AGH
Akademia Górniczo-Hutnicza im. Stanisława Staszica w Krakowie
Wydział Fizyki i Informatyki Stosowanej

Praca magisterska

Paweł Kmak

kierunek studiów: **informatyka stosowana**

specjalność: **informatyka w nauce i technice**

Rozwój systemu sterowania dla robota mobilnego

Opiekun: **dr hab. inż. Marek Idzik**

Kraków, listopad 2009

Oświadczam, świadomy odpowiedzialności karnej za poświadczenie nieprawdy, że niniejszą pracę dyplomową wykonałem osobiście i samodzielnie i nie korzystałem ze źródeł innych niż wymienione w pracy.

.....
(czytelny podpis)

Kraków, 16 listopada 2009

Tematyka pracy magisterskiej i praktyki dyplomowej Pawła Kmaka, studenta V roku studiów kierunku informatyka stosowana, specjalności informatyka w nauce i technice

Temat pracy magisterskiej: **Rozwój systemu sterowania dla robota mobilnego**

Opiekun pracy:	dr hab. inż. Marek Idzik
Recenzenci pracy:	dr hab. inż. Khalid Saeed
Miejsce praktyki dyplomowej:	WFiIS AGH, Kraków

Program pracy magisterskiej i praktyki dyplomowej

1. Omówienie realizacji pracy magisterskiej z opiekunem.
2. Zebranie i opracowanie literatury dotyczącej tematu pracy.
3. Dobór odpowiednich podzespołów elektronicznych.
4. Praktyka dyplomowa:
   - projekt płytki kamery,
   - montaż oraz testy zaprojektowanego obwodu,
   - dyskusja i analiza wyników,
   - sporządzenie sprawozdania z praktyki.
5. Kontynuacja prac związanych z tematem pracy magisterskiej.
6. Przeprowadzenie testów wykonanego urządzenia.
7. Analiza wyników testów, ich omówienie i zatwierdzenie przez opiekuna.
8. Opracowanie redakcyjne pracy.

Termin oddania w dziekanacie: 16 listopada 2009

....

(podpis kierownika katedry) (podpis opiekuna)

dr hab. inż. Marek Idzik
Wydział Fizyki i Informatyki Stosowanej AGH
Katedra Oddziaływań i Detekcji Cząstek

Merytoryczna ocena pracy przez opiekuna:

Praca magisterska studenta 5-go roku Pawła Kmaka miała na celu projekt i budowę mobilnego robota. Robot ten reagować miał na sygnały wizyjne i w oparciu o ich analizę powinien autonomicznie podejmować właściwe decyzje. Należy uczciwie stwierdzić, ze tak sformułowany temat (na własne życzenie magistranta) wiązać się musiał z ogromnym nakładem pracy. Wynika to z faktu, iż do osiągnięcia zamierzonego celu niezbędne było interdyscyplinarne podejście i praktyczne rozwiązanie szeregu problemów z zakresu elektroniki, informatyki, analizy obrazów oraz mechaniki.

Z dużą satysfakcją muszę przyznać, że magistrant poradził sobie bardzo dobrze z postawionym sobie zadaniem. Nie wchodząc w szczegóły pracy, jej najlepszym podsumowaniem jest jej efekt, w postaci ruchomego robota, z własną kamerą i bezprzewodową komunikacją, który analizuje i rozpoznaje zaobserwowane wzorce i na tej podstawie autonomicznie podejmuje decyzje o dalszej marszrucie.

Wykonaną pracę oceniam bardzo wysoko. Jedynym powodem dla którego ostateczna ocena pracy jest tylko bardzo dobra, są opóźnienia w jej wykonaniu wynikające z powodów znanych tylko magistrantowi.
Końcowa ocena pracy przez opiekuna: 5.0

Data: 13.11.2009r. Podpis: .....

Skala ocen: 5.0 – bardzo dobra, 4.5 – plus dobra, 4.0 – dobra, 3.5 – plus dostateczna, 3.0 – dostateczna, 2.0 – niedostateczna

dr hab. Khalid Saeed
Wydział Fizyki i Informatyki Stosowanej AGH
Katedra Informatyki Stosowanej i Fizyki Komputerowej

Merytoryczna ocena pracy przez recenzenta:

Recenzowana praca dotyczy rozwoju systemu sterowania dla robota mobilnego. Student zrealizował zakres pracy i osiągnął zamierzony cel. W pracy autor podał krótki i wystarczający wstęp na temat rozwoju robotyki oraz teoretyczny ogólny opis etapów analizy i rozpoznawania obrazów. Zaprojektował i precyzyjnie zbudował robota dla 32-bitowego mikrokontrolera ARM7, AT91SAM7S, który steruje pracą robota i pozostałymi podzespołami (akwizycją, analizą i rozpoznawaniem obrazu) po odpowiednim oprogramowaniu systemu. Przedstawione eksperymenty i testy pracy robota ukazały prawidłowe działanie systemu z wysoką skutecznością rozpoznawania obiektów.

Cały system informatyczno-techniczny został wykonany przez studenta - projekt, implementacja komputerowa, hardwareowa, softwareowa oraz sama mechanika budowy robota oraz projekt płyt drukowanych i ich zmontowanie.

Recenzowana praca zasłużyła na wyróżnienie. Nie znaleziono błędów merytorycznych, poprawny jest styl językowy, ciekawe ilustracje i wyraźne rysunki są dopełnieniem pracy.

Uważam, ze praca powinna być kontynuowana w celu powiększenia zastosowań robota (z szybszym procesorem i lepszą jakością otrzymanych obrazów - większą rozdzielczością).

Końcowa ocena pracy przez recenzenta: 5.0

Data: 4.11.2009r. Podpis: .....

Skala ocen: 5.0 – bardzo dobra, 4.5 – plus dobra, 4.0 – dobra, 3.5 – plus dostateczna, 3.0 – dostateczna, 2.0 – niedostateczna

Spis treści

Wstęp

Niedawno obchodziliśmy 40-tą rocznicę jednego z największych osiągnięć ludzkości, jakim niewątpliwie było lądowanie statku misji Apollo 11 na powierzchni Księżyca. Miało to miejsce 20 lipca 1969 roku, o godzinie 20:17 czasu uniwersalnego [16]. Neil Armstrong i Buzz Aldrin sprawili, iż Srebrny Glob jest pierwszym, dziś możemy dodać jedynym obcym ciałem niebieskim, któremu dane było gościć przedstawicieli rodzaju ludzkiego. Prawdopodobnie na kolejny duży krok w dziedzinie załogowej eksploracji przyjdzie nam poczekać kolejne dekady. Nie oznacza to jednak, że ludzkość stanęła w miejscu.

Eksplorację kontynuowały sondy bezzałogowe. Pierwszą planetą, na której pomyślnie wylądowała ziemska aparatura była Wenus, którą w 1970 roku odwiedził lądownik radzieckiej sondy Wenera-7. Kolejnym na liście stał się Mars, którego ostatecznie, 6 lat po Wenus podbiła amerykańska sonda Viking-1 – i to właśnie Mars jest obecnie najlepiej zbadaną przez człowieka planetą Układu Słonecznego [16].

Dla poznania Czerwonej Planety kluczowe okazało się ostatnie 15 lat. W tym czasie wysłano wiele zaawansowanych misji, z czego 2 miały za zadanie wylądować na powierzchni i zbadać ją, przy pomocy mobilnych robotów sterowanych z Ziemi, wyposażonych w aparaturę naukową. Pathfinder wysłany przez NASA wylądował na powierzchni w 1997 roku. Lądownik miał na pokładzie małego robota, który w trakcie misji pokonał dystans ok. 100 metrów, wykonując zdjęcia i analizy chemiczne gruntu. Transmisja danych odbywała się za pośrednictwem lądownika-matki, więc łazik nie mógł zbytnio się oddalać [16]. Po sukcesie Pathfindera na powierzchni planety w 2004 roku, znów za sprawą NASA, wylądowały 2 bliźniacze roboty – Mars Exploration Rover (MER): Spirit i Opportunity. Łaziki mają znacznie lepsze wyposażenie naukowe, mogą przejechać znacznie większe odległości i nie są uzależnione od stacji bazowej. Ich misja trwa nadal [19]. Więcej informacji o misji MER znajduje się w dalszej części pracy.

To właśnie misje Pathfinder oraz Mars Exploration Rover były inspiracją autora, do stworzenia mobilnego robota z możliwością pracy autonomicznej, w oparciu o dane pozyskane z otoczenia.

Podstawową ideą projektu było stworzenie mobilnej platformy, wyposażonej w kamerę, z możliwością komunikacji bezprzewodowej. Urządzenie miało być jednak samowystarczalne, zdolne do zbierania informacji o otoczeniu, wykorzystując jedynie instrumenty zainstalowane na pokładzie. Dane z kamery miały być bezpośrednio dostępne dla mikroprocesora sterującego robota, w celu wykonywania prostej obróbki i analiz. Urządzenie miało być także przygotowane do bezprzewodowego przesłania zgromadzonych danych, jak również posiadać miało tryb zdalnej kontroli – wszystko to w oparciu o interfejs bezprzewodowy.

Po stworzeniu urządzenia spełniającego powyższe wymagania, kolejnym krokiem było poddanie uzyskiwanych z kamery obrazów analizie i rozpoznawaniu, w celu wydobycia użytecznych informacji, mogących posłużyć do sterowania robotem. W ten sposób pojazd miał zyskać możliwość pracy autonomicznej.

Postawione wymagania zostały spełnione, a wszystkie etapy ich realizacji zostały przedstawione w kolejnych rozdziałach niniejszego opracowania.

W rozdziale pierwszym przedstawiono ogólną klasyfikację robotów, ze szczególnym uwzględnieniem robotów mobilnych. Pokazano również czołowe osiągnięcia robotyki mobilnej na przykładzie konstrukcji komercyjnych, naukowych i wojskowych, stworzonych w ostatnich latach.

W drugim rozdziale zaprezentowano projekt elektroniki i mechaniki wykonanego przez autora pracy robota. W pierwszej części omówiono najważniejsze podzespoły robota ze szczególnym uwzględnieniem kamery, stanowiącej podstawę systemu sterowania. Dalej zawarto projekty wykonanych płytek elektronicznych, ich szczegółową specyfikację oraz przedstawiono projekt części mechanicznej.

Rozdział trzeci omawia metody przetwarzania, analizy i rozpoznawania obrazu, wykorzystane przy projektowaniu systemu sterowania.

Czwarty rozdział opisuje firmware robota. Omówiono w nim odbiór danych z kamery, implementację systemu sterowania oraz obsługę komunikacji bezprzewodowej. Ostatni podrozdział zawiera opis wszystkich komend obsługiwanych przez robota.

Rozdział piąty poświęcono programowi służącemu do zarządzania pracą robota. Zawarto tutaj szczegółowy opis wszystkich jego funkcji.

W rozdziale szóstym opisano wykonane testy. W kolejnych podrozdziałach przedstawiono wyniki testów transmisji bezprzewodowej, systemu wizyjnego oraz systemu analizy i rozpoznawania obrazu.

Pracę kończy podsumowanie, w którym zebrano najważniejsze zadania wykonane przez autora w celu realizacji wyznaczonych celów.

Rozdział 1

Rozwój robotyki

Robot jest urządzeniem mechanicznym wykonującym określone zadania w sposób automatyczny. Szybki rozwój techniki sprawił, iż obecnie roboty wykorzystywane są do przeróżnych zadań i charakteryzują się bardzo zróżnicowanym poziomem skomplikowania – zarówno pod względem mechaniki, jak i systemów sterujących. Chcąc zagłębić się w dziedzinę nauki jaką jest robotyka, należy precyzyjniej określić zakres omawianych problemów.

1.1 Podział robotów

Często spotykany jest podział robotów na trzy generacje [17, 18]. Za kryterium podziału przyjmuje się tu możliwości i skomplikowanie zastosowanego w robocie systemu sterowania.

Roboty pierwszej generacji — są to urządzenia zaprogramowane do wykonywania określonej sekwencji czynności zapisanej w pamięci. Potrafią działać bez ingerencji człowieka. Nie są zdolne do samodzielnego pozyskiwania informacji o zewnętrznym środowisku, w którym pracują. Do grupy tej zaliczają się roboty przemysłowe.

Roboty drugiej generacji — wyposażone są w czujniki, dzięki którym dokonują pomiarów podstawowych parametrów pracy oraz otoczenia. Potrafią rozpoznać dany obiekt w zbiorze, niezależnie od miejsca pracy, czy też położenia obiektu. Roboty drugiej generacji powinny samodzielnie wybierać sposób realizacji wyznaczonych celów, dostosowany do aktualnego stanu otoczenia.

Roboty trzeciej generacji — są to urządzenia wyposażone w sztuczną inteligencję. Potrafią działać w zmieniających się warunkach oraz nieznanym otoczeniu. Dokonują złożonych pomiarów, potrafią rozpoznawać złożone kształty i klasyfikować

złożone sytuacje. System sterowania robotem trzeciej generacji powinien posiadać zdolności adaptacyjne.

Ponadto roboty podzielić można na stacjonarne i mobilne. Roboty mobilne mogą poruszać się w różny sposób oraz w różnych środowiskach. Wyróżnić tu można roboty kołowe, kroczące, latające, pływające i inne.

Projekt wykonany w ramach niniejszej pracy jest robotem mobilnym klasy drugiej, poruszającym się na 4 kołach. Informacje o otoczeniu zewnętrznym pozyskiwane są dzięki analizie obrazu, wykonywanej przez pokładowy system sterujący.

1.2 Sposoby sterowania robotem mobilnym

Najprostszy system sterowania może być listą instrukcji sterujących poszczególnymi podzespołami robota. Po uruchomieniu programu robot automatycznie wykonuje wszystkie zapisane w programie instrukcje, co w efekcie prowadzi do wykonania określonego zadania. Robot sterowany takim systemem jest robotem pierwszej generacji.

Podejście takie traci jednak sens podczas sterowania robotem mobilnym, gdyż ruch powoduje zmianę układu elementów otoczenia względem robota. W zależności od położenia, wykonanie tych samych zadań może wymagać zastosowania zupełnie różnych instrukcji sterujących. W związku z tym, konieczne jest pozyskiwanie informacji o otoczeniu.

Informacje mogą być pozyskiwane za pomocą różnych czujników i sensorów. Najprostszą metodą badania otoczenia jest zastosowanie czujników zbliżeniowych lub ultradźwiękowych. Jest to rozwiązanie często stosowane, pozwalające na określenie odległości pomiędzy robotem a okolicznymi obiektami. Inne stosowane czujniki to między innymi: akcelerometry, żyroskopy, inklinometry, czujniki warunków środowiskowych (temperatury, ciśnienia itp.), mikrofony, kamery.

1.2.1 Roboty mobilne ze sterowaniem autonomicznym

Roboty autonomiczne są w stanie poruszać się i wykonywać wyznaczone zadania bez udziału człowieka, w oparciu o pozyskiwane przy pomocy czujników informacje o otoczeniu. Sposób realizacji wyznaczonych zadań ma być samodzielnie określany przez system sterowania robota.

Autonomiczne systemy sterowania działające w oparciu o kamery i analizę ob-

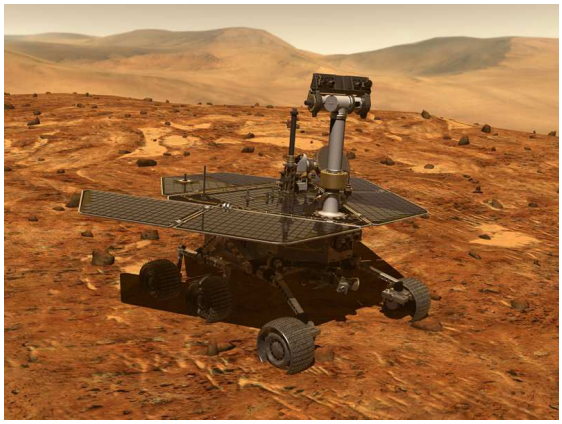
razu, są jednymi z ciekawszych rozwiązań. Analiza obrazu daje ogromne możliwości, równocześnie jednak pozyskiwanie potrzebnych informacji z obrazu jest sprawą bardzo trudną. Jak na razie mózg człowieka jest niedoścignionym wzorcem w tej dziedzinie. Warto jednak rozwijać tego typu systemy, gdyż analiza obrazu jest rozwiązaniem przyszłościowym. Obecnie największą autonomię osiągają roboty wykorzystujące obraz jako jedno ze źródeł informacji.

Jednym z celów projektu realizowanego w ramach niniejszej pracy, było stworzenie prototypowego systemu analizy i rozpoznawania obrazu, mogącego dostarczyć dane wejściowe dla autonomicznego systemu sterowania robota.

1.3 Przykłady projektów komercyjnych/naukowych

1.3.1 Mars Exploration Rover

Jednym z najbardziej niezwykłych zastosowań robotów mobilnych jest niewątpliwie eksploracja Marsa. Spirit i Opportunity (rys. 1.1) to 2 bliźniacze roboty wysłane przez Amerykańską Agencję Kosmiczną NASA w celu przeprowadzenia badań powierzchni Czerwonej Planety i zebrania jak największej ilości danych naukowych.



Rysunek 1.1: Mars Exploration Rover – NASA, JPL [19].

Każdy z robotów porusza się na 6-u niezależnie napędzanych kołach. Zasilanie zapewniają akumulatory litowe ładowane za pomocą paneli słonecznych. Roboty posiadają szereg instrumentów naukowych, niektóre z nich umieszczone zostały na ruchomym wysięgniku o kilku stopniach swobody. Pojazdy posiadają ponadto:

- 2 sprzężone kamery stereograficzne o rozdzielczości 1024x1048 pikseli każda

- 2 kamery nawigacyjne
- 4 kamery unikania niebezpieczeństw, umożliwiające wykrywanie przeszkód na drodze pojazdu

System sterowania robotów zrealizowano w oparciu o 32-bitowy procesor Rad6000 o mocy obliczeniowej 20MIPS. Ma on do dyspozycji 128MB pamięci RAM i 256MB pamięci Flash. Maksymalna częstotliwość taktowania wynosi 25MHz [16, 19].

Roboty są w dużym stopniu autonomiczne ze względu na utrudnioną komunikację z Ziemią. Opóźnienie sygnału radiowego wynosi od kilku do kilkudziesięciu minut w jedną stronę – w zależności od aktualnej odległości między Ziemią a Marsem.

Funkcje autonomiczne

Oprócz podstawowych funkcji autonomicznych typu: zarządzanie zasilaniem, samodzielne nawiązywanie komunikacji z Ziemią, przeprowadzanie procedur awaryjnych, roboty wyposażone zostały w autonomiczny tryb poruszania się, pozwalający na samodzielne wytyczanie trasy.

Rejony Marsa, w których operują roboty, pokryte są licznymi skałami, będącymi dużym zagrożeniem dla pojazdów. Zastosowanie trybu przemieszczania autonomicznego pozwala usprawnić nawigację. Po przesłaniu przez obsługę misji punktu docelowego dla robota, autonomiczny system sterowania sam wyznacza optymalną trasę, analizując dane z kamer unikania niebezpieczeństw i omijając niebezpieczne przeszkody. Bezpośrednie sterowanie robotem w takich warunkach wymagałoby wykonania znacznie większej ilości sesji komunikacyjnych i przesłania dodatkowych danych nawigacyjnych – co przy znacznych opóźnieniach sygnału i niewielkich transferach znacznie wydłużyłoby czas podróży. Przy zastosowaniu trybu autonomicznego robot jest w stanie przejechać do 100 metrów dziennie.

Dodatkowa autonomia związana jest z manipulatorem robota wyposażonym w urządzenia naukowe. Poruszanie ramieniem również może odbywać się w sposób autonomiczny, pozwalający na bezpieczne umieszczenie aparatury naukowej wśród skał, eliminując ryzyko przypadkowego uszkodzenia. Również tutaj, autonomia robota zmniejsza udział człowieka w sterowaniu i pozwala przyspieszyć badania.

Roboty pracują na Marsie od 2004 roku. Znacznie przekroczyły swój gwarantowany czas pracy, obliczony na 90 dni. Jak na razie są to najbardziej zaawansowane roboty mobilne wysłane na powierzchnię obcej planety. Przyszłe planowane i

przygotowywane obecnie misje, dostarczą na powierzchnię Marsa roboty w jeszcze większym stopniu autonomiczne [16, 19, 20].

1.3.2 ASIMO

Asimo jest robotem humanoidalnym stworzonym przez firmę Honda. Robot porusza się na 2 nogach i dynamicznie balansuje masą, co powoduje, że jest on jednym z najbardziej rozwiniętych robotów humanoidalnych na świecie. Jego sposób poruszania się bardzo dobrze odwzorowuje ruch człowieka. Robot ma 130cm wzrostu i waży 54kg.

Dzięki zastosowanej analizie obrazu robot potrafi wykrywać ruchy otaczających go obiektów. Szacuje odległości od obiektów oraz ich kierunek poruszania się. Po wydaniu odpowiedniej komendy może podążać za ruchomym obiektem.

Asimo jest również w stanie śledzić ruchy ludzkiej ręki oraz rozpoznawać gesty. Daje mu to możliwość np. przywitania się z człowiekiem który wyciągnął do niego dłoń. Rozpoznaje również przedmioty i środowisko, w którym się znajduje. Dzięki temu porusza się bezpiecznie unikając niebezpieczeństw oraz omijając przedmioty i ludzi. Potrafi również rozpoznawać dźwięki i twarze [18, 22].

1.3.3 Przykłady innych projektów

W morskim przemyśle paliwowym wykorzystywane są autonomiczne łodzie podwodne – AUV. Tworzą one szczegółowe mapy dna morskiego wykorzystywane przy projektowaniu optymalnej infrastruktury podwodnej. Potrafią samodzielnie nawigować i poruszać się pod wodą, bez udziału człowieka wykonując powierzone im misje. Informacji o otoczeniu dostarcza szeroka gama czujników. Wykorzystywane są również przez wojsko do wyszukiwania min [18].

W armii użytkowane są również bezzałogowe samoloty autonomiczne (UAV). Przykładem jest tutaj projekt Predator. Pojazd jest używany do prowadzenia rozpoznania lub ataku. Może pozostawać w powietrzu do 24 godzin prowadząc rozpoznanie określonego terenu w sposób autonomiczny. Po zlokalizowaniu celu może przeprowadzić atak – po otrzymaniu odpowiedniego polecenia. Decyzja o rozpoczęciu ataku nie może być podjęta w sposób samodzielny. Predator wyposażony jest w kamerę kolorową umieszczoną z przodu samolotu. Dodatkowo urządzenie posiada radar, kamerę podczerwieni, GPS. Komunikacja odbywa się za pośrednictwem łącza satelitarnego. Maksymalny pułap to 7500m. Samolot może przenosić 2 kierowane

pociski typu Hellfire klasy powietrze – ziemia [18, 21].

Rozdział 2

Projekt i budowa robota

Zaprojektowanie i zbudowanie robota było procesem dość złożonym. Prace projektowe rozpoczęły się od analizy dostępnych na rynku podzespołów elektronicznych, pod kątem doboru sprzętu mogącego sprostać postawionym wymaganiom. Głównym problemem był tu dobór układów odpowiedzialnych za komunikację bezprzewodową oraz znalezienie kamery z odpowiednim interfejsem komunikacyjnym o wymaganych parametrach.

