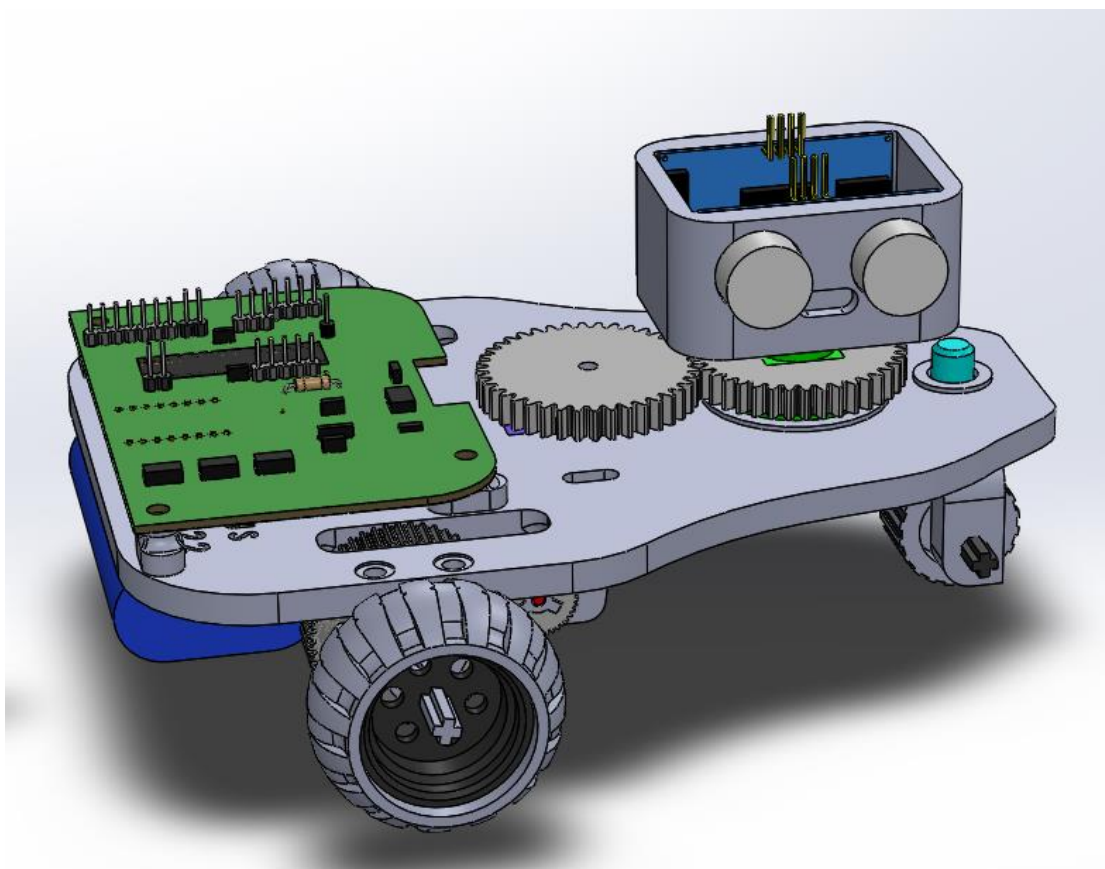
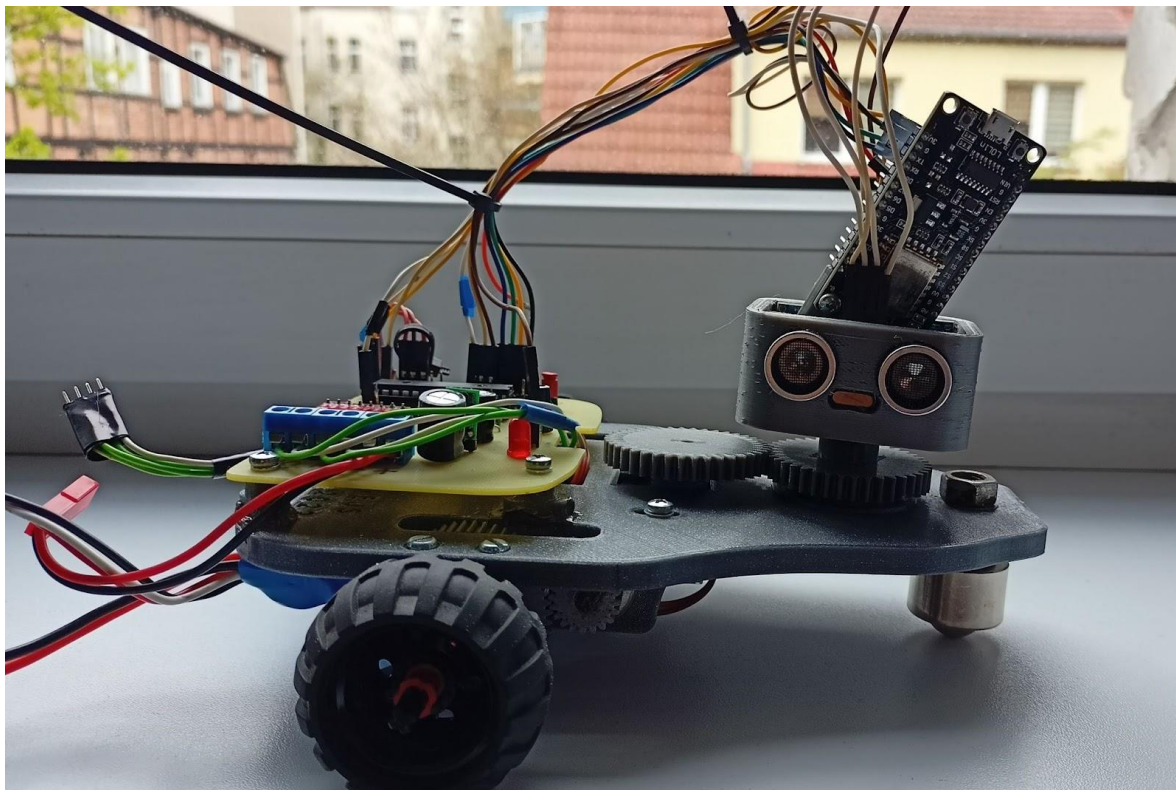


ULTRASONIC MAPPING ROBOT

BY KACPER ŁOŚ 2022



1. Opis i cele projektowe:

Ultrasonic Mapping Robot jest robotem skanującym przestrzeń za pomocą ultradźwiękowych czujników odległości.

Cel projektu:

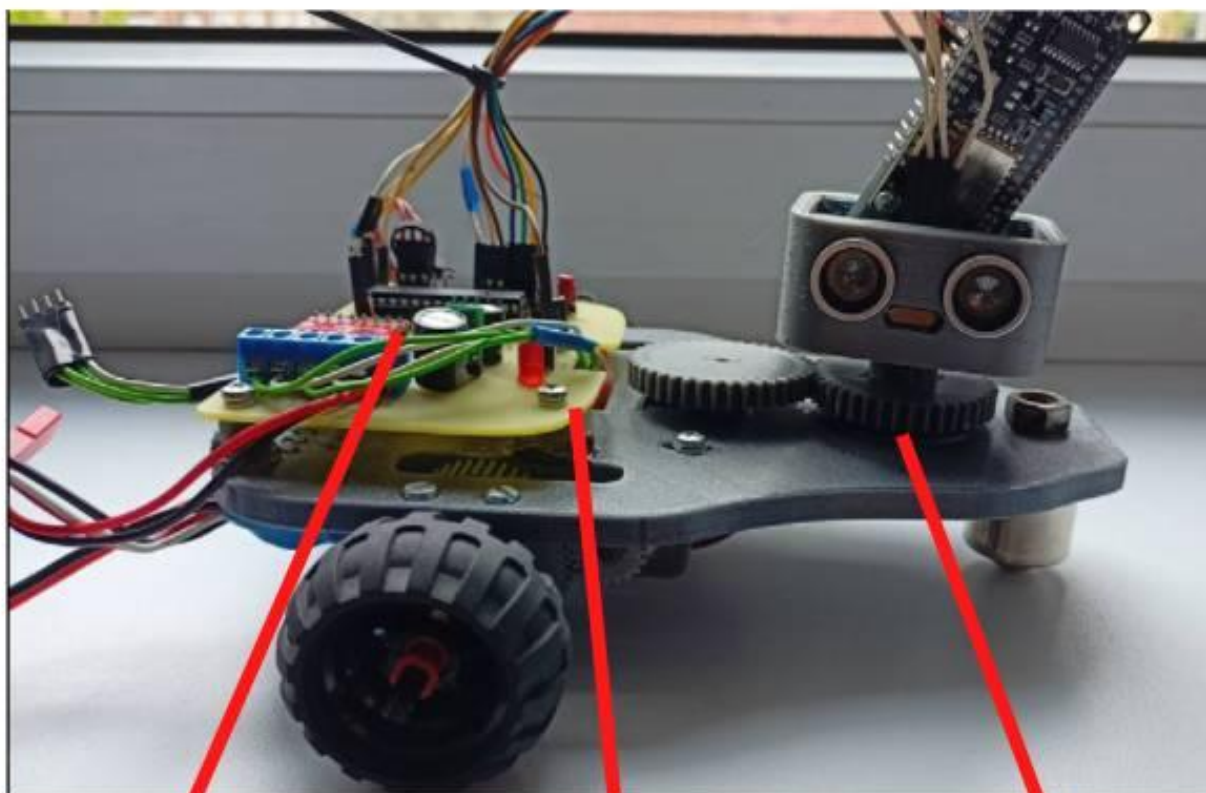
- Stworzenie urządzenia, które będzie mogło skanować przestrzeń, w której się znajduje, a zgromadzone dane przesyłać za pomocą komunikacji WiFi do chmury internetowej w myśl idei IoT (Internet of Things). Dane z chmury mogą być później analizowane i na ich podstawie będzie sporządzana mapa 2D przestrzeni. Do tego robot powinien mieć możliwość zdalnego sterowania, aktualizacji i wgrywania nowego oprogramowania poprzez komunikację WiFi.

Wymagania:

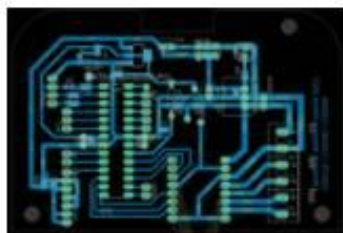
- Autonomiczność - robot powinien samodzielnie wykrywać i omijać przeszkody na swojej drodze, opracowywać i optymalizować trasę wraz z punktami odpowiednimi do procesu skanowania, wysyłać i odbierać dane z/do chmury internetowej,
- Niezawodność – robot musi być przygotowany na różne typy przestrzeni, w której się znajduje, musi niezawodnie wykrywać przeszkody i orientować się w terenie. Układ mechaniczny powinien być skonstruowany w taki sposób, aby robot nie psuł się, a ewentualne usterki mogły być szybko zdiagnozowane i naprawione.
- Łatwość montażu i regulacji – montaż wszystkich potrzebnych elementów powinien być łatwy i umożliwiać szybką wymianę części. Aby zapewnić **niezawodność**, system napędowy powinien posiadać regulację np. ustawienia zębatek, w celu odpowiedniego spasowania i pozbycia się niepotrzebnych luzów.
- Niezawodność zasilania – aby wszelkie pomiary i działanie robota było poprawne, należy skonstruować wielostopniowy układ zasilający, który będzie w stanie zasilать urządzenia o bardzo dużym poborze prądu takie jak silniki oraz serwomechanizm, ale zapewni czyste, niezaszumione zasilanie czujnikom, a przede wszystkim mikrokontrolerowi.

Aby całe urządzenie sprostało wymaganiom i osiągało postawione cele, całość prac nad robotem została podzielona na 3 równoległe rozwijane etapy:

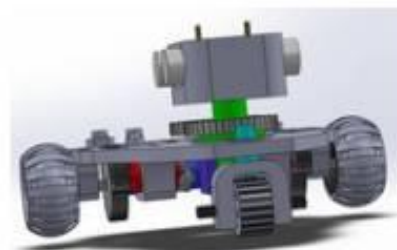
- MECHANIKA
- ELEKTRONIKA
- OPROGRAMOWANIE



Oprogramowanie



Elektronika



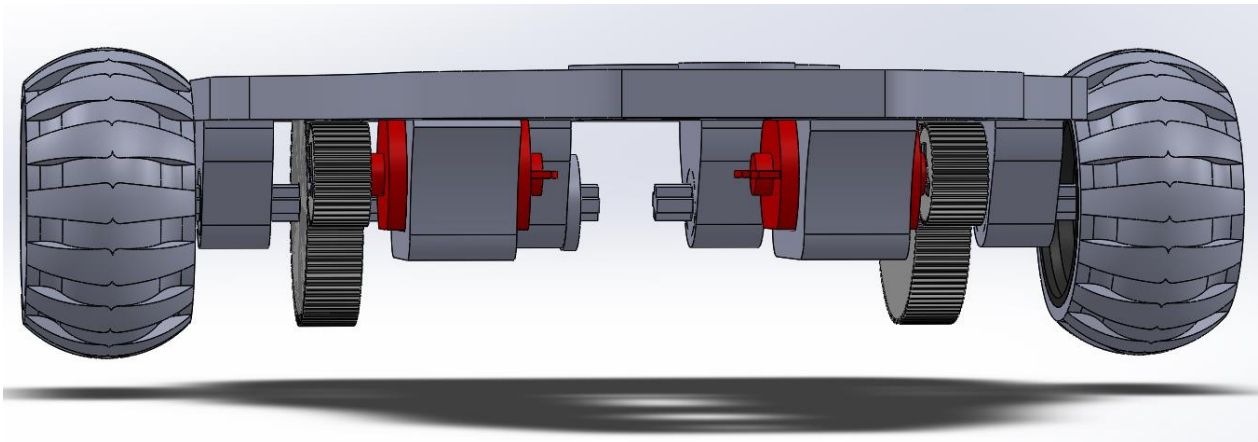
Mechanika

2. Mechanika:

Część mechaniczna robota składa się z kilku głównych modułów:

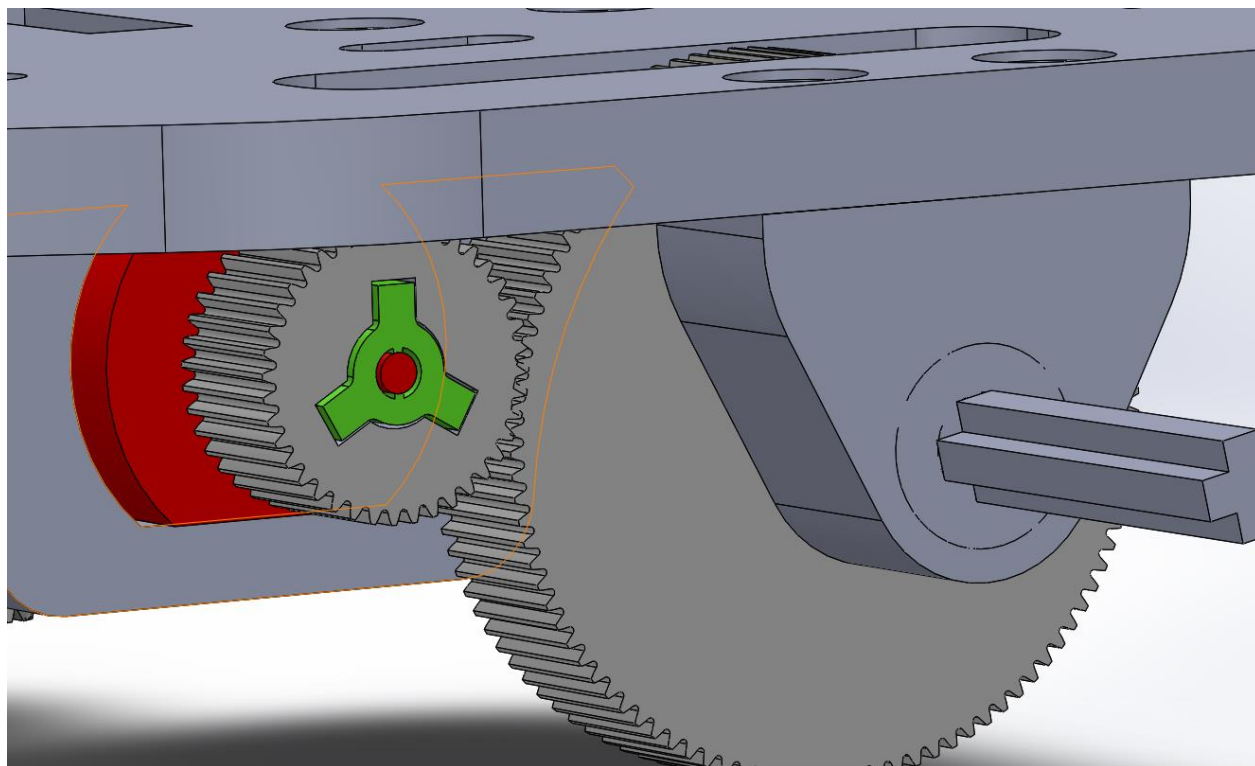
- Moduł napędowy złożony z dwóch silników DC przekładni zębatej 1:2.5 oraz specjalnych tulei i wałów łączących koła z silnikami,
- Moduł wieżyczki skanującej złożony z serwomechanizmu, który za pomocą przekładni zębatej 1:1 obraca specjalnym trzpieniem osadzonym na łożysku kulkowym. Na trzpieniu osadzona jest wieżyczka, w której zamontowane są czujniki odległości,
- Modułu ramy czyli głównej podstawy robota, ze specjalnie wyciętymi otworami i miejscami do mocowania łożyska, silników i innych elementów.

