POLITECHNIKA POZNAŃSKA

WYDZIAŁ INFORMATYKI I TELEKOMUNIKACJI
Instytut Informatyki

Praca dyplomowa licencjacka

SPRZĘTOWY GENERATOR LICZB LOSOWYCH WYKORZYSTUJĄCY RZUTY KOSTKĄ

Julia Samp, 151775
Jakub Kędra, 151790
Wojciech Kot, 151879
Jakub Prusak, 151178

Promotor
dr inż. Jędrzej Potoniec

POZNAŃ 2025

Spis treści

Rozdział 1

Wstęp

Wstęp do pracy powinien zawierać następujące elementy:

- krótkie uzasadnienie podjęcia tematu;
- cel pracy (patrz niżej);
- zakres (przedmiotowy, podmiotowy, czasowy) wyjaśniający, w jakim rozmiarze praca będzie realizowana;
- ewentualne hipotezy, które autor zamierza sprawdzić lub udowodnić;
- krótką charakterystykę źródeł, zwłaszcza literaturowych;
- układ pracy (patrz niżej), czyli zwięzłą charakterystykę zawartości poszczególnych rozdziałów;
- ewentualne uwagi dotyczące realizacji tematu pracy np. trudności, które pojawiły się w trakcie realizacji poszczególnych zadań, uwagi dotyczące wykorzystywanego sprzętu, współpraca z firmami zewnętrznymi.

No więc co do tego i tamtego testy kurde ten. lol.
xD
tak.

Rozdział 2

Fragment teoretyczny

Współczesne systemy kryptograficzne oraz aplikacje wymagają generowania liczb losowych, które muszą charakteryzować się wysoką jakością i odpornością na przewidywalność. Najlepiej spełniają te wymagania 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 generowanie kluczy szyfrujących.

2.1 Zalety dostępnych rozwiązań

AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

2.2 Wady dostępnych rozwiązań TRNG

Generatory Liczb Losowych Oparte na Zjawiskach Fizycznych (TRNG) oferują niezrównaną jakość losowości, jednak ich wdrożenie i użytkowanie wiąże się z pewnymi wyzwaniami. Do najważniejszych problemów można zaliczyć wydajność, koszty implementacji, stabilność, złożoność integracji oraz aspekty bezpieczeństwa.

2.2.1 Wydajność, koszty i stabilność

Wydajność TRNG jest zazwyczaj niższa niż w przypadku generatorów pseudolosowych (PRNG). Proces generowania liczb losowych przy użyciu zjawisk fizycznych, takich jak szum termiczny czy fluktuacje kwantowe, może być czasochłonny, co sprawia, że TRNG mogą nie spełniać wymagań aplikacji o wysokiej częstotliwości operacji kryptograficznych.

Urządzenia TRNG, szczególnie te oparte na zaawansowanych technologiach, są kosztowne w produkcji i utrzymaniu. Wysokie koszty związane są z wykorzystaniem specjalistycznych komponentów, takich jak elementy fotoniki kwantowej czy czujniki szumów kwantowych. Jednocześnie jakość generowanych liczb losowych może być wrażliwa na czynniki zewnętrzne, takie jak zakłócenia elektromagnetyczne czy zmiany temperatury, co wymaga dodatkowych mechanizmów korekcji.

2.2.2 Integracja i skalowalność

Integracja TRNG z istniejącymi systemami może być skomplikowana, szczególnie w przypadku systemów wbudowanych. Konieczne bywa dostosowanie sprzętu lub oprogramowania, co zwiększa złożoność projektu i jego koszty. Ponadto TRNG nie zawsze są skalowalne, co może być problematyczne w aplikacjach wymagających elastyczności lub masowej produkcji.

2.2.3 Bezpieczeństwo i inne ograniczenia

Chociaż TRNG oferują wyższy poziom bezpieczeństwa niż PRNG, nie są one całkowicie odporne na ataki. Manipulacje fizyczne lub przechwytywanie sygnałów generowanych przez urządzenie mogą prowadzić do obniżenia jakości generowanych liczb losowych. W przypadku rozwiązań opartych na chmurze istnieje dodatkowe ryzyko związane z transmisją danych, co może wpływać na ich bezpieczeństwo.

