Robot mobilny klasy minisumo „Shine”

Karol Sydor

Koło Naukowe Robotyków „KoNaR”
www.konar.pwr.wroc.pl
Wrocław 2009

Spis treści

1 Wstęp

Niniejszy opis ma na celu przedstawienie konstrukcji robota klasy minisumo - "shine"[1]. Informacje tutaj zawarte mają na celu możliwie dokładne przedstawienie zastosowanych rozwiązań mechanicznych, elektronicznych i algorytmicznych[2]. Opis podzielono na opis konstrukcji mechanicznej, elektroniki i oprogramowania, przez co opisy poszczególnych modułów są nieco rozbite. Mam nadzieję że treść zainspiruje czytelników. W razie wątpliwości lub pytań - proszę o kontakt: karol@elektroda.net

2 Konstrukcja mechaniczna

2.1 Główne założenia

W celu opracowania konstrukcji mechanicznej przyjęto następujące założenia:

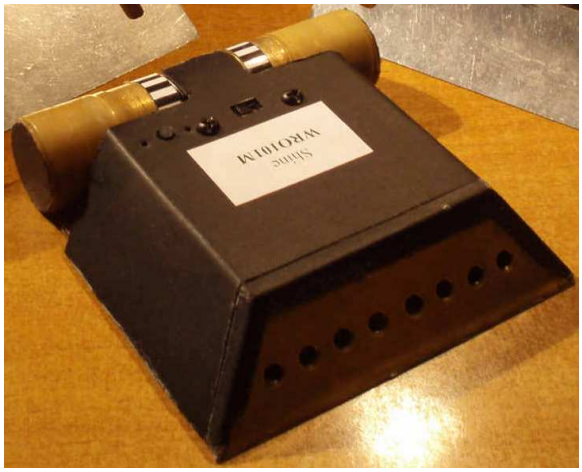
- Robot ma być napędzany za pomocą dwóch silników i pary kół, trzecim punktem podparcia ma być pług
- Koła napędowe przesunięte jak najdalej do tyłu, uniesienie robota od przodu, nie może powodować zmniejszenia przyczepności
- Masa robota ma skupiać się jak najniżej, oraz jak najbliżej osi kół
- Robot ma być możliwie jak najniższy, aby utrudnić jego namierzenie i obniżyć położenie środka ciężkości
- Robot musi posiadać prosty i niezawodny układ dalmierzy
- Robot zostanie wyposażony cztery czujniki białej linii, umieszczone na planie kwadratu, jak najdalej od siebie
- Robot ma posiadać układ określający przemieszczenie wzdłużne, działający niezawodnie, na każdej powierzchni i również w trakcie uniesienia
- Koła napędowe wyposażone w enkodery, co w połączeniu z czujnikiem przemieszczenia umożliwi realizację algorytmów kontroli trakcji
- Robot powinien posiadać czujniki umożliwiające wykrycie uniesienia, co ma wyłączać czujniki białej linii i rozpocząć atak lub unik.

[1] Treści tutaj zawarte mają wyłącznie charakter edukacyjny. Nie ponoszę odpowiedzialności za szkody powstałe w wyniku korzystania z tego opisu. Nie zezwalam na kopiowanie całości, bądź fragmentów, bez mojej zgody.

[2] Dołożyłem wszelkich starań, aby w opisie nie było błędów. Nie gwarantuję jednak że ich nie ma

- Pług powinien być wykonany solidnie, aby nie ulegał deformacji.

Powstała konstrukcja spełnia wszystkie powyższe założenia (rys.1). Nazwa pochodzi od błysku metalu, jaki robot uzyskiwał po wypolerowaniu. Przed występem na zawodach "Robotic Arena 08" robot został pokryty matowym, czarnym materiałem. Takie malowanie utrudnia namierzanie za pomocą czujników podczerwieni. Po dawnym blasku pozostała tylko nazwa...



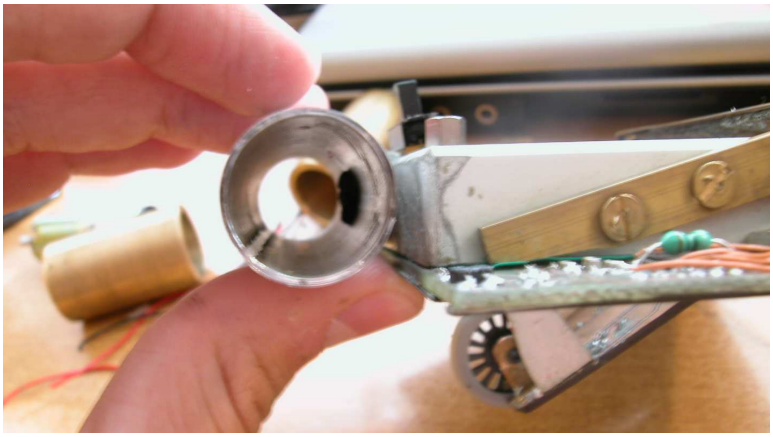
Rysunek 1: Robot shine

2.2 Oś napędowa

Oś napędowa, wraz z kołami, została wytoczona na zlecenie. Składa się z rurki w której zamocowane są silniki, oraz kół nasadzanych na całość. Rurka została wytoczona ze stali nierdzewnej. Jej średnica wewnętrzna została tak dobrana, aby ciasno wchodziły do niej silniki. Dodatkowo, posiada na środku dwa otwory mocujące i jeden na doprowadzenie zasilania silników (rys.2). Oś napędowa zamocowana jest do ramy za pomocą dwóch śrub M3, dodatkowo całe połączenie wypełnione jest klejem typu "płynny metal". Koła zostały wytoczone z mosiądzu, co zapewniło im odpowiednią twardość i masę (rys.3).

2.2.1 Pierwsze podejście - silniki Copal

Na początku zastosowano silniki Copal HG16-30, z przekładnią 1:30. Koła posiadały otwór nieprzelotowy o średnicy 3mm, oraz dwie śruby, prostopadłe do wału, którymi były mocowane. Jedna śruba dochodziła do fazowania na wale, natomiast druga - z zaostrzonym końcem, wchodziła w wyżłobienie, po przeciwnej stronie. Układ dwóch śrub działających w przeciwnych kierunkach miał zapewnić możliwość redukcji "bicia" poprzez odpowiednie



Rysunek 2: Wnętrze osi napędowej

Rysunek 3: Wnętrze koła, silniki

ich dokręcenie, oraz zapewnić że koła nie zsuną się podczas walki. Jednak w praktyce, ze względu na zastosowanie nieodpowiedniego pasowania - "bicie" było znaczące. Najsłabszym elementem całej konstrukcji okazały się jednak same przekładnie silników - o plastikowych trybach w układzie przekładni planetarnej. Po wstępnych testach, okazało się że jeden trybów w przekładni uległ uszkodzeniu (rys.4). Ponieważ przekładnie zostały zniszczone jeszcze przed jakąkolwiek walką, zrezygnowano z tych silników.

2.2.2 Drugie podejście - silniki Canon

Napędy wymieniono na Canon DG16-T14G1B o przełożeniu 1:34. Posiadają one przekładnie zębate o metalowych trybach. Z przekładni wychodzi trzpień

Rysunek 4: Uszkodzona przekładnia silnika

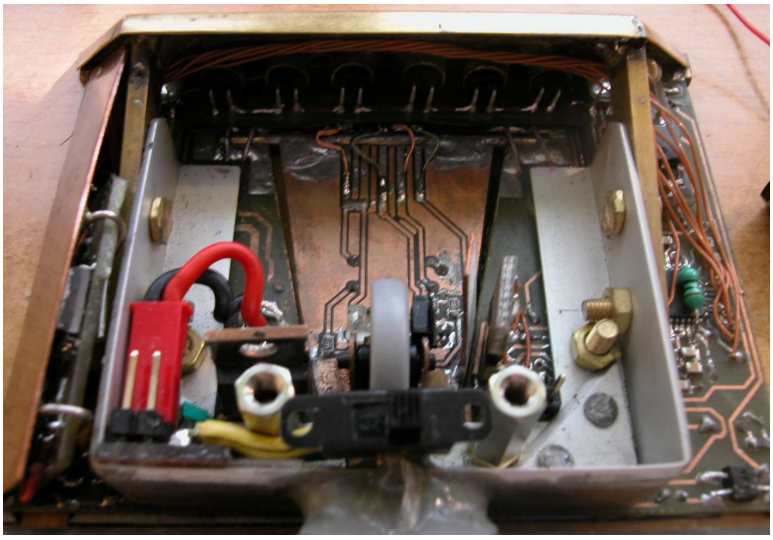
fazowany z dwóch stron, o średnicy 4mm. Wytoczono nowe otwory w kołach - przelotowe. W celu wyeliminowania problemów z osiowością, silniki zamocowano za pomocą kleju typu "płynny metal". Dodatkowo całość skręcono za pomocą śrub. Połączenie jest nierozbieralne. W zamian udało się zredukować bicie do akceptowalnego poziomu.

2.3 Rama, przód robota, mocowanie płyty głównej

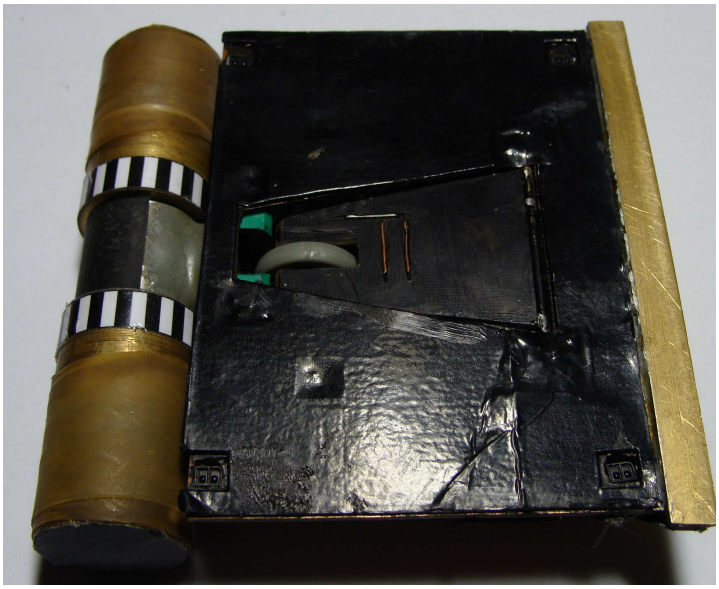
Rama robota wykonana jest z kątownika aluminiowego (dostępne w sklepach budowlanych). Kątownik jest nacięty i wygięty w kształcie litery "U". W miejscach zgięcia, przynitowano dodatkowe wsporniki. Do ramy od spodu przykręcona jest (4 śruby M3) płyta główna robota. W celu zabezpieczenia przed zwarciami pomiędzy PCB a ramą umocowano taśmę izolacyjną. Od spodu zabezpieczono elektronikę za pomocą kartonu oklejonego czarną folią. Do boków ramy przykręcono (4 śruby M3) grube mosiężne szyny o przekroju 7*2.5mm. Do nich przylutowano płytę czołową, wykonaną z laminatu dwustronnego. W płycie czołowej wlutowano, odpowiednio ścięte, rurki mosiężne o przekroju wewnętrznym 5mm. W rurkach tych zamocowano diody LED dalmierza (rys.5).

