Politechnika Wrocławska

Katedra Metrologii Elektronicznej i Fotonicznej

Laboratorium Optoelektroniki i Fotoniki

Optyczne sortowanie obiektów według kolorów

Optoelektronika II – Projekt

Prowadzący:

dr inż. Dariusz Wysoczański

Grupa 15

Błażej Chodorowski (263671)

Mikołaj Siewierski (263686)

Kacper Piątek (263581)

Wrocław 2024

Spis treści

1. Wstęp

Sortowanie jest nieodłącznym elementem każdego procesu produkcyjnego. Niezależnie od tego, czy mamy na celu rozdzielanie strumienia elementów ze względu na kształt, kolor czy wielkość, jest to nieunikniony proces podczas przygotowywania składników czy kontroli jakości. Dlatego też kluczowym aspektem jest, aby proces sortowania był szybki, ale jednocześnie precyzyjny.

Niniejszy raport przedstawia szczegółową analizę i rezultaty projektu dotyczącego stworzenia urządzenia do optycznego sortowania obiektów kolorowych. Głównym celem projektu było opracowanie efektywnego, dokładnego i szybkiego systemu zautomatyzowanego, który umożliwiałby precyzyjne klasyfikowanie kolorów kultowych cukierków, które wybrano jako obiekt testów.

W kolejne sekcje raportu skoncentrują się na fizycznych podstawach działania urządzenia, przedstawione zostaną wykorzystane zjawiska i omówione, w jaki sposób zostały one zaimplementowane. Następnie nastąpi kompletny opisu części sprzętowej, prezentujący schematy działania oraz fotografie kluczowych elementów. Szczegółowo opisana zostanie rolę poszczególnych komponentów oraz sposób, w jaki przyczyniają się do osiągania założonych celów.

W sekcji programowej przedstawione będą algorytmy sterowania urządzeniem i sortowania. Opisane zostaną wszystkie kluczowe funkcje programu, a także przedstawione ich znaczenie dla poprawnego działania systemu. Kolejnym krokiem będzie omówienie procesu pierwszego uruchomienia, procedury rozruchowej oraz kalibracji urządzenia, co pozwoli na skuteczne dostosowanie go do specyfikacji i wymagań produkcyjnych.

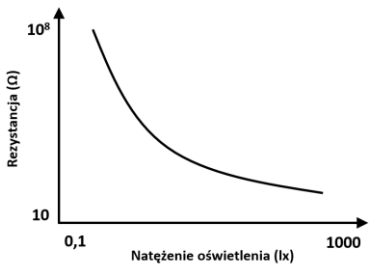
Raport szczegółowo przeanalizuje wyniki pomiarów testowych, wyciągając z nich istotne wnioski dotyczące poprawności działania urządzenia. Dodatkowo, zadba o potrzeby użytkownika końcowego, prezentując pełną wersję instrukcji, która opisze wszystkie niezbędne informacje potrzebne do skutecznej implementacji urządzenia w jego docelowym środowisku. W ten sposób raport ten nie tylko dokumentuje przebieg projektu, ale także stanowi kompleksowe narzędzie dla zainteresowanych stron, obejmując zarówno aspekty techniczne, jak i praktyczne użytkowania urządzenia.

2. Wprowadzenie

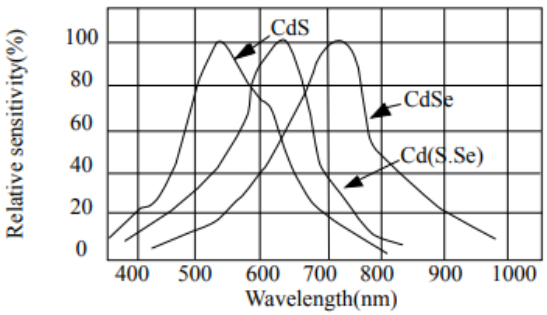
Kolor to właściwość percepcyjna obiektu, która pojawia się u obserwatora, gdy promień światła pada na powierzchnię obiektu. Fizyczny kolor obiektu zależy od tego, jak pochłania i rozprasza światło. Większość obiektów w pewnym stopniu rozprasza światło i nie odbija ani nie przepuszcza światła w sposób lustrzany. Kolor pochodzi od długości fali i intensywności światła w widzialnym spektrum. Ludzie są w stanie widzieć w zakresie widzialnym o długości fali od około 380 nm do 740 nm. Siatkówka oka posiada specjalne rodzaje komórek, które są wrażliwe na światło – czopki. Dzięki temu człowiek jest w stanie rozpoznawać kolory. [1]

W projekcie do detekcji kolorów został zaprojektowany czujnik który wykorzystuje fotorezystor i diodę elektroluminescencyjną programowalną. Dioda emituje na przemian światło czerwone, zielone i niebieskie, a fotorezystor służy do pomiaru intensywności światła odbitego od obiektu. Komponenty te zostały wybrane ze względu na niski koszt i dużą dostępność. Wyspecjalizowane czujniki do pomiaru koloru posiadają wysoką selektywność jednak są bardzo drogie. Warto zatem pokazać, że za pomocą kilku powszechnych elementów można wykonać czujnik kolorów do prostych zastosowań.

Fotorezystor jest elementem optoelektronicznym, pasywnym, który zmniejsza rezystancję w odniesieniu do odbierania jasności (światła) na wrażliwej powierzchni elementu. Opór fotorezystora maleje wraz ze wzrostem natężenia światła padającego - wykazuje fotoprzewodnictwo. [2] Fotorezystor zastosowany do budowy czujnika wykonany był z kadmu (Cd).



Wykres 1 Zależność rezystancji fotorezystora od natężenia oświetlenia [2]



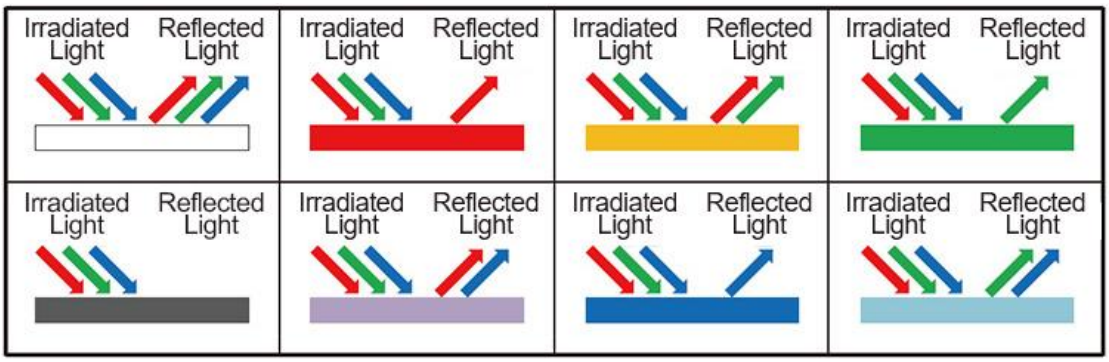
Wykres 2 Zależność czułości fotorezystora od długości fali dla różnych materiałów [3]

Dioda elektroluminescencyjna jest elementem półprzewodnikowym który emituje światło, gdy przepływa przez nią prąd. Elektrony w półprzewodniku rekombinują z dziurami elektronowymi, uwalniając energię w postaci fotonów. Barwa światła (odpowiadająca energii fotonów) zależy od energii wymaganej, aby elektrony przekroczyły pasmo wzbronione półprzewodnika. [4] Do budowy czujnika użyta została dioda programowalna WS2812B. Uprościło to budowę czujnika, ponieważ za pomocą jednej diody możliwe jest świecenie różnymi kolorami, sterowanie realizowane jest za pomocą prostych poleceń oraz zredukowana została liczba połączeń i elementów dodatkowych jak np. rezystory ograniczające prąd.

Tabela 1 Wybrane parametry diody WS2812b [5]

Kolor	Parametr	
	Długość Fali	Światłość
Czerwony	621 nm	310 mcd
Zielony	520 nm	780 mcd
Niebieski	471 nm	215 mcd

Zasada działania czujnika kolorów opiera się na pochłanianiu i odbijaniu światła. Kiedy obiekt zostanie naświetlony światłem zawierającym elementy RGB, kolor odbitego światła zmieni się w zależności od koloru obiektu co ilustruje Rysunek 1. Na przykład, jeśli obiekt jest czerwony, składowa światła odbitego będzie czerwona. W przypadku obiektu żółtego odbite światło będzie czerwone i zielone, a jeśli obiekt jest biały, odbite zostaną wszystkie trzy elementy. [6] W zaprojektowanym czujniku badany obiekt nie zostaje naświetlony jednocześnie wszystkimi składowymi RGB, lecz kolejno oświetlany jest kolorem czerwonym, zielonym i niebieskim. W ten sposób uzyskuje informacje na temat odbicia poszczególnych składowych RGB.



Rysunek 1 Odbicie światła dla różnych kolorów obiektów [6]

Do pomiaru natężenia światła odbicia wykorzystano fotorezystor szeregowo połączony z rezystorem. Układ ten tworzy dzielnik napięcia, na którym zmiany rezystancji elementu optycznego wpływają na stosunek podziału dzielnika i w prosty sposób można mierzyć wartość napięcia wyjściowego, która odpowiada natężeniu światła odbitego. Wartość tego rezystora nie jest przypadkowa. Pełni on funkcję ograniczenia prądowego. Założona została wartość maksymalnego prądu płynącego w obwodzie $I_{MAX} = 2.5mA$ dla w pełni oświetlonego fotorezystora przy zasilaniu $U_{cc} = 5V$. Obliczono rezystancję rezystora z pominięciem rezystancji oświetlonego fotorezystora zakładając, że jego rezystancja wynosi około 10 Ω. [3]

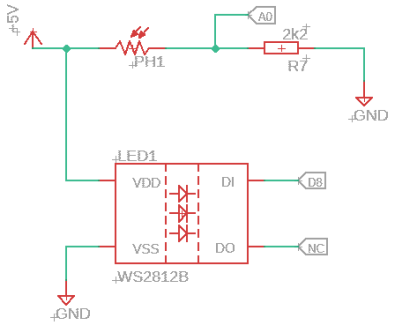
$$R = \frac{U_{cc}}{I_{MAX}} = \frac{5\ V}{2.5\ mA} = 2\ K\Omega \quad (1)$$

Biorąc po uwagę szereg E24 wybrano rezystor $2.2\ K\Omega$. Dla wybranej wartości rezystora obliczono maksymalną moc jaka może wydzielić się na fotorezystorze korzystając z warunku dopasowania energetycznego. Dla prądu stałego warunek ten to równość rezystancji obciążenia z rezystancją wewnętrzną źródła. [7] Przyjęto zatem rezystancję fotorezystora równą $R_f = 2.2\ K\Omega$.

$$P_{MAX} = \frac{{U_{cc}}^2}{R_f} = \frac{2.5^2\ \mathrm{V}^2}{2.2\ k\Omega} = 2.84\ mW$$

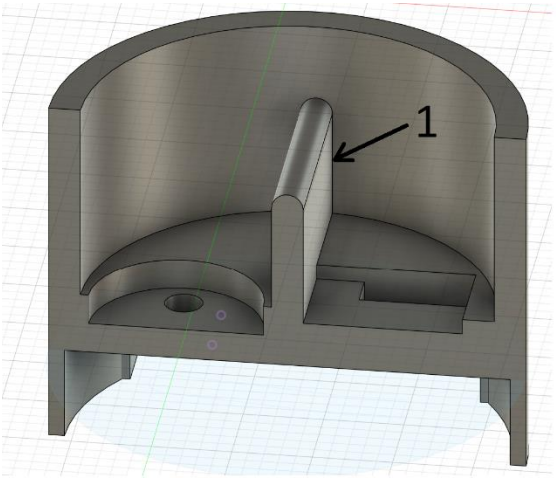
(2)

Maksymalna moc wydzielana na fotorezystorze wynosząca $P_{MAX} = 2.84\ mW$ nie powinna powodować znacznego nagrzewania elementu co wpływałoby na dokładność i powtarzalność pomiarów.



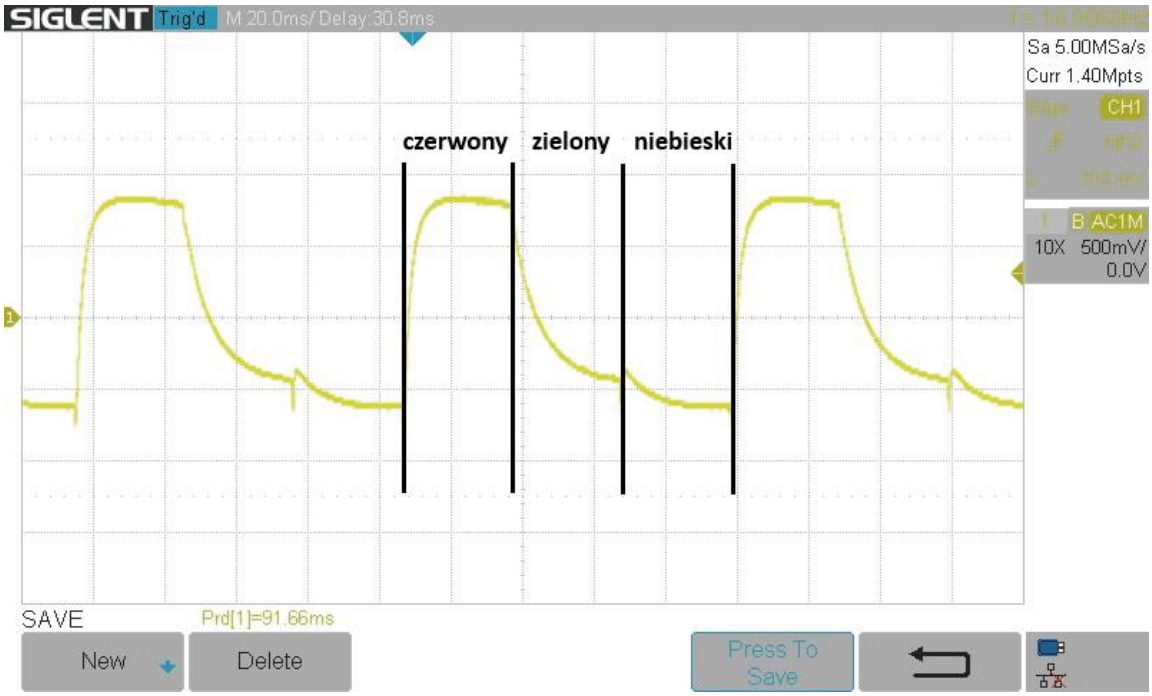
Rysunek 2 Schemat ideowy czujnika kolorów

W celu minimalizacji wpływy świecenia diody na fotorezystor zastosowano w obudowie czujnika barierę optyczną oznaczoną na rysunku niżej cyfrą 1. Dzięki niej czujnik rejestruje większy stosunek światła odbitego względem emitowanego przez diodę.



