Projekt Systemy Mikroprocesorowe

MazeBot

Robot unikający przeszkód z regulacją napędu silników prądu stałego

Maciej Ziaja Bartosz Staszulonek

26 stycznia 2019

Spis treści

1 Wstęp

1.1 Abstrakt

Projekt polega na konstrukcji mobilnego robota, który unika przeszkód. Układ porusza się na dwóch kołach oraz wykrywa elementy otoczenia używając czujnika ultradźwiękowego zamocowanego na wieży serwomechanizmu. Platforma robota napędzana jest za pomocą pary silników prądu stałego, których kąt obrotu jest odczytywany przez czujniki szczelinowe. W celu zwiększenia precyzji działania konstrukcji zbudowano układy regulacji prędkości silników. Wykonano dwa typy układów regulacji: synchronizujący prędkość obrotu obu silników, w celu zachowania kierunku jazdy na wprost, oraz kaskadowy układ regulacji skrętu platformy robota. Układ regulacji skrętu robota przypomina zasadą działania prosty serwomechanizm. Przedstawiono metodę identyfikacji obiektu regulacji i strojenia regulatorów z użyciem linii pierwiastkowych. Na potrzeby projektu wykonano także schemat układu elektronicznego i zrealizowano go w postaci obwodu drukowanego PCB. Zaprojektowano także podwozie robota i wydrukowano je w technologii 3D.

1.2 Cel i zakres projektu

Projekt obejmował budowę platformy bazującej na mikroprocesorze z serii ATmega32, obsługującym peryferia:

- czujnika ultradźwiękowego wykrywającego przeszkody,
- wieży serwomechanizmu, na której zamontowany jest czujnik i która kieruje go w różne strony,
- silników prądu stałego stanowiących napęd platformy,
- czujników szczelinowych odczytujących kąt obrotu kół platformy,
- odbiornika podczerwieni pozwalającego na zdalne wyłączenie robota, jeśli ten się oddali.

Współpraca wymienionych komponentów pozwala robotowi na unikanie przeszkód. Działanie robota powinno polegać na jeździe przed siebie, do czasu napotkania przeszkody. W momencie wykrycia przeszkody robot powinien się zatrzymać, a następnie zbadać czy w jego otoczeniu znajduje się niezagrodzona droga i skierować się w jej stronę. Prototyp układu wykonano z użyciem układu Arduino Leonardo. Na wykonanym prototypie wykonano testy współpracy komponentów i wyciągnięto wnioski dotyczące budowy finalnej konstrukcji oraz wymaganych parametrów komponentów. Działanie prototypu prowadziło do następujących decyzji projektowych, które podyktowały dalszy tok projektu i jego zakres:

- w celu wydajnego wykorzystania przestrzeni na platformie zdecydowano się na wykonanie własnego układu elektronicznego i obwodu drukowanego,
- zaprojektowano własne podwozie dopasowane do używanych peryferiów,
- wykonano układy regulacji jazdy na wprost oraz skręcania platformy.

2 Organizacja projektu

2.1 Harmonogram

Projekt należało zrealizować w przeciągu 4 miesięcy. Proces jego wykonania składał się z dwóch części: testów komponentów wraz z budową prototypu oraz konstrukcji finalnego układu, na podstawie wniosków wyciągniętych z testów konstrukcji prototypowej. Tabela 1 przedstawia ramowy rozkład pracy w czasie.

Termin	Zakres pracy
01.10.2018 - 20.10.2018	Zebranie komponentów układu, ustalenie zakresu prac
20.10.2018 - 01.11.2018	Test komponentów i peryferiów z użyciem Arduino Leonardo
01.11.2018 - 14.11.2018	Budowa prototypu, programowanie głównej logiki programu
14.11.2018 - 01.12.2018	Projekty obwodu drukowanego PCB i podwozia
01.12.2018 - 20.12.2018	Synteza i oprogramowanie układów regulacji
20.12.2018 - 01.01.2019	Złożenie finalnego układu
01.01.2019 - 20.01.2019	Strojenie układów regulacji, budowa dokumentacji

Tabela 1: Harmonogram pracy

2.2 Budżet

Budżet projektu wynosił 150 złotych, najbardziej znaczącym wydatkiem w projekcie było wydrukowanie obwodu elektronicznego. Zaletą projektu jest modułowość konstrukcji podwozia, przez co elementy można łatwo wymieniać w przypadku chęci zastosowania droższych lub tańszych komponentów. Finalny kosztorys projektu przedstawia tabela 2, dokładny spis elementów elektronicznych na obwodzie drukowanym przedstawiono w dokumencie CSV załączonym do plików projektu. Całkowity koszt komponentów projektu wyniósł 149,86 złotych.

Komponent	Koszt	Producent	Dostawca
Obwód drukowany PCB	53,00 PLN	Elpin PCB	Elpin PCB
ATmega32u4	15,91 PLN	Microchip Atmel	TME Electronic Components
Elementy pasywne	∽7,50 PLN	Panasonic, ST Electronics	TME Electronic Components
Elementy aktywne	∽4,00 PLN	ST Electronics	TME Electronic Components
Silniki prądu stałego z kołami i dyskami enkoderów	19,90 PLN	Dagu	Botland
Serwomechanizm	6,90 PLN	Tower Pro	Botland
Ultradźwiękowy czujnik odległości HC-SR04	7,90 PLN	Elec	Botland
Czujniki szczelinowe	9,90 PLN	Vishay	Botland
Odbiornik podczerwieni TSOP4836 - 36khz	2,95 PLN	Vishay	Botland
Bateria Li-Pol	21,90 PLN	Dualsky	Botland

Tabela 2: Kosztorys, rozdysponowanie budżetu

3 Budowa prototypu, analiza problemów

Realizację projektu zaczęto od zgromadzenia peryferiów koniecznych do realizacji konstrukcji robota. W pierwszej fazie prototypowania komponenty testowano osobno używając Arduino Leonardo. Zapoznano się z ich praktycznymi możliwościami i działaniem dostępnych bibliotek.

Następnie połączono wszystkie komponenty i przeprowadzono ich jednoczesne testy oraz zmierzono zasilanie używając multimetru wraz z zasilaczem regulowanym. Elementy prototypowe umieszczono na platformie testowej i oceniono działanie konstrukcji. Okazało się, że działanie robota jest możliwe, ale sterowanie silnikami w torze otwartym nie było wystarczająco precyzyjne aby unikać przeszkód. Przy symetrycznym wysterowaniu silników robot nie poruszał się w kierunku prostym, różnice w działaniu poszczególnych silników są zbyt duże by pracowały one symetrycznie co jest wymagane by platforma jechała na wprost. Skręcanie platformy o zadany kąt również okazało się nieprecyzyjne, obroty o na przykład 90 stopni w prawo były mało powtarzalne i obarczone dużym błędem. Zdecydowano się na syntezę układów regulacji, aby wyeliminować te problemy.

Testy platformy prototypowej pokazały także, że złożoność układu jest zbyt duża aby budować

go na płytce stykowej/prototypowej. Dodatkowo serwomechanizm, który jest bardzo wrażliwy na zakłócenia zasilania, działał źle gdy był połączony przez wiele kabli, które razem tworzyły dużą rezystancję. Obserwacje te skłoniły nas do zaprojektowania własnego obwodu drukowanego, który mieściłby wszystkie potrzebne komponenty elektroniczne na małej przestrzeni. Konstrukcja mechaniczna platformy prototypowej okazała się mało solidna i nie była dostosowana do używanych przez nas modułów elektronicznych. W celu zwiększenia wytrzymałości i schludności wykonania układu zdecydowano się na projekt własnego podwozia robota.

4 Projekt układu elektronicznego

4.1 Działanie i uzasadnienie doboru elementów elektronicznych

4.1.1 Mikroprocesor ATmega32u4

Centralnym elementem robota jest mikroprocesor AVR o architekturze RISC. Ośmiobitowe mikroprocesory ATmega są popularnym wyborem przy konstrukcji układów wbudowanych w przemyśle i urządzeniach elektronicznych o niedużym stopniu wymaganej wydajności. Ich zaletą jest powszechność oraz fakt, że na procesorach z tej rodziny bazuje popularna platforma Arduino, w ramach której dostępny jest szeroki zakres bibliotek programistycznych do obsługi peryferiów.
Przy wyborze konkretnego modelu mikroprocesora kierowaliśmy się następującymi wymaganiami:

- obsługa czterech przerwań zewnętrznych:
  1. pochodzących z enkodera lewego koła,
  2. pochodzących z enkodera prawego koła,
  3. pochodzących z czujnika ultradźwiękowego,
  4. pochodzących z odbiornika podczerwieni,
- obsługa trzech wyjść PWM, za pomocą liczników w celu:
  1. sterowania prędkością lewego silnika prądu stałego,
  2. sterowania prędkością prawego silnika prądu stałego,
  3. sterowania położeniem serwomechanizmu, na którym zamontowany jest czujnik ultradźwiękowy,
- możliwość użycia sumarycznie dwunastu wejść wyjść cyfrowych do komunikacji z peryferiami,
- pożądana jest wbudowana obsługa komunikacji szeregowej za pomocą interfejsu USB w celu łatwego debugowania układu i zdjęcia pomiarów dynamiki układu przy identyfikacji obiektu regulacji i strojeniu regulatorów,
- powyższe wymagania powinny być spełnione przy użyciu bootloadera i bibliotek Arduino.

Powyższe wymagania spełnia mikroprocesor ATmega32u4 firmy Atmel, co można wywnioskować z jego dokumentacji [6]. Na tym procesorze bazuje układ Arduino Leonardo, który wykorzystano przy konstrukcji prototypu. Mikroprocesor ATmega32u4 z użyciem bibliotek Arduino pozwala na używanie do pięciu przerwań zewnętrznych oraz siedmiu kanałów generujących sygnały PWM.

Istotne jest, że kanały PWM są obsługiwane przez cztery oddzielne liczniki. Biblioteki Arduino (szczególnie obsługi serwomechanizmu) wchodzą łatwo w konflikty i korzystają z liczników w sposób uniemożliwiający ich współdzielenie przy generowaniu sygnałów PWM. Dlatego przy doborze mikroprocesora i rozdysponowaniu jego zasobów starano się, aby serwomechanizm był obsługiwany przez oddzielny licznik, który nie jest dzielony z innymi funkcjami układu. Oznacza to, że pin sterujący serwem nie powinien generować sygnałów z użyciem tego samego licznika, co inne kanały PWM, aby uniknąć konfliktów bibliotek Arduino.

Wybrany model mikroprocesora jest dostępny tylko w obudowach powierzchniowych QFP (*Quad*

Flat Package), przy wyborze tej jednostki należy pamiętać, że wymaga ona umiejętności precyzyjnego lutowania. Procesor może pracować przy zasilaniu 5 V, jest to dogodna wartość ponieważ reszta peryferiów w układzie ma takie same napięcie zasilania.

4.1.2 Silniki prądu stałego

Ze względu na ograniczenia budżetowe zdecydowano się na jedne z najtańszych dostępnych silników, wyprodukowane przez firmę Dagu. Silniki prądu stałego oferują największe momenty siły przy małych obrotach, nadają się dobrze do napędzania i przyspieszania platform robotów, oferują możliwość płynnego sterowania. Ich wadą jest fakt, że bez dodatkowego układu czujników nie można określić ich kąta obrotu. Należy mieć na uwadze, że ze względu na obecność komutatora i szczotek silniki prądu stałego zużywają się w czasie pracy. Parametry wybranych przez nas silników są następujące:

- maksymalne napięcie zasilania: 6 V,
- moment obrotowy: $0.8\,\mathrm{kg}\cdot\mathrm{cm}$, $0.78\,\mathrm{N}\cdot\mathrm{m}$,
- obroty silnika bez obciążenia: $(90 \pm 10)\ \frac{obr}{min}$,
- pobór prądu silnika bez obciążenia: 190 mA (max. 250 mA),
- pobór prądu silnika przy zatrzymanym wale: 1 A.

Do silników przymocowano koła o parametrach:

- średnicy: 65 mm,
- szerokości: 30 mm.

4.1.3 Sterownik silników prądu stałego - mostek H

Najbardziej popularnym sposobem sterowania silnikami prądu stałego jest użycie mostka H. Jest to układ składający się z czterech tranzystorów polowych, których dreny i źrodła są odpowiednio połączone ze sterowanym silnikiem. Para tranzystorów połączona z tą samą szczotką silnika ma złączone bramki, ale przeciwny stan otwarcia. Złączone bramki są wyprowadzone z układu i stanowią wejścia sterowania. W zależności od tego na które z wejść podamy stan wysoki przez jeden z tranzystorów z każdej pary będzie płynął prąd, powodując obroty silnika w odpowiednim kierunku. Układem elektronicznym, który zawiera w sobie dwa mostki H, co pozwala na sterowanie parą silników, jest L298N. Dodatkowo układ posiada dodatkowe tranzystory do sterowania prędkością silników za pomocą sygnału PWM, który stopniowo otwiera i domyka tranzystor polowy przepuszczający sygnały wejściowe. Parametry sterownika są następujące:

- maksymalne napięcie zasilania silników: 45 V,
- napięcie części logicznej: od 4,5 V do 7 V,
- maksymalny prąd zasilający silnik: 2 A.