2.4 Pług

Od spodu co płyty czołowej przylutowany jest pług (rys.7). Jest on wyszlifowany z profilu mosiężnego, o przekroju 7*2.5mm. Do niego przylutowana jest też płyta główna robota (rys.6) W celu zapewnienia jednolitej struktury przedniej części, do pługa, oraz płyty czołowej przylutowana jest cienka



Rysunek 5: Wygląd wnętrza robota



Rysunek 6: Spód robota

(0.3mm) blacha mosiężna. Nawiercono w niej eliptyczne otwory pod diody LED. Dzięki odpowiedniemu uformowaniu krawędzi, całość wygląda jednolicie.

Rysunek 7: Mocowanie pługa

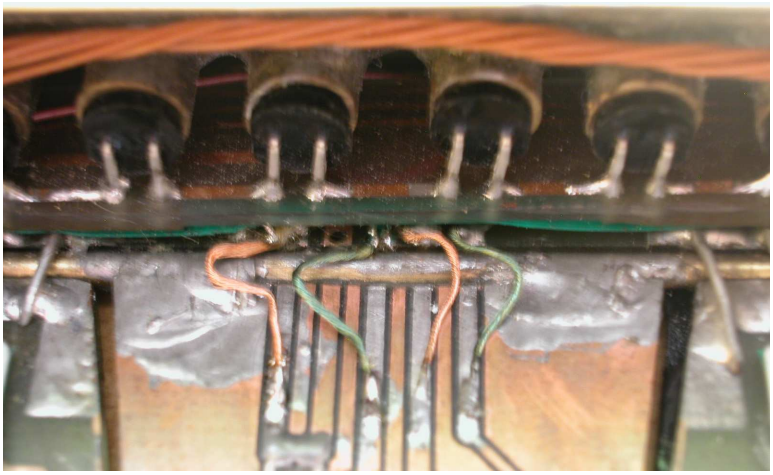
Rysunek 8: Obudowa robota

2.5 Obudowa

Obudowa robota wykonana jest z dwóch warstw blachy (rys.8). Spodnia warstwa wykonana jest z blachy tytanowo-cynkowej o grubości 0.55mm. Wyprofilowana jest w ten sposób, że z przodu wchodzi "na zakładkę" pod płytę czołową, oraz pod mosiężną osłonę po bokach. Druga warstwa wykonana jest z blachy mosiężnej o grubości 0.4mm. Dzięki zastosowaniu dwóch rodzajów metalu, uzyskano bardzo dużą sztywność, masywność i estetyczny wygląd. Osłona zamocowana jest do ramy za pomocą dwóch dystansów i śrub M3. Posiada otwory na przycisk i dwie diody LED. Obudowa została wypolerowana na wysoki połysk. Przed zawodami "Robotic Arena 08" została dodatkowo oklejona warstwą specjalnego materiału w kolorze sadzy.

2.6 Klapo-czujnik

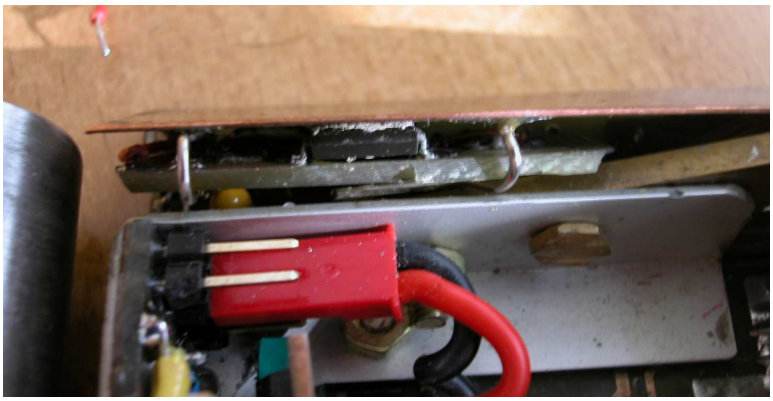
Na spodzie robota znajduje się klapka, zamocowana poprzez zawias do płyty głównej robota. Na tej klapce znajduje się czujnik przemieszczenia wzdłużnego, oraz przylutowana pionowo miedziana blaszka, będąca częścią czujnika uniesienia. Sygnały elektryczne z klapki przenoszone są na płytę główną za pomocą elastyczne linki (rys.9), zawierających sznurek teflonowy oraz miedziane emaliowane druciki (tego typu linka znajduje się w cienkich kablach słuchawek dousznych). Zawias wykonany jest z mosiężnej rurki o średnicy 2mm. Zamocowana jest ona swobodnie w płycie głównej. Klapo-czujnik posiada ograniczenie maksymalnego wychylenia (kropla cyny przylutowana do mosiężnej rurki). Zawias posiada znikome luzy boczne, co gwarantuje poprawne składanie klapki i dużą swobodę ruchu (klapka opada z łatwością).



Rysunek 9: Zawias klapoczujnika

2.7 System chłodzenia

Ponieważ zastosowany driver silników wydziela pewne ilości ciepła, konieczne było zastosowanie radiatora. Zamontowano kawałek blachy miedzianej grubości 0.6mm. Blacha przymocowana jest za pomocą dwóch zacisków sprężynowych (rys.10). Zastosowano termopastę srebrową w celu zapewnienia jak najlepszego przekazywania ciepła. Chłodzenie pasywne wystarcza w zupełności, zwłaszcza że silniki Canon posiadają o połowę mniejszy prąd maksymalny (w stosunku do modeli Copal). Po założeniu osłony miedziany radiator przylega do niej, co dodatkowo poprawia wydajność chłodzenia.



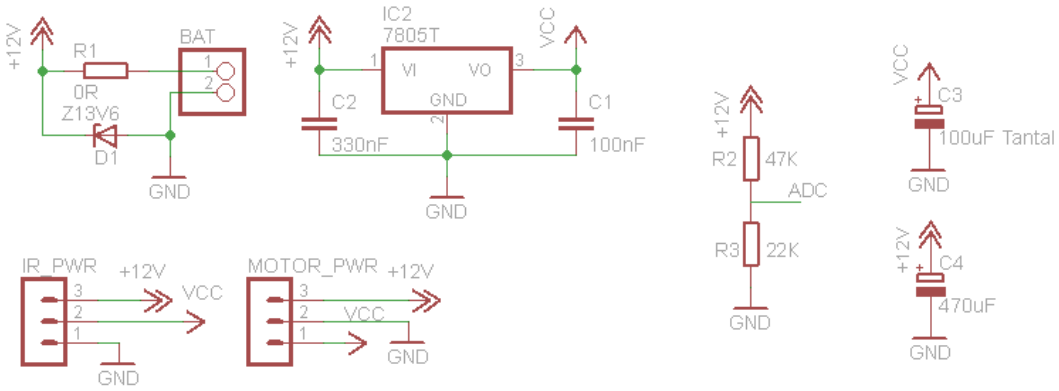
Rysunek 10: Mocowanie radiatora

3 Układ napędowy

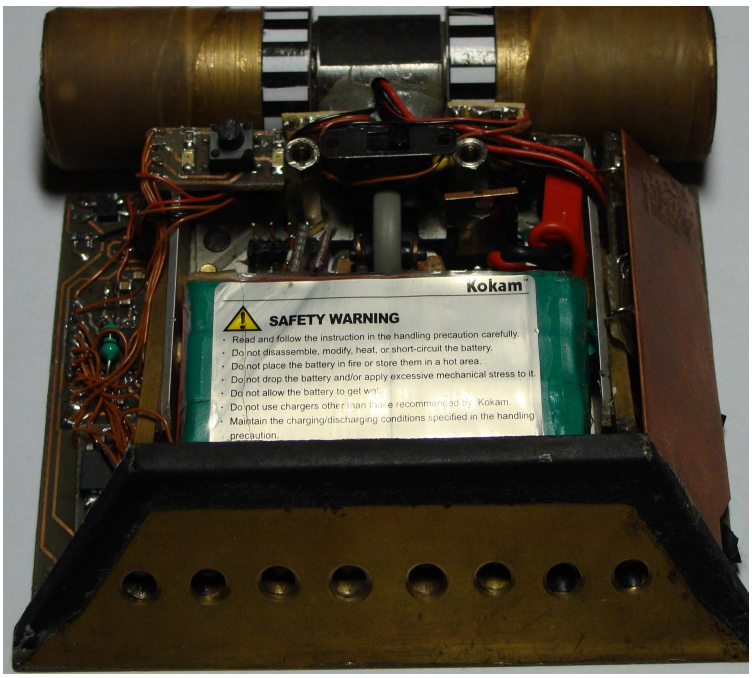
Układ napędowy robota składa się z dwóch silników z przekładniami. Zastosowane silniki Canon DG16, cechują się momentem obrotowym dochodzącym do 2.5mNm). Fabryczne przekładnie 1:34 zwiększają moment obrotowy i zmniejszają prędkość obrotową. Koła zamocowane są bezpośrednio na wale wyjściowym przekładni. Wyliczony maksymalny moment obrotowy to około 65mNm. Znamionowa prędkość obrotowa kół to 360rpm, co przy średnicy kół 25mm daje prędkość maksymalną dochodzącą do 0.5m/s. Planowano zastosować silniki o większej prędkości obrotowej, jednak priorytetowe były wymiary zespołu napędowego (średnica 16mm, długość 37mm). Niewielki moment obrotowy sprawia że robot nie traci łatwo przyczepności. Zastosowane dodatkowo algorytmy uników w przypadku spychania, sprawiają że robot pomimo niewielkiej mocy, jest godnym przeciwnikiem dla robotów o większych silnikach.