Po dobraniu odpowiednich podzespołów przyszła kolej na zaprojektowanie i wykonanie płytek drukowanych, integrujących wszystkie elementy elektroniczne i zapewniających wymaganą funkcjonalność projektu.

Ostatnim etapem budowy było wykonanie części mechanicznej robota składającej się z:

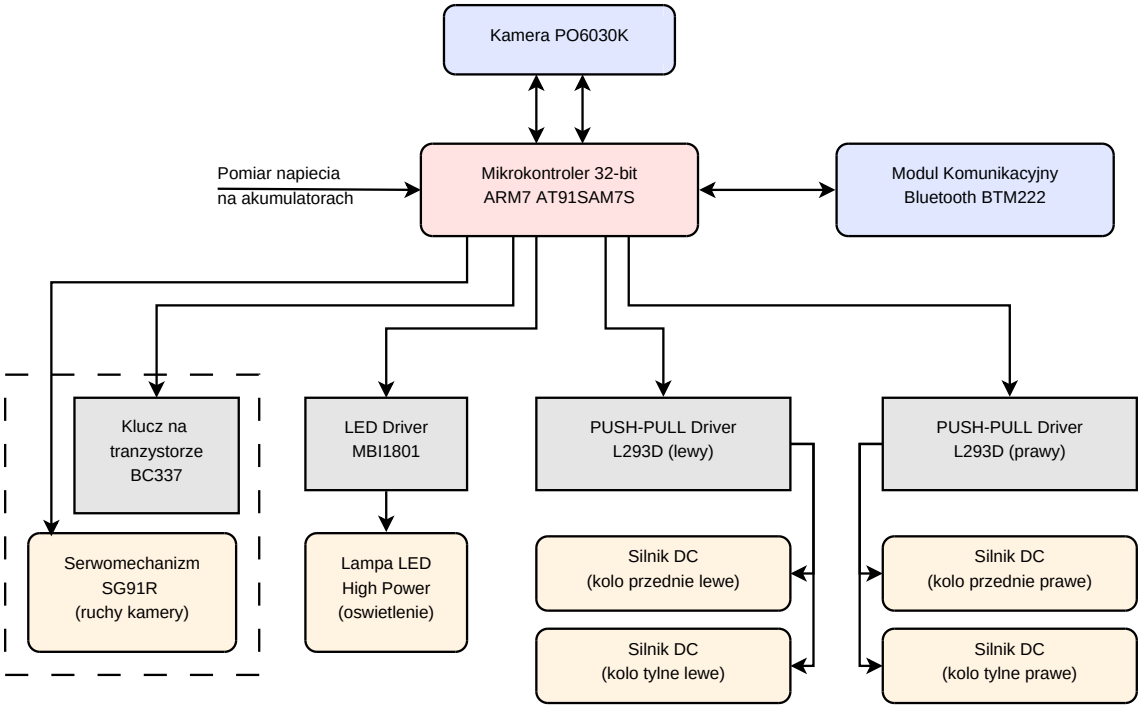
- podwozia zawierającego w sobie stworzone wcześniej obwody elektroniczne, silniki, akumulatory zasilające
- ruchomej wieży z zainstalowaną w środku kamerą.

Równolegle z powyższymi pracami, pisane było oprogramowanie sterujące pod dwie różne architektury sprzętowe. Pierwszy program działa na pokładzie robota i został zaprojektowany dla 32–bitowego mikrokontrolera ARM7, AT91SAM7S sterującego pracą robota i wszystkimi jego podzespołami. Jest to najważniejsze oprogramowanie systemu, odpowiedzialne za całą funkcjonalność, w szczególności za akwizycję, analizę i rozpoznawanie obrazu z kamery.

Program drugi, dedykowany na komputer PC, umożliwia bezprzewodową komunikację z robotem. Za jego pośrednictwem możliwa jest konfiguracja robota, zdalna kontrola, odczyt różnego rodzaju parametrów pracy i pozyskanych danych – w tym obrazu z kamery. Szczegółowe informacje o oprogramowaniu znajdują się w rozdziale trzecim.

2.1 Najważniejsze podzespoły robota

Zależności pomiędzy poszczególnymi podzespołami robota ilustruje uproszczony schemat z rysunku 2.1.



Rysunek 2.1: Uproszczony schemat zależności pomiędzy najważniejszymi podzespołami robota.

2.1.1 Mikrokontroler

Jako jednostkę sterującą wybrano mikrokontroler 32–bitowy, z rdzeniem ARM7 firmy Atmel [14] — AT91SAM7S256 [5]. Wybór padł właśnie na ten układ, ze względu na jego duże możliwości i bogatą dokumentację. Najważniejsze parametry techniczne procesora przedstawiają się następująco:

- 256KB pamięci Flash
- 64KB pamięci SRAM
- zegar 96MHz
- kontroler DMA
- 4 kanały PWM

- 8 kanałów ADC
- interfejsy UART, SPI, TWI/I^2C, USB Slave,
- 32-bitowy port wejścia/wyjścia z tolerancją 5V
- zasilanie 3.3V

Producent gwarantuje poprawne działanie do 55MHz, jednak przeprowadzone przez autora testy dla zegara 96MHz nie wykazały nieprawidłowości, więc zdecydowano się wykorzystać dodatkową moc obliczeniową.

Powyższy układ zakupiono w firmie Propox, w postaci modułu MMsam7s [13] ze zintegrowanym rezonatorem kwarcowym oraz stabilizatorem napięcia 3.3V.

Mikrokontroler posiada szereg dodatkowych urządzeń peryferyjnych i funkcji tutaj nie wymienionych. Szczegółowe informacje znajdują się w dokumentacji technicznej układu [5].

2.1.2 Kamera

W momencie kompletowania podzespołów, na polskim rynku była dostępna praktycznie tylko jedna kamera, spełniająca stawiane wymagania. Była to kamera Pixelplus PO3030K, która została wycofana przez dystrybutora z rynku, w trakcie testowania prototypowych elementów robota. W zamian za wycofany układ, na rynek wprowadzona została kamera PO6030K [6], o podobnych parametrach i możliwościach, jednak innych wymaganiach co do zasilania oraz zmienionym ułożeniem pinów wyjściowych. Spowodowało to konieczność wykonania nowej wersji jednej z płytek drukowanych.

Warto zaznaczyć, że w niniejszym projekcie wymagania dotyczące kamery były dość restrykcyjne. Priorytetowym założeniem było zapewnienie mikrokontrolerowi bezpośredniego dostępu do danych, co z kolei wymuszało konieczność posiadania przez kamerę odpowiedniego interfejsu komunikacyjnego – najlepiej typu równoległego. W szczególności niemożliwe było wykorzystanie kamery internetowej ze złączem usb, czy też kamery analogowej, przeznaczonej do monitoringu (nawet takiej ze zintegrowanym transmiterem bezprzewodowym).

Ostatecznie w finalnej wersji projektu znalazła się kamera PO6030K, spełniająca powyższe wymagania. Jej najważniejsze cechy:

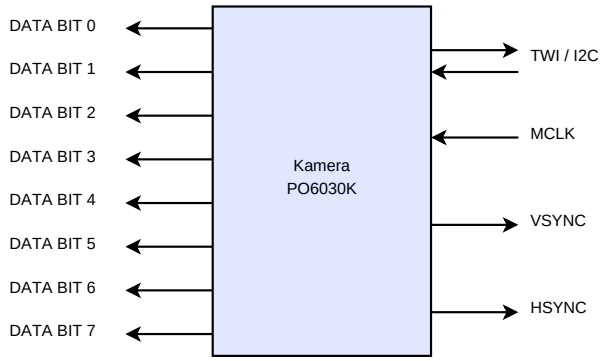
- sensor typu CMOS

- zintegrowana elektronika, obsługująca sensor i transmisję danych, oferująca podstawową korekcję obrazu
- obraz kolorowy lub w skali szarości, o rozdzielczości maksymalnej 640x480px
- maksymalnie 30 klatek na sekundę, przy zegarze 24 MHz
- 3 napięcia zasilania: 3.3V (I/O), 2.8V (część analogowa), 1.8V (rdzeń)

Układ PO6030K posiada 2 interfejsy komunikacyjne:

1. Szeregowy interfejs konfiguracyjny (TWI/I^2C)
2. Równoległy interfejs transmisji danych

Kamera udostępnia również dodatkowe sygnały służące do synchronizacji transmisji. Najważniejsze linie komunikacyjne pokazano na rysunku 2.2.



Rysunek 2.2: Podstawowe linie komunikacyjne kamery PO6030K.

Pierwszy służy do zapisu/odczytu wartości rejestrów konfiguracyjnych, sterujących pracą kamery. W ten sposób możliwa jest zmiana parametrów, takich jak:

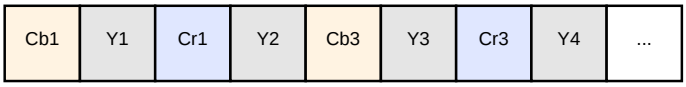
- format transmitowanego obrazu (YCbCr, RGB, Mono)
- ograniczenie przesyłanego obrazu do danego wycinka
- konfiguracja czasów i trybów naświetlania, bilansu bieli
- konfiguracja sygnałów synchronizujących transmisję (HSYNC, VSYNC)
- i wiele innych

Należy jednak wspomnieć, że dokumentacja kamery [6] miejscami jest nieprecyzyjna, czasami zdarzają się błędy – przez co niemożliwe jest ustawienie niektórych trybów pracy lub tryby te nie działają prawidłowo. Na pierwszy problem napotyka się już przy pierwszej próbie konfiguracji kamery, gdyż jej adres sprzętowy podany w dokumentacji jest nieprawidłowy. Rozwiązaniem jest automatyczna próba nawiązania komunikacji, testując wszystkie 256 możliwych adresów, co umożliwia znalezienie tego prawidłowego. Błędy w dokumentacji nie są jednak na tyle poważne, aby uniemożliwić korzystanie z kamery. Ponadto kamera pracuje poprawnie w konfiguracji fabrycznej.

Interfejs równoległy służy do odbierania kolejnych bajtów obrazu. Po wykonaniu procedury startowej i podaniu sygnału zegarowego MCLK, dane obrazu wystawiane są na 8-bitowy port wyjściowy kamery, według porządku zależnego od konfiguracji rejestrów. Dodatkowo dostępne są sygnały synchronizujące VSYNC i HSYNC, dzięki którym można ustalić współrzędne aktualnie transmitowanego piksela obrazu.

Sposób transmisji obrazu

Nowe dane na port wyjściowy wystawiane są domyślnie, na opadającym zboczu zegara MCLK. Po resecie, kamera wysyła obraz o rozdzielczości 640x480 pikseli, w trybie YCbCr422. W trybie tym, na każdy piksel przypadają średnio 2 bajty danych. Dla każdego piksela wysyłana jest jego składowa luminancji (Y), zajmująca 8 bitów. Dodatkowo dla połowy pikseli, równomiernie rozłożonych na obrazie, wysyłane są 2 składowe chrominancji (Cb i Cr), zajmujące 8 bitów każda. Jest to sposób na zmniejszenie ilości informacji przypadających na pojedynczą klatkę obrazu, bez znaczącej utraty jakości – gdyż ludzkie oko znacznie silniej reaguje na różnice w luminancji (składowa Y) niż w chrominancji. Kolejność transmitowanych bajtów dokładniej pokazuje rysunek 2.3. Widać stąd, że na pierwszy piksel składają się 3 bajty danych (Cb, Y, Cr), w skład drugiego wchodzi tylko luminancja (Y), trzeci piksel znów zawiera pełne dane itd.



Rysunek 2.3: Sekwencja danych w trybie YCbCr422. Cb, Cr – dane chrominancji, Y – luminancja. Indeksem oznaczono numer piksela.

Współczynniki Y, Cb, Cr powiązane są ze składowymi R, G, B układem równań 2.1 [15].

$$\begin{cases} Y = 0.299R + 0.587G + 0.114B \\ Cb = 0.564(B - Y) + 128 \\ Cr = 0.713(R - Y) + 128 \end{cases} \tag{2.1}$$

Szczegółowe informacje na temat transmisji oraz rejestrów konfiguracyjnych znajdują się w dokumentacji kamery [6].

Przejście do przestrzeni RGB

Aby po odebraniu danych z kamery powrócić do przestrzeni RGB, należy wykonać kolejną transformację. Odpowiednie wzory wyprowadzić można, korzystając z układu równań 2.1. Z równań na Cb i Cr należy wyliczyć współczynniki B oraz R.

$$B = 1.773Cb + Y - 227.0 \tag{2.2}$$

$$R = 1.403Cr + Y - 179.5 \tag{2.3}$$

Wstawiając 2.2 oraz 2.3 do równania na Y z układu 2.1 otrzymamy:

$$G = -0.7144Cr - 0.3443Cb + Y + 135.5 \tag{2.4}$$

2.1.3 Moduł komunikacji bezprzewodowej

Do transmisji bezprzewodowej wykorzystany został moduł Bluetooth BTM222 [7], ze względu na przyjazną obsługę, relatywnie wysoką prędkość transmisji oraz przede wszystkim dużą uniwersalność i wygodę użytkowania. Parametry urządzenia:

- maksymalna prędkość transmisji: 460.8 Kbps
- zasięg w terenie otwartym: 100m
- interfejs szeregowy
- konfiguracja poprzez komendy SPP AT
- zasilanie 3.3V

Moduł konfigurowany jest za pomocą komend podanych w dokumentacji [7]. Komendy wydaje się poprzez interfejs szeregowy UART. Każdą komendę należy poprzedzić wysłaniem znaków 'AT' i zakończyć znakami [cr] + [lf] (powrót karetki + nowa linia).

Po zestawieniu połączenia, po stronie komputera PC moduł widoczny jest jako wirtualny port COM. Port UART modułu staje się w tym momencie przezroczysty dla danych. Komunikacja może odbywać się w obydwie strony równocześnie.

2.1.4 Napęd

Napęd robota zrealizowany jest w oparciu o 4 silniki DC, ze zintegrowaną przekładnią zmniejszającą obroty. Nominalna prędkość obrotowa motorów to 95RPM, przy zalecanym napięciu zasilania 6V. Pojazd skręca, poprzez generowanie różnicy w prędkościach obrotowych i/lub kierunkach obrotów silników, znajdujących się po przeciwległych bokach podwozia.

Sterowanie silnikami odbywa się za pośrednictwem dwóch driverów Push–Pull, pełniących rolę scalonych układów typu H-bridge – L293D [10] (rys. 2.1). Sterowanie prędkością obrotową możliwe jest poprzez wykorzystanie modulacji PWM. Dla każdego silnika przewidziany jest oddzielny kanał.

Oprócz silników zapewniających ruch postępowy, robot został wyposażony w ruchomą wieżę, której napęd stanowi serwomechanizm analogowy. Jest on sterowany bezpośrednio z mikrokontrolera. W razie potrzeby zasilanie serwa może zostać odcięte przez zewnętrzny tranzystor (rys. 2.1).

2.1.5 Oświetlenie

W wieży oprócz kamery, zamontowana została również biała dioda LED o mocy 1W wraz z kolimatorem 45°. Lampa pozwala na pracę w warunkach niedostatecznego oświetlenia.

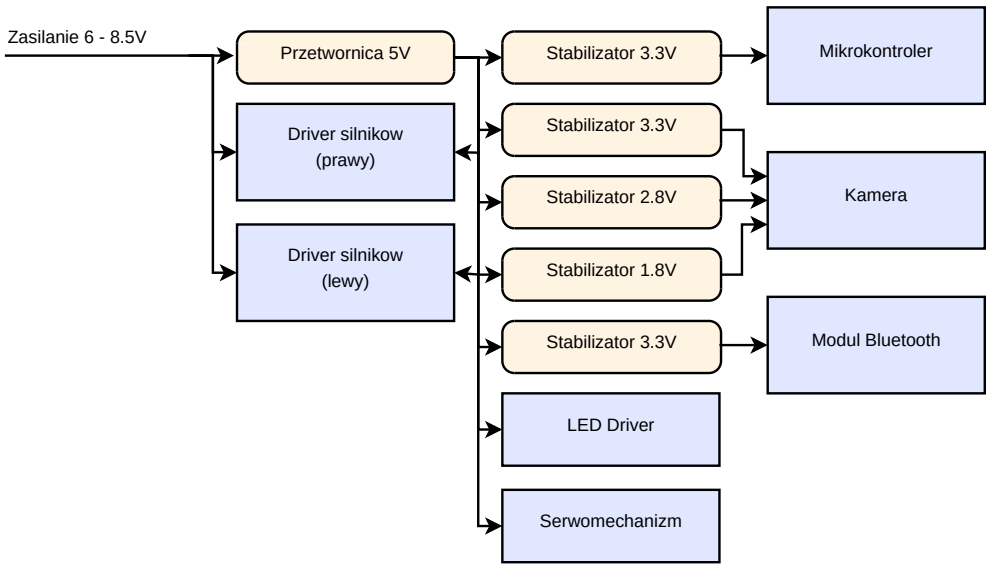
Ze względu na dużą moc diody, konieczne było zainstalowanie układu stabilizującego prąd. Wykorzystano driver o symbolu MBI1801 [9], dedykowany specjalnie do celów zasilania diod LED.

2.1.6 Zasilanie

Do zasilania urządzenia wykorzystano 2 cylindryczne ogniwa litowo-jonowe o rozmiarze 18650 (przypominające standardowe baterie AA, lecz nieco większe). Ich napięcie nominalne to 3.7V, pojemność 2400mAh. Akumulatory połączone są w układzie szeregowym. Czas pracy po naładowaniu może wynosić nawet do 10 godzin.

Rzeczywiste napięcie na akumulatorach nie jest jednak stałe w czasie i zależy od stopnia naładowania akumulatorów, oraz od aktualnego obciążenia. Przy pełnym naładowaniu, napięcie uzyskiwane z szeregowego połączenia 2 akumulatorów wynosi 8.5V (4.25V / ogniwo). Po rozładowaniu napięcie jest na poziomie 6V (3.0V / ogniwo). Jest to minimalne napięcie, uznawane jeszcze za bezpieczne dla ogniw litowo–jonowych. Po jego osiągnięciu, zalecane jest podłączenie akumulatorów do ładowarki. Dalsze rozładowywanie spowodowałoby szybki spadek napięcia do wartości ok 5V, a następnie odcięcie zasilania przez układ zabezpieczający, w celu ochrony akumulatorów przed zniszczeniem.

Elektronika robota potrzebuje napięć z przedziału 5V – 1.8V. Potrzebne zasilania wytwarzane są dwustopniowo. Pierwszy stopień stanowi przetwornica impulsowa step-down, zbudowana na układzie LM2575 [8], przetwarzająca napięcie uzyskane z akumulatorów na poziom 5V. Użycie przetwornicy pozwoliło zaoszczędzić część energii i zwiększyło sprawność procesu konwersji napięcia, w stosunku do standardowego stabilizatora liniowego. Zysk jest największy w momencie gdy akumulatory są w pełni naładowane. Uzyskane napięcie 5V wykorzystywane jest do zasilania części urządzeń. Wszelkie pozostałe napięcia generowane są z 5V przy użyciu stabilizatorów liniowych [11] – stopień drugi. Wyjątkiem są silniki napędowe. Drivery sterujące pracą silników zasilane są bezpośrednio z akumulatorów. Schemat blokowy układu zasilania przedstawia rysunek 2.4.



Rysunek 2.4: Schemat blokowy układu zasilania robota.

Układ zasilania zaprojektowany został w taki sposób, iż możliwe jest wykorzy-

stanie napięć wejściowych z przedziału 6V – 35V (również 5V po przełączeniu odpowiednich zworek, odłączających przetwornicę). Jedynym ograniczeniem jest tutaj rodzaj zainstalowanych silników. Obecnie wykorzystywane silniki zawężają możliwe napięcie wejściowe do przedziału 6V – 9V. Możliwe jest jednak podłączenie silników o innym napięciu nominalnym, przykładowo 12V – wtedy napięcie zasilania powinno wynosić od 12 do 15V. Użyte drivery pozwalają na zastosowanie silników pobierających maksymalnie 600mA prądu ciągłego.

2.2 Projekt części elektronicznej

Elektronika została podzielona na 2 odrębne płytki drukowane ze względu na jej przeznaczenie.

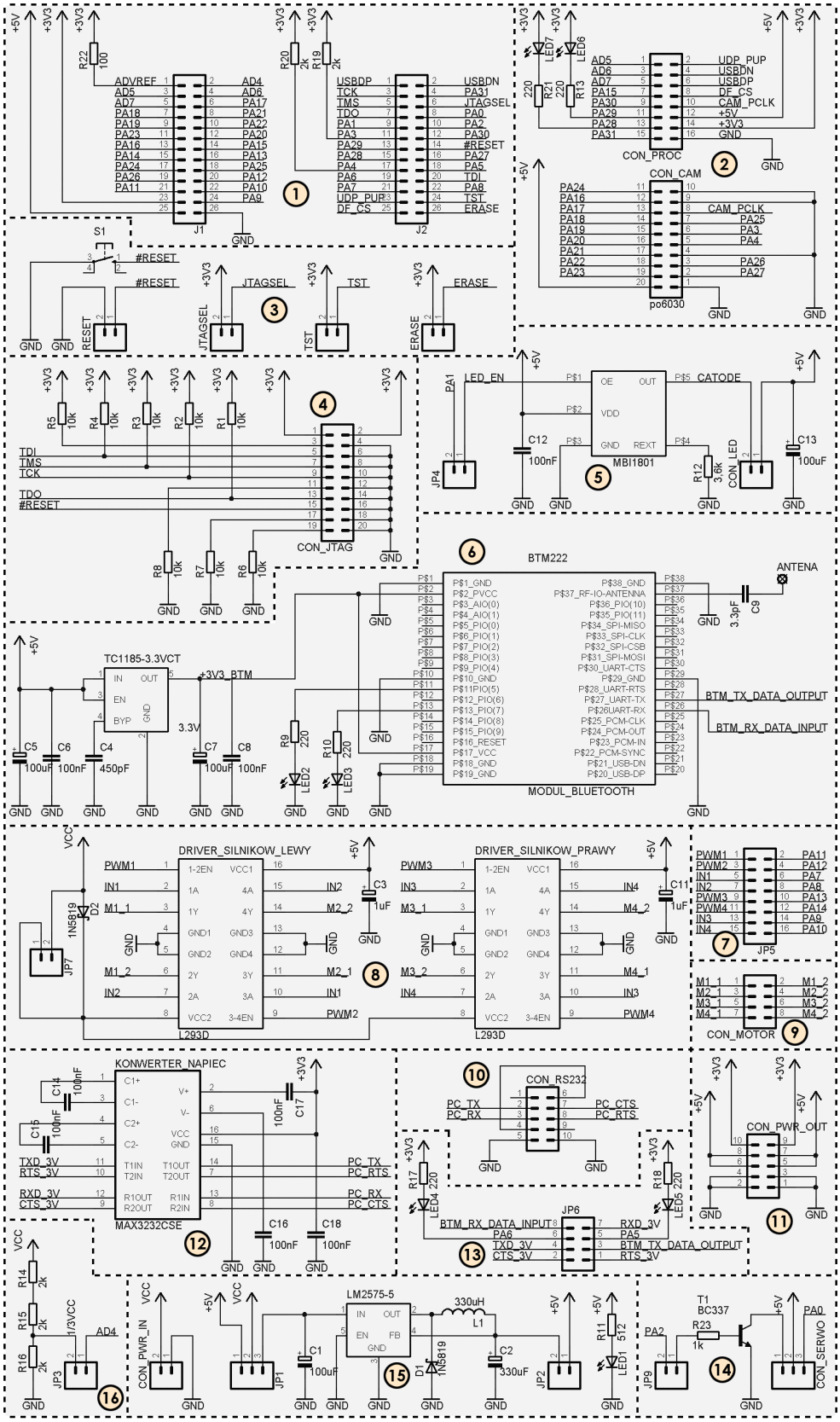
Płyta główna integruje w sobie 32 bitowy mikrokontroler sterujący ARM7 – AT91SAM7S firmy Atmel oraz układy elektroniczne odpowiedzialne za zasilanie, sterowanie urządzeniami wykonawczymi oraz komunikację przewodową i bezprzewodową za pośrednictwem protokołu Bluetooth. Element ten zainstalowany jest w podwoziu robota.