Sterownik dostępny jest w obudowie przewlekanej *Multiwatt15*, jego producentem jest firma STMicroelectronics. Porównując parametry mostka, zawarte w dokumentacji [7] z parametrami silników możemy stwierdzić, że jest on wystarczający do ich wysterowania.

4.1.4 Czujnik ultradźwiękowy

Najbardziej popularnym i łatwo dostępnym miernikiem odległości jest czujnik ultradźwiękowy HC-SR04. Czujniki ultradźwiękowe mierzą odległość na podstawie czasu, który jest potrzebny fali ultradźwiękowej na dotarcie do przeszkody i powrót do czujnika po jej odbiciu. Pomiar czujnikiem rozpoczyna się od podania na wejście TRIG stanu wysokiego przez 10 µs. Powoduje to wygenerowanie przez czujnik ciągu ośmiu sygnałów ultradźwiękowych o częstotliwości 40 kHz. Po odbiciu sygnału od przeszkody i jego powrocie do czujnika odległość można obliczyć według wzoru 1, gdzie:

- d to odległość mierzona,
- t_h to czas trwania stanu wysokiego,
- v_s to prędkość rozchodzenia się fali dźwiękowej w powietrzu, typowo 340 $\mathrm{m\,s^{-1}}$.

$$d = \frac{t_h \cdot v_s}{2} \tag{1}$$

Model czujnika HC-SR04, zgodnie z dokumentacją [8] charakteryzuje się następującymi parametrami:

- napięcie zasilania: 5 V,
- średni pobór prądu: 15 mA,
- zakres pomiarowy: od 2 cm do 200 cm,
- częstotliwość pracy: 40 kHz,
- wymiary: $45 \times 20 \times 15$ mm.

Zasilanie czujnika i jego zakres pracy są odpowiednie dla projektu. Przeszkody, które chcemy wykrywać będą znajdywały się w odległości parunastu centymetrów od robota, czyli w zakresie pracy czujnika. Należy mieć na uwadze, że czujniki ultradźwiękowe wykrywają najlepiej duże przeszkody. Dodatkowo istotne jest, że pomiar czujnika ultradźwiękowego trwa w czasie, przebycie odpowiedniej drogi przez falę ultradźwiękową nie jest natychmiastowe. Kwestia ta zostanie wzięta pod uwagę w czasie tworzenia oprogramowania.

4.1.5 Serwomechanizm

Aby umożliwić robotowi kierowanie czujnika odległości w różnych kierunkach, został on zamieszczono na wieży serwomechanizmu. Ponieważ wymagamy aby czujnik mógł być skierowany w lewo, prawo oraz na wprost wystarczające jest serwo 180 stopni. Waga czujnika ultradźwiękowego jest znikoma, dlatego serwomechanizm można wybrać kierując się jak najniższą ceną.

4.1.6 Odbiornik podczerwieni TSOP4836

Robota wyposażono w odbiornik podczerwieni ułatwiający jego zatrzymanie. W celu łatwej współpracy z domowymi pilotami na podczerwień należało wybrać odbiornik wspierający popularne formaty: NEC Code, Toshiba Micom Format, Sharp Code, RC5 Code, RC6 Code, R–2000 Code. . . .

Wybraliśmy odbiornik TSOP4836 pracujący z falami o częstotliwości 36 kHz. Zgodnie z dokumentacją [9] firmy Vishay odbiornika należy go zabezpieczyć kondensatorem filtrującym i rezystorem. Parametry odbiornika są następujące:

- napięcie zasilania: od 4,5 V do 5,5 V,
- średni pobór prądu: 1,5 mA.

Przy projektowaniu obwodu drukowanego i lutowaniu odbiernika należy zwrócić uwagę na jego wyprowadzenia. Wiele czujników podczerwieni o identycznych obudowach i podobnych parametrach ma bardzo różne kolejności wyprowadzeń.

4.1.7 Stabilizator napięcia L7805CV

W konstruowanym projekcie wszystkie układy logiczne zasilane są napięciem 5 V, natomiast silniki zasilane są napięciem 7,4 V. Wynika stąd, że najlepiej zastosować baterię siedmiowoltową, a zasilanie części logicznej dostarczać poprzez regulator napięcia. Najbardziej popularnym regulatorem jest układ 7805, produkowany przez firmę STMicroelectronics [10]. W wersji CV jego parametry są następujące:

- maksymalne napięcie wejściowe: 35 V35 V,

- napięcie wyjściowe: 5 V (z dokładnością 2 %),
- maksymalny prąd wyjściowy: 1,5 A.

Stabilizator jest dostępny w różnych obudowach, przewlekanych i powierzchniowych. Ze względu na łatwość montażu, dostępność i możliwość zamontowania radiatora wybraliśmy model w obudowie przewlekanej TO-220. Po porównaniu parametrów stabilizatora z parametrami komponentów, które zasili można stwierdzić, że jest on wystarczający by spełnić zapotrzebowania projektu.

4.1.8 Akumulator

Testy pokazały, że przy zasilaniu 7,4 V układ pobiera prąd do 0,3 A, co dostarcza nam informacji potrzebnych do doboru baterii dla robota. Zdecydowano się na zastosowanie akumulatora litowo-polimerowego firmy Dualsky. Jego najważniejsze parametry są następujące:

- akumulator składa się z dwóch ogniw Li-pol,
- napięcie nominalne: 7,4 V,
- pojemność: 800 mAh,
- prąd rozładowania ciągły: 20 A,
- wbudowane przewody ze wtykiem JST.

4.1.9 Elementy pasywne

Użyte komponenty elektroniczne wymagają użycia kondensatorów filtrujących, zastosowano kondensatory o pojemnościach podanych w dokumentacjach filtrowanych elementów. Kondensatory ceramiczne są umieszczone w obudowach przewlekanych, starano się dobrać kondensatory elektrolityczne w obudowach powierzchniowych, aby zredukować miejsce, którą zajmują. Spis wszystkich elementów układu, w tym elementów pasywnych znajduje się w osobnym pliku `csv` dostępnym na stronie projektu.

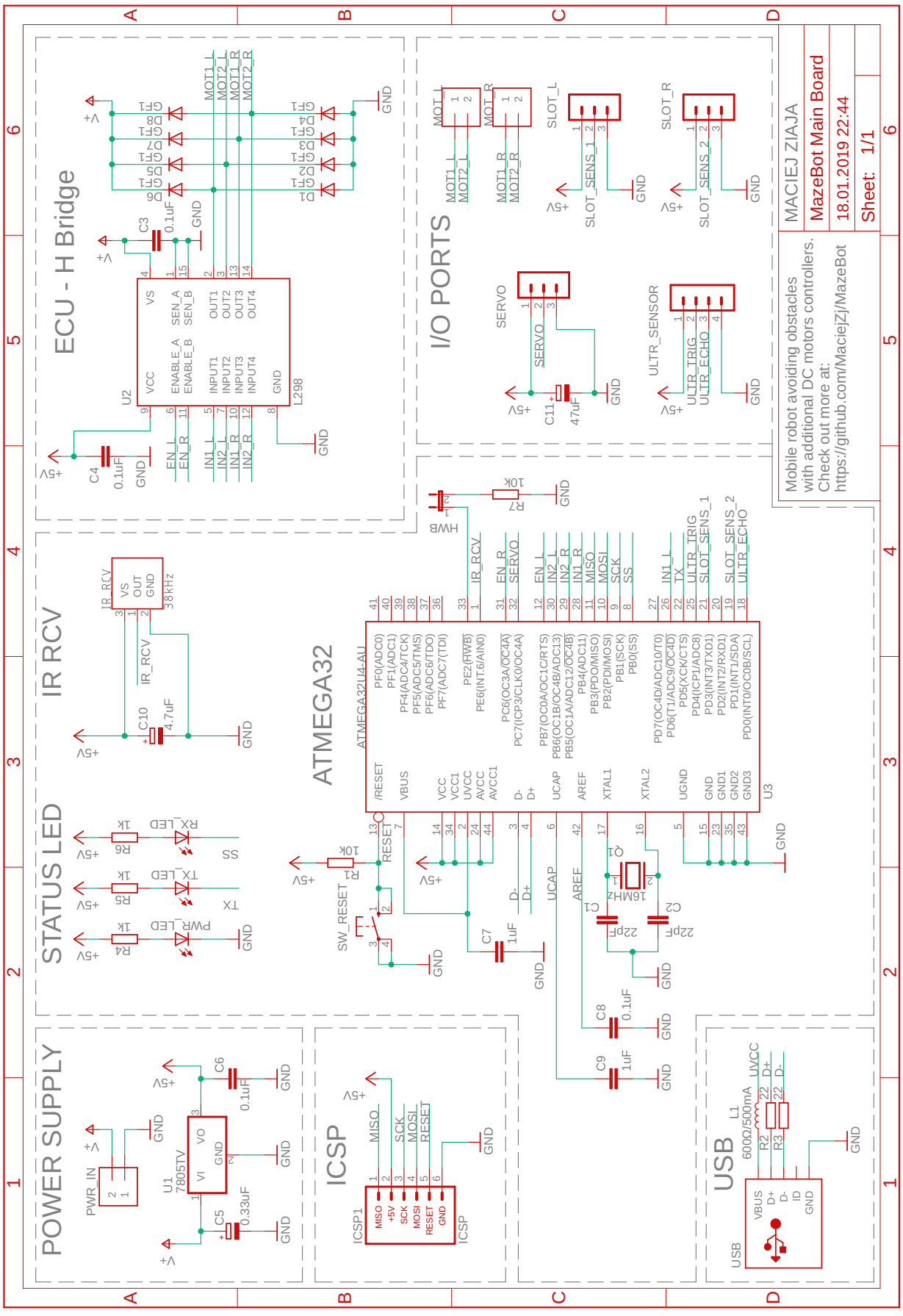
4.2 Schemat układu elektronicznego

Na rysunku 1 przedstawiono utworzony schemat układu elektronicznego. Projekt układu wykonano w programie Autodesk Eagle[1], korzystając z bibliotek komponentów.

Centralnym elementem układu jest mikroprocesor ATmega32u4 oznaczony symbolem *U3*, znajduje się on w polu *C3* schematu. Źródłem taktowania mikroprocesora jest zewnętrzny rezonator kwarcowy 16 MHz. Do jego poprawnego działania konieczne są odpowiednie kondensatory. Aby osiągnąć zamierzone taktowanie użyto kondensatorów o pojemności 22 pF. Mikroprocesor zasilono napięciem 5 V, które jest filtrowane kondensatorami 0,1 µF, zasilanie portu USB jest filtrowane kondensatorami 1 µF. Aby ułatwić restartowanie układu, na schemacie umieszczono przycisk, który umożliwia zwieranie pinu reset z ziemią. Jeżeli przycisk jest przytrzymany dostatecznie długo układ mikroprocesorowy zostanie zresetowany. Aby podczas resetu nie doszło do zwarcia obwodu zasilania, dodano rezystor *R1*. Układ resetu znajduje się w polu *B2* schematu.

W polu *A2* schematu umieszczono diody sygnalizacyjne. Dioda *PWR_LED* sygnalizuje obecność napięcia na wyjściu stabilizatora. Diody *TX_LED* oraz *RX_LED* są związane z transmisją szeregową, podłączono je tak że są aktywne kiedy wyjścia procesora są w stanie niskim. Dzięki temu diody świecą kiedy transmisja szeregowa nie ma miejsca i sygnalizują tym samym poprawne działanie procesora. Jest to wygodne ponieważ dioda *PWR_LED* sygnalizuje jedynie obecność zasilania układu, ale nie informuje użytkownika o właściwym uruchomieniu mikroprocesora. Każda z diod podłączona jest przez odpowiedni rezystor ograniczający płynący prąd. W polu *C1* znajduje się port USB w wersji MICRO. Linie portu szeregowego zabezpieczono dławikiem ferrytowym, oraz rezystorami. Do programowania mikroprocesora zastosowano złącze ICSP sześciopinowe, znajdujące się w polu *D1* schematu.

[1] Autodesk Eagle: `https://www.autodesk.com/products/fusion-360/overview`



Rysunek 1: Schemat układu elektronicznego

W polu *A5* schematu znajduje się mostek L298. Aby zabezpieczyć układ do jego wyjść silnikowych dodano diody prostownicze skierowane przeciwnie do polaryzacji baterii. W czasie pracy silników na ich uzwojeniu gromadzi się energia w postaci pola magnetycznego. Po zatrzymaniu silników i hamowaniu platformy energia to wróci do układu w postaci płynącego prądu. Dzięki obecności diod układ mostka jest zabezpieczony przez tymi prądami, popłyną one przez diody do baterii. Co więcej, ze względu na polaryzację diod, energia odzyskiwana z silników będzie ładować baterię, tworząc prosty układ aktywnego hamowania. W sekcji IO PORTS znajdują się złącza peryferiów układu.