4 Układ zasilania

Schemat układu przedstawiono na rys. 11. Do zasilania robota wykorzystano akumulator litowo-polimetowy. Posiada on trzy ogniwa, dając łączne napięcie 9.6-12.6V (Znamionowe 11.1V). Jego pojemność wynosi 450mAh (zamontowaną baterię pokazano na rys.12). Maksymalny prąd pobierany z ogniwa dochodzi do 1.2A - wartość średnia i szczytowo do 4.5A. Wybór akumulatora był zdeterminowany jego wielkością i napięciem (napięcie znamionowe silników to 12V). Zasilanie odcinane jest za pomocą wyłącznika hebelkowego. Napięcie bezpośrednio z akumulatora pobierane jest do sterownika silników (MOTOR_PWR), dalmierzy (IR_PWR), oraz dzielnika rezystorowego (R2,R3) podpiętego do przetwornika ADC mikrokontrolera. Dzięki temu mikrokontroler ma możliwość kontrolowania napięcia akumulatora. Część



Rysunek 11: Schemat układu zasilania



Rysunek 12: Robot z zamontowaną baterią

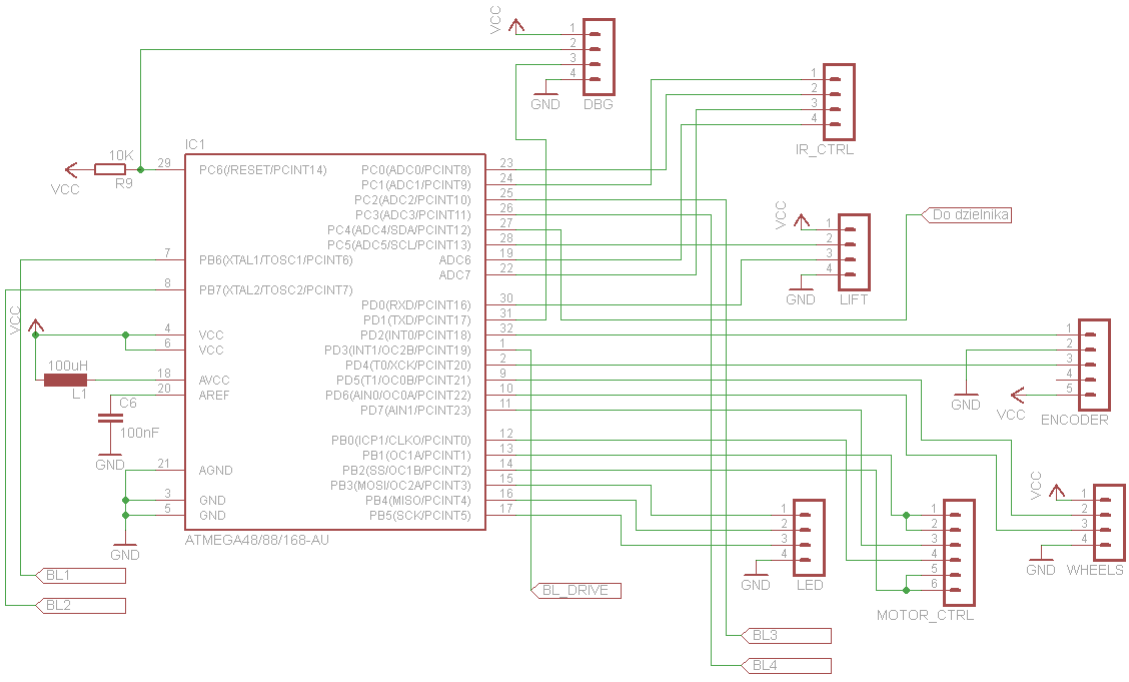
cyfrowa zasilana jest przez stabilizator liniowy LM7805 (IC2). Kondensatory C1-C4 filtrują napięcie zasilania. Rezystor R0 pełni rolę bezpiecznika, natomiast dioda D1 zabezpiecza układ przed przekroczeniem napięcia maksymalnego (13.6V) i odwrotnym podłączeniem zasilania (przepala bezpiecznik).

5 Układ sterowania

Układ sterowania robota znajduje się na płycie głównej. Z racji ograniczenia jej wielkości, oraz powierzchni do wykorzystania, część elementów umieszczona jest na płytkach dodatkowych. Zamontowano je w wolnych przestrzeniach wewnątrz robota i połączono za pomocą kynaru.

5.1 Mikrokontroler

Sercem całego robota jest mikrokontroler Atmega168 firmy Atmel. Programowanie mikrokontrolera odbywa się za pomocą jednoprzewodowego interfejsu DebugWire™. Jednostka taktowana jest z wewnętrznego oscylatora o częstotliwości 8MHz, jego dokładność jest wystarczająca do komunikacji przez UART. Otoczenie mikrokontrolera przedstawiono na rys. 13. (UWAGA! Na rysunku nie są widoczne kondensatory filtrujące VCC i AVCC!) Większość układów wykonawczych i sensorów została dołączona poprzez złącza w rastrze 2.54mm. Wyjaśnienie opisu złącz zestawiono w tab.1



Rysunek 13: Schemat otoczenia mikrokontrolera

W celu ułatwienia zmiany konfiguracji parametrów robota, wyprowadzono linię TX USART-a. Za pomocą prostego interfejsu można podłączyć robota do palmtopa, poprzez szeregowym. Wystarczy program terminalowy,

Symbol	Opis
DBG	Debuggowanie, programowanie i komunikacja
IR_CTRL	Dalmierz
LIFT	Czujnik uniesienia
Do dzielnika	Dzielnik napięcia baterii
ENCODER	Czujnik przemieszczenia wzdłużnego
WHEELS	Enkodery na kołach
MOTOR_CTRL	Driver silnika
LED	Diody LED i przycisk
BL1-4	Czujnik białej linii - kanały 1-4
BL_DRIVE	Sterowanie LED nadawczymi białej linii

Tablica 1: Opis złącz

aby mikrokontroler mógł wyświetlić parametry, takie jak napięcie zasilania, czy dane czujników.

5.2 Końcówka mocy

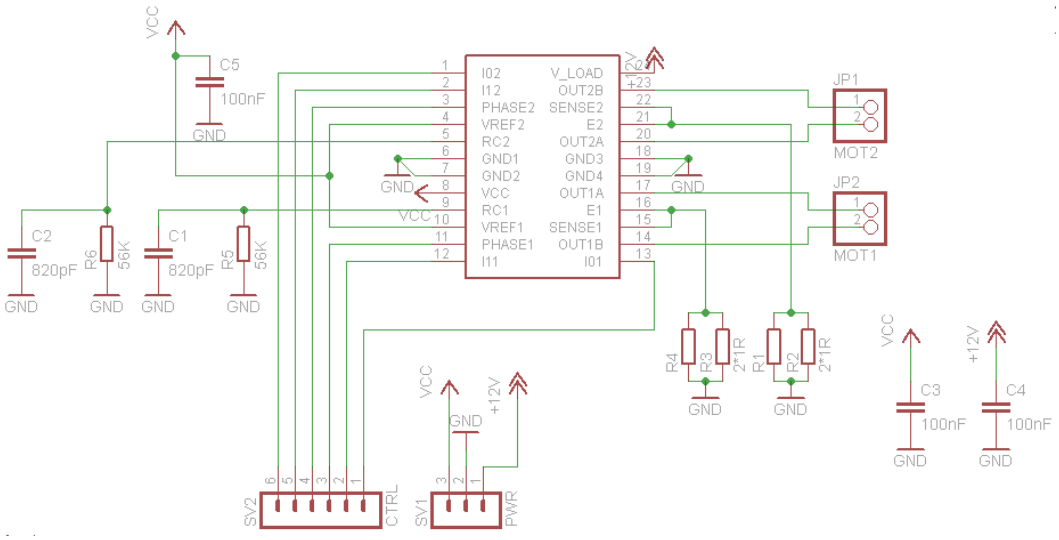
W robocie zastosowano scalony, podwójny driver silnika, z wbudowanym sprzętowym regulatorem prądu - UDN2916. Maksymalny prąd jaki jest w stanie dostarczyć wynosi 750mA dla każdego silnika (1A szczytowo). Każdy kanał posiada dwa wejścia sterujące określające prąd maksymalny, oraz jedno wejście oznaczające kierunek. Wykorzystano rezystory pomiarowe o wartości 0.5Ω co przy napięciu odniesienia Vref = 5V odpowiada maksymalnemu prądowi = 1A. Ponieważ maksymalny prąd zatrzymanego silnika Canon dochodzi do 600mA, ograniczenie działa tylko w przypadku hamowania przeciwprądem, co trwa na tyle krótko że sterownik wytrzymuje takie warunki. Nie wykorzystano możliwości pracy układu z mniejszymi wartościami prądu. Regulację prędkości dla każdego kanału zrealizowano podłączając oba wejścia sterujące prądem, do jednego pinu mikrokontrolera generującego impulsy PWM. Schemat układu końcówki mocy przedstawiono na rys. 14

6 Czujniki

6.1 Czujniki białej linii

W robocie zastosowano zespół czujników białej linii (rys.15) Schemat rozwiązania pokazano na rys.16.

Z racji oszczędności miejsca, zastosowano zupełnie nowe podejście do klasycznego problemu czujników białej linii. Zrezygnowano z komparatora analogowego o regulowanym progu. W zamian, dobrano odpowiednio rezystory polaryzujące, a próg działania reguluje się poprzez zmianę prądu płynącego przez szeregowo połączone diody nadawcze. Zasilane są one z

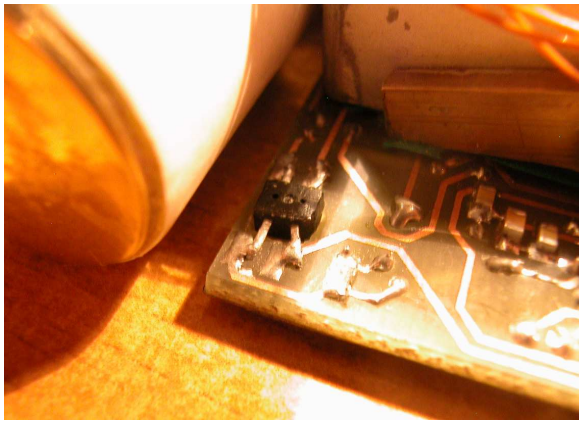


Rysunek 14: Schemat końcówki mocy

pinu mikrokontrolera. Generator sygnału PWM pracujący z częstotliwością 31KHz, wraz z kondensatorem C1 i rezystorem R1, sprawia że możliwa jest regulowana zmiana napięcia. Powoduje to zmniejszenie prądu płynącego w obwodzie i w rezultacie zmniejszenie natężenia emitowanego strumienia promieniowania IR. Im mniejsze natężenie, tym układ jest mniej czuły na kolor biały. Kod do obsługi czujników przedstawiono poniżej:

```
void init_BL(unsigned char power){ //inicjacja
        sbi(DDRD,3); //ustawienie kierunku pinu
        TCCR2A = (1<<COM2B1)|(1<<WGM21)|(1<<WGM20);
        TCCR2B = (1<<CS20);
        OCR2B = power;
}
unsigned char BL(void){ //odczytywanie stanu
        unsigned char tmp;
        tmp = (PINB & 0b11000000)>>6;
        tmp += (PINC & 0b00001100);
        return tmp;
}
void off_BL(void){ //wylaczenie diod nadawczych
        TCCR2A = 0;
        TCCR2B = 0;
        cbi(PORTD,3);
}
```

