Politechnika Poznańska

Dokumentacja techniczno-rozruchowa

PAPAJ

robot typu line-follower

Autorzy:
Łukasz Kosiak (c)
Łukasz Królik
Piotr Kurzawa
Patryk Łapiezo
Sebastian Pawlak
Bartosz Sławianowski
Kacper Stępień
Pedro Brêda

Prowadzący:
dr. inż. Rafał Klaus

5 czerwca 2015

Spis treści

1 Opis

Papaj jest robotem typu line-follower stworzonym na potrzeby corocznego konkursu RoboDay na Politechnice Poznańskiej, wykonanym przez zdolnych i dumnych studentów z grupy I4.2.

Nasze urządzenie jest klasycznym przykładem line-followera. Posiada on napęd na tylne koła (każde z nich posiada własny silnik), jest wyposażony w ślizgacze w przedniej części robota oraz 7 czujników linii. Całość jest zasilana popularnymi „paluszkami", co przekłada się na niską cenę.

Papaj został sensacyjnym zwycięzcą RoboDay 2015 w kategorii line-follower zostawiając w tyle droższe i bardziej rozbudowane konstrukcje.

2 Nazwa

Robocza nazwa Papaja brzmiała „Pommes frites" od napisu, który znajdował się na pudełku będącym miejscem jego przechowywania, jednak z powodu niedawnych obchodów 95-rocznicy urodzin Wielkiego Polaka Jana Pawła II postanowiliśmy nazwać naszego robota na jego cześć.

Dołożyliśmy tym samym swoją cegiełkę do rozpowszechniania jedynej słusznej prawdy o naszym papieżu.

3 Budżet

Jednym z kryteriów, które przyjęliśmy w trakcie projektowania robota była jego cena, która nie powinna przekroczyć 30 zł/os. W związku z tym staraliśmy się maksymalnie ciąć koszty, szukając u siebie i wśród znajomych części, które mogłyby przydać się do zbudowania robota - szczególnie koncentrując się na silniku i zasilaniu. Niestety, zarówno zasilanie, jak i silnik okazały się zbyt mało wydajne i zostaliśmy zmuszeni do zakupu nowych.

Zdecydowaliśmy się także na wspieranie lokalnego przemysłu elektronicznego, kupując części w sklepach elektronicznych w całym Poznaniu, zamiast korzystając z ofert dostępnych w internecie. Dzięki temu udało nam się oszczędzić na ewentualnych kosztach wysyłki.

Nie byliśmy w stanie wycenić niektórych elementów, które miały niebagatelny wpływ na Papaja. Mamy tu na myśli szczególnie korki do wina, których wartość niełatwo ustalić bez ich powiązania z butelkami od wina (niekoniecznie pustymi).

Łączny koszt robota wyniósł 168,6 zł, co w przeliczeniu na osobę daje niewiele ponad 24 polskich nowych złotych. Jest to naszym zdaniem bardzo udany wynik.

Tabela 1: Elementy wykorzystane w Papaju (w zł)

Nazwa	Cena	Ilość	Koszt
środek drobnokrystaliczny B327	8	1	8
laminat jednostronny 190x3	15	1	15
LEDy 3mm	0,2	3	0,6
sterownik silnika L293D	11	1	11
potencjometr drutowy	4,9	1	4,9
gniazdo jednorzędowe BLS-09	0,45	2	0,9
BL-T piny do gniazd BLS	0,1	24	2,4
gniazdo jednorzędowe BLS-02	0,1	2	0,2
podstawka DIL-28	0,5	1	0,5
zestaw przewodów	7	1	7
płytka prototypowa SD12NB	14	1	14
listwa dwurzędowa PLD40	1,5	1	1,5
stabilizator napięcia L-7805C	1,9	1	1,9
transoptory CNY70	3	7	21
rezonator kwarcowy	1,8	1	1,8
dławik 12uH	0,2	1	0,2
LEDy 5mm	0,4	5	2
silnik + koło	17	2	34
listwa jednorzędowa PLS-40	1,4	2	2,8
papier kredowy	0,6	1	0,6
zestaw 8 baterii AA	15	1	15
komparatory	0,7	2	1,4
koszyki	2,5	2	5
ATmega8A-PU	6	1	6
łącznie			157,7

Tabela 2: Wykorzystane rezystory (w zł)

Nazwa	Koszt	Ilość	Suma
metaloxid 150Ω	0,2	3	0,6
metaloxid 100kΩ	0,2	7	1,4
węglowy 10kΩ	0,3	8	2,4
węglowy 240Ω	0,3	7	2,1
łącznie			6,5

Tabela 3: Wykorzystane kondensatory (w zł)

Nazwa	Koszt	Ilość	Suma
ceram. dyskowy 100 nF	0,2	12	2,4
elektrolityczny 10 uF	0,5	2	1
elektrolityczny 47 uF	0,2	2	0,4
elektrolityczny 22 uF	0,2	1	0,2
ceramiczny 22 pF	0,2	2	0,4
łącznie			4,4

4 Schemat ideowy

4.1 Jednostka centralna

Sercem robota jest mikrokontroler ATmega8A-PU. Zdecydowaliśmy się na niego ze względu na cenę, łatwość programowania oraz jego popularność, co miało szczególnie znaczenie podczas problemów z działaniem robota. Doskonała dokumentacja do ATmegi i wiele rozwiązań powszechnie dostępnych w Sieci zdecydowanie ułatwiło nam okiełznanie Papaja i zabezpieczenie go przed niepożądanym działaniem. Poza tym uważamy, że Arduino jest dla słabych i jego wybór mógłby być drogą na skróty, co uwłaszczałoby naszej godności, honorowi i rozumowi człowieka.