2.3 Podsumowanie wad TRNG

TRNG oferują wysoki poziom losowości, ale wiążą się z wyzwaniami, takimi jak niska wydajność, wysokie koszty, wrażliwość na zakłócenia, skomplikowana integracja oraz zagadnienia bezpieczeństwa. Mimo to są niezastąpione w aplikacjach wymagających maksymalnego poziomu ochrony danych, a dalszy rozwój technologii może przyczynić się do przezwyciężenia obecnych ograniczeń.

Rozdział 3

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.

3.1 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.

3.2 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 detekcja fizycznych "bąbli"w lampach lawowych ("Entropy Wall"), nieprzewidywalnego fizycznie trójstopniowego wahadła, rozpadu radioaktywnego Uranu 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.

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.

3.3 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:

3.3.1 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.

3.3.2 Koszty implementacji

Urządzenia TRNG, zwłaszcza te oparte na zaawansowanych technologiach, 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.

3.3.3 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.

3.3.4 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.

3.3.5 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.

3.3.6 Ograniczona dostępność i skalowalność

Choć rynek TRNG rośnie, nadal jest on stosunkowo niszowy w porównaniu do bardziej powszechnych 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.

3.3.7 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.

3.3.8 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.

3.4 Podsumowanie dostępnych na rynku TRNG

Rozdział 4

Budowa sprzętowego generatora liczb losowych

4.1 Projektowanie robota

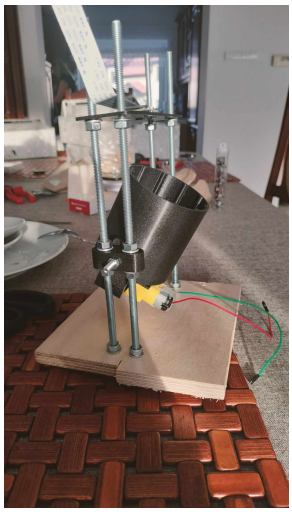
4.1.1 Prototypowanie

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 dość czasochłonny to jest to najlepsza dostępna nam technologia do wykorzystania w tego rodzaju projektach.

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

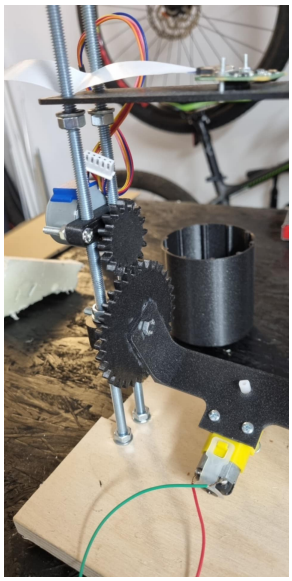
Proces projektowania robota rozpoczęto od przeanalizowania różnych metod wykonywania rzutu kością. Po przeanalizowaniu kilku pomysłów, zdecydowano się na rozwiązanie wykorzystujące obrotowy kubek, wewnątrz którego kość porusza się i odbija od ścianek. Wariant ten roboczo nazwano betoniarką od podobnej 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. 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 4.1: bałagan na stole

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 z wystarczającym do wychylenia kubka momentem obrotowym (34.3mN.m). Silnik ten obraca układem dwóch kół zębatych.



Rysunek 4.2: zebatki

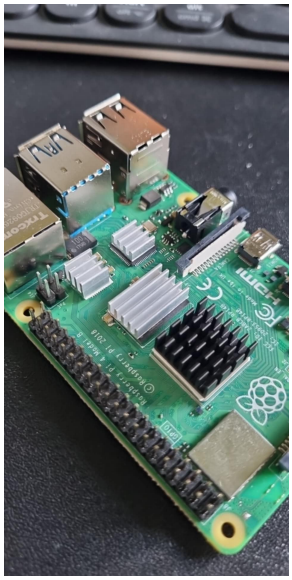
Do pierwszych testów robota zaprojektowano kilka wariantów kształtów kubków. Najlepszym wariantem okazał się być cylindryczny kubek z dodatkowymi pionowymi żebrami, o które kostka się odbijała podczas kręcenia.