2.1 Moduł napędowy:

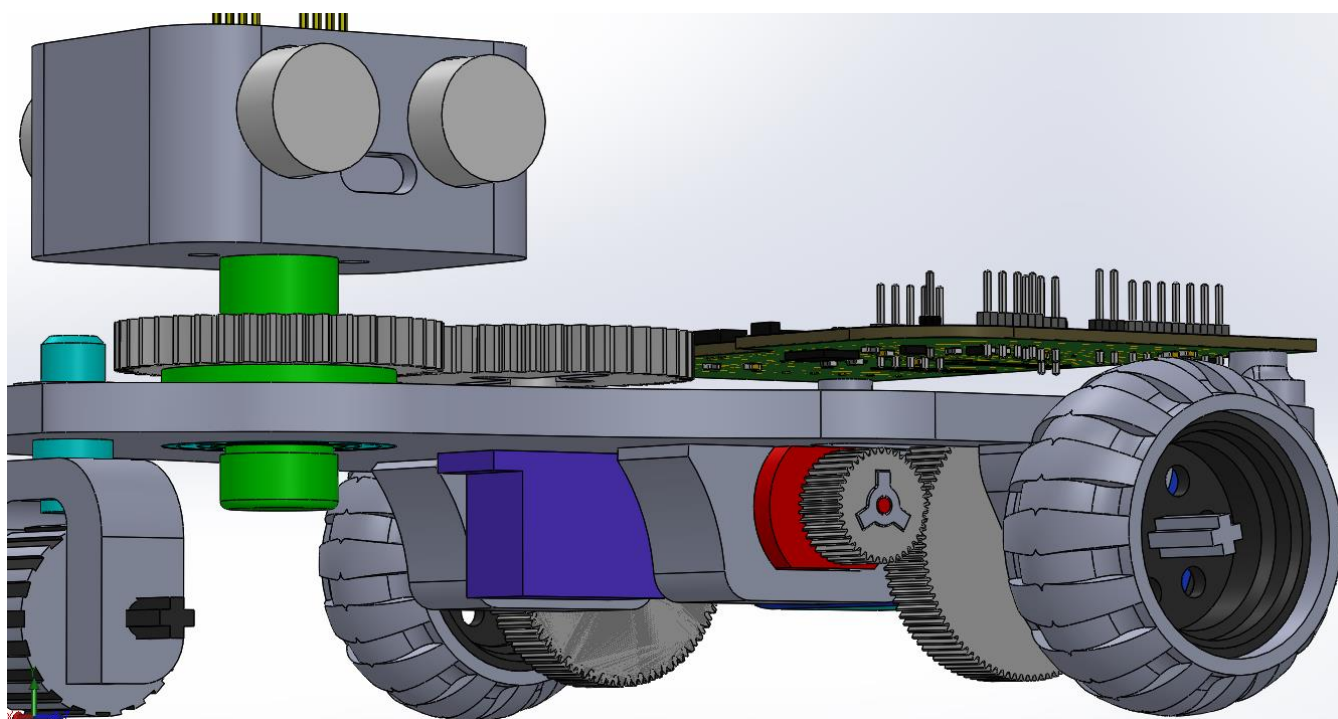


Aby wykorzystać jak najwięcej elementów „z odzysku”, w celu minimalizacji kosztów, postanowiono wykorzystać silniki DC, które wylutowano ze starych zabawek. Do tego koła i krzyżakowe wały wykorzystano z zestawów LEGO. Zębatki, tuleje i chwytaki wydrukowano na drukarce 3D z wykorzystaniem materiału PETG.

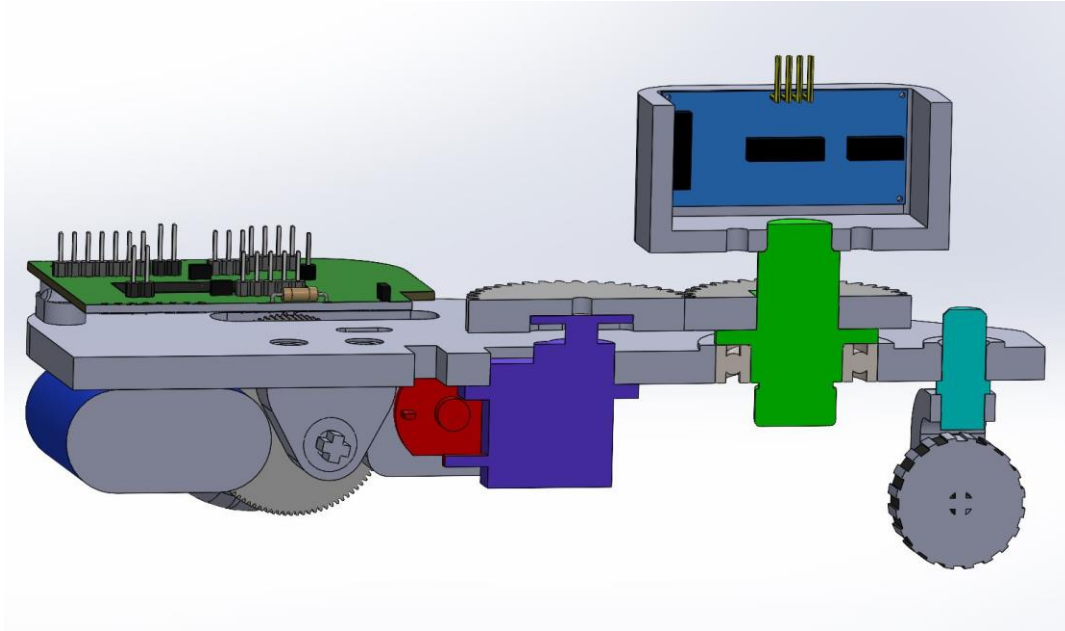
Na wał silnika wciśnięto specjalny element (zielony), który z kolei został wklejony za pomocą dwu-składnikowego kleju na małą zębatkę.



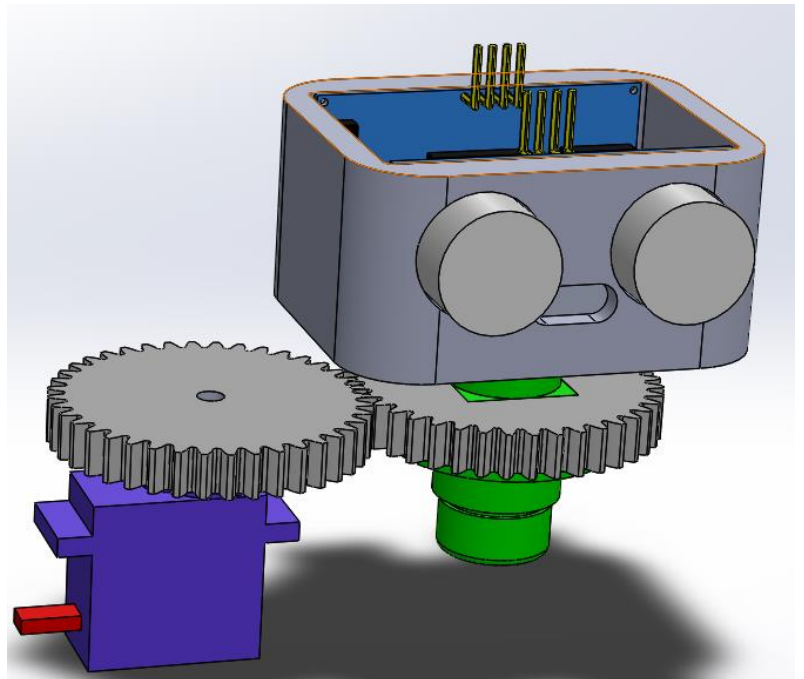
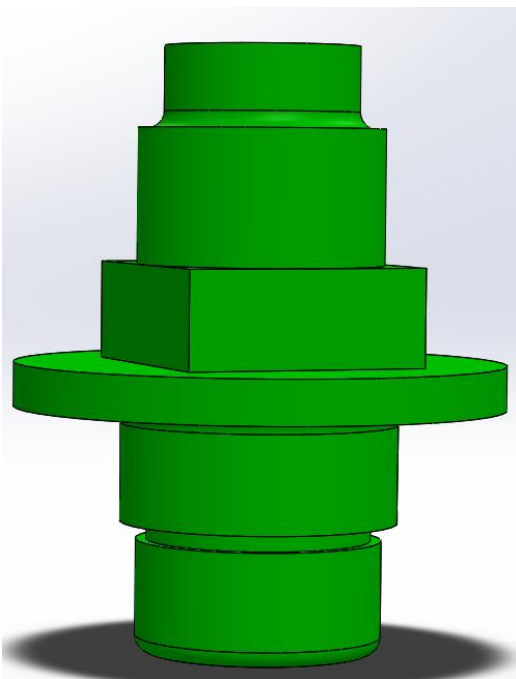
Zdecydowano się na takie rozwiązanie, ponieważ przyklejając dużą zębatkę bezpośrednio na wał, utracono by możliwość wymiany zębatki na nową.



2.2 Moduł wieżyczki skanującej:

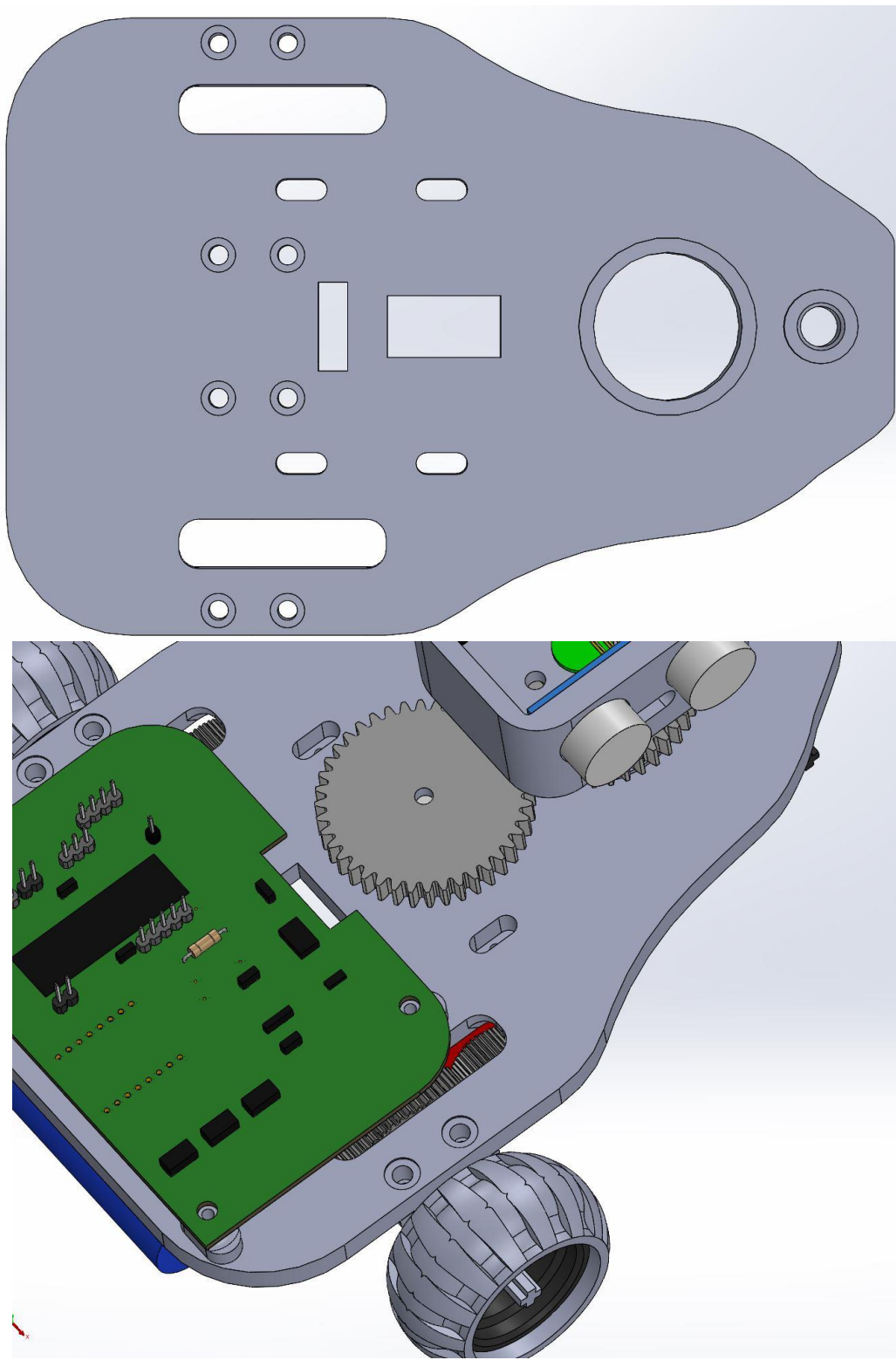


Aby móc skanować przestrzeń za pomocą dwóch czujników odległości, zaprojektowano specjalną wieżyczkę, która może wykonywać obrót o 180 stopni. Większy obrót spowodowałby wiele problemów m.in. z doprowadzeniem kabli. Wieżyczka jest osadzona na wielostopniowym trzpieniu (zielony), który z kolei obraca się w łożysku kulkowym, które ogranicza tarcie do minimum i pozwala zastosować dużo słabszy napęd. Na trzpieniu osadzona jest również zębatka, która obracana jest przez taką samą zębatkę zamocowaną na serwomechanizmie.

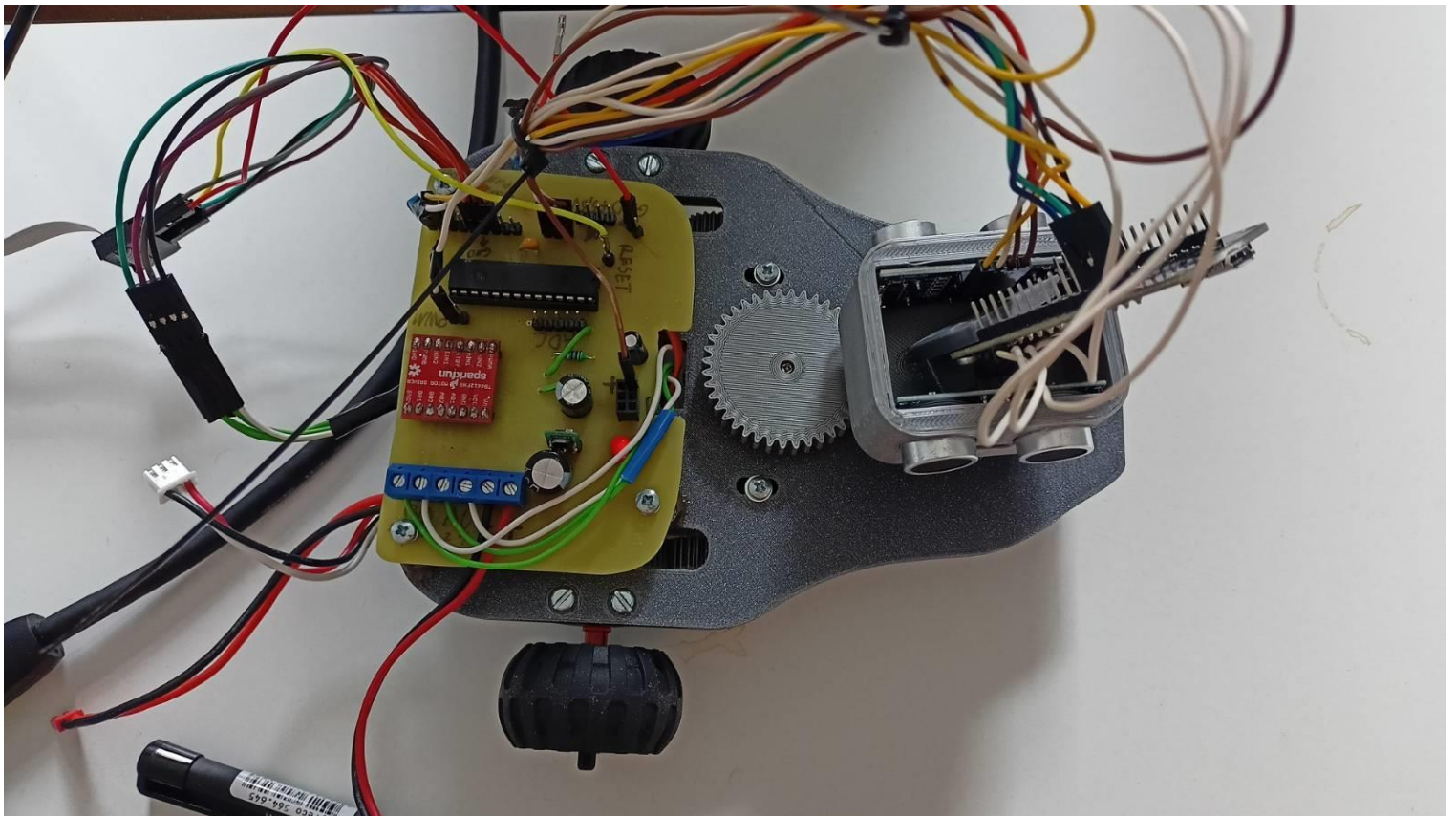
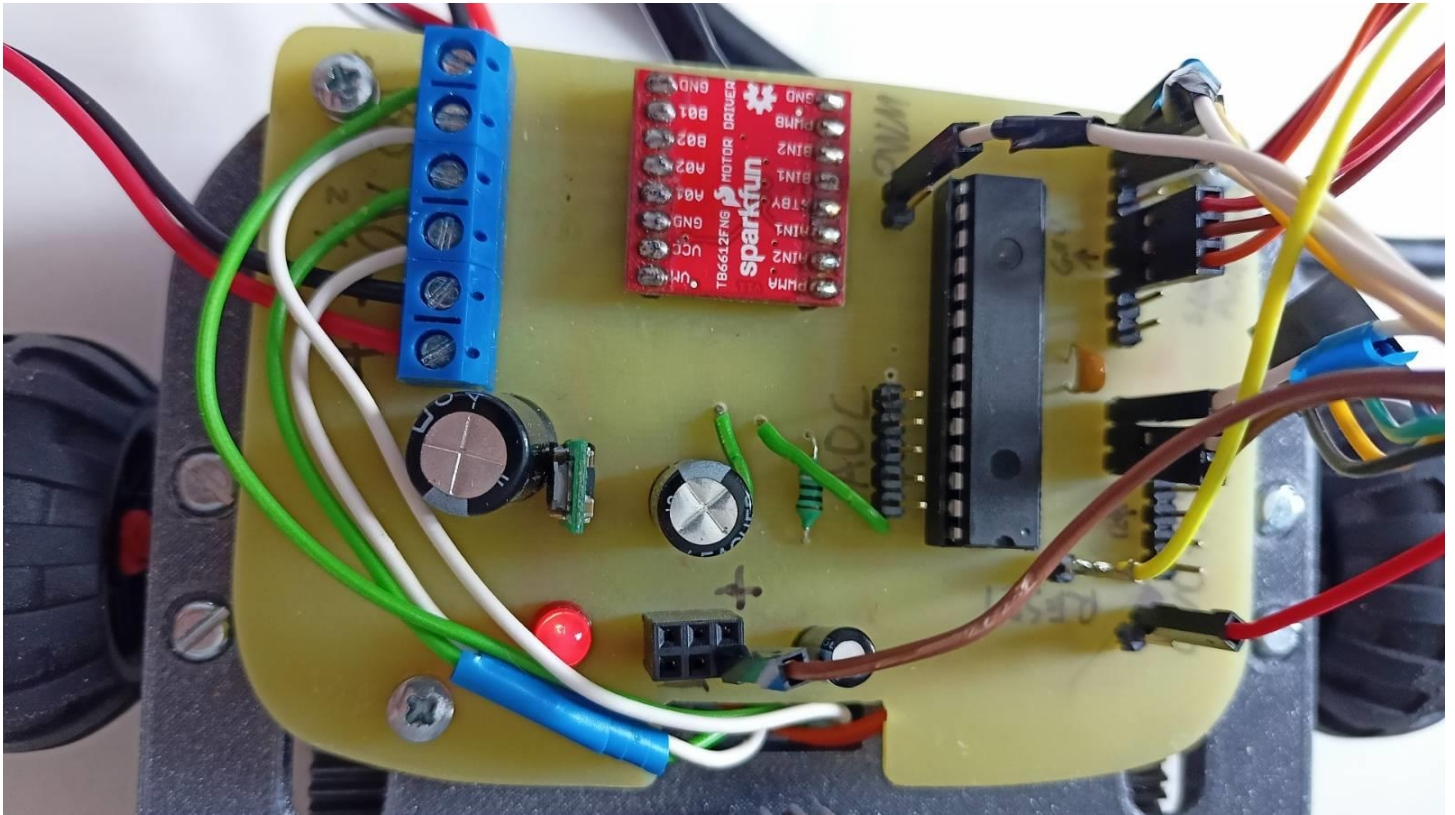