Utwierdziliśmy się w przekonaniu, że nasza decyzja była słuszna, gdy okazało się, że naszym wyborem zainspirowało się wiele grup występujących w konkursie RoboDay. Jedna z nich nawet wylądowała na podium, zatem możemy z całą pewnością stwierdzić, że jest to też nasz sukces. Można nawet zaryzykować stwierdzenie, że de facto zajęliśmy dwa miejsca na podium, co wprawia nas w zadumę i zachęca do dalszej pracy.

4.2 Zasilanie

Zasilanie było zdecydowanie największym wyzwaniem podczas budowy Papaja. Pod uwagę były brane najróżniejsze opcje, łącznie z wykorzystaniem używanych akumulatorków pochodzących z telefonów marki Nokia. Zastosowane rozwiązanie, które walnie przyczyniło się do naszego zwycięstwa, przyszło do nas z najmniej spodziewanej strony - sklepu niejakiego Gembary, który zasugerował nam rozwiązanie „tanio i tanio".

Papaj zasilany jest ośmioma popularnymi paluszkami (baterie AA) połączonymi szeregowo, co zdecydowanie zmniejszyło koszty, zapewniało większą elastyczność i dawało nam całe 12V do wykorzystania. Rozwiązanie to ma jednak dużą wadę - znacząco zwiększa masę robota, co w połączeniu z umiejscowieniem baterii w tylnej części robota powodowało, że Papaj podnosił się i rwał ku radości całego zespołu.

W celu stabilizowania napięcia dla elektroniki na poziomie 5V zastosowany został stabilizator L-7805C i dał radę. Silniki natomiast delektowały się pełnym napięciem z baterii wynoszącym 12V.

4.3 Czujniki odbiciowe

Papaj w celu wywęszenia linii korzysta z 7 transoptorów CNY70 - ustaliliśmy tę liczbę doświadczalnie metodą prób i błędów. Poza tym liczba 7 jest liczbą szczęśliwą i w naszym przypadku sprawdziło się to w stu procentach.

Osobną kwestią był sposób ich połączenia z mikrokontrolerem. Na początku myśleliśmy nad rozwiązaniem programowym, tj. podłączenie sygnału z czujników bezpośrednio do wejść ADC. To rozwiązanie jednak zostało szybko porzucone z następujących powodów:

- ATmega8A-PU ma tylko 6 wejść ADC,
- programowanie ADC na ATmegach jest niezbyt ciekawe,
- nie chciało się nam.

Stąd też podjeliśmy decyzję o wykorzystaniu rozwiązania sprzętowego, czyli podłączenia transoptorów do zewnętrznych komparatorów (odpowiedzialnych za zamianę sygnału analogowego na cyfrowy), które z kolei są podłączone do wejść cyfrowych ATmegi.

Ponieważ warunki oświetleniowe w korytarzu BT nie należą do specjalnie dobrych, niezbędne było użycie potencjometru, dzięki któremu poprzez zmianę napięcia odniesienia ustalaliśmy według potrzeby próg decyzji (tj. przejścia między sygnałem 0 i 1). Miało to szczególne znaczenie podczas zawodów, gdyż na trasie białe nie było białe, a czarne też nie było czarne.