Płytka kamery ma za zadanie zapewnić prawidłowe warunki pracy dla minikamery PO6030K firmy Pixelplus, poprzez dostarczenie niezbędnych elementów zewnętrznych oraz stabilizację napięć zasilających (3.3V, 2.8V, 1.8V). Płytka z kamerą umieszczona jest w ruchomej wieży robota, co daje jej dodatkowy stopień swobody. Obydwie płytki zostały wyposażone w gniazda 20-pinowe, dzięki czemu możliwe jest ich łatwe połączenie za pomocą taśmy IDC.

2.2.1 Schematy ideowe, layout

Do zaprojektowania schematów i masek wykorzystany został program Eagle Layout Editor. Kompletny schemat ideowy płyty głównej robota, przedstawiony został na rysunku 2.5. Na schemacie zaznaczono:

(1) Złącza J1, J2 do podłączenia mikrokontrolera;
(2) Złącze z niewykorzystanymi liniami procesora oraz złącze kamery;
(3) Zworki sterujące pracą procesora;
(4) Interfejs programowania JTAG;
(5) Driver sterujący lampą LED;
(6) Moduł komunikacyjny Bluetooth wraz ze stabilizatorem napięcia 3.3V;



Rysunek 2.5: Schemat ideowy płyty głównej robota.

(7) Zworki konfiguracyjne JP5;

(8) Drivery silników napędowych;

(9) Złącze silników napędowych;

(10) Złącze RS232;

(11) Złącze z wyprowadzonym zasilaniem;

(12) Konwerter poziomów logicznych MAX3232 dla złącza RS232;

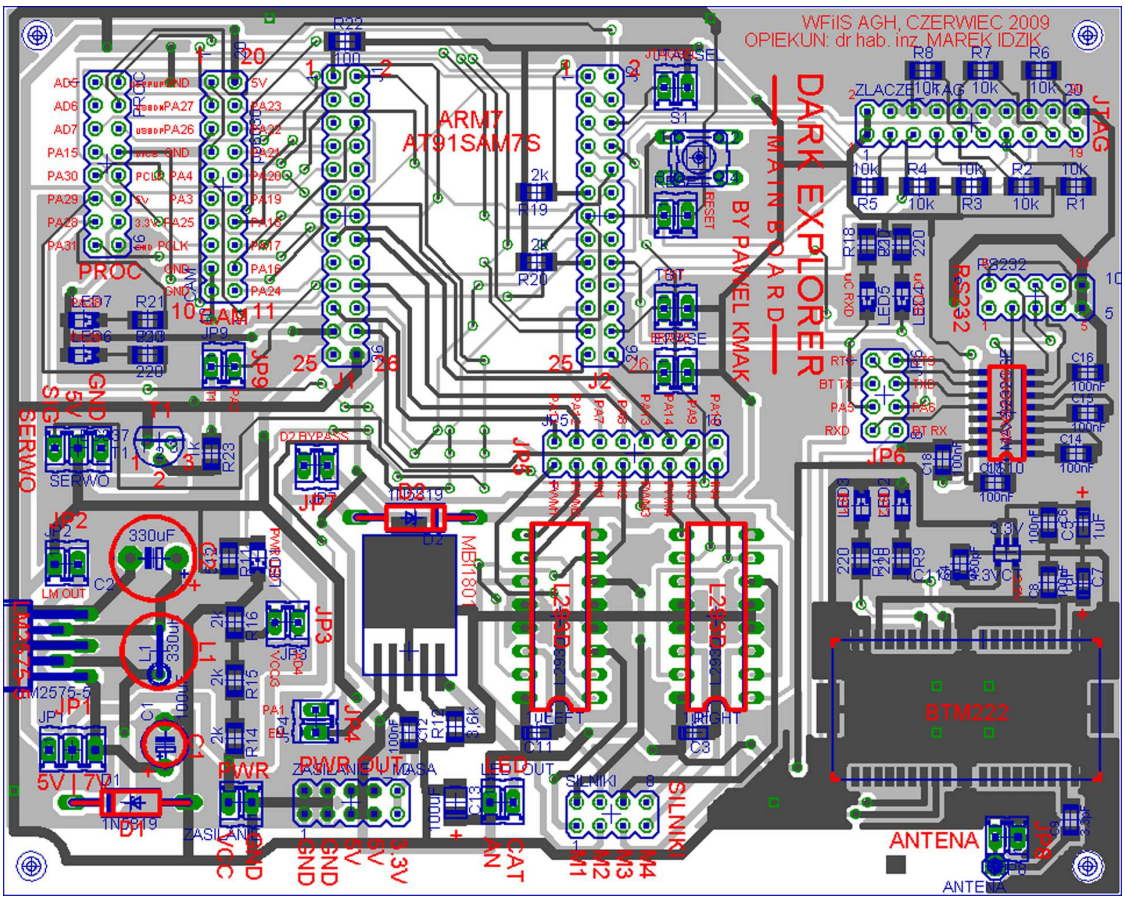
(13) Zworki JP6 konfigurujące pracę interfejsów komunikacyjnych;

(14) Sterownik serwomechanizmu;

(15) Przetwornica napięcia 5V;

(16) Układ do pomiaru napięcia zasilania VCC.

Na podstawie zaprojektowanego schematu stworzono layout płyty głównej robota, pokazany na rysunku 2.6.



Rysunek 2.6: Projekt layout'u płyty głównej. Widoczne warstwy to: górna warstwa metalizacji (kolor ciemnoszary), dolna warstwa metalizacji (kolor jasnoszary), pady lutownicze elementów przewlekanych (kolor zielony), zarysy obudów elementów (kolor niebieski), warstwa opisu (kolor czerwony).

Zainstalowany konwerter poziomów logicznych MAX3232 [12] jest układem awaryjnym, niewykorzystywanym podczas normalnej pracy. Umożliwia on komunikację kablową procesora lub modułu Bluetooth z komputerem PC, poprzez protokół RS232 – na wypadek awarii modułu Bluetooth lub w celach serwisowych czy konfiguracyjnych.

Szczegółowe informacje na temat złącz i zworek konfiguracyjnych, widocznych na schematach, znajdują się w rozdziale 2.3.

Schemat ideowy płytki dla kamery przedstawia rysunek 2.7. Na schemacie zaznaczono:

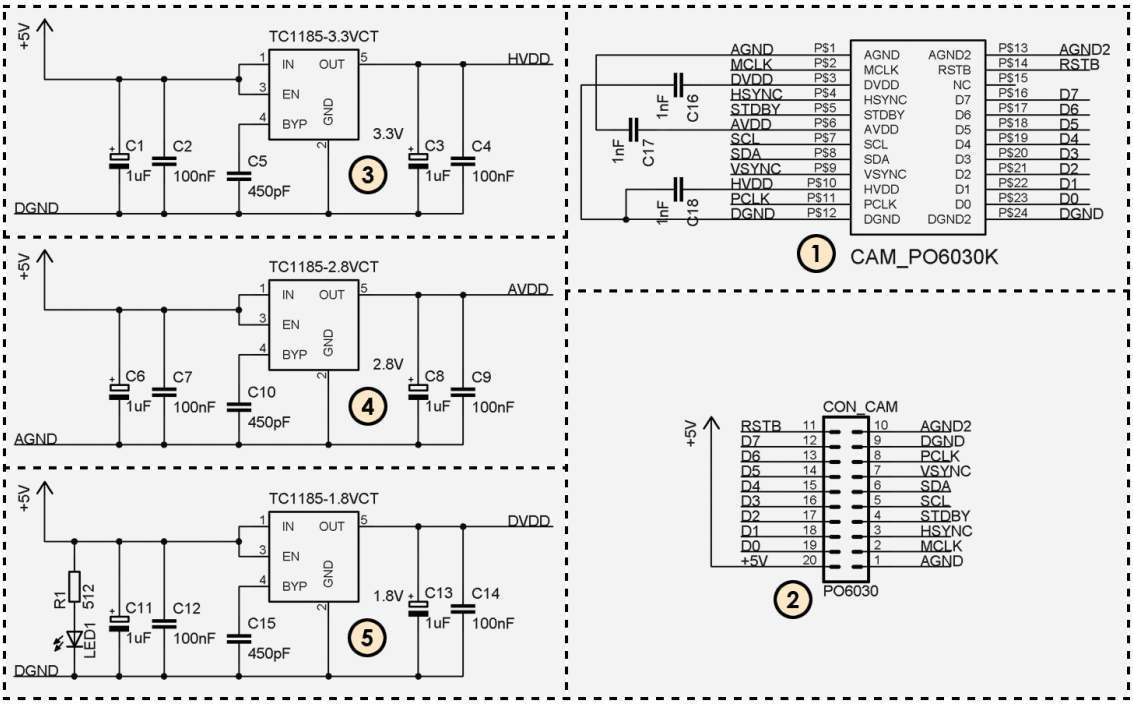
(1) Złącze kamery PO3060K wraz z kondensatorami filtrującymi;

(2) Złącze do połączenia płytki kamery z płytą główną robota;

(3) Stabilizator napięcia 3.3V;

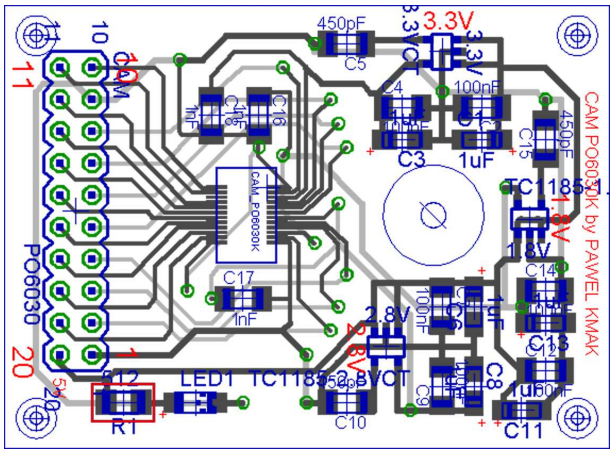
(4) Stabilizator napięcia 2.8V;

(5) Stabilizator napięcia 1.8V.



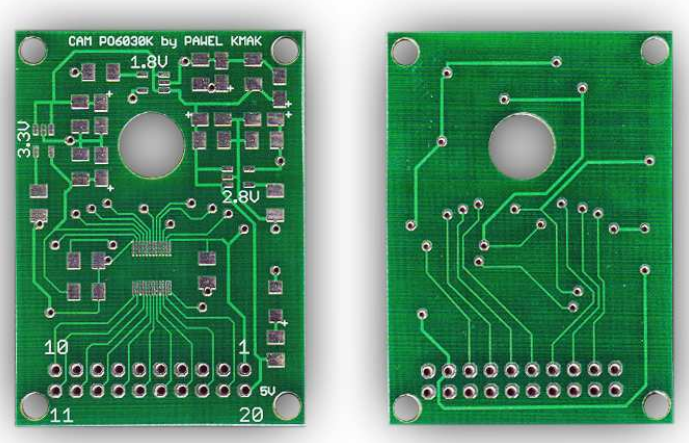
Rysunek 2.7: Schemat ideowy płytki kamery

Layout płytki dla kamery pokazany został na rysunku 2.8.

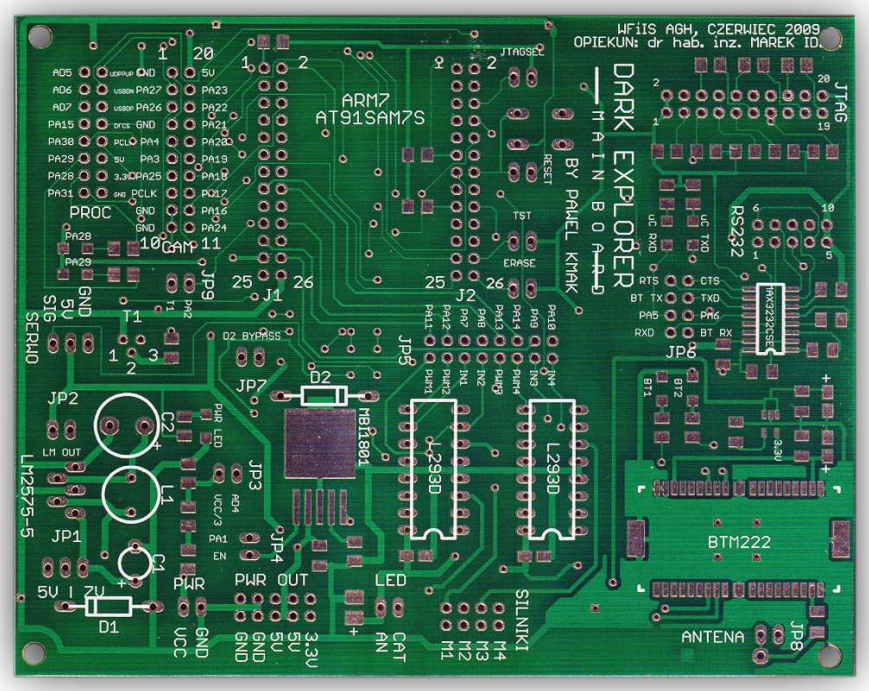


Rysunek 2.8: Projekt layout'u płytki dla miniaturowej kamery PO6030K. Widoczne warstwy to: górna warstwa metalizacji (kolor ciemnoszary), dolna warstwa metalizacji (kolor jasnoszary), pady lutownicze elementów przewlekanych (kolor zielony), zarysy obudów elementów (kolor niebieski), warstwa opisu (kolor czerwony).

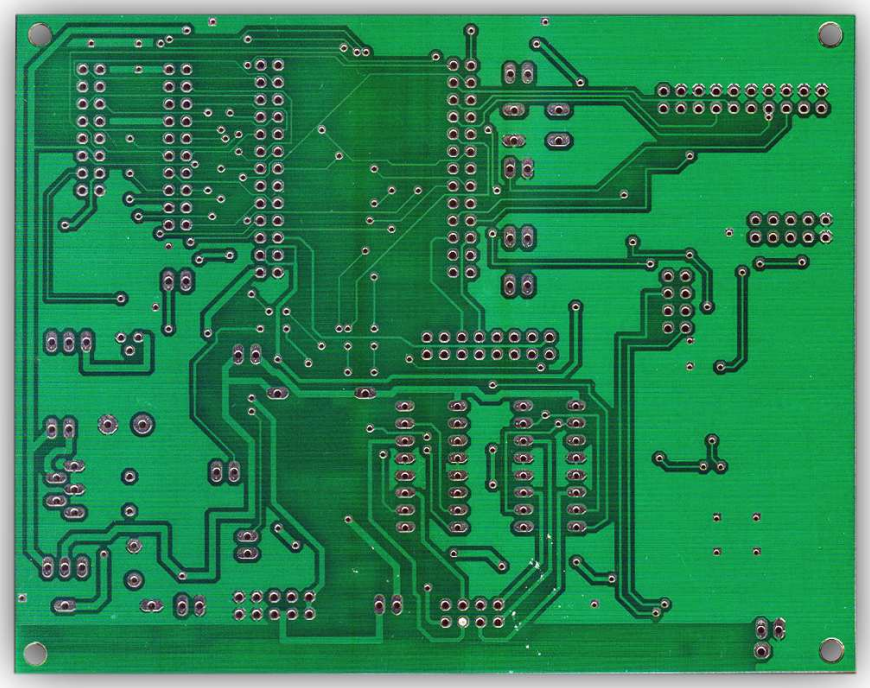
Produkcja płytek została zlecona specjalistycznej firmie i zajęła ok. 3 tygodnie. Rysunki 2.9 – 2.11 przedstawiają wyprodukowane płytki. Wymiary płytek to 4.8 x 3.5cm dla płytki kamery i 12.8 x 9.8cm dla płyty głównej. Po tym etapie można już było przystąpić do lutowania elementów i wykonać pierwsze testy.



Rysunek 2.9: Płytka kamery - warstwy górna i dolna.



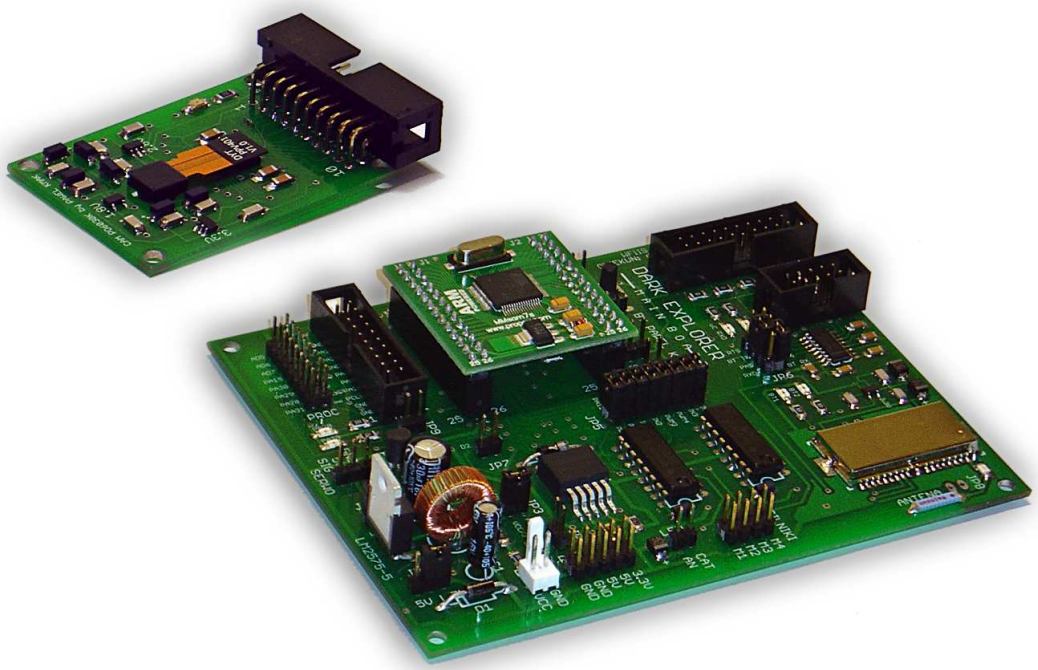
Rysunek 2.10: Płyta główna - warstwa górna.



Rysunek 2.11: Płyta główna - warstwa dolna.

2.2.2 Montaż

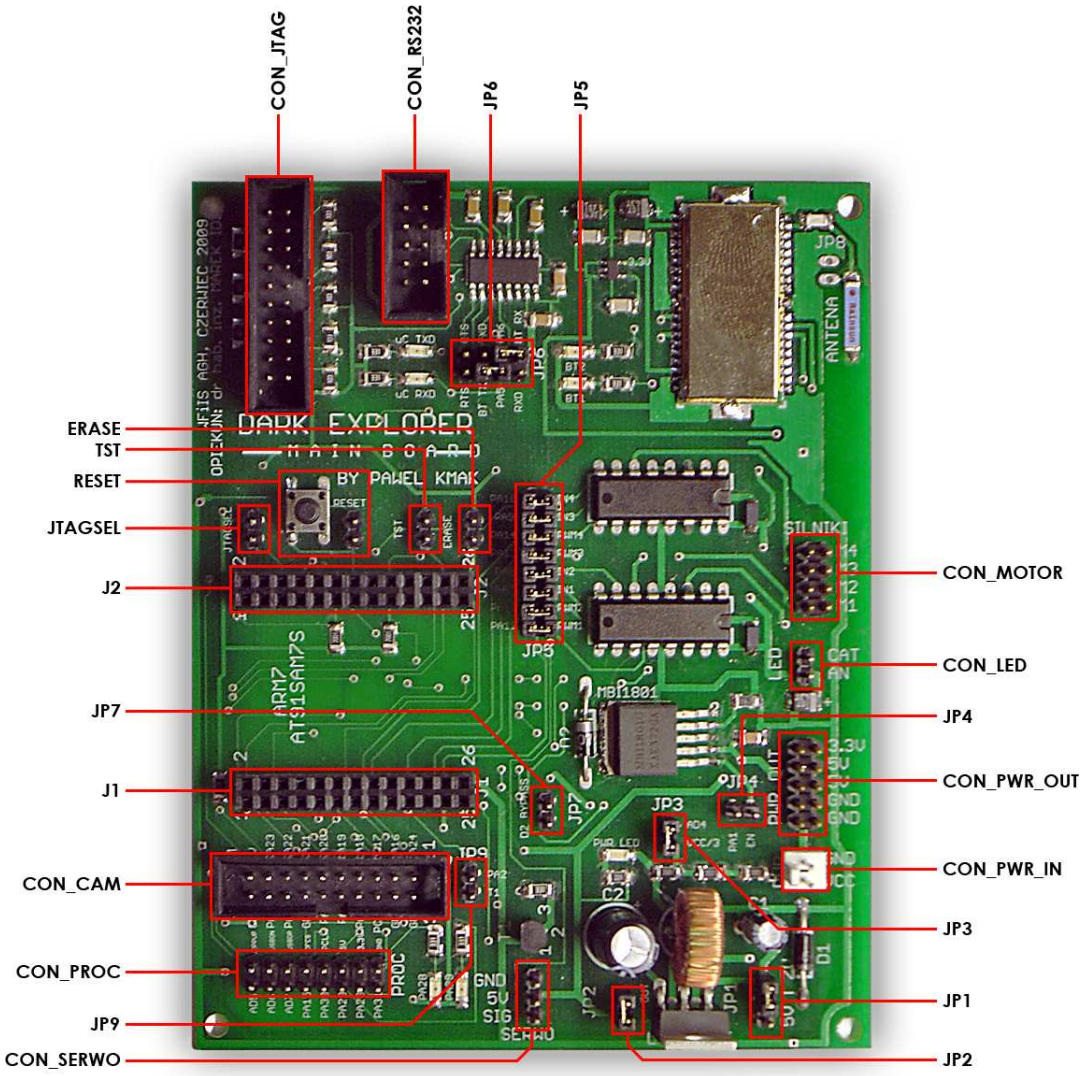
Wszystkie luty zostały wykonane za pomocą standardowej lutownicy grzałkowej. Tam, gdzie to możliwe, wykorzystano elementy do montażu powierzchniowego SMD, w większości w standardowym rozmiarze 1206. Zmontowane płytki pokazano na rysunku 2.12.