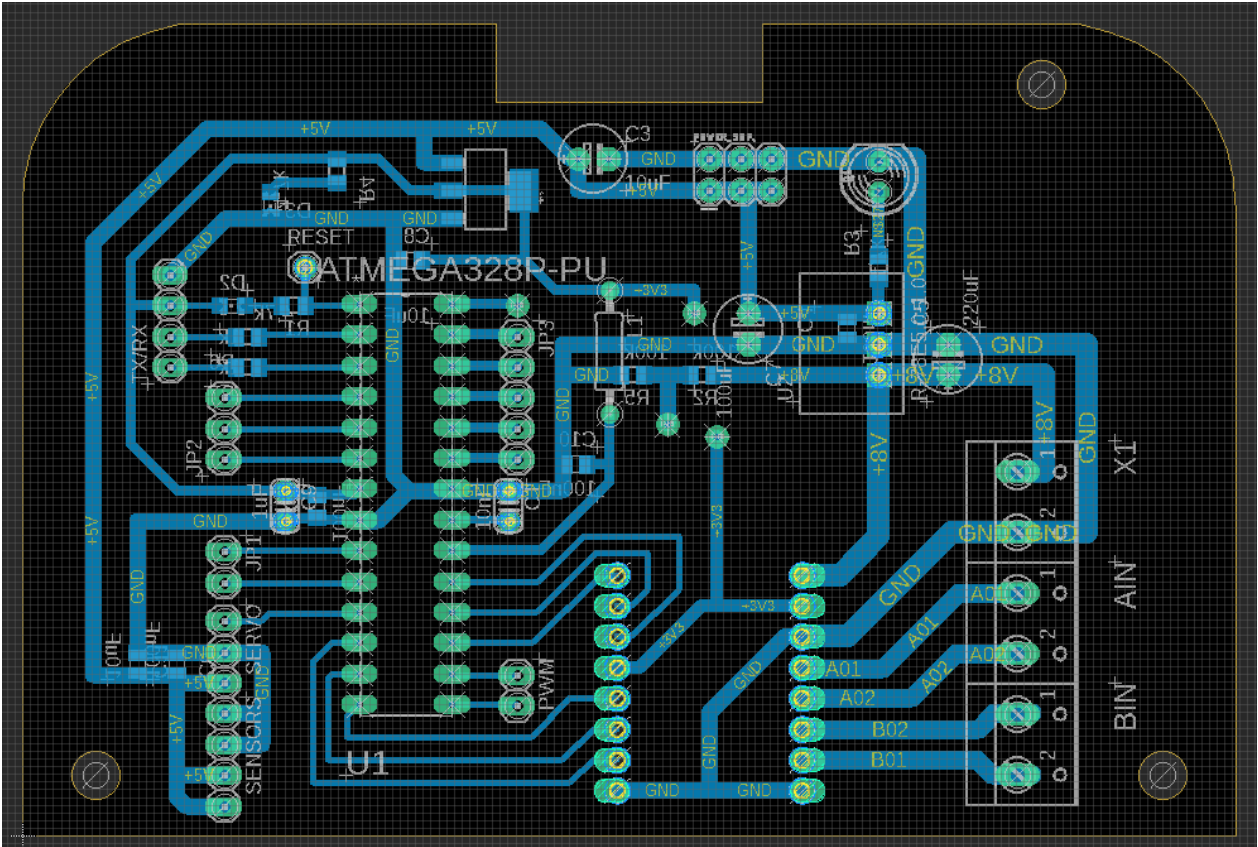
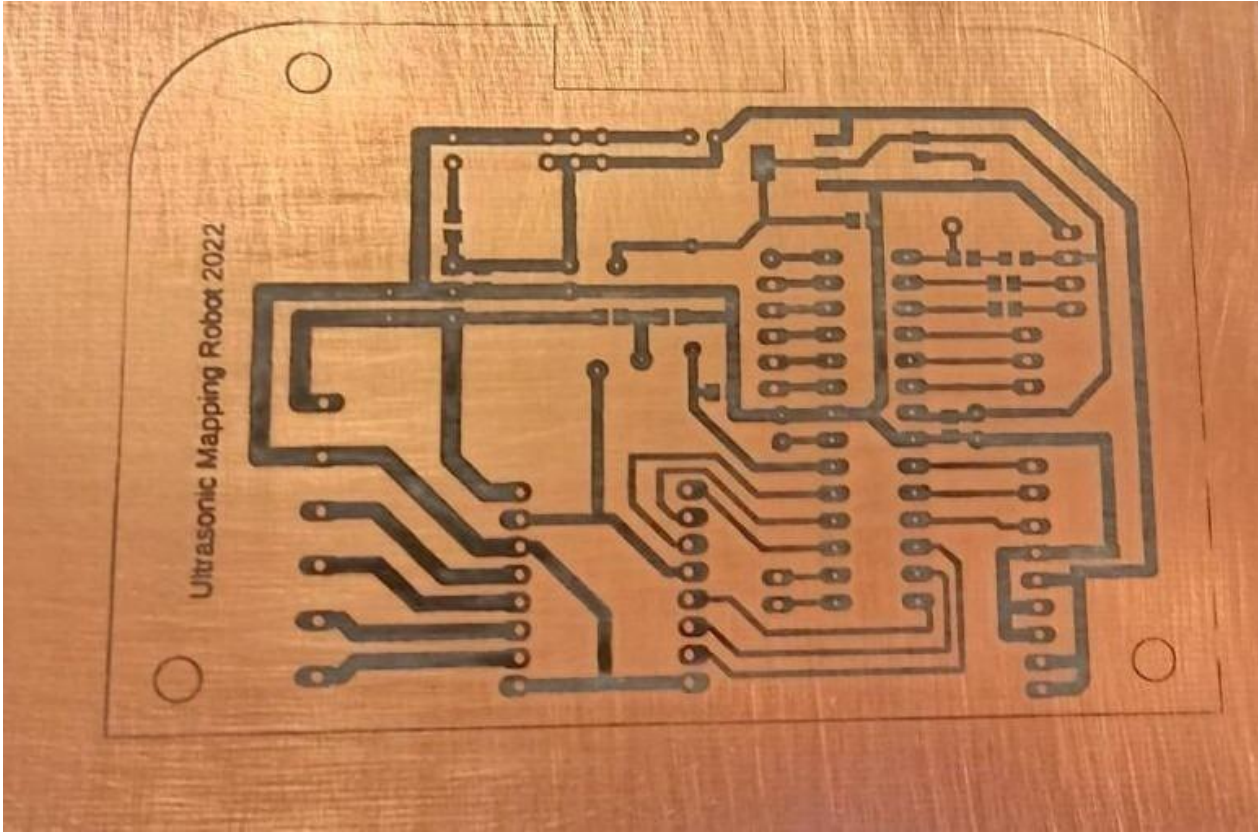
2.3 Podstawa główna:

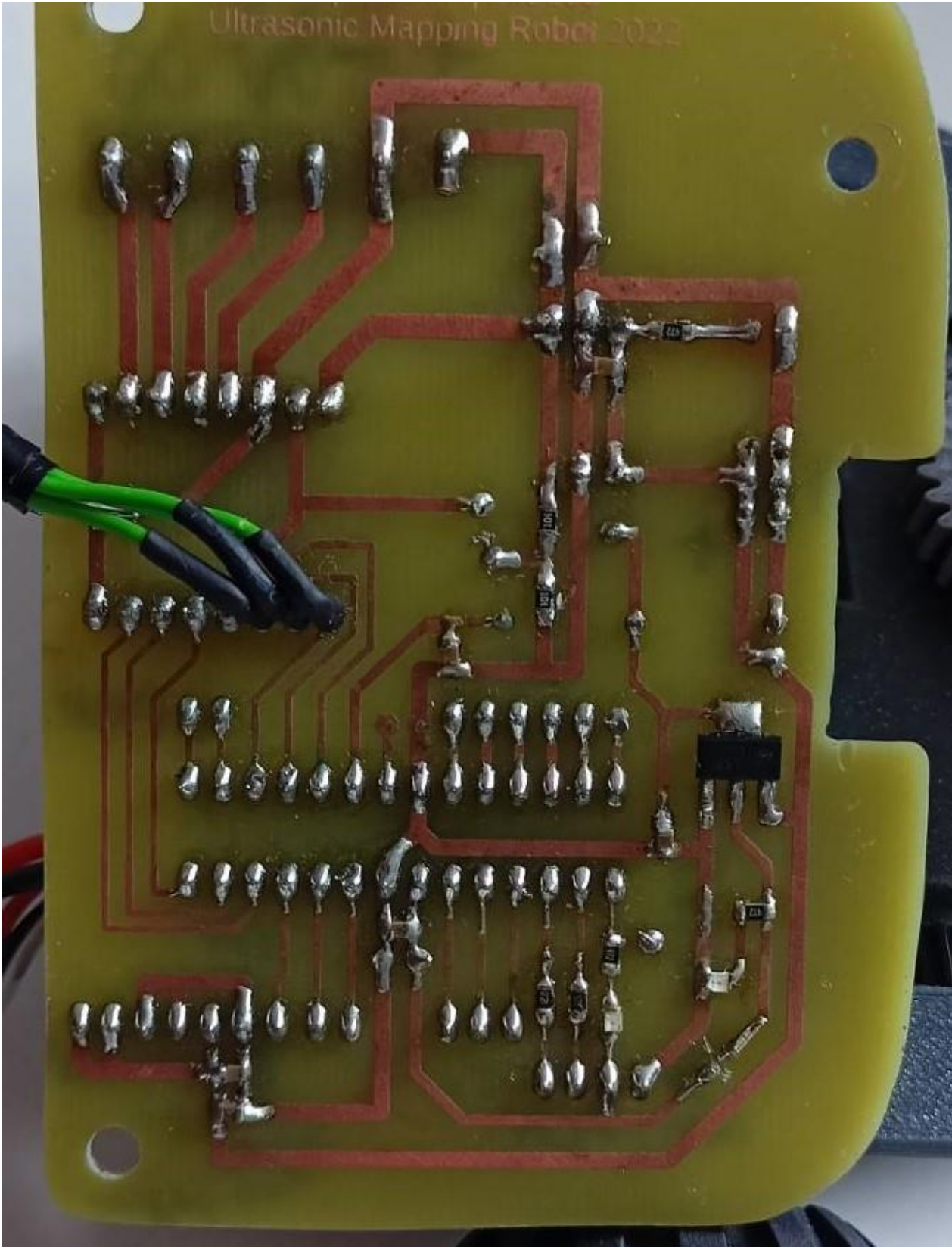
Dzięki specjalnie umiejscowionym otworom i wycięciom, a także dobranym tolerancjom i pasowaniu, silniki, zębatki i ogólnie elementy napędowe mogą być łatwo regulowane, dopasowywane pod większe lub mniejsze np. zębatki w celu zwiększenia momentu napędowego lub zwiększenia prędkości obrotowej.



3. Elektronika:







Układ elektroniczny został podzielony na 4 części:

- Sekcja zasilania 8V - zasilanie silników
- Sekcja zasilania 5V – zasilanie czujników, oraz obniżenie napięcia dla stabilizatora 3.3V
- Sekcja zasilania 3.3V – zasilanie mikrokontrolera ATMEGA 328 oraz modułu ESP8826
- Sekcja sygnałowa – połączenie wszystkich wyprowadzeń i urządzeń peryferyjnych potrzebnych do prawidłowego działania robota.

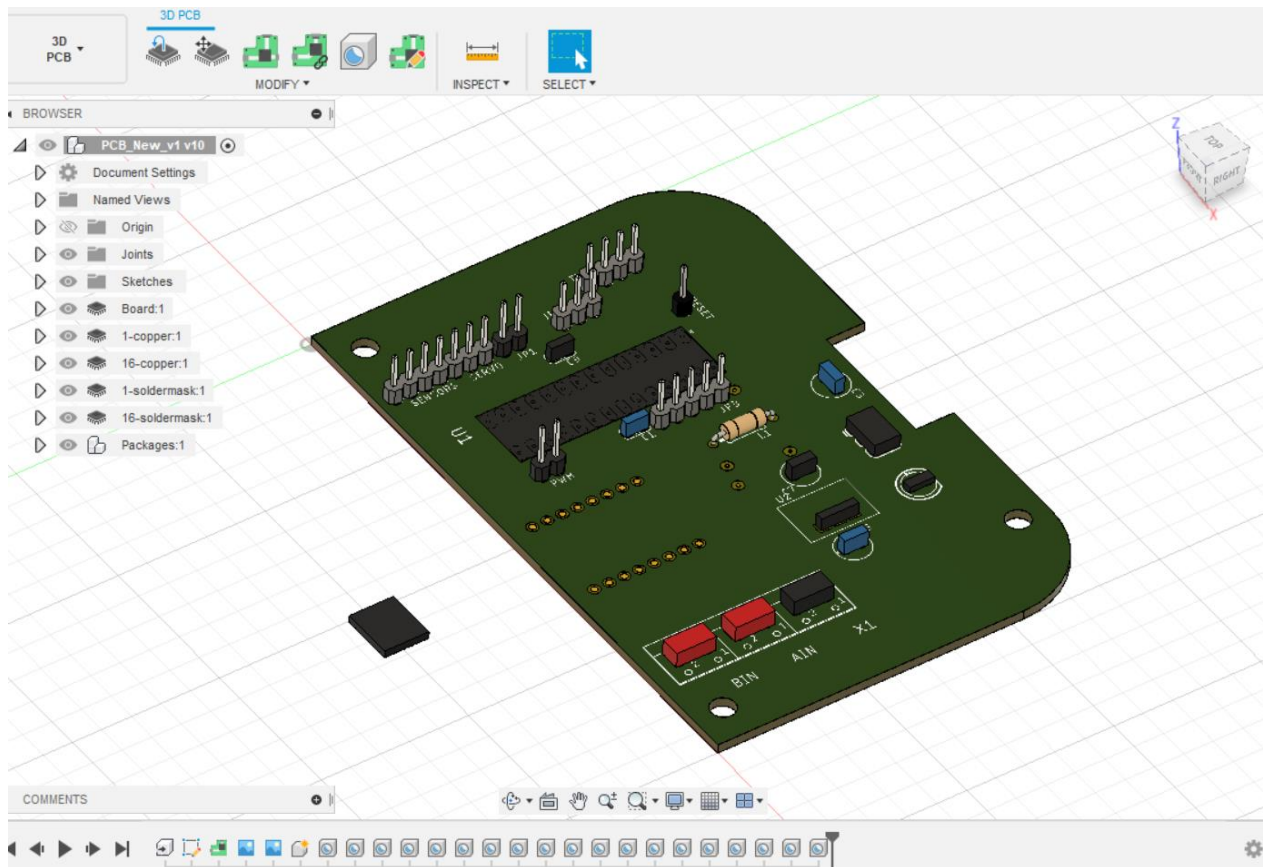
Układ składa się z takich elementów jak:

1. Mikrokontroler AVR - Atmega 328
2. Dwukanałowy sterownik silników SparkFun TB6612FNG
3. Przetwornica Step Up – Down 5V
4. Stabilizator LDO 3,3V LM1117MP – SMD SOT223
5. Rezystory, kondensatory, dławiki, i inne elementy pasywne w obudowie THT oraz SMT
6. Diody LED
7. Złącza ARK oraz goldpin

Do układu podłączone są następujące urządzenia:

1. Silnik DC – 2szt.
2. Serwomechanizm TowerPro SG-90 micro 180 stopni – 1szt.
3. Ultradźwiękowy czujnik odległości HC – SR04 – 2szt.
4. Akumulator LiPo 7.4V 2S 18650 – 1szt.
5. Moduł WiFi ESP8266 + NodeMCU v3 – 1szt.