5 Mechanika

5.1 Płytka

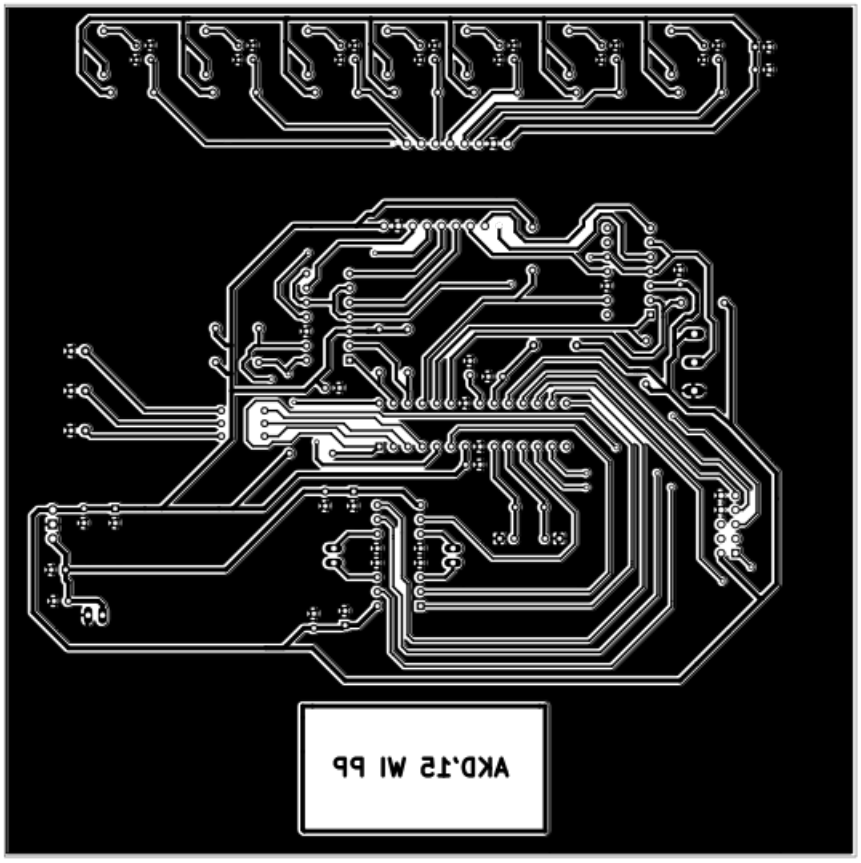
Centralnym elementem, łączącym komponenty robota w całość jest drukowana płytka PCB (Printed Circuit Board). Jej ścieżki zastąpić miały dziesiątki kabli i przewodów łączących elementy. Proces budowy takiej płytki nie jest trudny i skomplikowany, pod warunkiem oczywiście, że człowiek wykona swoją pracę starannie, dokładnie i z pełnym oddaniem.

5.1.1 Projektowanie

Płytkę należało zaprojektować w odpowiednio dla tego celu stworzonym programie. Początkowo nasza płytka była tworzona w słynnym EAGLE'u, ostatecznie jednak porzuciliśmy go na rzecz KiCADa, ponieważ jest to narzędzie open-source, a takie oprogramowanie jest szczególnie bliskie naszemu sercu. Niestety, mimo naszej niechęci do zamkniętego oprogramowania nadal byliśmy zmuszeni skorzystać z systemu operacyjnego Windows.

Projekt ten musiał być wykonany w odbiciu lustrzanym, tak, aby po odbiciu obrazu na płytkę, była ona w prawidłowym położeniu. Aby ułatwić sobie nieco sprawę, umieściliśmy na płytce napis „AKD'15 WI PP" ku chwale Akademii Kreatywnego Działania dr Rafała Klausa.

Na wyjściu programu otrzymaliśmy jednostronną płytkę PCB. Niestety sieć połączeń nie zapewniała możliwości poprowadzenia naraz wszystkich połączeń bezkolizyjnie, dlatego też potrzebne było poprowadzenie paru połączeń dodatkowymi kablami. Dzięki temu udało się uniknąć konieczności projektowania znacznie bardziej skomplikowanej konstrukcyjnie płytki dwustronnej.



Rysunek 1: Projekt płytki

5.1.2 Drukowanie

Profesjonaliści biegli w konstruowaniu płytek dysponują sprzętem, który umożliwia konstruowanie płytek o wielu warstwach. Sprzętem takim dysponuje również nasza politechnika, lecz jak się okazało, korzystanie z podobnego sprzętu zostało zakazane. Zostaliśmy zobligowani do metody chałupniczej, tzw. żelazkowej, profesjonalnie nazywaną metodą termotransferową. Drukowanie postępowało w następujących etapach:

- Wydrukować drukarką laserową, w możliwie jak najlepszej jakości, obraz płytki powstałego w wyniku realizacji punktu 1. Preferowanym typem papieru jest tzw. papier kredowy, którego przewaga nad zwykłym papierem jest taka, że jego powierzchnia jest bardzo śliska, dzięki czemu słabiej doń przywiera toner.

- Dociąć laminat do rozmiaru wydrukowanego obrazu. Jeśli nie jest on pierwszy raz drukowany, należy go bardzo dokładnie umyć z resztek toneru oraz innych zabrudzeń.

- Przyłożyć papier kredowy do laminatu i dokładnie nałożyć papier, brzegi zaś kartki nie będące zadrukowane tonerem, użyć do przyklejenia do kartki. Zwrócić uwagę na fałdki papieru, brudy oraz inne podobne defekty.

- Ogrzewać płytki rozgrzanym żelazkiem. Ma ono być możliwie jak najcieplejsze. Dokładnie zgrzać całą powierzchnię kartki, zwracając również uwagę na brzegi kartki, zazwyczaj traktowane po macoszemu. Czas trwania kroku powinien wynosić ok. 15-20 minut. W wyniku okazać się ma, że cały toner został odklejony z papieru i odbił się na miedzianej stronie laminatu.

