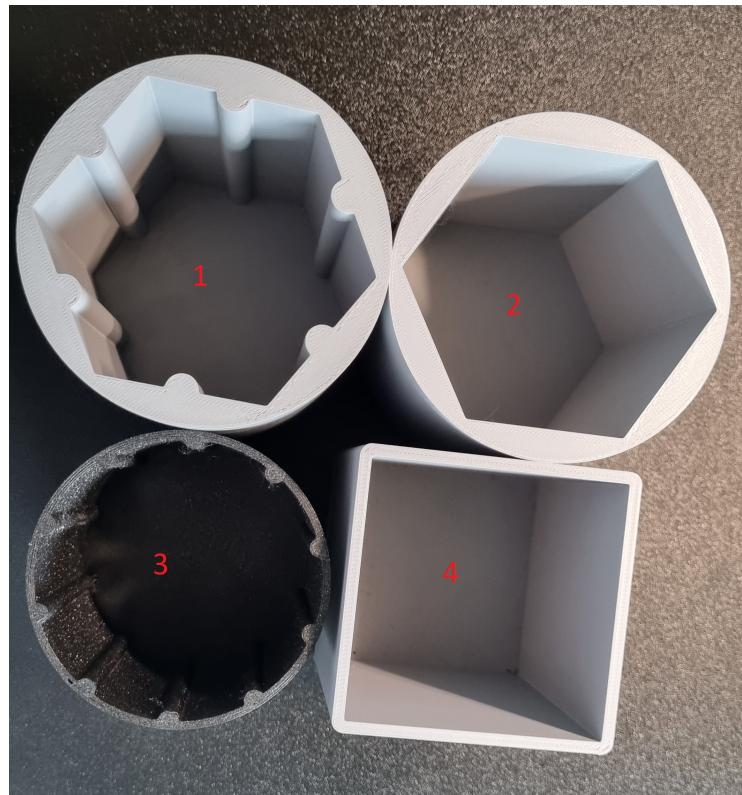
Proces projektowania robota rozpoczęto od przeanalizowania różnych metod wykonywania rzutu kościa. Po przeanalizowaniu kilku pomysłów, zdecydowano się na rozwiązanie wykorzystujące obrotowy kubek, wewnątrz którego kość porusza się i odbija od ścianek. Wariant ten roboczo nazywano betoniarką od podobnej zasady działania. Taki mechanizm zapewnia rzut zbliżony do takiego wykonanego przez człowieka, a jednocześnie jest prosty w konstrukcji. Obracający się kubek został zaprojektowany tak, aby jego prędkość obrotową i czas trwania obrotu można było w prosty sposób kontrolować poprzez kod napisany w Pythonie. W celach testowych został skonstruowany prototypowy model robota.

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 napędzający kubek oraz sterownik służący do zasilania i sterowania ruchem silnika.



RYSUNEK 3.1: Pierwszy prototyp

Do pierwszych testów robota zaprojektowano kilka wariantów kształtów kubków. Przy ich projektowaniu wymiary wzorowano na dostępnych tradycyjnych kubkach do gry w kości. Przyjęto, że odpowiedni do tego zadania kubek powinien zawierać pewnego rodzaju nierówności na ściankach. Dzięki temu kostka nie będzie się ślizgać po ściance a zacznie się odbijać od tych nierówności, co będzie lepiej imitowało rzut kościami wykonany przez człowieka. Z tego powodu z założenia odrzucono tradycyjny model kubka do gry w cylindrycznym kształcie o gładkich ściankach. Zaprojektowano cztery wersje kubka, które następnie przetestowano umieszczając w środku kości do gry i obracając kubek wokół osi przechodzącej przez środek kubka. Najlepszym wariantem okazał się być cylindryczny kubek z dodatkowymi pionowymi żebrami (numer 3 na poniższym zdjęciu), o które kostka się odbijała podczas kręcenia. Jego główną zaletą nad resztą testowanych kubków był fakt, że nie posiada on narożników, w których kość, trafiając podczas obracania kubkiem, mogła utknąć dociskana przez siłę odśrodkową.



RYSUNEK 3.2: Testowane warianty kubków

W trakcie poszukiwania literatury w temacie rzutów kością natrafiono na artykuł opisujący eksperyment, który wykorzystywał bardzo podobne urządzenie. Urządzenie opisano w następujący sposób: *For the machine-throwing, we used a newly constructed, electrically-driven machine which had a rectangular cage, 25 inches in length and four by four inches at the ends, made of quarter-inch wire mesh. This cage rotated on an axis through the middle, throwing the dice from one end to the other over a rough course which made them bound about considerably.*[3]

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. Z początku planowano wykorzystanie serwomechanizmu jednak to rozwiązanie odrzucono, ponieważ większość dostępnych serwomechanizmów, ma ograniczony obrót do 180° lub 360° a to limitowałoby możliwości mechanizmu służącego do wychylania kubka. Ostatecznie w tym celu wybrano mały silnik krokowy momentem obrotowym 34.3 mN.m, który jest wystarczający do wychylania kubka. Silnik ten obraca układem dwóch kół zębatych przedstawionych na zdjęciu poniżej.

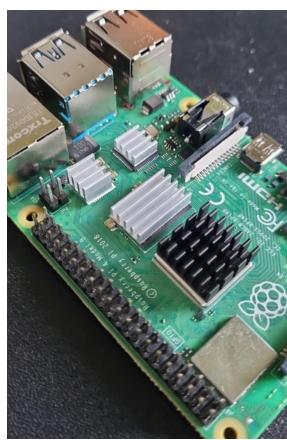


RYSUNEK 3.3: Koła zębate

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 znacząco upraszcza cały mechanizm robota oraz bardzo przyspiesza proces losowania liczby. Ten wariant nazywano roboczo blenderem, również od podobnej zasady działania mechanizmu.