Schemat elektroniczny oraz projekt płytki PCB został wykonany w programie Autodesk Eagle. Zdecydowano się na ten właśnie program ze względu na jego synchronizację z programem Autodesk Fusion 360. W Fusion 360 można dobrać odpowiedni kształt płytki idealnie pasujący do podstawy i obudowy głównej robota. Dzięki synchronizacji w chmurze internetowej można importować model 3D i kształt płytki PCB do programu Eagle, gdzie nanoszone są wszystkie ścieżki i footprinty, po czym gotowy układ eksportuje się znów do Fusion 360, aby stworzyć wirtualny model 3D układu.



Lutowanie elementów i wiercenie przeprowadzono ręcznie za pomocą stacji lutowniczej z możliwością regulacji temperatury, wiertarki i wiertel (0.6, 0.7, 1mm) oraz tzw. trzeciej ręki pomocnej w utrzymaniu płytki i układów w miejscu lutowania. Wykorzystano cynę bezołowiową, o różnej grubości. Dodatkowo wykorzystano kalafonię i specjalną pastę ułatwiającą lutowanie np. przewodów.

3.1 Proces trawienia płytki PCB:

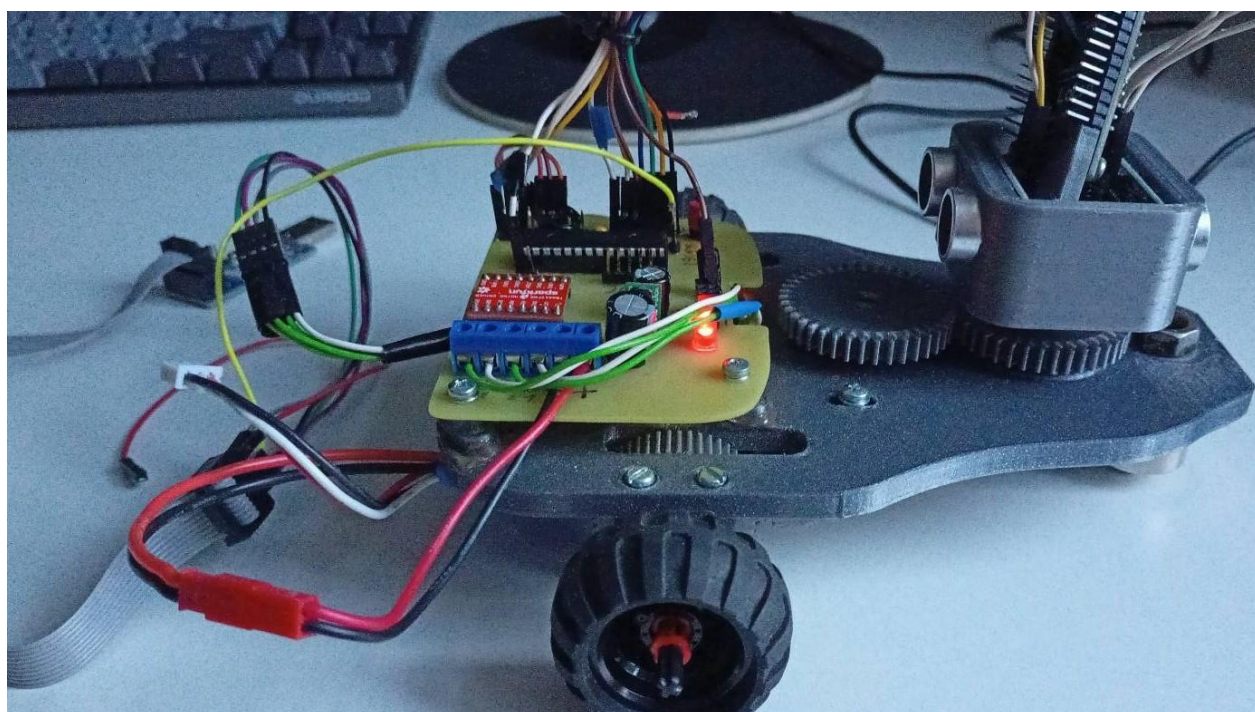
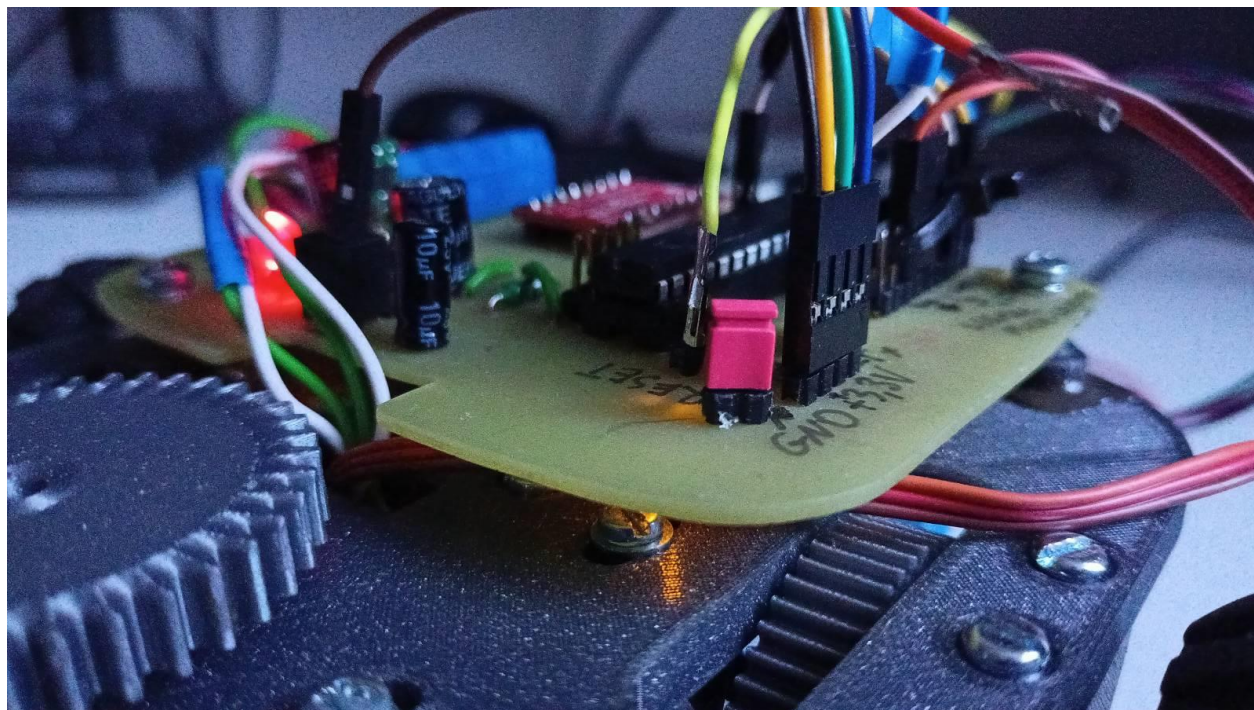
Na początku została przetestowana metoda z wykorzystaniem zmywacza do paznokci, jednak nie udało się osiągnąć nią zadowalających rezultatów. Cztery próby kończyły się brakiem kilku ścieżek na docelowej płytce, mimo wykorzystania różnego rodzaju papieru w tym kredowego oraz specjalnego papieru do drukowania zdjęć.

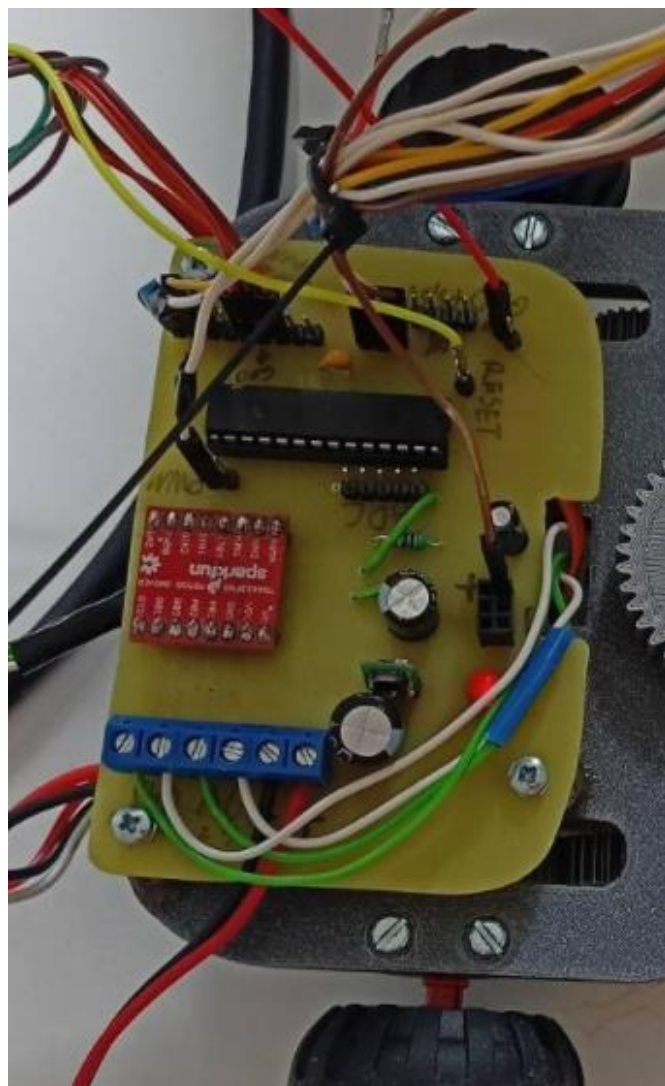
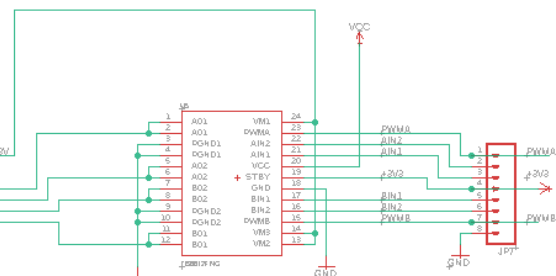
Ostatecznie płytka została wytrawiona metodą termotransferu tzw. metodą żelazkową.

- Projekt układu został wydrukowany na papierze kredowym.
- Przed transferem płytka została wytarta papierem ściernym P1200 w celu pozbycia się warstwy tlenków obecnej na wierzchniej warstwie miedzi. Tlenki te wpływają bardzo negatywnie na transfer tuszu z papieru na płytkę, sam proces trawienia (wytrawiacz nie może działać bezpośrednio na miedź), a także utrudnia proces lutowania.
- Następnie, całą powierzchnię wyczyszczono alkoholem izopropylowym (IPA).
- Wydruk został naklejony na płytkę oraz starannie wprasowany przez około 8-10 minut.
- Następnie, całość została umieszczona w wodzie na czas 10 minut, po którym naklejony papier z łatwością odchodził zostawiając przetransferowane ścieżki.
- Po udanym transferze wydruku, płytka została umieszczona w wytrawiaczu na około 2 godziny. Przy poprzednich projektach płytek, proces trawienia trwał ok 40 min, jednak w tym wypadku woda zamiast 40C miała tylko 20C. Temperatura wody znacząco wpływa na czas reakcji wytrawiacza z miedzią. Aby przyspieszyć proces trawienia, roztwór był regularnie mieszany.
- Na koniec płytka została oczyszczona i wycięta do docelowego kształtu.

Wykorzystano laminat jednostronny, który pozwala na wykonywanie tylko układów jednowarstwowych. Mimo dużej liczby połączeń (konieczność podłączenia mikrokontrolera ATMEGA 328) udało się tak zoptymalizować rozmieszczenie elementów i ścieżek na płytce, że ograniczono użycie tzw. mostków do dwóch połączeń.

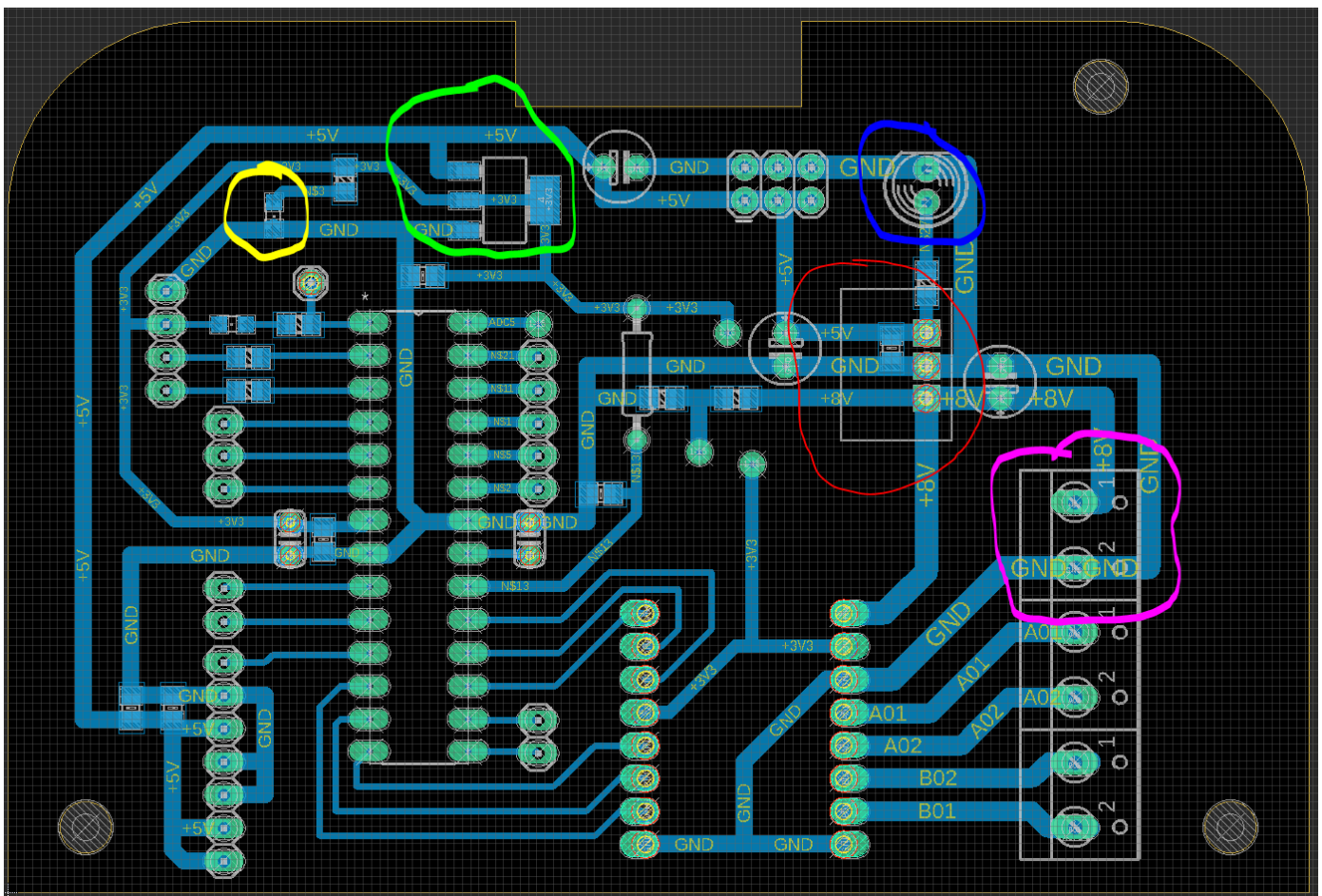
W celu zminimalizowania ilości otworów do przewiercenia (elementy THT czyli w obudowie przewlekanej takie otwory wymagają), zdecydowano się na zastosowanie jak największej ilości elementów montowanych powierzchniowo tzw. SMT.