- Włożyć płytkę do wody. Można dodać parę kropel detergentu. Płytkę trzymać w wodzie przez około 15 minut - dzięki temu papier zostanie dobrze odmoczony i będzie lepiej odchodzić od tonera. Po odmoczeniu płytki odkleić taśmę klejącą i zdjąć kartkę. Jeśli w wyniku tego nie odejdzie większa część obrazu, można przejść do kolejnego punktu, w przeciwnym razie należy powtórzyć całą procedurę powtórnie. Nasza grupa była zmuszona do trzykrotnego drukowania płytki na nowo. Nie należy zatem zrażać się niepowodzeniami, bo jest to klucz do sukcesu.

- Ostrożnie, z wyczuciem odpowiednim ścierać palcem resztki papieru - TYLKO W MIEJSCACH KTÓRE NIE SĄ POKRYTE CZARNYM TONEREM. Ułatwieniem jest z pewnością to, że białe miejsca nie są do niczego przyklejone, dzięki czemu teoretycznie łatwiej odchodzą. Należy co jakiś czas wkładać ponownie płytkę do wody, z powodu, że woda co jakiś czas znika z powierzchni płytki - głównie przez kontakt z dłońmi oraz w wyniku parowania. Należy uważnie odsłonić ścieżki, tak, że będą one błyszczeć miedzią, z której w rzeczywistości są zbudowane. Jest to najżmudniejsza część metody termotransferowej (nam zajęła około 2 godzin).

Metoda ta jest przyczyną bólu rąk, dlatego też ważne jest by działać w zespole oraz wykazać się kreatywnością w konstrukcji narzędzi, np. małego rylca do zdzierania resztek papieru. Pomocą w takim wypadku jest Sz. P. Antonio Vivaldi, którego muzyka pomogła nam przejść ten uciążliwy proces. Serdecznie polecamy jego utwory młodym elektrotechnikom.

5.1.3 Trawienie

Należy znaleźć odpowiednie naczynie, które umożliwiało zanurzenie w nim płytki w całości. Naczynie winno mieć odpowiednie parametry mechaniczno-chemiczne, takie jak: brak zewnętrznych otworów umożliwiające dyfuzję płynu trawiącego z otoczeniem (np. dziury) czy odporność na rozpuszczenie.

Nalewamy do naczynia wody o temperaturze około 50 stopni. Do tej wody wsypujemy proszek wytrawiacza i mieszamy (nie palcem, ale szczoteczka do zębów się nada). Wkładamy laminat do kwasu i czekamy patrząc co się dzieję, można też delikatnie przecierać powierzchnię szczoteczką do zębów. Woda zacznie się robić turkusowa (utleniona miedź). Jeśli temperatura roztworu będzie spadać, można ją podgrzać w garnku, a następnie znowu zalać laminat.

Zwykle trawienie trwa od 15-30 minut, w zależności od stężenia kwasu i temperatury. Kiedy wytrawi się miedź dookoła farby i wydaje nam się że jest gotowa, wyciągamy płytkę z kwasu. Następnie delikatnie myjemy wacikiem namoczonym w rozpuszczalniku farbę. Naszym oczom ukaże się piękna ścieżka z miedzi. Myjemy ją mydłem z rozpuszczalnika i suszymy.

5.1.4 Wiercenie i lutowanie

Po operacji trawienia płytka była gotowa do wiercenia. Została ona zatem potraktowana wiertłem 0,7 mm umieszczonym w wiertarce udostępnionej nam przez naszego mentora, dr Klausa.

Ponieważ wiercenie przebiegło bez większych problemów, natychmiast podjęliśmy się lutowania elementów na płytce. Najpierw płytka została posmarowana roztworem denaturatu i kalafonii - dzięki temu rozwiązaniu cyna łatwiej przylega do płytki. Do lutowania użyliśmy ambitnego sprzętu znanym powszechnie jako stacja lutownicza o regulowanej temperaturze z wykorzystaniem cyny 0,8 mm.

Teoretycznie lutowanie powinien ułatwić fakt, że mieliśmy do czynienia wyłącznie z elementami przelotowymi (tj. posiadającymi przewody, które są potem wkładane w otwory przelotowe na płytce drukowanej i lutowane), jednak spotkaliśmy się z paroma istotnymi problemami:

- masa zworek (niepożądanych zwarć pomiędzy ścieżkami), które trzeba było usunąć;
- zimne luty (które mogły powodować przerwanie obwodu i wymagały poprawy),
- wszystkie ścieżki trzeba było mozolnie testować miernikiem,

- gorąca kalafonia nam się wylała, co spowodowało, że dostaliśmy tylko 4.5 za wygląd Papaja;
- brak doświadczenia, co skutecznie wydłużyło prace.

Z powyższych powodów koszmar zwany lutowaniem skończył się po mniej więcej dwóch dniach. Jeżeli tak powstaje Chocapic, to współczujemy firmie Nestlé.

Niemniej było to dla nas miłe doświadczenie i uważamy, że było warto, bo człowiek po takiej działalności czuje dobrze.

5.1.5 Uwagi

Często zdarza się, że palec producenta płytki jest zbyt szorstki (tak że zrywa większe połacie tonera) lub jego oko jest na tyle nieuważne, że nie dostrzegł że w którymś miejscu płytki ścieżka jest pokryta papierem. Nie jest to powód do tego, aby ulec załamaniu nerwowemu oraz innym objawom głębokiej depresji.