Regulacja odbywa się w bardzo wąskim zakresie. Przy odpowiednio dobranych rezystorach polaryzujących (są one inne dla różnych typów czu-



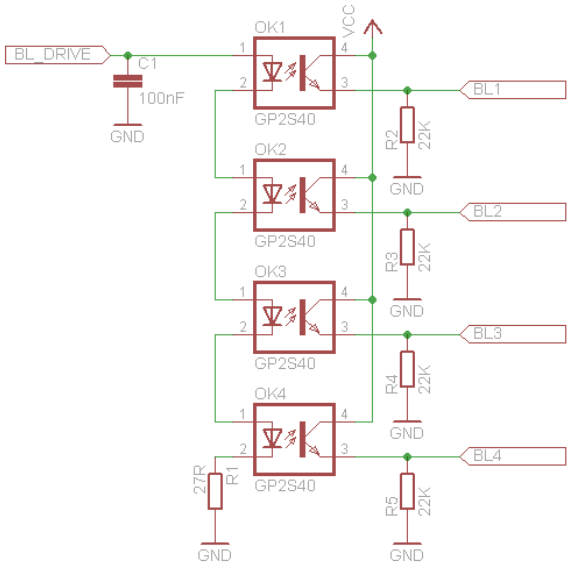
Rysunek 15: Umiejscowienie jednego z czujników białej linii

jników) w zupełności wystarczy. Niebagatelną zaletą czujnika jest niewielka ilość elementów dodatkowych i oszczędność prądu (diody połączone szeregowo). Wymaga użycia sprzętowego generatora PWM (musi być względnie szybki)

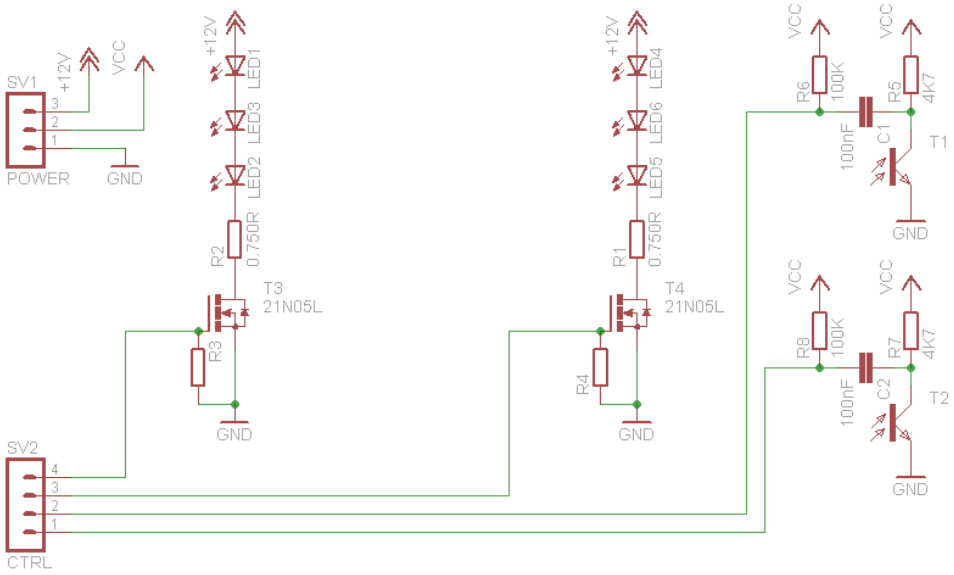
6.2 Układ optycznego detektora obiektów - dalmierze

Z racji niewielkiej przestrzeni przeznaczoną na dalmierze, zastosowano czujnik składający się z diod nadawczych i fototranzystorów. Schemat czujnika pokazano na rys.17. Wzorowano się na czujniku zastosowanym w robocie "Stealth2" którego autorem jest **Peter Waller**.

Czujnik jest dwukanałowy. Każdy kanał składa się z 3 diod nadawczych IR, załączanych za pomocą tranzystora mosfet. Ponieważ diody sterowane są impulsowo, prąd przez nie przepływający jest dość znaczny, dochodzi maksymalnie do 3A. Zastosowane diody IR - SFH416 są w stanie wytrzymać takie natężenie. Warunkiem jest zapewnienie na tyle krótkiego czasu świecenia, że struktura krzemowa nie ulegnie trwałemu zniszczeniu. Diody czujnika umieszczono w mosiężnych rurkach przylutowanych do przedniej ściany robota. Zapewnia to chłodzenie, dodatkowo oddziela fototranzystory od LED nadawczych. Jest to bardzo ważne, gdyż przy tym natężeniu światła nawet najmniejsza szpara którą mogłoby się ono przedostać do fototranzystorów, powodowała fałszowanie odczytu. Jako układ odbiorczy zastosowano zwykłe fototranzystory. Dodano też kondensator który stanowi prosty filtr górnoprzepustowy. W momencie kiedy diody nadawcze świecą pełną mocą, dokonywany jest pomiar napięcia na wyjściu czujnika. Czas zapalania LED i moment pomiaru dobrano doświadczalnie, aby zapewnić minimalny czas świecenia diod nadawczych. Diody są zapalane, następnie po 20μs startowany jest przetwornik pomiarowy. Ponieważ układ sample&hold przetwornika uruchamia się dopiero po pewnym czasie, diody świecą jeszcze



Rysunek 16: Schemat zespołu czujników białej linii



Rysunek 17: Schemat układu optycznego detektora obiektów

przez 30μs. Po wykonaniu pomiaru dane są poddawane progowaniu. Kod obsługujący sonar przedstawiono poniżej:

```
#define SONAR_NIC 99    //zwracane kiedy nie ma przeszkody
#define SONAR_R_T 1000  //progi wykrywania
#define SONAR_L_T 1000
```

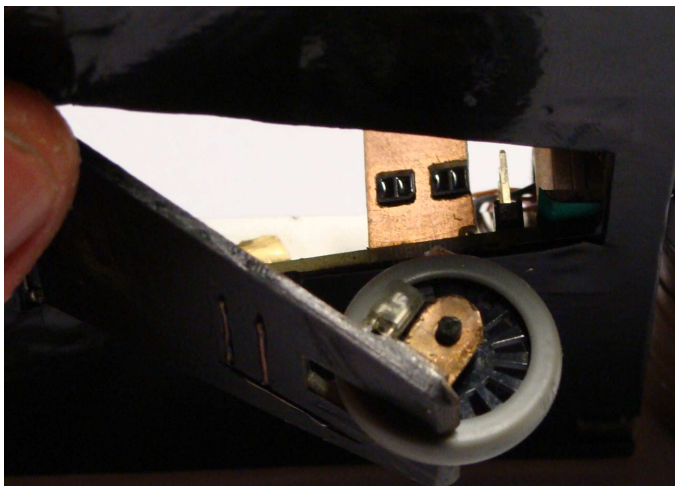
```
void sonar_ir_init(void){ //inicjacja
        cbi(PORTC,0);
        cbi(PORTC,1);
        sbi(DDRC,0);
        sbi(DDRC,1);
        TCCR0B = (1<<CS01);  //ustawienie timera
}
void timer_wait(unsigned char time){ //odmierzanie czasu
        TCNT0 = 0;
        sbi(TIFR0,OCF0A);
        OCR0A = time;
        while(bit_is_clear(TIFR0,OCF0A)){}
        return;
}
unsigned int sonar_ir_R(void){ //pomiar, prawy kanal
        ADC_ustaw_kanal(6);
        sbi(PORTC,0); //zapalanie LED IR
        timer_wait(30);
        ADC_start();  //pomiar przetwornika
        timer_wait(20);
        cbi(PORTC,0); //gaszenie LED IR
        return ADC_odczyt();
}
unsigned int sonar_ir_L(void){
        ADC_ustaw_kanal(7);
        sbi(PORTC,1);
        timer_wait(30);
        ADC_start();
        timer_wait(20);
        cbi(PORTC,1);
        return ADC_odczyt();
}
void pomiar_sonar_ir(void){ //pomiar i obrobka danych
        unsigned int Lewy, Prawy;
        Lewy = sonar_ir_L();
        Prawy = sonar_ir_R();
        if (Prawy<SONAR_R_T){
                gPomiary.DalmierzR = Prawy/128;
        }else{
                gPomiary.DalmierzR = SONAR_NIC;
        }
        if (Lewy<SONAR_L_T){
                gPomiary.DalmierzL = Lewy/128;
        }else{
```

```
                gPomiary.DalmierzL = SONAR_NIC;
        }
}
```

Do pomiaru czasu wykorzystano 8-bitowy timer0. Jego preskaler (dzielnik częstotliwości zegara systemowego) ustawiono na 8, dzięki temu przy częstotliwości taktowania 8MHz, uzyskano rozdzielczość 1μs. Zastosowanie timera zmniejsza wydłużenie się cyklu, spowodowane obsługiwanymi przerwaniami. To bardzo ważne, gdyż zbytnie wydłużenie czasu kiedy diody LED IR świecą, doprowadzi do ich zniszczenia (w czasie testów jeden komplet z hukiem wyparował, zamieniając krzemowe struktury w kupę smrodu). Istotne jest też, aby wyłączyć dalmierz w momencie kiedy program jest debugowany w systemie (OCD przez DebugWire). Zatrzymanie programu w nieodpowiednim momencie może również doprowadzić do zniszczenia czujnika. Problem ten można wyeliminować stosując zewnętrzny układ wygaszający diody. Zrezygnowano z tego z racji bardzo ograniczonej przestrzeni wewnątrz robota.

Czujnik robota zdaje egzamin, jednak może zostać oślepiony przez bardzo silne światło. Głównym atutem czujnika jest jego szybkość - 120 pomiarów na sekundę (częstotliwość można jeszcze zwiększyć) co zapewnia błyskawiczne wykrywanie przeciwnika.