3.1. Sekcja zasilania 8V:

Silniki są bezpośrednio zasilane z akumulatora, który podpięty jest do złącza ARK (fioletowy). Dzięki temu uzyskujemy wysokie napięcie 8V, co za tym idzie moment obrotowy. Jednak najważniejszy powód dla którego silniki nie są zasilane przez przetwornicę albo stabilizator, a bezpośrednio z akumulatora jest związany z prądami wstecznymi. W momencie gdy rozpędzony silnik elektryczny zostaje zmuszony do przerywania pracy, a robot chcąc się zatrzymać i przestanie dostarczać prąd do silników, w uzwojeniach silnika wygeneruje się napięcie (zjawisko nazywane regenerative braking), które „cofając się” zniszczyło by stabilizator/przetwornicę. Jednak dzięki zasilaniu bezpośredniemu, możemy ten problem rozwiązać, a dodatkowo, energię wygenerowaną przez silnik możemy odzyskać w akumulatorze.



3.2 Sekcja zasilania 5V:

Do zasilania czujników odległości wykorzystywana jest przetwornica impulsowa (czerwony) dająca napięcie na wyjściu ok 5V. Aby napięcie to miało jak najmniejsze tętnienie oraz by odszumić zasilanie, zastosowano filtr złożony z kondensatora elektrolitycznego oraz ceramicznego połączonych równolegle w parę.

Przetwornica ta pełni również drugą, niezwykle istotną rolę z punktu widzenia odprowadzania ciepła i „przygotowywania” niskiego napięcia dla stabilizatora 3.3V znajdującego się dalej w obwodzie.

3.3. Sekcja 3.3V:

Stabilne napięcie 3.3V zostało zapewnione przez stabilizator LDO 3,3V LM1117MP SMD (zielony). W związku z tym, że jest to stabilizator SMD, o niskim napięciu LDO (low drop-out) i o stosunkowo małej powierzchni, jego zdolność do odprowadzania ciepła (generowanego na skutek obniżania napięcia z wyższego do stabilnego 3.3V) jest niewielka. W specyfikacji można znaleźć odpowiednie parametry opisujące ilość ciepła, którą może odprowadzić taki stabilizator do otoczenia. Niestety gdyby stabilizator LM1117MP miał za zadanie uzyskać napięcie 3.3V z napięcia na wejściu równemu ponad 8V, cały układ praktycznie by się spalił. Dlatego tak ważne jest zastosowanie **przetwornicy impulsowej 5V** z poprzedniej sekcji.

Stabilizator LDO 3.3V, a także specjalne filtry LC oraz pary kondensatorów, są kluczowe do stabilnego, czystego i niewrażliwego na przeciążenia w całym obwodzie napięcia zasilania, które doprowadzone jest do mikrokontrolera Atmega.

Zasilanie mikrokontrolera zostało rozbite na dwa układy tzn. poprzez parę kondensatorów do Vcc Atmegi, oraz poprzez filtr LC do Vref czyli napięcia referencyjnego. Zasilanie referencyjne, które jest pozbawione szumów i jest stabilne, pozytywnie wpływa na odczyty prowadzone za pomocą wbudowanego przetwornika analogowo-cyfrowego ADC.

3.4. Sekcja sygnałowa:

Oprócz ścieżek łączących wyprowadzenia goldpin / ARK, do których podłączone są różne urządzenia, w tej sekcji warto wspomnieć o dzielniku napięcia oraz diodach sygnalizacyjnych.

Dzielnik napięcia zbudowany jest z dwóch rezystorów 101kOhm w obudowie SMD. Niestety nie było dostępnych rezystorów o wysokiej dokładności np. 0.5% tolerancji, które byłyby idealne na dzielnik napięcia ze względu na ich dokładność wykonania, a także niską podatność na tzw. efekt płynięcia rezystorów związany z ich temperaturą. Dzielnik ten dzieli napięcie na akumulatorze na pół. Jest to konieczne, ponieważ akumulator ma średnie napięcie ok 8V, a do pinów analogowych Atmegi można doprowadzić maks. 5V.

Aktualizacja

Po wczytaniu się w notę katalogową Atmegi, a konkretnie przetwornika ADC i napięcia REF okazało się, że przy zasilaniu atmegi 3,3V maksymalne napięcie odczytywane za pomocą przetwornika nie może wynosić 5V. Dlatego zastosowano dzielnik napięcia z rezystorów 101kOhm i 68kOhm. Spodziewając się maksymalnie 8,4V na baterii, daje to nam maks. napięcie na przetwornik równe 3.38 V.

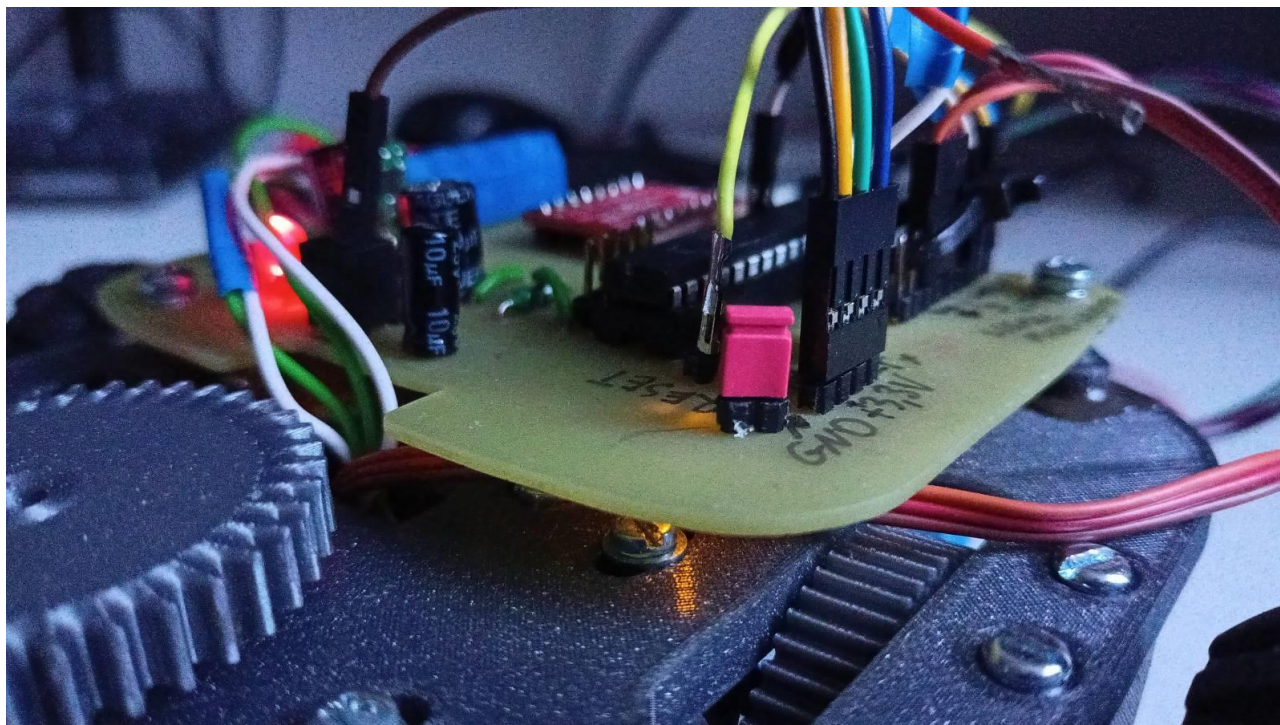
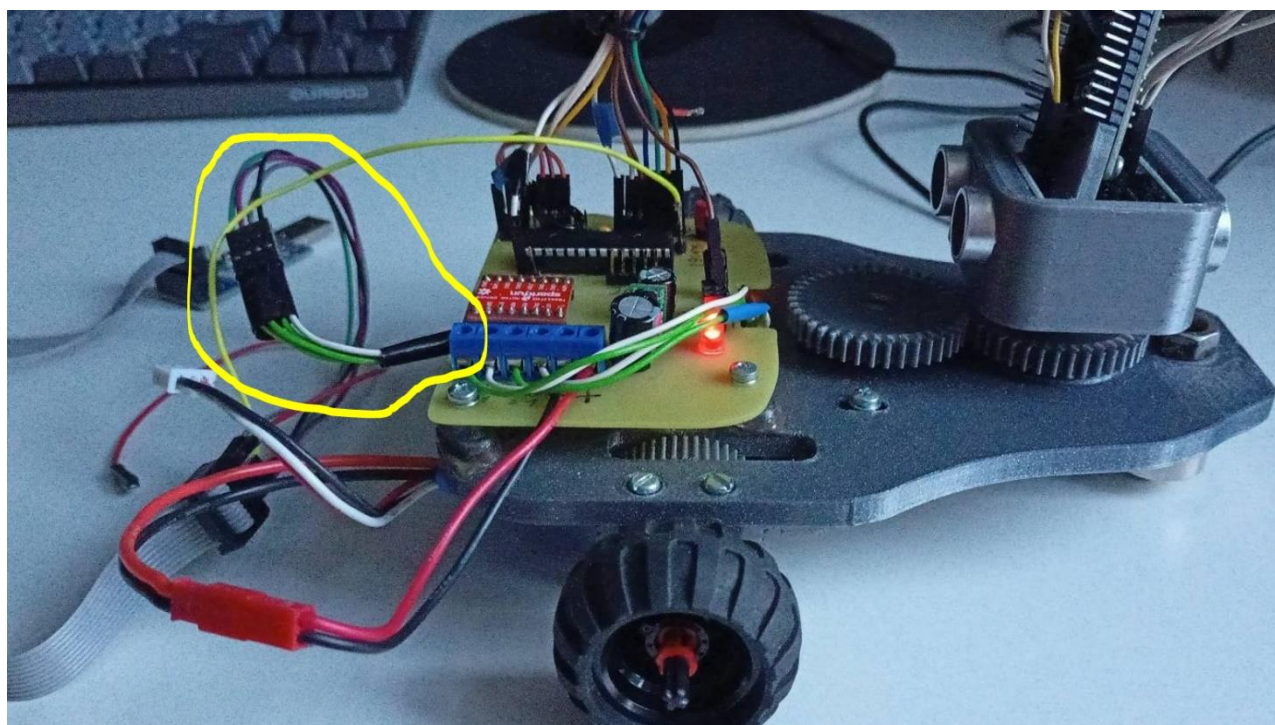
Diody LED sygnalizujące są niezwykle pomocne podczas testowania i diagnozy ewentualnych uszkodzeń i nieprawidłowości w układzie. **Dioda Led SMD** (żółta) sygnalizuje napięcie zasilania 3.3V, którym zasilany jest mikrokontroler. **Dioda Led THT** (niebieska) sygnalizuje napięcie zasilania 5V wychodzące z przetwornicy. Dzięki temu po podłączeniu zasilania z akumulatora, od razu widać czy na płytce prawidłowo działa przetwornica oraz stabilizator i że urządzenia są podpięte do odpowiedniego zasilania.

Szerokości ścieżek na płytce zostały dobrane w zależności od spodziewanego prądu. Do tego ścieżki masy zostały pogrubione, aby zapewnić dużą powierzchnię.

Dodatkowo na płytce wyprowadzono napięcie +5V oraz GND za pomocą portu, do którego można podłączać typowe dla np. arduino kabelki. Dzięki temu do płytki można podłączać zewnętrzne urządzenia na np. płytce stykowej aby testować nowe funkcjonalności.

Aby umożliwić przewodowe programowanie mikrokontrolera, zbieranie danych, a także bezpośrednie diagnozowanie ewentualnych problemów, wyprowadzono specjalną złączkę (żółty). Dzięki temu można w łatwy sposób wpiąć programator ISP do pinów ATmegi (miso, mosi, sck oraz reset). Aby korzystać z tej funkcjonalności należy odłączyć mikrokontroler od zasilania na płytce za pomocą zworki (czerwona).

Po wyciągnięciu zworki, na dolny goldpin należy podać zasilanie 3.3V np. z programatora.



4. Oprogramowanie:

Początkowo oprogramowanie sterujące zostało napisane w środowisku Arduino, z pomocą pakietu MiniCore a następnie bezpośrednio skompilowane pod mikrokontroler Atmega328p. Było to wygodne, szybkie oraz odporne na błędy rozwiązanie, które pozwoliło upewnić się, że cała część elektroniczna oraz mechaniczna działa poprawnie po złożeniu wszystkiego razem. Zmiana metodyki programowania z Arduino na AVR pokazała jak wiele rzeczy działa „od środka” w Arduino, oraz uwydatniła elastyczność konfiguracji w przypadku programowania AVR.

Przez ograniczenia czasowe, oraz czasochłonność procesu przepisywania programu od nowa, z pomocą AVR zostały oprogramowane najważniejsze elementy kluczowe do sterowania robotem (serwomechanizm, sensor ultradźwiękowy, silniki, diody oraz ADC). Uzyskany w ten sposób program wykorzystuje inne piny niż założone w początkowym projekcie, natomiast w pełni poprawnie obsługuje wszystkie potrzebne moduły sterujące.

4.1 Główny Program

Główny program składa się jedynie z dwóch komponentów – **inicjalizacji** modułów i **sterowania** robotem. Poniżej została przedstawiona struktura całego programu:

```
1  [...]
6
7  #define F_CPU 8000000UL
8  #include <avr/io.h>
9  #include <util/delay.h>
10 #include <avr/interrupt.h>
11
12 // ----- zmienne HC-SR04 -----
13 uint16_t timer0_overflows = 0;
14 uint8_t echo_finished = 0;
15
16 // ----- Obsługa Przerwań -----
17 ISR(TIMERO_OVF_vect) [...]
23 ISR(PCINT2_vect) [...]
55
56 void init();
57 void control();
58
59 int main() {
60     init();
61     control();
62 }
63
64 // ----- Funkcje inicjalizujące ----
```

```
63
64 // ----- Funkcje inicjalizujące ----
65 void init_timer1() [...]
85 void init_timer2() [...]
101
102 void init_info_diode() [...]
115 void init_servo() [...]
125 void init_engines() [...]
142 void init_sensor() [...]
178 void init_adc() [...]
212
213 void init() [...]
227
228 // ----- Funkcje Sterujące -----
229 void set_info_diode(float duty_fraction) [...]
240 uint16_t make_scan() [...]
288 void drive_forward() [...]
320 void update_adc() [...]
353 void rotate_right_servo(uint16_t duty_low, uint16_t duty_high) [...]
374 void undo_rotate_right_servo(uint16_t duty_low, uint16_t duty_high) [...]
395
396 void control() [...]
```

Funkcja inicjalizująca **init** konfiguruje wszystkie potrzebne moduły. Następnie przekazuje wykonanie do funkcji sterującej **control**. Inicjalizacja każdego modułu ma swoją osobną funkcję. Dokładniejszy opis każdej z nich znajduje się w rozdziale dotyczącym danego modułu.

```
213 void init() {
214     /* Funkcja odpowiedzialna za inicjalizację wszystkich modułów */
215
216     // Zainicjalizuj liczniki 1 i 2
217     init_timer1();
218     init_timer2();
219
220     // Zainicjalizuj poszczególne moduły
221     init_info_diode();
222     init_servo();
223     init_engines();
224     init_sensor();
225     init_adc();
226 }
```

4.2 Algorytm Sterowania

W celu przetestowania wszystkich potrzebnych modułów, zaprojektowany został prosty algorytm sterujący. Mała złożoność pod względem zaawansowania algorytmu, pozwoliła skupić się na naprawie błędów w poszczególnych komponentach, bez potrzeby analizowania zachowania samego robota. Natomiast, poprzez skalowalność, korzystając z oprogramowanych modułów, można bardzo łatwo uzyskać bardziej zaawansowany algorytm sterowania.

```
396 void control() {
397     /* Funkcja odpowiedzialna za sterowanie robotem */
398     double delay_time = 10;
399
400     while (1) {
401         // zapal diodę informacyjną: 95% światła
402         set_info_diode(.95f);
403         _delay_ms(delay_time);
404
405         // Wykonaj obrót 90 stopni w prawo -> (0-90 stopni na serwo)
406         rotate_right_servo(750, 1000);
407         // Wykonaj skan odległości i zapal diodę odległości
408         make_scan();
409
410         // zapal diodę informacyjną: 75% światła
411         set_info_diode(.75f);
412         _delay_ms(delay_time);
413         // if (dist_cm < 10) break;
414
415         // Wykonaj obrót 90 stopni w lewo -> (90-0 stopni na serwo)
416         undo_rotate_right_servo(750, 1000);
417         make_scan();
418
419         // Wykonaj obrót 90 stopni w lewo -> (90-0 stopni na serwo)
420         undo_rotate_right_servo(750, 1000);
421         make_scan();
422
423         // zapal diodę informacyjną: 50% światła
424         set_info_diode(.50f);
425         _delay_ms(delay_time);
426
427         drive_forward();
428
429         // zapal diodę informacyjną: 25% światła
430         set_info_diode(.25f);
431         _delay_ms(delay_time);
432
433         update_adc();
434
435         // Sygnał o zakończeniu działania programu
436         set_info_diode(0.10f);
437         _delay_ms(5000);
438     }
439 }
```

Funkcja wykonuje w pętli następujące kroki:

1. Obróć serwomechanizm w prawo i zmierz dystans do przeszkody
2. Powróć serwomechanizmem na pozycję początkową i zmierz dystans na wprost
3. Jedź do przodu
4. Sprawdź napięcie poprzez konwerter ADC

Pomiędzy każdym krokiem wyświetlamy informacje o obecnym stanie na diodzie informacyjnej sygnałem PWM (odpowiednio wypełnienie: 95%, 75%, 50%, 25% i 10%). Zmiana naświetlenia diody pomaga rozróżnić czy robot znajduje się w odpowiednim stanie oraz okazała się bardzo pomocna w trakcie procesu testowania.

4.3 Konfiguracja Timerów

Timer0

Timer0 jest wykorzystywany w module sensora HC-SR04. Służy jako licznik czasu trwania sygnału Echo zwracanego przez sensor. Ponieważ jest to jego jedyne zastosowanie, cała inicjalizacja dzieje się na potrzeby funkcji mierzącej odległość a sam timer nie posiada funkcji inicjalizującej. Dokładniejszy opis wykorzystania licznika znajduje się w rozdziale „sensor”.