Komponenty z fabrycznymi wyprowadzeniami przez goldpiny, połączono takimi samymi złączami od strony układu. Silniki i zasilanie są złączone przez konektory ARC dokręcane śrubokrętem.

4.3 Projekt układu drukowanego

4.3.1 Parametry techniczne układu drukowanego

Ze względu na ograniczoną przestrzeń na powierzchni platformy robota zdecydowano się na wykonanie własnego obwodu drukowanego w technologii dwustronnej, z metalizowanymi otworami i soldermaską. W celu zachowania zgodności z procesem technologicznym producenta płytek[2] przyjęto następujące ustawienia DRC (*Design rules check*):

- grubość laminatu wraz ze ścieżkami: 1,5 mm,
- grubość ścieżek: 0.035 mm,
- odległości między ścieżkami, przelotkami (*via*) oraz padami: 6 mil,
- odległości ścieżek od krawędzi płytki: 40mil,
- minimalna odległość otworów: 6 mil,
- minimalna szerokość ścieżki: 6 mil,,
- minimalna średnica otworu: 0,35mm,
- szerokość termoizolacji padów od wylewek (ułatwia lutowanie): 10mil.

Dwuwymiarowe wizualizacje obwodu drukowanego wykonanego w programie Autodesk Eagle przedstawiają rysunki 2 oraz 3. Przy tworzeniu płytki zastosowano technikę wylewki masy, czyli tworzenia dużych płaszczyzn przewodnika, zamiast prowadzenia poszczególnych ścieżek. Ułatwia to rozmieszczenie połączeń i zmniejsza rezystancję ścieżek masy. Zastosowano materiały zawierające ołów jako bardziej wytrzymałe i łatwiejsze w lutowaniu. W amatorskim lutowaniu nie używa się temperatur przy których wydzielają się trujące opary ołowiu.

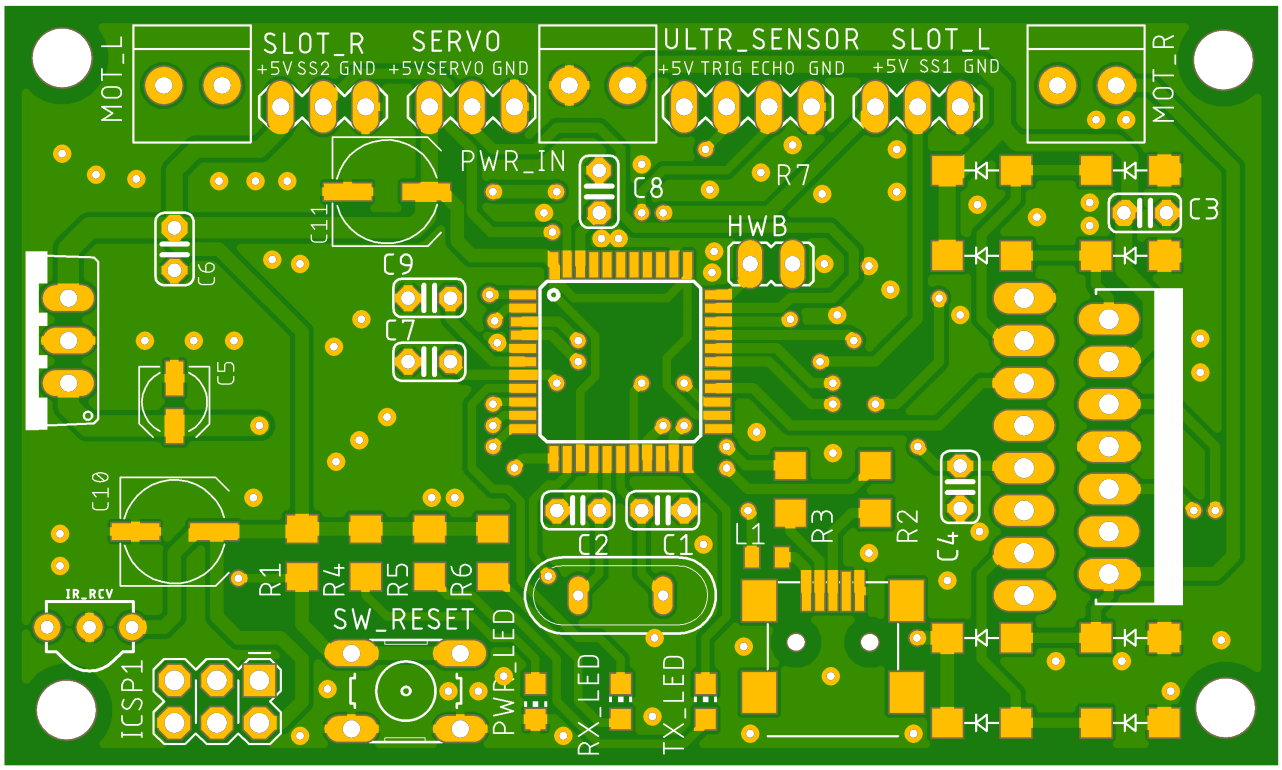
4.3.2 Rozmieszczenie elementów na powierzchni układu drukowanego

W centralnej części obwodu drukowanego umieszczono mikroprocesor. Rozdysponowanie wyprowadzeń sygnałowych mikroprocesora jest kompromisem między ograniczeniami bibliotek Arduino, możliwościami procesora oraz ilością miejsca dostępnego na płytce. Górną krawędź układu zajmują konektory peryferiów, dolna krawędź została przeznaczona na elementy służące interakcji z użytkownikiem. Płytka została wykonana w technologii mieszanej[3], w celu zaoszczędzenia miejsca starano się używać elementów montowanych powierzchniowo, tam gdzie było to możliwe i wygodne. Podczas prowadzenia ścieżek układu należy mieć na uwadze ich szerokość. Znając parametry techniczne płytki oraz prądy płynące przez poszczególne ścieżki można obliczyć ich minimalną szerokość za pomocą kalkulatorów dostępnych online[4]. Obudowę rezonatora kwarcowego oddzielono od powierzchni obwodu wyciętą gumą. Na obudowach elementów aktywnych umieszczono radiatory.

[2]Elpin: `https://www.elpinpcb.com.pl/prototypy/`

[3]To znaczy użyto zarówno elementów przewlekanych jak i montowanych powierzchniowo

[4]Kalkulator scieżek *4PCB* `https://www.4pcb.com/trace-width-calculator.html`



Rysunek 2: Górna warstwa obwodu drukowanego

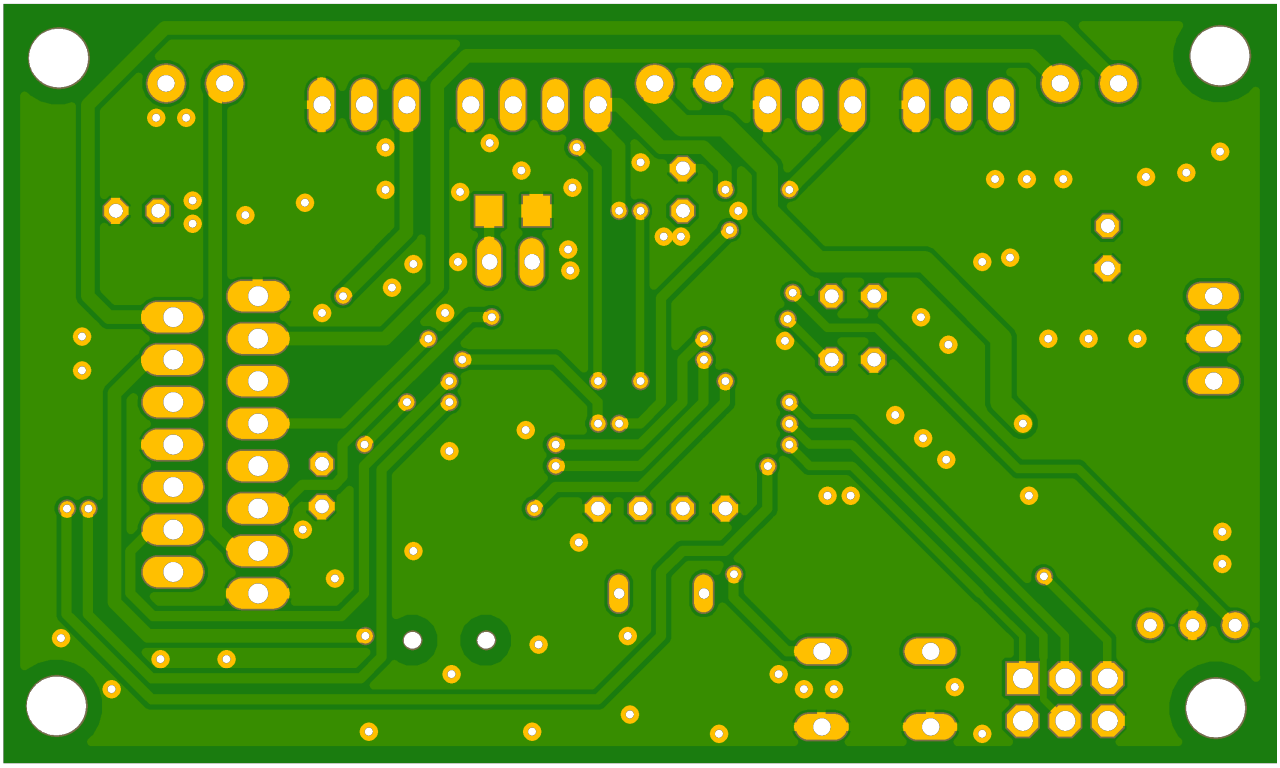
5 Projekt podwozia robota

W celu zwiększenia trwałości i solidności budowy podwozia zdecydowano się zaprojektować własne oraz wydrukować je w technologii 3D. Do jego modelowania użyto programu Autodesk Fusion 360[5]. Podwozie zaprojektowano tak aby osiągnąć jego modułowość. Platforma robota składa się z osobnych części:

- głównego podwozia z uchwytami na silniki,
- dwóch obudów na czujniki szczelinowe,
- uchwytu na serwomechanizm,
- elementu mocującego czujnik ultradźwiękowy na szczycie serwomechanizmu,
- obudowy baterii.

Obudowy komponentów są łączone z podwoziem za pomocą śrub. Jeżeli zaistniałaby potrzeba wymiany danego komponentu, na przykład serwomechanizmu, wystarczy wydrukować nową obudowę jedynie na wymieniane peryferium. Przy projekcie uwzględniono utworzenie otworów oraz uchwytów na kable. Akumulator litowo-polimerowy jest połączony z obwodem przez przełącznik dwupozycyjny, co pozwala na wyłączenie zasilania. Podwozie wydrukowano w technologii 3D za pomocą posiadanej drukarki, dzięki czemu koszta jego utworzenia są znikome.

[5]Autodesk Fusion 360: `https://www.autodesk.com/products/fusion-360/overview`



Rysunek 3: Dolna warstwa obwodu drukowanego

6 Synteza układów regulacji

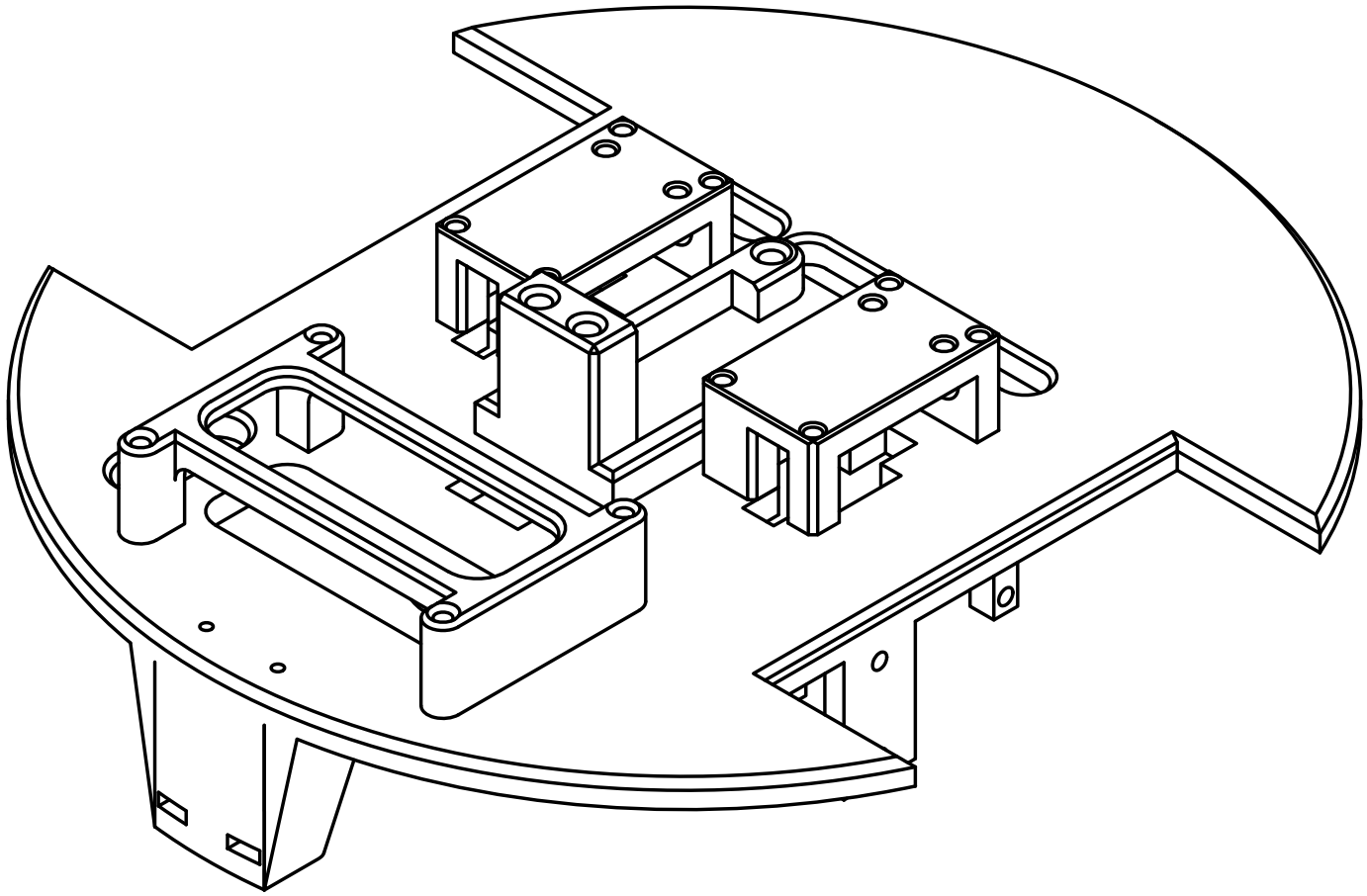
Budowa i testy układu prototypowego wykazały, że bez zastosowania dodatkowych środków układ porusza się w sposób nieprecyzyjny. Zdecydowano się na syntezę układów regulacji w celu poprawienia precyzji działania robota.