Podczas testów pierwszej wersji robota wykorzytującej obrotowy kubek powstał pomysł alternatywnego rozwiązania. Rozwiązanie to implementuje inne podejście do rzutu kością. Zamiast obracać cały kubek, a dodatkowo wyhylać go, wykorzystany został trwale zamontowany kubek, na którego dnie znajduje się śmigło, które podcina leżącą na dnie kość. Takie rozwiązanie znacząco

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

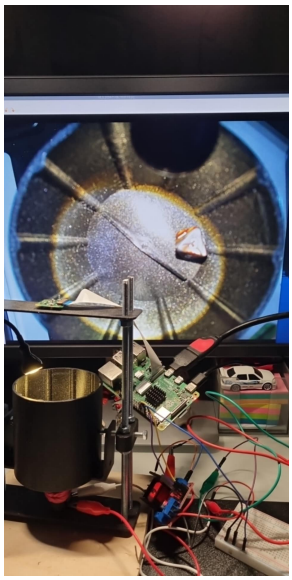
Przy projektowaniu drugiego wariantu robota został wykorzystany ten sam stelaż złożony z metalowych prętów co w pierwszym wariancie. Na drukarce 3D wydrukowano dodatkowe części, niezbędne do realizacji tego wariantu. Zaprojektowano i wydrukowano nowy kubek, śmigło oraz mocowanie dla silnika. Kubek został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

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, co zapewniło stabilne i bezpieczne działanie całego systemu.

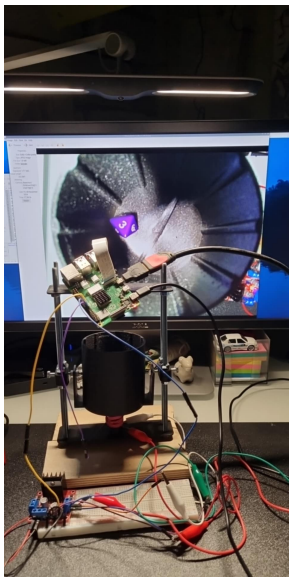


Rysunek 4.3: zimno

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.

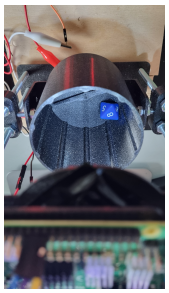


Rysunek 4.4: gorzej



Rysunek 4.5: lepiej

W obu wariantach dużym problemem był słaby obraz z kamery. W tym celu zaprojektowano system oświetlenia składający się z diod LED sterowanych za pomocą układu ULN2803A Darlington. Dzięki temu wnętrze kubka stało się dużo jaśniejsze co pozwala kamerze na robienie zdjęć lepszej jakości. Dodatkową 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 również miało duże znaczenie podczas projektowania robota, ponieważ jednym z założeń postawionych na początku budowy była kompaktowość. Dzięki tym zabiegom otrzymywane zdjęcia stały się dużo bardziej wyraźne oraz pole widzenia kamery było ograniczone 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 4.6: jasno

Po skonstruowaniu obu wersji robota oczywistym stało się, że wariant blendera będzie znacznie szybciej (około 3-krotnie) generował wyniki. Dodatkowo był ona zdecydowanie prostszy w budowie przez mniejszą liczbę ruchomych elementów. Z tego powodu 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 wariant blendera.

4.1.2 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 esperymentalnie podczas testów prototypów. Dodatkowo w tylnej oraz dolnej części obudowy pozostawiono przestrzeń na resztę elementów składowych robota. Tą przestrzeń wyznaczono poprzez zwymiarowanie pozostałych elementów takich jak silnik, sterownik, wentylator czy Raspberry Pi. Te wymiary posłużyły do wyznaczenia minimum potrzebnej przestrzeni. Do tego minimum dodano (CO) 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 mosiężne elementy, które za pomocą lutownicy wgrzewa się w wydruk 3D. Posiadają one wewnętrzny gwint dzięki czemu można do łączenia wydruków wykorzystywać śruby nie uszkadzając samego wydruku przy wielokrotnym skręcaniu i rozkręcaniu robota.