Rysunek 3 Przekrój poprzeczny obudowy czujnika kolorów

Należy mieć na uwadze wadę fotorezystora jaką jest bezwładność czasowa. Występuje bowiem opóźnienie pomiędzy ekspozycją na światło, a późniejszym spadkiem rezystancji i opóźnienie przy przechodzeniu z oświetlonego do ciemnego otoczenia. Wykonane zostały pomiary oscyloskopowe, aby ustalić doświadczalnie jak najkrótszy czas naświetlania. Założenie projektowe tego czujnika zakładało możliwość wykonywania szybkich pomiarów. Wyznaczony czas naświetlania poszczególną składową RGB wynosił 30 ms. Całkowity czas pomiaru koloru wyniósł 90ms.



Rysunek 4 Oscylogram wyjścia czujnika kolorów z oznaczonymi przedziałami naświetlania poszczególnymi kolorami

Biorąc pod uwagę charakterystykę czułości fotorezystora [Wykres 2] z której wynika, że dla fotorezystora użytego w czujniku maksymalna wartość czułości przypada dla długości fali równej około 640nm oraz długości fali emitowane przez diodę wynoszą 621 nm, 520 nm i 471 nm. To czułość zbudowanego czujnika dla tych długości fali sięga odpowiednio około 95, 40 i 25%. Z tego powodu należy przeprowadzić kalibrację czujnika oraz normalizację wartości. W omawianym projekcie kalibrację wykonuje się poprzez badanie odbicia do białego elementu oraz kierując czujnik w ciemną przestrzeń. W ten sposób uzyskane wartości są normalizowane do wartości z zakresu [0,255], ponieważ projekt wykorzystuje przestrzeń barw RGB.

$$Y = \frac{X - MIN}{MAX - MIN} \times 255 \quad (3)$$

Y – wartość znormalizowana

MIN – wartość minimalna

MAX – wartość maksymalna

X – wartość przed normalizacją

3. Założenia

W tym rozdziale przedstawiamy kluczowe założenia funkcjonalne i konstrukcyjne projektu. Założenia te stanowią podstawę planowania, definiując, jak system będzie działał oraz jakie cechy konstrukcyjne są kluczowe dla jego skuteczności i zgodności z oczekiwaniami.

3.1 Założenia funkcjonalne

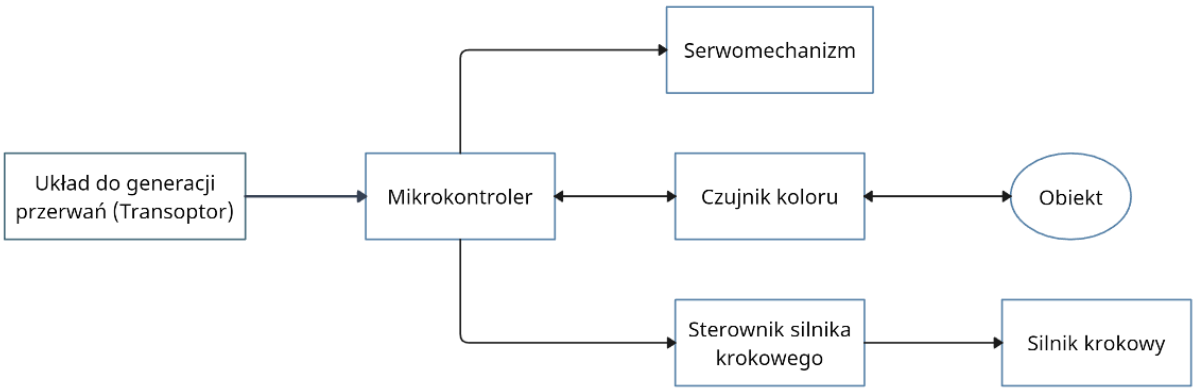
- Rozpoznawanie kolorów – sorter powinien precyzyjnie rozpoznawać kilka podstawowych kolorów. Jest to kluczowe dla skutecznego sortowania.
- Separacja cukierków według kolorów – urządzenie ma za zadanie oddzielać cukierki, kierując je do odpowiednich pojemników.
- Prostota użytkowania – obsługa sortera powinna być prosta i intuicyjna, minimalizując potrzebę zaawansowanej interakcji użytkownika.
- Uzyskanie wysokiej wydajności – szybkie sortowanie umożliwi obsługę dużej ilości cukierków w krótkim czasie.
- Zapewnienie bezpieczeństwa dla operatora urządzenia – wszystkie funkcje i elementy sortera muszą być zaprojektowane z myślą o bezpieczeństwie operatora, eliminując potencjalne zagrożenia podczas korzystania z urządzenia.
- Bezawaryjność – zapewnienie, że sorter kolorowych cukierków będzie zdolny do ciągłej pracy bez konieczności częstych przestojów z powodu usterek.
- Niski koszt budowy – zaprojektowanie urządzenia z myślą o wykorzystaniu ekonomicznych materiałów i komponentów elektronicznych.

3.2 Założenia konstrukcyjne

- Rozpoznawanie kolorów – zaprojektowanie i wykonanie czujnika kolorów oraz implementacja algorytmu do analizy składowych RGB i klasyfikacji koloru.
- Separacja cukierków według kolorów – zastosowanie mechanicznego systemu sortującego opartego na serwomechanizmie, który na podstawie klasyfikacji koloru skieruje obiekt do odpowiedniego pojemnika.
- Prostota użytkowania – minimalizacja liczby przycisków potrzebnych do obsługi urządzenia, stworzenie prostej obsługi kalibracji.
- Uzyskanie wysokiej wydajności – zastosowanie silnika o wysokiej mocy i prędkości obrotowej, użycie serwomechanizmu o dużej prędkości kątowej, budowa czujnika w sposób umożliwiający szybkie pomiary kolorów oraz optymalizacja algorytmów.
- Zapewnienie bezpieczeństwa dla operatora urządzenia – solidne wykonanie urządzenia, zaprojektowanie pełnej obudowy dla elementu obrotowego.
- Bezawaryjność – wybór wysokiej jakości materiałów konstrukcyjnych w krytycznych miejscach oraz użycie dobrej jakości elementów wykonawczych jak serwomechanizm czy silnik zapewniających odporność na zużycie.
- Niski koszt budowy - wybór ekonomicznych materiałów konstrukcyjnych w miejscach niekrytycznych dla działania urządzenia. Wykonanie czujnika kolorów z tanich i łatwo dostępnych elementów. Wybór prostego i taniego mikrokontrolera posiadającego wszystkie niezbędnie funkcje.

4. Opis części sprzętowej

Dla lepszego zrozumienia układu projektu na rysunku zamieszczonym niżej przedstawiono schemat blokowy urządzenia. Prezentuje on poszczególne główne układy zawarte w urządzeniu.

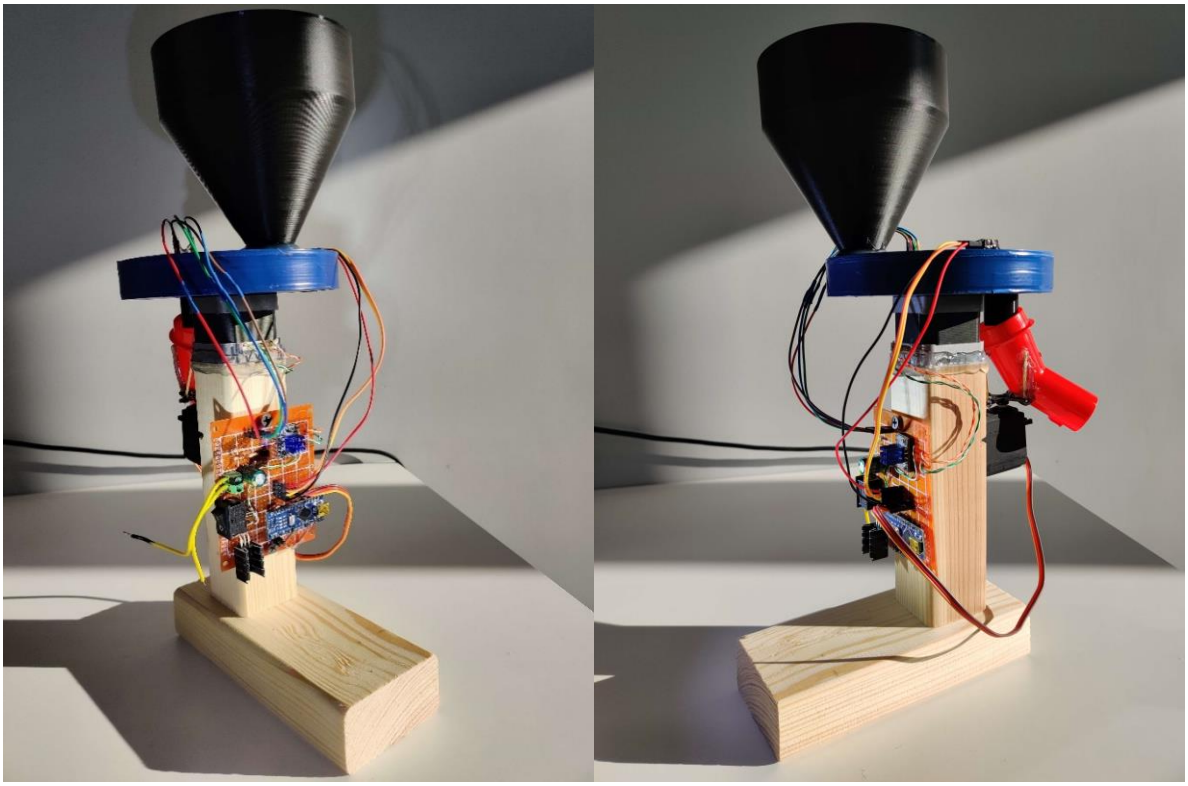


Rysunek 5 Schemat blokowy urządzenia.

Skonstruowane urządzenie składa się z 6 głównych bloków. Jednym z nich jest czujnik wykrycia szczeliny, inaczej nazywany w projekcie układem do generacji przerwania bazującym na transoptorze. Kolejne bloki są następujące: mikrokontroler, serwomechanizm i sterownik silnika krokowego wraz z silnikiem krokowym oraz czujnik koloru i obiekt oddziaływujący na ten czujnik.

4.1 Aspekt mechaniczny urządzenia

Ważnym aspektem w omawianiu całego urządzenia jest również jego aspekt mechaniczny. Obiekt składa się z pełnej konstrukcji zapewniającej stabilność urządzenia podczas pracy, jak również zastosowane są elementy konstrukcyjne służące do przemieszczania badanych elementów, w tym wypadku cukierków. Efekt końcowy projektu przedstawiono na rysunku niżej.

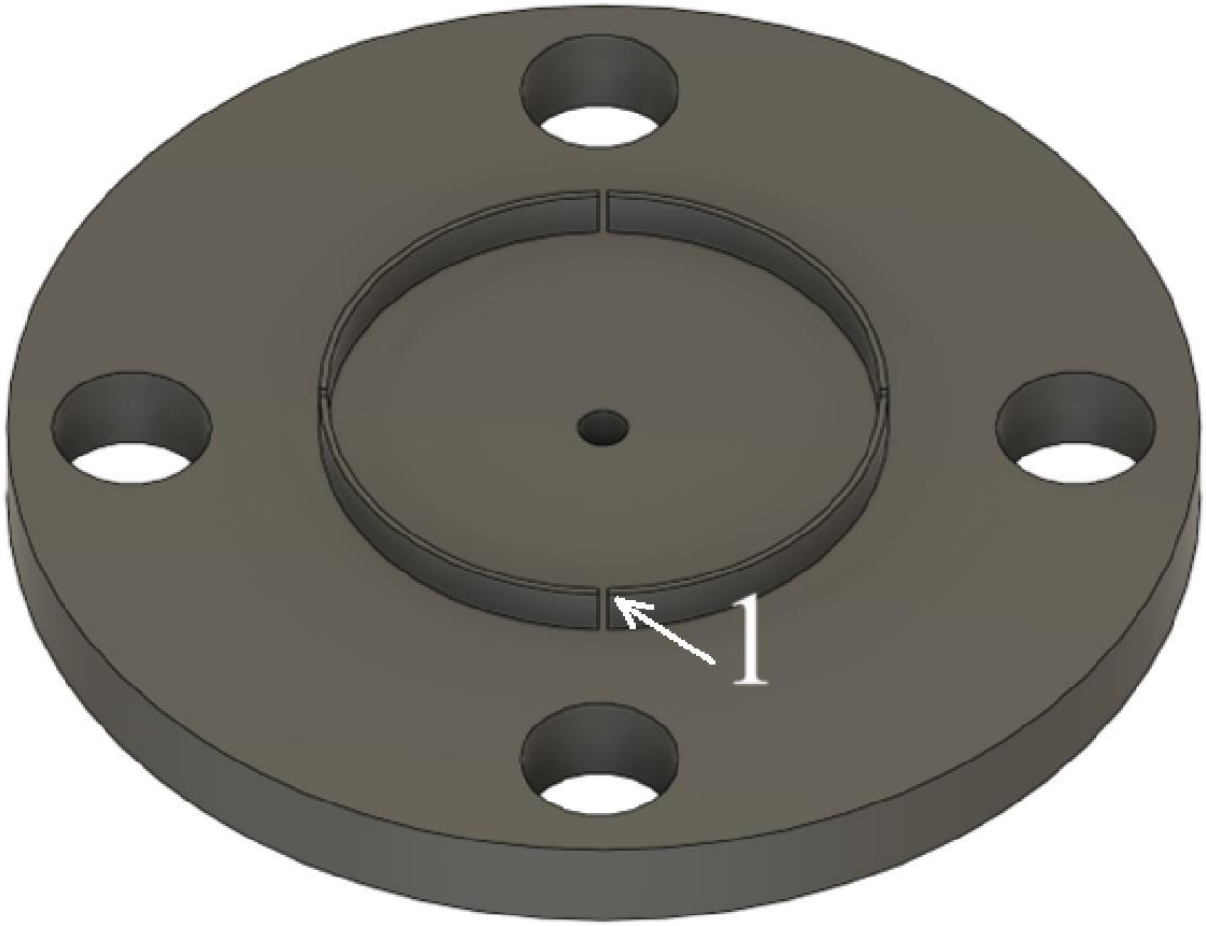


Rysunek 6 Zdjęcia przedstawiające efekt końcowy projektu.