6.1 Regulator synchronizacji prędkości silników

Bez zastosowania układu regulacji robot nie porusza się prosto przy symetrycznym wysterowaniu lewego i prawego silnika, to znaczy że platforma skręca kiedy nie jest to pożądane. Problem ten jest spowodowany różnicami w wykonaniu poszczególnych silników, niesymetrycznym obciążeniem platformy oraz niedoskonałościami jej konstrukcji. To zagadnienie można próbować rozwiązać następującymi sposobami:

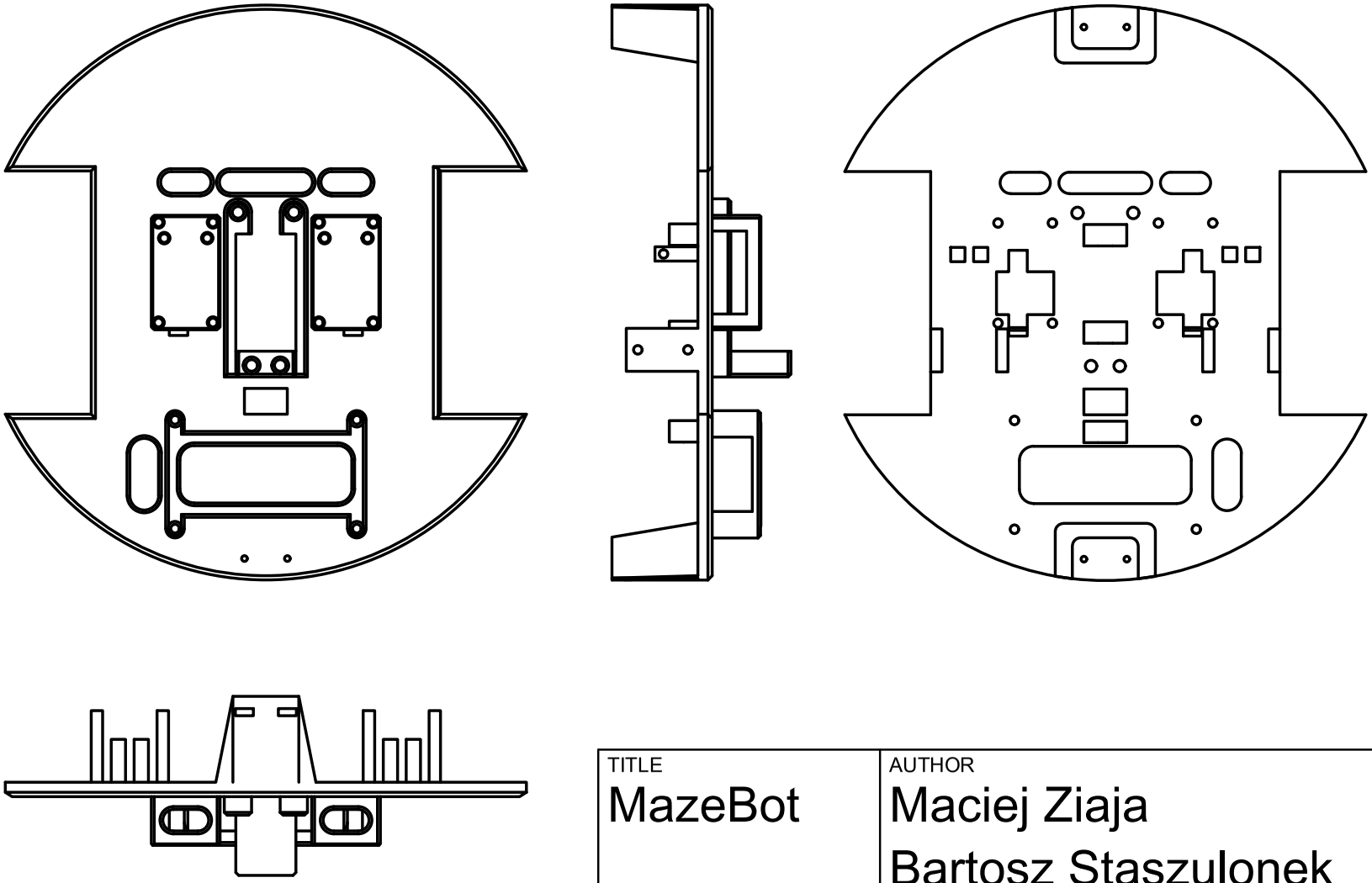
- wyważaniem platformy robota odpowiednimi odważnikami, jest to najgorsze rozwiązanie, nie dające pewnych wyników, jest też zależne od stopnia naładowania baterii, który wpływa na silniki niesymetrycznie,
- sterowaniem PWM w otwartym torze, tak aby uwzględnić niesymetryczność silników, sterowanie należy ustalić drogą eksperymentalną, ta metoda również jest podatna na rozładowanie baterii oraz zmianę obciążenia platformy i inne zakłócenia,
- konstrukcją regulatorów, które na podstawie odczytów z enkoderów szczelinowych synchronizują prędkość silników.

Ze względu na największą pewność efektów, niskie koszty oraz wartość edukacyjną zdecydowano się na rozwiązanie wykorzystujące układy regulacji.



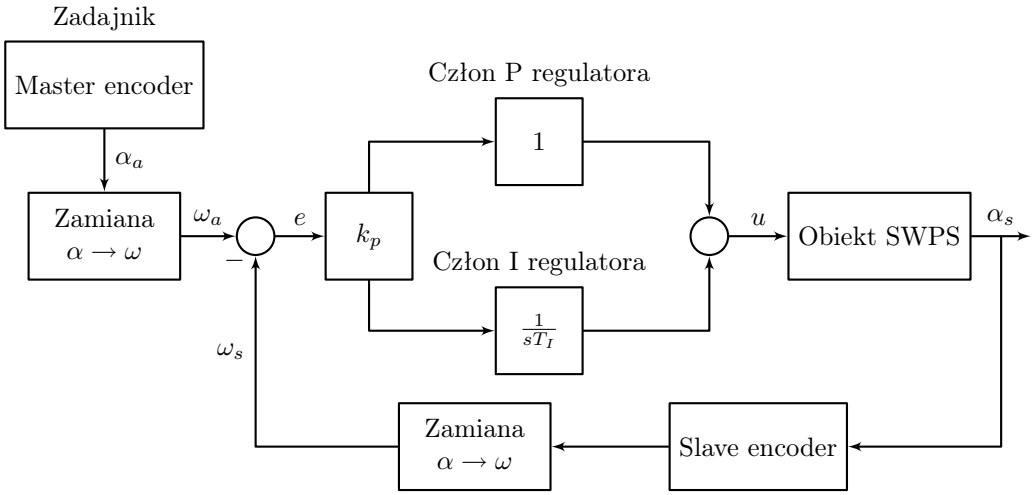
TITLE	AUTHOR	
MazeBot	Maciej Ziaja Bartosz Staszulonek	
REV 1.0	DATE 20.12.18	SHEET 1/2

Rysunek 4: Projekt podwozia robota, rzut izometryczny



Rysunek 5: Projekt podwozia robota, rzuty

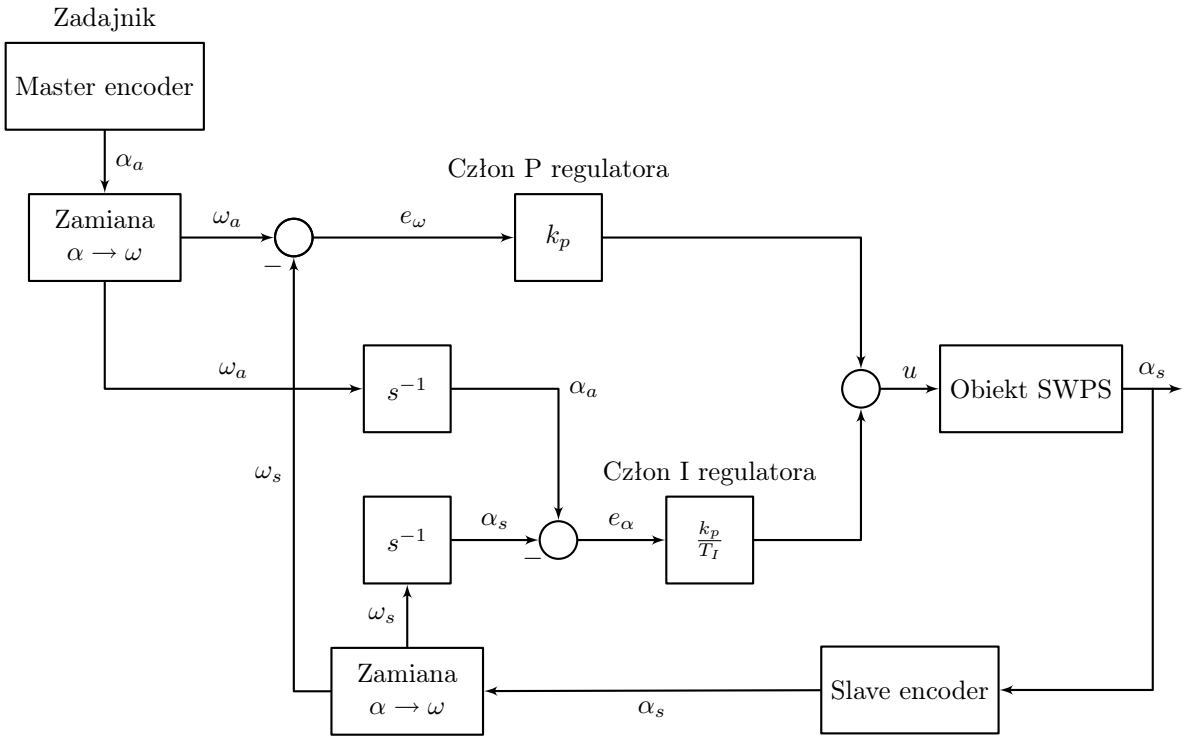
Syntezowany układ będzie miał na celu regulację prędkości silnika *slave* na podstawie odczytu prędkości silnika *master*. Dodatkowo silnik *master* będzie regulowany ze stałą wartością zadaną prędkości w celu zapewnienia większej pewności jego pracy. Regulator prędkości silnikam *slave* powinien być typu PID lub PI. Obecność członu całkującego jest wymagana aby osiągnąć odpowiedni astatyzm układu. Dzięki całkowaniu układ wyrówna nie tylko chwilową prędkość platformy, ale również sumaryczną przebytą przez robota drogę, co jest kluczowe aby utrzymać poprawny kierunek jazdy. Dodatkowo obliczenia związane z członem całkującym są w naszym przypadku proste, ponieważ odczyt z enkoderów charakter drogi przebytej przez robota, czyli całki z prędkości. Regulator prędkości silnika *master* może być typu proporcjonalnego. Zwiększanie astatyzmu tego silnika nie ma znaczenia dla jego poruszania się po lini prostej. W regulatorach nie zastosowano członu różniczkującego, który nie jest konieczny, a utrudniałby konstrukcję układów i sprawiał wyzwania obliczeniowe. Schemat 6 obrazuje konstrukcję planowanego układu regulacji prędkości silników. Układem zadajnika dla regulatora *slave* jest pomiar