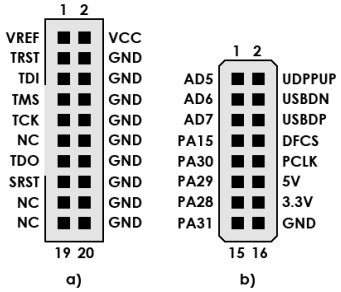
Rysunek 2.12: Zmontowane płyki: 1) Płyta główna robota z zainstalowanym modułem mikrokontrolera; 2) Płytka kamery z zainstalowaną kamerą - widok perspektywiczny.

2.3 Specyfikacja płyty głównej

Płyta główna posiada szereg złącz i zworek konfiguracyjnych które zostały zaznaczone na rysunku 2.13. W tabelach 2.1 – 2.4 oraz na rysunkach 2.14 – 2.16 przedstawiono szczegółową specyfikację.



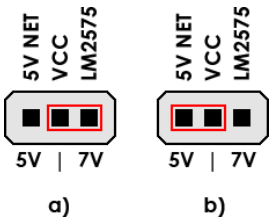
Rysunek 2.13: Zworki konfiguracyjne i złącza płyty głównej robota.



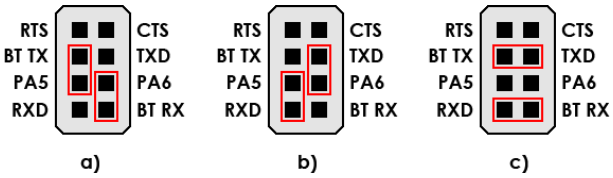
Rysunek 2.14: Rozkład wyprowadzeń złącz płyty głównej. a) Złącze programowania JTAG (CON_JTAG); b) Złącze z pinami mikrokontrolera do zewnętrznego wykorzystania (CON_PROC).

Tabela 2.1: Złącza płyty głównej robota

Nazwa złącza	**Funkcja**
CON_PROC	Złącze wyprowadzające wolne piny mikrokontrolera, które mogą zostać wykorzystane w przyszłej rozbudowie projektu.
CON_JTAG	Standardowe 20-pinowe złącze programowania/debuggowania JTAG, zgodne z obowiązującym standardem. Złącze zostało również wyprowadzone na zewnątrz obudowy.
CON_RS232	Złącze umożliwiające połączenie kablowe z komputerem za pośrednictwem interfejsu RS232.
CON_CAM	Złącze do połączenia płyty głównej z płytką kamery PO6030K.
CON_MOTOR	Złącze dla 4 silników DC.
CON_LED	Złącze dla diody świecącej LED, zainstalowanej w wieży.
CON_PWR_OUT	Napięcia zasilania 3.3V, 5V oraz masa do zewnętrznego wykorzystania.
CON_PWR_IN	Złącze zasilania. Napięcie wejściowe powinno mieścić się w przedziale 6 – 9 V.
CON_SERWO	Standardowy konektor dla serwomechanizmu analogowego.
J1, J2	Złącza dla modułu MMsam7s [13] z procesorem AT91SAM7S [5].



Rysunek 2.15: Możliwe konfiguracje zworki JP1.



Rysunek 2.16: Możliwe konfiguracje zworki JP6.

Tabela 2.2: Zworki konfiguracyjne płyty głównej robota

Zworka	**Funkcja**
JP1	Zworka wyboru napięcia zasilającego. Możliwe konfiguracje pokazane na rys. 2.15 a, b. W konfiguracji (a) napięcie zasilające powinno być z zakresu 6 – 9V. W konfiguracji (b) napięcie musi wynosić 5V (napięcie powyżej 6V spowoduje uszkodzenie układów). Domyślnie: (a)
JP2	Zworka odłączająca wyjście przetwornicy LM2575 od sieci 5V. Domyślnie: zamknięta
JP3	Podłącza 1/3 napięcia zasilającego do kanału AD4 przetwornika analogowo cyfrowego. Umożliwia pomiar napięcia akumulatorów. Domyślnie: zamknięta
JP4	Podłącza sterowanie układem MBI1801 do portu PA1 mikrokontrolera. Umożliwia sterowanie diodą LED. Domyślnie: zamknięta
JP5	Łączy linie PA7 – PA14 ze sterownikami silników L293D. Domyślnie: zamknięta
JP6	Ustawia tryb pracy interfejsów komunikacyjnych. Możliwe konfiguracje pokazane na rys. 2.16 a, b, c. W konfiguracji (a) komunikacja za pośrednictwem interfejsu Bluetooth, (b) komunikacja przewodowa RS232, (c) tryb serwisowy podłączający moduł Bluetooth do złącza RS232 płyty głównej. Domyślnie: (a)
JP7	Obejście diody D2 ograniczającej napięcie podawane na silniki. Zworkę należy zamknąć w wypadku zasilania niskim napięciem (5–6V). Domyślnie: otwarta

Tabela 2.3: Zworki konfiguracyjne płyty głównej robota (kontynuacja)

Zworka	**Funkcja**
JP9	Zamknięcie zworki powoduje zwarcie bazy tranzystora T1 z linią PA2 procesora. Umożliwia to sterowanie zasilaniem serwa. Zasilanie jest włączone gdy na bazę podamy stan wysoki. Domyślnie: zamknięta
RESET	Przycisk resetu mikrokontrolera z dodatkową zworką umożliwiającą wyprowadzenie zewnętrznego przycisku. Domyślnie: otwarta
ERASE	Założenie zworki powoduje skasowanie zawartości pamięci Flash mikrokontrolera oraz bitów NVM. Domyślnie: otwarta
TST	Zamknięcie zworki i podłączenie pinów PA0–PA2 do wysokiego poziomu logicznego podczas resetu procesora powoduje wejście do bootloadera SAM-BA. Domyślnie: otwarta
JTAGSEL	Zmiana trybu pracy interfejsu JTAG. Otwarcie powoduje włączenie trybu emulacji w systemie (ICE), zamknięcie powoduje włączenie trybu Boundary Scan. Domyślnie: otwarta

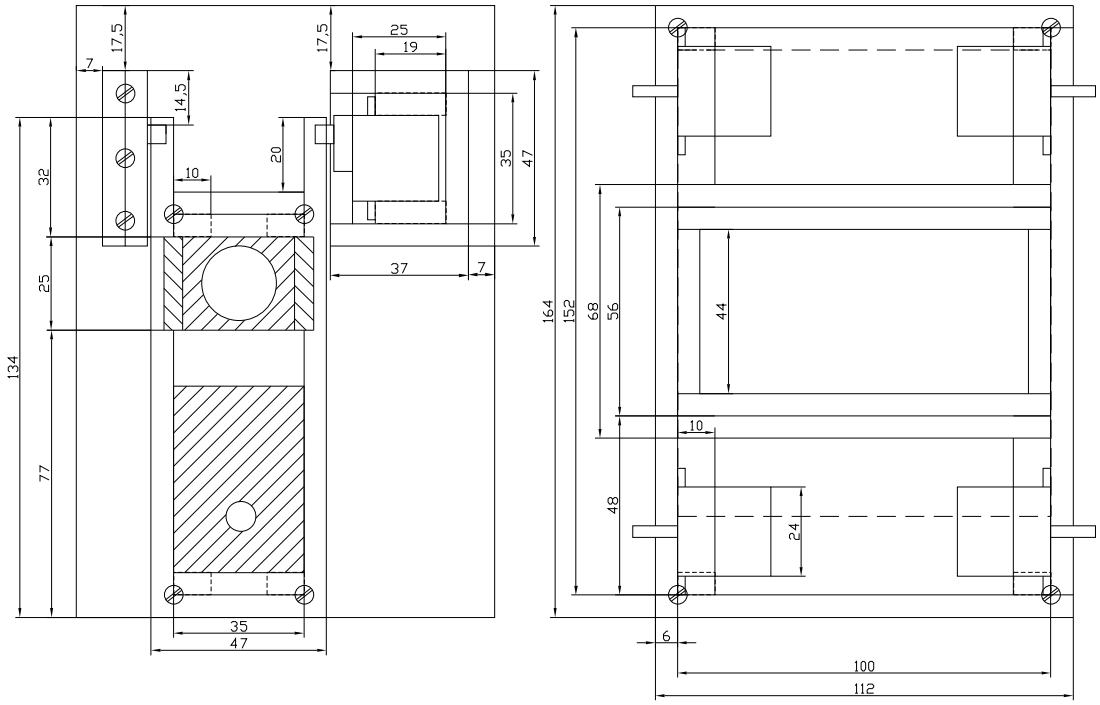
Ponadto płyta główna została wyposażona w 7 sygnalizacyjnych diod LED, informujących o stanie całego systemu oraz o aktualnie wykonywanych zadaniach.

Tabela 2.4: Diody sygnalizacyjne

Dioda	**Funkcja**
PWR LED	Informuje o włączonym zasilaniu.
BT1	Informuje o stanie połączenia Bluetooth. Gdy mruga - brak połączenia, świecenie ciągłe oznacza aktywne połączenie.
BT2	Zapala się, gdy moduł Bluetooth odbiera lub wysyła dane.
uC RXD	Monitoruje kanał odbiorczy interfejsu UART mikrokontrolera.
uC TXD	Monitoruje kanał nadawczy interfejsu UART mikrokontrolera.
PA28, PA29	Diody podłączone do pinów mikrokontrolera, mogą pełnić dowolną funkcję sygnalizacyjną. Domyślnie informują o aktywności kamery PO6030K.

2.4 Część mechaniczna

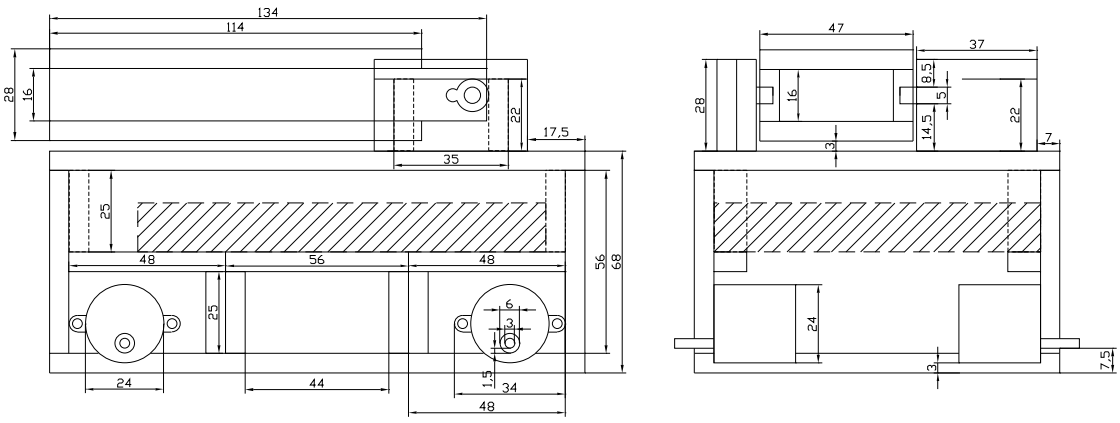
Aby można było zintegrować razem elementy tworzące robota (płytki drukowane, silniki, akumulatory itp.), należało wykonać obudowę. Została ona wykonana z 45 elementów wyciętych ze spienionego PCW o grubości 6mm. Wierzchnia pokrywa została wycięta z pleksi przezroczystej. Projekt stworzony w programie AutoCAD przedstawiony jest na rysunkach 2.17, 2.18.



Rysunek 2.17: Projekt obudowy robota, rzut z góry. Po lewej wieża z kamerą, po prawej podwozie.

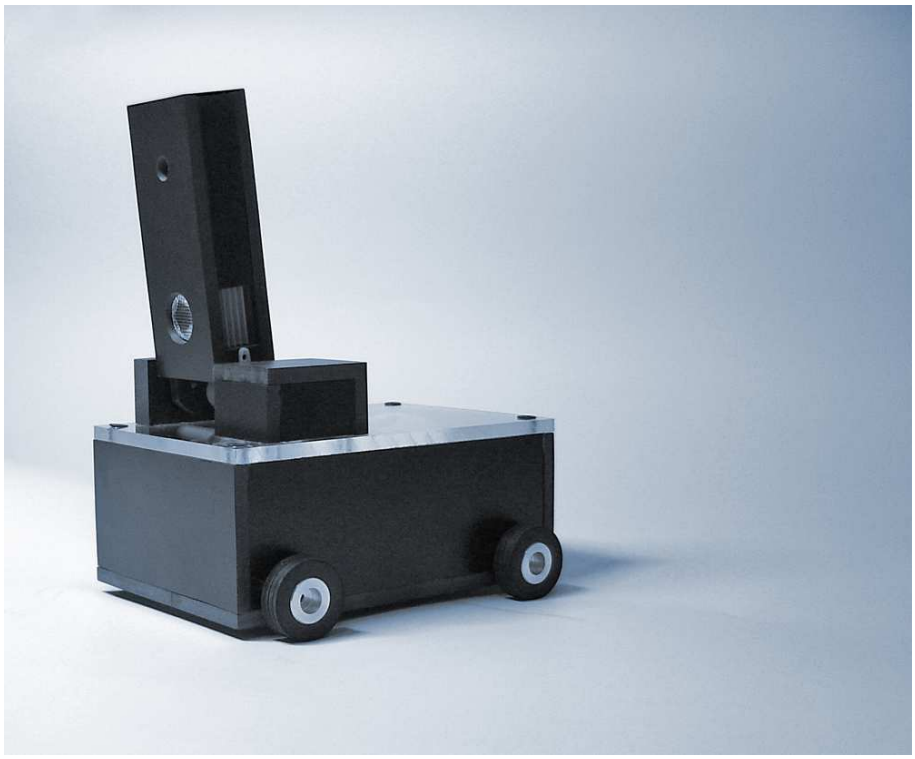
Obudowa składa się z podwozia oraz z zamontowanej na nim ruchomej wieży. W podwoziu zainstalowane zostały 4 silniki wraz z kołami oraz płyta główna. Miejsce pomiędzy silnikami wykorzystane zostało na stworzenie łatwo dostępnej komory na akumulatory zasilające. Na tylnym panelu umieszczone zostały:

- włącznik zasilania
- przycisk reset
- złącze programowania JTAG
- złącze RS232 (do celów serwisowych)



Rysunek 2.18: Projekt obudowy robota, rzut z boku (po lewej), rzut z przodu (po prawej).

Wieża zamontowana jest na pokrywie podwozia. Może obracać się wokół jednej osi w granicach 0–180°. W wieży zamontowana jest płytka z kamerą oraz lampa LED. Za zmianę kąta położenia wieży odpowiada miniaturowy serwomechanizm. Zmontowany robot pokazany został na rysunku 2.19.



Rysunek 2.19: Zmontowany robot z zainstalowanymi wszystkimi podzespołami.

Rozdział 3

Metody analizy i rozpoznawania obrazów

Aby obraz wykorzystać jako źródło informacji, należy wykonać szereg operacji, które w efekcie pozwolą na określenie jego konkretnych cech. Pierwszą czynnością jest oczywiście akwizycja obrazu i przekształcenie go na postać cyfrową. Następnie odbywa się przetwarzanie pozyskanego obrazu w celu przygotowania go do analizy. Podczas przetwarzania poprawiana jest jakość obrazu oraz eksponowane są jego najistotniejsze z punktu widzenia procesu rozpoznawania elementy. W kolejnym etapie zwanym analizą, obraz jest w procesie segmentacji ostatecznie dzielony na odrębne elementy. Potem dla każdego obiektu wyznaczane są cechy opisujące jego kształt i wygląd, co kończy analizę. Po tym etapie dalsza praca opiera się już o dane liczbowe – obraz jest teraz reprezentowany przez grupy liczb (zwane wektorami cech), które opisują poszczególne obiekty pierwotnego obrazu. Samo rozpoznawanie polega na określeniu przynależności obiektów obrazu do konkretnych klas, co w najprostszym wydaniu może być zrealizowane, poprzez porównanie wektorów cech badanych obiektów z wektorami cech obiektów wzorcowych.

Dokładniejsza systematyka omawianych procesów wygląda następująco [2]:

1. Akwizycja obrazu w postaci cyfrowej
2. Przetwarzanie obrazu
    - przekształcenia geometryczne
    - przekształcenia punktowe (bezkontekstowe)
    - przekształcenia kontekstowe - filtry
    - przekształcenia widmowe

- przekształcenia morfologiczne

3. Analiza obrazu
   - segmentacja
   - indeksacja
   - pomiary
   - obliczenie współczynników kształtu
4. Rozpoznawanie obrazu
   - klasyfikacja poszczególnych obiektów

Poniżej zostały ogólnie omówione najważniejsze etapy przetwarzania, analizy i rozpoznawania obrazu. Ponadto dla wybranych etapów, pokazano przykłady zastosowania konkretnych metod, które będą pomocne w wyjaśnieniu zasady działania całego systemu akwizycji, przetwarzania, analizy i rozpoznawania obrazu, zaprojektowanego dla robota mobilnego. System rozpoznawania robota został dokładnie opisany w kolejnym rozdziale pracy.

3.1 Akwizycja obrazu

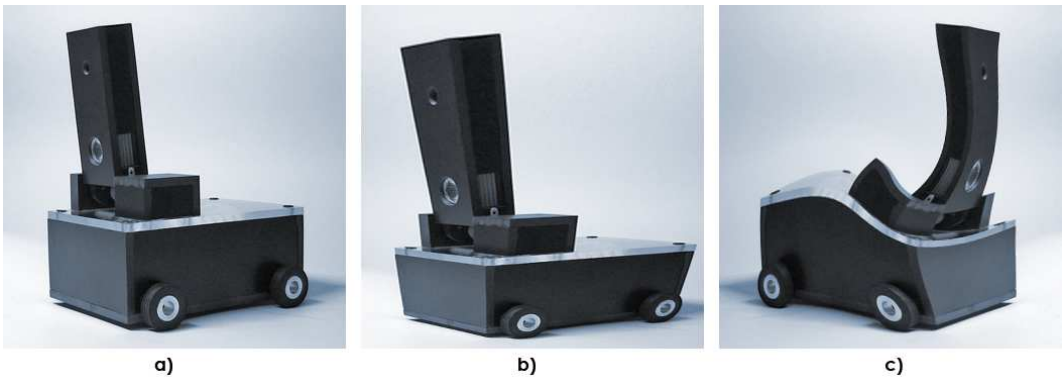
Polega na odebraniu klatki obrazu z urządzenia zewnętrznego (najczęściej kamery CCD/CMOS) i zapisaniu jej w pamięci systemu rozpoznawania w sposób cyfrowy, umożliwiający łatwy dostęp do poszczególnych pikseli obrazu. Obraz może być kolorowy lub w odcieniach szarości. Ważnym parametrem obrazu jest jego rozdzielczość. Zbyt mała spowoduje znaczne zniekształcenia obrazu i rozpoznawanie będzie obarczone dużym błędem lub w ogóle niemożliwe. Z kolei zbyt duża rozdzielczość znacznie wydłuży czas potrzebny na przetwarzanie i analizę, co może spowodować, że cały system będzie bezużyteczny z powodu zbyt dużych opóźnień.

3.2 Przetwarzanie obrazu

Przekształcenia geometryczne

Najważniejsze przekształcenia geometryczne to przesunięcia, obroty, odbicia oraz różnego rodzaju zniekształcenia. Najczęściej wykorzystywane są do korekcji błędów

geometrii obrazu, wynikłych np. z powodu niedoskonałości obiektywu. Przykładowe przekształcenia pokazane zostały na rysunku 3.1.



Rysunek 3.1: a) Oryginalny obraz; b) Obraz po deformacji geometrycznej; c) Obraz odbity lustrzanie i zdeformowany;

Przekształcenia punktowe

Polegają na przeprowadzeniu operacji na poszczególnych punktach obrazu, niezależnie od stanu elementów sąsiednich. Modyfikowane są jedynie wartości poszczególnych punktów (np. kolory, jasność), relacje geometryczne pozostają niezmienione. Przykładowe przekształcenia punktowe to: zmiana jasności obrazu, wyrównanie histogramu czy też binaryzacja obrazu. Na szczególną uwagę zasługuje binaryzacja, gdyż większość algorytmów analizy obrazu pracuje na obrazie binarnym.

Binaryzacja

Celem binaryzacji jest redukcja ilości informacji. Binaryzacja polega na zamianie obrazu mającego wiele odcieni szarości, na obraz którego piksele mogą mieć jedynie wartości 1 lub 0. Najczęściej stosowana są [2]:

- binaryzacja z dolnym progiem

$$L'(i,j) = \begin{cases} 0;\ L(i,j) \leqslant a \\ 1;\ L(i,j) > a \end{cases} \tag{3.1}$$

gdzie:

$L(i,j)$ - jasność punktu na obrazie źródłowym
$L'(i,j)$ - wartość punktu na obrazie wynikowym
a - próg

- binaryzacja z górnym progiem

$$L'(i,j) = \begin{cases} 0;\ L(i,j) \geqslant a \\ 1;\ L(i,j) < a \end{cases} \tag{3.2}$$

- binaryzacja z podwójnym ograniczeniem

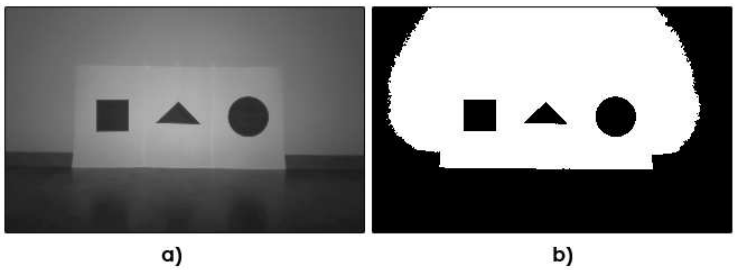
$$L'(i,j) = \begin{cases} 0;\ L(i,j) \leqslant a_1 \\ 1;\ a_1 < L(i,j) \leqslant a_2 \\ 0;\ L(i,j) > a_2 \end{cases} \tag{3.3}$$

gdzie:

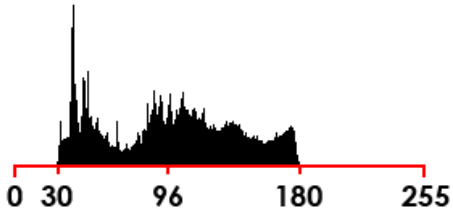
a_1, a_2 - to progi binaryzacji takie, że $a_1 < a_2$

Dobór progu binaryzacji a jest kluczowy dla tego przekształcenia. W większości przypadków próg wybiera się na podstawie histogramu obrazu (rys. 3.3), ustawiając wartość progową a w minimum lokalnym, pośrodku 2 lokalnych maksimów. W ten sposób stosując binaryzację z dolnym lub górnym progiem, można z obrazu wydobyć obszary o jasności mniejszej lub większej od progu. Binaryzacja z podwójnym progiem jest odpowiednikiem zastosowania filtracji z dolnym i górnym progiem jednocześnie. Przykład zastosowania binaryzacji pokazano na rysunku 3.2.