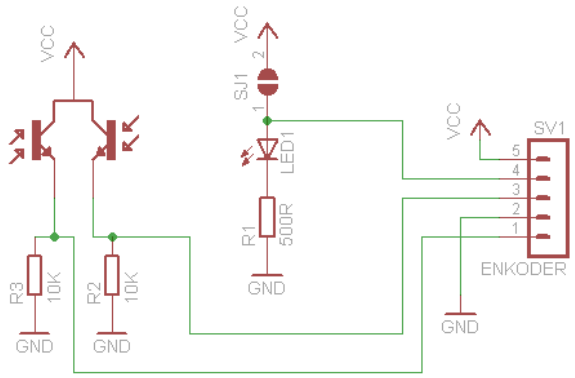
6.3 Czujnik przemieszczenia wzdłużnego



Rysunek 18: Odchylony klapoczujnik, w tle widać czujniki optyczne detektora uniesienia

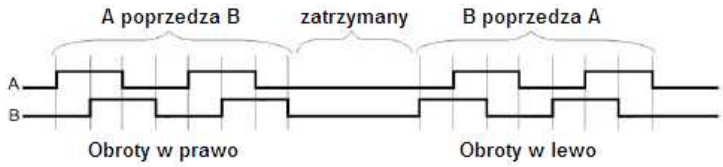
Czujnik przemieszczenia wzdłużnego (rys.18) wykonano w oparciu o elementy z myszki. Rolka (scroll) została umocowana w sposób gwarantujący

znikome opory ruchu. Ponieważ umieszczona jest na końcu klapki, zagwarantowano jej kontakt z podłożem nawet kiedy robot jest uniesiony.



Rysunek 19: Schemat czujnika przemieszczenia wzdłużnego

Schemat czujnika przedstawiono na rys.19. Wykorzystano podwójny fototranzystor umieszczony w jednej obudowie. Dioda nadawcza posiada zworkę, która umożliwia podłączenie zasilania na stałe. Opcjonalnie można kontrolować jej zasilanie poprzez linię nr.4 w złączu ENKODER. (zrezygnowano z racji oszczędności pinów mikrokontrolera i miejsca na płytce). Zasadę działania enkodera pokazano na rys.20.



Rysunek 20: Zasada kodowania kierunku obrotów enkodera

Jest to standard enkoderów kwadraturowych. Rozpoznawanie programowe zrealizowano za pomocą jednego przerwania sprzętowego. W momencie kiedy na kanale A występuje zbocze, sprawdzany jest stan kanału B. Jeśli przy opadającym zboczu A stan na B jest wysoki, lub przy rosnącym A stan B jest niski, rozpoznawany jest jeden kierunek. W przeciwnym wypadku kierunek jest odwrotny. Ponieważ warunek jest zależny od rodzaju zbocza, zastosowano dość rozbudowany podprogram przerwania. Przy inicjacji ustawiane jest rozpoznawanie tylko jednego rodzaju zbocza. Jest to przestawiane w czasie obsługi przerwania (po rosnącym przerwaniu na pewno będzie opadające itd.) Kod przedstawiono poniżej:

```
volatile register signed char enkoder asm("r2");
unsigned char zbocze = 0; //zmienne globalne
```

```
void Enkoder_init(void){ //inicjacja
        EICRA |= (1<<ISC01);
        EIFR |= (1<<INTF0);
        EIMSK |= (1<<INT0);
        sei();
}

SIGNAL (SIG_INTERRUPT0){ //przerwanie
        if (zbocze){
                if bit_is_set(PIND,4){
                        enkoder--;
                }
                else{
                        enkoder++;
                }
                cbi(EICRA,ISC00);
                zbocze = 0;
        }
        else{
                if bit_is_clear(PIND,4){
                        enkoder--;
                }
                else{
                        enkoder++;
                }
                sbi(EICRA,ISC00);
                zbocze = 1;
        }
}
```

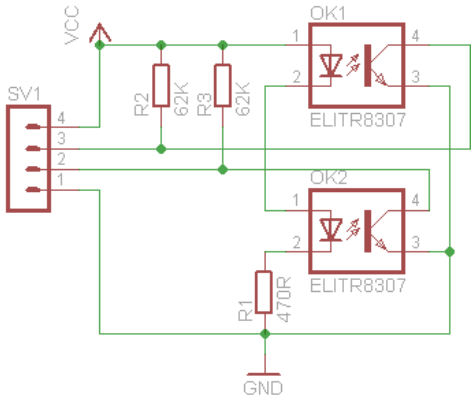
W celu przyśpieszenia przerwania zmienną "enkoder" zadeklarowano jako rejestrową. Jej wartość jest przechowywana w rejestrze R2. Zastosowane rozwiązanie nie wykrywa wszystkich zboczy - rozdzielczość enkodera można byłoby zwiększyć dwukrotnie, stosując przerwanie na obu kanałach. Byłoby to rozwiązanie nieoptymalne dla zastosowanego mikrokontrolera (drugie przerwanie zewnętrzne dzieli wyprowadzenie ze sprzętowym PWM, wykorzystanym do sterowania czujnikiem białej linii). Uzyskana rozdzielczość jest wystarczająca do algorytmów kontroli trakcji i uników.

6.4 Opto-mechaniczny układ wykrywania uniesienia

Czujnik uniesienia wykorzystuje w działaniu dwa czujniki optyczne (rys.18). Dzięki specjalnemu malowaniu pionowej blaszki, w czarne i białe pasy - uzyskano prosty i niezawodny system wykrywania uniesienia. Miedzianą blaszkę pokryto białą farbą, następnie za pomocą markera namalowano

czarne pasy (dobierając progi działania). Czujnik na wyjściu generuje trzy stany:

- Brak uniesienia
- Delikatne uniesienie, należy wyłączyć wykrywanie białej linii
- Skrajne uniesienie, niebezpieczeństwo wywrotki



Rysunek 21: Schemat czujnika uniesienia

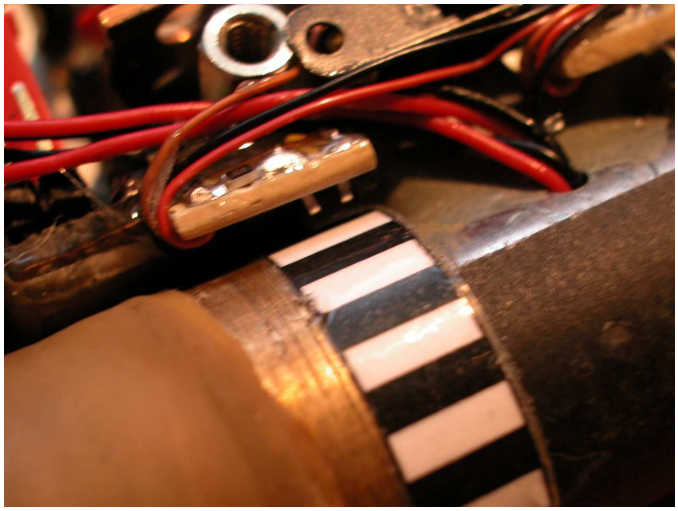
Schemat czujnika pokazano na rys.21 Zastosowano dwa czujniki ITR8307 firmy EverLight (znane też pod nazwą ELITR8307) Diody nadawcze połączono szeregowo. Ponieważ czujniki pracują w osłoniętej przestrzeni, dobrano niewielki prąd diod nadawczych. Wymusiło to zastosowanie dość dużej wartości rezystorów polaryzujących fototranzystory. Wyjścia czujnika podłączono do pinów ogólnego przeznaczenia w mikrokontrolerze. Funkcja obsługująca czujnik wygląda następująco:

```
unsigned char Lift(void){
        if(bit_is_clear(PIND,0))
                return 2;
        if(bit_is_clear(PINC,5))
                return 1;
        return 0;
}
```

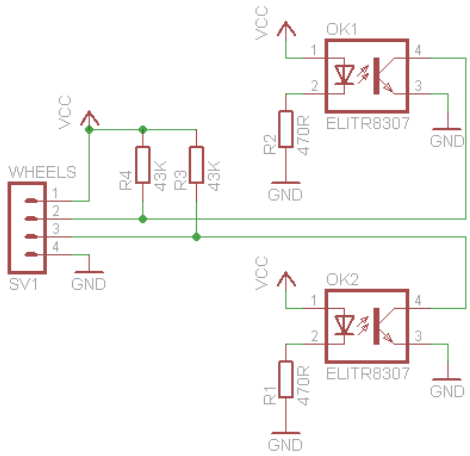
Czujniki nie wymagają inicjacji programowej, gdyż piny mikrokontrolera domyślnie ustawiane są jako wejściowe, w stanie wysokiej impedancji.

6.5 Czujniki prędkości obrotowej kół napędowych

Wygląd jednego z dwóch czujników pokazano na rys.22. Na koła naklejono pasek kodowy. Składa się on z czarno-białych obszarów. Na obwodzie koła



Rysunek 22: Mocowanie czujnika i pasek kodowy



Rysunek 23: Schemat czujnika prędkości obrotowej kół napędowych

znajduje się 16 pól czarnych i tyle samo białych. Pasek wydrukowano na drukarce laserowej i pokryto przezroczystą taśmą samoprzylepną. Schemat części elektrycznej pokazano na rys.23. Wartości rezystorów polaryzujących dobrano doświadczalnie. Wpływ oświetlenia zewnętrznego zminimalizowano przez bardzo niewielką odległość czujnika od paska kodowego (od 0.3 do 0.7mm). Wyjścia czujnika podłączono do pinów ogólnego przeznaczenia w mikrokontrolerze, wykorzystano funkcję PinChange Interrupt, jako przerwanie sprzętowe. Kod przedstawiono poniżej:

```
volatile register unsigned char poprzedni asm("r3");
unsigned char stan = 0, Lewy = 0, Prawy = 0; //zmienne globalne
```

```
void Wheels_init(void){ //inicjacja przerwan
        PCIFR = 0b00000111;       //zerowanie flag
        PCMSK2 = (1<<PCINT22)|(1<<PCINT21);
        PCICR = (1<<PCIE2);
        sei();
}


SIGNAL (SIG_PIN_CHANGE2){ //przerwanie
        stan = PIND^poprzedni;
        poprzedni = PIND;
        if(stan&0b01000000) Prawy++;
        if(stan&0b00100000) Lewy++;
}
```

Zmienną globalną "poprzedni" zadeklarowano jako zmienną przechowywaną w rejestrze R3 procesora. Powoduje to szybsze działanie funkcji, gdyż tego typu zmienne mogą być wykorzystywane w argumentach pojedynczej instrukcji procesora. Dzięki temu wyrażenie "PIND XOR poprzedni" zostanie wykonane w jednym takcie zegarowym. Pozostałe zmienne zostały prawidłowo zoptymalizowane przez kompilator, dzięki czemu kod przerwania jest bardzo szybki.