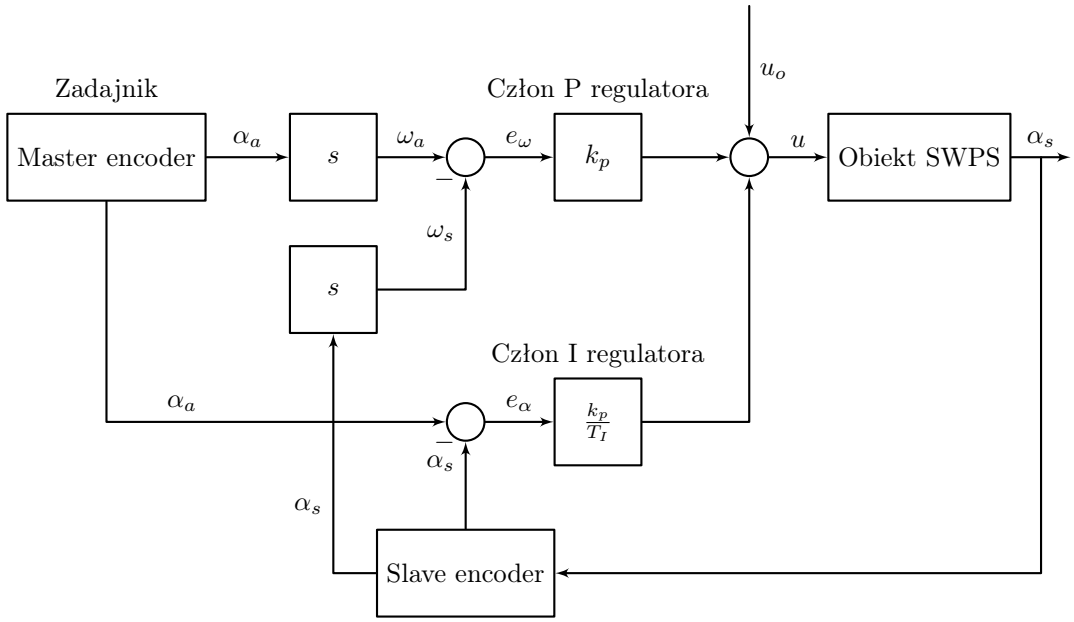
Rysunek 6: Wyjściowy schemat blokowy układu regulacji

pracy silnika *master*. Układ *slave* objęty jest pętlą sprzężenia zwrotnego z enkoderem szczelinowym w torze pomiarowym. Za regulację prędkości odpowiada układ typu PI Ilość zliczonych szczelin oznaczono na schemacie jako α, prędkość silników (ilość szczelin w czasie) oznaczono jako ω, silnik *master* oznaczono indeksem m, a silnik *slave* indeksem s. Jest to wyjściowy schemat regulatora, który nie jest dopasowany do charakteru toru pomiarowego i implementacji programistycznej.

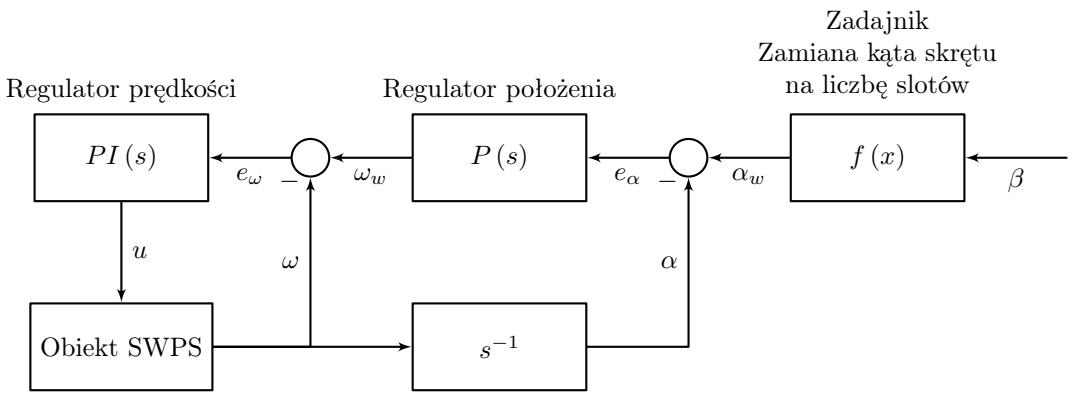
Układ należy przekształcić w celu jego łatwiejszej implementacji. Czujniki szczelinowe zwracają do układu pomiar liczby szczelin na tarczy enkodera, co jest ekwiwalentem całki z prędkości, dlatego będziemy dążyć do usunięcia operacji całkowania z układu i bloku *zamiana* α *na* ω. W pierwszym kroku przekształceń układu skorzystamy z rozłączności operacji całkowania i przeniesiemy to działanie przed regulator oraz obliczymy całkę sygnału i wartości zadanej osobno, a uchyb wyznaczymy z różnicy tych całek. Układ po tym przekształceniu obrazuje schemat 7, który jeszcze jedną modyfikację. W następnym kroku można zauważyć, że blok *zamiana* α *na* ω jest operacją różniczkowania, czyli mnożenia przez operator s. Wynika z tego że możemy skrócić bloki całkowania $\frac{1}{s}$ i różniczkowania s. Finalny projekt układu regulacji przedstawia schemat 8. Tarcze enkoderów wprowadzają do układu dyskretyzację oraz błąd kwantyzacji. Silnik *slave*, który nadąża za silnikiem *master* zacznie się obracać po nim, a obrót od startu układu do pierwszej szczeliny silnika *master* sprawia, że w układzie pojawia się błąd. Błąd ten jest zależny od odległości punktu pomiarowego enkodera do pierwszej zliczonej szczeliny. Pozycja punktu pomiaru w momencie startu robota jest losowa, podobnie błąd z tego wynikający. Aby zmniejszyć ten problem do układu zdecydowano się dodać element sterowania w torze otwartym narzucający punkt pracy regulatorów. Dzięki temu w momencie startu oba silniki są wysterowane do jazdy na wprost. Wartości punktu pracy dla obu silników należy wyznaczyć eksperymentalnie, są one zależne od niesymetryczności charakterystyk silników.



Rysunek 7: Schemat blokowy układu regulacji trakcie przekształceń



Rysunek 8: Finalna konstrukcja układu regulacji



Rysunek 9: Schemat blokowy kaskadowego układu regulacji

6.2 Kaskadowy regulator skrętu platformy

Kolejnym problemem jest zapewnienie powtarzalności i precyzji obrotów platformy robota. W tym celu należy zbudować regulator położenia platformy, którego wartością zadaną jest ilość szczelin zliczona przez enkodery. Ilość szczelin potrzebna do obrotu platformy o zadany kąt jest zależna od konstrukcji mechanicznej robota, w sposób który wyraża wzór 2, gdzie:

- α to liczba slotów potrzebnych do obrotu o zadany kąt,
- β to zadany kąt obrotu całej platformy robota,
- α_{disk} to liczba szczelin na tarczy enkodera,
- $wheelbase$ to rozstaw osi podwozia,
- d_w to średnica kół.

$$\alpha = round\left(\frac{\beta}{360^\circ} \cdot 2 \cdot \alpha_{disk} \cdot \frac{wheelbase}{d_w}\right) \tag{2}$$

Wzór wynika z proporcji obrotu koła do obrotu całej platformy robota wokół osi przebiegającej przez jej środek. We wzorze można zauważyć podwajanie liczby szczelin, wynika to z wyzwalania przerwania zliczania szczelin zarówno na zboczu rosnącym jak i malejącym w celu zwiększenia rozdzielczości pomiaru.

Jeżeli w układzie zastosujemy regulator proporcjonalny z pojedynczą pętlą sprzężenia zwrotnego napotkamy na problem szczególnie widoczny przy obrotach o mały kąt przy dużych oporach ruchu. Kiedy taki regulator wyznaczy wysterowanie silników, na podstawie uchybu ilości zliczonych szczelin, może się okazać że jest ono zbyt małe, aby poruszyć podwoziem silnika. Dodane do regulatora członu całkującego spowoduje, że po czasie kiedy wartość uchybu całki będzie odpowiednio duża układ ruszy z miejsca. Musimy jednak pamiętać, że układ regulacji położenia wpływa na obiekt o charakterze całkującym (bo ilość zliczonych slotów jest ekwiwalentem drogi). Regulacja z członem całkującym obiektów całkujących prowadzi do utworzenia niestabilnych układów regulacji. Rozwiązaniem tego problemu jest zastosowanie regulatora o strukturze kaskadowej, widocznego na rysunku 9. Regulator taki składa się z dwóch pętli sprzężenia, zewnętrznej i wewnętrznej oraz towarzyszących im regulatorów: odpowiednio położenia i prędkości. Regulator położenia ustala prędkość zadaną potrzebną do wykonania obrotu na podstawie uchybu liczby szczelin. Regulator prędkości otrzymuje swoją wartość zadaną od regulatora położenia. Regulator wewnętrzny, prędkości może zawierać człon całkujący, ponieważ nie reguluje samodzielnie układu całkującego. Ponieważ pokonanie oporu ruchów przy starcie jest trudniejsze od kontynuowania rozpoczętego obrotu[5] to zdecydowano się na odwijanie całkowania (zerowanie) przy każdej zliczonej szczelinie. Mogłoby to spowodować niepłynny ruch robota, natomiast sprawdzono eksperymentalnie, że obrane nastawy są wystarczające, aby człon proporcjonalny powodował płynny ruch obracającej się platformy, przy częstym zerowaniu członu całkującego. Powoduje to też łagodne zatrzymanie robota, co sprzyja eliminowaniu poślizgów i minimalizuje wybieg silnika i przeregulowania.

6.2.1 Wyznaczenie przedziału błędów w układzie.

Układ nie jest wolny od błędów związanych z jego konstrukcją mechaniczną i implementacją rozwiązań. Tarcza enkodera ze skończoną liczbą szczelin powoduje dyskretyzację działania układu i powstanie opóźnienia w synchronizacji silników *master* oraz *slave*. Ponieważ błąd ten ma charakter błędu kwantyzacji można go łatwo oszacować. Zgodnie z [4] błąd taki ma rozkład równomierny, jego skrajne wartości można wyliczyć według wzoru 3 gdzie:

- α_{disk} to liczba szczelin na tarczy enkodera,
- liczbę szczelin podwojono, ponieważ przerwanie wyzwalane jest na obu zboczach w celu zwiększenia rozdzielczości enkodera.

$$\Delta\alpha_{max} = \pm\frac{360^\circ}{\alpha_{disk}\cdot 2}\cdot\frac{1}{2} = \pm 4.5^\circ \tag{3}$$

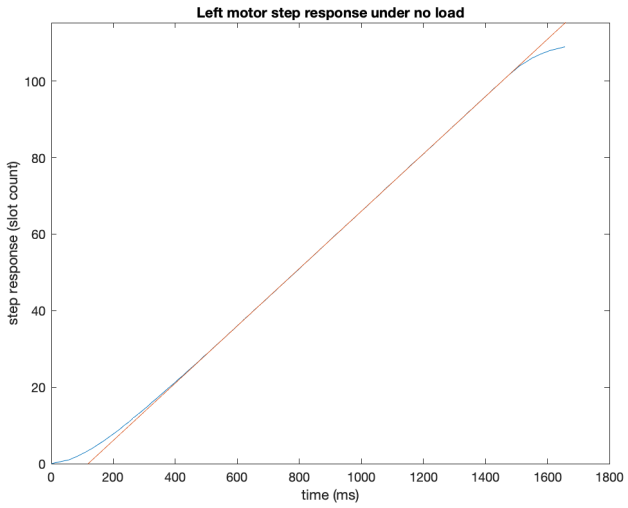
Ponieważ rozkład szumu kwantyzacji jest równomierny prawdopodobieństwo błędu o każdej wartości z tego przedziału wynosi $\frac{1}{q}$, gdzie: $q = \left(\frac{360^\circ}{\alpha_{disk}\cdot 2}\right)$. Należy również pamiętać, że pętla sprzężenia zwrotnego i pomiar enkoderów nie obejmują prawdziwego położenia platformy, a jedynie obroty koła. Wszelkie poślizgi powodują ruch który nie zostanie zarejestrowany przez enkodery. Należy pamiętać, że silniki mają wybieg, to znaczy nie zatrzymują się natychmiastowo po wyłączeniu sterowania. Duże wybiegi, przy dużych prędkościach będą powodować oscylacje w układzie skrętu o kąt zadany. Jest to niepożądane, nie tylko ze względu na niską jakość regulacji, ale komplikuje oprogramowanie układu, ponieważ enkodery nie mierzą w którą stronę kręcą się silniki, a jedynie zliczają kolejne impulsy. Dlatego przy strojeniu regulatora kluczowe jest aby dobrać nastawy gwarantujące brak oscylacji w układzie. Dodatkowo podczas programowania ograniczono wartość prędkości zadanej wypracowywanej w regulatorze kaskadowym przez regulator położenia. Zapobiega to zbyt gwałtownym startom platformy, z którymi mogłyby się wiązać niepożądane poślizgi.