W przypadku obrazów kontrastowych, zawierających ciemne obiekty na jasnym tle (lub odwrotnie), wystarczające może okazać się ustawienie progu na wartość, będącą średnią jasnością obrazu. Pozwoli to nieco uprościć algorytm doboru progu (brak konieczności generowania histogramu i wyszukiwania odpowiedniego minimum).



Rysunek 3.2: a) Oryginalny obraz odebrany z kamery; b) Obraz po binaryzacji z dolnym progiem (wzór 3.1) – piksele o wartości 1 zaznaczono kolorem czarnym;



Rysunek 3.3: Histogram obrazu z rysunku 3.2 (a). Zaznaczono minimalną, maksymalną oraz średnią jasność występującą na zdjęciu. Średnia jasność jest progiem binaryzacji.

Przekształcenia kontekstowe

Polegają na zmianie wartości danego punktu obrazu w oparciu o jego otoczenie. Innymi słowy wartość danego punktu obrazu jest obliczana przez funkcję, której argumentami są wartości pikseli sąsiednich. Za punkty sąsiednie najczęściej przyjmuje się 8 pikseli otaczających ten aktualnie przetwarzany – w przypadku pracy na siatce kwadratowej. Przekształcenia kontekstowe często nazywane są filtrami [2].

Przykładowe zastosowania tych przekształceń to tłumienie szumu, wykrywanie krawędzi czy usuwanie różnych specyficznych wad obrazu.

Przekształcenia widmowe

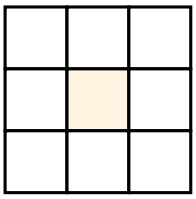
Związane są ściśle z interpretacją częstotliwościową obrazu. Przeważnie opierają się na transformacji Fouriera rozszerzonej na 2 wymiary. Podejście to okazało się owocne dla pewnych specjalistycznych zastosowań, między innymi przy przetwarzaniu obrazów medycznych, satelitarnych, astronomicznych czy przy analizie tekstur [2].

Przekształcenia te jednak nie są szeroko stosowane w dziedzinie przetwarzania obrazów, pomimo popularności zastosowania transformacji Fouriera w przetwarzaniu sygnałów jednowymiarowych. Interpretacja częstotliwościowa obrazu jest sztuczna, a wyniki manipulacji w dziedzinie częstotliwościowej dość nieprzewidywalne.

Przekształcenia morfologiczne

Przekształcają te punkty obrazu, których otoczenie jest zgodne z elementem strukturalnym obrazu, zdefiniowanym dla danego przekształcenia. Elementem strukturalnym jest wycinek obrazu (przy siatce kwadratowej najczęściej jest to kwadrat

3x3 piksele – rys. 3.4), z wyróżnionym elementem centralnym.



Rysunek 3.4: Element strukturalny obrazu dla siatki kwadratowej

Przekształcenie polega na sprawdzeniu zgodności otoczenia każdego punktu obrazu z elementem strukturalnym, przy założeniu, że aktualnie badany punkt jest centralnym punktem elementu strukturalnego. Jeśli konfiguracja pikseli obrazu jest zgodna z elementem strukturalnym, wykonywana jest, ustalona wcześniej, operacja na badanym punkcie. Zazwyczaj jest to po prostu zmiana jasności. Typowe przekształcenia morfologiczne to erozja, dylatacja, ścienianie, szkieletyzacja. Bardziej skomplikowane przekształcenia utworzyć można poprzez kolejne wykonywanie kilku prostszych [2].

Przekształcenia morfologiczne dają duże możliwości, pozwalają na szczegółową kontrolę wprowadzanych zmian w obrazie – a to dzięki temu, że modyfikują jedynie te punkty, których otoczenie jest zgodne z elementem strukturalnym. Łączenie prostych przekształceń daje możliwość tworzenia skomplikowanych i bardzo wyspecjalizowanych operacji. Przekształcenia te mają jednak wadę - dużą złożoność obliczeniową. Podczas pojedynczego przebiegu po obrazie konieczne jest sprawdzenie wielu punktów, a do wykonania większości przekształceń potrzeba więcej niż jednego przebiegu.

Wartości poszczególnych pikseli elementu strukturalnego definiuje się według konwencji pokazanej na rysunku 3.5 [2]:

$$\begin{bmatrix} 1 & X & X \\ X & 0 & X \\ X & X & X \end{bmatrix}$$

Rysunek 3.5: Przykładowy element strukturalny.

gdzie:

0 - to piksel o jasności mniejszej od wartości progu („zgaszony”)

1 - to piksel o jasności większej od wartości progu („zapalony”)

X - to piksel o dowolnej jasności

Jeżeli punkt centralny elementu strukturalnego jak i jego otoczenie są zgodne z aktualnie badanym wycinkiem obrazu, to wykonywane jest zdefiniowane przekształcenie na punkcie wycinka obrazu odpowiadającym punktowi centralnemu elementu strukturalnego.

3.3 Analiza obrazu

3.3.1 Segmentacja

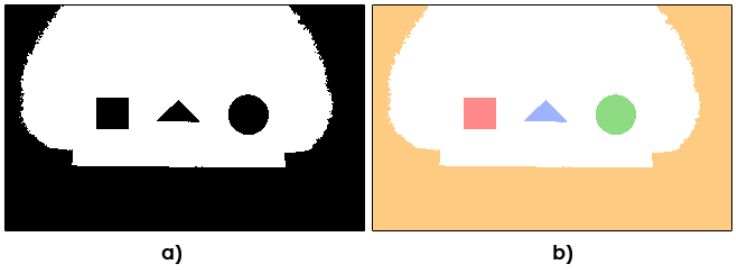
Po wstępnym przetworzeniu obrazu, konieczny jest jego podział na fragmenty odpowiadające poszczególnym obiektom widocznym na obrazie – proces ten zwany jest segmentacją obrazu. Wydzielanie obszarów z obrazu źródłowego, odbywa się w oparciu o kryteria jednorodności takie jak kolor, poziom jasności czy faktura.

Opracowano wiele technik segmentacji [2], odpowiednich dla różnych typów analizowanych obrazów. Jedne działają w oparciu o podział obrazu na coraz mniejsze segmenty, poprzez rekursywne progowanie. Inne w oparciu o wykrywanie krawędzi obiektów, jeszcze inne opierają się na statystyce. Na potrzeby niniejszej pracy, wystarczająca okazała się segmentacja w oparciu o jednokrotne progowanie obrazu (binaryzację). Operacja ta została opisana w rozdziale 3.2 a efekty pokazano na rysunku 3.2.

3.3.2 Indeksacja

Celem indeksacji jest przypisanie wszystkim pikselom obrazu po segmentacji etykiet, informujących o tym, do jakiego obiektu dany piksel należy. Etykieta jest liczbą skojarzoną z danym pikselem, zapisywaną najczęściej na niewykorzystanych bitach piksela. Do zapamiętania obrazu binarnego wystarcza 1 bit na piksel, natomiast na ten cel zazwyczaj przeznacza się minimum 1 bajt – 7 wolnych bitów można przeznaczyć na etykietę. Wszystkie piksele wchodzące w skład danego obiektu muszą tworzyć obszar spójny. Obraz powstający po indeksacji pokazano na rysunku 3.6b.

Technik indeksacji jest wiele. Najprostszą metodą dającą zawsze poprawny wynik jest rekurencyjny algorytm powodziowy. Algorytm przyjmuje na wejście współrzędne jednego piksela. Aby rozpocząć przypisywanie pikseli do obiektu należy uruchomić algorytm, wskazując dowolny punkt startowy. Algorytm działa w następujący sposób:



Rysunek 3.6: a) Obraz po segmantacji; b) Wynik indeksacji przeprowadzonej na obrazie (a). Kolory reprezentują tu różne etykiety;

1. Algorytm sprawdza czy piksel o podanych współrzędnych ma wartość 1.

2. Jeśli warunek 1 jest spełniony, algorytm ustawia pikselowi pierwszą wolną etykietę (poprzez zmianę wartości piksela) i wywołuje sam siebie 8 razy dla wszystkich pikseli sąsiadujących z aktualnie badanym.

Algorytm zakończy się, kiedy wszystkie piksele tworzące dany obszar spójny otrzymają swoją etykietę. Algorytm należy wywoływać do momentu, aż obraz nie będzie posiadał pikseli bez przypisanej etykiety.

Powyższy algorytm daje poprawny wynik, lecz ze względu na to, iż jest to algorytm rekurencyjny, ma dwie poważne wady: sprawdza kilkukrotnie te same punkty oraz potrzebuje bardzo dużego stosu. Wady te definitywnie wykluczają jego użycie – szczególnie w systemach embedded, gdzie ilość pamięci operacyjnej jest mocno ograniczona.

Rozwiązaniem jest algorytm Smitha [4], nie posiadający wspomnianych wad i wykonujący dokładnie to samo zadanie, co przedstawiony wcześniej rekurencyjny algorytm powodziowy. Poprawa efektywności została osiągnięta, dzięki operowaniu na poziomych segmentach obrazu a nie na pojedynczych pikselach. Segmentem tutaj jest poziomy ciąg „zapalonych" pikseli, o maksymalnej możliwej długości, ograniczony z obydwu stron pikselami o wartości 0. Działanie algorytmu polega na indeksowaniu całych segmentów przetwarzanego obszaru – w górę i w dół od pozycji początkowej. Algorytm Smitha nie jest rekurencyjny. Również wykorzystuje stos, jednak jego wymagania pamięciowe są znacznie mniejsze. Działa znacznie szybciej od algorytmu rekurencyjnego.

Po wykonaniu indeksacji, dla wszystkich obiektów można wykonać indywidualne pomiary, gdyż znane są granice dla każdego z nich.

3.3.3 Pomiary obiektów

Wykonanie pomiarów jest finalnym etapem analizy obrazu. Polegają one na przyporządkowaniu wszystkim wydzielonym z obrazu obiektom, pewnych wielkości liczbowych opisujących je. Wszelkie dalsze operacje wykonywane są już wyłącznie w oparciu o wyznaczone tu parametry.

Aby wielkości opisujące obiekty były przydatne podczas procesu rozpoznawania, muszą one [2]:

- przyjmować jednakowe wartości dla obiektów mających ten sam kształt (np. trójkąty o różnej wielkości)
- dobrze różnicować obiekty o różnym kształcie (np. trójkąty i kwadraty)
- wykazywać niezmienniczość względem typowych przekształceń obrazu - obrotów, translacji, zmiany skali

Wielkości otrzymane poprzez wykonanie bezpośrednich pomiarów na obrazie (pole, obwód itp.) zazwyczaj nie spełniają wymaganych założeń, dlatego parametry te są odpowiednio dobierane i kombinowane w sposób zapewniający spełnienie stawianych wymagań. Tak powstają wielkości nazywane współczynnikami kształtu.

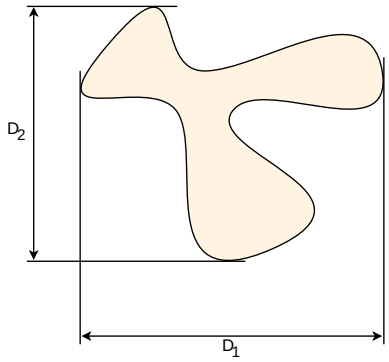
Dalej przedstawiono sposób obliczania kilku parametrów bezpośrednich, koniecznych do późniejszego wyznaczenia współczynników kształtu.

Pole powierzchni

Jest chyba najprostszym do obliczenia parametrem. Polega na zliczeniu pikseli wchodzących w skład danego obszaru. Pomiar może być bardzo dokładny, aczkolwiek jego wartość jest w znacznym stopniu uzależniona od sposobu segmentacji i binaryzacji obrazu.

Średnice Fereta

Opisują rozciągłość obiektu w poziomie i pionie. Również bardzo proste do obliczenia – wystarczy policzyć różnicę odpowiednich maksymalnych i minimalnych współrzędnych punktów, wchodzących w skład obiektu. Dokładniej pokazuje to rysunek 3.7.



Rysunek 3.7: Średnice Fereta: D_1 – pozioma, D_2 – pionowa.

Momenty bezwładności

Obliczając momenty bezwładności pierwszego rzędu (M_{1X}, M_{1Y}) można wyznaczyć położenie środka ciężkości obiektu. Momenty drugiego rzędu (M_{2X}, M_{2Y}, M_{2XY}) są miarą bezwładności obiektu [2].

$$M_{1X} = \frac{1}{P(F)} \sum_{F} x_i \tag{3.4}$$

$$M_{1Y} = \frac{1}{P(F)} \sum_{F} y_i \tag{3.5}$$

$$M_{2X} = \frac{1}{P(F)} \sum_{F} (x_i - M_{1X})^2 \tag{3.6}$$

$$M_{2Y} = \frac{1}{P(F)} \sum_{F} (y_i - M_{1Y})^2 \tag{3.7}$$

$$M_{2XY} = \frac{1}{P(F)} \sum_{F} (x_i - M_{1X})(y_i - M_{1Y}) \tag{3.8}$$

gdzie:

F - analizowany obiekt

$P(F)$ - pole powierzchni obiektu F

(x_i, y_i) - współrzędne poszczególnych pikseli obiektu

Obwód

Dokładne wyznaczenie obwodu figury jest sprawą trudną. Rozwiązaniem nasuwającym się automatycznie jest zliczenie punktów brzegowych badanego obszaru, lecz metoda ta jest bardzo niedokładna, szczególnie dla małych figur.

Ciekawym rozwiązaniem jest zastosowanie formuły Croftona [2] dla siatki kwadratowej. Wzór pozwala obliczyć obwód figury, poprzez sumowanie rzutów figury w czterech kierunkach (0°, 45°, 90°, 135°). Metoda uwzględnia również różnice w odległościach, występujące między punktami dla różnych kierunków rzutowania.

$$L = \frac{\pi}{4} \cdot [a \cdot (N_0 + N_{90}) + \frac{a}{\sqrt{2}} \cdot (N_{45} + N_{135})] \tag{3.9}$$

gdzie:

L - obwód figury

a - odległość między punktami siatki

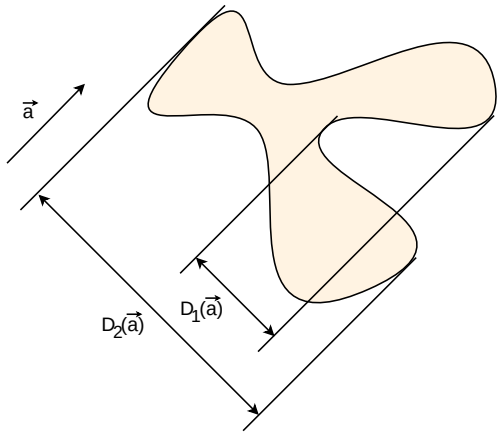
$N_0, N_{90}, N_{45}, N_{135}$ - długości rzutów figury dla danych kątów

Metoda nie daje dokładnych wyników, z powodu wykorzystania jedynie czterech kierunków rzutowania, ale jest szybka i prosta w realizacji. Dalej pokazano sposób na wyznaczenie rzutów $N_0 - N_{135}$.

Długości rzutów

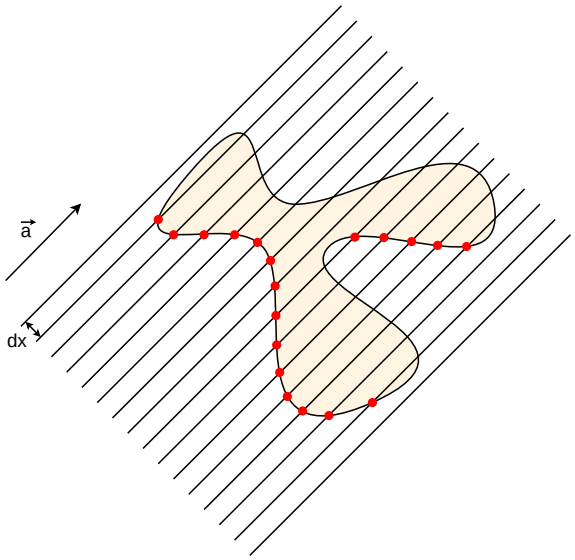
Rzutem $D(\vec{a})$ figury w kierunku wektora rzutowania $\vec{a}$ nazywamy największą odległość pomiędzy wszystkimi prostymi, równoległymi do wektora $\vec{a}$, mającymi część wspólną z analizowaną figurą [2]. W przypadku figury wklęsłej z rysunku 3.8, w rejonie wklęsłości, proste przecinające figurę mają więcej niż 2 punkty wspólne z brzegiem. W takiej sytuacji wyznaczyć można 2 rzuty cząstkowe $D_1(\vec{a})$, $D_2(\vec{a})$, których suma tworzy rzut rozwinięty [2]:

$$D(\vec{a}) = D_1(\vec{a}) + D_2(\vec{a}) \tag{3.10}$$



Rysunek 3.8: Rzuty figury dla kierunku rzutowania 45°.

Długość rzutu jest łatwa do wyznaczenia na obrazie rastrowym. Dla danej figury należy zliczyć wszystkie punkty, których otoczenie odpowiada „wchodzeniu" siecznej do środka figury (rys. 3.9). Wynik należy pomnożyć przez odległość między siecznymi - dx [2]. Dla obrazu rastrowego wielkość dx jest odległością między punktami siatki.



Rysunek 3.9: Wyznaczenie rzutu figury na obrazie rastrowym

Do wyszukiwania wymaganych punktów, przy wyznaczaniu rzutów w kierunkach 0°, 45°, 90°, 135° można użyć elementów strukturalnych z rysunku 3.10 [2].

$$\begin{bmatrix} X & X & X \\ X & 0 & 1 \\ X & X & X \end{bmatrix} \quad \begin{bmatrix} X & X & 1 \\ X & 0 & X \\ X & X & X \end{bmatrix} \quad \begin{bmatrix} X & 1 & X \\ X & 0 & X \\ X & X & X \end{bmatrix} \quad \begin{bmatrix} 1 & X & X \\ X & 0 & X \\ X & X & X \end{bmatrix}$$

a) b) c) d)

Rysunek 3.10: Elementy strukturalne do wyznaczania rzutów dla kierunków rzutowania a) 0°, b) 45°, c) 90°, d) 135°.

3.3.4 Współczynniki kształtu

Definicja współczynnika kształtu oraz własności jakie powinien on posiadać, zostały podane na początku rozdziału 3.3.3 - Pomiary obiektów. Znanych jest wiele różnych współczynników kształtu [2]. Na łamach niniejszej pracy przedstawione zostaną 2 współczynniki, które zostały wykorzystane w zrealizowanym projekcie. Zo-

stały one wybrane w wyniku przeprowadzonych prób, podczas których wykazały się najlepszą efektywnością.

Pierwszy współczynnik [2] wyliczany jest na podstawie obwodu i pola powierzchni badanego obiektu. Służy do ogólnego charakteryzowania kształtu. Jest szybki w obliczaniu, co jest jego niewątpliwą zaletą.

$$W_1 = \frac{L^2}{4\pi \cdot S} \tag{3.11}$$

gdzie:
L - obwód obiektu
S - pole powierzchni obiektu

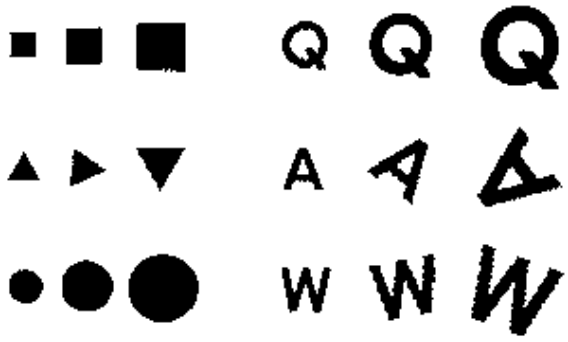
Współczynnik Blaira-Blissa [2] oblicza się nieco dłużej, za to dokładność opisu uzyskana za jego pomocą jest większa.

$$W_2 = \frac{S}{\sqrt{2\pi \cdot \sum_i {r_i}^2}} \tag{3.12}$$

gdzie:
S - pole powierzchni obiektu
r_i - odległość piksela obiektu od środka ciężkości obiektu
i - numer piksela obiektu

Sumowanie przebiega po wszystkich pikselach składających się na dany obiekt.

Na rysunku 3.11 pokazano przykładowe obiekty, dla których wyliczono współczynniki W_1 i W_2. Wyniki zapisano w tabeli 3.1. Z uwagi na fakt, że obrazy testowe są w dosyć niskiej rozdzielczości, współczynniki kształtu przyjmują nieco inne wartości dla różnych wariantów tego samego obiektu. Jest to zjawisko normalne przy niskich rozdzielczościach, gdyż błąd dyskretyzacji obrazu rośnie wraz ze zmniejszaniem jego rozdzielczości przestrzennej. Zniekształcenia są widoczne szczególnie na brzegach obiektów co powoduje obniżenie dokładności wyznaczanych obwodów. Ostatecznie przekłada się to na błąd współczynnika W_1, dla którego obwód jest jednym z dwóch kluczowych parametrów.

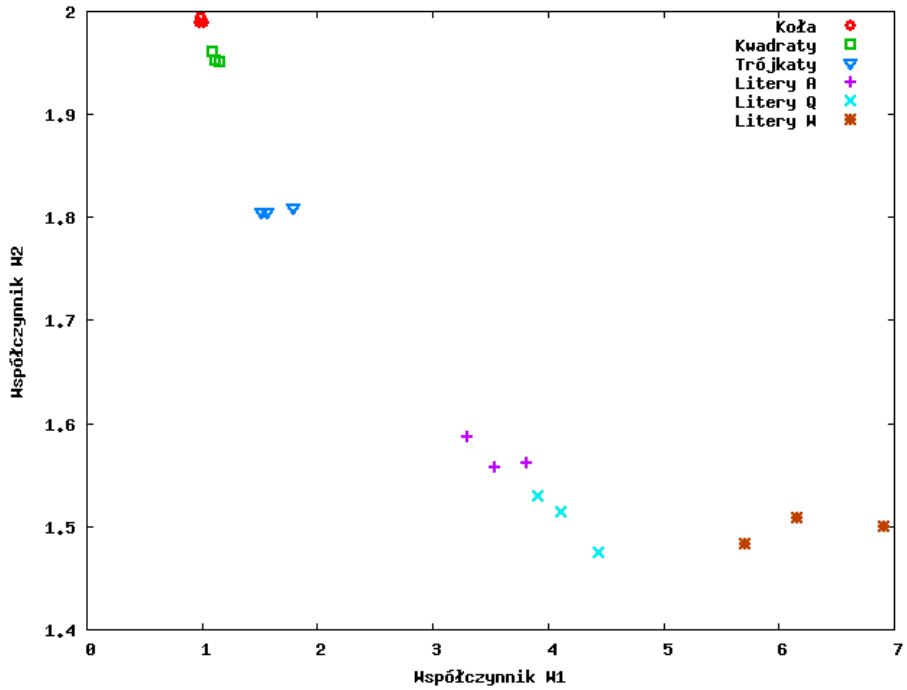


Rysunek 3.11: Zestawienie przykładowych obiektów zarejestrowanych przez kamerę robota. Każda z 6-ciu pokazanych klas (koła, kwadraty, trójkąty, litery A, Q, W) reprezentowana jest przez 3 obiekty o różnej skali, co w rzeczywistości odpowiada różnym odległościom obiektów od kamery robota. Dodatkowo niektóre obiekty zostały obrócone o losowy kąt. Prezentowany obraz ma w oryginale rozmiary 320x200 pikseli.