Górna 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 scianę 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 wewnątrz kubka.

Kamera wraz z diodami LED służącymi do oświetlenia wnętrza kubka znajdujde się w pokrywie. Kamere zamontowano przy użyciu dwóch śrub M2 natomiast diody LED umieszczono 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. 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 wewnętrznych żeber. Zmieniony został natomiast sposób mocowania kubka w taki sposób żeby był on przystosowany do zamocowania w obudowie. W tym celu zaprojektowano cztery mocowania za pomocą, których kubek jest przykręcany do obudowy. Dodatkowo pogrubiono dno kubka tak żeby można było w nim zamocować inserty służące do mocowania uchwytu silnika. Ostatnią modyfikacją było podwyższenie kubka w taki sposób żeby wysokością sięgał on aż do mocowania kamery - górnej pokrywy. Dzięki temu podczas rzutów zniknął problem z wypadającą kością co było dość częstym zajwiskiem podczas testów prototypu. Niestety takie Rozwiązanie spowodowało, że wnętrze kubka przestało być widoczne z zewnątrz. Jednak zostało uznane, że nie jest to konieczne do osiągnięcia celów projektu.

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 tego drugiego uchwytu wzorowano sie na projekcie pierwszego jednak dodano dodatkowy trzeci cylindryczny słupek na śrubę, która miała służyć do kontrolowania wychylenia całego uchwytu na boki.

Do mocowania Raspberry Pi zaprojektowano specjalną płykę mocowaną 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 płytnce pozostawiono również miejsce na zamontowanie szyny zasilania. Płytke 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 wydukowanego na drukarce 3D. Dzięki temu znalazł się on bezpośrednio przy ściance obudowy a dodatkowo jego mocowanie nadal spełnia założenie modułowości poprzez mocowanie za pomocą śrub i insertów.

Układ sterujący silnikiem LN298 przykręcono do obudowy pośrednio poprzez specjalnie zaprojektowane mocowanie. Dzięki temu wykorzystano gotowe otwory na śruby znajdujące się w płytce układu LN298.

Do dna obudowy robota przymocowano za pomocą śrub również wentylator.

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 wewnątrz przycisk, zaprojektowano specjalne wycięcie. Dzięki dodatkowemu zmniejszeniu grubosci sciana obudowy w tym miejscu jest bardziej elastyczna co pozwala na kliknięcie przycisku znajdującego się po wewnętrznej stronie sciany obudowy. Na dnie obudowy zaprojektowo również specjalne słupki służące za podpórki dla kubka. W słupkach tyych zaprojektowano otwory na inserty dzięki, którym kubek można przykręcić do obudowy gwarantując tym jego stabilność.

Podczas projektowania obudowy przewidziano także takie elemementy jak wycięcia od spodu bezpośrednio pod wentylatorem służące za wlot powietrza oraz wycięcia w tylenej sciance służące za wylot powietrza. Dodatkowo w dnie umieszczono duży utwór możliwiający swobodne mocowanie oraz dostęp do silnika, diod LED oraz gniazda zasilania.

Podczas testów ostatecznej wersji robota napotkano problem 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 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 12V silnik z przekładnią, który ma mniejszą częstotliowść obrotu ale ma większy moment obrotowy. Dodatkową 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 znacząco łatwiejszy.

4.2 Dokumentacja techniczna

4.2.1 Hardware

aaaaaaaaaaaaaaa

4.2.2 Software

bbbbbbbbbbbbbbbbbbbb

Rozdział 5

Odczytywanie losowego wyniku z kości

5.1 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.