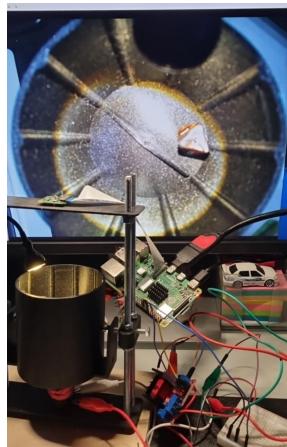
Przy projektowaniu drugiego wariantu robota został wykorzystany ten sam stelaż złożony z metalowych prętów co w pierwszym wariantie. Na drukarce 3D wydrukowano dodatkowe części, niezbędne do realizacji tego wariantu. Zaprojektowano i wydrukowano nowy kubek, śmiegle oraz mocowanie dla silnika. Kubek został 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, które miały pomóc w rozproszeniu nadmiaru ciepła, oraz wentylatora, który wspomagał cyrkulację powietrza wokół procesora. Dzięki temu rozwiązaniu udało się obniżyć temperaturę pracy procesora, która teraz utrzymuje się w granicach 55°C w czasie rzutów i spada po ich zakończeniu do około 45°C, co zapewnia stabilne i bezpieczne działanie całego systemu.

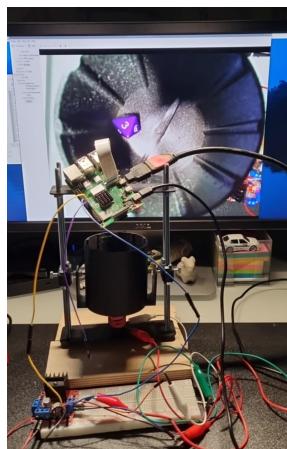


RYSUNEK 3.4: Dodane radiatory

Duże znaczenie ma również wykorzystywana kość. Od jej koloru i tekstury zależy jakość zdjęć zrobionych przez zamonotowaną kamerę. Poniżej przedstawiono dwa przykłady zdjęć i różnic w ich czytelności zależnych od koloru kości.

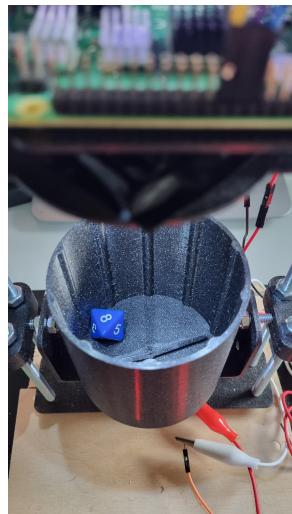


RYSUNEK 3.5: Kość z teksturą



RYSUNEK 3.6: Jednobarwna kość

W obu wariantach dużym problemem był złyj jakości obraz z kamery. W tym celu zaprojektowano system oświetlenia składający się z diod LED sterowanych przy pomocy układu ULN2803A. Dzięki temu wnętrze kubka stało się dużo jaśniejsze co pozwala kamerze na robienie zdjęć lepszej jakości. Dodatkowo rozświetlenie wnętrza kubka na tyle poprawiło jakość zdjęć, że pozwoliło to na obniżenie kamery względem kubka. Spowodowało to że wysokość prototypu zmniejszyła się o około 5cm - co było znaczącą poprawą, ponieważ jednym z założeń 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 czemu nie trzeba było wykorzystywać dodatkowych rezystorów a ilość połączeń została minimalna.



RYSUNEK 3.7: Oświetlone wnętrze kubka

Po skonstruowaniu obu wersji robota oczywistym stało się, że wariant blendera będzie znacznie szybciej (około 4-krotnie) wykonywał rzut kością. Pojedyńczy rzut kością 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 zużywają. 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.

3.1.2 Ostateczna wersja robota

Projektowanie ostatecznej wersji robota rozpoczęto od przeanalizowania wad, zalet oraz ogólnych cech (takich jak wymiary) prototypów. Postanowiono, że finalna wersja robota będzie wykorzystywała kubek o tej samej średnicy co prototypowa. Dookoła tych wymiarów zaprojektowano resztę konstrukcji. Obudowę zaprojektowano w taki sposób żeby była w stanie pomieścić kubek oraz kamerę umieszczoną na wysokości 120mm nad dnem kubka. Wysokość tą określono na podstawie jakości zdjęć jako najlepszą, podczas testów prototypów. Dodatkowo w tylnej oraz dolnej części obudowy pozostawiono przestrzeń na resztę elementów składowych robota. Rozmiar tej przestrzeni wyznaczono poprzez zwymiarowanie pozostałych elementów takich jak silnik, sterownik, wentylator czy Raspberry Pi. Te wymiary posłużyły do wyznaczenia minimalnej potrzebnej przestrzeni. To minimum zwiększyliśmy w taki sposób żeby we wnętrzu pomieściły się również przewody i śruby oraz żeby po całkowitym złożeniu pozostała przestrzeń do swobodnego manipulowania elementami składowymi.

Każdy element został zaprojektowany w taki sposób żeby cały robot był modułowy. Żeby spełnić to założenie, każdy element zaprojektowano w taki sposób żeby posiadał on specjalne miejsca na inserty lub śruby. Inserty to możliwe elementy, które za pomocą lutownicy wgrzewają 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

Góra część obudowy - pokrywa - 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 pojedyńczej śruby oraz magnesów neodymowych. Dzięki temu w przypadku kiedy konieczny jest dostęp do kamery lub diod LED wystarczy zdjąć tylną ścianę robota oraz odkręcić pojedyńczą 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.

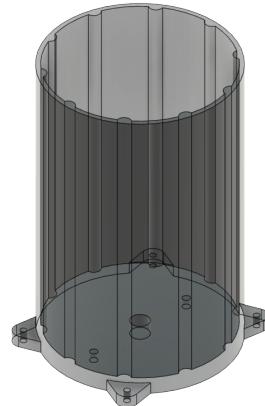


RYSUNEK 3.8: Pokrywa obudowy

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, 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 zajwiskiem podczas testów prototypu. Niestety takie rozwiązanie spowodowało, że wnętrze kubka przestało być widoczne z zewnątrz. Jednak zostało uznane, że widok z zewnątrz na wnętrze kubka nie jest konieczny do osiągnięcia celów projektu.



RYSUNEK 3.9: 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 pierwszego, jednak dodano dodatkowy trzeci cylindryczny słupek na śrubę, która miała służyć do kontrolowania wychylenia całego uchwytu w osi przód-tyl. 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 kości, 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. 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 kościami, 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 12V silnik z przekładnią, który ma mniejszą częstotliwość obrotu (około 1000RPM zamiast 4000RPM) 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 .