Tabela 3.1: Wartości współczynników W_1, W_2 wyznaczone dla obiektów z rysunku 3.11

Badane obiekty	**Współczynniki** W_1	**Współczynniki** W_2
Koła	0.974; 1.019; 0.990	1.988; 1.988; 1.995
Kwadraty	1.093; 1.110; 1.150	1.961; 1.952; 1.951
Trójkąty	1.516; 1.562; 1.789	1.804; 1.804; 1.809
Litery A	3.295; 3.536; 3.813	1.587; 1.558; 1.562
Litery Q	3.918; 4.111; 4.437	1.530; 1.513; 1.475
Litery W	5.711; 6.162; 6.914	1.483; 1.508; 1.500

Pomimo znaczących fluktuacji współczynnika W_1, korzystne jest jego wykorzystanie w parze, ze znacznie bardziej stabilnym współczynnikiem W_2, co pokazano na wykresie z rysunku 3.12. Obiekty z rysunku 3.11 zostały zlokalizowane w dwuwymiarowej przestrzeni cech, na podstawie wartości współczynników kształtu W_1, W_2. Można zauważyć, że obiekty tego samego typu znajdują się blisko siebie, tworząc grupy. Jest to zjawisko pożądane. Stopień podobieństwa dwóch dowolnych obiektów określić można bowiem obliczając odległość między nimi. Jest to popularna metoda [3] pozwalająca na klasyfikację obiektów i została przedstawiona dokładniej w rozdziale 3.4.



Rysunek 3.12: Reprezentacja obiektów z rysunku 3.11 w postaci punktów w dwuwymiarowej przestrzeni cech (W_1, W_2).

Warto również zauważyć, że współczynnik W_1, pomimo swych wad, różnicuje obiekty będące literami lepiej niż W_2, co szczególnie widać na przykładzie liter „Q" i „W". Niepokojące jest jedynie bliskie sąsiedztwo liter „A" oraz „Q", mogące prowadzić do błędnej klasyfikacji w tym obszarze. W takim wypadku celowym byłoby wprowadzenie odpowiedniego, trzeciego współczynnika oraz przejście do przestrzeni trójwymiarowej.

3.4 Rozpoznawanie obrazu

Zadanie rozpoznania obrazu polega na przyporządkowaniu rozmaitego typu obiektów do pewnych klas, przy braku znanych reguł przynależności. Jedyna możliwa do wykorzystania informacja zawarta jest w ciągu uczącym, złożonym z obiektów wzorcowych, dla których znana jest prawidłowa klasyfikacja [3].

Problem klasyfikacji obrazu daje się łatwo przedstawić matematycznie, gdyż od momentu zakończenia procesu analizy, charakterystyczne cechy obrazu opisane są przez wartości liczbowe. Dzięki temu, do procesu rozpoznawania obrazu daje się wykorzystać aparat matematyczny, co pozwala na głębokie teoretyczne rozważania

i produkcję nowych, czasem bardzo złożonych metod. Jest to rozwinięta dziedzina nauki, powstało wiele prac opisujących różnorodne metody rozpoznawania [3].

Na potrzeby tej pracy, przedstawiona zostanie najpopularniejsza metoda rozpoznawania, dająca dobre wyniki w większości typowych zastosowań. Jest to metoda minimalnoodległościowa [3], polegająca na przedstawieniu obiektów, jako punktów w n-wymiarowej przestrzeni cech – gdzie n jest liczbą różnych współczynników, opisujących obiekt. Dzięki takiej interpretacji, możliwe jest obliczanie odległości między poszczególnymi punktami (obiektami). Klasyfikacja obiektu polega w tym wypadku na przypisaniu badanego obiektu do takiej klasy, jaką reprezentuje najbliższy obiekt wzorcowy. Jako metryki można użyć odległości euklidesowej.

W przypadku gdy każdy obiekt opisany jest przez 3-elementowy wektor cech, wzór na odległość jest postaci:

$$d = \sqrt{[W_1(F_i) - W_1(H_j)]^2 + [W_2(F_i) - W_2(H_j)]^2 + [W_3(F_i) - W_3(H_j)]^2} \quad (3.13)$$

gdzie:

d - odległość między dwoma obiektami

W_1, W_2, W_3 - współczynniki kształtu

F - badany obiekt

H - obiekt wzorcowy

i - numer badanego obiektu

j - numer obiektu wzorcowego

Rozdział 4

Oprogramowanie robota - firmware

Oprogramowanie zostało napisane w języku C. Skompilowany kod został zapisany w pamięci flash 32-bitowego mikrokontrolera, zainstalowanego na płycie głównej robota. Oprogramowanie to odpowiedzialne jest za wszelkie operacje wykonywane przez procesor, w szczególności steruje pracą wszystkich podzespołów robota. Najważniejsze funkcje oprogramowania mikrokontrolera to:

- sterowanie pracą kamery i odbiór danych
- przetwarzanie, analiza i rozpoznawanie obrazu
- sterowanie pracą podzespołów wykonawczych robota
- obsługa komunikacji bezprzewodowej

Kod źródłowy powyższego oprogramowania został dołączony w załączniku A.

4.1 Odbiór danych z kamery

Zdolność do samodzielnej akwizycji obrazu za pośrednictwem dołączonej kamery jest realizacją jednego z podstawowych założeń projektu. Podstawowe parametry techniczne kamery oraz sposoby transmisji danych zostały omówione w rozdziale 2.1.2. Tutaj informacje te zostaną nieco rozszerzone.

Tryb pracy kamery

Z przyczyn omówionych w rozdziale 2.1.2 kamera pracuje na ustawieniach domyślnych. W trybie tym transmitowany jest obraz o rozdzielczości 640x480 pikseli w

formacie YCbCr422. Ponadto działają zaimplementowane w logice kamery podstawowe funkcje kontroli i poprawy jakości obrazu: automatyczna kontrola ekspozycji, automatyczny bilans bieli oraz kompensacja światła tylnego (backlight compensation). Ustawienia te zapewniają dobrą jakość obrazu.

Procedura startowa

Po włączeniu zasilania kamera musi zostać odpowiednio zainicjalizowana. Procedura startowa polega na wprowadzeniu kamery w stan resetu i odczekaniu 100 cykli zegara MCLK. Następnie kamera jest wyprowadzana ze stanu reset, a zegar jest wyłączany po wygenerowaniu pojedynczego impulsu.

Etapy odbioru danych

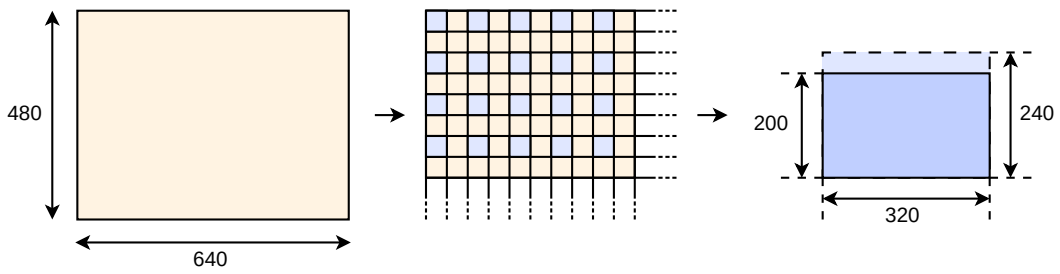
Jeśli w programie zostanie zgłoszona gotowość na odbiór nowej klatki obrazu, ponownie włączany jest zegar MCLK. Rozpoczyna się transmisja ramki obrazu – nie jest ona jednak zapisywana w pamięci, lecz pomijana (drop frame). Pierwsza klatka obrazu jest zazwyczaj prześwietlona, gdyż kamera była dłuższy czas nieaktywna, a światło cały czas padało na matrycę. Podczas transmisji pierwszej klatki kamera ma również szanse ustawić prawidłowe parametry ekspozycji. Koniec transmisji klatki stwierdzany jest na podstawie obserwacji stanu linii VSYNC.

Następnie rozpoczyna się transmisja kolejnej klatki – ta jest już zapisywana w pamięci SRAM mikrokontrolera. Śledzenie współrzędnych aktualnie wysyłanego piksela odbywa się na bazie próbkowania sygnałów HSYNC/VSYNC. Dane pikseli odbierane są z równoległego 8-bitowego portu kamery.

Gdy transmisja dobiegnie końca, zegar MCLK jest wyłączany. Nowa kaltka obrazu znajduje się w pamięci SRAM.

Format odbieranych danych

Pomimo iż kamera wysyła obraz o rozdzielczości 640x480 pikseli, niemożliwe jest jego zapisanie w oryginalnej rozdzielczości w pamięci SRAM mikrokontrolera. Zapis kolorowego obrazu o tych rozmiarach wymagałby 600KB pamięci, w odcieniach szarości 300KB. Do dyspozycji jest jednak tylko 64KB. W związku z tym została wybrana rozsądna rozdzielczość maksymalna - 320x200 pikseli. Po pominięciu informacji o kolorach obraz taki wymaga 62.5KB pamięci. Redukcja rozdzielczości obrazu odbywa się według schematu z rysunku 4.1. Wykonywana jest w trakcie odbierania obrazu z kamery, poprzez pomijanie niepotrzebnych danych.



Rysunek 4.1: Schemat poglądowy pokazujący sposób na redukcję rozdzielczości dla obrazu wynikowego w odcieniach szarości. Po otrzymaniu obrazu 320x240 pikseli, górne 40 linii obrazu jest obcinane.

Zaimplementowano również możliwość zapisu obrazu 160x100 pikseli w wersji kolorowej lub w odcieniach szarości. Dla obrazu kolorowego proces skalowania jest nieco bardziej skomplikowany, ze względu na specyficzne rozłożenie informacji o kolorach, występujące w trybie YCbCr422.

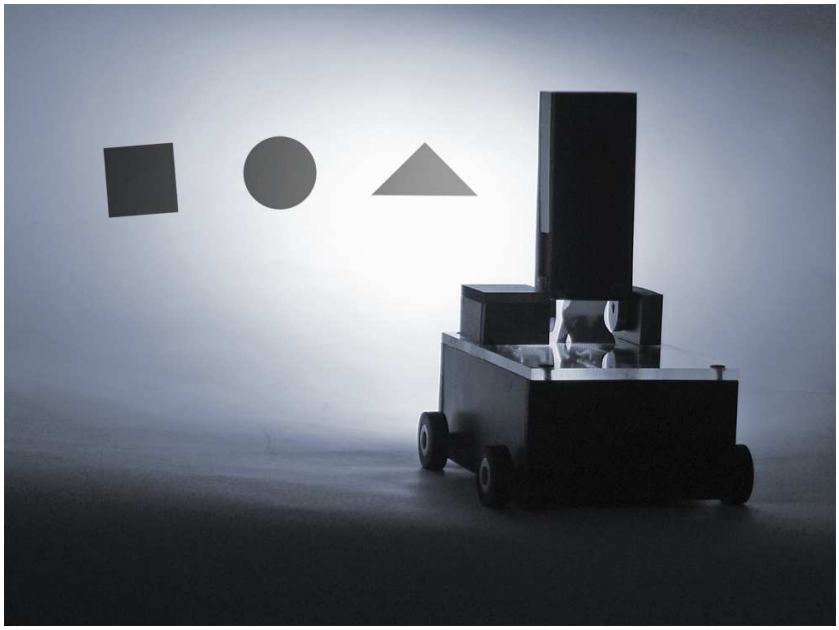
4.2 Przetwarzanie, analiza i rozpoznawanie obrazu

Implementacja trybu autonomicznego była ostatnim etapem prac nad projektem. Początkowo planowano rozpoznawanie i śledzenie obiektów ruchomych, jednak okazało się, że moc obliczeniowa mikrokontrolera nie pozwoli na obróbkę danych z wystarczającą szybkością. Zdecydowano się więc na rozpoznawanie obiektów statycznych.

Robot jest w stanie rozpoznawać ciemne obiekty dowolnych kształtów, znajdujące się na jasnym, kontrastowym tle (rys. 4.2). Do testów wykorzystane zostały różnorodne figury i kształty w kolorze czarnym, wydrukowane na białym papierze A4. Analiza i rozpoznawanie obrazu wykonywane są w celu zlokalizowania obiektu wzorcowego, w aktualnie obserwowanej przestrzeni. Kształt wzorca zapamiętywany jest wcześniej, poprzez ustawienie symbolu w zasięgu widzenia kamery i wskazanie jego lokalizacji na obrazie.

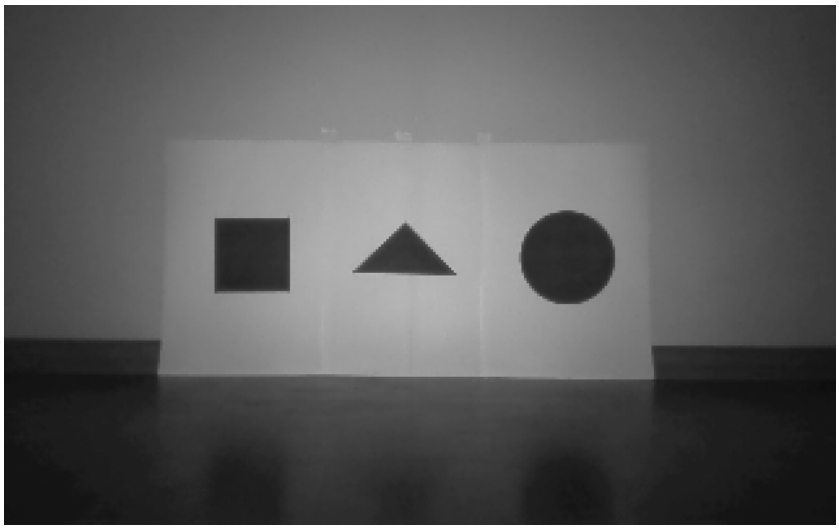
Po uruchomieniu trybu autonomicznego, robot zaczyna analizować obserwowaną przestrzeń w celu znalezienia obiektu wzorcowego wśród aktualnie widzianych obiektów. Po rozpoznaniu, wyznaczane jest położenie szukanego obiektu, a robot zaczyna przemieszczać się w jego kierunku. Podczas jazdy robot przeprowadza kolejne analizy w celu aktualizacji współrzędnych obiektu i przeprowadzenia ewentualnych korekt kursu. Efektem działania trybu autonomicznego, jest więc samodzielne przemiesz-

czenie się robota w pobliże rozpoznanego przez siebie obiektu.



Rysunek 4.2: Robot podczas pracy w autonomicznym trybie rozpoznawania obrazu. Zdjęcie pokazuje pracę w nocy, z wykorzystaniem lampy LED robota.

Analizie poddawany jest obraz w skali szarości, o rozdzielczości 320x200 pikseli (rys 4.3). Wszelkie operacje na obrazie wykonywane są zgodnie z ideą pokazaną w rozdziale 3.



Rysunek 4.3: Przykładowy obraz z kamery robota (8bit, 320x200).

Przetwarzanie obrazu

Przetwarzanie wstępne wykonywane jest automatycznie, przez moduł kamery (bilans bieli, kompensacja światła). Podczas odbierania danych z kamery, obraz jest pozbawiany informacji o kolorach, a jego rozdzielczość redukowana jest dwukrotnie. Operacje te zostały opisane w rozdziale 4.1.

Kolejnym krokiem jest przeprowadzenie binaryzacji obrazu (rys. 4.4). Jest ona wykonywana według wzoru 3.1 (rozdział 3.2). Próg binaryzacji a obliczany jest dynamicznie dla każdej klatki obrazu – jest to średnia jasność dla całego zdjęcia. Rozwiązanie takie pozwala na automatyczne dostosowanie progu do jasności i kontrastowości obrazu, bez konieczności dodatkowej normalizacji tych parametrów. Co ważne, jest to również rozwiązanie szybkie i nieskomplikowane – co przy ograniczonej mocy obliczeniowej mikrokontrolera ma niebagatelne znaczenie.

Rysunek 4.4: Obraz z rysunku 4.3 po przeprowadzonej segmentacji. Piksele „zapalone" zaznaczono kolorem czarnym.

Zadaniem robota jest rozpoznawanie czarnych obiektów na białym tle, w związku z tym zastosowanie średniej jasności zdjęcia jako progu binaryzacji było intuicyjne i dało zadowalające rezultaty, nawet w trudnych warunkach oświetleniowych.

Analiza obrazu

Po wykonaniu binaryzacji z odpowiednio dobranym progiem, na utworzonym 1-bitowym obrazie pozostają wyraźnie zarysowane kształty interesujących nas obiektów. Może się zdarzyć, że na obrazie pokażą się również niechciane elementy, w

postaci małych grup zapalonych pikseli. Obiekty te mogłyby zafałszować wyniki dalszych obliczeń - są więc usuwane z obrazu w trakcie jego indeksacji. Jeśli po zakończeniu indeksacji danego obiektu okaże się, że składa się on z mniej niż 100 punktów, obiekt jest usuwany. Indeksacja przebiega według algorytmu Smitha opisanego w rozdziale 3.3.2. Podczas indeksacji dla każdego z obiektów wyznaczane są również prostokąty obcinania, których wymiary równe są odpowiednim średnicom Fereta obiektów.

W tym momencie wszystkie obiekty wydobyte z pierwotnego zdjęcia, traktowane są jak odrębne obrazy. Każdy z obrazów zawiera informację o kształcie obiektu który reprezentuje. Informacja ta zostaje teraz przeliczona na pojedyncze wartości liczbowe.

Najpierw dla każdego obiektu wyznaczane są parametry bezpośrednie, szczegółowo opisane w rozdziale 3.3.3. Są to: momenty bezwładności pierwszego rzędu, pole, długości rzutów N_0, N_{45}, N_{90}, N_{135} oraz obwód. Obwód wyznaczany jest z formuły Croftona (wzór 3.9), a do wyliczenia długości rzutów wykorzystywane są elementy strukturalne z rysunku 3.10. Następnie wykorzystując wyznaczone parametry, obliczane są współczynniki kształtu W_1, W_2, omówione w rozdziale 3.3.4. Obliczenia wykonywane są według wzorów 3.11 i 3.12.

Rozpoznawanie obrazu

Rozpoznawanie obrazu oparte jest o metodę minimalnoodległościową, opisaną w rozdziale 3.4. Istnieje jedna klasa wzorcowa, do której należy zapamiętany odpowiednią komendą obiekt wzorcowy. Wektory cech wszystkich analizowanych obiektów, porównywane są z wektorem cech obiektu wzorcowego, poprzez obliczanie odległości według wzoru 3.13. Element którego odległość od zapamiętanego wzorca jest najmniejsza, uznawany jest za obiekt należący do klasy wzorcowej.

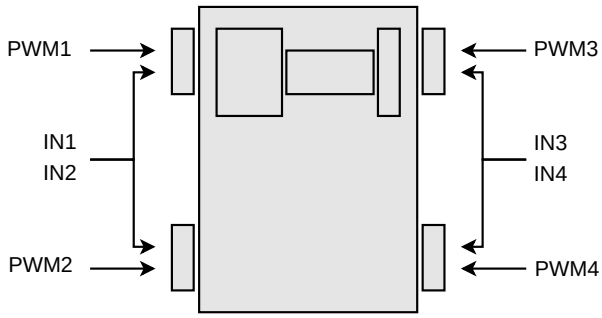
Cały algorytm akwizycji, przetwarzania, analizy i rozpoznawania wykonywany jest cyklicznie. Efektem zakończenia pojedynczego przebiegu, jest podjęcie przez robota decyzji który z aktualnie obserwowanych obiektów należy do klasy obiektów wzorcowych. Zaraz potem robot zaczyna kierować się w stronę wybranego przez siebie obiektu (jego środka ciężkości), a cały algorytm zaczyna się od początku. Czas potrzebny na pojedynczy przebieg algorytmu to ok 3s. Test systemu przeprowadzony został w rozdziale 6.3.

4.3 Sterowanie pracą podzespołów wykonawczych robota

Firmware robota steruje pracą głównych silników napędowych, serwomechanizmu odpowiedzialnego za ruchy kamery oraz lampy led zapewniającej dodatkowe oświetlenie dla kamery. Ponadto na wejście ADC mikrokontrolera podawane jest aktualne napięcie akumulatorów (podzielone przez 3 na dzielniku rezystorowym) co umożliwia jego pomiar i oszacowanie pozostałej ilości energii.

Sterowanie kierunkiem i prędkością obrotu silników

Do sterowania pracą silników potrzebna jest elektronika wykonawcza omówiona w rozdziale 2.1.4. Pracę elektroniki kontrolują następujące sygnały cyfrowe: IN1, IN2, IN3, IN4, PWM1, PWM2, PWM3, PWM4. Przydział sygnałów do odpowiadających im silników/driverów pokazany jest na rysunku 4.5.

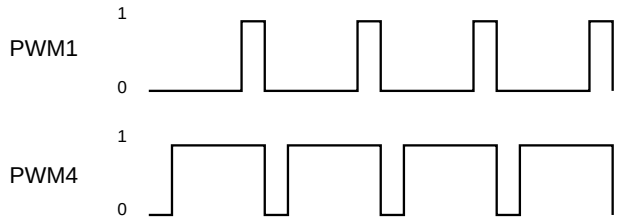


Rysunek 4.5: Sygnały sterujące pracą poszczególnych silników robota.

Pary sygnałów (IN1, IN2) oraz (IN3, IN4) odpowiadające odpowiednio lewym i prawym silnikom, tworzą 2-bitowe wejścia sterujące kierunkiem ich obrotów. Zarówno dla lewych jak i dla prawych silników możliwe są następujące kombinacje:

- (0,0) lub (1,1) - silniki zatrzymane
- (1,0) - obroty w przód
- (0,1) - obroty w tył

Ponadto każdy silnik posiada swój sygnał PWM, sterujący szybkością jego obrotów. Sygnał PWM (Pulse Width Modulation) jest przebiegiem prostokątnym o zmiennym współczynniku wypełnienia. Poprzez zmianę wypełnienia możliwa jest regulacja prędkości obrotowej silnika. Przykładowe przebiegi PWM pokazano na rysunku 4.6.

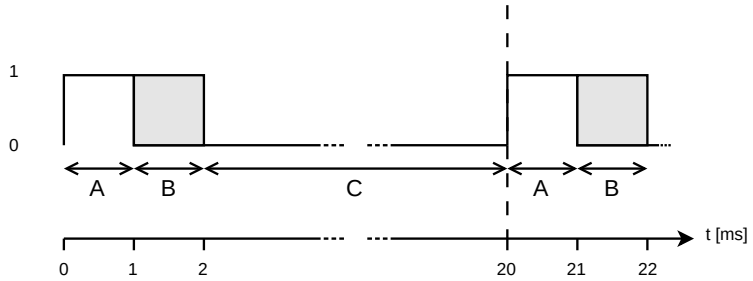


Rysunek 4.6: Przykładowe przebiegi 2 sygnałów PWM. Silnik sterowany sygnałem PWM1 będzie obracał się bardzo powoli, PWM4 osiągnie prędkość obrotów zbliżoną do maksymalnej - przy braku obciążenia i odpowiedniej konfiguracji sygnałów IN1-4.