5.1.1 algorytm

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

Przetwarzanie obrazów składa się z kilku etapów, które dokładnie opisano poniżej, a workflow prezentuje się tak: Obraz wejściowy -¿ Odnalezienie kości za pomocą maski na komponencie nasycenia -¿ Stworzenie i wycięcie "Bounding Box"wokół maski -¿ Przeskalowanie do odpowiedniego rozmiaru -¿ Konwersja do skali szarości -¿ Zapisanie gotowego obrazu

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

```
image = Image.open(image_path).convert("RGB")
```

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:

```
hsv_image = image.convert("HSV")
h, s, v = hsv_image.split()
blurred_v = s.filter(ImageFilter.GaussianBlur(radius=5))
```

Na podstawie komponentu nasycenia (S) oraz progowania tworzona jest maska binarna, która identyfikuje obszary o wysokim nasyceniu (a więc naszą kość):

```
v_array = np.array(blurred_v)
threshold = 224
binary_mask = (v_array > threshold).astype(np.uint8) * 255
```

W celu usunięcia niewielkich luk w masce binarnej stosowana jest operacja zamknięcia morfologicznego z wykorzystaniem strukturalnego elementu prostokątnego:

```
kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (100, 100))
closed_mask = cv2.morphologyEx(binary_mask, cv2.MORPH_CLOSE, kernel)
```

Maska jest analizowana w celu zlokalizowania największego konturu, który określa obszar zainteresowania. Na tej podstawie oryginalny obraz jest kadrowany, a następnie przeskalowywany do wymiarów 64×64 pikseli:

```
contours, _ = cv2.findContours(closed_mask, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
if contours:
    x, y, w, h = cv2.boundingRect(max(contours, key=cv2.contourArea))
    cropped_image = image.crop((x, y, x + w, y + h))
    resized_image = cropped_image.resize(size, Image.Resampling.LANCZOS)
```

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

```
grayscale_image = resized_image.convert("L")
```

5.2 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.

5.3 Opis Modelu SI

Model sztucznej inteligencji został zaprojektowany w celu klasyfikacji zdjęć ośmiościennych kości do jednej z ośmiu klas, odpowiadających cyfrom od 1 do 8. Do implementacji modelu wykorzystano moduł Keras, który umożliwia łatwe tworzenie i trenowanie sieci neuronowych.

5.3.1 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, takie jak:

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

Przykład konfiguracji generatora danych:

```
datagen = ImageDataGenerator(
    rescale=1.0 / 255.0,
    validation_split=0.3,
```

```
    rotation_range=90,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.1,
    zoom_range=0.1
)
```

5.3.2 Architektura modelu

Model jest wielowarstwową siecią konwolucyjną (CNN) składającą się z następujących elementów:

- Warstwy wejściowej

  ```
  layers.Input(shape=(64, 64, 1))
  ```

- Trzech warstw konwolucyjnych z funkcją aktywacji ReLU:

  ```
  layers.Conv2D(32, (3, 3), activation='relu')
  layers.MaxPooling2D((2, 2))
  ```

- Warstwy spłaszczającej (Flatten),
- Dwóch w pełni połączonych warstw (Dense), z których ostatnia używa funkcji aktywacji softmax do klasyfikacji na 8 klas:

  ```
  layers.Dense(128, activation='relu')
  layers.Dense(8, activation='softmax')
  ```

5.3.3 Trenowanie modelu

Model został skompilowany z wykorzystaniem optymalizatora Adam, funkcji strat sparse categorical crossentropy oraz metryki dokładności. Proces trenowania obejmował 20 epok:

```
model.compile(
    optimizer='adam',
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy']
)
history = model.fit(
    train_data,
    validation_data=val_data,
    epochs=20,
    steps_per_epoch=train_data.samples,
    validation_steps=val_data.samples
)
```

5.3.4 Wyniki

Podczas trenowania osiągnięto następujące końcowe wyniki:

- Dokładność na zbiorze treningowym: `final_train_acc`,
- Dokładność na zbiorze walidacyjnym: `final_val_acc`,
- Strata na zbiorze treningowym: `final_train_loss`,
- Strata na zbiorze walidacyjnym: `final_val_loss`.

Wyniki zostały również zwizualizowane na wykresach przedstawiających zmianę dokładności i straty w trakcie trenowania. ¡Tu dać wykresy¿

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

5.4 Funkcja predict

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

5.4.1 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. TODO (to jeszcze nie istnieje, ale będzie istnieć w finalnej części, gdzie przetwarzanie będzie bardziej "potokowe" niźli wstadowe jak teraz, jeśli tak to można określić) Obraz wejściowy wczytywany jest w trybie skali szarości i zmieniany na rozmiar zgodny z wymaganiami modelu (64×64 pikseli):

```
img = image.load_img(img_path, target_size=(64, 64), color_mode='grayscale')
img_array = image.img_to_array(img) / 255.0
img_array = np.expand_dims(img_array, axis=0)
```

Obraz jest normalizowany do zakresu $[0, 1]$, co pozwala na skuteczniejsze działanie sieci neuronowej.