Na samej górze znajduje się lejek, który służy do przetrzymywania i dozowania cukierków to tarczy obrotowej. Lejek zamocowany jest do górnej pokrywy, która wraz z boczną oraz dolną pokrywą zamyka w sobie badane cukierki wraz z tarczą obrotową. Dzięki temu światło zewnętrzne nie trafia do wnętrza, co powoduje, iż wyniki pomiaru czujnika koloru są niezależne od warunków oświetleniowych otoczenia. Do górnej pokrywy oprócz lejka zamocowany jest wyżej wymieniony czujnik koloru, jak również czujnik detekcji przerwania. Trzy wymienione wyżej pokrywy tworzą zamknięty obiekt, w którego wnętrzu znajduje się tarcza obrotowa służąca do przemieszczania cukierków. Jest ona obracana za pomocą silnika krokowego, zamocowanego do dolnej pokrywy, jak również do drewnianej belki służącej do podtrzymania wszystkich wyżej wymienionych elementów. jak również serwomechanizmu zamocowanego pod dolną pokrywą. Serwomechanizm poprzez rurkę segreguje poszczególne obiekty pomiarowe w odpowiednie pojemniki.

4.1.1 Tarcza obrotowa

Tarcza obrotowa jest bardzo ważną częścią mechaniczną, służącą do przemieszczania cukierków, jak również jest ona używana do detekcji przerwania.

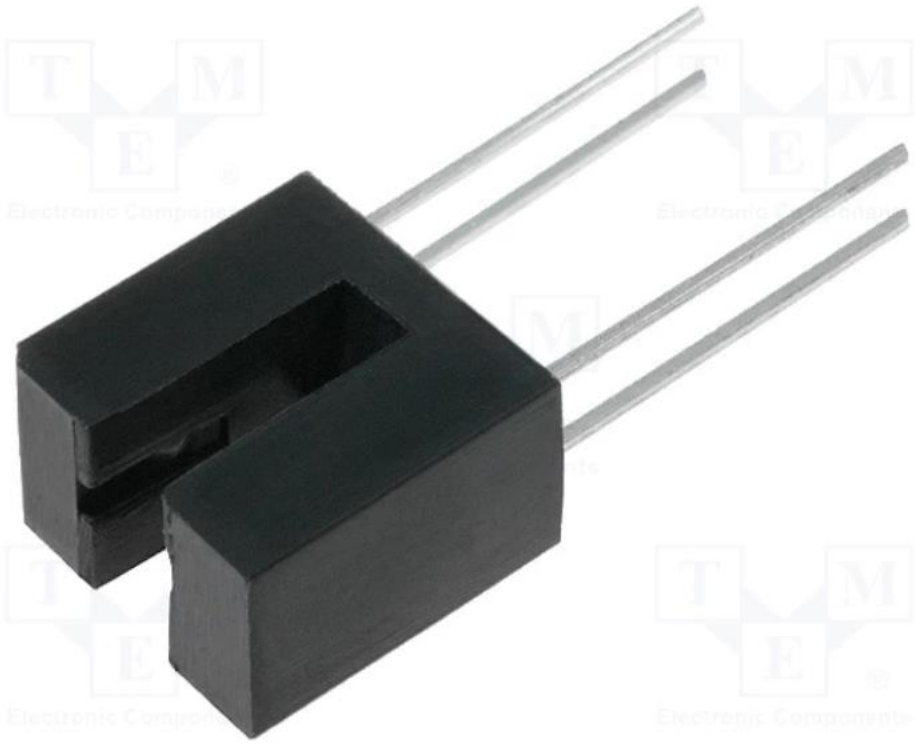


Rysunek 7 Tarcza obrotowa.

Jak można zauważyć na Rysunek 7 „Tarcza obrotowa" jest to duży, lecz niski walec, w którym przy zewnętrznych krawędziach znajdują się 4 dziury, służące do przemieszczania cukierków od zbiornika, przez czujnik koloru, do zlotu na serwomechanizm. Pośrodku tarczy znajduje się mała dziura, która służy do nałożenia tarczy na silnik krokowy. Na tarczy znajduje się również okrąg o wysokości 5 mm i 1 mm grubości, który posiada 4 przerwy przy wcześniej wymienionych dziurach od przemieszczenia cukierka (oznaczone na zdjęciu wyżej numerem 1).

4.2 Układ do generacji przerwań

Czujnik wykrycia szczeliny odpowiada za wykrycie przerwy w okręgu na tarczy obrotowej, o której była mowa w poprzednim punkcie. Składa się on z transoptora oraz tranzystora NPN. Do anody diody w transoptorze przez rezystor 1.2 kΩ podłączono napięcie 5V. Katodę zaś jak również emiter fotodetektora podłączono do masy. Do kolektora podłączono przez rezystor 44 kΩ napięcie 5V oraz również kolektor fototranzystora podłączono do bazy tranzystora NPN. Emiter tranzystora podłączono do masy, zaś kolektor przez rezystor 10 kΩ podłączono do napięcia 5V. Sygnał z kolektora podłączono również do mikrokontrolera na pin D2. Zdjęcie użytego transoptora prezentuje Rysunek 8.



Rysunek 8 Transoptor użyty w układzie generacji przerwania. [8]

Gdy między transoptorem znajduje się przerwa we wspomnianym okręgu, na jego wyjściu napięcie wynosi w okolicach 2.5 V. Jest to napięcie za wysokie jak na stan niski i z tego powodu zastosowano tranzystor NPN na kolektorze transoptora. Rezystor 44 kΩ został zastosowany dla ograniczenia prądu na bazie tranzystora NPN i został dobrany poprzez symulacje układu w programie LtSpice.

Po zastosowaniu dodatkowego tranzystora, gdy między transoptorem znajduje się przerwa, fototranzystor otwiera się, lecz nie w pełni, co powoduje tylko częściowe zwarcie napięcia z kolektora do masy. Z tego powodu na kolektorze napięcie wynosi w okolicach 2.5 V. W tym przypadku tranzystor NPN pozostaje zamknięty, zatem pełne napięcie 5V, tj. stan wysoki trafia na mikrokontroler i wywołuję funkcję przerwania. W przeciwnym przypadku zaś, gdy pomiędzy transoptorem znajduje się pełen okrąg, fototranzystor pozostaje w pełni zamknięty. W tym przypadku na jego kolektorze pozostaje pełne 5V, które trafia przez rezystor 44 kΩ na bazę tranzystora powoduje jego pełne otwarcie i zwarcie kolektora do emitera. Zwarcie to powoduje, iż stan niski trafia na mikrokontroler.

4.3 Czujnik koloru

Jeden z najważniejszych elementów całego projektu. Jest to czujnik składający się z diody elektroluminescencyjnej oraz fotorezystora połączonego szeregowo z rezystorem. Fotorezystor z szeregowo połączonym rezystorem tworzą dzielnik napięciowy. Pomiędzy nie dodano punkt pomiarowy, który doprowadzono do analogowego wejścia mikrokontrolera. Układ ten powoduje, iż przy zwiększeniu natężenia światła, zwiększa się również wartość napięcia w punkcie pomiarowym. Drugą częścią czujnika jest dioda elektroluminescencyjna sterowana za pomocą mikrokontrolera, która trzykrotnie naświetla badany obiekt, zmieniając kolor. Pierwszy pomiar występuje przy naświetleniu kolorem czerwonym, następny przy kolorze zielonym, a ostatni z pomiarów przy kolorze niebieskim. Każdy pomiar trwa 30ms i po tym czasie następuje pomiar napięcia przez mikrokontroler. Cały pomiar trwa 90ms. Konstrukcję czujnika można zobaczyć na rysunku niżej.

Rysunek 9 Własnoręcznie zrobiony czujnik detekcji koloru.

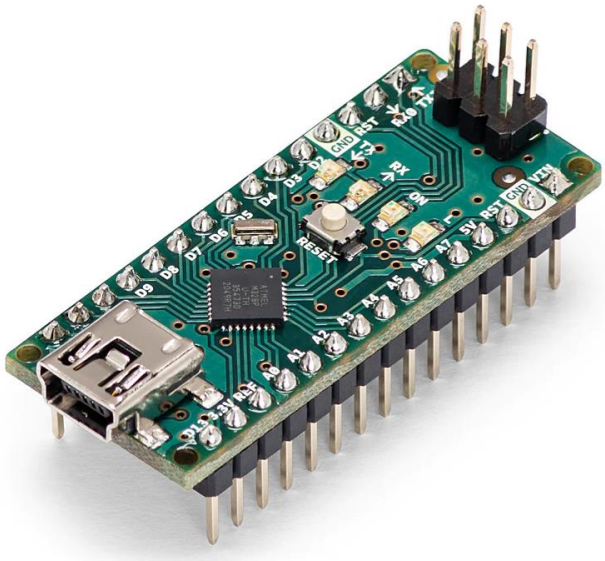
4.4 Stabilizator 5V

W trakcie rozwoju projektu zaobserwowano, iż na wyjściu 5V z mikrokontrolera pojawiają się duże szumy. Z tego powodu do układu dołożono prosty stabilizator 5V bazującego na układzie LM7805. Napięcie wejściowe układu wynosi od 7 do 20V a maksymalny prąd wyjściowy wynosi 2.2A. [9] Temperatura pracy układu wynosi od 0 do 125 °C. Z tego powodu na układzie zastosowano prosty radiator dla lepszego odprowadzania ciepła. Na wejście układu podano napięcie 9V. W układzie zastosowano na miedzy napięciem wejściowym, a masą oraz napięciem wyjściowym a masą zastosowano kondensatory filtrujące o wartości 47nF. Układ przedstawiono na rysunku niżej.

Rysunek 10 Układ LM7805 zastosowany do obniżenia napięcia z 9V na 5V.

4.5 Mikrokontroler

Mikrokontroler w projekcie odpowiada za wykonanie pomiarów jak również sterowanie i synchronizację poszczególnych elementów. W projekcie użyto mikrokontrolera ATmega328P korzystając z gotowej płytki Arduino Nano, którą prezentuje Rysunek 11. Jest to 8-bitowy mikrokontroler zasilany napięciem z zakresu od 1.8 – 5.5 V, taktowany częstotliwością 16 MHz. [10] Płytka ta jest stosunkowo tania oraz dobrze dostępna. W projekcie wykorzystano dwa wyjścia PWM – jedno do sterowania silnikiem krokowym, drugie zaś doprowadzono do sterownika silnika krokowego. Jeden pin cyfrowy wykorzystano do sterowania diodą elektroluminescencyjną w czujniku koloru oraz drugi pin cyfrowy wykorzystano do detekcji stanu wysokiego i niskiego z układu generacji przerwania. Po detekcji przerwania mikroprocesor przechodził do wykonania instrukcji obsługi przerwania. Do pomiaru napięcia z czujnika koloru wykorzystano jeden z analogowych pinów mikroprocesora. Pomiar ten jest przetwarzany. Korzystając z wartości zakresu dla pomiaru kalibracyjnego, dla każdego z trzech pomiarów dochodzi do zamiany wartości zmierzonej na wartość z zakresu od 0 do 255. Dzięki temu kolor obiektu reprezentowany jest przez standard RGB, obiekt zaś wtedy dopasowany zostaje do najbliższych wartości dla wcześniej zmierzonych kolorów obiektów testowych i wykonane są instrukcje ustawienia serwomechanizmu.



Rysunek 11 Gotowa płytka Arduino Nano bazująca na mikrokontrolerze ATmega328P. [11]

4.6 Serwomechanizm

Serwomechanizm jest to układ silnika z przekładnią oraz wewnętrznym układem odpowiadającym za sterowanie silnikiem. Model użytego serwomechanizmu jest to Tower Pro SG-5010 przedstawiony na rysunku niżej. Zastosowany układ pozwala na obrót silnika od 0° do 180°. Zasilane było ono napięciem 5V. Dla takiego napięcia prędkość układu wynos 0,2s / 60° oraz moment obrotowy 5,5kg/cm co jest wartością wystarczającą z dużym zapasem. [12] Sygnał PWM służący do sterowania układu generowany jest przez mikrokontroler pozwala na precyzyjny obrót serwomechanizmu do odpowiedniego miejsca. Na serwomechanizmie zamontowano zagiętą rurkę, do której wpada zmierzony obiekt i obrót serwomechanizmu powoduje wpadniecie cukierka do odpowiedniego pojemnika.

Rysunek 12 Użyty w projekcie serwomechanizm firmy Tower Pro model SG-5010. [12]

4.7 Sterownik silnika krokowego

Jako sterownik silnika krokowego zastosowano sterownik bipolarnego silnika krokowego bazującego na układzie TMC2225, który przedstawia Rysunek 13. Napięcie pracy układu wynosi od 4.75 do 36V i jest sterowany przez interfejs step z układu mikrokontrolera. [13] Korzystamy z ustawienia układu w trybie 4 mikrokroków, dzięki czemu sterowanie układu odbywa się bardzo precyzyjnie.

Rysunek 13 Użyty w projekcie sterownik silnika krokowego bazujący na układzie TMC2225. [14]

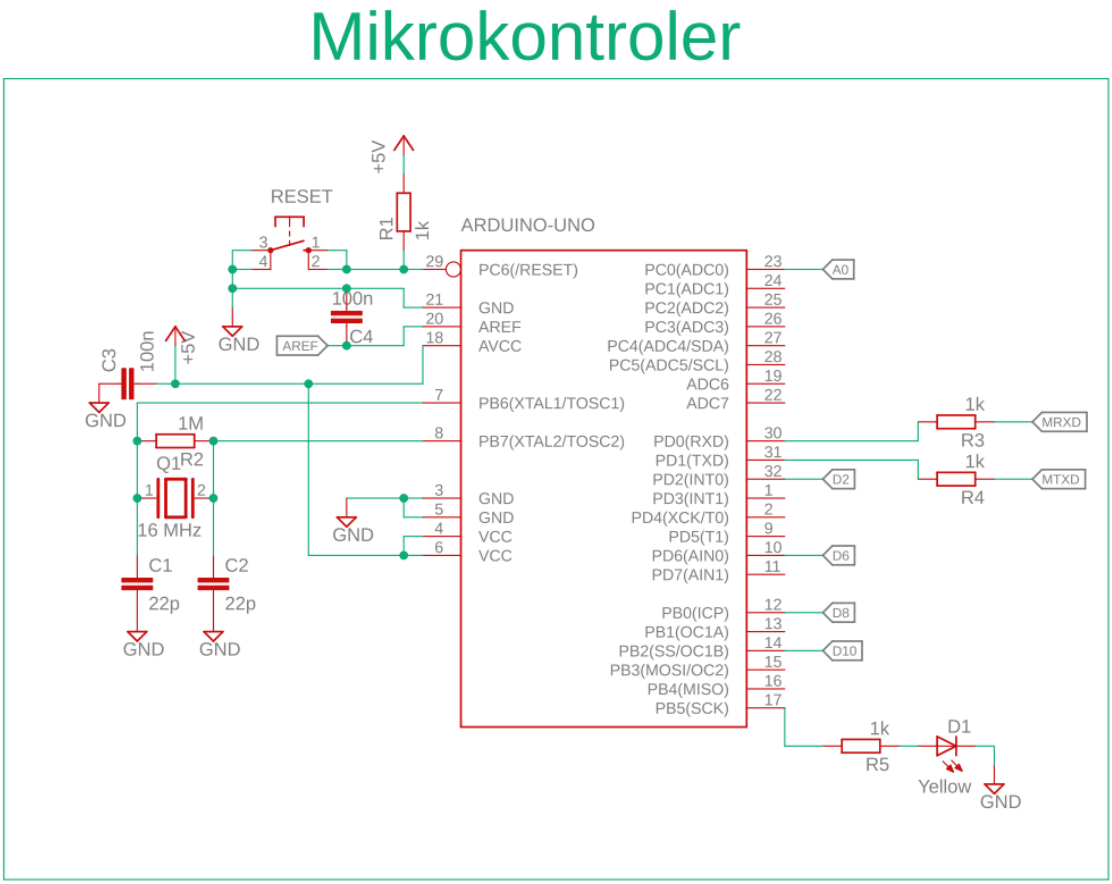
4.8 Silnik krokowy

Wykorzystany silnik krokowy jest to silnik unipolarny JAPAN SERVO model KH42JM2B217A, zaprezentowany na rysunku niżej. Napięcie zasilana silnika wynosi 5V. Pomimo, iż silnik ten jest unipolarny, wykorzystano go jako silnik bipolarny i z tego powodu podniesiono jego napięcie zasilania do 9V. [15] Wykonanie jednego kroku przez silnik odpowiada przesunięciu o 1.8°.