RYSUNEK 3.10: Pierwsza wersja uchwytu silnika oraz śmigło względem kubka



RYSUNEK 3.11: Druga wersja uchwytu silnika oraz śmigło względem kubka

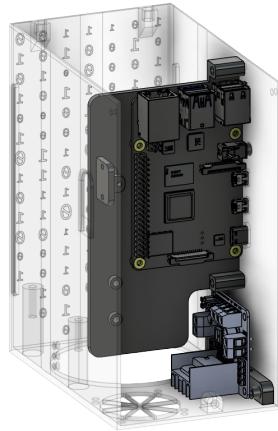
Do mocowania Raspberry Pi zaprojektowano specjalną płytę przykręcana do boku obudowy. W projekcie tej płytki uwzględniono otwory do przymocowania Raspberry Pi oraz układu ULN2803A Darlington. Dodatkowo przygotowano specjalne miejsce do mocowania przycisku. Na płytce pozostało również miejsce na zamontowanie szyny zasilania. Płytkę tą umieszczono 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 znalazła się ona bezpośrednio przy ścianie obudowy, a dodatkowo jego mocowanie nadal spełnia założenie modułowości poprzez mocowanie za pomocą śrub i insertów.



RYSUNEK 3.12: Płytki do montażu elektroniki

Układ sterujący silnikiem L298 przykręcono do obudowy pośrednio, poprzez specjalnie zaprojektowane i wydrukowane mocowanie. Dzięki temu wykorzystano gotowe otwory na śruby znajdujące się w płytce układu L298.

Do dna obudowy robota przymocowano za pomocą śrub również wentylator.

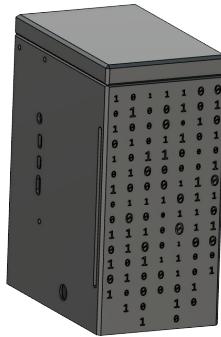


RYSUNEK 3.13: Zamocowany sterownik L298 z widoczną płytą na której zamocowana jest Raspberry Pi 4b

Obudowę zaprojektowano w taki sposób żeby pomieściła wszystkie powyższe elementy. W jej scianach zaprojektowano otwory na wyjścia Raspberry Pi, diody LED oraz gniazdo zasilania. W scianie obudowy na wysokości miejsca, w którym znajduje się wewnętrzny przycisk, zaprojektowano specjalne wycięcie. Dzięki dodatkowemu zmniejszeniu grubości sciana 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 wyłot powietrza. Dodatkowo w dnie umieszczono duży utwór umożliwiający swobodne mocowanie oraz dostęp do silnika, diod LED oraz gniazda 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.



RYSUNEK 3.14: Projekt gotowego robota od frontu

3.2 Dokumentacja techniczna

3.2.1 Hardware

Ogólne parametry robota:

Wymiary zewnętrzne: 180mm x 134mm x 84mm

Masa: 0,5 Kg

Komponenty:

Elektronika:

- Raspberry Pi 4B,
- Raspberry Pi Camera V2,
- L298 - *The L298 is an integrated monolithic circuit in a 15-lead multiwatt and PowerSO-20 packages. It is a high-voltage, high-current dual full-bridge driver designed to accept standard TTL logic levels and drive inductive loads such as relays, solenoids, DC and stepping motors..[5],*
- ULN2803A - *The ULN2801A, ULN2802A, ULN2803A and ULN2804A each contain eight Darlington transistors with common emitters and integral suppression diodes for inductive loads. Each Darlington features a peak load current rating of 600 mA (500 mA continuous) and can withstand at least 50 V in the OFF state..[4],*
- silnik DC 12V,
- diody LED 3mm,
- przycisk monostabilny THT,
- gniazdo wtykowe DC w formacie 5,5 x 2,1 mm.

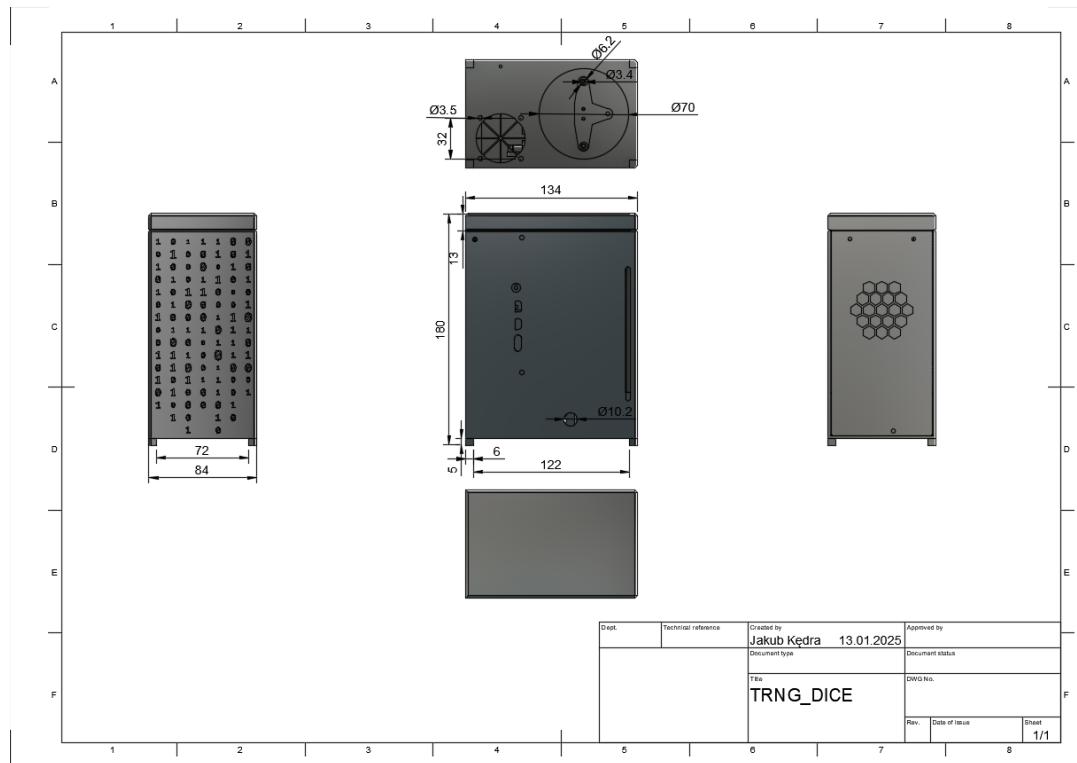
Elementy strukturalne (Wydrukowane na drukarce 3D. Materiał, którego użyto to PLA):

- obudowa,
- tylna ściana obudowy,
- górná pokrywa,
- płytka do montażu kamery i diod LED,
- uchwyt do silnika,
- płytka do montażu elektroniki,
- śmigło.