W przypadku zdarcia czarnych elementów obrazu płytki, należy zaopatrzyć się w możliwie jak najcieńszy flamaster, o właściwościach zapewniających odporność na wodę. Właściwość ta daje ogromne szanse na to, że również ów tusz oprze się wytrawiaczowi. Należy wówczas dorysować zerwane części obrazu. W przypadku jednak, gdyby metoda ta nie pomogła, należy poprowadzić kabel zwykły między brakującymi ścieżkami.

Innym defektem jest połączenie się ścieżek, w wyniku niedbałego ściągnięcia papieru z powierzchni płytki. Wówczas może dojść do zwarcia, czyli sytuacji, gdzie przewody nie stykają się w kontrolowany sposób, a raczej gdzieś pośrodku. Należy problem ten rozwiązać poprzez przecięcie ścieżek nożem.

5.2 Układ jezdny

Napęd robota składa się z dwóch silniczków. Ich działanie opiera się na metodzie różnicowej, stosowanej chociażby w czołgach. Każde koło jest oddzielnie napędzane i skręt wynika z tego, że jedno koło kręci się szybciej od drugiego. W tym celu został wykorzystany układ L293, który miał na celu sterowanie pracą silników. Regulacja prędkości obrotowej kół odbywa się poprzez PWM (regulację współczynników wypełnienia).

Ruch kółek czerpie się z działania silniczków elektrycznych, których szybkość jest zależna od ilości napięcia, jakie zostanie dostarczone na styki. Konieczną procedurą było zatem sprawienie, że ruch silniczka miałby wpływ na ruch kółka. Należało skonstruować jakiś mechanizm przeniesienia jednego ruchu na drugi.

„Należało" - słowo to padło nieprzypadkowo, ponieważ zaniechaliśmy konstrukcji własnej roboty. Byłby to mechanizm niedoskonały, który znacznie spowolniłby robota z powodu niedoskonałości technicznych układu przełożenia. Z tego też powodu postanowiliśmy zainwestować więcej pieniędzy w silniczki, które miało gotową zębatkę dopasowaną do dołączonego kółka. Układ ten bardzo dobrze się sprawdził, nie „gubił" kroków i dzięki temu robot nasz mógł poruszać się tempem bardzo szybkim.

Najprostsza jednak wersja napędu opiera się na przekładni pasowej zbudowanej z popularnej gumki recepturki. Daje ona rezultaty, jest do tego opcją budżetową i ekonomiczną, jednakże ma mały współczynnik tarcia statycznego i nie może osiągnąć zachwycającej prędkości. Rozwiązanie dylematu prędkość vs. koszty zostawiamy Czytelnikowi, którego bardzo serdecznie pozdrawiamy. Osoby posiadające zabawki mogą się również postarać o sprawdzenie, czy z ich komponentów nie dałoby się stworzyć napędu. Przydatne byłby wszelkiego rodzaju zębatki. Ważna jest kreatywność, otwarty umysł i wiara we własne możliwości.

6 Algorytm

Algorytm opiera się na zasadzie działania regulatora PID. W każdej iteracji (około 3000 na sekundę) program sprawdza, który najbardziej skrajny czujnik wykrywa linię. Każdemu czujnikowi przypisane są odpowiednie wagi - po prawej stronie robota są dodatnie, a po lewej ujemne. Następnie wyliczane są wartości członów:

- proporcjonalnego (bezpośrednio wskazującego na kierunek, w którym znajduje się linia),
- różniczkującego (odpowiedzialnego za nagłą zmianę kierunku jazdy),
- całkującego (odpowiedzialnego za stopniowe zmniejszanie promienia skrętu w wypadku pokonywania długiego zakrętu).

Ich suma podstawiana jest do zmiennej *steeringValue*, której wartość następnie zmniejsza/zwiększa się tak, aby mieściła się w przedziale (-1000; 1000). Jeśli jest ujemna (czyli robot powinien skręcać w lewo), ustawiana jest maksymalna prędkość prawego koła, a prędkość lewego jest pomniejszana o wartość bezwzględną *steeringValue*. Jeśli jest dodatnia (czyli robot powinien skręcać w prawo), ustawiana jest maksymalna prędkość lewego koła, a prędkość prawego jest pomniejszana o wartość bezwzględną *steeringValue*.

Przy bardzo ostrych skrętach robot wyjeżdża poza linię (stan zerowy wszystkich czujników). W takim wypadku, aby przyspieszyć jego powrót (który normalnie zostałby również wykonany, dzięki członowi całkującemu), tymczasowo ignorowana jest wartość *steeringValue* i uruchomiona zostaje procedura przywracania robota na trasę, poprzez wykonanie gwałtownego skrętu w stronę ostatniego czujnika, pod którym wcześniej wykryta była linia, tj. w wypadku ostrego skrętu w prawo prędkość lewego koła jest ustawiana na maksymalną, natomiast prawe koło jest przez krótki okres wycofywane, co jeszcze bardziej zmniejsza promień skrętu. Po ponownym wykryciu linii program powraca do normalnej pracy.