5.4.2 Predykcja klasy

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

```
prediction = model.predict(img_array)
predicted_class = np.argmax(prediction, axis=1)
```

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

5.4.3 Interpretacja wyniku

Na podstawie indeksu przewidywanej klasy wybierana jest odpowiadająca jej etykieta (od 1 do 8):

```
class_names = ['1', '2', '3', '4', '5', '6', '7', '8']
predicted_label = class_names[predicted_class[0]]
return predicted_label
```

Wynik funkcji to etykieta odpowiadająca numerowi na kości, co umożliwia dalsze wykorzystanie tego wyniku w systemie rozpoznawania liczb.

5.5 Przetwarzanie wyniku

Finalnie, zaetykietowany wynik przetwarzany jest na ciąg trzech bitów, odpowiadających etykiecie klasy. Wyjątkiem w tym przetwarzaniu jest etykieta 8, gdyż w zapisie binarnym zawierała by 4 bity, zamiast oczekiwanych trzech, jednak biorąc 3 najmniej znaczące bity (000) jesteśmy w stanie zapewnić unikatowe kodowanie każdemu wynikowi. W ten sposób, klasy 1-7 generują bity będące ich dosłowną reprezentacją binarną, a 8 jest równoznaczne wyrzuceniu "zera"na kostce.

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. [Kotulski, 2001, s. 60]

W punktach 6.1.2 - 6.1.5 zostały opisane testy wskazane w owej normie. 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 wszystkich odczytanych wartości ścianek kości jest równy.

H_1: Rozkład wszystkich odczytanych wartości ścianek kości nie jest równy.

Aby zweryfikować hipotezę zerową, dla wszystkich n rzutów odczytano, ile razy wypadła każda z k ścian kości ośmiościennej (O_i). Wyniki te porównano z oczekiwaną 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 ośmiościennej, to k jest równe 8, co daje wzór:

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

Poziomem istotności (α) nazywa się prawdopodobieństwo popełnienia błędu pierwszego rzędu. Najczęściej przyjmuje się poziom istotności $\alpha = 0{,}05$. [Koziarska i Metelski, 2016, s. 82]

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 krtycznej, 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 oczekiwaną.

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

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 oczekiwanymi 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 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 oczekiwanym, gdy:

$$2{,}17 < X < 46{,}17$$

H_0: Rozkład 4-bitowych segmentów w ciągu jest zgodny z oczekiwanym.
H_1: Rozkład 4-bitowych segmentów w ciągu nie jest zgodny z oczekiwanym.

6.2 Wyniki testów

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

6.2.1 Test chi-kwadrat

Dla odczytanych rzutów kością otrzymano statystykę testową 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

Ścianka	Ilość otrzymanych rzutów
1	849
2	682
3	952
4	831
5	866
6	1124
7	816
8	547

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 oczekiwanymi 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	Serie zer	Serie jedynek
1	2343 - 2657	2717	2303
2	1135 - 1365	559	1141
3	542 - 708	559	678
4	251 - 373	263	374
5	111 - 201	113	158
6 i więcej	111 - 201	82	177

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	Kombinacja bitów	Liczba wystąpień
0000	197	1000	255
0001	249	1001	355
0010	277	1010	311
0011	312	1011	355
0100	279	1100	322
0101	309	1101	335
0110	363	1110	366
0111	338	1111	377

6.2.6 Wnioski

Dla czterech z pięciu testów odrzucono hipotezę zerową. W wynikach testu chi-kwadrat (6.2.1) można zauważyć, 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 liczb 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 (6.2.2), 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

Instytut Informatyki, Wydział Informatyki i Telekomunikacji
Politechnika Poznańska

Skład przy użyciu systemu LaTeX - MiKTeX oraz workflow za pomocą Github Actions.