6.3 Identyfikacja obiektu regulacji

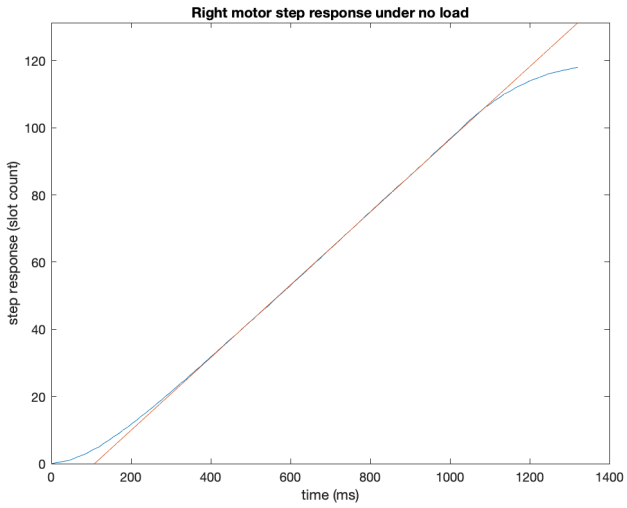
W celu poprawnego strojenia regulatorów należy zbudować model obiektu regulacji. W naszym przypadku obiektami są silniki prądu stałego. Prostym sposobem identyfikacji obiektu jest podanie na jego wejście skoku jednostkowego i analiza uzyskanej odpowiedzi czasowej. Eksperyment identyfikacyjny wykonano podając skok w postaci wysterowania silników wartością PWM równą 100. Należy pamiętać, że z punktu widzenia układów regulacji, które utworzymy na odstawie identyfikacji, ta wartość wysterowania będzie równa jedności. Przy zadawaniu wartości silnikom sygnały będzie należało odpowiednio skalować. Odpowiedź czasową na wymuszenie jednostkowe otrzymano przez komunikację poprzez port szeregowy USB. Utworzono program testowy, który drukował przez port liczbę slotów i czas zliczeń. Dane przekazywano komputerowi w formacie zgodnym z CSV (*comma separated value*), dzięki temu ich import do programu MATLAB mógł się odbyć w łatwy sposób. Program identyfikujący parametry obiektów na podstawie wyników eksperymentu przedstawia listing 1.

Odpowiedź układu zarejestrowano dla silników bez obciążenia i dla silników pod obciążeniem podwozia. Wyniki eksperymentu zobrazowano na wykresach. Rysunki 10(a) oraz 10(b) przedstawiają odpowiedzi silników bez obciążenia, rysunki 11(a), 11(b) przedstawiają odpowiedzi silników pod obciążeniem podwozia robota.

Ponieważ jako odpowiedź układu zliczano sloty, które są ekwiwalentem drogi należało spodziewać się odpowiedzi o charakterze całkującym. Wyniki identyfikacji potwierdziły te przypuszczenia, wykresy przypominają odpowiedź układu o transmitancji całki z inercją. Można zauważyć, że odpowiedzi silnika lewego i prawego różnią się, co potwierdza obserwacje dotyczącą ich niesymetrycznej pracy. Widać także, że przy obciążeniu silników podwoziem mają one większą bezwładność, czego również należało się spodziewać. Podczas strojenia układów regulacji należy bazować na parametrach obiektów identyfikowanych pod obciążeniem podwozia. Zaprojektowane układy mają na celu regulację prędkości, stąd odpowiedź należałoby zróżniczkować aby uzyskać odpowiedź prędkości w czasie. Nie jest to jednak konieczne, zróżniczkowanie odpowiedzi da wykres inercji pierwszego rzędu, z którego należy

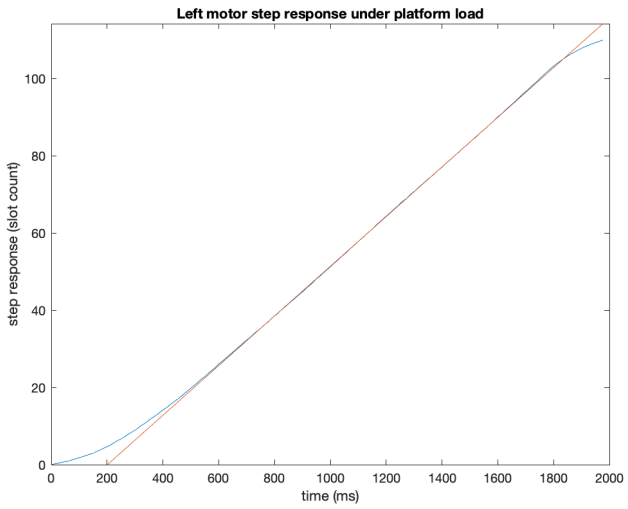


(a) Silnik lewy bez obciążenia

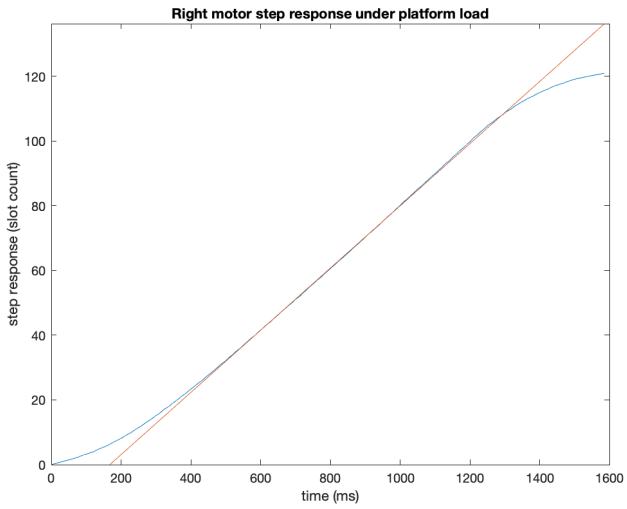


(b) Silnik prawy bez obciążenia

Rysunek 10: Wykresy odpowiedzi silników na skok jednostkowy bez obciążenia



(a) Silnik lewy z obciążeniem podwozia



(b) Silnik prawy z obciążeniem podwozia

Rysunek 11: Wykresy odpowiedzi silników na skok jednostkowy pod obciążeniem podwozia

Listing 1: Identyfikacja parametrów transmitancji silników

```
function [kv, T] = motorIdentif(csv_name)
    % Plot step response.
    figure()
    objectResponse = csvread(csv_name);
    t = objectResponse(:, 1);
    y = objectResponse(:, 2);
    plot(t, y);
    hold on;
    % Fit line to the middle part of the step response.
    coeffs = polyfit(t(31:86), y(31:86), 1);
    fittedX = linspace(min(t), max(t), 200);
    fittedY = polyval(coeffs, fittedX);
    % Plot fitted line.
    plot(fittedX, fittedY);
    hold off;
    ylim([0 inf]);
    xlabel('time (ms)');
    ylabel('step response (slot count)');
    % Calculate and return object transfer function parameters.
    T = -coeffs(1) / coeffs(2);
    kv = coeffs(1);
end
```

odczytać stałą czasową i wzmocnienie układu. Te same parametry da się określić również z wykresu odpowiedzi drogi w czasie. W celu ich określenia należy wykreślić prostą do wykresu odpowiedzi. Punkt przecięcia prostej z osią X będzie określał stałą czasową układu, natomiast nachylenie prostej jego wzmocnienie. Ponieważ układ przybliżamy transmitancją całki z inercją, jego transmitancja będzie miała postać przedstawioną na wzorze 5. Parametry stałej czasowej T i wzmocnienia k, można obliczyć z dopasowanej do odpowiedzi prostej, według wzorów 4.

$$\begin{aligned} y(t) &= a \cdot t + b \\ k &= a \\ T &= -\frac{a}{b} \end{aligned} \tag{4}$$

$$K_{o1}(s) = \frac{k}{s(1+sT)} \tag{5}$$

Identyfikacja układu i wykresy zostały wykonane w środowisku MATLAB, kod programu identyfikacyjnego przedstawia listing 1.

6.4 Strojenie regulatorów

Ponieważ regulatory PI będą regulowały prędkość silników należy przekształcić odpowiedź tak by uzyskać wykres prędkości w czasie. Uzyskana identyfikacja obiektu całkującego opisuje zmiany drogi w czasie, aby uzyskać transmitancję związaną ze zmianami prędkości należy zróżniczkować transmitancję obiektu całkującego przemnażając ją przez operator s, co obrazują przekształcenia we wzorze 6. Jak widać zmiany prędkości w czasie są opisane modelem inercyjnym.

$$K_{o2}(s) = s \cdot K_{o1}(s) = s \cdot \frac{k}{s(1+sT)} = \frac{k}{1+sT} \tag{6}$$

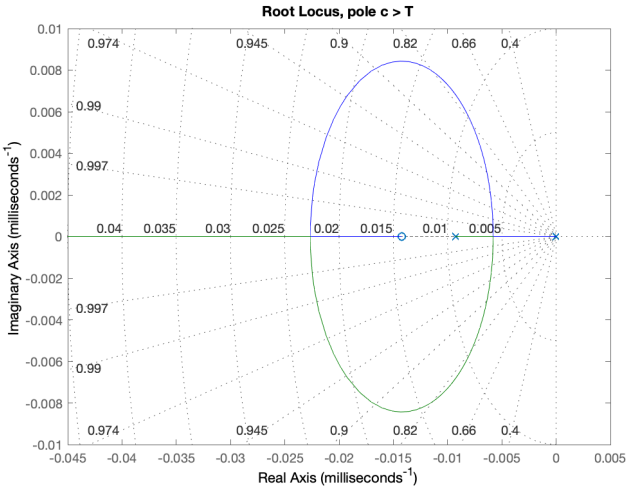
Zdecydowano się na regulator PI, który ma wystarczający astatyzm, oraz jest łatwy w implementacji. W przypadku naszego układu, który opisujemy transmitancją niskiego rzędu odpowiednim sposobem strojenia będzie użycie lini pierwiastkowych. W celu wykreślenia linii najlepiej zapisać transmitancję regulatora i obiektu w postaci zero-biegunowej, którą przedstawia wzór 7.

$$PI\left(s\right) \cdot K_{o2}\left(s\right) = \left(\frac{s+c}{s}\right) \cdot \frac{k}{1+sT} \tag{7}$$

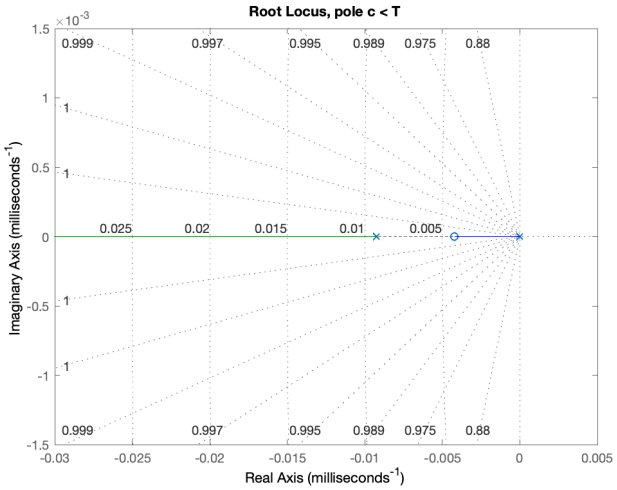
Należy zdecydować o wartości dodanego zera c. Jego stosunek do stałej czasowej T zdecyduje o kształcie linii pierwiastkowych. Można rozpatrzeć trzy przypadki:

- zero większe do stałej czasowej: $c > T$,
- zero równe stałej czasowej: $c = T$
- zero mniejsze od stałej czasowej: $c < T$.

Przypadek równości tych parametrów jest najbardziej korzystny, doprowadzi do skrócenia fragmentów transmitancji, w takim wypadku regulator działałby jak korektor, a układ byłby najszybszym z możliwych. Mimo tego jest to przypadek, który należy odrzucić. Aby osiągnąć równość parametrów należałoby przeprowadzić identyfikację o nieosiągalnej dokładności, jednak wyznaczona wartość stałej czasowej może, i zapewne różni się od rzeczywistych parametrów układu. Mało prawdopodobne jest zrealizowanie korektora, a dobór zera równego zidentyfikowanej stałej czasowej da w rzeczywistości niespodziewane rezultaty, ze względu na niedoskonałości procesu identyfikacji. Należy rozpatrzyć pozostałe dwa przypadki, liniowe pierwiastkowe dla nich wykreślono na rysunkach 12(a) oraz 12(b). Program kreślącey linie pierwiastkowe w środowisku MATLAB, na podstawie identyfikacji obiektu przedstawia listing 2.



(a) Wariant z zerem większym od stałej czasowej



(b) Wariant z zerem mniejszym od stałej czasowej

Rysunek 12: Rozpatrywane linie pierwiastkowe układu z regulatorem PI

Prędkość działania finalnego układu jest zdeterminowana przez położenie bieguna, który znajduje się najbliżej osi urojonej Y. Im bardziej od osi oddalony jest ten biegun tym szybszy układ. Przy analizie linii pierwiastkowych należy również pamiętać, że położenia biegunów o niezerowej części urojonej oznaczają oscylacje układu. Jak wspomniano w sekcji 6.2.1 zależy nam na konstrukcji układu aperiodycznego[6], czyli powinniśmy umieścić nasze bieguny na osi X.