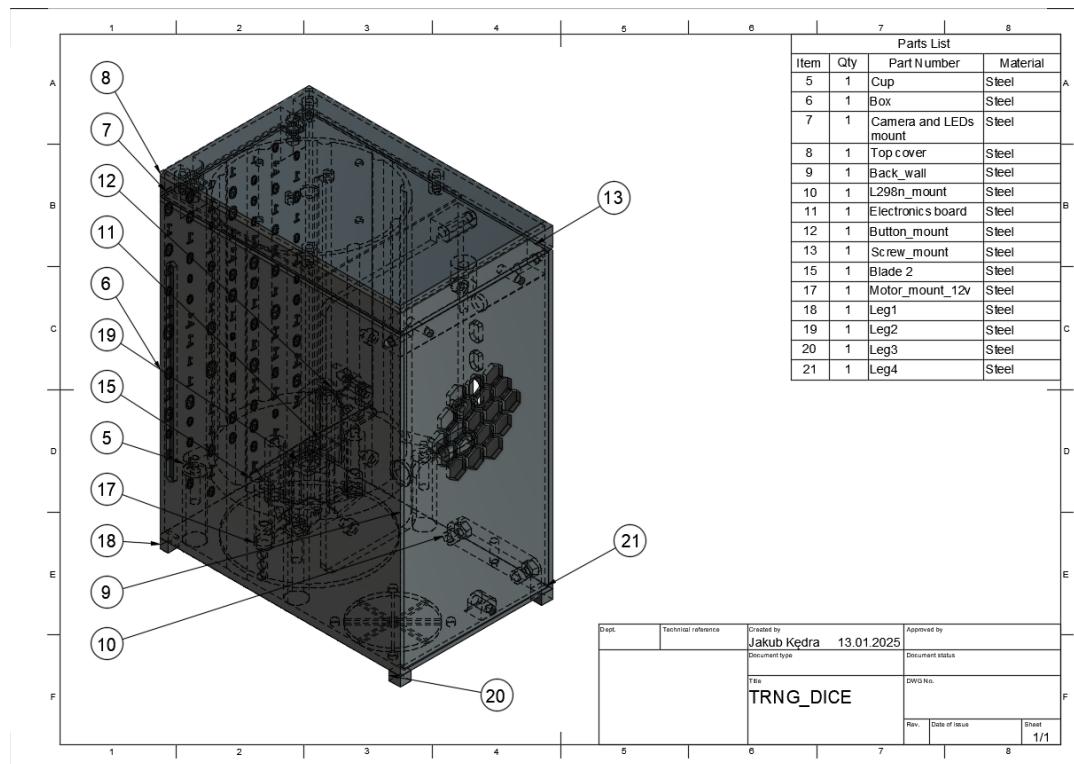
Zasilanie:

- Zasilacz DC 12V - wtyczka 5,5 x 2,1 mm.

Diagramy i schematy:



RYSUNEK 3.15: Wymiary zewnętrzne



RYSUNEK 3.16: Komponenty robota

3.2.2 Software

bbbbbbbbbbbbbbbbbbbbbbbbb

Rozdział 4

Odczytywanie losowego wyniku z kości

Kolejnym krokiem przetwarzania jest odczyt wylosowanej wartości z kości, jednak jak pokazano na schemacie 4.1 najpierw zdjęcie musi zostać poddane odpowiedniemu przetwarzaniu, które zostało opisane w tym rozdziale.