Timer1

Timer1 jest wykorzystywany jako generator sygnału PWM dla serwomechanizmu oraz diody informacyjnej. Ponieważ wykorzystany w układzie serwomechanizm (SG90) wymaga określonej częstotliwości taktowania równej 50Hz, oraz pracy na poziomie 5%-10% wypełnienia parametry zostały dobrane tak aby spełnić to wymaganie.

Timer1 jest 16-bitowym licznikiem co pozwala mu na odliczanie do wartości maksymalnej 65535. W celu zapewnienia stabilnej częstotliwości – krytycznej dla działania serwomechanizmu, wykorzystano tryb generowania Phase-Correct PWM. Przez ustawienie bitów WGM1 rejestrów kontrolnych na wartość 10

$(1010)_2$ ustawiono wartość graniczną zliczania TOP wyznaczaną przez rejestr ICR1 – pozwala to na skorzystanie z obu wyjść PWM (OCR1A i OCR1B) do manipulowania wypełnieniem (w naszym przypadku, na możliwość generowania sygnału dla serwo oraz diody z różnym wypełnieniem jednocześnie).

Dodatkowo zmieniono dzielnik częstotliwości na wartość 8. Po ustawieniu wartości granicznej TOP na 10000 otrzymujemy wynikową częstotliwość 50Hz zgodnie ze wzorem podanym w dokumentacji dla trybu Phase-Correct Timera1:

$$f = \frac{f_{clk}}{2 \cdot N \cdot TOP} = \frac{8 \text{ MHz}}{2 \cdot 8 \cdot 10000} = 50 \text{ [Hz]}$$

Takie wartości dodatkowo pozwalają na posługiwanie się liczbami całkowitymi do opisu zakresu wypełnienia: 5% (500), 7.5% (750), 10% (1000) dla kąta obrotu serwomechanizmu. Poniżej znajduje się fragment kodu opisujący inicjalizację odpowiednich rejestrów kontrolnych:

```
65 void init_timer1() {
66     /* Funkcja ustawia rejestry dla Timera1
67     * Timer1 generuje sygnał PWM dla Serwomechanizmu, oraz głównej diody informacyjnej
68     @ Pin servo: OC1B (PB2)
69     @ Pin diody: OC1A (PB1)
70     > TCCR1A, TCCR1B, ICR1, DDRB
71     */
72
73     // Ustaw generowanie sygnału PWM w trybie Phase-Correct dla Timera 1
74     // Wartość TOP dla licznika jest ustawiana poprzez rejestr ICR1 (WGM1 = 1010)
75     TCCR1A |= (1 << WGM11);
76     TCCR1B |= (1 << WGM13);
77
78     // Ustaw dzielnik częstotliwości na wartość 8 (CS1 = 010),
79     // wynikowa częstotliwość generowanego sygnału PWM to: 8Mhz / 8 / TOP (10000) / 2 (Phase-Correct) = 50Hz
80     TCCR1B |= (1 << CS11);
81
82     // Ustaw wartość TOP - górne ograniczenie zliczania Timera1
83     ICR1 = 10000;
84 }
```

Timer2

Timer2 jest wykorzystywany jako generator sygnału PWM dla diody odległości sensora oraz sterowania mocą silników. Ze względu na wykorzystanie z silnikami, został włączony tryb Phase-Correct PWM. Odpowiednie dobranie dzielnika częstotliwości pozwoliło na osiągnięcie wynikowej częstotliwości 15.625kHz.

```
85 void init_timer2() {
86     /* Funkcja ustawia rejestry dla Timera2
87     * Timer2 generuje sygnał PWM dla mocy silników, oraz diody odległości sensora HC-SR04
88     @ Pin moc silników: OC2A (PB3)
89     @ Pin diody: OC2B (PD3)
90     > TCCR2A, TCCR2B
91     */
92
93     // Ustaw generowanie sygnału PWM w trybie Phase-Correct dla Timera 2
94     // Wartość TOP dla licznika wynosi 255 (0xFF) (WGM2 = 001)
95     TCCR2A |= (1 << WGM20);
96
97     // Ustaw dzielnik częstotliwości na wartość 1 (CS2 = 001),
98     // wynikowa częstotliwość generowanego sygnału PWM to: 8Mhz / 510 / 1 (Phase-Correct) = 15.625kHz
99     TCCR2B |= (1 << CS20);
100 }
```

4.4 Moduły Sterujące

4.4.1 Serwomechanizm

Inicjalizacja

Najważniejsza część konfiguracji dla serwomechanizmu odbywa się w funkcji inicjalizującej timer1. Po ustaleniu taktowania na 50Hz oraz przechowania progu zliczania w rejestrze ICR1, należy jedynie ustawić pin serwo jako pin wyjściowy poprzez rejestr DDRD.

```
115 void init_servo() {
116     /* Funkcja ustawia rejestry dla servo
117     * Większość ustawień zawiera się w funkcji inicjalizującej timer1 (init_timer1),
118     * ponieważ w oparciu o niego generowany jest sygnał PWM
119     @ Pin serwo: OC1B (PB2)
120     > DDRB
121     */
122
123     // Ustaw pin dla serwo jako wyjściowy
124     DDRB |= (1 << DDB2);
125 }
```

Obrót serwomechanizmem

Użycie serwomechanizmu składa się z dwóch kroków:

1. Ustawienie progu (wypełnienia) na rejestrze OCR1B
2. Ustawienie wartości bitów COM1B w rejestrze TCRR1A co jest odpowiedzialne za rozpoczęcie generowania sygnału PWM na pinie serwo.

W celu dodatkowego „spowolnienia” ruchu obrotu (zbyt szybkie obracanie powodowało rozregulowywanie się całej głowy robota), zaimplementowany został powolny obrót, który stopniowo zmienia wartość wypełnienia od ustalonego 7.5% (punkt rotacji 0 stopni na serwo) do 10% (rotacja o 90 stopni). Ponieważ wartość wypełnienia obliczamy wzorem:

$$\frac{OCR1B}{TOP} \cdot 100$$

Gdzie wartość TOP przechowuje u nas rejestr ICR1 (10000). Odpowiednie wartości OCR1B to 750 i 1000. Poniżej znajduje się kod procedur obracających serwo poprzez ustawienie stopniowo wartości OCR1B oraz ustawienie bitów generowania sygnału non-inverted PWM na czas działania obrotu.

```
353 void rotate_right_servo(uint16_t duty_low, uint16_t duty_high) {
354     /* Funkcja rotuje serwomechanizm od pozycji początkowej 0 stopni do pozycji 90 stopni
355     @ Pin serwo: OC1B (PB2)
356     > TCRR1A, OCR1B
357     */
358
359     // Ustaw serwo na wartość startową - wypełnienie 750 / TOP = 7.5% - 1.5 ms dla 5000 - 0 stopni
360     OCR1B = duty_low;
361
362     // Włącz sygnał non-inverted PWM na pinie OC1B (PB2)
363     TCRR1A |= (1 << COM1B1);
364
365     // Pętla przechodząca przez zakres 0-90 stopni (wypełnienie 7.5% - 10%) w spowolniony sposób
366     for (uint16_t duty = duty_low; duty < duty_high; duty += 10) {
367         OCR1B = duty;
368         _delay_ms(20);
369     }
370
371     // Wyłącz sygnał non-inverted PWM na pinie OC1B (PB2)
372     TCRR1A &= ~(1 << COM1B1);
373 }
374
375 void undo_rotate_right_servo(uint16_t duty_low, uint16_t duty_high) {
376     /* Funkcja rotuje serwomechanizm spowrotem od pozycji 90 stopni do pozycji początkowej 0 stopni
377     @ Pin serwo: OC1B (PB2)
378     > TCRR1A, OCR1B
379     */
380
381     // Ustaw serwo na wartość startową - wypełnienie 1000 / TOP = 10% - 2 ms dla 5000 - 90 stopni
382     OCR1B = duty_high;
383
384     // Włącz sygnał non-inverted PWM na pinie OC1B (PB2)
385     TCRR1A |= (1 << COM1B1);
386
387     // Pętla przechodząca przez zakres 90-0 stopni (wypełnienie 10% - 7.5%) w spowolniony sposób
388     for (uint16_t duty = duty_high; duty > duty_low; duty -= 10) {
389         OCR1B = duty;
390         _delay_ms(20);
391     }
392
393     // Wyłącz sygnał non-inverted PWM na pinie OC1B (PB2)
394     TCRR1A &= ~(1 << COM1B1);
395 }
```

4.4.2 ADC – Analog to Digital Converter

Inicjalizacja

```
179 void init_adc() {
180     /* Funkcja ustawiająca rejestry ADC
181     @ Pin Analogowy: ADC5
182     @ Piny diod informacyjnych: PD0 i PD1
183     > ADMUX, ADCSRA, ADCSRB, DIDR0, DD0D, PORTD
184     */
185
186     // -- ADC Multiplexer
187     // Ustaw referencyjne napięcie (górne) napięcie na ADC5 (REFS = 01)
188     ADMUX |= (1 << REFS0);
189     // Ustaw wybrany kanał MUX na wartość 5 dla ADC5 (MUX = 0101)
190     ADMUX |= (1 << MUX2) | (1 << MUX0);
191
192     // -- ADC Control and Status Register A
193     // Włącz obsługę ADC (ADEN = 1), uruchom pierwszą konwersję (ADSC = 1),
194     // oraz automatyczne uruchamianie konwersji (ADATE = 1)
195     ADCSRA |= (1 << ADEN) | (1 << ADSC) | (1 << ADATE);
196
197     // Ustaw dzielnik częstotliwości na wartość 64 (ADPS = 110),
198     // wynikowa częstotliwość próbkowania ADC = 8MHz / 64 = 125kHz
199     ADCSRA |= (1 << ADPS2) | (1 << ADPS1);
200
201     // -- ADC Control and Status Register B
202     // Ustaw rejestr na wartość 0 -> Free Running Mode (ADFS = 000) - Tryb zapętlenia konwersji ADC
203     ADCSRB = 0;
204
205     // -- Digital Input Disable Register 0
206     // Ustaw rejestr odłączający cyfrowe wejście na pinie analogowym ADC5
207     DIDR0 |= (1 << ADC5D0);
208
209     // Ustaw piny diod informacyjnych o obecnie zmierzonym poziomie napięcia na wyjściowe
210     DD0D |= (1 << DD0D0) | (1 << DD0D1);
211
212     // Ustaw stan diod informacyjnych na niski
213     PORTD &= ~(1 << DD0D0);
214     PORTD &= ~(1 << DD0D1);
215 }
```

Inicjalizacja konwertera analogowo-cyfrowego rozpoczyna się od ustawienia bitów rejestru ADMUX. Napięcie referencyjne, według którego odmierzana jest wartość na pinie ADC5 jest napięciem zasilania Vcc 5[V], dlatego ustawione zostają bity REFS na wartość 1 = $(01)_2$. Bity MUX przyjmują wartość 5 $(0101)_2$ ze względu na wykorzystanie kanału piątego ADC5.

W statusie kontrolnym ADC zostają ustawione flagi ADEN (włącz ADC), ADSC (uruchom pierwszą konwersję ADC), ADATE (automatycznie uruchamiaj konwersję po otrzymaniu sygnału trigger zdefiniowanego w rejestrze ADCSRB).

Następnie w tym samym rejestrze definiowany jest dzielnik częstotliwości na wartość 64 (ADPS = 110). Wynikowa częstotliwość wynosi 125kHz, która jest wartością w zalecanym zakresie przez producenta (100kHz – 200kHz).

Następnie zdefiniowanie rejestru ADCSRB na wartość 0 ustawia wspomniany wcześniej trigger na „Free Running Mode” co oznacza, że konwersja ADC będzie wykonywana w zapętleniu bez przerwy.

Na samym końcu ustawione zostają bity odpowiedzialne za wyjście diod oraz odłączenie wejścia cyfrowego na pinie analogowym ADC5.

Odczyt wartości ADC

```
323 void update_adc() {
324     /* Funkcja mierzy poziom napięcia ADC oraz zapala diody informacyjne
325     @ Piny diod informacyjnych: PD0 i PD1
326     > PORTD
327     */
328
329     // Przeskalowane wartości brzegowe dla { 0V, 1.7V, 2.5V, 5V }
330     // z dodatkowym zapasem na możliwe odchylenia
331     uint16_t threshold_values[4] = { 0, 327, 491, 962 };
332
333     // Korzystając z rejestru ADC sprawdź wartość dolnych 10 bitów
334     uint16_t adc_value = ADC & 0x3ff;
335
336     // Sprawdź odpowiedni próg dla zmierzonej wartości
337     // Ostatecznie i jest równe indeksowi w tablicy z mniejszym napięciem od obecnego
338     uint8_t i = 1;
339     while (i < 4 && adc_value >= threshold_values[i]) { i++; }
340     i--;
341 }
```

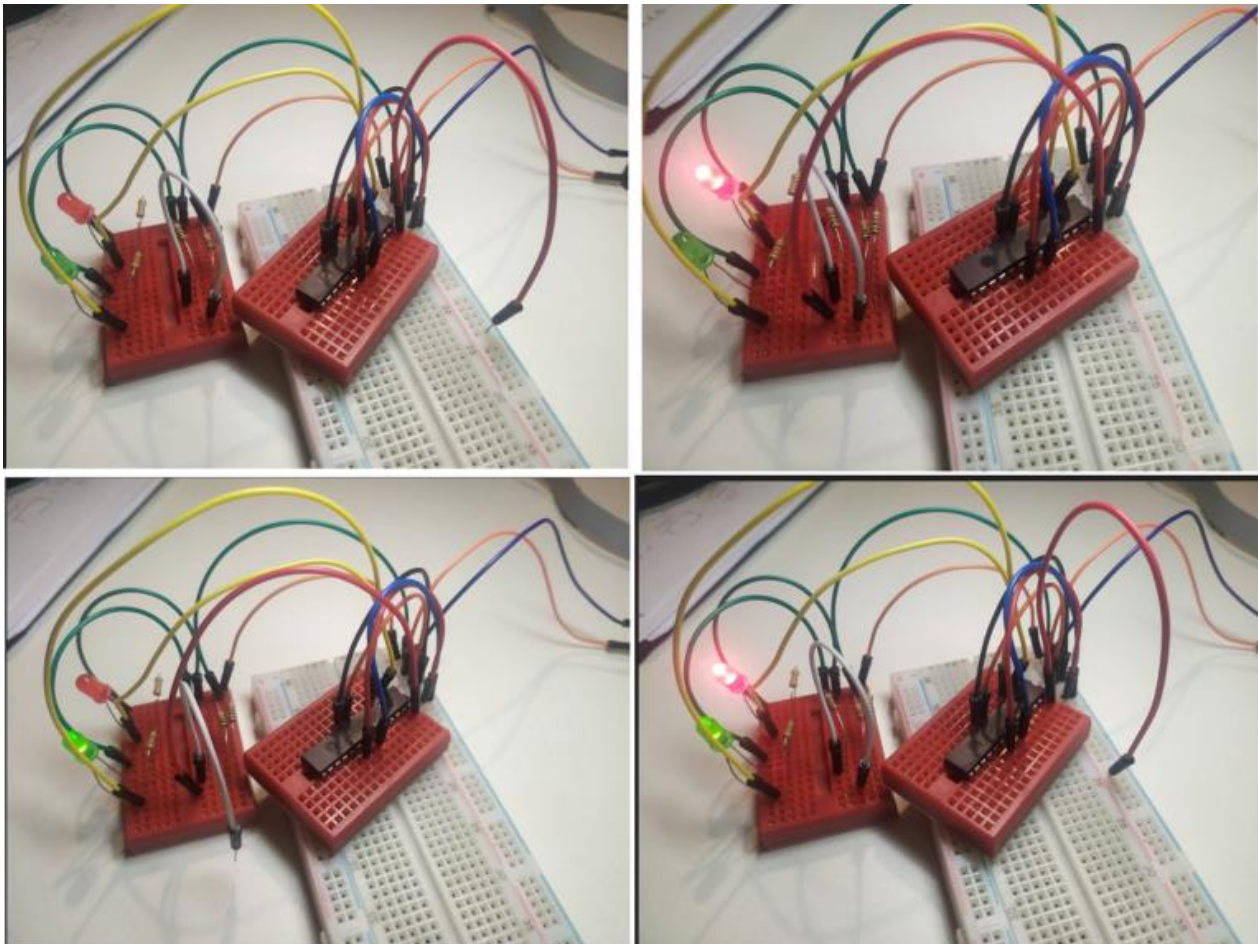
Wykorzystanie odpowiednio skonfigurowanego ADC polega na sprawdzeniu dolnych 10 bitów rejestru ADC. Odczytana wartość jest liczbą całkowitą z zakresu 0-1023. Producent podaje wzór, według którego

za pomocą wartości referencyjnej napięcia należy odczytywać napięcie na pinie analogowym. Korzystając z tego wzoru została przygotowana tablica wartości progowych, w których znajdują się przeskalowane wartości dla napięcia: 0V, 1.7V, 2.5V, 5V, odpowiednio pomniejszone o możliwe odchylenie w dół.

```
342 // Zapal diody informacyjne, które wyznaczają zmierzony próg w kodzie binarnym
343 // Jeżeli dolny bit zapalony, (i = xxx1) zapal diodę PD0, w innym przypadku zgaś (i = xxx0)
344 if (i & (1 << 0)) {
345     PORTD |= (1 << DDD0);
346 } else {
347     PORTD &= ~(1 << DDD0);
348 }
349 // Jeżeli górny bit zapalony, (i = xx1x) zapal diodę PD1, w innym przypadku zgaś (i = xx0x)
350 if (i & (1 << 1)) {
351     PORTD |= (1 << DDD1);
352 } else {
353     PORTD &= ~(1 << DDD1);
354 }
355 }
```

Po obliczeniu indeksu progu (0-3), wyświetlana zostaje informacja na dwóch diodach. Zakodowana informacja to numer progu w systemie binarnym (odpowiednio zapalone diody symbolizują jeden bit).