Prędkość kół ustawiana jest za pomocą metod *setLeftMotorPwm* oraz *setRightMotorPwm*, przyjmujących wartości z zakresu od 0 do 1000 i przekształcających je liniowo do przedziału opisywanego przez stałe *lmin*, *lmax* lub *rmin* oraz

rmax, które definiują minimalne oraz maksymalne wypełnienie sygnału PWM na wyjściu do odpowiedniego silnika. Przykładowo przy ustawieniu:

```
const int lmin = 0;
const int rmin = 0;
const int rmax = 380;
const int lmax = 400;
```

wywołanie funkcji:

```
setLeftMotorPwm(900);
```

spowodowałoby ustawienie wysokiej prędkości dla lewego silnika (wypełnienie PWM = 360).

7 Podziękowania

Na wstępie chcielibyśmy podziękować naszemu prowadzącemu, dr. inż. Rafałowi Klausowi, za nieocenione wsparcie i wkład w budowę naszego robota. Jesteśmy dumni ze współpracy z dr. Klausem i mamy cichą nadzieję, że również on jest z naszej pracy dumny.

Dziękujemy także Maciejowi Uniejewskiemu i Mateuszowi Sarbinowskiemu, którzy wytykając nam brak ambicji i odrzucając naszą propozycję współpracy, zachęcili nas do tytanicznej pracy zakończonej pełnym sukcesem. Mamy nadzieję, że taka sytuacja w przyszłości się nie powtórzy - choć obiecujemy, że tego Wam nigdy nie zapomnimy. Z litości nie będziemy wspominać, jakie pozycje w konkursie zajęły grupy, do których dwie powyższe jednostki należały.

Nie możemy zapomnieć również o Janie Pawle II, który dawał nam inspirację i chęci do tworzenia robota, pomimo wszelkich przeciwności losu.

Na koniec pragniemy serdecznie podziękować grupie tworzącej robota „Mały, ale wariat" z Aleksandrą Kobus na czele. Doprawdy rzadko zdarza się sytuacja, aby grupa, która sama tworzyła trasę finalną i miała przywilej testowania swojego robota na swojej trasie parę dni przed zawodami, zawody te przegrała. Szczerze gratulujemy.

8 Kod źródłowy

```
#include </io.h>
#include <util/delay.h>

/*************************************************************/
//                                  Global variables
/*************************************************************/
int k[7] = {1000, 400, 200, 0, -200, -400, -1000};
int intPartBoundaries[7] = {450, 350, 250, 0, -250, -350, -450};

/*
Kp = Proptional Constant.
Ki = Integral Constant.
Kd = Derivative Constant.
err = Expected Output - Actual Output ie. error;
int  = int from previous loop + err; ( i.e. integral error )
der  = err - err from previous loop; ( i.e. differential error)
dt = execution time of loop.
*/

float kP = 2;
float kD = 1;
float kI = 4;

float err;
float integralError;
float differentialError;

const float dt = 0.0003;
const int brakeForce = 600;

float diffPart = 0;
float intPart = 0;
int propPart = 0;

int current_read = 0;
int previous_read = 0;


/**
 * Global variable for storing the binary input from sensors.
 * The bits are as following:
 * 2^7 = none
 * 2^6 = most left sensor
 * 2^5 = second most left
```

```
 * 2^0 = most right
 *
 * bit = 1 -> Sensor is above the line
 * bit = 0 -> Sensor is not above the line
 */
int sensors = 0;
int activeSensor = 3;
int lastKnownNonCenterSensor = 3;

int isSensorDetected;

/*************************************************************/
//                                  Functions declarations
/*************************************************************/

void initialize(void);
inline void updateSensors(void);
inline int  getSteeringValue(void);

inline void computeP(void);
inline void computeI(void);
inline void computeD(void);

/**
 * Changes the speed of the motor.
 * Value is between 0 and 1000 (1000 is a max speed)
 *
 */
void setLeftMotorPwm(int value);
void setRightMotorPwm(int value);


/*************************************************************/
//                                                      Main
/*************************************************************/

int main(void)
{
        initialize();

        int steeringValue = 0;
        activeSensor = 3;
        isSensorDetected=1;

        while(1)
        {
```