Sygnały PWM generowane są sprzętowo, przez zintegrowany z procesorem czterokanałowy kontroler. Stworzone oprogramowanie steruje generatorem procesora w oparciu o dane pozyskane z analizy obrazu lub komendy wydane za pośrednictwem linku bezprzewodowego. Szczegółowe informacje znaleźć można w kodzie źródłowym programu oraz w dokumentacji mikrokontrolera SAM7 [5, 14].

Sterowanie wychyleniem serwomechanizmu

Standardowy sposób sterowania serwomechanizmem pokazany jest na rysunku 4.7. Przez pierwszą milisekundę (A) sygnał sterujący powinien być w stanie wysokim. W drugiej milisekundzie (B) powinna nastąpić zmiana stanu z wysokiego na niski. Przez następne 18ms (C) linia sterująca powinna być w stanie niskim. Sekwencja powinna być powtarzana co 20ms.



Rysunek 4.7: Sygnał sterujący wychyleniem osi serwa analogowego. Wychylenie jest proporcjonalne do czasu trwania impulsu.

Sumarycznie więc minimalna długość pojedynczego impulsu to 1ms, a maksymalna 2ms. Od długości impulsu uzależniony jest kąt wychylenia osi serwa. Dla impulsu 1ms wychylenie serwa wynosi 0°, 1.5ms daje wychylenie 90°, a 2ms 180°. Dla tanich serw ich czasy mogą nieco odbiegać od standardowych.

Sterowanie serwem zostało zrealizowane w oparciu o timer PIT mikrokontrolera (Periodic Interval Timer). Sprzętowy licznik odmierza precyzyjnie czas trwania każdego impulsu i generuje żądanie przerwania w momencie, kiedy zachodzi konieczność zmiany stanu linii sterującej. Podejście takie jest optymalne i w minimalnym stopniu angażuje rdzeń procesora. Procedura obsługi wywoływana jest tylko 100 razy w ciągu sekundy i polega jedynie na zmianie stanu linii sterującej. Taki sposób sterowania można rozszerzyć na dowolną ilość serwomechanizmów.

Istnieje również możliwość wyłączenia serwomechanizmu poprzez odcięcie zasilania. W tym celu wykorzystany został drugi niezależny sygnał, sterujący tranzystorem kluczującym.

Sterowanie oświetleniem

Włączanie lub wyłączanie lampy realizowane jest za pomocą pojedynczej linii sterującej, podłączonej do wejścia Enable drivera LED.

Pomiar napięcia na akumulatorach

Przetwornik ADC (Analog-to-Digital Converter) przyporządkowuje wartości napięcia doprowadzonego na jego wejście liczbę całkowitą z zakresu 0-1023 (dokładność 10-bit). 0 odpowiada zerowemu potencjałowi na wejściu, a 1023 odpowiada wartości napięcia referencyjnego, ustawionego dla przetwornika - w tym wypadku jest to 3.3V. Jest to jednocześnie maksymalna wartość napięcia, możliwa do zmierzenia. Jako że wartość napięcia na akumulatorach może wynosić nawet 8.5V, zastosowany został dzielnik rezystorowy (1/3) - trzy szeregowe rezystory 2K o tolerancji 1% (schemat ideowy – rysunek 2.5).

Po dokonaniu pomiaru możliwe jest obliczenie rzeczywistego napięcia, korzystając ze wzoru:

$$U = (V_{ref}/1023) \cdot V_{adc} \cdot 3 \tag{4.1}$$

gdzie:

U - napięcie rzeczywiste na akumulatorach

V_{ref} - napięcie referencyjne przetwornika (=3.3V)

V_{adc} - wartość otrzymana w procesie konwersji

Znając minimalne, maksymalne i aktualne napięcie na akumulatorach, możliwe jest oszacowanie ile procent energii zostało w bateriach. Oszacowanie to jest dość

dokładne, gdyż napięcie akumulatora li-ion jest silnie skorelowane z poziomem jego naładowania - w przybliżeniu zależność ta jest liniowa.

$$E = \frac{U - V_{min}}{V_{max} - V_{min}} \cdot 100\% \tag{4.2}$$

gdzie dodatkowo:

E - procent energii pozostałej w akumulatorach, liczony względem energii maksymalnej (wartość przybliżona)

V_{min} - minimalne, bezpieczne dla akumulatora napięcie (=6V)

V_{max} - napięcie maksymalne przy pełnym naładowaniu (=8.5V)

4.4 Obsługa komunikacji bezprzewodowej

Komunikacja została zrealizowana w oparciu o sprzętowy port UART mikrokontrolera (Uniwersal Asynchronous Receiver Transmitter). Poprzez port dane transmitowane są z i do modułu Bluetooth, który odpowiedzialny jest za link bezprzewodowy. Transmisja odbywa się z prędkością 460.8 kbit/s.

Kanał nadawczy

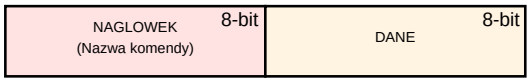
W związku z faktem, iż wysyłane są znaczne ilości danych (między innymi obrazy z kamery) zdecydowano się na realizację kanału nadawczego w oparciu o sprzętowy kontroler PDC mikrokontrolera (Peripheral Direct Memory Access Controller - DMA). W przypadku wysyłania dużego bloku danych, rozwiązanie takie pozwala wpisać do rejestru urządzenia DMA adres pierwszego bajtu z bloku oraz ilość bajtów do wysłania. Kontroler DMA sam zajmie się transmisją danych, nie obciążając rdzenia procesora - podczas transmisji wykonanie programu głównego może być kontynuowane.

Duże bloki danych dzielone są na pakiety zawierające 3200 bajtów i wysyłane kolejno, po potwierdzeniu przez stronę odbiorczą poprawnego odebrania poprzedniego pakietu. W razie wystąpienia błędów transmisji, uszkodzone pakiety wysyłane są ponownie. Rozwiązanie takie minimalizuje straty danych wywołane błędami transmisji, które mogą się zdarzyć poprzez zakłócenia występujące na drodze sygnału radiowego.

Kanał odbiorczy

Kanał odbiorczy zrealizowany jest w oparciu o system przerwań. Kanał DMA nie został tu użyty, gdyż odbierane są niewielkie ilości danych.

Standardowa ramka danych ma stałą długość i zawiera 2 bajty (rys. 4.8). Każda odbierana ramka traktowana jest jak nowe polecenie do wykonania. Bajt pierwszy zarezerwowany jest na nazwę komendy, interpretowaną według tablicy ASCI jako pojedyncza litera. Bajt drugi zawiera dane użyteczne.



Rysunek 4.8: Standardowa ramka odbieranych danych.

W momencie odebrania kompletnej komendy, następuje jej interpretacja i wykonanie. Wyróżnić można 2 rodzaje komend:

1. Polecenia sterujące i konfiguracyjne
2. Komendy nakazujące rozpoczęcie transmisji konkretnych danych

Komendy z grupy pierwszej służą do zdalnego sterowania robotem oraz zmian parametrów pracy. Wykonanie komendy polega zazwyczaj na przekazaniu odebranych parametrów do danego urządzenia peryferyjnego lub zmianie wartości konkretnych zmiennych systemowych. Komendy te nie wysyłają danych zwrotnych ani nie potwierdzają wykonania.

Wykonanie komend z grupy drugiej najczęściej polega na wpisaniu do odpowiedniego rejestru kontrolera DMA adresu bloku danych do wysłania. Dla niektórych komend wymagane jest wykonanie operacji dodatkowych - np. dla komend dotyczących transmisji obrazu z kamery wymagane jest uruchomienie procedury akwizycji danych.

4.5 Specyfikacja obsługiwanych komend

Rozdział ten zawiera szczegółową specyfikację wszystkich komend robota. W celu zwiększenia przejrzystości, komendy podzielono na kilka kategorii ze względu na ich zastosowanie. W tabeli 4.1 wyspecyfikowano komendy zdalnego sterowania robotem. Tabela 4.2 zawiera komendy pozwalające odbierać dane od robota. Komendy zawarte w tabeli 4.3 służą do konfiguracji trybu autonomicznego.

Tabela 4.1: Komendy zdalnego sterowania

Kom.	**Wartość**	**Funkcja**
a	2-255	Zmiana współczynnika wypełnienia na przebiegach PWM1 i PWM2 jednocześnie - silniki lewe
b	2-255	Zmiana współczynnika wypełnienia na przebiegach PWM3 i PWM4 jednocześnie - silniki prawe
l	0	Silniki lewe - stop
	1	Silniki lewe - obroty w przód
	2	Silniki lewe - obroty w tył
r	0	Silniki prawe - stop
	1	Silniki prawe - obroty w przód
	2	Silniki prawe - obroty w tył
s	0-255	Sterowanie kątem wychylenia wieży z kamerą
w	0	Wyłączenie zasilania serwa sterującego wieżą
	1	Włączenie zasilania serwa sterującego wieżą
d	0	Wyłączenie zasilania lampy LED
	1	Włączenie zasilania lampy LED

* Orientacja lewo/prawo według rysunku 4.5

Tabela 4.2: Komendy przesyłania telemetrii/obrazu

Kom.	**Wartość**	**Funkcja**
t	dowolna	Pomiar napięcia na akumulatorach i transmisja wyniku
f	0	Rozpoczęcie akwizycji obrazu 320x200, 8-bit i przesłanie pierwszego bloku
	1-19	Przesyłanie kolejnych 19 bloków
x	0	Rozpoczęcie akwizycji obrazu kolorowego 160x100 i przesłanie pierwszego bloku
	1-9	Przesyłanie kolejnych 9 bloków
p	0	Rozpoczęcie akwizycji obrazu 160x100, 8-bit i przesłanie pierwszego bloku
	1-4	Przesyłanie kolejnych 4 bloków

Tabela 4.3: Komendy trybu autonomicznego

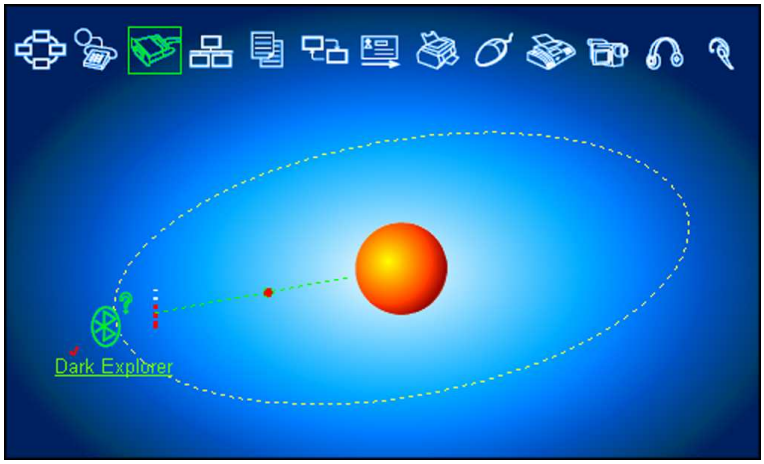
Kom.	**Wartość**	**Funkcja**
i	0-4	Przesłanie podglądu obrazu binarnego (5 bloków)
	10	Włączenie trybu autonomicznego (rozpoznawanie obrazu)
	11	Wyłączenie trybu autonomicznego
	12	Włączenie przesyłania informacji synchronizacyjnych oraz uruchomienie dodatkowego buforowania obrazu dla podglądu w trybie autonomicznym
	13	Wyłączenie przesyłania informacji synchronizacyjnych oraz buforowania
j	0-4	Przesłanie podglądu obrazu binarnego - wersja alternatywna (5 bloków)
m	0-255	Zapamiętanie wskazanego obiektu wzorcowego - współrzędna x
n	0-255	Zapamiętanie wskazanego obiektu wzorcowego - współrzędna y

Dodatkowe informacje znaleźć można w dołączonych kodach źródłowych (dod. A).

Rozdział 5

Zdalne zarządzanie pracą robota

Komunikacja odbywa się za pośrednictwem protokołu Bluetooth. Po włączeniu zasilania robota możliwe staje się jego wykrycie przez większość urządzeń obsługujących ten standard transmisji danych. Robot zgłasza się pod nazwą Dark Explorer (rys 5.1). Aby ustanowić połączenie, wymagane jest podanie hasła "1234". Po zestawieniu połączenia możliwa jest komunikacja z robotem za pomocą wirtualnego portu szeregowego COM.



Rysunek 5.1: Połączenie bezprzewodowe z robotem zestawione za pomocą jednego z popularnych programów zarządzających łącznością Bluetooth.

5.1 Program nadzorujący na komputer PC

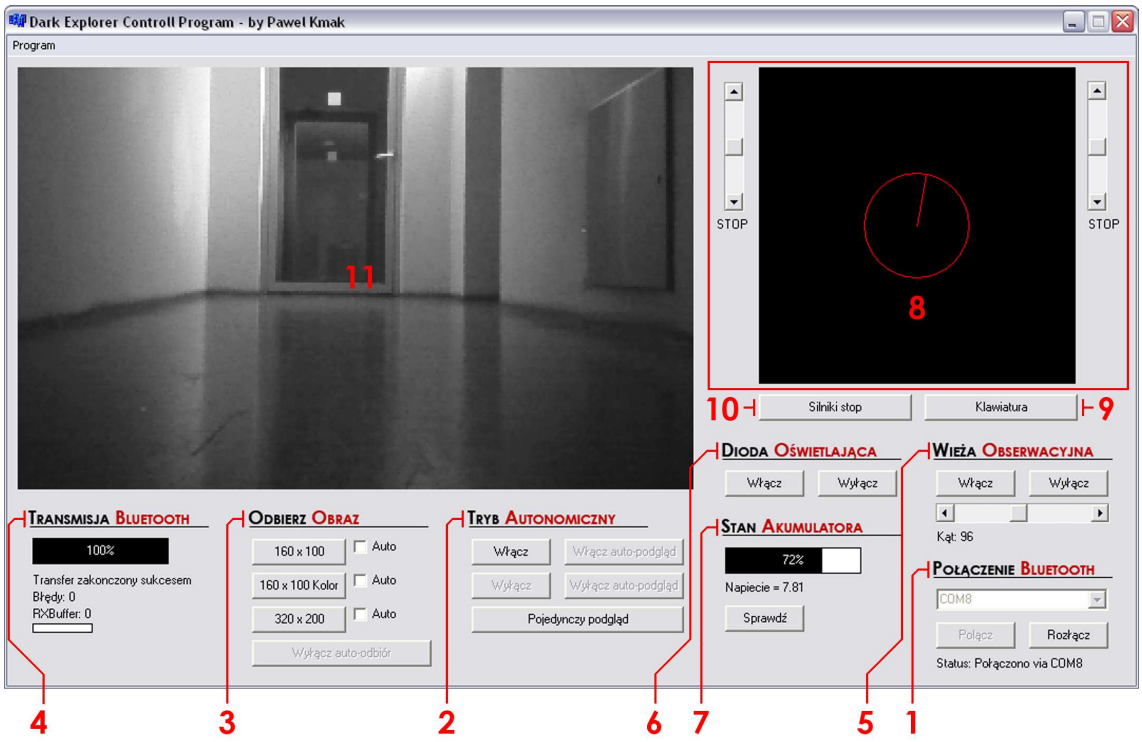
Do zarządzania pracą robota stworzony został program na komputer PC. Komunikuje się on z robotem za pośrednictwem komend opisanych w rozdziale 4.5. Przy użyciu programu nadzorującego jest więc możliwa zmiana trybów pracy, kontrola

parametrów, podgląd obrazu z kamery, jak i sterowanie pracą podzespołów robota.

Program napisany został w języku C++ i został przystosowany do pracy na komputerze z zainstalowanym systemem Windows. Fragmenty kodów źródłowych oprogramowania nadzorującego dołączone zostały w załączniku B.

5.1.1 Funkcje programu

Na rysunku 5.2 pokazano interfejs programu nadzorującego. Jego elementy zostały pogrupowane według wykonywanych zadań. Poniżej znajduje się szczegółowy opis wszystkich funkcji.



Rysunek 5.2: Interfejs programu nadzorującego.

(1) Połączenie Bluetooth

Program komunikuje się z robotem za pośrednictwem wirtualnego portu COM stworzonego w systemie przez oprogramowanie zarządzające łącznością Bluetooth. Aby umożliwić komunikację programu nadzorującego z robotem należy:

- ustanowić połączenie z robotem za pomocą programu zarządzającego systemem Bluetooth w komputerze (np. BlueSoleil)

- wybrać numer portu COM, który został przyporządkowany do robota po nawiązaniu połączenia
- kliknąć przycisk Połącz

(2) Tryb autonomiczny

Przyciski Włącz/Wyłącz pozwalają przełączać robota do trybu pracy autonomicznej w oparciu o rozpoznawanie obrazu (komendy i10 oraz i11). Możliwe jest również uruchomienie funkcji auto-podgląd pozwalającej na bieżąco odbierać obraz po binaryzacji, w oparciu o który firmware robota wykonuje analizy i rozpoznawanie (opcja dostępna dopiero po włączeniu trybu autonomicznego). Włączenie funkcji auto-podgląd wiąże się z wysłaniem komendy i12, która uruchamia dodatkowe buforowanie obrazu. Odbieranie podglądu realizowane jest poprzez cykliczne użycie komend i0 – i4.

Aby uzyskać podgląd aktualnego stanu pamięci obrazu robota (bez buforowania) można użyć funkcji „Pojedynczy podgląd". Funkcja była przydatna podczas testowania algorytmów przetwarzania i analizy. Najlepiej używać jej po wyłączeniu trybu autonomicznego, kiedy pamięć obrazu nie zmienia się. Do jej obsługi wykorzystywane są komendy j0 – j4.

Po włączeniu trybu autonomicznego należy jeszcze wskazać wzorzec, który ma być rozpoznawany. Aby wybrać obiekt wzorcowy, należy:

- ustawić robota przed obiektem w odpowiedniej odległości
- uruchomić tryb autonomiczny wraz z auto-podglądem i poczekać na pierwszy obraz
- wskazać na obrazie obiekt wzorcowy (pojedyncze kliknięcie lewym przyciskiem myszy na obszarze 11 – rys. 5.2)

Aby obiekt wzorcowy został poprawnie zapamiętany, robot oraz obiekt muszą pozostać w bezruchu przez ok 3 sekundy, od momentu wskazania. Lokalizacja obiektu, którego kształt ma być zapamiętany, przesyłana jest komendami „m" oraz „n".

Możliwe jest trzykrotne zapamiętanie tego samego obiektu wzorcowego w celu poprawy jakości rozpoznawania (z różnych odległości lub pod różnym kątem). Klasa wzorcowa obiektów będzie wtedy dokładniej określona. Zazwyczaj wystarcza jednak pojedyncze wskazanie.

(3) Odbierz obraz

Funkcje z tej grupy pozwalają w dowolnym momencie podejrzeć obraz rejestrowany przez kamerę robota. Do wyboru są 3 różne opcje podglądu:

1. Podgląd w odcieniach szarości, o rozdzielczości 160x100 pikseli (16kB)
2. Podgląd kolorowy w rozdzielczości 160x100 pikseli (32kB)
3. Podgląd w odcieniach szarości, o rozdzielczości 320x200 pikseli (64kB)

Wykorzystywane są do tego celu komendy to „p", „x", „f". Dla każdej z opcji można uruchomić tryb „Auto", który cyklicznie odbiera kolejne obrazy z maksymalną możliwą częstotliwością.

(4) Transmisja Bluetooth

Zgrupowane są tu wskaźniki pozwalające śledzić odbiór danych kanałem bezprzewodowym. Wyświetlane są następujące dane:

- całkowity postęp odbioru danych
- postęp odbioru danego bloku danych (RXBuffer)
- numer aktualnie odbieranego bloku
- ilość błędnych bloków

Funkcja odbioru danych została uzupełniona o prosty algorytm kontroli błędów. Sprawdzana jest poprawność danych w każdym bloku. W przypadku wystąpienia błędów, wysyłana jest prośba o retransmisję danego bloku.

(5) Wieża obserwacyjna

Przyciski Włącz/Wyłącz pozwalają sterować zasilaniem serwomachanizmu sterującego wieżą (komendy w1 oraz w0). Suwak służy do ustawienia kąta wychylenia wieży z przedziału 0°– 180° (komenda „s"). Aby ustawić pożądany kąt, wcześniej należy włączyć zasilanie serwa.

(6) Dioda oświetlająca

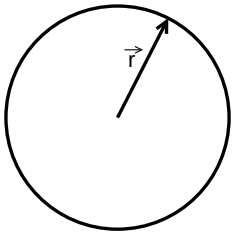
Przyciski Włącz/Wyłącz sterują zasilaniem lampy LED. Komendy sterujące zasilaniem to d1 oraz d0.

(7) Stan akumulatora

Przycisk „Sprawdź" powoduje przesłanie komendy t0, na którą robot odpowiada dwoma bajtami danych, zawierającymi aktualną wartość napięcia na akumulatorach. Po odpowiednich przeliczeniach według wzorów 4.1 i 4.2 z rozdziału 4.3, wyświetlane jest aktualne napięcie w voltach oraz szacowana pozostała ilość energii w akumulatorach.

(8) Wirtualny dżojstik zdalnego kierowania

Do zdalnego sterowania silnikami robota służy wirtualny dżojstik pokazany na rysunku 5.3. Długość wektora $\vec{r}$ jest związana z prędkością jazdy robota. Dla długości równej promieniowi okręgu prędkość jest maksymalna.

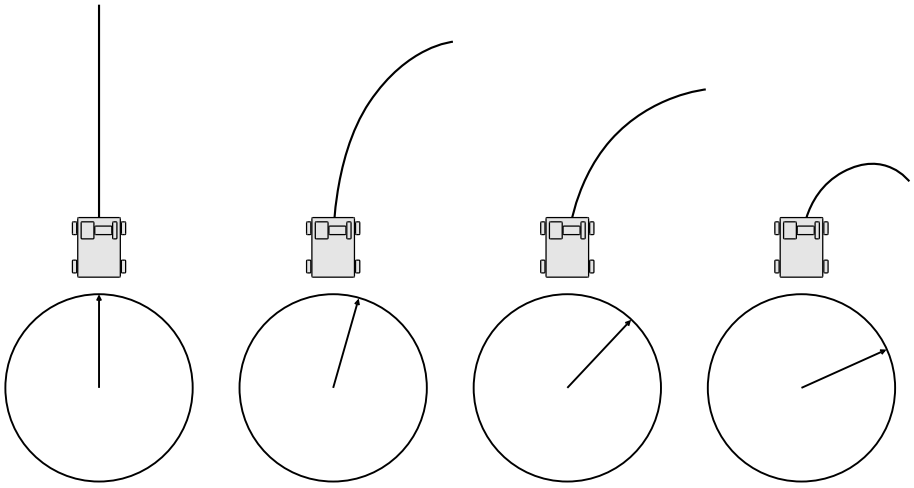


Rysunek 5.3: Wirtualny dżojstik.