Schemat 4.1 w formie tekstowej

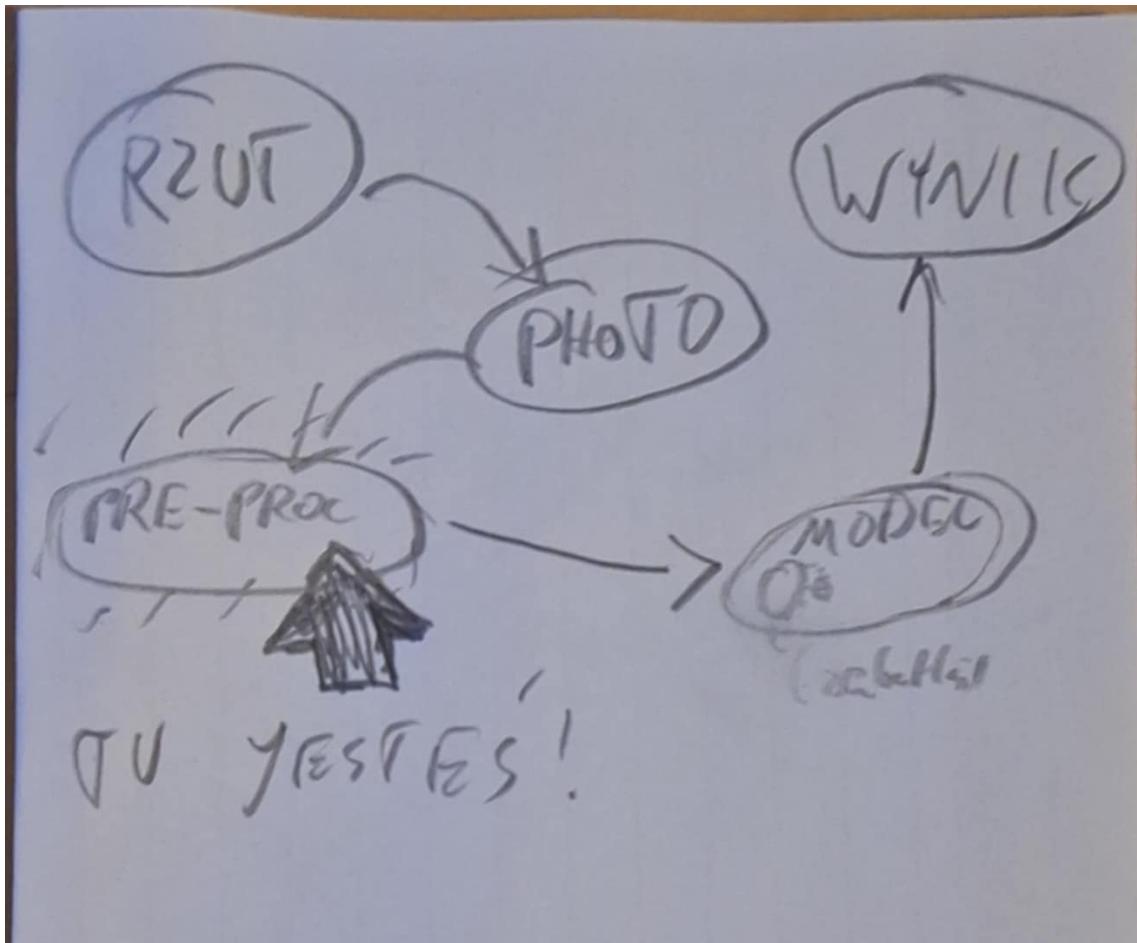
Maszyna (robot) wykonuje rzut kością

Kamera umieszczona w jego górnej części robi zdjęcie

Zdjęcie to jest przekazywane do algorytmu wykonującego preprocessing

Następnie, wyuczony przez nas model SI odczytuje wyrzucony na kości wynik i zwraca cyfrę

Cyfra jest przekazywana na wyjście



RYSUNEK 4.1: Ten schemat jest do zmiany na ładniejszy, ale ja nie umiem rysować więc jest taki "na teraz"

4.1 Przetwarzanie wstępne obrazów

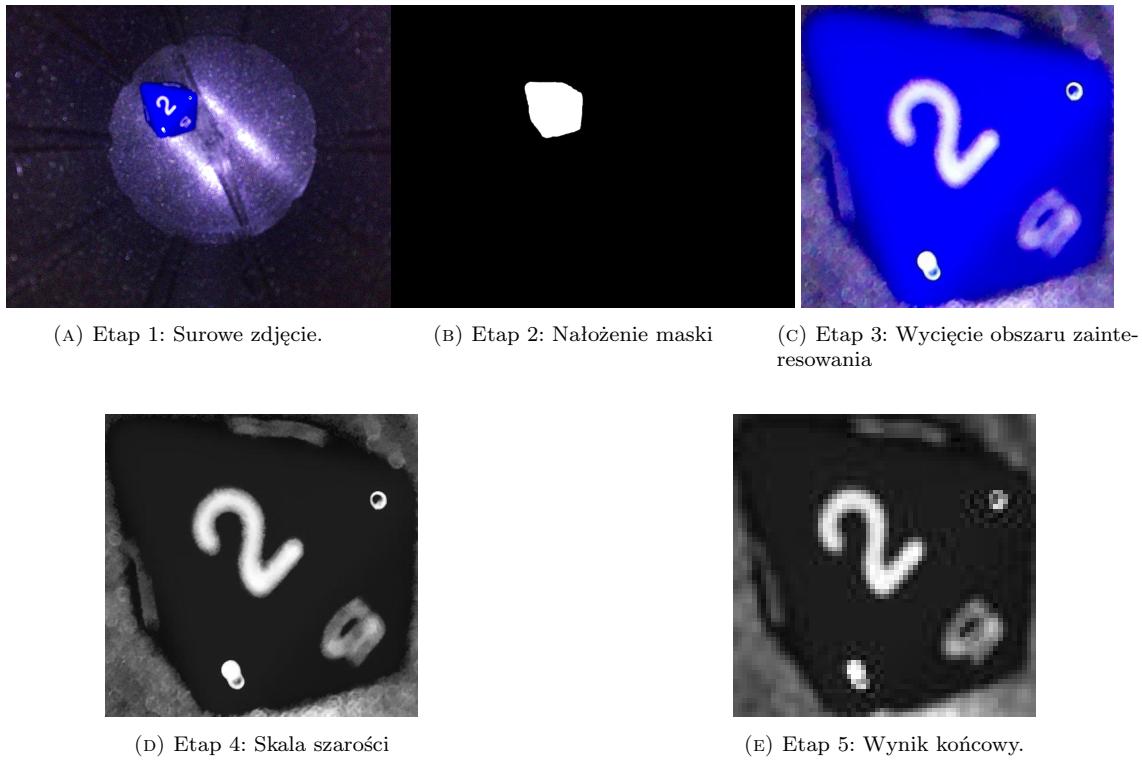
W niniejszym rozdziale omówiono proces przetwarzania wstępnego obrazów kości, który przekształca dane pochodzące z fizycznego komponentu naszej maszyny (zdjęcia kości) na dane wejściowe dla modelu sztucznej inteligencji. Przez dane wejściowe dla modelu SI rozumie się tutaj odpowiednio sformatowane obrazy, a więc takie w skali szarości, o rozmiarach 64x64 piksele, zawierające jedynie kość wyciętą ze zdjęcia całego bębna maszyny.

4.1.1 Algorytm

Przetwarzanie obrazów składa się z kilku etapów, które dokładnie opisano poniżej, a przepływ pracy prezentuje się tak:

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 rysunku 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.

- TODO — refator this part, bo nieczytelne
- TODO – cytowania, dokumentacje Pillow i OpenCV?

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

Zdjęcia w formacie JPEG są wczytywane za pomocą biblioteki Pillow, a następnie konwertowane do przestrzeni barw RGB, co zapewnia jednolitość formatów danych wejściowych.

Obrazy są przekształcane do przestrzeni barw HSV, aby oddzielić komponenty odpowiadające za barwę (H), nasycenie (S) oraz jasność (V). Nasycenie jest następnie wygładzane przy użyciu filtra Gaussa:

- TODO — jakieś cytowanie, o tym czemu filtr Gaussa? oraz dodać coś o parametrach filtra, konkretnie

Na podstawie komponentu nasycenia za pomocą progowania tworzona jest maska binarna, która identyfikuje obszary o wysokim nasyceniu (a więc naszą kość).

Próg został dobrany eksperymentalnie, kończąc na wartości *wartość!*, tak aby w kontrolowanym środowisku, jakim jest bęben robota, przy stałym oświetleniu opisany w poprzednim rozdziale generowana maska obejmowała obszar zainteresowania - kość, ale nie obejmowała tła (kubka). Korzysta to przede wszystkim z tego że kubek został wytworzony z czarnego materiału, a używana kość jest jednolitego, jasnego koloru.