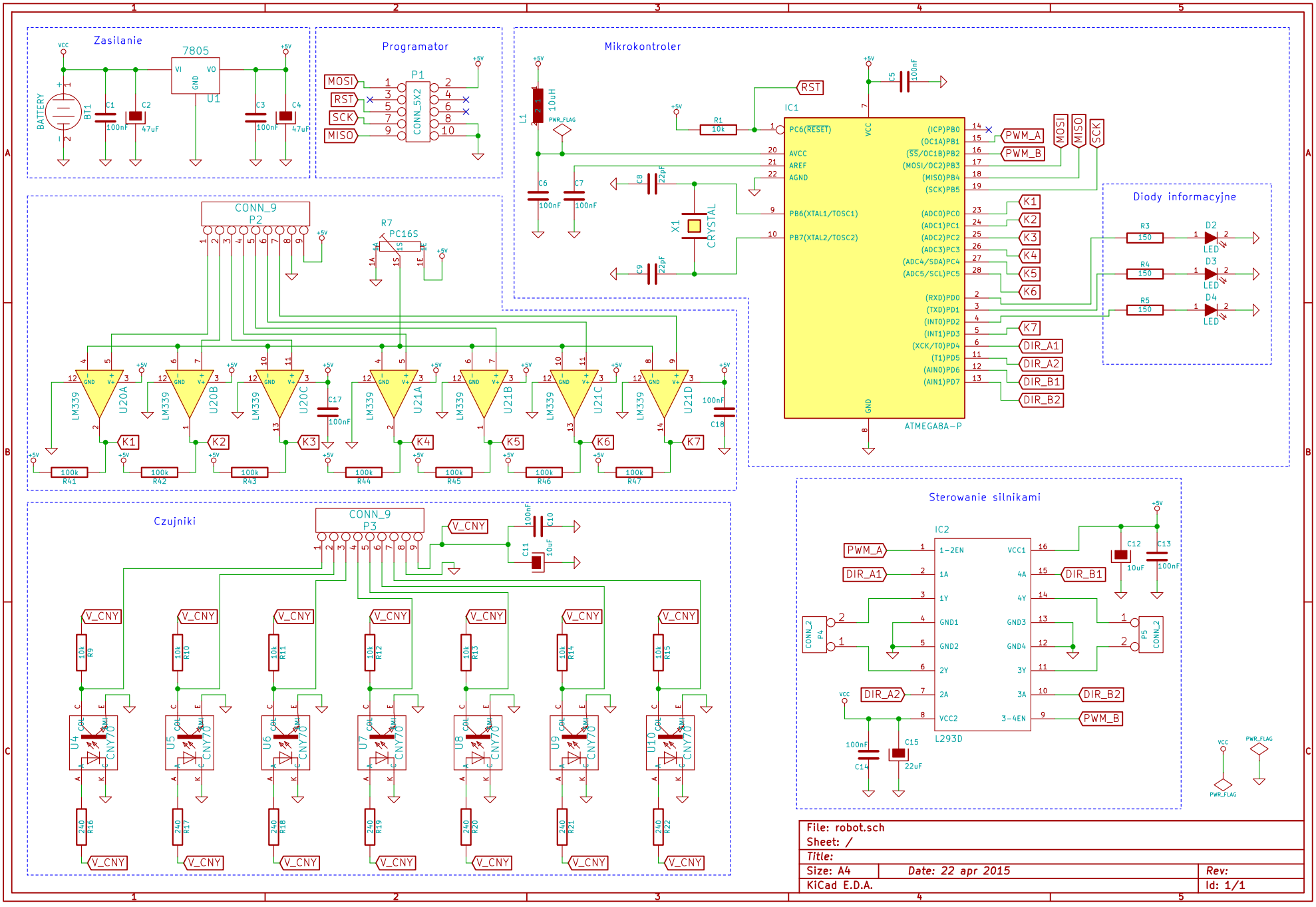
```
91                  updateSensors();
92
93                  current_read = getSteeringValue();
94                  differentialError = current_read - previous_read;
95
96                  computeP();
97                  computeI();
98                  computeD();
99
100                 steeringValue = diffPart + (int)intPart +
                    ↪  propPart;
101
102                 if (steeringValue > 1000)
103                         steeringValue = 1000;
104                 else if (steeringValue < -1000)
105                         steeringValue = -1000;
106
107                 if(!isSensorDetected && lastKnownNonCenterSensor
                    ↪  != 3)
108                 {
109                         if (lastKnownNonCenterSensor > 3)
110                         {
111                                 intPart = diffPart - 0;
112                                 setRightMotorPwm(1000);
113                                 PORTD &= ~(1 << 6); //DIR B1
114                                 PORTD |= (1 << 7); //DIR B2
115                                 setLeftMotorPwm(brakeForce);
116                         }
117                         else if (lastKnownNonCenterSensor < 3)
118                         {
119                                 intPart = diffPart = 0;
120                                 PORTD &= ~(1 << 4); //DIR A1
121                                 PORTD |= (1 << 5); //DIR A2
122                                 setRightMotorPwm(brakeForce);
123                                 setLeftMotorPwm(1000);
124                         }
125
126                 }
127                 else if(steeringValue > 0)
128                 {
129                         PORTD |= (1 << 4); //DIR A1
130                         PORTD &= ~(1 << 5); //DIR A2
131                         PORTD |= (1 << 6); //DIR B1
132                         PORTD &= ~(1 << 7); //DIR B2
133                         setRightMotorPwm(1000 - steeringValue);
134                         setLeftMotorPwm(1000);
```

```
135                 }
136                 else
137                 {
138                         PORTD |= (1 << 4); //DIR A1
139                         PORTD &= ~(1 << 5); //DIR A2
140                         PORTD |= (1 << 6); //DIR B1
141                         PORTD &= ~(1 << 7); //DIR B2
142                         setRightMotorPwm(1000);
143                         setLeftMotorPwm(1000 + steeringValue);
144                 }
145
146                 previous_read = current_read;
147         }
148 }
149
150
151 /*************************************************************/
152 //                                Functions definitions
153 /*************************************************************/
154
155 inline void computeP()
156 {
157         propPart = current_read * kP;
158 }
159
160 inline void computeI()
161 {
162         intPart += current_read * kI * dt;
163
164         if (intPart < 0 && intPart <
          ↪  intPartBoundaries[activeSensor] )
165                 intPart = intPartBoundaries[activeSensor];
166         else if (intPart > intPartBoundaries[activeSensor] )
167                 intPart = intPartBoundaries[activeSensor];
168
169
170 }
171
172 inline void computeD()
173 {
174         diffPart = diffPart * (1 - 0.002) + (differentialError *
          ↪  kD);
175 }
176
177
178 /**
```

```
179  * Set global sensor variable.
180  * Each bit is responsible for one sensor.