Można zauważyć że przypadek $c > T$ pozwala na zbudowanie szybszego regulatora. W takiej konfiguracji, przy dużych wzmocnieniach najwolniejszy biegun znajduje się z lewej strony założonego zera, czyli dalej od osi urojonej Y. W konfiguracji $c < T$ najwolniejszy biegun znajduje się po prawej stronie

[6] tzn. bez oscylacji.

Listing 2: Program kreślący linie pierwiastkowe układu na podstawie identyfikacji obiektu

```
% Get object parameters based on step response in csv file
% and print the response of each motor.
[kl, Tl] = motorIdentif('motorLeftIdentif.csv');
title('Left motor step response under no load');
[kll, Tll] = motorIdentif('motorLeftIdentifLoad.csv')
title('Left motor step response under platform load');
[kr, Tr] = motorIdentif('motorRightIdentif.csv')
title('Right motor step response under no load');
[krl, Trl] = motorIdentif('motorRightIdentifLoad.csv')
title('Right motor step response under platform load');

% Plot root locus for right motor (which acts as slave).
% First option for pole c < T
figure
s = tf('s');
motRight = kr / (s + Tr);
c = Tr - 0.005;
motRight = motRight * (s + c)/s;
motRight.TimeUnit = 'milliseconds';
rlocus(motRight)
title('Root Locus, pole c < T');

% Second option for pole c > T
figure
s = tf('s');
motRight = kr / (s + Tr);
c = Tr + 0.005;
motRight = motRight * (s + c)/s;
motRight.TimeUnit = 'milliseconds';
rlocus(motRight)
title('Root Locus, pole c > T');

```

zera, czyli taki układ jest wolniejszy. Przypadek $c > T$ zdaje się bardziej korzystny, jednak niesie ze sobą ryzyko oscylacji. Teoretycznie najlepsze położenie bieguna w tej konfiguracji to miejsce w którym pierwiastki wracają z koła wartości urojonych z powrotem na oś rzeczywistą X. Jest to jednak lokacja ryzykowna ze względu na niedoskonałości identyfikacji. Przez jej błędy możemy omyłkowo, w układzie rzeczywistym, ulokować bieguny z częścią urojoną, czyli wprowadzić oscylacje. Dla pewności wyeliminowania oscylacji należy zwiększyć wzmocnienie układu i umieścić pierwiastki dalej od koła wartości urojonych. Wadą takiego rozwiązania jest spowolnienie układu przez zbliżenie bieguna do osi Y oraz wymaganie dużego wzmocnienia, które może być niemożliwe dla układu realizującego algorytm sterowania i elementów wykonawczych. Konfiguracja $c < T$ może być praktycznie lepsza, niż mogło zdawać się pierwotnie, nie niesie ryzyka oscylacji. Fakt, że taka konfiguracja jest wolniejsza niesie pewne zalety, mało gwałtowne ruchy robota minimalizują ryzyko poślizgów kół. W celu nastrojenia układów regulacji należy wykonać eksperymenty i porównać oczekiwania wynikające z linii pierwiastkowych z rzeczywistością.

7 Implementacja programistyczna

7.1 Podział i organizacja kodu

Kod odpowiadający za różne funkcjonalności robota umieszczono w oddzielnych plikach, w których znajdują się funkcje budujące warstwy abstrakcji peryferiów konstrukcji. Zdecydowano się na następujący podział plików i funkcji:

- `defines` zawiera stałe programu oraz makroinstrukcje,
- `servoSensor` zawiera funkcje budujące warstwę abstrakcji czujnika ultradźwiękowego na wieży serwomechanizmu, pozwala na kierowanie czujnika w różne strony, wykrywanie przeszkód oraz poszukiwanie wolnej drogi w otoczeniu,
- `platformMotors` zawiera funkcje budujące warstwę abstrakcji platformy robota, pozwala na jazdę na wprost oraz skręty, bazuje w działaniu na układach regulacji,
- `HC_SR04` zawiera zmodyfikowaną bibliotekę obsługi czujnika ultradźwiękowego w trybie nieblokującym, biblioteka jest wykorzystywana przez funkcje z pliku `servoSensor`,
- `main` zawiera główną logikę programu, wywołuje funkcje poszukiwań drogi bez przeszkód i podążania w tym kierunku, kontroluje także wyłączanie i włączanie układu po odebraniu rozkazów wysłanych podczerwienią.

7.2 Algorytm omijania przeszkód

Kluczowym elementem omijania przeszkód jest znajdywanie drogi, na której nie znajdują się przeszkody. Proces znajdywania nowego kierunku jazdy rozpoczyna się w momencie, w którym na obecnym kierunku jazdy zostaje wykryta przeszkoda. Wtedy robot się zatrzymuje i losuje po której stronie najpierw będzie szukał wolnej drogi. Potem zwraca w tym kierunku czujnik ultradźwiękowy i wykrywa obecność przeszkód. Jeżeli te nie zostaną wykryte platforma skręca w wylosowanym kierunku i rozpoczyna jazdę na wprost. W przeciwnym przypadku robot kieruje czujnik w przeciwną stronę i bada czy tam znajduje się przeszkoda, jeżeli nie to rozpocznie jazdę w tym kierunku. Kiedy obie strony robota są zablokowane ten zwróci się w kierunku z którego przyjechał. Opisane zachowanie obrazuje schemat na rysunku 13.

7.3 Algorytmy regulacji

7.3.1 Algorytm regulacji synchronizacji prędkości silników

Algorytm synchronizacji został zaprogramowany według schematu na rysunku 13. Kod realizujący założenia schematu przedstawiono na listingu 3. Parametry regulatora należy przekształcić względem tych uzyskanych z analizy linii pierwiastkowych, aby ułatwić obliczenia i czytelność kodu. Formuła regulatora zastosowana w kodzie ma postać jak we wzorze 9, gdzie:

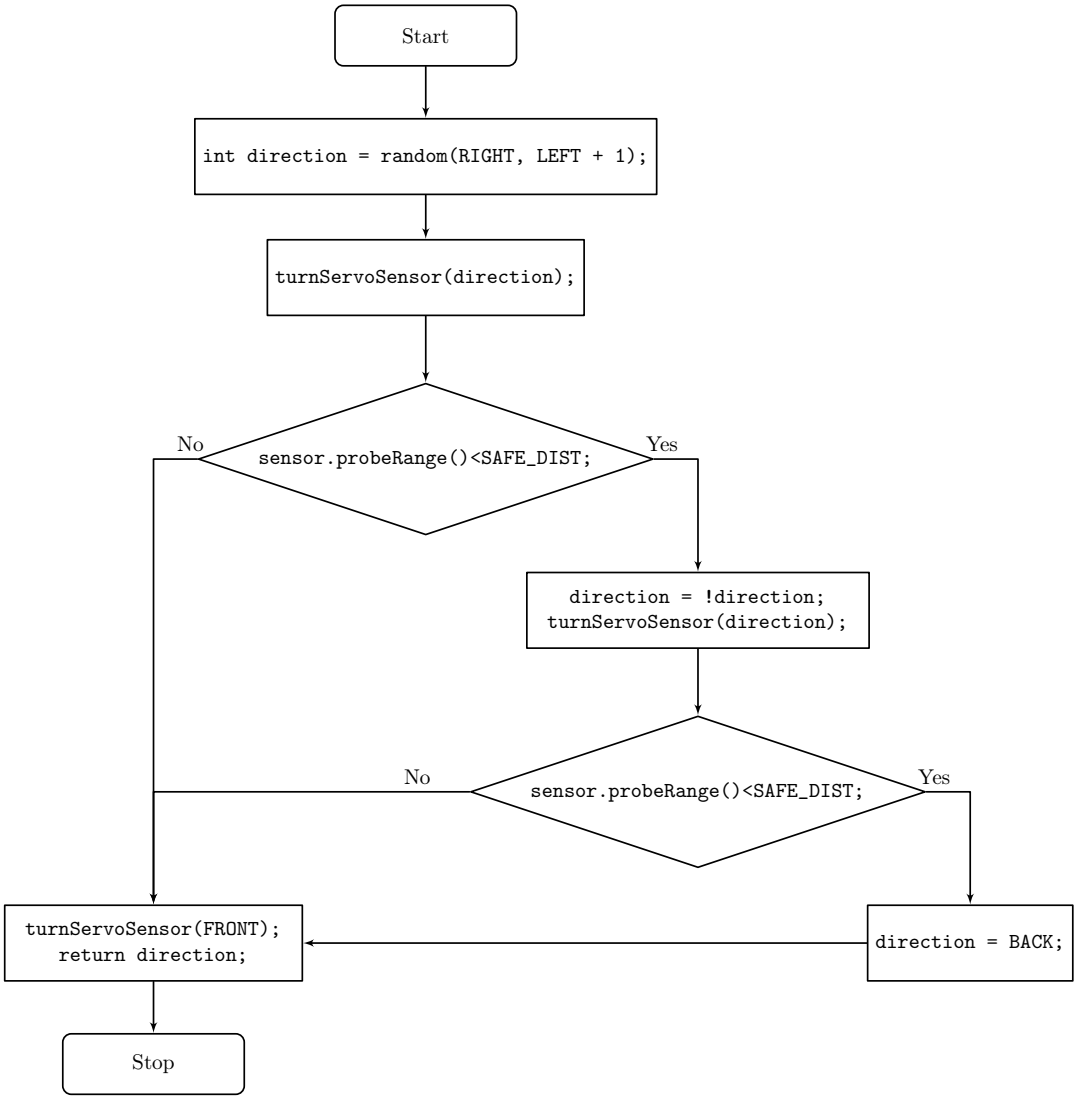
- $\Delta\alpha$ to uchyb ilości zliczonych slotów,
- $\Delta\omega$ to uchyb chwilowej prędkości,

Parametry regulatora w takiej postaci można wyliczyć według wzoru 8, gdzie:

- k_r to wzmocnienie regulatora odczytane z linii pierwiastkowych,
- c to wartość zera transmitacnji ze wzoru 7.

$$\begin{aligned} P &= k_r \\ I &= k_r \cdot c \end{aligned} \tag{8}$$

$$PI\left(\Delta\alpha, \Delta\omega\right) = P \cdot \Delta\alpha + I \cdot \Delta\omega \tag{9}$$



Rysunek 13: Schemat blokowy algorytmu znajdywania kierunku bez przeszkód

Listing 3: Algorytm regulacji synchronizacji prędkości silników

```
161 void headingVeloFix() {
162         int actuationLeft;        ///< Actuation of the left motor, steers the
            ↪ motor's PWM.
163         int actuationRight;        ///< Actuation of the right motor, steers the
            ↪ motor's PWM.
164
165         float Pm = 10;               ///< Proportional coefficient of master
            ↪ velocity regulator.
166         float Ps = 15;               ///< Proportional coefficient of slave velocity
            ↪ regulator.
167         float Is = 5;               ///< Integral coefficient of slave velocity
            ↪ regulator.
168
169         /* Master regulator */
170         float veloDeltaLeft = ((float)200.0/20.0) - (200.0/((float)
            ↪ motLeftDeltaTime));
171         actuationLeft = 112 + round(Pm * veloDeltaLeft);
172         if(actuationLeft > 255) actuationLeft = 255;
173
174         /* Slave regulator */
175         float veloDelta = ((200.0/((float) motLeftDeltaTime)) - (200.0/((float)
            ↪ motRightDeltaTime)));
176         int slotDelta = motLeftCounter - motRightCounter;
177         actuationRight = 88 + round(Ps * veloDelta + Is * slotDelta);
178         if(actuationRight > 255) actuationRight = 255;
179
180         /* Actuate motors */
181         motorRotateRight(actuationRight);
182         motorRotateLeft(actuationLeft);
183 }
```

7.3.2 Algorytm regulacji skrętu o kąt zadany

Algorytm regulacji skrętu o kąt zadany został zaprogramowany według schematu na rysunku 9. Kod pętli wewnątrz której realizowana jest regulacja przedstawiony jest na listingu 4. Wypracowanie sterowania rozpoczyna się od wyznaczenia uchybu ilości slotów wymaganych do wykonania obrotów. Wartość zadana `target` regulatora położenia jest wyliczana z makrofunkcji realizującej wzór 2. Wartość zadana prędkości została ograniczona, aby zapobiec poślizgom. Wartość zadana regulatora prędkości jest wypracowywana przez regulator położenia. Różniczkowanie ilości slotów w celu otrzymania prędkości chwilowej zrealizowano przez odwracanie czasu zliczonego między ostatnimi przerwaniami wygenerowanymi przez czujniki szczelinowe. Kod przerwań czujników przedstawia listing 5 Istotne jest aby pamiętać o rzutowaniu wartości `int` na `float` przy dzieleniu, aby zachować precyzję obliczeń. Całkowanie uchybu odbywa się przez mnożenie go przez czas jaki upłynął od ostatniego przerwania czujnika szczelinowego. W rozdziale 6.2 opisującym założenia budowy regulatora kaskadowego przedstawiono pomysł konstrukcji zerującej wartość całki przy każdym przerwaniu czujnika szczelinowego. Odpowiednikiem tego założenia w kodzie jest brak zapamiętywania i sumowania poprzednich wartości całki, przy każdej nowej szczelinie czas od tego przerwania jest liczony od nowa. Jest to bardzo

uproszczona realizacja członu całkującego, jednak zupełnie wystarczająca w naszym przypadku[7].