W celu usunięcia niewielkich luk w masce binarnej stosowana jest operacja zamknięcia morfologicznego. Operacja ta jest szczególnie adekwatna w przypadku niejednolitej kości, takiej, która zawierałaby obszary o wyższym, jak i o niższym nasyceniu. W takim przypadku powstała maska mogłaby nie obejmować całej kości przy obecnie ustalonym progu nasycenia lub taka, która obejmowałaby osobno dwie rozłączne części kości. W związku z tym operacja domknięcia morfologicznego ujednolica maskę przed dalszym przetwarzaniem.

- TODO — cytowanie o zamknięciu morfologicznym.

Następnie maska jest analizowana w celu zlokalizowania największego konturu jątaczającego, z wykorzystaniem strukturalnego elementu prostokątnego, o dynamicznie dobranym rozmiarze obejmującym całość obliczonej w poprzednim kroku maski nasycenia, którego implementacja została zaczerpnięta z biblioteki OpenCV [tu cytowanie], które określa obszar zainteresowania.

Na tej podstawie oryginalny obraz jest kadrowany w kształt prostokąta wokół obszaru zainteresowania, a następnie przeskalowywany do wymiarów 64×64 pikseli.

— TODO — to cytowanie wyżej z OpenCV

Ostatecznie obraz jest konwertowany do skali szarości, co redukuje wymiarowość danych i umożliwia naszej sieci neuronowej skupienie się na strukturze obrazu.

Dodatkowo, konwersja do skali szarości pozwala znacznie lepiej poradzić sobie z problemem odblasków, powstających gdy kość niefortunnie ułoży się pod takim kątem, że odbija światło którejś z diód wprost do kamery.

— TODO — tu koniec tego segmentu do refactoru, jest okropny

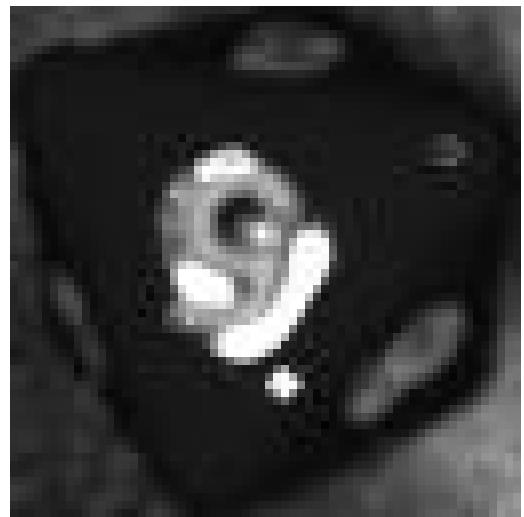
4.1.2 Zidentyfikowane trudności i ich rozwiązania

Wspomniane wcześniej odblaski znacznie pogarszają skuteczność odczytywania wyniku z kostki. Jednak zastosowanie skali szarości pozwoliło w znacznym stopniu pozbyć się tego problemu, uwidaczniając widoczną na kości cyfrę, co pokazują rysunki 4.3 oraz 4.4.

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



(A) Odblask na przeskalowanym zdjęciu.



(B) Odblask po zmianie na skalę szarości.

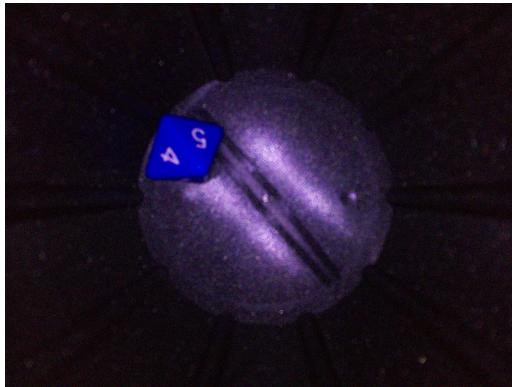
RYSUNEK 4.3: Porównanie przetwarzania odblasku: część a) i b).

Inną trudnością, która objawiała się w początkowych fazach pracy były zupełnie czarne obrazy, w wyniku problemu z działaniem diod, jednak problem okazał się być spowodowany przez hardware – pomogło lepsze zaizolowanie przewodów kamery i diód.

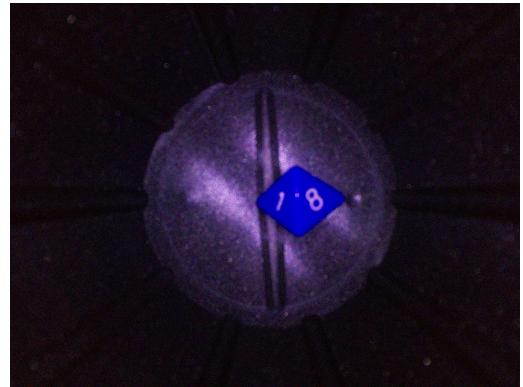
Mimo zażegnania problemu w taki sposób, zapewniono również obsługę wyjątku, który nadarzy się gdy do preprocessu trafi zupełnie czarne zdjęcie – taki przypadek jest pomijany. W przypadku trzech takich nieprawidłowych zdjęć pod rząd generator przerywa pracę zgłaszając błąd. W podobny sposób postarano się o wykrycie zacięcia się śmiegi, co znaczy że preprocess dostaje cały czas takie same zdjęcie do obróbki. Tu również zadbane o to, by to wykrywać, jednak w związku z

ryzykiem losowego podobieństwa kilku rzutów następujących po sobie, a więc ryzyka nieprawidłowego wykrycia błędu, przerwanie pracy następuje dopiero po 12 identycznych odczytach z rzędu. Statystyczna szansa na takie zdarzenie przy prawidłowej pracy maszyny wynosi $\frac{1}{8^{12}} \approx 1.455 \times 10^{-11}$

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 kostka zatrzymała się na śmigle napędowym, przez co wynik może być niejednoznaczny, co pokazano na rysunkach 4.4.



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



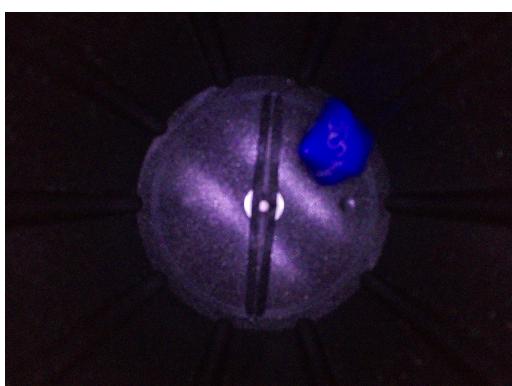
(B) Kość zatrzymana na śmigle (ujęcie 2).