Testy



Przeprowadzono testy badające zachowanie czujnika napięcia. W tym celu wykonano dzielnik z rezystorów, w którym dodatkowo dwa rezystory są podłączone do siebie równolegle. Przyłączenie lub odłączanie rezystora powoduje wygenerowanie napięcia odpowiednio 2.5V lub 1.7V. Na rysunku widać biały kabel, który odpowiada za „podłączenie” rezystora równoległego do dzielnika napięcia. W przypadku gdy pin ADC5 (czerwony kabel) jest połączony do masy (pierwszy obraz) widać, że nie świeci się żadna dioda co symbolizuje próg 0. Następnie podłączamy pin ADC5 do dzielnika napięcia z rezystorem równoległym. Generuje to napięcie $\sim 1.7V$ i jak widać zapala czerwoną diodę co koduje liczbę 1 – próg pierwszy. Na następnych obrazkach widać podłączenie do dzielnika napięcia bez rezystora równoległego oraz podłączenie do zasilania.

4.4.3 Silniki

Inicjalizacja

```
126 void init_engines() {
127     /* Funkcja ustawia piny obu silników (kierunki przód i tył dla lewego i prawego silnika) oraz pin mocy silnika
128     @ Piny lewego silnika: PD6 (przód) i PD7 (tył)
129     @ Piny prawego silnika: PB4 (przód) i PB5 (tył)
130     @ Pin mocy silnika: OC2A (PB3)
131     > DDRD, DDRB
132     */
133
134     // Ustaw piny lewego silnika jako piny wyjściowe
135     DDRD |= (1 << DDD6) | (1 << DDD7);
136
137     // Ustaw piny prawego silnika jako piny wyjściowe
138     DDRB |= (1 << DDB4) | (1 << DDB5);
139
140     // Ustaw pin mocy silników jako pin wyjściowy
141     DDRB |= (1 << DDB3);
142 }
```

Inicjalizacja silników polega na ustawieniu wszystkich pinów sterujących oboma silnikami (prawy i lewy) dla każdego kierunku (przód / tył) oraz pinu odpowiedzialnego za moc silników jako piny wyjściowe.

Sterowanie Silnikami

Sterowanie silnikami odbywa się poprzez wygenerowanie sygnału PWM na pinie mocy silnika. Poprzez wypełnienie reguluje się siłę z jaką obracają się koła. Ustawienie stanu wysokiego na pinie kierunku w którym obracają się koła oraz stanu niskiego na pinie kierunku przeciwnego, sprawia, że motor kręci się w ustalonym kierunku z określoną mocą.

Poniżej została przedstawiona funkcja odpowiedzialna za poruszanie się do przodu. Nie napisano funkcji dla każdego kierunku, natomiast wyglądałyby one analogicznie, różniąc się jedynie ustawionymi stanami wysokimi i niskimi na pinach kierunkowych.

```

291 void drive_forward() {
292     /* Funkcja porusza prawym i lewym silnikiem z taką samą mocą przez określony kwant czasu
293     @ Piny lewego silnika: PD6 (przód) i PD7 (tył)
294     @ Piny prawego silnika: PB4 (przód) i PB5 (tył)
295     @ Pin mocy silnika: OC2A (PB3)
296     > PORTD, PORTB, TCCR2A, OCR2A
297     */
298
299     // -- Ustaw piny odpowiedzialne za poruszanie do przodu na stan wysoki
300     // lewy silnik przód
301     PORTD |= (1 << DDD6);
302     // prawy silnik przód
303     PORTB |= (1 << DDB4);
304
305     // Ustaw wypełnienie na pinie mocy silników na 50 = ~20%
306     OCR2A = 50;
307     // Włącz sygnał non-inverted PWM na pinie mocy silników OC2A
308     TCCR2A |= (1 << COM2A1);
309
310     // Pozostań w tym stanie przez 2 sekundy - jedź do przodu
311     _delay_ms(2000);
312 }

```

Po czasie 2s, ustawiany jest stan niski na pinach kierunkowych, następnie moc silnika zmieniana na 0 oraz dodatkowo odłączone zostaje generowanie sygnału non-inverted PWM na pinie mocy silnika.

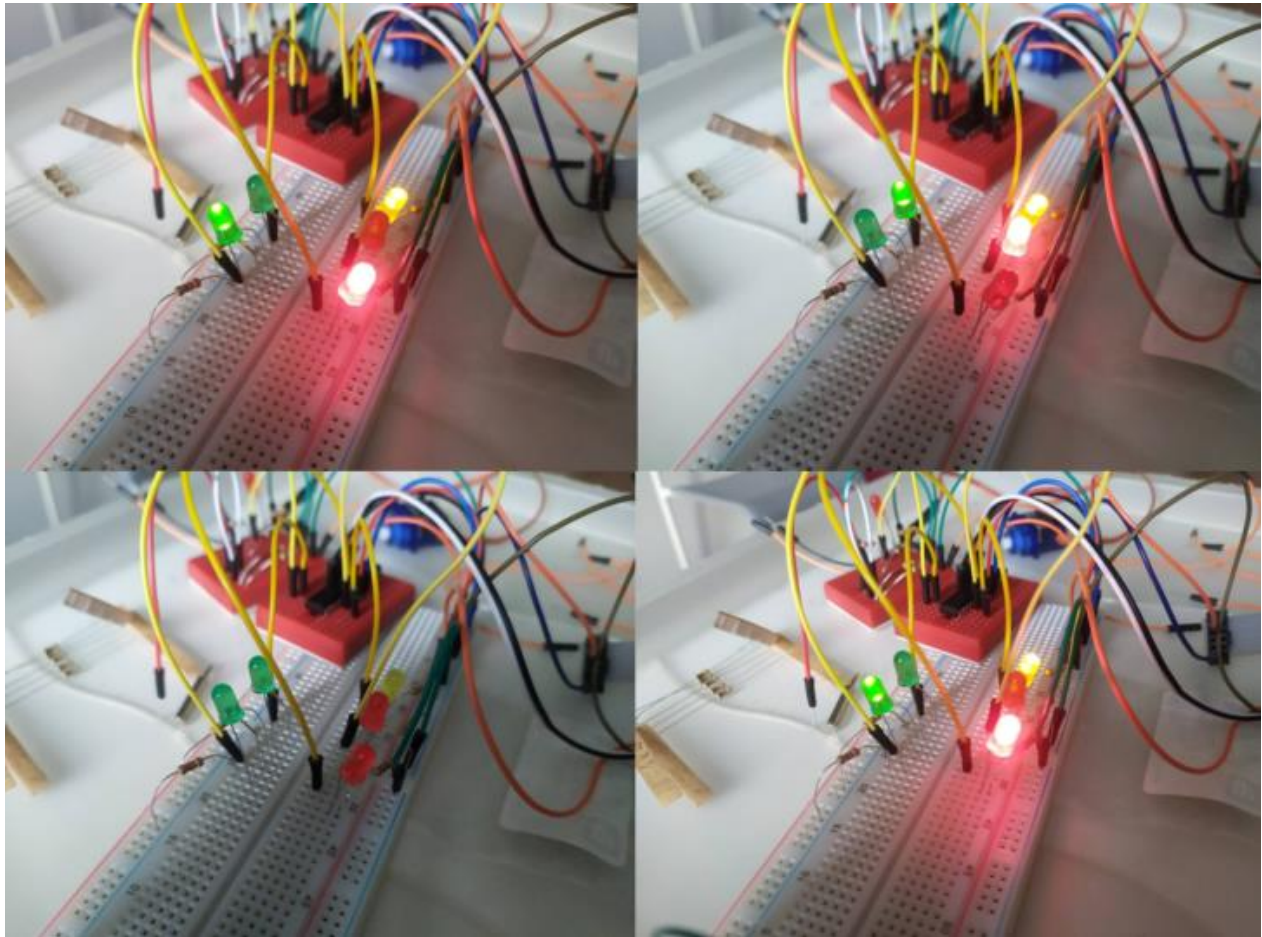
```

313     // -- Ustaw piny odpowiedzialne za poruszanie do przodu na stan niski
314     // lewy silnik przód
315     PORTD &= ~(1 << DDD6);
316     // prawy silnik przód
317     PORTB &= ~(1 << DDB4);
318
319     // Wyłącz sygnał PWM na pinie mocy silników OC2A
320     TCCR2A &= ~(1 << COM2A1);
321     OCR2A = 0;
322 }

```

Testy

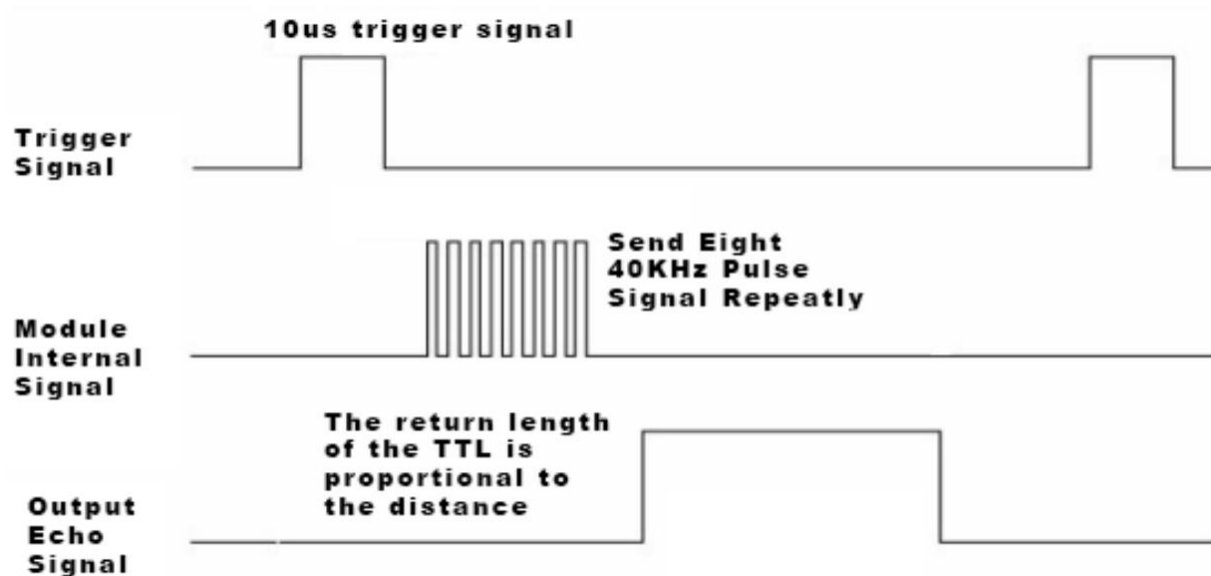
Niestety testy na prawdziwym module silników nie były możliwe, ponieważ używany w robocie mostek H spalił się w trakcie dostrajania parametrów. W tym celu zostało utworzone środowisko testowe z wykorzystaniem diod.



Na zamieszczonym zdjęciu, zielone diody odpowiadają za silnik lewy, czerwone za silnik prawy. Żółta dioda otrzymuje sygnał PWM mocy silników. W trakcie testów uruchamiany był kod poruszający silnikami na zmianę do przodu i do tyłu z różnym wypełnieniem PWM. Co niestety słabo widać na zdjęciu żółta dioda zmienia zakres przy jeździe w tył (1 i 4 zdjęcie) z 40 do 20, w przód od 100 do 50. Pomiędzy każdym kierunkiem silniki zatrzymują się na 2 sekundy (3 zdjęcie).

4.4.4 Ultrasonic Distance Sensor

Sensor wykorzystuje ultradźwiękowy moduł HC-SR04 aby odczytywać odległość od przeszkód wyrażoną w centymetrach. Zgodnie z dokumentacją dostarczoną przez producenta sensor działa na podstawie wyzwalanego sygnału Trigger, trwającego 10 us. Po otrzymaniu sygnału, sensor wysyła 8 impulsów, na podstawie których otrzymuje odpowiedź od otoczenia. Odpowiedź wyrażona jest jako stan wysoki na wyjściowym porcie sensora (wejściowym porcie w programie) nazywanym pinem Echo. Czas trwania stanu wysokiego w us na pinie Echo jest 58 razy mniejszy niż dystans od przeszkody wyrażony w centymetrach.



Inicjalizacja

Sensor początkowo inicjalizowany jest poprzez ustawienie pinów Trigger oraz Echo na odpowiednio pin wyjściowy, oraz pin wejściowy:

```

143 void init_sensor() {
144     /* Funkcja ustawia rejestry wykorzystywane przez sensor HC-SR04
145     * Sensor wykorzystuje timer0 aby odmierzać czas trwania sygnału wysokiego na pinie ECHO
146     * Dodatkowo wykorzystuje mechanizm przerwań aby reagować na zmieniającą się wartość sygnału na tym pinie
147     * Dlatego funkcja jest powiązana z rutynami obsługi przerwań dla przerwań PCINT2 oraz TIMER0_OVF
148     * Dodatkowo dioda odległości wykorzystuje Timer2, którego konfiguracja znajduje się w funkcji init_timer2()
149     @ Pin trigger: PD2
150     @ Pin echo: PD4 (PCINT20)
151     @ Pin diody odległości: (OC2B) PD3
152     > DDRD, PCICR, PCMSK2, TCCR2A
153     */
154
155     // Ustaw pin Trigger jako pin wyjściowy
156     DDRD |= (1 << DDD2);
157     PORTD &= ~(1 << DDD2);
158
159     // Ustaw pin Echo jako pin wejściowy
160     DDRD &= ~(1 << DDD4);
161     PORTD &= ~(1 << DDD4);
162 }

```

Aby poprawnie mierzyć czas trwania sygnału wysokiego na pinie Echo, wykorzystany został Timer0 oraz specjalna procedura obsługi przerwań dla zmiany stanu pinu (przerwanie PCINT2). Na początku ustawiany jest bit PCIE2 rejestru PCICR, który uruchamia obsługę przerwań dla grupy 2 pinów PCINT. Następnie w rejestrze maskującym przerwania dla tej grupy ustawiony został bit PCINT20, który informuje mikrokontroler, aby wygenerował przerwanie PCINT2 w momencie kiedy nastąpi zmiana na którymś z pinów określonych przez bity PCMSK2 (w tym przypadku na pinie PCINT20 – pinie Echo). Następnie włączamy globalny system przerwań, który „odblokowuje” wywoływanie procedur obsługi przerwań.

```

164 // -- Ustawienia dla obsługi przerwań zmiany stanu na pinie ECHO
165 // Ustaw rejestr Pin Change Interrupt aby reagował na przerwania z 3-ciej grupy pinów przerwań PCI2 (PCINT16 - PCINT23)
166 PCICR |= (1 << PCIE2);
167 // Ustaw reagowanie na przerwanie dla zmiany stanu na pinie ECHO (PCINT20)
168 PCMSK2 |= (1 << PCINT20);
169
170 // Włącz globalny system przerwań - wykorzystywane przez sensor
171 sei();

```

Na sam koniec inicjalizowany jest pin PWM diody odległości, która pozwala sprawdzać dystans odmierzony przez sensor:

```

173 // -- Ustawienia diody odległości
174 // Ustaw pin diody odległości jako pin wyjściowy
175 DDRD |= (1 << DDD3);
176 // Włącz generowanie sygnału non-inverted PWM na diodzie odległości
177 TCCR2A |= (1 << COM2B1);
178 }

```

Procedury Obsługi Przerwań ISR

Atmega328p definiuje 24 rejestry, które mogą obsługiwać przerwania typu PCINT (Pin Change Interrupt). Wywoływane są one w momencie kiedy na pinie zostanie wykryte zbocze narastające lub malejące, zmieniające wartość określoną na tym pinie. 24 rejestry podzielone zostały na 3 grupy – PCIE0 (PCINT0-PCINT7), PCIE1 (PCINT8 – PCINT15) i PCIE2 (PCINT16 – PCINT23). W kodzie ustawione zostało przerwanie tylko dla grupy 2 (PCIE2 = 1 dla rejestru PCICR) oraz tylko dla pinu Echo (PCINT20). W ten sposób mikrokontroler wywoła obsługę przerwania zmiany stanu dla grupy 2 tylko w momencie kiedy sygnał zmieni się na pinie Echo. Procedura obsługi przerwania wygląda następująco:

```

24 ISR(PCINT2_vect) {
25     /* Rutyna obsługi przerwania zmiany stanu na pinie Echo (PCINT20)
26     * Rutyna dodatkowo włącza lub wyłącza obsługę przerwania przepełnienia timera0
27     @ Pin Echo: PD4 (PCINT20)
28     > PIND, TNCT, TIMSK, TCCR0B, echo_finished
29     */
30
31     // Jeżeli zbocze narastające - start sygnału Echo
32     if (PIND & (1 << PIND4)) {
33
34         // Ustaw wartość licznika na 0
35         TCNT0 = 0;
36
37         // Włącz obsługę przerwania przepełnienia timera0
38         TIMSK0 = (1 << TOIE0);
39
40         // Ustaw dzielnik częstotliwości timera na równy 1 (CS0 = 001)
41         // Timer0 pracuje z częstotliwością 8MHz - czas trwania jednego okresu: .125 us, przepełnienia: 32 us
42         TCCR0B = (1 << CS00);
43
44     // Jeżeli zbocze malejące - koniec sygnału Echo
45     } else {
46         // Zatrzymaj mechanizm przerwań dla przepełnienia Timera0
47         TIMSK0 &= ~(1 << TOIE0);
48
49         // Zatrzymaj zliczanie Timera0
50         TCCR0B &= ~(1 << CS00);
51
52         // Ustaw wartość zmiennej odpowiedzialną za koniec oczekiwania w funkcji make_scan
53         echo_finished = 1;
54     }
55 }

```

Procedura rozpoczyna się od określenia czy obecna zmiana sygnału to zbocze opadające czy zbocze narastające. W przypadku zbocza narastającego (Pin Echo przechodzi w stan wysoki) procedura uruchamia Timer0 rozpoczynając od wartości 0 w trybie zliczania z częstotliwością równą zegarowi wewnętrznemu (CS0 = 001) czyli 8MHz. Timer0 odmierza wartości od 0 do 255 i w momencie osiągnięcia wartości progowej (255) resetuje się, wywołując przerwanie TIMERO_OVF. Chcąc rejestrować to przerwanie aby odmierzać czas, dodatkowo w kodzie pojawia się ustawienie bitu TOIE0 (Timer0 Interrupt Enable) rejestru TIMSK0 co włącza obsługę przerwania przepełnienia Timera0.