7.4 Kod funkcji obsługującej przerwanie czujnika szczelinowego

Kod funkcji obsługującej przerwanie czujnika szczelinowego[8] przedstawiono na listingu 5. Funkcje obsługi przerwań czujników szczelinowych mają za zadanie inkrementację licznika szczelin oraz obliczenie czasu od ostatniego przerwania. W funkcji należało uwzględnić inne zachowanie przerwań przy początku ruchu robota. Przy testach czujników okazało się, że przy zmianie stanu czujnika następuje zjawisko podobne do odbijania styków, co sprawiało że fałszywie zliczana była duża ilość szczelin. Aby temu zapobiec ignorowane są sygnały które pojawiły się zbyt szybko względem ostatniej zmiany. Tworząc kod przerwań należy pamiętać, że ich obsługa w mikrokontrolerach ATmega wyłącza działanie liczników, przez co funkcje `millis()` w przerwaniach zwracają czas włączenia przerwania, a czas który upłynął na obsłudze przerwania jest pomijany przy następnych wywołaniach funkcji `millis()`. To zjawisko mogłoby powodować błędy jeżeli zależałoby nam na precyzyjnym zliczaniu czasu od wydarzenia, kiedy podczas zliczania obsługujemy dużo przerwań o długim kodzie obsługi. Taka potrzeba nie zachodzi jednak w naszym przypadku, więc użycie funkcji `millis()` w kodzie obsługi przerwań jest bezpieczne.

8 Wykorzystane technologie

8.1 Narzędzia programistyczne

Ze względu na ograniczenia programu Arduino IDE zdecydowano się na użycie środowiska PlatformIO[9], które jest wtyczką do programu Visual Studio Code[10]. Taka konfiguracja oferuje:

- bogate kolorowanie składni języków programowania,
- zaawansowany system autouzupełniania kodu z użyciem technologii *IntelliSense*,
- automatyczne wykrywanie portów szeregowych z podłączonymi płytkami,
- konstrukcja bazująca na plikach *make* i programie AVRDUDE.

Kod projektu wersjonowano z użyciem programu Git oraz korzystając z usług serwisu GitHub, który oferuje internetowy interfejs do zarządzania repozytorium, system *issue tracking* oraz zarządzanie projektem za pomocą tablic kanban.

8.2 Komunikacja z mikroprocesorem

Do komunikacji z mikroprocesorem przydatny okazał się program AVRDUDE. Szczególnie jego polecenia testu poprawnej komunikacji i odczytu *fuse bitów*. Poniżej przedstawiono przykład użycia polecenia testu mikrokontrolera oraz jego wynik:

```
$ sudo avrdude -p m32u4 -c usbasp -P usb
avrdude: AVR device initialized and ready to accept instructions
Reading | ################################################## | 100% 0.01s
avrdude: Device signature = 0x1e9587
```

[7] Działanie członu całkującego można sprawdzić w sposób eksperymentalny. W tym celu wyłączono człon proporcjonalny, zerując jego parametr `P`. Robot powinien na początku nie ruszyć, a wraz z brakiem jego ruchu powinien być całkowany coraz większy uchyb i zadawane coraz większe wysterowanie silników. Po chwili powinien być słyszany odgłos silników próbujących pokonać opory ruchu. Ostatecznie wartość całki powinna być na tyle duża, że silniki ruszą i nastąpi obrót kół o kąt powodujący zliczenie jednej szczeliny. Ponieważ całkę zerujemy przy każdej zliczonej szczelinie, robot się zatrzyma i zacznie całkować uchyb od nowa. Proces ten będzie powtarzał się aż robot wykona stopniowo pełen skręt. Przewidywania te potwierdził eksperyment.

[8] Funkcje obsługi przerwań lewego i prawego czujnika należało napisać oddzielnie, ale ich kod jest praktycznie taki sam. Opisano tu kod obsługi jednego czujnika, ale drugi różni się tylko operacjami na innych zmiennych globalnych, logika jego działania jest identyczna.

[9] PlatformIO: `https://platformio.org`

[10] Visual Studio Code: `https://code.visualstudio.com`

```
avrdude: safemode: Fuses OK
avrdude done.  Thank you.
```

Polecenie odczytu *fusebitów* ma z kolei następujący format i daje wynik:

```
$ sudo avrdude -p m32u4 -c usbasp -P usb -U hfuse:r:hfuse.hex:h
avrdude: AVR device initialized and ready to accept instructions
Reading | ################################################## | 100% 0.01s
avrdude: Device signature = 0x1e9587
avrdude: reading hfuse memory:
Reading | ################################################## | 100% 0.00s
avrdude: writing output file "hfuse.hex"
avrdude: safemode: Fuses OK
avrdude done.  Thank you.
```

Jak widać w powyższych instrukcjach, do programowania mikrokontrolera użyto programatora USBasp. Przy zakupie takiego programatora warto upewnić się, że kupujemy wersję z najnowszym firmwarem, dzięki czemu programator będzie miał od razu dostępną funkcję automatycznego dopasowania do taktowania zegara mikrokontrolera. Program Arduino IDE pozwala na wypalenie w pamięci mikrokontrolera bootloadera Arduino. Opcja ta jest dostępna w pasku menu w zakładce narzędzia.

9 Perspektywy rozwoju, podsumowanie

Wykonany projekt spełnił założenia, dzięki zastosowaniu układów regulacji napędu silników robot jest stanie poruszać się na tyle precyzyjnie by wymijać przeszkody. Wszystkie elementy układu oraz części obwodu drukowanego działają w sposób poprawny. Załączniki do dokumentacji zawierają materiały multimedialne prezentujące osiągniętą dokładność pracy robota. Projekt miał charakter edukacyjny, jego wyników nie da się porównać z produktami komercyjnymi, jednak zdobyte podczas jego realizacji doświadczenie i fragmenty pracy mogą być przydatne przy konstrukcji robotów magazynowych, robotów labiryntowych oraz przy budowie dedykowanych serwomechanizmów.

Precyzję układu można zwiększyć stosując tarcze enkoderów o większej ilości szczelin, lub stosując enkodery umieszczane przed przekładnią silników. Ciekawą ścieżką rozwoju może być zamiana układu pomiarowego z enkodera badającego obroty kół na system *Yaw, Pitch Roll*. Jest to zestaw współpracujących ze sobą: żyroskopu, akcelerometru oraz magnetometru w celu ustalenia kierunku układu w przestrzeni poprzez kąty Eulera. Taki układ pomiarowy cechowałby się daleko większą dokładnością i obejmowałby pętlą sprzężenia zwrotnego nie tylko obroty silnika, ale rzeczywisty kierunek robota, eliminując błędy związane z poślizgiem kół. Na obecnym etapie projektu taki system pomiarowy znacząco przekroczyłby budżet, jednak w przyszłości może on stanowić ciekawe zagadnienie. Jego wprowadzenie pozwoliłoby na zachowanie sporej ilości aktualnego kodu, zmiany wymagałby jedynie tor pomiarowy i zadajnik.

Na bieżącym etapie układ nie zapamiętuje ścieżki którą przebył, aby to zmienić można skonstruować drzewo binarne wykonanych skrętów i odległości między nimi. Dzięki temu układ mógłby analizować przebytą drogę i szukać wyjścia z bardziej złożonych labiryntów. Kod programu, pliki Autodesk Eagle oraz dokumentacja są dostępne na stronie GitHub projektu: `https://github.com/MaciejZj/MazeBot`.

Literatura

[1] Rafał Baranowski, *Mikrokontrolery AVR ATmega w praktyce* Wydawnictwo BTC, Warszawa 2005

[2] Stefan Węgrzyn, *Podstawy automatyki* Państwowe Wydawnictwo Naukowe, Warszawa 1974.

[3] Y. Takashi, M.J. Rabins, D.M. Auslander, *STEROWANIE i systemy dynamiczne*, Wydawnictwo Naukowo Techniczne, Warszawa 1976.

[4] Praca zbiorowa pod redakcją Dariusza Buchczika i Stanisława Walusia, *Laboratorium Podstaw Miernictwa*, Wydawnictwo Politechniki Śląskiej, Gliwice 2014.

[5] David Halliday, Robert Resnick, Jearl Walker, *Podstawy Fizyki. Tom 1*, Wydawnictwo Naukowe PWN, Warszawa 2003.

[6] Atmel: *ATmega16U4/ATmega32U4 8-bit Microcontroller with 16/32K bytes of ISP Flash and USB Controller*, http://ww1.microchip.com/downloads/en/devicedoc/atmel-7766-8-bit-avr-atmega16u4-32u4_datasheet.pdf

[7] STMicroelectronics: *L298 DUAL FULL-BRIDGE DRIVER*, https://www.sparkfun.com/datasheets/Robotics/L298_H_Bridge.pdf

[8] Elec: *Ultrasonic Ranging Module HC - SR04*, https://www.mouser.com/ds/2/813/HCSR04-1022824.pdf

[9] Vishay: *TSOP48 Photo Modules for PCM Remote Control Systems*, http://mkpochtoi.narod.ru/TSOP4836_ds.pdf

[10] STMicroelectronics: *TL7800 series positive voltage regulators*, https://datasheet.octopart.com/L7805CV-STMicroelectronics-datasheet-7264666.pdf

[11] Praca zbiorowa pod redakcją Krzysztofa Zioło *LABORATORIUM ELEKTRONIKI I Elementy półprzewodnikowe i układy podstawowe*. Wydanie III zmienione . Wydawnictwo Politechniki Śląskiej, Gliwice 1999.

Listing 4: Algorytm regulacji skrętu o kąt zadany

```
85  while(1) {
86          int slotDeltaRight = target - motRightCounter;  ///< Error of position (precedent)
            ↪  regulator for right motor.
87          int slotDeltaLeft = target - motLeftCounter;    ///< Error of position (precedent)
            ↪  regulator for left motor.
88
89          /* P regualtor formula, calcualtes velocity setpoint */
90          setPointVeloRight = Pp * slotDeltaRight;
91          setPointVeloLeft = Pp * slotDeltaLeft;
92
93          /* Avoid too high velocity set points to elude wheel slips during long turns */
94          if(setPointVeloRight > 16) setPointVeloRight = 16;
95          if(setPointVeloLeft > 16) setPointVeloLeft = 16;
96
97          /* Calulate velocity errors */
98          float veloDeltaRight = setPointVeloRight - (1.0/((float) motRightDeltaTime));
99          float veloDeltaLeft = setPointVeloLeft - (1.0/((float) motLeftDeltaTime));
100         /* Get current time to compare it with last time of ISR and estimate integral */
101         unsigned long compareTime = millis();
102
103         /* PI regulator formula, calculates motors actuations */
104         actuationRight = round(Pv * veloDeltaRight + Iv * veloDeltaRight * (compareTime -
            ↪  lastTimeRight));
105         actuationLeft = round(Pv * veloDeltaLeft + Iv * veloDeltaLeft * (compareTime -
            ↪  lastTimeLeft));
106
107         /* Prevent oversaturation of PWM */
108         if (actuationRight > 255) actuationRight = 255;
109         if (actuationLeft > 255) actuationLeft = 255;
110
111         /* If target is reached stop the motor, otherwise actuate it */
112         if (motLeftCounter >= target) {
113                 motorLeftStop(); leftIsStopped = true;
114         } else
115                 motorRotateLeft(direction * actuationLeft);
116
117         if (motRightCounter >= target) {
118                 motorRightStop(); rightIsStopped = true;
119         } else
120                 motorRotateRight(-direction * actuationRight);
121
122         /* If both motors are stopped exit function */
123         if(leftIsStopped && rightIsStopped) {
124                 delay(MOTOR_STOP_DELAY);
125                 return;
126         }
```

Listing 5: Kod funkcji obsługującej przerwanie czujnika szczelinowego

```
185 void motorLeftCounterInt() {
186         /* When ISR is entered first time in the movement we shouldn't calculate time
            ↪  delta */
187         if(motLeftCounter == 0) {
188                 lastTimeLeft = millis();
189                 motLeftCounter++;
190                 return;
191         }
192
193         /** Straightforward low pass filter, debounces the slot sensor
194          *          when it is on the edge of a slot.
195          */
196         unsigned long timeNow = millis();
197         if(timeNow - lastTimeLeft < 2) return;
198
199         /**         Increment slot counter and calculate time delta to provide
200          *          information for velocity calculation
201          */
202         motLeftCounter++;
203         motLeftDeltaTime = timeNow - lastTimeLeft;
204
205         lastTimeLeft = timeNow;
206 }
```