RYSUNEK 4.4: Porównanie dwóch ujęć kości zatrzymanej na śmigle.

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

Zdarzają się oczywiście rzadkie przypadki, w których kość wyląduje równo na śmigle, tak że wynik jest idealnie czytelny, tak jakby wylądowała normalnie na podstawce kubka, ale wtedy problem z lądowaniem na śmigle nie istnieje, gdyż nie przeszkadza to w tym że kość i tak zostanie podbita przy następnym obrocie śmigła.

Ostatnim rodzajem problemu jaki został napotkany był przypadek w którym kostka nie skończyła jeszcze swojej fazy lotu, lub wpadła w bardzo długie wirowanie – efektywnie uniemożliwiając odczytanie wyniku.



(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 to proste czekanie stosowane obecnie.

— TODO — jakoś to trzeba pewnie sprawdzić, bo nie mogę sobie tak gadać, nie? Bo nie wiem, ale coś bym o tym pewnie napisał. O, jednak można tak gadać, ale to nadal w TODO (for now!)

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, dla przykładu 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 k8 do jednej z ośmiu klas, odpowiadających cyfrom od 1 do 8 na każdej ze ścianek. Do implementacji modelu wykorzystano przede wszystkim moduł tensorflow oraz wchodzący obecnie w jego skład Keras, który umożliwia łatwe tworzenie i trenowanie sieci neuronowych.

— TODO — linki z przypisami, cytowania dokumentacji Keras, Tensorflow

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, takie jak:

— TODO — cytowanie dokumentacji ImageDataGenerator

- obrót o losowy kąt w zakresie do 90°,
- przesunięcia poziome i pionowe,
- transformacje perspektywiczne (shear) ,
- losowe powiększenia (zoom) .

— TODO — dać obrazek surowy i każdą transformację. może też jakieś ich wszystkich złożenie, więc 6 obrazków? dodatkowo, niech to będą obrazki 5 i potem porównanie lustrzanej 5 i 2 !!!

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 rozróżnialne przez wytrenowany model.

4.2.2 Architektura modelu

— TODO — opis sieci splotowych CNN, czym są opis warstwy gęstej, opisy funkcji aktywacji
Model jest wielowarstwową siecią splotową (ang. convolutional neural net (CNN)) składającą się z następujących elementów:

- Warstwy wejściowej w rozmiarze 64 na 64 piksele, ale o jednym kanale (skala szarości)
- Trzech warstw splotowych z funkcją aktywacji ReLU:
 - warstwa 1
 - warstwa 2
 - warstwa 3
- Warstwy spłaszczającej (Flatten),
- Dwóch w pełni połączonych warstw (Dense), z których pierwsza również używa aktywacji ReLu, a ostatnia używa funkcji aktywacji softmax do klasyfikacji na 8 klas.

4.2.3 Użyte funkcje aktywacji

— TODO — opis

- ReLu
- Softmax

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.

4.2.4 Trenowanie modelu

Model został nauczony z wykorzystaniem optymalizatora Adam, funkcji strat sparse categorical crossentropy dostępnej w module tensorflow oraz metryki dokładności (ang. accuracy). Proces trenowania obejmował 20 epok. Na wejściu, model oczekiwany był obrazu przedstawiającego wnętrze robota, wraz z kostką, znormalizowanego do przedziału [0, 1].

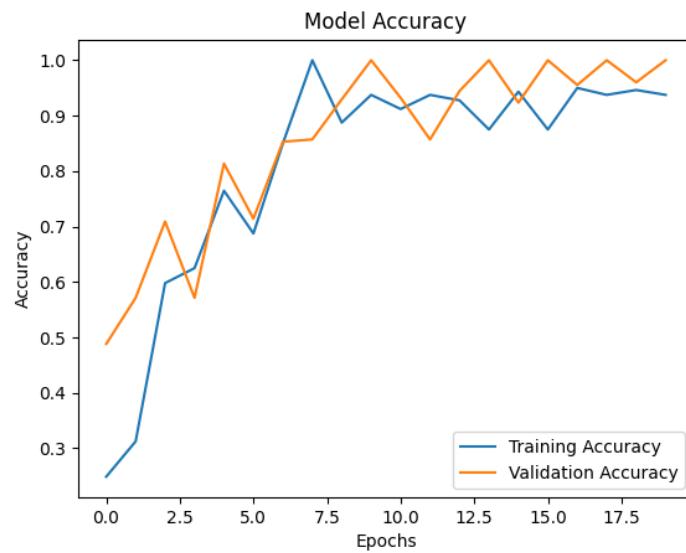
— TODO — cytowania dot. - Adam - sparse categorical crossentropy - accuracy

4.2.5 Wyniki

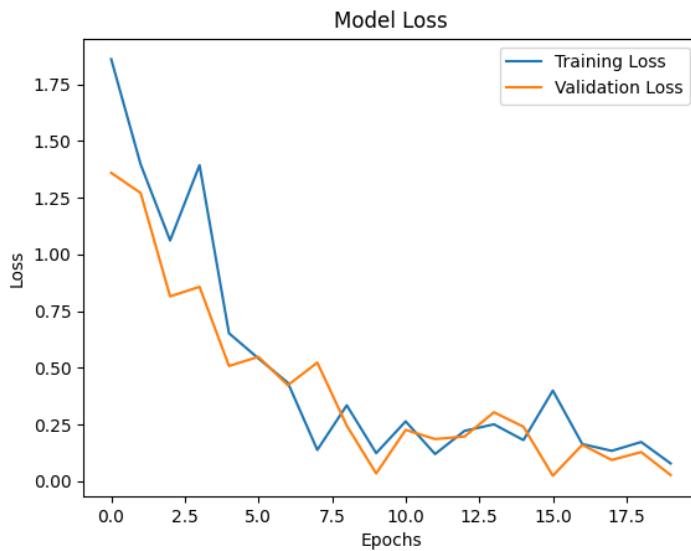
Podczas trenowania osiągnięto następujące końcowe wyniki:

- 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 przedstawiających zmianę dokładności i straty w trakcie trenowania.



RYSUNEK 4.6: Wykres dokładności modelu.



RYSUNEK 4.7: Wykres straty modelu.

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

4.2.6 Podsumowanie

— TODO — if needed, to be checked i guess

4.3 Funkcja predict

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

- Wczytanie i przetworzenie obrazu
- Predykcji klasy (klasyfikacji obrazu)
- Interpretacja wyniku

Opisanych w następujących podrozdziałach.