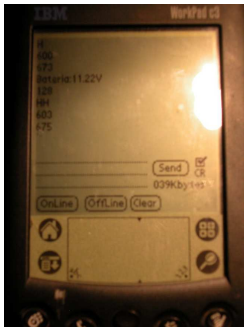
7 Oprogramowanie

Oprogramowanie mikrokontrolera przygotowano w środowisku AVRStudio 4.15 z wykorzystaniem kompilatora avr-gcc (pakiet WinAVR, wersja 20081205) skompilowany program zajmuje mniej niż 5KB pamięci FLASH i około 350 bajtów pamięci RAM. W zasadzie wystarczającym byłby mikrokontroler Atmega88.

7.1 Debugowanie

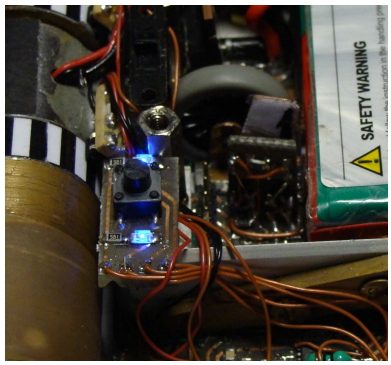
Debugowanie programu dokonywane było na dwa sposoby. Błędy w kodzie, eliminowane były za pomocą interfejsu DebugWire. Należało wyłączyć obsługę dalmierza optycznego. Zatrzymanie pracy mikrokontrolera w momencie kiedy załączone są diody nadawcze, doprowadziłoby do zniszczenia czujnika. Do kalibracji czujników, czy weryfikacji pracy automatu, wykorzystano interfejs USART. Do robota podłączono palmtopa (PALM V), który posiada sprzętowy interfejs rs232. Pełnił on rolę terminala (rys.24).

Interfejs ten zapewniał tylko komunikację od mikrokontrolera do Palma. W zupełności wystarczyło to do przeprowadzenia kalibracji czujników i weryfikacji ich działania.



Rysunek 24: Przykładowe dane z czujników wysłane przez USART

7.2 Inicjacja robota



Rysunek 25: Przycisk i LED sygnalizacyjne zaraz po uruchomieniu

Po uruchomieniu robota wykonywane są procedury inicjujące. Zapalane są obydwie diody sygnalizacyjne (rys.25). Kiedy przycisk startujący zostanie wciśnięty. Rozpoczynane jest odliczanie 5s. do rozpoczęcia walki. Jednak jeśli przycisk zostanie przytrzymany odpowiednio długo, robot przechodzi w tryb ustawienia trybu walki. W tym trybie ustalany jest preferowany tryb szukania przeciwnika. Za pomocą kolejnych kodów błyskowych wskazywane są odpowiednie tryby szukania. Puszczenie przycisków w momencie kiedy aktywny jest dany kod błyskowy, spowoduje ustawienie tego trybu i przejście do stanu oczekiwania. Dostępne jest 5 trybów szukania:

- Obrót w miejscu w lewo
- Jazda po łuku w lewo
- Jazda na wprost
- Jazda po łuku w prawo
- Obrót w miejscu w prawo

W zależności od przeciwnika, przed walką można wybrać "na szybko" odpowiedni tryb szukania.

7.3 Uniki

Dzięki zastosowaniu dodatkowego enkodera, który daje informację o rzeczywistym przemieszczeniu wzdłużnym, możliwa jest realizacja uników. W momencie kiedy robot jest w stanie ataku, czy ucieczki, sprawdzane jest przemieszczenie rzeczywiste. Jeżeli robot ma się poruszać do przodu (sterowanie dla silników dodatnie), a dwa kolejne odczyty z enkodera wskazują że robot porusza się w tył, wtedy przechodzi w stan uniku. Jeżeli odczyt z układu dalmierzy, wskazuje że przeszkoda jest bliżej z lewej strony, robot wykonuje unik przez jazdę w tył po ostrym łuku w prawo (jeśli przeszkoda jest z drugiej strony, analogicznie po łuku w lewo).

7.4 Kontrola trakcji

Robot został wyposażony w enkodery na kołach, co w połączeniu z niezależnym enkoderem na klapoczujniku, umożliwiło realizację prostego algorytmu kontroli trakcji. Zasada działania opiera się na zliczaniu impulsów. Co 200ms, lub jeśli przynajmniej jedno z kół obróciło się o 4 impulsy enkodera, sprawdzana jest liczba impulsów, jakie zostały zliczone z enkodera klapoczujnika. Jeżeli ta liczba jest mniejsza od 4, wtedy zwiększany jest współczynnik redukcji mocy (o ile nie osiągnął wartości maksymalnej). W przeciwnym wypadku jest on zerowany. Następnie, wartość tego współczynnika, jest odejmowana od sterowania (tylko jeżeli sterowanie jest dodatnie - czyli kiedy robot jedzie do przodu). Zasada działania algorytmu i wszystkie nastawy, zostały dobrane doświadczalnie.

Kod znajdujący się w pętli głównej:

```
if((TC_timer == 100)|
   (Enkoder_kolaR() > 4)|
   (Enkoder_kolaL() > 4)){
        TC_timer = 0;
        if((gSterowanie.speedL > 100)&
           (gSterowanie.speedR > 100)){
                if(Enkoder_kolaL() > 4)
                        Kontrola_trakcjiL();
                if(Enkoder_kolaR() > 4)
                        Kontrola_trakcjiR();
                }else{
                        gStan.TC_R = 0;
                        gStan.TC_L = 0;
                }
                Enkoder_TC_reset();
```

```
}
```

Funkcje algorytmu:

```
void Kontrola_trakcjiL(void){
        Enkoder_kolaL_reset();
        if (Enkoder_val() <4){
                if(gStan.TC_L < 80) gStan.TC_L += 20;
                return;
        }
        gStan.TC_L = 0;
}
void Kontrola_trakcjiR(void){
        Enkoder_kolaR_reset();
        if (Enkoder_val() <4){
                if(gStan.TC_R < 80) gStan.TC_R += 20;
                return;
        }
        gStan.TC_R = 0;
}
```

Procedura redukcji mocy, przy niskopoziomowej realizacji sterowania:

```
if(sterowanie->speedL>0){
        OCR1A = sterowanie->speedL*2 + 1 - gStan.TC_L;
        cbi(PORTD,7);
}
```

Algorytm działa poprawnie. Problem stanowią silniki, które są zbyt słabe żeby pokazać zalety tego rodzaju algorytmu. Maksymalnie zredukowaną moc, dobrano tak żeby robot utrzymał minimalny poślizg, a nie zatrzymywał silników całkowicie. Przeprowadzone testy wskazały, że utrzymanie niewielkiego poślizgu, daje lepsze rezultaty, niż zatrzymanie koła i później próba ruszenia z pełną przyczepnością.

7.5 Monitorowanie stanu akumulatora

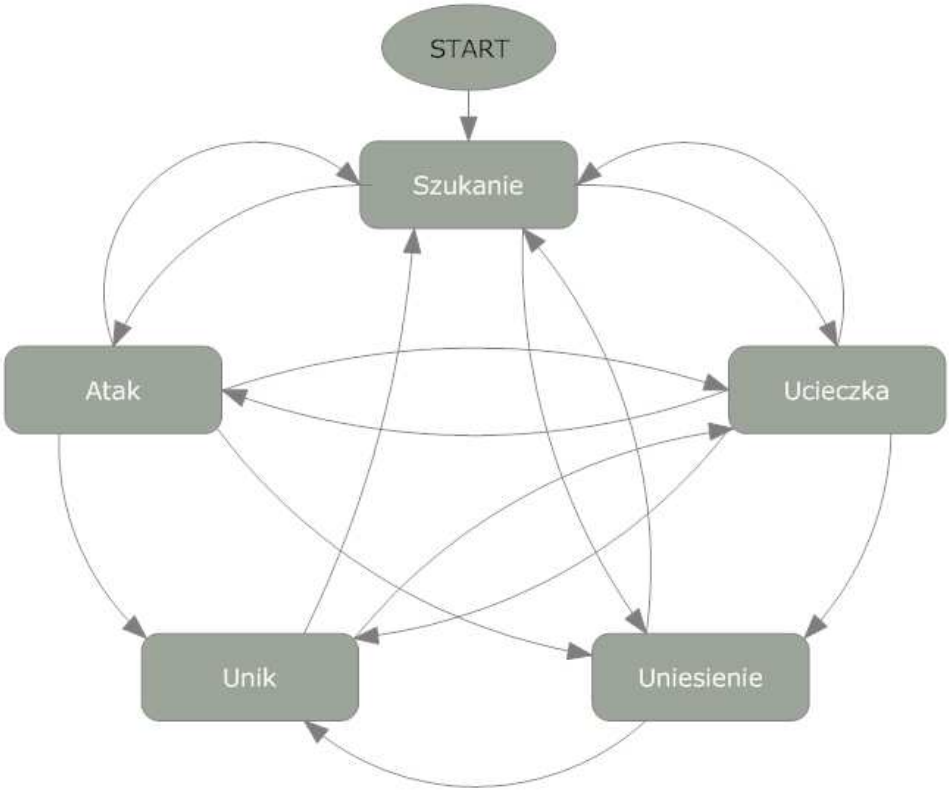
W robocie zastosowano drogi akumulator litowo-polimerowy (KOKAM, 11.1V, 450mAh, 30C). Głównym niebezpieczeństwem, jest nadmierne rozładowanie. Robot sprawdza stan baterii przy uruchomieniu. Jeśli jej napięcie wynosi mniej niż 10.5V (3.5V na ogniwo) robot przechodzi w tryb awaryjny. W trakcie walki napięcie jest mierzone co 5s. Jeżeli w trakcie walki, napięcie spadnie poniżej 9.3V (3.1V na ogniwo) robot przerywa walkę i przechodzi w tryb awaryjny. W tym trybie wyłączane są dalmierze, układ zasilania silników i czujniki białej linii. Mikrokontroler przechodzi w stan PowerDown. Przejście w tryb awaryjny, sygnalizowane jest kilkukrotnym, naprzemiennym, szybkim migotaniem LED sygnalizacyjnych.