Rysunek 14 Użyty w projekcie silnik krokowy firmy Japan Servo.

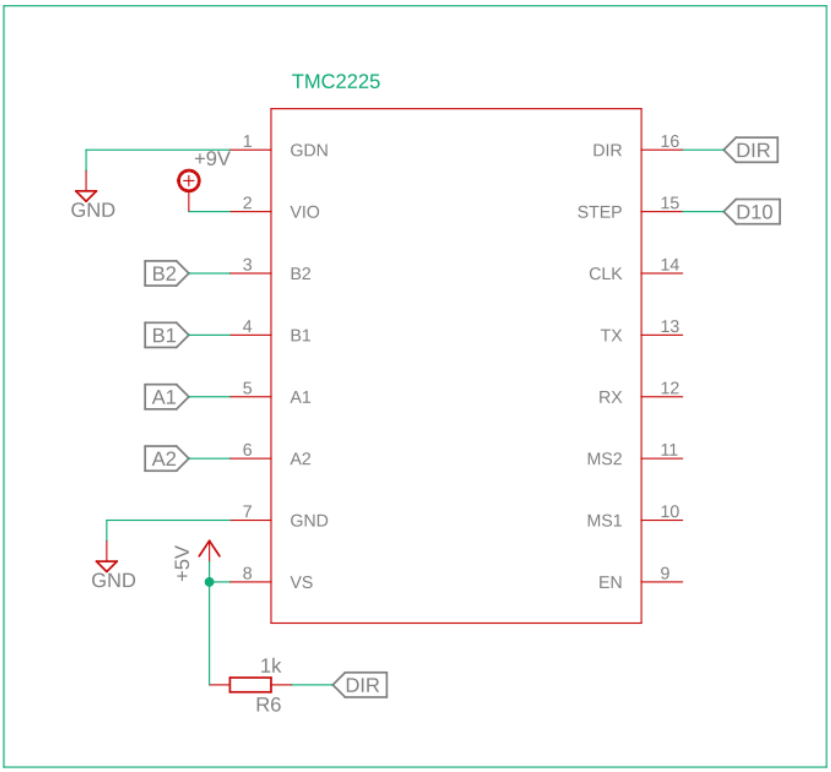
4.9 Schemat ideowy

Ważnym aspektem podczas konstrukcji urządzenia jest stworzenie schematu ideowego, dzięki któremu można skonstruować własną płytkę PCB, czy też jest on dużym wsparciem podczas próby naprawy urządzenia. Rysunki niżej prezentują stworzony schemat ideowy urządzenia. Przedstawiają one konstrukcję oraz połączenie każdego z układów użytych do budowy urządzenia.



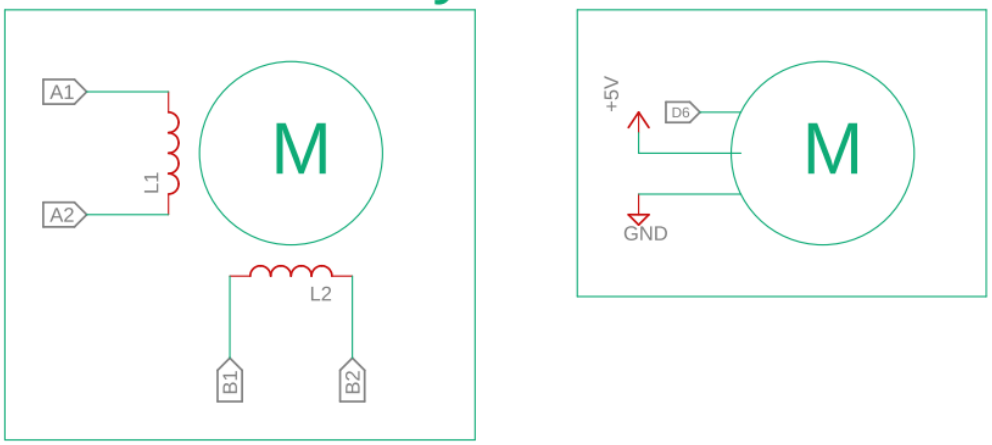
Rysunek 15 Schemat ideowy użytego mikrokontrolera ATmega328P bazującego na płytce Arduino Nano.

Sterownik silnika krokowego

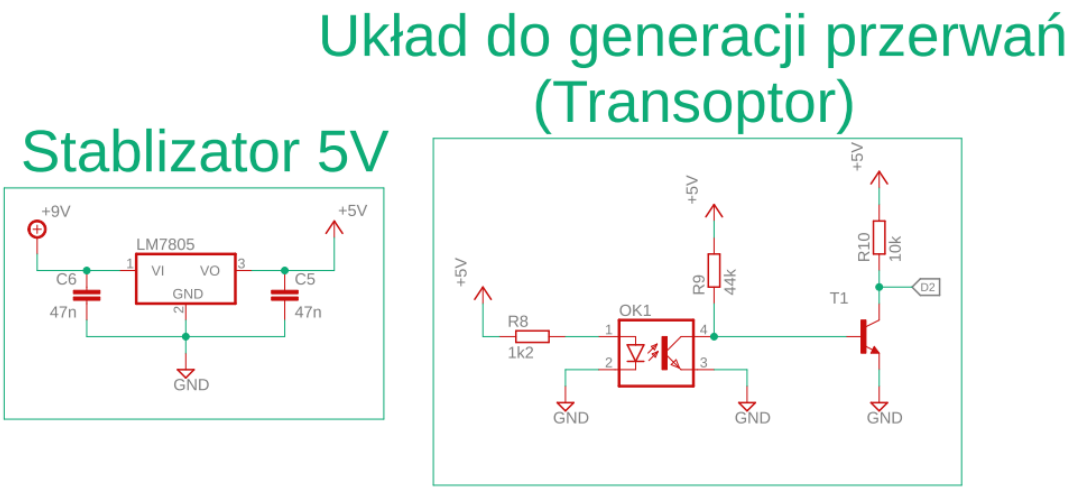


Rysunek 16 Schemat ideowy użytego sterownika silnika krokowego.

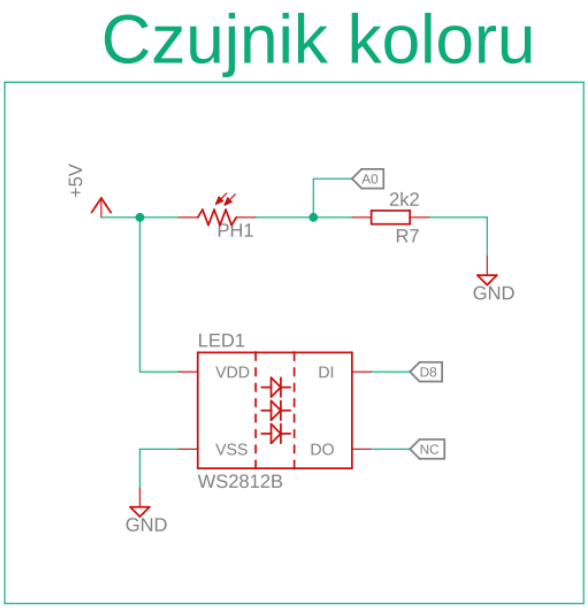
Silnik krokowy Serwomechanizm



Rysunek 17 Schemat ideowy silnika krokowego (po lewej) oraz serwomechanizmu (po prawej).



Rysunek 18 Schemat ideowy użytego stabilizatora bazującego na układzie LM7805 (po lewej) oraz układu do generacji przerwań bazującego na transoptorze (po prawej).



Rysunek 19 Schemat ideowy skonstruowanego czujnika koloru.

Schemat wykonano w programie EAGLE. Wartości poszczególnych elementów zostały zapisane na schemacie. Układ zasilany jest z zasilacza laboratoryjnego napięciem 9V. Napięcie 9V trafia na sterownik silnika krokowego oraz na przetwornicę obniżającą napięcie do 5V. Wygenerowane napięcie wykorzystywane jest przez następujące elementy: mikrokontroler ATmega328P, czujnik koloru, układ do generacji przerwań oraz serwomechanizm.

5. Opis części programowej

W tej sekcji zawarte zostały informacje na temat części programowej projektu. Przedstawiona została struktura plików programu oraz opisy najważniejszych fragmentów kodu i algorytmów.

5.1 Struktura programu

Ze względu na konieczność oprogramowania wielu różnych urządzeń, projekt został zapisany w sposób obiektowy wewnątrz wielu plików. Struktura plików wygląda w sposób następujący:

Katalog Projektu:

- Skittles_Sorter.ino – plik główny projektu
- SorterRGB.h, SorterRGB.cpp – obsługa algorytmów rozpoznawania kolorów
- StepperMotor.h, StepperMotor.cpp – sterowanie silnikiem krokowym
- SensorRGB.h, SensorRGB.cpp – obsługa czujnika kolorów

Każdy z plików służy do obsługi jednego urządzenia lub procesu pozwalając na łatwe wprowadzenie modyfikacji i aktualizacji oraz zwiększa czytelność.

Najważniejsze funkcje w każdym z plików zostaną omówione w kolejnych punktach.

5.2 Główny program

Na początku programu dostępne są stałe, używane do inicjalizacji obiektów obsługujących urządzenia wykonawcze oraz czujników. Jeżeli używa się modułu sterującego innego niż dołączony należy skonfigurować wszystkie stałe _PIN.

```
  /**
  * DO KONFIGURACJI:
  * Numery pinów ustawić zgodnie ze schematem połączeń w wypadku użycia własnego
modułu sterowania!
  * Ważne:
  * SENSOR_SENS_PIN - dowolny pin analogowy
  * SERVO_PIN - dowolny pin obsługujący PWM
  */
#define NUMPIXELS 1           /**< Liczba pikseli w sensorze RGB. */
#define SENSOR_CTRL_PIN 8     /**< Numer pinu sterującego sensorem RGB. */
#define SENSOR_SENS_PIN A5    /**< Numer pinu odczytu sygnału z sensora RGB (pin
analogowy). */

#define MOTOR_CTRL_PIN 10     /**< Numer pinu sterującego silnikiem krokowym. */
#define SERVO_PIN 6           /**< Numer pinu sterującego serwem. */
#define INTERRUPT_PIN 2       /**< Numer pinu przerwania związanego z
transoptorem. */
```

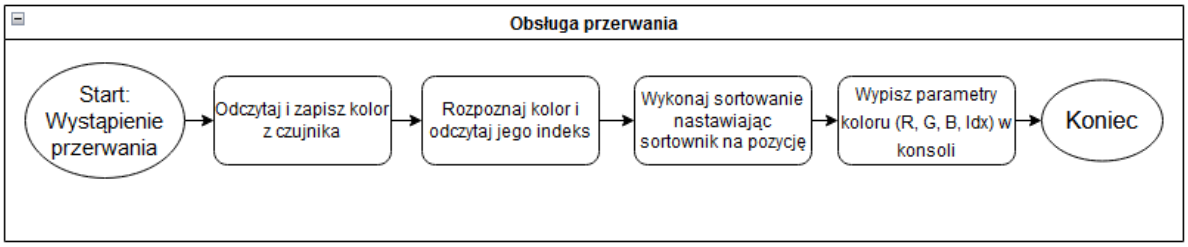
W funkcji konfiguracyjnej `setup()` inicjalizowane jest serwo, pin przerwania oraz rozpoczyna się komunikacja szeregowa z komputerem używana w celu wyświetlania danych kalibracyjnych i debugowania:

```
/**
 * @brief Funkcja inicjalizująca serwo, przerwanie oraz komunikację szeregową.
 */
void setup() {
    sinkServo.attach(SERVO_PIN);

    pinMode(INTERRUPT_PIN, INPUT_PULLUP);
    attachInterrupt(digitalPinToInterrupt(INTERRUPT_PIN), handleInterrupt,
RISING);

    Serial.begin(9600);
}
```

Główna pętla programu wykonuje się stale i ma za zadanie wyzwalanie metody obrotu silnika krokowego z zadaną częstotliwością:

```
/**
 * @brief Pętla główna programu.
 */
void loop() {
    stepper.makeStep(motorSpeed);
}
```

Następnie w funkcji obsługi przerwania dochodzi do wykonania sekwencji pomiarowej zgodnie ze schematem blokowym (Rysunek 20).



Rysunek 20 Schemat blokowy funkcji obsługi przerwania

```
/**
 * @brief Funkcja obsługująca przerwanie związane z odczytem koloru po
wyzwoleniu transoptora.
 *
 * > Sczytuje kolory nowego obiektu
 * > Określa kolor obiektu na podstawie kolorów odniesienia
 * > Nastawia serwo na podstawie koloru
 * > Wyświetla w konsoli dane kalibracyjne
 */
void handleInterrupt() {

    sensor.readColor(R, G, B);
    colorIndex = sorter.getClosestColorIndex(R, G, B);
    sinkServo.write(colorIndex*45);

    Serial.print(R);
    Serial.print(',');
    Serial.print(G);
    Serial.print(',');
    Serial.print(B);
    Serial.print('\n');

    Serial.print("indeks: ");
    Serial.print(colorIndex);
    Serial.print('\n');
}
```

W ramach obsługi przerwania wykonuje się cała sekwencja pomiarowa. Odczytywana jest wartość koloru, obliczany jest indeks koloru, nastawia się serwo na pozycję przypisaną kolorowi i wyświetlane są dane o zbadanym elemencie służące do kalibracji i debugowania.