4.3.1 Wczytanie i przetworzenie obrazu

W przypadku funkcji dokonującej predykcji 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) oraz zmieniany w skalę szarości.

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

4.3.2 Predykcja klasy

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

```
prediction = model.predict(obraz_wejściowy)
klasyfikacja = np.argmax(prediction, axis=1)
```

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

4.3.3 Interpretacja wyniku

Na podstawie indeksu przewidywanej klasy wybierana jest odpowiadająca jej etykieta (od 1 do 8). Wynik funkcji jest tą etykietą odpowiadającą numerowi na kości, co umożliwia dalsze wykorzystanie tego wyniku w systemie rozpoznawania liczb.

W dalszym przetwarzaniu, aby przejść z cyfr w systemie dziesiętnym na zapis bitowy, dokonywana jest jedna nieoczywista transformacja. Zakres $[1, 8]$ da się zapisać za pomocą trzech bitów, wykorzystując go w pełni (co było również przesłanką w wyborze kości o dokładnie takiej ilości ścianek). Zdecydowano się interpretować ściankę 8 jako cyfrę 0, wobec czego, w transformacji na zapis bitowy, 8 jest interpretowane jako 000, zamiast 1000 jak mogłoby się intuicyjnie wydawać.

Finalnie, zaetykietowany wynik przetwarzany jest na ciąg trzech bitów, odpowiadających etykiecie klasy.

Rozdział 5

Przetwarzanie wyniku

Po odczytaniu wartości z kości przez model następuje zagregowanie bitowych reprezentacji wyników w kolejkę FIFO z której grupami po 8 bitów wysyłane są kolejne generowane ciągi, bla bla bla a właściwie po co grupować je po 8? wsm nie wiem, Kuba wytlumacz mi, proszę. Dlaczego nie możemy wypluwać tak po 3 tak jak generujemy, tylko akurat po 8

bo mamy k8? XDD (hehe)

nie no, chodzi o (int) prawda? czy jakaś łatwiejsza komunikacja, albo coś?

Rozdział 6

Testy

6.1 Użyte testy

Najszerzej stosowanym w świecie narzędziem weryfikacji generatorów (z racji dominowania standardów amerykańskich w dziedzinie ochrony informacji) jest zestaw testów podany przez normę amerykańską FIPS-140-2, dotyczącą bezpieczeństwa modułów kryptograficznych. W badaniu generatorów ciągów bitów losowych norma przewiduje cztery testy istotności. Każdy z nich przeprowadzany jest dla ciągu długości 20000 bitów (w przypadku wykorzystywania dłuższych ciągów przebadane muszą być kolejne podciągi o tej długości). Poziom istotności tych testów, czyli prawdopodobieństwo odrzucenia ciągu mogącego pochodzić z prawidłowego źródła bitów, jest równy 0,0001. [1]

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

Do przeprowadzenia testów rozważano również wykorzystanie biblioteki *TestU01*. 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 następującą hipotezę:

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 ósmiościennej (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:

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

Zatem dla otrzymanego stopnia swobody oraz założonego stopnia istotności, wartość krytyczna wynosi 14,0671. 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:

$$9725 < X < 10275$$

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. Test serii bada, czy zmiany między wartościami bitów nie są zbyt częste bądź zbyt rzadkie. Oczekiwane rozkłady wystąpień poszczególnych serii przedstawiono w tabeli 6.1.

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:

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

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.

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 14,0671. Odrzucono hipotezę H_0 i przyjęto hipotezę H_1 : rozkład odczytanych wyników rzutów kością nie jest równy. Otrzymany rozkład przedstawiono w tabeli 6.2.

TABELA 6.2: Rozkład otrzymanych wyników rzutu kością

	Liczba otrzymanych rzutów	
Wynik	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 otrzymanego ciągu zliczono 10694 bitów o wartości 1. Jest ona większa od oczekiwanej liczby jedynek. Hipotezę zerową odrzucono.

6.2.3 Test serii

Wyniki testów serii przedstawiono w tabeli 6.3. Rozkład serii w badanym ciągu 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 odrzucono.

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 zarówno dla zer, jak i dla jedynek, najdłuższa znaleziona seria zawierała 13 bitów. Jest to liczba większa od maksymalnej przyjętej długości najdłuższego ciągu bitów, wynoszącej 25. Nie ma podstaw do odrzucenia hipotezy zerowej.

6.2.5 Test pokerowy

Dla testu pokerowego otrzymano statystykę testową równą 120,026. Jest ona większa od oczekiwanej, zatem odrzucono hipotezę zerową. Rozkład segmentów w ciągu 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

6.2.6 Wnioski

Dla czterech z pięciu testów odrzucono hipotezę zerową. W wynikach testu statystyki rozkładu chi-kwadrat (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 na 000, co można zaobserwować w wynikach pozostałych testów. Sprawia to, że jedynki pojawiają się częściej w ciągu niż zera, co ukazuje już test monobitowy (6.2.1), gdzie liczba jedynek w ciągu jest większa niż oczekiwana. Podobny problem można zaobserwować w wynikach testu pokerowego (6.2.5), gdzie segmenty zawierające trzy lub cztery bity o wartości 1 występują znacznie częściej niż segmenty zawierające więcej bitów o wartości 0.

Rozdział 7

Zakończenie

Zakończenie pracy Lorem Ipsum Dolor sit itd. itp.

Literatura

- [1] Zbigniew Kotulski. *Matematyka Stosowana*. Polskie Towarzystwo Matematyczne, 2001.
- [2] National Institute of Standards and Technology. *Security Requirements for Cryptographic Modules*. National Institute of Standards and Technology, 2001.
- [3] J. B. Rhine. Dice thrown by cup and machine in pk tests. *Journal of Parapsychology*, 7(3), 1943.
- [4] STMicroelectronics. Eight darlington arrays. [on-line]
<https://www.st.com/en/interfaces-and-transceivers/uln2803a.html>, 2018.
- [5] STMicroelectronics. Dual full bridge driver. [on-line]
<https://www.st.com/en/motor-drivers/l298.html>, 2023.



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

Instytut Informatyki, Wydział Informatyki i Telekomunikacji
Politechnika Poznańska

Skład przy użyciu systemu L^AT_EX - MiK^TeX oraz workflow za pomocą Github Actions.