7.6 Automat stanu

Za zachowanie robota na ringu odpowiada automat stanu. Wyszczególniono 5 odrębnych stanów:

- Szukanie - robot przemieszcza się po ringu w poszukiwaniu przeciwnika
- Atak - robot atakuje, korygując trajektorię w zależności od odczytu dalmierzy
- Ucieczka - robot ucieka z białej linii, wg. jednej z zaprogramowanych trajektorii (w zależności od tego które czujniki wykryły linię)
- Uniesienie - robot przestaje reagować na białą linię i realizuje trajektorię zależną od odczytu czujników optycznych.
- Unik - robot cofa się wg. jednej z dwóch trajektorii, wybieranych w zależności od odczytu z dalmierzy.

Uproszczony diagram stanów przedstawiono na rys.26, nie zaznaczono na nim warunków, ani sekwencji startowej itp.



Rysunek 26: Uproszczony diagram stanów

W momencie kiedy warunek przejścia do danego stanu jest spełniony w momencie kiedy robot jest jeszcze w tym stanie - wtedy w nim pozostaje. Po zakończeniu realizacji trajektorii danego stanu, automat przechodzi do stanu wyjściowego - "Szukanie". Każdy stan wiąże się z pewną trajektorią. Jest ona opisana za dwuwymiarowej tablicy. Liczba kolumn, stanowi jednocześnie liczbę odruchów elementarnych, składających się na trajektorię. W każdym wierszu (stanowiącym odruch elementarny) znajdują się informacje:

- prędkość lewego koła
- prędkość prawego koła
- czas trwania ruchu
- blokada automatu

Ostatnie pole, umożliwia zablokowanie automatu, na czas realizacji odruchu elementarnego. Zostało to wykorzystane przy stanie ucieczki, kiedy robot znajduje się na białej linii, nie reaguje na warunki przejścia do innych stanów. W momencie kiedy zjedzie z linii i zakończy zablokowany odruch elementarny, będzie dalej realizował trajektorię stanu, która sprawi że znajdzie się dalej od białej linii, albo przejdzie do dostępnych stanów, zgodnie z wynikami warunków.

Kod automatu stanu jest rozbudowany i jego najbardziej istotne elementy zostały zestawione w dodatku A.

8 Podsumowanie i wnioski

Głównym celem jaki towarzyszył procesowi projektowania i wykonania robota, było stworzenie możliwie niskiej i zwartej konstrukcji. Bardzo nisko umieszczony środek ciężkości, znajduje się bardzo blisko osi napędowej. Sprawia że robot bardzo dobrze trzyma się ringu. W momencie kiedy przód robota zostanie uniesiony, jego przyczepność wzrasta, dzięki czemu jeszcze efektywniej przepycha przeciwnika. Wysunięcie kół napędowych do tyłu, sprawia że robot przy ataku z boku jest bardzo trudny do przepchnięcia. Zamiast dać się zepchnąć zaczyna się obracać. Znaczna szybkość robota sprawia, że bardzo trudno dogonić go i zepchnąć od tyłu. Niewielkie wymiary robota, ujawniły całą masę nieznanych wcześniej problemów. Przede wszystkim, we wnętrzu robota znajduje się bardzo mała ilość wolnego miejsca. Spowodowało to że nie można było zastosować sprawdzonych dalmierzy, jak miniaturowe czujniki sharpa, czy sonary. Podpatrzenie rozwiązań z robotów tworzonych za granicą zaowocowało opracowaniem nowego typu czujnika. Zastosowane napędy dają bardzo dużą szybkość, jednak mogłyby być odrobinę mocniejsze. Ponieważ konstrukcja układu napędowego jest nierozbieralna, zanieczyszczenia dostające się pod koła, mogą doprowadzić

do zatarcia mechanizmu. Bardzo dobry efekt dało zastosowanie klapoczujnika z enkoderem. Dzięki wykrywaniu uniesienia, robot staje się niewrażliwy na "białe pługi" (kiedy robot zostaje podważony przez pług przeciwnika, może go pomylić z białą linią i pomóc sobie w przegranej). Ponadto umożliwia to wykrycie ataku bocznego i próbę ucieczki. Umieszczony na końcu klapoczujnika enkoder okazał się bardzo dobrym rozwiązaniem. W porównaniu z myszką optyczną, jest zdecydowanie prostszy i bardziej niezawodny. Co prawda nie pozwala na wykrywanie przemieszczenia bocznego, ale do zabezpieczenia przed taką sytuacją wystarczy czujnik uniesienia, lub czujniki białej linii. Dzięki prostym enkoderom na kołach, udało się zrealizować kontrolę trakcji, oraz algorytm uniku. O ile kontrola trakcji przy tak słabych silnikach nie jest widowiskowa, to możliwość wykonywania uników przydała się w czasie walk i dzięki niej robot wygrał kilka z nich.

Algorytm automatu stanu został zaczerpnięty z oprogramowania robota NoName, chciałbym podziękować **Mariuszowi Janiakowi** za udostępnienie kodu i rozjaśnienie tematu. Co ciekawe algorytm był napisany na 32bitowy mikrokontroler MC68332, po drobnych poprawkach (kwestie różnic kompilatora i zastosowanie 8bitowych zmiennych gdzie to tylko możliwe) ruszył na 8 bitowej Atmedze. Na potrzeby tego projektu kod i struktura zostały zmienione, ale doskonale pokazuje to jak mocno przenośny może być kod napisany w C.

A Kody źródłowe automatu stanu

Zawartość pliku automat.c:

```
#include "automat.h"
#include "motors.h"
#include "LED.h"

AU_Stan_str Stan;

void Automat_init(void){
        gPomiary.DalmierzL = SONAR_NIC;
        gPomiary.DalmierzR = SONAR_NIC;
        AU_ustaw_stan_szukaj(&Stan);
}
void Automat_wykonaj_krok(void){
        if ((gStan.Akcja) && (gAkcja.timer < gAkcja.Koniec)){
                gSterowanie.speedL =
                        gAkcja.tab[gAkcja.wsk][ACTION_SPEED_L];
                gSterowanie.speedR =
                        gAkcja.tab[gAkcja.wsk][ACTION_SPEED_R];
                gStan.Blokada     =
```

```
                        gAkcja.tab[gAkcja.wsk][ACTION_BLOCK];
        gAkcja.timer++;
        if (gAkcja.timer > gAkcja.tab[gAkcja.wsk][ACTION_TIME])
                        gAkcja.wsk++;
        }else{
            gSterowanie.speedL = MOTOR_STOP;
                gSterowanie.speedR = MOTOR_STOP;
            gStan.Blokada    = 0;
            gStan.Akcja    = 0;
        }
        Motor(&gSterowanie);
}
void Automat_wybor_stanu(void){
        if(gStan.Blokada == RUCH_ZABLOKOWANY)
        return;
    switch(Stan.stan)
    {
        case STAN_SZUKANIE:
        {
            if(Warunek_BL()&(Lift() == 0)) {
                        AU_ustaw_stan_ucieczka(&Stan);
                    }else if(Warunek_Lift()) {
                        AU_ustaw_stan_lift(&Stan);
                    }else if(Warunek_Atak()) {
                        AU_ustaw_stan_atak(&Stan);
                    }else if(gStan.Akcja == AKCJA_ZAKONCZONA){
                        AU_ustaw_stan_szukaj(&Stan);
                    }
                    break;
        }
        case STAN_ATAK:
        {
            if(Warunek_BL()&(Lift() == 0)) {
                        AU_ustaw_stan_ucieczka(&Stan);
                    }else if(Warunek_Unik()) {
                        AU_ustaw_stan_unik(&Stan);
                    }else if(Warunek_Lift()) {
                        AU_ustaw_stan_lift(&Stan);
                    }else if(Warunek_Atak()) {
                        AU_ustaw_stan_atak(&Stan);
                    }else if(gStan.Akcja == AKCJA_ZAKONCZONA){
                        AU_ustaw_stan_szukaj(&Stan);
                    }
                    break;
```

```
        }
        case STAN_UCIECZKA:
        {
                        if(Warunek_BL()&(Lift() == 0)) {
                                AU_ustaw_stan_ucieczka(&Stan);
                        }else if(Warunek_Lift()) {
                                AU_ustaw_stan_lift(&Stan);
                        }else if(Warunek_Unik()) {
                                AU_ustaw_stan_unik(&Stan);
                        }else if(Warunek_Atak()) {
                                AU_ustaw_stan_atak(&Stan);
                        }else if(gStan.Akcja == AKCJA_ZAKONCZONA){
                                AU_ustaw_stan_szukaj(&Stan);
                        }
                        break;
                }
                case STAN_LIFT:
                {
                        if(Warunek_Unik()) {
                                AU_ustaw_stan_unik(&Stan);
                        }else if(Warunek_Lift()) {
                                AU_ustaw_stan_lift(&Stan);
                        }else if(gStan.Akcja == AKCJA_ZAKONCZONA){
                                AU_ustaw_stan_szukaj(&Stan);
                        }
                        break;
                }
                case STAN_UNIK:
                {
                        if(Warunek_BL()&(Lift() == 0)) {
                                AU_ustaw_stan_ucieczka(&Stan);
                        }else if(Warunek_Unik()) {
                                AU_ustaw_stan_unik(&Stan);
                        }else if(gStan.Akcja == AKCJA_ZAKONCZONA){
                                AU_ustaw_stan_szukaj(&Stan);
                        }
                        break;
                }
        default:
        {
                        AU_ustaw_stan_szukaj(&Stan);
        }
}
return;
```

```
}
```