5.3 Obsługa czujnika kolorów

Klasa SensorRGB dziedziczy po bibliotece Adafruit_NeoPixel. Wiąże się to z wykorzystaniem diody adresowalnej (ARGB). W metodzie odczytu koloru korzysta się głównie z metod zawartych w tej bibliotece.

```
/**
 * @brief Odczytuje wartość subkoloru (czerwonego, zielonego lub niebieskiego).
 *
 * @param red Wartość 1, jeśli chcesz odczytać subkolor czerwony, 0 w przeciwnym
razie.
 * @param green Wartość 1, jeśli chcesz odczytać subkolor zielony, 0 w
przeciwnym razie.
 * @param blue Wartość 1, jeśli chcesz odczytać subkolor niebieski, 0 w
przeciwnym razie.
 * @return int Odczytana wartość subkoloru.
 */
int SensorRGB::readSubColor(bool red, bool green, bool blue) {
    int measureValue;
    /**
    * Nastawianie subkoloru badanego
    */
    clear();                                                    /**< Wyczyść
kolejkę kolorów. */
    setPixelColor(0, Color(red * 255, green * 255, blue * 255));  /**< Zapisz
kolor do kolejki. */
    show();                                                     /**< Wyślij
kolor na diodę. */
    delay(30);                                                  /**<
Opóźnienie na bezwładność fotorezystora KONFIGUROWALNE. */
    measureValue = analogRead(sensorPin);                       /**< odczytaj
wartość z fotorezystora. */

    /**
    * Wyłącz czujnik po pomiarze
    */
    clear();
    show();

    return measureValue;
}
```

Najważniejsza dla czujnika jest metoda `readSubColor`. Jako argumenty przyjmuje ona wartości boolowskie dla każdego podkoloru (r, g, b). Służą one jako przełączniki zadanego na diodę koloru, np. jeżeli r = 1, dioda świeci na czerwono, r, g = 1, na żółto itd.

Opis funkcji:

- `clear();` - nastawia kolory do wysłania na diodę na 0 (brak świecenia);
- `setPixelColor(0 ,Color(red*255, green*255, blue*255));` - nastawia kolor podany przez użytkownika, przyjmuje jako argumenty: numer diody w łańcuchu, kolor w formacie RGB 0-255;
- `show();` - wysyła nastawiony kolor na diodę;
- `analogRead(sensorPin);` - odczytuje wartość analogową na zadanym pinie w formacie 0-1023;

Na bazie tej metody stworzono metody publiczne:

```
/**
 * @brief Odczytuje surowe dane z czujnika RGB.
 *
 * @param r Referencja do zmiennej, w której zostanie zapisana wartość
czerwonego subkoloru.
 * @param g Referencja do zmiennej, w której zostanie zapisana wartość zielonego
subkoloru.
 * @param b Referencja do zmiennej, w której zostanie zapisana wartość
niebieskiego subkoloru.
 */
void SensorRGB::readRawData(int &r, int &g, int &b) {
    r = readSubColor(1, 0, 0);
    g = readSubColor(0, 1, 0);
    b = readSubColor(0, 0, 1);
}
/**
 * @brief Odczytuje kolor z czujnika RGB i mapuje go na zakres 0-255.
 *
 * DO KALIBRACJI :
 * struktura funkcji: map(subColor, sensMin, sensMax, newMin, newMax);
 * Dolny przedział (odbicia wewnętrzne czujnika) - wystaw czujnik w ciemną,
pustą przestrzeń
 * Górny przedział (odbicie od obiektu) - ustaw czujnik w odległości roboczej od
białego obiektu
 *
 * @param r Referencja do zmiennej, w której zostanie zapisana wartość
czerwonego subkoloru.
 * @param g Referencja do zmiennej, w której zostanie zapisana wartość zielonego
subkoloru.
 * @param b Referencja do zmiennej, w której zostanie zapisana wartość
niebieskiego subkoloru.
 */
void SensorRGB::readColor(int &r, int &g, int &b) {
    readRawData(r, g, b);
    r  = map(r, 50, 855, 0, 255);
    g  = map(g, 30, 675, 0, 255);
    b  = map(b, 30, 570, 0, 255);
}
```

Służą one do jednoczesnego odczytu wszystkich składowych (`readRawData`) i do skorygowania odczytu poprzez dane kalibracyjne (`readColor`).

`r  = map(r, 130, 710, 0, 255);` - za pomocą proporcji skaluje jedną zmienną do drugiej, jako argumenty przyjmuje: zmienną do przeskalowania, zakres zmian przed przeskalowaniem min-max, zakres zmian po przeskalowaniu min-max.

5.4 Obsługa silnika krokowego

Ze względu na zastosowany sterownik silnika krokowego pojawiła się konieczność generowania przebiegu prostokątnego o zadanej częstotliwości. W tym celu napisano metodę:

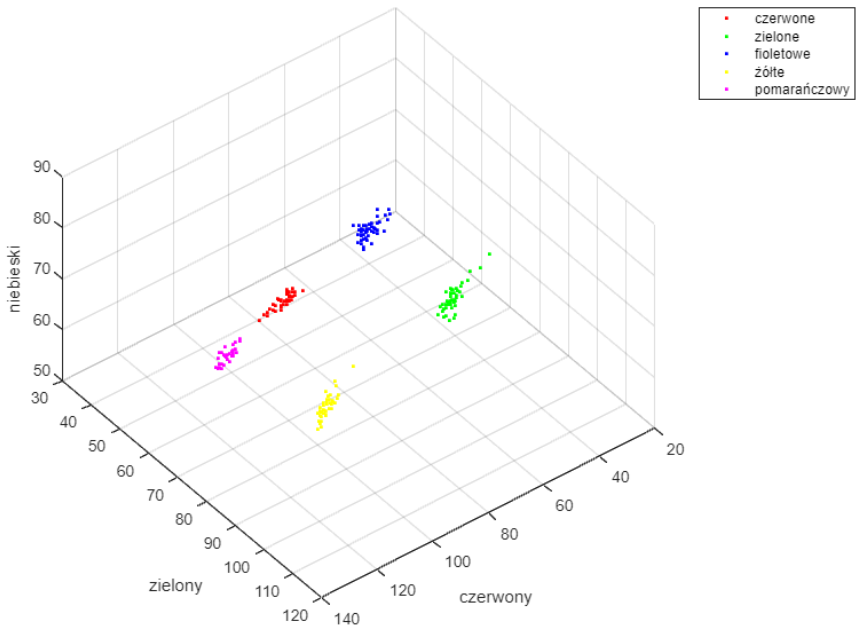
```
/**
 * @brief Podaje na sterownik określoną częstotliwość (ustala prędkość).
 *
 * @param frequency Częstotliwość tików na sterownik.
 */
void StepperMotor::makeStep(int frequency) {

  int periodMicroseconds = 1000000 / frequency;
  int halfPeriod = periodMicroseconds / 2;

  digitalWrite(motorPin, HIGH);
  delayMicroseconds(halfPeriod);

  digitalWrite(motorPin, LOW);
  delayMicroseconds(halfPeriod);
}
```

Wykorzystuje ona dyskretną manipulację stanami logicznymi wystawianymi na zadany pin. Na bazie podanej przez użytkownika częstotliwości sygnału oblicza czas połowy okresu sygnału. Następnie nastawia stan na pinie, odczekuje czas półokresu i powtarza zmieniając stan na przeciwny.

5.5 Algorytm rozpoznawania koloru i sortowania

Sercem programu jest algorytm służący do rozpoznawania kolorów.

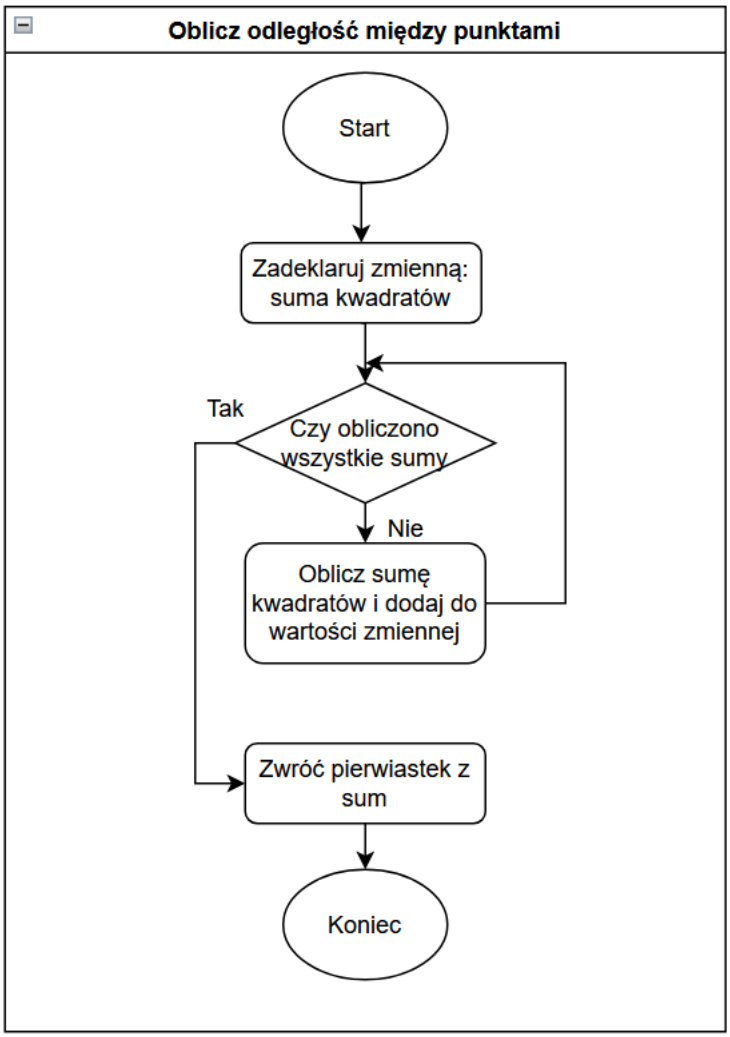
Zanim przejdzie się do implementacji, należy zrozumieć teorię stojącą za algorytmem. Każdy z możliwych kolorów cukierków można potraktować jako punkt w przestrzeni trójwymiarowej. Jeżeli jako osie w tej przestrzeni przyjmiemy wartości składowych R, G i B cukierka, każdy kolor możemy oznaczyć jako punkt w tej przestrzeni. Badając dostatecznie dużo cukierków można zauważyć, że tworzą one chmury o wielkości zależnej od wariancji kolorów [Rysunek 21].



Rysunek 21 Chmury kolorów uzyskana w ramach procedury kalibracji

Na stałe do pamięci programu zapisuje się więc uśrednione współrzędne każdego koloru. Po dokonaniu pomiaru koloru nowego cukierka jest on kolejnym punktem w przestrzeni. Badając odległość tego nowego punktu od tych zapisanych w pamięci możemy określić najmniejszą odległość między punktami. W ten sposób możemy z dużym prawdopodobieństwem przypisać nowy punkt do jednego z tych już istniejących.

Znając teorię można przejść do implementacji. Należy zacząć od najbardziej podstawowej metody opisanej schematem blokowym [Rysunek 22].



Rysunek 22 Schemat blokowy metody obliczania odległości między punktami w przestrzeni trójwymiarowej

```
/**
 * @brief Oblicza odległość między dwoma punktami w przestrzeni kolorów RGB.
 *
 * Bezpośrednia implementacja wzoru na odległość między punktami w 3d:
 * d = √((x2−x1)^2+(y2−y1)^2+(z2−z1)^2)
 *
 * @param pointA Tablica z wartościami koloru pierwszego punktu.
 * @param pointB Tablica z wartościami koloru drugiego punktu.
 * @return double Odległość między punktami.
 */
double SorterRGB::calculateDistance(int pointA[COLUMNS], int pointB[COLUMNS]) {
    double sumOfSquares = 0.0;

    for (int i = 0; i < COLUMNS; i++) {
```

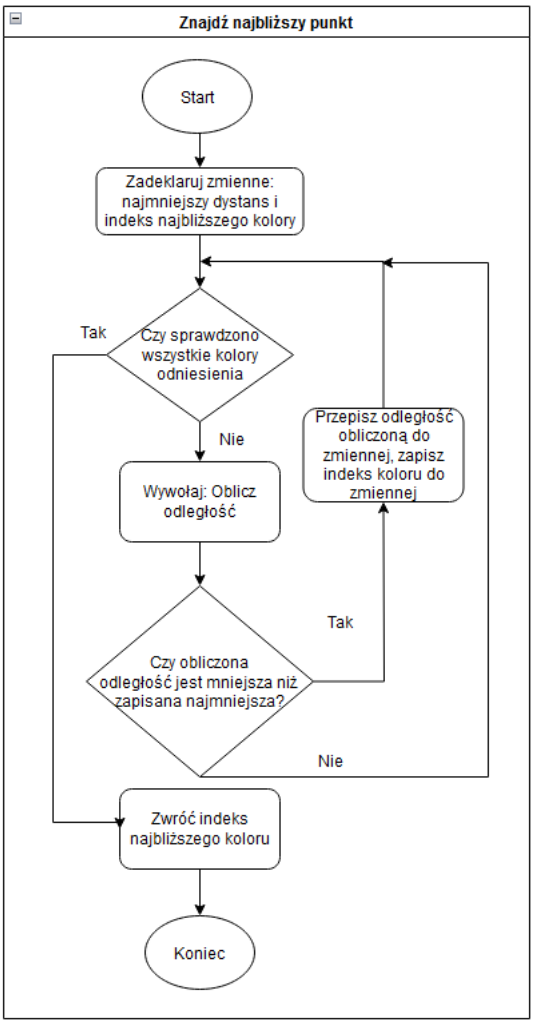
```
        sumOfSquares += pow(pointA[i] - pointB[i], 2);
    }
    return sqrt(sumOfSquares);
}
```

Jest to metoda w sposób dosłowny implementująca wzór na obliczanie odległości między punktami w przestrzeni 3d:

$$distance = \sqrt{(x - x')^2 + (y - y')^2 + (z - z')^2} \quad (4)$$

Jako argumenty przyjmuje dwa punkty w formie tabeli składające się z 3 składowych (R, G, B). Następnie dla każdego indeksu w tabeli odczytuje wartość w punkcie, odejmuje te wartości od siebie, podnosi tą różnicę do kwadratu. Każdy taki kwadrat różnicy jest sumowany. Metoda zwraca pierwiastek z obliczonej sumy.