W przypadku zbocza opadającego, czyścimy ustawione flagi, zatrzymując timer0 oraz ustawiamy wartość zmiennej informującej o końcu przetwarzania sygnału Echo (echo_finished) na 1.

```

17 // ----- Obsługa Przerwań -----
18 ISR(TIMERO_OVF_vect) {
19     /* Rutyna obsługi przerwania Timera0 */
20
21     // Zwiększ wartość zmiennej liczącą czas trwania sygnału Echo
22     timer0_overflows++;
23 }

```

Procedura obsługi przerwania timera0 jest krótka i sprowadza się do zwiększenia wartości zmiennej liczącej liczbę przepełnień timera0.

Pomiar Odległości

```

241 uint16_t make_scan() {
242     /* Funkcja wykonuje skan w oparciu o sensor HC-SR04, zwracając wynik w cm
243     * dodatkowo zapala przeskalowaną wartość na diodzie odległości z pomocą sygnału PWM
244     @ Pin trigger: PD2
245     @ Pin echo: PD4 (PCINT20)
246     @ Pin diody odległości: (OC2B) PD3
247     > PORTD, OCR2B
248     */
249
250     // Ustaw zmienną liczącą czas trwania sygnału Echo na 0
251     timer0_overflows = 0;
252
253     // Wyzwól sygnał wysoki na pinie Trigger na 10 us (specyfikacja HC-SR04)
254     PORTD |= (1 << PIND2);
255     _delay_us(10);
256     PORTD &= ~(1 << PIND2);

```

Po ustawieniu wszystkich rejestrów oraz procedur obsługi przerwania, chcąc wykonać skan wartość odliczonych przepełnień timera0 zostaje wyzerowana oraz wysłany zostaje impuls na pin Trigger zgodnie ze specyfikacją.

```

265 // Oczekuj na zakończenie przetwarzania wysokiego sygnału na pinie Echo
266 while (!echo_finished) {}
267 echo_finished = 0;

```

Po wysłaniu impulsu na pinie Trigger, mikrokontroler oczekuje na wystąpienie przerwania zmiany stanu pinu Echo. W tym celu program oczekuje w pętli, która sprawdza wartość zmiennej echo_finished, która zostaje ustawiona po zakończeniu przetwarzania procedury obsługi przerwania zbocza opadającego. Jeżeli to nastąpi zmienna jest ustawiana na zero.

```
269 // Oblicz odległość w cm na podstawie czasu trwania impulsu Echo
270 // Timer0 odmierza czas z częstotliwością 8MHz i zapamiętujemy każde 256 odliczeń
271 // Częstotliwość 8Mhz daje okres równy 0.125 us * 256 = 32 us za każde przepełnienie timera0
272 uint16_t echo_time_us = timer0_overflows * 32;
273
274 // Specyfikacja sensora HC-SR04 podaje wzór na obliczenie odległości w cm
275 // dist in cm = echo time in us / 58
276 uint16_t dist_cm = echo_time_us / 58;
```

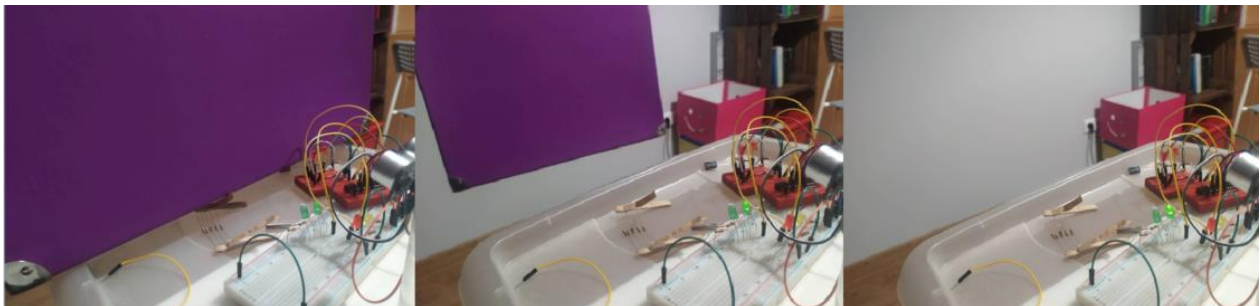
Następnie zostaje obliczona odległość do przeszkody w cm na podstawie ilości przepełnień Timera0. Znając częstotliwość z jaką działa Timer0 – 8MHz oraz ilość zliczonych wartości w każdym przepełnieniu (256), można wyznaczyć czas trwania jednego przepełnienia timera0, które wynosi 32 us.

Czas trwania stanu wysokiego Echo to iloczyn ilości przepełnień i czasu trwania jednego przepełnienia. Zgodnie ze specyfikacją, aby otrzymać dystans w centymetrach, należy otrzymaną wartość podzielić przez 58.

```
278 // Zapal diodę w zależności od wyliczonej odległości
279 // jeżeli wartość poza zakresem, zgaś diodę
280 if (dist_cm > 2 && dist_cm < 250) {
281     OCR2B = dist_cm;
282 } else {
283     OCR2B = 0;
284 }
285
286 // Zwróć obliczony wynik
287 return dist_cm;
```

Na sam koniec, ustawione zostaje wypełnienie na diodzie odległości. Ponieważ producent zapewnił, że moduł HC-SR04 działa w odległościach 2 – 400 cm, zakres ten zostaje lekko przycięty aby zapewnić przedział wartości 2 – 255. Następnie obliczony wynik jest zwracany jako wartość 16-bitowa.

Testy



Działanie sensora zostało przetestowane z pomocą diody odległości. W zależności od zmierzonej odległości w centymetrach zielona dioda ściemniała się dla przeszkód bliskich (1 zdjęcie) i rozjaśniała dla przeszkód dalekich (3 zdjęcie)

4.5 Konfiguracja mikrokontrolera Atmega328p

Wyprowadzenie pinów w programie

ATMega328P and Arduino Uno Pin Mapping

Arduino function					Arduino function
reset	(PCINT14/RESET) PC6	1	28	PC5 (ADC5/SCL/PCINT13)	analog input 5
digital pin 0 (RX) <i>adc-d0</i>	(PCINT16/RXD) PD0	2	27	PC4 (ADC4/SDA/PCINT12)	analog input 4
digital pin 1 (TX) <i>adc-d1</i>	(PCINT17/TXD) PD1	3	26	PC3 (ADC3/PCINT11)	analog input 3
digital pin 2 <i>trigg</i>	(PCINT18/INT0) PD2	4	25	PC2 (ADC2/PCINT10)	analog input 2
digital pin 3 (PWM) <i>hc-d</i>	(PCINT19/OC2B/INT1) PD3	5	24	PC1 (ADC1/PCINT9)	analog input 1
digital pin 4 <i>echo</i>	(PCINT20/XCK/T0) PD4	6	23	PC0 (ADC0/PCINT8)	analog input 0
VCC	VCC	7	22	GND	GND
GND	GND	8	21	AREF	analog reference
crystal	(PCINT6/XTAL1/TOSC1) PB6	9	20	AVCC	VCC
crystal	(PCINT7/XTAL2/TOSC2) PB7	10	19	PB5 (SCK/PCINT5)	digital pin 13 <i>e-r-b</i>
digital pin 5 (PWM)	(PCINT21/OC0B/T1) PD5	11	18	PB4 (MISO/PCINT4)	digital pin 12 <i>e-r-f</i>
digital pin 6 (PWM) <i>e-l-f</i>	(PCINT22/OC0A/AIN0) PD6	12	17	PB3 (MOSI/OC2A/PCINT3)	digital pin 11 (PWM) <i>e-pwr</i>
digital pin 7 <i>e-l-b</i>	(PCINT23/AIN1) PD7	13	16	PB2 (SS/OC1B/PCINT2)	digital pin 10 (PWM) <i>servo</i>
digital pin 8	(PCINT0/CLKO/ICP1) PB0	14	15	PB1 (OC1A/PCINT1)	digital pin 9 (PWM) <i>info-d</i>

Digital Pins 11, 12 & 13 are used by the ICSP header for MOSI, MISO, SCK connections (Atmega168 pins 17, 18 & 19). Avoid low-impedance loads on these pins when using the ICSP header.

Na powyższym zdjęciu zaznaczono przeznaczenie każdego wykorzystanego pinu w oprogramowaniu. Idąc rosnącymi wartościami pinów:

- PD0 i PD1 – piny diod informacyjnych napięcia ADC
- PD2 – pin Trigger sensora dla modułu HC-SR04
- OC2B (PD3) – pin diody odległości dla sensora
- PD4 (PCINT20) – pin Echo sensora
- PD6 – pin lewego silnika oznaczający ruch do przodu
- PD7 – pin lewego silnika oznaczający ruch do tyłu
- OC1A (PB1) – pin głównej diody informacyjnej
- OC1B (PB2) – pin serwomechanizmu
- OC2A (PB3) – pin mocy silników
- PB4 – pin prawego silnika oznaczający ruch do przodu
- PB5 – pin prawego silnika oznaczający ruch do tyłu
- ADC5 (PC5) – pin analogowy ADC

Programator Xgecu

Oprócz wgrywania programów za pomocą programatora USBasp, wykorzystano również programator XGecu. Pozwala on całkowicie odczytać / nadpisać wartość pamięci Flash, FuseBitów czy EEPROM. Dzięki temu urządzeniu możliwe było diagnozowanie błędów i problemów z Atmega 328p. Dzięki możliwości odczytania pamięci mikrokontrolera można zaobserwować np. skutki złe wgranego programu. Miało to miejsce, gdy kabelki łączące programator USBasp z mikrokontrolerem nie stykały i wgrywanie programu było nie pełne. Dzięki programatorowi Xgecu można było zauważyć błędy w ustawieniu FuseBitów, przypadkowemu wyczyszczeniu bootloadera Arduino itp.



Chip Program APP Version: 11.00

Program Range

ATMEGA328P

☒ FLASH
☒ EEPROM
☒ Config
☒ LOCK Bit

Start Adr: 00000000 End Adr: 00007FFF
☒ Read RC Calibration Bytes

Programming ...Succeeded

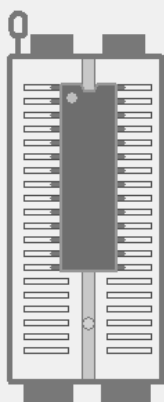
Save Log

Pins Detected Passed!
ID: 0x 1E 95 0FOK!
Erase ...Succeeded. Time : 0.031 S
Programming FLASH ...Succeeded. Time : 2297ms
Verifying FLASH ...Succeeded. Time : 922ms
Programming EEPROM ...Succeeded. Time : 422ms
Verifying EEPROM ...Succeeded. Time : 47ms
Programming Config ...Succeeded. Time : 0ms
Verifying Config ...Succeeded. Time : 63ms
Programming Lock bits.....Elapsed time: 0ms
Verify Lock bits.....Elapsed time: 62ms
Programming ...Succeeded

Program

BACK

Location in Socket


ZIF40

Przykładowe zdjęcie pokazuje widok po wgraniu finalnego programu.

Select IC

ATMEGA328P @DIP28

IC Information(No Project opened)

ChipType: MCU/MPU

ChkSum: 0x007F 9ACC

IC Size: 0x8000 Bytes (32768 Bytes) + 0x400 Bytes

Set Interface

☒ ZIF socket

☐ ICSP port

☐ ICSP_VCC Enable

Vcc Current Imax:

Default

☒ 8 Bits

☐ 16 Bits

Address	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	ASCII
0000-0000:	0C	94	3A	00	0C	94	A2	03	0C	94	A2	03	0C	94	A2	03	
0000-0010:	0C	94	A2	03	0C	94	FD	01	0C	94	A2	03	0C	94	A2	03	
0000-0020:	0C	94	A2	03	0C	94	A2	03	0C	94	A2	03	0C	94	A2	03	
0000-0030:	0C	94	A2	03	0C	94	A2	03	0C	94	A2	03	0C	94	A2	03	
0000-0040:	0C	94	00	03	0C	94	A2	03	0C	94	A2	03	0C	94	A2	03	
0000-0050:	0C	94	A2	03	0C	94	A2	03	0C	94	A2	03	0C	94	A2	03	
0000-0060:	0C	94	A2	03	0C	94	A2	03	01	00	01	08	00	07	48	01	H.
0000-0070:	08	01	0B	80	11	24	1F	BE	CF	EF	D8	E0	DE	BF	CD	BF	\$......
0000-0080:	E8	E6	F0	E0	40	E0	17	C0	B5	91	A5	91	35	91	25	91	@.....5.%.
0000-0090:	05	91	07	FD	0C	C0	95	91	85	91	EF	01	F9	2F	E8	2F	/./
0000-00A0:	05	90	0D	92	A2	17	B3	07	D9	F7	FE	01	04	C0	1D	92	
0000-00B0:	A2	17	B3	07	E1	F7	E4	37	F4	07	31	F7	0E	94	97	03	7..1....
0000-00C0:	0C	94	62	00	F8	94	FF	CF	0F	C0	0E	94	7F	03	38	F0	...b.....8.
0000-00D0:	0E	94	86	03	20	F0	95	23	11	F0	0C	94	8D	03	0C	94	...#.....
0000-00E0:	9F	03	11	24	0C	94	79	03	0E	94	43	02	70	F3	95	9F	...\$.y...C.p...
0000-00F0:	C1	F3	95	0F	50	E0	55	1F	62	9F	F0	01	72	9F	BB	27	...P.U.b...r...'
0000-0100:	F0	0D	B1	1D	63	9F	0A	27	F0	0D	B1	1D	0A	1F	64	9F	...c...'.....d.

Mozna również spojrzeć na ostatnie adresy w pamięci aby zauważyć, że faktycznie nie ma tam bootloadera (wszystkie komórki ustawione na wartość 255)

Address	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	ASCII
0000-7F00:	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	
0000-7F10:	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	
0000-7F20:	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	
0000-7F30:	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	
0000-7F40:	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	
0000-7F50:	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	
0000-7F60:	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	
0000-7F70:	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	
0000-7F80:	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	
0000-7F90:	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	
0000-7FA0:	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	
0000-7FB0:	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	
0000-7FC0:	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	
0000-7FD0:	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	
0000-7FE0:	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	
0000-7FF0:	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	

FLASH EEPROM Config Device.Info

FuseBity

Przed wgrywaniem programu, należało poprawnie skonfigurować fusebity używane przez mikrokontroler Atmega328p. Wynikowa konfiguracja wyglądała następująco:

Fuse Low Byte	Fuse High Byte	Extended Fuse Byte	Lock Bit Byte
☐ CKDIV8=0	☐ RSTDISBL=0	☐ BODLEVEL2=0	☐ BLB12=0
☐ CKOUT=0	☐ DWEN=0	☐ BODLEVEL1=0	☐ BLB11=0
☐ SUT1=0	☒ SPIEN=0	☐ BODLEVEL0=0	☐ BLB02=0
☒ SUT0=0	☐ WDTON=0		☐ BLB01=0
☒ CKSEL3=0	☐ EESAVE=0		☐ LB2=0
☒ CKSEL2=0	☒ BOOTSZ1=0		☐ LB1=0
☐ CKSEL1=0	☒ BOOTSZ0=0		
☒ CKSEL0=0	☐ BOOTRST=0		

Calibration Byte Programming
☒ Disable ☐ Write to Fla ☐ Write to EE Write to Address: 0x|00000000|
Calib.Byte(S) in Device | 89 |
☐ Calib.Byte Val. Set b: | 00 |
☒ Calib.Val source Add | 00 |

Note: Bits CHECKED is Programmed as '0'

Najważniejszym ustawieniem było wyczyszczenie bitu CKDIV8 rejestru Fuse Low. Przy ustawionym bicie wewnętrzny sygnał oscylatora 8MHz jest dzielony przez 8 co skutkuje wynikowym zegarem o taktowaniu 1MHz. Napisane oprogramowanie jest dostosowane pod oscylator o taktowaniu 8MHz dlatego bit CKDIV8 przyjmuje wartość 1 (niezaprogramowany).

Pozostałe bity posiadają wartości domyślne takie jak SUT odpowiadające za czas rozruchu mikrokontrolera po uruchomieniu czy bity CSKEL odpowiadające za wybranie źródła zegara, ponieważ nie było zastosowania w ich dalszej konfiguracji.