Zawartość pliku stany.c:

```
#include "stany.h"
#include "akcje.h"
#include "warunki.h"
#include "LED.h"

void AU_ustaw_stan_szukaj(AU_Stan_str *pAU_StanAktualny){
    pAU_StanAktualny->stan = STAN_SZUKANIE;
    pAU_StanAktualny->podstan = PODSTAN_NULL;
        gStan.namierzony = 0;
        if(gPomiary.Prawolewo == SZUKANIE_NORMALNE){
        Inicjuj_Akcje(Tablica_szukanie,
                      Tablica_szukanie_rozmiar);
        return;
        }
        switch(gPomiary.Prawolewo){
                case 0:{
                                Inicjuj_Akcje(Tablica_szukanie_pocz1,
                                Tablica_szukanie_rozmiar_pocz);
                                break;
                           }
                case 1:{
                                Inicjuj_Akcje(Tablica_szukanie_pocz2,
                                Tablica_szukanie_rozmiar_pocz);
                                break;
                           }
                case 2:{
                                Inicjuj_Akcje(Tablica_szukanie_pocz3,
                                Tablica_szukanie_rozmiar_pocz);
                                break;
                           }
                case 3:{
                                Inicjuj_Akcje(Tablica_szukanie_pocz4,
                                Tablica_szukanie_rozmiar_pocz);
                                break;
                           }
                case 4:{
                                Inicjuj_Akcje(Tablica_szukanie_pocz5,
                                Tablica_szukanie_rozmiar_pocz);
                                break;
                           }
        }
```

```
        gPomiary.Prawolewo = SZUKANIE_NORMALNE;
}

void AU_ustaw_stan_atak(AU_Stan_str *pAU_StanAktualny){

    pAU_StanAktualny->stan = STAN_ATAK;
    pAU_StanAktualny->podstan = PODSTAN_NULL;
        if((gPomiary.DalmierzR==SONAR_NIC)&
                                (gStan.namierzony == 1)){
        Inicjuj_Akcje(Tablica_atak1, Tablica_atak_rozmiar);
        return;
        }
        if((gPomiary.DalmierzL==SONAR_NIC)&
                                (gStan.namierzony == 1)){
        Inicjuj_Akcje(Tablica_atak5, Tablica_atak_rozmiar);
        return;
        }
        if(gPomiary.DalmierzL<gPomiary.DalmierzR){
        Inicjuj_Akcje(Tablica_atak2, Tablica_atak_rozmiar);
                gStan.namierzony = 1;
        return;
        }
        if(gPomiary.DalmierzL==gPomiary.DalmierzR){
        Inicjuj_Akcje(Tablica_atak3, Tablica_atak_rozmiar);
                gStan.namierzony = 1;
        return;
        }
        if(gPomiary.DalmierzL>gPomiary.DalmierzR){
        Inicjuj_Akcje(Tablica_atak4, Tablica_atak_rozmiar);
                gStan.namierzony = 1;
        return;
        }

}

void AU_ustaw_stan_ucieczka(AU_Stan_str *pAU_StanAktualny){

    pAU_StanAktualny->stan = STAN_UCIECZKA;
    pAU_StanAktualny->podstan = PODSTAN_NULL;

        switch(gPomiary.BL){
                case 0b0010:{
                                Inicjuj_Akcje(Tablica_ucieczka1,
                                Tablica_ucieczka_rozmiarA);
```

```
                                        break;
                                }
                        case 0b1000:{
                                        Inicjuj_Akcje(Tablica_ucieczka2,
                                        Tablica_ucieczka_rozmiarA);
                                        break;
                                }
                        case 0b0100:{}
                        case 0b1100:{
                                        Inicjuj_Akcje(Tablica_ucieczka3,
                                        Tablica_ucieczka_rozmiarB);
                                        break;
                                }
                        case 0b0001:{}
                        case 0b0011:{
                                        Inicjuj_Akcje(Tablica_ucieczka4,
                                        Tablica_ucieczka_rozmiarB);
                                        break;
                                }
                        case 0b0101:{
                                        Inicjuj_Akcje(Tablica_ucieczka5,
                                        Tablica_ucieczka_rozmiarB);
                                        break;
                                }
                        case 0b1010:{}
                        default:
                                Inicjuj_Akcje(Tablica_ucieczka6,
                                Tablica_ucieczka_rozmiarB);
                }
        return;
}

void AU_ustaw_stan_lift(AU_Stan_str *pAU_StanAktualny){

        pAU_StanAktualny->stan = STAN_LIFT;
        pAU_StanAktualny->podstan = PODSTAN_NULL;
                if (Warunek_Atak()){
                        Inicjuj_Akcje(Tablica_lift1, Tablica_lift_rozmiar);
                        return;
                }
                Inicjuj_Akcje(Tablica_lift2, Tablica_lift_rozmiar);
}

void AU_ustaw_stan_unik(AU_Stan_str *pAU_StanAktualny){
```

```
        pAU_StanAktualny->stan = STAN_UNIK;
    pAU_StanAktualny->podstan = PODSTAN_NULL;
        if(gPomiary.DalmierzL<gPomiary.DalmierzR){
                Inicjuj_Akcje(Tablica_unik1, Tablica_unik_rozmiar);
                return;
        }else{
                Inicjuj_Akcje(Tablica_unik2, Tablica_unik_rozmiar);
        }

}

void Inicjuj_Akcje(int (*tab)[ACTION_TAB_SIZE],int size)
{
  gStan.Akcja = 0;
  gAkcja.tab = tab;
  gAkcja.wsk = 0;
  gAkcja.timer = 0;
  gAkcja.Koniec = tab[size-1][ACTION_TIME];
  gStan.Akcja = 1;
}
```

Zawartość pliku warunki.c:

```
#include "warunki.h"
#include "LED.h"

unsigned char Warunek_BL(void){
        if(gPomiary.BL > 0) return 1;
        return 0;
}
unsigned char Warunek_Atak(void){
        if(gPomiary.DalmierzL<SONAR_NIC) return 1;
        if(gPomiary.DalmierzR<SONAR_NIC) return 1;
        return 0;
}
unsigned char Warunek_Lift(void){
        return Lift();
}
unsigned char Warunek_Unik(void){
        if ((gPomiary.Enkoder_v < -2) &
                (gPomiary.Enkoder_vp < -2) &
                (gSterowanie.speedL > 0) &
                (gSterowanie.speedR > 0)) {
                LED_R;
                return 1;
```

```
        }else{
                LED_r;
        }
        return 0;
}
```

Zawartość pliku akcje.h:

```
#ifndef __AKCJE_H__
#define __AKCJE_H__
//=======================

#define V3 127
#define V2 90
#define V1 50
#define V0 0

#define F100_F100(czas,blokada) {(czas),V3,V3,(blokada)}
#define R100_R100(czas,blokada) {(czas),-V3,-V3,(blokada)}
#define F100_R100(czas,blokada) {(czas),V3,-V3,(blokada)}
#define R100_F100(czas,blokada) {(czas),-V3,V3,(blokada)}
#define F0_F100(czas,blokada) {(czas),V0,V3,(blokada)}
#define F0_R100(czas,blokada) {(czas),V0,-V3,(blokada)}
#define F100_F0(czas,blokada) {(czas),V3,V0,(blokada)}
#define R100_F0(czas,blokada) {(czas),-V3,V0,(blokada)}

#define R100_R66(czas,blokada) {(czas),-V3,-V2,(blokada)}
#define R66_R100(czas,blokada) {(czas),-V2,-V3,(blokada)}
#define F66_F100(czas,blokada) {(czas),V2,V3,(blokada)}
#define F100_F66(czas,blokada) {(czas),V3,V2,(blokada)}

#define F66_F66(czas,blokada) {(czas),V2,V2,(blokada)}
#define R66_R66(czas,blokada) {(czas),-V2,-V2,(blokada)}
#define F66_R66(czas,blokada) {(czas),V2,-V2,(blokada)}
#define R66_F66(czas,blokada) {(czas),-V2,V2,(blokada)}

#define F66_F33(czas,blokada) {(czas),V2,V1,(blokada)}
#define F66_R33(czas,blokada) {(czas),V2,-V1,(blokada)}
#define F66_F33(czas,blokada) {(czas),V2,V1,(blokada)}
#define R66_F33(czas,blokada) {(czas),-V2,V1,(blokada)}

#define F100_F33(czas,blokada) {(czas),V3,V1,(blokada)}
#define F33_F100(czas,blokada) {(czas),V1,V3,(blokada)}
#define F100_R33(czas,blokada) {(czas),V3,-V1,(blokada)}
#define R33_F100(czas,blokada) {(czas),-V1,V3,(blokada)}
```

```
#define R100_R33(czas,blokada) {(czas),-V3,-V1,(blokada)}
#define R33_R100(czas,blokada) {(czas),-V1,-V3,(blokada)}

#define F33_F33(czas,blokada) {(czas),V1,V1,(blokada)}
#define R33_R33(czas,blokada) {(czas),-V1,-V1,(blokada)}
#define F33_R33(czas,blokada) {(czas),V1,-V1,(blokada)}
#define R33_F33(czas,blokada) {(czas),-V1,V1,(blokada)}
#define stop(czas,blokada) {(czas),V0,V0,(blokada)}

#define Tablica_szukanie_rozmiar 4
int Tablica_szukanie[4][ACTION_TAB_SIZE]=
{F66_F33(100,0),F66_R33(400,0),

#define Tablica_szukanie_rozmiar_pocz 1
int Tablica_szukanie_pocz1[1][ACTION_TAB_SIZE]={R66_F66(500,0)};
int Tablica_szukanie_pocz2[1][ACTION_TAB_SIZE]={F33_F100(500,0)};
int Tablica_szukanie_pocz3[1][ACTION_TAB_SIZE]={F66_R66(500,0)};
int Tablica_szukanie_pocz4[1][ACTION_TAB_SIZE]={F100_F33(500,0)};
int Tablica_szukanie_pocz5[1][ACTION_TAB_SIZE]={F100_F100(300,0)};

#define Tablica_atak_rozmiar 1
int Tablica_atak1[1][ACTION_TAB_SIZE]={R33_F100(300,0)};
int Tablica_atak2[1][ACTION_TAB_SIZE]={F33_F100(100,0)};
int Tablica_atak3[1][ACTION_TAB_SIZE]={F100_F100(100,0)};
int Tablica_atak4[1][ACTION_TAB_SIZE]={F100_F33(100,0)};
int Tablica_atak5[1][ACTION_TAB_SIZE]={F100_R33(300,0)};

#define Tablica_lift_rozmiar 1
int Tablica_lift1[1][ACTION_TAB_SIZE]={F100_F100(100,0)};
int Tablica_lift2[1][ACTION_TAB_SIZE]={R100_F100(100,0)};

#define Tablica_unik_rozmiar 1
int Tablica_unik1[1][ACTION_TAB_SIZE]={R100_R33(300,0)};
int Tablica_unik2[1][ACTION_TAB_SIZE]={R33_R100(300,0)};

#define Tablica_ucieczka_rozmiarA 3
int Tablica_ucieczka1[3][ACTION_TAB_SIZE]={R66_R100(100,1),
                        R100_R100(200,0),F100_R100(400,0)};
int Tablica_ucieczka2[3][ACTION_TAB_SIZE]={R100_R66(100,1),
                        R100_R100(200,0),R100_F100(400,0)};
#define Tablica_ucieczka_rozmiarB 2
int Tablica_ucieczka3[2][ACTION_TAB_SIZE]={F100_F66(100,1),
                        F100_F66(400,0)};
```

```
int Tablica_ucieczka4[2][ACTION_TAB_SIZE]={F66_F100(100,1),
                                           F66_F100(400,0)};
int Tablica_ucieczka5[2][ACTION_TAB_SIZE]={F100_F100(100,1),
                                           F100_R100(400,0)};
int Tablica_ucieczka6[2][ACTION_TAB_SIZE]={R100_R100(100,1),
                                           R100_F100(300,0)};
//=========================
#endif
```