Obliczanie odległości między punktami wykorzystane zostaje w metodzie znajdowania najmniejszej odległości. Zgodnie ze schematem blokowym [Rysunek 23] napisano kod.



Rysunek 23 Schemat blokowy metody znajdowania najmniejszego dystansu między punktem badanym a punktami odniesienia

```
/**
 * @brief Znajduje najbliższy punkt odniesienia do danego koloru.
 *
 * @param newPoint Tablica z wartościami koloru.
 * @param referencePoints Tablica z punktami odniesienia.
 * @return int Indeks najbliższego punktu odniesienia.
 */
int SorterRGB::findClosestPoint(int newPoint[COLUMNS], int
referencePoints[REFERENCE_POINTS_COUNT][COLUMNS]) {
    double smallestDistance = -1.0;
    int indexOfClosestPoint = -1;

    for (int i = 0; i < REFERENCE_POINTS_COUNT; i++) {
        double distance = abs(calculateDistance(newPoint, referencePoints[i]));
        if (smallestDistance == -1.0 || distance < smallestDistance) {
            smallestDistance = distance;
            indexOfClosestPoint = i;
        }
    }

    return indexOfClosestPoint;
}
```

Metoda `findClosestPoint` służy do porównania odległości pomiędzy punktem zmierzonym
a punktami odniesienia. Wykorzystuje w tym celu omówioną wcześniej metodę `calculateDistance`. Dla każdego punktu odniesienia oblicza odległość od nowego punktu. Jeżeli odległość jest mniejsza od zapisanej, uznana jest za najmniejszą i zapisany jest indeks koloru dla którego wystąpiła. Po skończeniu pętli, indeks zostaje zwrócony jako wynik metody.

Ostatnią składową łańcucha metod jest:

```
/**
 * @brief Zwraca indeks najbliższego koloru na podstawie wartości RGB.
 *
 * DO KALIBRACJI:
 * Manualnie posortować elementu wg. koloru
 * Określić ilość kolorów, ustawić odpowiednie @see{REFERENCE_POINTS_COUNT}
 * Każdy kolor przepuścić przez sorter mając otwartą konsolę
 * Wypisane wartości przenieść do arkusza kalkulacyjnego i uśrednić
 * Skorygować wartości w tablicy @see{referencePoints} do wartości uśrednionych
 *
 * @param R Wartość składowej czerwonej.
 * @param G Wartość składowej zielonej.
 * @param B Wartość składowej niebieskiej.
 * @return int Indeks najbliższego koloru.
```

```
 */
int SorterRGB::getClosestColorIndex(int R, int G, int B) {
    int referencePoints[REFERENCE_POINTS_COUNT][COLUMNS] = {
        {184, 102, 76},     //1 Red
        {206, 131, 94},    //2 Orange
        {122, 143, 92},     //3 Green
        {210, 179, 127},   //4 Yellow
        {112, 82, 76}     //5 Purple
      //  {0, 0, 0}
    };

    int newPoint[COLUMNS] = {R, G, B};
    int colorIndex = findClosestPoint(newPoint, referencePoints);
    return colorIndex;
}
```

W tej metodzie definiowana jest tablica `referencePoints`, która zawiera zestaw referencyjnych kolorów w formie trójek (R, G, B). Każda trójka reprezentuje jeden kolor, na przykład, `{184, 102, 76}` to kolor czerwony. Są to wartości zbadane podczas kalibracji i należy je dostosować do badanego zestawu elementów.

Następnie tworzona jest tablica `newPoint`, która zawiera wartości RGB przekazane jako argumenty do metody. Są to kolory zbadane i dostarczone wcześniej metodą czujnika kolorów.

Na podstawie wartości nowego punktu i tablicy referencyjnej wywołuje się metodę `findClosestPoint`, która zwraca indeks koloru referencyjnego najbliższego kolorowi zbadanemu.

Ostatecznie zwraca się zmienną `colorIndex`, która przechowuje obliczony najbliższy indeks. Na jego podstawie w programie głównym nastawiane jest serwo .

6. Uruchomienie, kalibracja

W niniejszej sekcji omówione zostaną: procedura rozruchowa urządzenia oraz procedura kalibracji urządzenia.

6.1 Procedura rozruchowa

Uruchomienie urządzenia wymaga odłączenia zworki P1 3) izolującej mikrokontroler. Następnie należy podłączyć źródło zasilania 9V do przewodów zasilających. Urządzenie włączy się po przełączeniu włącznika w pozycję I.

6.2 Procedura kalibracyjna

Przy kalibracji należy włożyć zworkę P1 3) podającą zasilanie z przewodu USB na elementy wykonawcze. Mikrokontroler podłączyć przez port USB do komputera i otworzyć kod źródłowy. Włączyć konsolę.

6.2.1 Kalibracja czujnika:

- W kodzie źródłowym, plik „ SensorRGB.cpp> `readColor()`” wyzerować wartości sensMin i semsMax odczytu analogowego, wgrać program na mikrokontroler
- Czujnik koloru przyłożyć do białego przedmiotu na wysokość operacyjną czujnika,
- Z konsoli odczytać kilka wartości (zależy od wariancji wyników), wartości uśrednić i zapisać do sensMax,
- Czujnik koloru zwrócić w ciemną, otwartą przestrzeń (np. pod stół) minimalizując ilość odbić,
- Z konsoli odczytać wartości, uśrednić, zapisać do sensMin,
- Program wgrać na mikrokontroler.

6.2.2 Kalibracja sortownika

- W kodzie źródłowym, plik „ SorterRGB.h> `REFERENCE_POINTS_COUNT`” ustalić ilość kolorów obsługiwanych przez sortownik,
- W kodzie źródłowym, plik „ SorterRGB.cpp> `getClosestColorIndex()`” stworzyć odpowiednią ilość punktów odniesienia, zgodną z zadaną ilością kolorów,
- Manualnie posortować elementy badane, przepuścić przez urządzenie odczytując wartości w konsoli,
- Przenieść dane z pomiaru kilkunastu – kilkudziesięciu elementów do arkusza kalkulacyjnego i obliczyć średnią/medianę,
- Do każdego punktu odniesienia zapisać wynik obliczeń z arkusza,
- Program wgrać na mikrokontroler.

7. Pomiary testowe

Przeprowadzone pomiary testowe wykazują dużą i powtarzalną dokładność czujnika. Zestawienie danych technicznych skonstruowanego urządzenia zaprezentowano w tabeli niżej.

Tabela 2 Dane techniczne urządzenia.

Dane techniczne	**Wartość**	**Jednostka**
Wysokość	355	mm
Szerokość	130	mm
Długość	195	mm
Maksymalny promień obiektu	5	mm
Maksymalna wysokość obiektu	10	mm
Waga	859	g
Napięcie wejściowe	9	V
Prąd wejściowy	1	A
Maksymalny błąd pomiaru	5	%
Typowy błąd pomiaru	2	%

Wartość maksymalnego błędu pomiaru podana w tabeli wyżej jest wartością maksymalną błędu, jaki urządzenie może osiągnąć przy prawidłowej kalibracji. Typowy błąd urządzenia wynosi w okolicach 2%. W przypadku, gdy błąd przekracza podaną maksymalną wartość należy powtórzyć kalibrację, opisaną w punkcie 6.2.2 (Kalibracja sortownika).

Rysunek 24 Kolory jakie powstają po odczytaniu wartości kalibracyjnych.

Analizując kolory powstałe po odczytaniu wartości kalibracyjnych dla poszczególnych kolorów pokazane na obrazie wyżej, można zaobserwować znaczącą różnicę w kolorze pomiędzy poszczególnymi kolorami. Kolory odczytane odpowiadają kolorom spodziewanym. Można zaobserwować różnice między kolorem, a realnym odcieniem badanego obiektu. Ma na to wpływ fakt, iż każdy cukierek jest odmienny, niektóre cukierki posiadają literę „m” na sobie, inne posiadają tę literę słabo widoczną, zaś inne nie posiadają jej w ogóle. Dodatkowo niektóre

cukierki użyte do kalibracji posiadały uszczerbki, co skutkowało pojawieniem się na nich dodatkowych białych kawałków, co może powodować „wybielenie” odczytanego odcienia. Dodatkowo znaczący wpływ na odczyt ma użyta do przemieszczania tarcza, która również odbija część światła.

8. Instrukcja obsługi

Sorter cukierków oparty na czujniku kolorów został zaprojektowany do automatycznego rozpoznawania i sortowania cukierków na podstawie ich kolorów. Ta instrukcja pomoże Ci w szybkim i skutecznym korzystaniu z urządzenia.

Tabela 1 Dane techniczne sortera

Dane techniczne	**Wartość**	**Jednostka**
Wymiary sortera (W/D/H)	130/195/355	mm
Maksymalny promień obiektu sortowanego	5	mm
Maksymalna wysokość obiektu sortowanego	10	mm
Waga sortera	859	g
Napięcie wejściowe	9	V
Prąd wejściowy	1	A
Maksymalny błąd pomiaru	5	%
Typowy błąd pomiaru	2	%

Instrukcja obsługi:

1. Ustaw urządzenie na płaskiej, równej powierzchni.
2. Umieść przed sorterem pojemnik na posortowane cukierki.
3. Podłącz zasilacz o napięciu znamionowym 9V i wydajności prądowej minimum 1 A.
4. Uruchom sorter ustawiając przełącznik zasilania w pozycje I.
5. Odczekaj chwilę, aż zakończy się proces inicjalizacji.
6. Wsyp do zasobnika cukierki do posortowania.
7. Po zakończeniu sortowania wyłącz urządzenie ustawiając przełącznik zasilania w pozycję O.

Czyszczenie i konserwacja:

1. Regularnie czyść urządzenie z zewnątrz przy użyciu wilgotnej ściereczki wraz z wnętrzem zasobnika na cukierki. Upewnij się, że sorter jest wyłączony przed rozpoczęciem czyszczenia.

Rozwiązywanie problemów:

1. Urządzenie nie reaguje
   1.1.Sprawdź, czy zasilacz jest poprawnie podłączony.
   1.2.Wyłącz i ponownie włącz sorter cukierków.
   1.3.Odblokuj wlot do komory transportowej za pomocą długiego patyczka.
2. Urządzenie nie sortuje poprawnie.
   2.1.Sprawdź wartość napięcia zasilania.
   2.2.Wyczyść urządzenie.
   2.3.Zleć kalibrację urządzenia autoryzowanemu serwisowi.

Informacje kontaktowe

W razie problemów lub pytań skontaktuj się z grupą projektową.

E-mail: 263671@student.pwr.edu.pl
E-mail: 263686@student.pwr.edu.pl
E-mail: 263581@student.pwr.edu.pl

9. Podsumowanie

Celem projektu było stworzenie efektywnego urządzenia do sortowania kolorowych cukierków. Wykorzystano własnego pomysłu i wykonania czujnik kolorów, precyzyjny mechanizm podawania oraz szybki i skuteczny mechanizm sortowania.

Urządzenie osiągnęło imponujące rezultaty podczas testów. Zdolność rozpoznawania kolorów była na wysokim poziomie. Jedyne pomyłki zdarzały się tylko w przypadku kolorów bardzo zbliżonych do siebie. Skuteczność sortowania wyniosła 98% dla próbki statystycznej wielkości 146. Mechanizmy manipulacyjne działały sprawnie, umożliwiając precyzyjne przenoszenie i układanie cukierków zgodnie z ich barwą. W rezultacie uzyskano efektywny i niezawodny system sortowania.

Podczas budowy urządzenia napotkano na liczne problemy techniczne.

Pierwsze wersje czujnika kolorów charakteryzowały się słabym odwzorowaniem koloru badanego w rzeczywistości. Pomimo ciągłych iteracji nadal nie udało się uzyskać perfekcyjnego oddania barw. Problemy w każdej z iteracji były związane z kilkoma czynnikami: jakością wykorzystanych elementów (tanie diody i fotorezystory), geometrią czujnika (kąt nachylenia diody i fotorezystora), projektem obudowy czujnika (wysokość przegrody, długość szyjki czujnika). Jako że projekt czujnika jest wielopoziomowy nie da się dojść do optymalnego rozwiązania w sposób iteracyjny. Zagadnienie jest jednak poza obszarem wykonywania skrupulatnych obliczeń. Postawiono więc na projekt, który oferował racjonalne wyniki pomiarów choć wyniki mogłyby być jeszcze lepsze.

Pierwotnie, projekt nie uwzględniał zastosowania bariery optycznej. Określanie pozycji tarczy wiązało się z programowym liczeniem ilości kroków wykonanych przez silnik. Okazało się jednak, że brak sprzężenia zwrotnego powodował problemy np. w wypadku zaklinowania elementu badanego w mechanizmie. Nawet chwilowe zgubienie kroków rozstrajało kluczowe złożenie otworu tarczy i czujnika. Podjęto próbę programowego naprawienia błędu wprowadzając algorytm powrotu do pozycji poprawnej, lecz zmniejszało to niezawodność i szybkość pracy urządzenia.

System podajnika cukierków mógłby być projektem samym w sobie. Podczas testów, wielokrotnie zdarzały się zacięcia. Aby poprawić działanie systemu podawania, można by w lejku zamontować urządzenie wibracyjne lub mieszające. W efekcie dochodziłoby do ciągłej lub periodycznej agitacji cukierków a co za tym idzie utrudniałoby to ich blokowanie w mechanizmie podającym. Można by również zastosować mechanizm wybierający. Zaprojektować miskę z rosnącą spiralą po jej obwodzie. Wsypane cukierki wspinałyby się po spirali wraz z obrotem kołowym miski. Niestety i takie rozwiązanie niesie za sobą swoją gammę problemów. Ostatecznie postawiono na prosty lejek odtykany manualną interwencją operatora zdając sobie sprawę, że nie jest to rozwiązanie stałe.

Temat projektu, choć trywialny, niesie za sobą sporą gammę możliwości. O ile sortowani cukierków może nie być przydatne, tak znajomość technologii zastosowanych w projekcie może służyć jako kamień węgielny w projektach chociażby z dziedziny kontroli jakości. Dzięki modularnej strukturze możliwe jest iterowanie na podstawie tego projektu i adaptacja go do wielu dziedzin niezwiązanych z przemysłem spożywczym.

10. Bibliografia

[1] „Wikipedia,” [Online]. Available: https://en.wikipedia.org/wiki/Color. [Data uzyskania dostępu: 12 01 2024].

[2] „Wikipedia,” [Online]. Available: https://en.wikipedia.org/wiki/Photoresistor. [Data uzyskania dostępu: 12 01 2024].