181  * 1 = sensor detected line, 0 = otherwise.
182  * 2^6 - most left
183  * 2^5 - second most left
184  * ...
185  * 2^1 - second most right
186  * 2^0 - most right
187  */
188 void updateSensors()
189 {
190         sensors = (PINC & 0b00111111) | ((PIND & 8) ? 0b01000000
          ↪  : 0);
191
192         #ifdef WHITE
193         sensors = ~sensors;
194         #endif
195
196         int leftSensor;
197         int rightSensor;
198
199         //selecting active sensor
200
201         for (int i = 0; i < 3; ++i)
202         {
203                 leftSensor = (sensors & (1 << i));
204                 rightSensor = (sensors & (1 << (6 - i)));
205
206                 if (leftSensor && rightSensor)
207                 {
208                         activeSensor = 3;
209                         isSensorDetected = 1;
210                         return;
211                 }
212
213                 if (leftSensor)
214                 {
215                         activeSensor = i;
216                         lastKnownNonCenterSensor = i;
217                         isSensorDetected = 1;
218                         return;
219                 }
220
221                 if (rightSensor)
222                 {
223                         isSensorDetected = 1;
```

```
                        lastKnownNonCenterSensor = 6-i;
                        activeSensor = 6 - i;
                        return;
                }
        }

        if(sensors & (1 << 3))
        {
                activeSensor = 3;
                isSensorDetected = 1;
                return;
        }

        isSensorDetected = 0;
}

const int lmin = 0;
const int rmin = 0;

const int rmax = 380;
const int lmax = 400;

const int maxSpeed = 1000;

void setLeftMotorPwm(int value)
{
        OCR1B = lmin + (int)(value * ((float)lmax-lmin) /
        ↪  maxSpeed);
}

void setRightMotorPwm(int value)
{
        OCR1A = rmin + (int)(value * ((float)rmax-rmin) /
        ↪  maxSpeed);
}

int getSteeringValue()
{
        return k[activeSensor];
}

void initialize()
{
        // Enable output pins
        DDRB |= (1 << 1); //PWM A
        DDRB |= (1 << 2); //PWM B
```

```
268         DDRD |= (1 << 4); //DIR A1
269         DDRD |= (1 << 5); //DIR A2
270         DDRD |= (1 << 6); //DIR B1
271         DDRD |= (1 << 7); //DIR B2
272         DDRD |= (1 << 0); //Diode 0
273         DDRD |= (1 << 1); //Diode 1
274         DDRD |= (1 << 2); //Diode 2
275
276         //Enable input pins
277         DDRC &= ~(1 << 0); //C1
278         DDRC &= ~(1 << 1); //C2
279         DDRC &= ~(1 << 2); //C3
280         DDRC &= ~(1 << 3); //C4
281         DDRC &= ~(1 << 4); //C5
282         DDRC &= ~(1 << 5); //C6
283         DDRD &= ~(1 << 3); //C7
284
285         //direction constant
286         PORTD |= (1 << 4); //DIR A1
287         PORTD &= ~(1 << 5); //DIR A2
288         PORTD |= (1 << 6); //DIR B1
289         PORTD &= ~(1 << 7); //DIR B2
290
291         //PWM settings
292         ICR1  = 400;
293         //OCR1A = 300;
294         //OCR1B = 300;
295
296         TCCR1A = (1 << COM1A1) | (1 << COM1B1) | (1 << WGM11);
297         TCCR1B = (1 << WGM13) | (1 << WGM12) | (1 << CS10);
298 }
```


Zasilanie
Programator
Mikrokontroler
Diody informacyjne
Czujniki
Sterowanie silnikami
7805
BATTERY
CONN_5X2
CONN_9
ATMEGA8A-P
L293D
LM339
CNY70
CRYSTAL
PC16S
MOSI
MISO
SCK
RST
V_CNY
DIR_A1
DIR_A2
DIR_B1
DIR_B2
PWM_A
PWM_B
File: robot.sch
Sheet: /
Title:
Size: A4
Date: 22 apr 2015
Rev:
KiCad E.D.A.
Id: 1/1