[3] Token, [Online]. Available: https://www.tme.eu/Document/0b7aec6d26675b47f9e54d893cd4521b/PGM5506.pdf. [Data uzyskania dostępu: 12 01 2024].

[4] „Wikipedia,” [Online]. Available: https://en.wikipedia.org/wiki/Light-emitting_diode. [Data uzyskania dostępu: 12 01 2024].

[5] Worldsemi, [Online]. Available: https://www.tme.eu/Document/db42ace4406f6d6cc84f1108a91ee0d5/WS2812B-B.pdf. [Data uzyskania dostępu: 12 01 2024].

[6] „ROHM Semiconductor,” [Online]. Available: https://www.rohm.com/electronics-basics/sensor/color-sensor. [Data uzyskania dostępu: 12 01 2024].

[7] „Wikipedia,” [Online]. Available: https://pl.wikipedia.org/wiki/Dopasowanie_energetyczne. [Data uzyskania dostępu: 12 01 2024].

[8] „TME,” [Online]. Available: https://www.tme.eu/pl/details/h22a3-i/czujniki-fotoelektryczne-do-druku/isocom/h22a3/?brutto=1&currency=PLN&gad_source=1&gclid=CjwKCAiAqY6tBhAtEiwAHeRopZan5SCk1nVSLp4VQxe7HVg-SL6gehXsqGUSz9icI0Wti3EfSDFjOhoCiPAQAvD_BwE. [Data uzyskania dostępu: 14 01 2024].

[9] CDIL, [Online]. Available: https://www.tme.eu/Document/8aa8ed83dc2f8e8a5f3ad4a71fd201bc/lm7805.pdf. [Data uzyskania dostępu: 19 01 2024].

[10] ATmega, [Online]. Available: https://ww1.microchip.com/downloads/en/DeviceDoc/40001906A.pdf. [Data uzyskania dostępu: 14 01 2024].

[11] Arduino, [Online]. Available: https://store.arduino.cc/products/arduino-nano. [Data uzyskania dostępu: 14 01 2024].

[12] TowerPro, [Online]. Available: https://html.alldatasheet.com/html-pdf/1540157/ETC2/SG5010/110/1/SG5010.html. [Data uzyskania dostępu: 14 01 2024].

[13] Japan Servo, [Online]. Available: https://datasheetspdf.com/pdf-file/1399850/Servo/KH42JM2/1. [Data uzyskania dostępu: 14 01 2024].

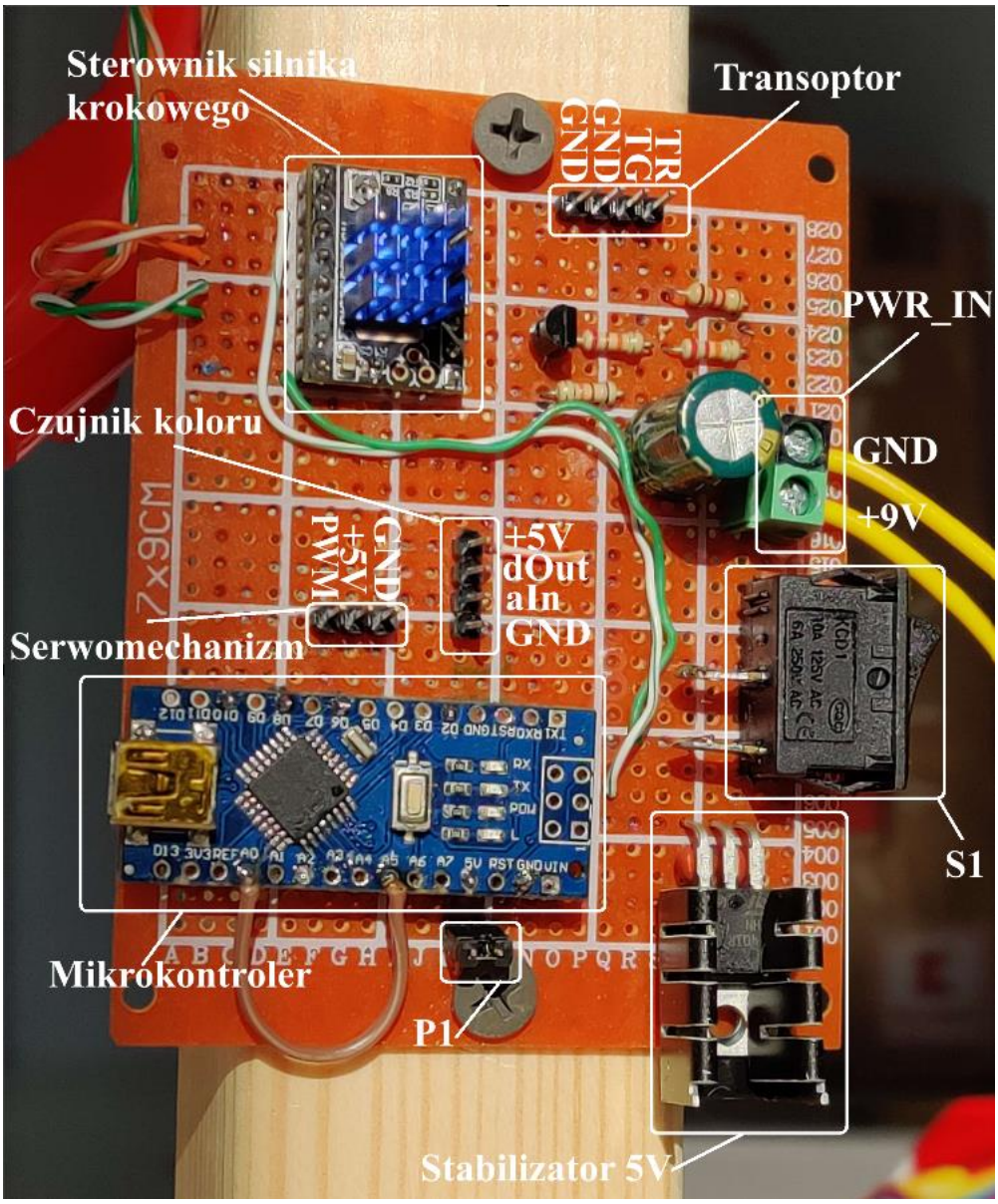
[14] „Kamami,” [Online]. Available: https://kamami.pl/elektronika/1186060-sterownik-bipolarnego-silnika-krokowego-z-ukladem-tmc2225.html. [Data uzyskania dostępu: 14 01 2024].

[15] Trinamic, [Online]. Available: https://www.analog.com/media/en/technical-documentation/data-sheets/TMC2225_datasheet_rev1.14.pdf. [Data uzyskania dostępu: 14 01 2024].

11. Dodatki

11.1 Opis gotowej płytki

Poniżej przedstawiono zdjęcie gotowej płytki wraz z podpisaniem poszczególnych wtyków.



Rysunek 25 Końcowy projekt płytki PCB wraz z podpisami.

Na powyższej płytce oznaczono piny do których należy podłączyć następujące rzeczy:

1) Serwomechanizm:
   a) PWM – podłączyć żółty przewód od serwomechanizmu.
   b) +5V – podłączyć pomarańczowy przewód od serwomechanizmu.
   c) GND – podłączyć brązowy przewód od serwomechanizmu.
2) Mikrokontroler ATmega328P na płytce Arduino Nano.
3) Zworka P1 – założenie zworki odpowiada podpięciu zasilania +5V z stabilizatora na mikrokontroler.

4) Stabilizator +5V wraz z radiatorem.
5) S1 – główny włącznik napięcia.
6) PWR_IN – złącze zasilania układu. Czarnym kolorem oznaczona kostka, do której należy podpiąć masę, do drugiej kostki należy podłączyć zasilanie +9V.
7) Transoptor – odpowiada generacji przerwania układu:
   a) GND – należy podłączyć brązowy przewód od transoptora.
   b) GND – należy podłączyć niebieski przewód od transoptora.
   c) TG – należy podłączyć zielony przewód od transoptora.
   d) TR – należy podłączyć czerwony przewód od transoptora.
8) Sterownik silnika krokowego.
9) Czujnik koloru:
   a) +5V – należy podłączyć czerwony przewód od czujnika.
   b) dOut –ustawienie diody w czujniku. Należy podłączyć pomarańczowy przewód od czujnika.
   c) aIn – odczyt wartości z fotorezystora w czujniku. Należy podłączyć żółty przewód od czujnika.
   d) GND – należy podłączyć czarny przewód od czujnika.

11.2Listingi programu

Poniżej znajdują się pełne listingi wszystkich plików stworzonych w ramach programu obsługi sortera.

11.2.1 Skittles_Sorter.ino

```
1. #include "SorterRGB.h"
2. #include "SensorRGB.h"
3. #include "StepperMotor.h"
4. #include <Servo.h>
5.
6. /**
7.  * @brief Skittles_Sorter - główny plik wykonawczy dla urządzenia sortującego kolorowe obiekty.
8.  *
9.  * Program zawiera: konfigurację pinów mikrokontrolera, kostruktory urządzeń składowych,
10.       * obsługę głównej pętli programu oraz obsługę przerwania.
11.       *
12.       * Autorzy: 263581, 263671, 263686
13.       * Data ostatniej modyfikacji: Styczeń 19, 2024
14.       */
15.
16.        /**
17.        * DO KONFIGURACJI:
18.        * Numery pinów ustawić zgodnie ze schematem połączeń w wypadku użycia własnego modułu sterowania!
```

```
19.         * Ważne:
20.         * SENSOR_SENS_PIN - dowolny pin analogowy
21.         * SERVO_PIN - dowolny pin obsługujący PWM
22.         */
23.       #define NUMPIXELS 1            /**< Liczba pikseli w sensorze
    RGB. */
24.       #define SENSOR_CTRL_PIN 8      /**< Numer pinu sterującego
    sensorem RGB. */
25.       #define SENSOR_SENS_PIN A5     /**< Numer pinu odczytu sygnału
    z sensora RGB (pin analogowy). */
26.
27.       #define MOTOR_CTRL_PIN 10      /**< Numer pinu sterującego
    silnikiem krokowym. */
28.       #define SERVO_PIN 6            /**< Numer pinu sterującego
    serwem. */
29.       #define INTERRUPT_PIN 2        /**< Numer pinu przerwania
    związanego z transoptorem. */
30.
31.       ////////////////////////////////////////////////////////////////
    ////////////////////////////////////////////////////
32.
33.       SensorRGB sensor(NUMPIXELS, SENSOR_CTRL_PIN, NEO_GRB +
    NEO_KHZ800, SENSOR_SENS_PIN);
34.       StepperMotor stepper(MOTOR_CTRL_PIN);
35.       Servo sinkServo;
36.       SorterRGB sorter;
37.
38.       int R, G, B;
39.       int colorIndex = 0;
40.       int motorSpeed = 270;
41.
42.       ////////////////////////////////////////////////////////////////
    ////////////////////////////////////////////////////
43.
44.       /**
45.        * @brief Funkcja inicjalizująca serwo, przerwanie oraz
    komunikację szeregową.
46.        */
47.       void setup() {
48.           sinkServo.attach(SERVO_PIN);
49.
50.           pinMode(INTERRUPT_PIN, INPUT_PULLUP);
51.           attachInterrupt(digitalPinToInterrupt(INTERRUPT_PIN),
    handleInterrupt, RISING);
52.
53.           Serial.begin(9600);
54.       }
55.
56.       ////////////////////////////////////////////////////////////////
    ////////////////////////////////////////////////////
57.
58.       /**
59.        * @brief Pętla główna programu.
60.        */
61.       void loop() {
62.           stepper.makeStep(motorSpeed);
63.       }
64.
65.       /**
66.        * @brief Funkcja obsługująca przerwanie związane z odczytem
    koloru po wyzwoleniu transoptora.
```

```
67.         *
68.         * > Zczytuje kolory nowego obiektu
69.         * > Określa kolor obiektu na podstawie kolorów odniesienia
70.         * > Nastawia serwo na podstawie koloru
71.         * > Wyświetla w konsoli dane kalibracyjne
72.         */
73.       void handleInterrupt() {
74.
75.           sensor.readColor(R, G, B);
76.           colorIndex = sorter.getClosestColorIndex(R, G, B);
77.           sinkServo.write(colorIndex*45);
78.
79.           Serial.print(R);
80.           Serial.print(',');
81.           Serial.print(G);
82.           Serial.print(',');
83.           Serial.print(B);
84.           Serial.print('\n');
85.
86.           Serial.print("indeks: ");
87.           Serial.print(colorIndex);
88.           Serial.print('\n');
89.   }
```

11.2.2 SorterRGB.h

```
1.  #ifndef SORTERRGB_H
2.  #define SORTERRGB_H
3.
4.  #define COLUMNS 3  // Liczba składowych: 3 (r + g + b)
5.  #define REFERENCE_POINTS_COUNT 5  // Liczba punktów odniesienia w
    tablicy
6.
7.  #include <Arduino.h>
8.
9.  #include <math.h>
10.
11.       /**
12.        * @brief SorterRGB - Klasa reprezentująca sortownik RGB.
13.        *
14.        * Klasa zawiera metody rozpoznawania koloru oraz sortowania
    na jego podstawie. Osiąga to poprzez
15.        * określenie indeksu koloru odniesienia najbliższego do
    koloru zbadanego.
16.        *
17.        * Autorzy: 263581, 263671, 263686
18.        * Data ostatniej modyfikacji: Styczeń 19, 2024
19.        */
20.       class SorterRGB {
21.
22.       private:
23.           //Servo sinkServo;     /**< Obiekt serwo do sterowania
    sortownikiem. */
24.           //int servoPin;        /**< Numer pinu do podłączenia
    serwa. */
25.
26.           /**
```

```
27.           * @brief Oblicza odległość między dwoma punktami w
    przestrzeni kolorów RGB (3d).
28.           *
29.           * @param pointA Tablica z wartościami koloru pierwszego
    punktu.
30.           * @param pointB Tablica z wartościami koloru drugiego
    punktu.
31.           * @return double Odległość między punktami.
32.           */
33.          double calculateDistance(int pointA[COLUMNS], int
    pointB[COLUMNS]);
34.
35.          /**
36.           * @brief Znajduje najbliższy punkt odniesienia do
    zmierzonego punktu w przestrzeni kolorów RGB (3d).
37.           *
38.           * @param newPoint Tablica z wartościami koloru.
39.           * @param referencePoints Tablica z punktami odniesienia.
40.           * @return int Indeks najbliższego punktu odniesienia.
41.           */
42.          int findClosestPoint(int newPoint[COLUMNS], int
    referencePoints[REFERENCE_POINTS_COUNT][COLUMNS]);
43.
44.      public:
45.          /**
46.           * @brief Konstruktor klasy SorterRGB.
47.           *
48.           * @param servoPin Numer pinu, do którego podłączone jest
    serwo.
49.           */
50.          SorterRGB();
51.
52.          /**
53.           * @brief Zwraca indeks najbliższego koloru na podstawie
    wartości RGB zbadanego obiektu.
54.           *
55.           * @param R Wartość składowej czerwonej.
56.           * @param G Wartość składowej zielonej.
57.           * @param B Wartość składowej niebieskiej.
58.           * @return int Indeks najbliższego koloru.
59.           */
60.          int getClosestColorIndex(int R, int G, int B);
61.      };
62.
63. #endif
```

11.2.3 SorterRGB.cpp

```
1. #include "SorterRGB.h"
2.
3. /**
4.  * @brief Konstruktor klasy SorterRGB.
5.  *
6.  * @param newServo Obiekt serwa, którym będzie sterować program.
7.  * @param pin Numer pinu, do którego podłączone jest serwo.
8.  */
9. SorterRGB::SorterRGB(){
10.      }
```

```
11.
12.      /**
13.       * @brief Oblicza odległość między dwoma punktami w
   przestrzeni kolorów RGB.
14.       *
15.       * Bezpośrednia implementacja wzoru na odległość między
   punktami w 3d:
16.       * d = √((x2-x1)^2+(y2-y1)^2+(z2-z1)^2)
17.       *
18.       * @param pointA Tablica z wartościami koloru pierwszego
   punktu.
19.       * @param pointB Tablica z wartościami koloru drugiego
   punktu.
20.       * @return double Odległość między punktami.
21.       */
22.      double SorterRGB::calculateDistance(int pointA[COLUMNS], int
   pointB[COLUMNS]) {
23.          double sumOfSquares = 0.0;
24.
25.          for (int i = 0; i < COLUMNS; i++) {
26.              sumOfSquares += pow(pointA[i] - pointB[i], 2);
27.          }
28.          return sqrt(sumOfSquares);
29.      }
30.
31.      /**
32.       * @brief Znajduje najbliższy punkt odniesienia do danego
   koloru.
33.       *
34.       * @param newPoint Tablica z wartościami koloru.
35.       * @param referencePoints Tablica z punktami odniesienia.
36.       * @return int Indeks najbliższego punktu odniesienia.
37.       */
38.      int SorterRGB::findClosestPoint(int newPoint[COLUMNS], int
   referencePoints[REFERENCE_POINTS_COUNT][COLUMNS]) {
39.          double smallestDistance = -1.0;
40.          int indexOfClosestPoint = -1;
41.
42.          for (int i = 0; i < REFERENCE_POINTS_COUNT; i++) {
43.              double distance = abs(calculateDistance(newPoint,
   referencePoints[i]));
44.              if (smallestDistance == -1.0 || distance <
   smallestDistance) {
45.                  smallestDistance = distance;
46.                  indexOfClosestPoint = i;
47.              }
48.          }
49.
50.          return indexOfClosestPoint;
51.      }
52.
53.      /**
54.       * @brief Zwraca indeks najbliższego koloru na podstawie
   wartości RGB.
55.       *
56.       * DO KALIBRACJI :
57.       * Manulanie posortować elementu wg. koloru
58.       * Określić ilość kolorów, ustawić odpowiednie
   @see{REFERENCE_POINTS_COUNT}
59.       * Każdy kolor przepuścić przez sorter mając otwartą konsolę
```

```
60.       * Wypisane wartości przenieść do arkusza kalkulacyjnego i
   uśrednić
61.       * Skorygować wartości w tablicy @see{referencePoints} do
   wartości uśrednionych
62.       *
63.       * @param R Wartość składowej czerwonej.
64.       * @param G Wartość składowej zielonej.
65.       * @param B Wartość składowej niebieskiej.
66.       * @return int Indeks najbliższego koloru.
67.       */
68.      int SorterRGB::getClosestColorIndex(int R, int G, int B) {
69.          int referencePoints[REFERENCE_POINTS_COUNT][COLUMNS] = {
70.              {184, 102, 76},     //1 Red
71.              {206, 131, 94},    //2 Orange
72.              {122, 143, 92},     //3 Green
73.              {210, 179, 127},   //4 Yellow
74.              {112, 82, 76}     //5 Purple
75.            //  {0, 0, 0}
76.          };
77.
78.          int newPoint[COLUMNS] = {R, G, B};
79.          int colorIndex = findClosestPoint(newPoint,
   referencePoints);
80.          return colorIndex;
81.  }
```

11.2.4 SensorRGB.h

```
1.  #ifndef SENSORRGB_H
2.  #define SENSORRGB_H
3.
4.  #include <Arduino.h>
5.  #include <Adafruit_NeoPixel.h>  /** < versja 1.12.0 */
6.
7.  /**
8.   * @brief SensorRGB - Klasa reprezentująca czujnik RGB.
9.   *
10.       * Klasa dziedziczy po klasie Adafruit_NeoPixel, co umożliwia
   korzystanie z funkcji obsługujących diody NeoPixel.
11.       * Pozwala na odczytywanie kolorów z czujnika w kilku
   określonych trybach:
12.       * readRawData -> dane RGB prosto z czujnika
13.       * readColor -> dane RGB skorygowane o wartości kalibracyjne
14.       * readGrayscale -> dane w sakli szarości
15.       *
16.       * Autorzy: 263581, 263671, 263686
17.       * Data ostatniej modyfikacji: Styczeń 19, 2024
18.       */
19.
20.      class SensorRGB : public Adafruit_NeoPixel {
21.      private:
22.          int sensorPin; /**< Numer pinu, do którego podłączony
   jest czujnik. */
23.
24.          /**
25.           * @brief Odczytuje wartość zadanego subkoloru
   (czerwonego, zielonego lub niebieskiego).
26.           *
```

```
27.           * @param red Wartość 1, jeśli chcesz odczytać subkolor
    czerwony, 0 w przeciwnym razie.
28.           * @param green Wartość 1, jeśli chcesz odczytać subkolor
    zielony, 0 w przeciwnym razie.
29.           * @param blue Wartość 1, jeśli chcesz odczytać subkolor
    niebieski, 0 w przeciwnym razie.
30.           * @return int Odczytana wartość subkoloru.
31.           */
32.          int readSubColor(bool red, bool green, bool blue);
33.
34.      public:
35.          /**
36.           * @brief Konstruktor klasy SensorRGB.
37.           *
38.           * @param numPixels Liczba pikseli obsługiwanych przez
    diody NeoPixel.
39.           * @param ctrlPin Numer pinu sterującego diodami
    NeoPixel.
40.           * @param type Typ diod NeoPixel.
41.           * @param sensorPin Numer pinu, do którego podłączony
    jest czujnik.
42.           */
43.          SensorRGB(uint16_t numPixels, uint8_t ctrlPin,
    neoPixelType type, int sensorPin);
44.
45.          /**
46.           * @brief Odczytuje surowe dane z czujnika RGB.
47.           *
48.           * @param r Referencja do zmiennej, w której zostanie
    zapisana wartość czerwonego subkoloru.
49.           * @param g Referencja do zmiennej, w której zostanie
    zapisana wartość zielonego subkoloru.
50.           * @param b Referencja do zmiennej, w której zostanie
    zapisana wartość niebieskiego subkoloru.
51.           */
52.          void readRawData(int &r, int &g, int &b);
53.
54.          /**
55.           * @brief Odczytuje kolor z czujnika RGB i mapuje go na
    zakres 0-255.
56.           *
57.           * @param r Referencja do zmiennej, w której zostanie
    zapisana wartość czerwonego subkoloru.
58.           * @param g Referencja do zmiennej, w której zostanie
    zapisana wartość zielonego subkoloru.
59.           * @param b Referencja do zmiennej, w której zostanie
    zapisana wartość niebieskiego subkoloru.
60.           */
61.          void readColor(int &r, int &g, int &b);
62.
63.          /**
64.           * @brief Odczytuje wartość w skali szarości z czujnika
    RGB.
65.           *
66.           * @param w Referencja do zmiennej, w której zostanie
    zapisana wartość subkoloru szarości.
67.           */
68.          void readGrayscale(int &w);
69.      };
70.
71. #endif
```

11.2.5 SensorRGB.cpp

```
1. #include "SensorRGB.h"
2.
3. /**
4.  * @brief Konstruktor klasy SensorRGB.
5.  *
6.  * @param numPixels Liczba pikseli obsługiwanych przez diody
   NeoPixel.
7.  * @param ctrlPin Numer pinu sterującego diodami NeoPixel.
8.  * @param type Typ diod NeoPixel.
9.  * @param sensorPin Numer pinu, do którego podłączony jest czujnik.
10.       */
11.      SensorRGB::SensorRGB(uint16_t numPixels, uint8_t ctrlPin,
   neoPixelType type, int sensorPin) : Adafruit_NeoPixel(numPixels,
   ctrlPin, type) {
12.          this->sensorPin = sensorPin;
13.          begin();
14.      }
15.
16.      /**
17.       * @brief Odczytuje wartość subkoloru (czerwonego, zielonego
   lub niebieskiego).
18.       *
19.       * @param red Wartość 1, jeśli chcesz odczytać subkolor
   czerwony, 0 w przeciwnym razie.
20.       * @param green Wartość 1, jeśli chcesz odczytać subkolor
   zielony, 0 w przeciwnym razie.
21.       * @param blue Wartość 1, jeśli chcesz odczytać subkolor
   niebieski, 0 w przeciwnym razie.
22.       * @return int Odczytana wartość subkoloru.
23.       */
24.      int SensorRGB::readSubColor(bool red, bool green, bool blue)
   {
25.          int measureValue;
26.          /**
27.          * Nastawianie subkoloru badanego
28.          */
29.          clear();
        /**< Wyczyść kolejkę kolorów. */
30.          setPixelColor(0, Color(red * 255, green * 255, blue *
   255));  /**< Zapisz kolor do kolejki. */
31.          show();
      /**< Wyślij kolor na diodę. */
32.          delay(30);
       /**< Opóźnienie na bezwładnoeść fotorezystora KONFIGUROWALNE.
   */
33.          measureValue =
   analogRead(sensorPin);                          /**< odczytaj
   wartość z fotorezystora. */
34.
35.          /**
36.          * Wyłącz czujnik po pomiarze
37.          */
38.          clear();
39.          show();
40.
41.          return measureValue;
```

```
42.       }
43.
44.       /**
45.        * @brief Odczytuje surowe dane z czujnika RGB.
46.        *
47.        * @param r Referencja do zmiennej, w której zostanie
   zapisana wartość czerwonego subkoloru.
48.        * @param g Referencja do zmiennej, w której zostanie
   zapisana wartość zielonego subkoloru.
49.        * @param b Referencja do zmiennej, w której zostanie
   zapisana wartość niebieskiego subkoloru.
50.        */
51.       void SensorRGB::readRawData(int &r, int &g, int &b) {
52.           r = readSubColor(1, 0, 0);
53.           g = readSubColor(0, 1, 0);
54.           b = readSubColor(0, 0, 1);
55.       }
56.
57.       /**
58.        * @brief Odczytuje kolor z czujnika RGB i mapuje go na
   zakres 0-255.
59.        *
60.        * DO KALIBRACJI :
61.        * struktura funkcji: map(subColor, sensMin, sensMax, newMin,
   newMax);
62.        * Dolny przedział (odbicia wewnętrzne czujnika) - wystaw
   czujnik w ciemną, pustą przestrzeń
63.        * Górny przedział (odbicie od obiektu) - ustaw czujnik w
   odległości roboczej od białego obiektu
64.        *
65.        * @param r Referencja do zmiennej, w której zostanie
   zapisana wartość czerwonego subkoloru.
66.        * @param g Referencja do zmiennej, w której zostanie
   zapisana wartość zielonego subkoloru.
67.        * @param b Referencja do zmiennej, w której zostanie
   zapisana wartość niebieskiego subkoloru.
68.        */
69.       void SensorRGB::readColor(int &r, int &g, int &b) {
70.           readRawData(r, g, b);
71.           r  = map(r, 50, 855, 0, 255);
72.           g  = map(g, 30, 675, 0, 255);
73.           b  = map(b, 30, 570, 0, 255);
74. }
```

11.2.6 StepperMotor.h

```
1. #ifndef STEPPERMOTOR_H
2. #define STEPPERMOTOR_H
3.
4. #include <Arduino.h>
5.
6. /**
7.  * @brief StepperMotor - Klasa sterowania silnikiem krokowym.
8.  *
9.  * Klasa umożliwia sterowanie silnikiem krokowym poprzez
   generowanie impulsów dla sterownika
10.        * TMC2225. Sterownik każdy impuls odczytywany jest jako
   sygnał i wykonuje krok/mikrokrok. Zmieniając
11.        * częstotliwoćś sygnału, zmieia się szybkość obrotu silnika.
12.        *
```

```
13.       * Autorzy: 263581, 263671, 263686
14.       * Data ostatniej modyfikacji: Styczeń 19, 2024
15.       */
16.
17.      class StepperMotor {
18.      private:
19.          int motorPin; /**< Numer pinu, do którego podłączony jest
    silnik. */
20.
21.      public:
22.          /**
23.           * @brief Konstruktor klasy StepperMotor.
24.           *
25.           * @param motorPin Numer pinu, do którego podłączony jest
    silnik.
26.           */
27.          StepperMotor(int motorPin);
28.
29.          /**
30.           * @brief Uruchamia silnik krokowy z określoną
    częstotliwością obrotów.
31.           *
32.           * @param frequency Częstotliwość obrotów silnika.
33.           */
34.          void makeStep(int frequency);
35.
36.      };
37.
38. #endif
```

11.2.7 StepperMotor.cpp

```
1.  #include "StepperMotor.h"
2.
3.  /**
4.   * @brief Konstruktor klasy StepperMotor.
5.   *
6.   * @param motorPin Numer pinu, do którego podłączony jest sterownik
    silnika.
7.   */
8.  StepperMotor::StepperMotor(int motorPin) {
9.      this->motorPin = motorPin;
10.       }
11.       /**
12.        * @brief Podaje na sterownik określoną częstotliwość (ustala
    prędkość).
13.        *
14.        * @param frequency Częstotliwość tików na sterownik.
15.        */
16.       void StepperMotor::makeStep(int frequency) {
17.
18.         int periodMicroseconds = 1000000 / frequency;
19.         int halfPeriod = periodMicroseconds / 2;
20.
21.         digitalWrite(motorPin, HIGH);
22.         delayMicroseconds(halfPeriod);
23.
24.         digitalWrite(motorPin, LOW);
25.         delayMicroseconds(halfPeriod);
26. }
```