Kierunek wektora odpowiada za zmianę promienia skrętu robota. Wektor skierowany w górę, powoduje jazdę na wprost, do przodu. Skierowanie wektora w dół spowoduje, że robot pojedzie prosto do tyłu. Skierowanie wektora w prawo lub w lewo, da efekt zakręcania w miejscu, odpowiednio w prawo lub lewo. Wszelkie kierunki pomiędzy wymienionymi tutaj czteroma skrajnymi, odpowiadają płynnemu przejściu pomiędzy opisanymi zachowaniami (rys. 5.4).

Aby sterować robotem za pomocą wirtualnego dżojstika, należy operować myszą w rejonie czarnego obszaru (rys. 5.2), trzymając wciśnięty jednocześnie lewy przycisk. Po zwolnieniu przycisku, automatycznie pobierany jest podgląd obrazu z kamery, a robot zatrzymuje się - ułatwia to nawigację bez kontaktu wzrokowego.

Dodatkowo możliwe jest też sterowanie silnikami za pomocą dwóch suwaków, znajdujących się nieopodal dżojstika. Suwak z lewej strony odpowiada za kierunek i prędkość obrotów kół lewych robota, z prawej – prawych. Do sterowania pracą silników służą komendy „a", „b", „l", „r".



Rysunek 5.4: Tor ruchu robota dla różnych kierunków wektora $\vec{r}$.

(9) Przycisk „Klawiatura"

Po wciśnięciu przycisku, możliwe jest sterowanie robotem za pomocą klawiszy „w", „a", „s", „d", powodujących odpowiednio: jazdę w przód, obrót w lewo, jazdę w tył, obrót w prawo. Naciśnięcie 2 przycisków jednocześnie, daje efekt superpozycji wciśnięcia każdego z osobna.

(10) Przycisk „Silniki stop"

Przycisk powoduje natychmiastowe zatrzymanie wszystkich silników napędowych robota. Może być przydatny w sytuacji awaryjnej.

(11) Okno obrazu

W tym obszarze wyświetlane są wszystkie rodzaje podglądów obrazu z kamery. Tutaj również w trybie autonomicznym, poprzez kliknięcie lewym przyciskiem myszy, wybierany jest obiekt wzorcowy dla algorytmów rozpoznawania obrazu.

Rozdział 6

Przeprowadzone testy

Po zakończeniu prac nad projektem przeprowadzono kilka testów mających na celu sprawdzenie funkcjonalności i rzeczywistych parametrów pracy robota.

6.1 Test transmisji bezprzewodowej

Według noty katalogowej, zasięg zainstalowanego w robocie modułu Bluetooth wynosi 100m. Jest to jednak wartość teoretyczna, ponieważ na rzeczywisty zasięg bardzo duży wpływ mają przeszkody, na które napotyka sygnał radiowy. Wszystkie testy przeprowadzane były przy maksymalnej, obsługiwanej przez moduł prędkości, wynoszącej 460.8 kbit/s. Transmisja odbywała się między robotem, a komputerem typu laptop, wyposażonym w zewnętrzny adapter Bluetooth klasy 1 – co oznacza, że jego teoretyczny zasięg również wynosi 100m.

Pierwsze testy transmisji przeprowadzono jeszcze w początkowych etapach prac nad projektem. W sytuacji, gdy sygnał radiowy na swojej drodze nie napotykał żadnych przeszkód mechanicznych, transmisja odbywała się bez błędów. W momencie, kiedy obydwa urządzenia znalazły się po przeciwnych stronach betonowej ściany, podczas przesyłania dużych pakietów danych (60kB), część informacji nie docierała do odbiorcy (z czego wszystkie bajty które dotarły, były poprawne). Gdy przesyłane były pojedyncze bajty danych, pomimo przeszkody problem utraty danych nie występował. Po dokładniejszych testach dała się zauważyć zależność, że im większe pakiety danych były wysyłane, tym większy ich procent nie docierał do odbiorcy. Oznaczało to, że mechanizm retransmisji danych zastosowany w module Bluetooth zawodzi.

Opisany problem został rozwiązany poprzez wprowadzenie dodatkowego mechanizmu retransmisji. Dane podzielono na bloki oraz wprowadzono odpowiednio do-

brane czasy oczekiwania na pojedynczy blok. Jeżeli dane nie zostaną odebrane w odpowiednim czasie, wysyłana jest komenda retransmisji i uszkodzony blok zostaje przesłany ponownie.

Po wprowadzeniu opisanego mechanizmu, możliwa stała się transmisja danych przez przeszkodę w postaci betonowej ściany o grubości 25 cm. Wcześniej przeszkoda tej grubości powodowała utratę części danych. Przy próbach transmisji przez większe przeszkody, ilość traconych danych przekraczała 90%, co uniemożliwiało odebranie danych w rozsądnym czasie.

Testy pokazały, że poprawna transmisja możliwa jest na odległość kilkunastu metrów, przez przeszkody typu betonowa ściana o grubości 25 cm, lub 2–3 ściany o grubości 10 cm. Biorąc pod uwagę specyfikę projektu, jest to wynik wystarczający.

6.2 Test systemu wizyjnego

6.2.1 Szybkość akwizycji danych

Odbiór danych z kamery odbywa się w sposób programowy i realizowany jest przez mikrokontroler. Zapis pojedynczej klatki do pamięci SRAM procesora trwa ok. 500 milisekund. Przed rozpoczęciem zapisu kolejnej klatki, obraz może być przesłany kanałem bezprzewodowym lub poddany analizie – w zależności od aktualnego trybu pracy robota.

W trybie autonomicznym, po odebraniu klatki obrazu, wykonywane są operacje przetwarzania i analizy. Dopiero po ich zakończeniu odbierana jest kolejna klatka. Kanałem bezprzewodowym przesyłany jest jedynie 1-bitowy podgląd obrazu po binaryzacji.

W trybie podglądu odebrany obraz przesyłany jest w całości kanałem bezprzewodowym. Kolejna klatka odbierana jest po zakończeniu transmisji. Warto wspomnieć, że transmisja bezprzewodowa obrazu rozpoczyna się jeszcze przed zakończeniem jego odbioru z kamery – dzięki wykorzystaniu kontrolera DMA.

W celu określenia ilości klatek obrazu, możliwych do odebrania i przetworzenia/przesłania w jednostce czasu, dokonano pomiarów przedstawionych w tabeli 6.1. Pomiary wykonano poprzez zmierzenie czasu potrzebnego na przetworzenie 10 kolejnych klatek obrazu, po czym wyliczono współczynniki FPS (Frames Per Second).

W momencie, kiedy robot działa w trybie autonomicznym i procesor zaangażowany jest w czasochłonną analizę i przetwarzanie obrazu, ilość klatek przetwarzanych

Tabela 6.1: Szybkość akwizycji obrazu

Tryb pracy	**Rozdzielczość**	**Czas dla 10 klatek**	**FPS**
podgląd	160x100, 8bit	11 sekund	0.9
podgląd	160x100, 16bit	19 sekund	0.53
autonomiczny	320x200, 8bit	21 sekund	0.48
podgląd	320x200, 8bit	35 sekund	0.29

w ciągu sekundy wynosi ok 0.5fps, co daje 1 klatkę na 2 sekundy.

Szybkość przesyłu obrazu za pośrednictwem kanału bezprzewodowego ograniczona jest maksymalną prędkością transmisji wynoszącą 460.8 kbit/s. Ilość odbieranych obrazów na jednostkę czasu, zależy więc od ich rozdzielczości oraz ilości bajtów na piksel (1 bajt dla obrazu w odcieniach szarości, 2 dla obrazu kolorowego YCbCr422) i wynosi:

- 0.9fps dla obrazu w odcieniach szarości, o rozdzielczości 160x100 pikseli
- 0.5fps dla obrazu kolorowego, o rozdzielczości 160x100 pikseli
- 0.3fps dla obrazu w odcieniach szarości, o rozdzielczości 320x200 pikseli

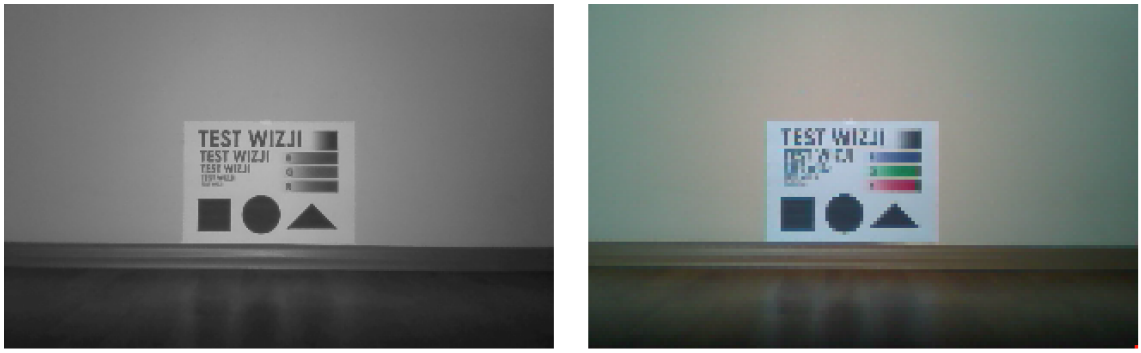
6.2.2 Jakość obrazu

W celu sprawdzenia jakości i szczegółowości odbieranego obrazu, wykonano planszę testową pokazaną na rysunku 6.1.

Rysunek 6.1: Wzór planszy testowej.

Obraz testowy został wydrukowany na arkuszu A4 (21x30 cm). Długość boku kwadratu widniejącego na obrazie, po wydrukowaniu wynosiła 5.5 cm. Próbę wykonano z odległości jednego (rys. 6.2) oraz dwóch metrów (rys. 6.3), dla rozdzielczości 320x200 oraz 160x100 pikseli.

Jak widać na rysunkach 6.2 oraz 6.3, obraz o rozdzielczości 160x100 pikseli nie oddaje poprawnie szczegółów planszy testowej, nawet z odległości jednego metra. Dlatego też wykorzystywany jest jedynie jako obraz nawigacyjny, podczas pracy w trybie zdalnego kierowania przez operatora. Największą zaletą tej rozdzielczości jest możliwość relatywnie szybkiego przesłania obrazu łączem bezprzewodowym. Obraz 320x200 pikseli, zawierający znacznie więcej detali, nadaje się do przeprowadzenia analizy kształtu obiektów z niewielkich odległości.



Rysunek 6.2: Obraz planszy testowej oddalonej o 1 metr od kamery robota. Rozdzielczość obrazu a) 320x200 pikseli, b) 160x100 pikseli.

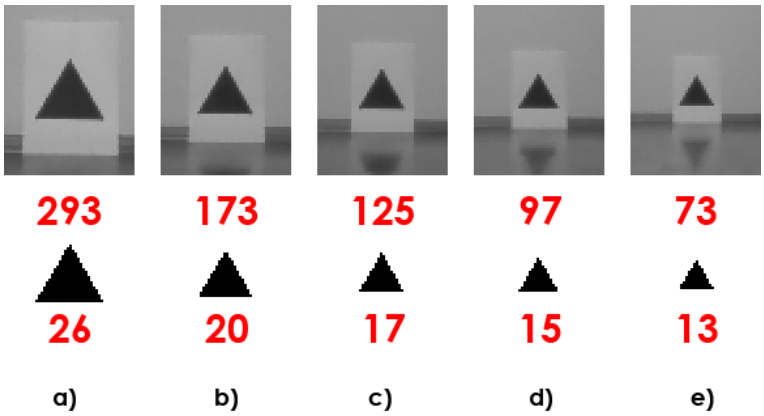
Rysunek 6.3: Obraz planszy testowej oddalonej o 2 metry od kamery robota. Rozdzielczość obrazu a) 320x200 pikseli, b) 160x100 pikseli.

6.3 Test systemu analizy i rozpoznawania

6.3.1 Rozmiary obiektów a odległość od kamery

Podczas binaryzacji odrzucane są obiekty składające się z mniej niż 100 pikseli, gdyż zawierają zbyt mało szczegółów. Dlatego też, aby rozpoznawanie obrazu przebiegło poprawnie, pole rozpoznawanych obiektów na obrazie cyfrowym (liczone w pikselach) musi mieć wartość większą niż 100. Pole liczone w ten sposób zależy od rzeczywistych wymiarów obiektu oraz od odległości obiektu od kamery. Ustalając rzeczywisty wymiar obiektu, można określić jego maksymalną odległość od kamery, dla której rozpoznawanie jest możliwe.

Test wykonany został dla trójkąta równobocznego, którego rzeczywista długość boku wynosiła 16 cm. Na rysunku 6.4 pokazano pięć obrazów trójkąta dla różnych odległości od kamery. Pokazane zdjęcia są wycinkami (50x65 pikseli) z oryginalnego obrazu otrzymanego z kamery.



Rysunek 6.4: Obrazy trójkąta o boku 16cm z odległości a) 2m, b) 2.5m, c) 3m, d) 3.5m, e) 4m. Dolny rząd pokazuje obrazy po binaryzacji. Nad każdym trójkątem podano jego pole, pod trójkątem – długość boku. Wszystkie wartości wyrażone są w pikselach.

Jak widać graniczna jest tutaj odległość wynosząca nieco ponad 3 metry. Warto zauważyć, że dla takiej odległości długość jednego piksela, to w rzeczywistości odległość 1cm. Dla odległości 3.5m, pole trójkąta wyniosło 97 pikseli – gdyby był on wzorcem, nie zostałby już poprawnie rozpoznany.

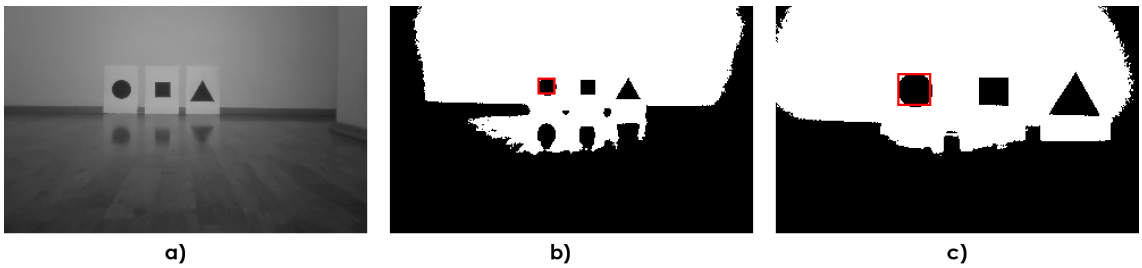
W ogólności dla wszystkich obiektów o rozmiarach porównywalnych z trójkątem testowym, rozsądna maksymalna odległość, przy której rozpoznawanie daje dobre wyniki, wynosi 2.5 metra.

6.3.2 Rozpoznawanie różnych obiektów

System analizy i rozpoznawania obrazu przetestowano dla dwóch różnych zestawów symboli. Dla każdego zestawu test rozpoczynał się od zapamiętania obiektu wzorcowego, będącego jednym z prezentowanych symboli. Następnie robot ustawiany był w odległości 2.5 metra naprzeciw obiektów testowych i uruchamiany był tryb autonomiczny.

Robot wraz z podglądem binarnym przesyła współrzędne rozpoznanego obiektu (współrzędne prostokąta obcinającego), więc poczynania robota można śledzić na bieżąco. Rozpoznany obiekt na obrazie zaznaczany jest czerwoną ramką.

Pierwszy test wykonano na symbolach widocznych na rysunku 6.5. Jako obiekt wzorcowy wybrano koło. Na rysunku widać ustawienie początkowe, oraz dwa kolejne obrazy binarne, odebrane w kilkusekundowych odstępach po uruchomieniu trybu autonomicznego.

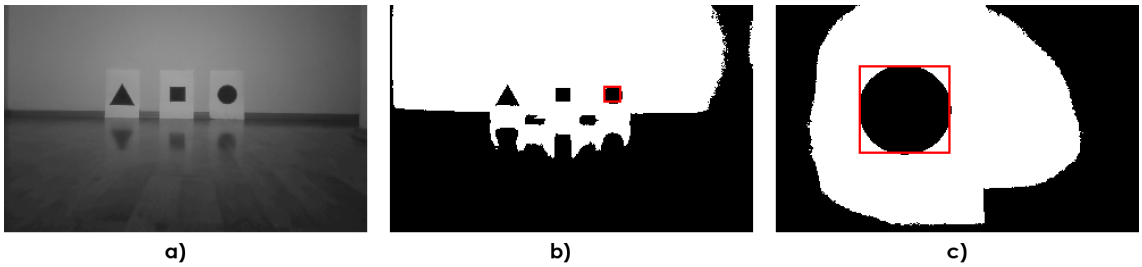


Rysunek 6.5: Pierwszy zestaw testowy; a) ustawienie początkowe, b, c) podgląd binarny po 2 i 8 sekundach od uruchomienia trybu autonomicznego.

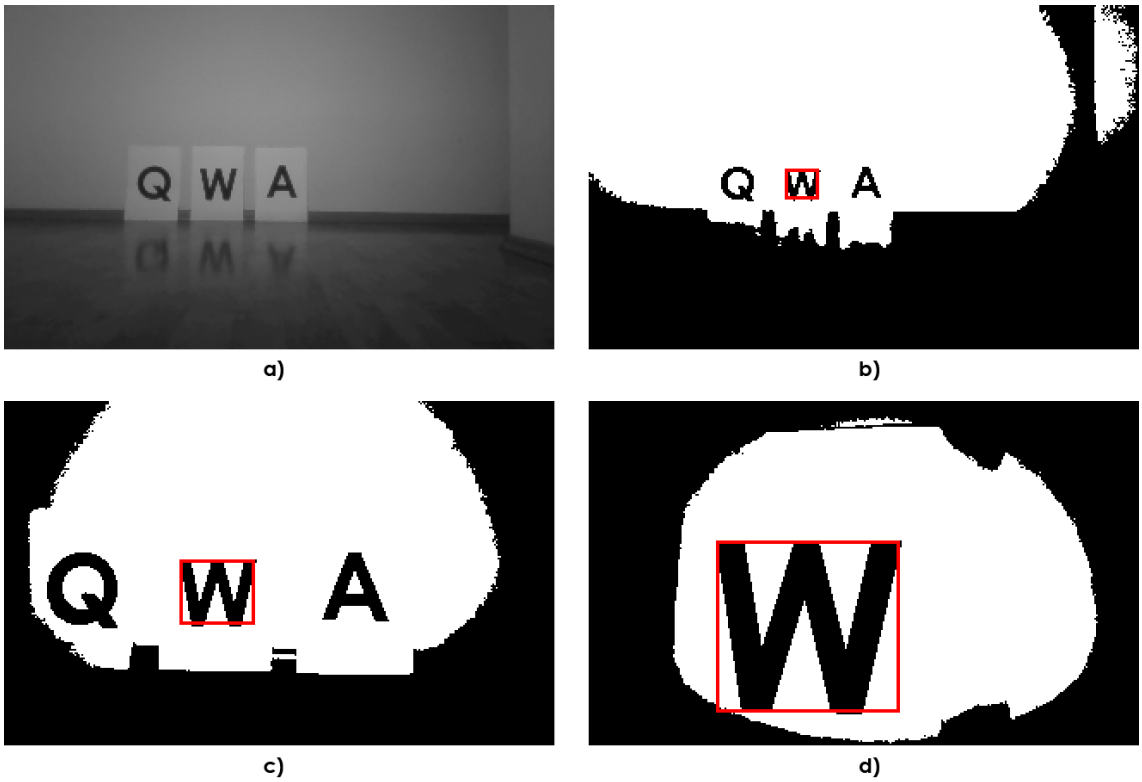
Jak widać na rysunku 6.5, robot poprawnie rozpoznaje symbol wzorcowy i zmierza w jego kierunku.

W kolejnym teście (rys. 6.6) użyto tych samych symboli lecz zamieniono ich kolejność. Robot również bezbłędnie rozpoznawał symbol wzorcowy.

W ostatnim teście rozpoznawanymi obiektami były litery. Wzorcową została wybrana litera „W". Przebieg testu ilustruje rysunek 6.7. Również w tym przypadku rozpoznawanie przebiegało poprawnie. Robot cały czas podążał w kierunku właściwego symbolu.

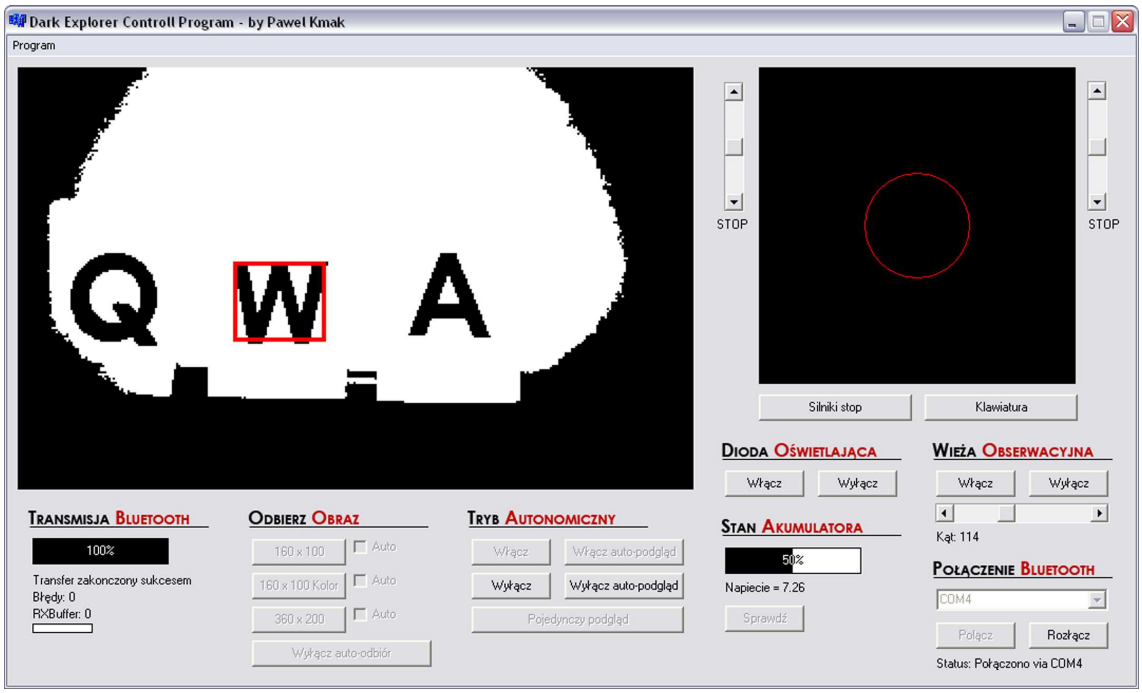


Rysunek 6.6: Pierwszy zestaw testowy – zmieniona kolejność symboli; a) ustawienie początkowe, b, c) podgląd binarny po 2 i 25 sekundach od uruchomienia trybu autonomicznego.



Rysunek 6.7: Drugi zestaw testowy; a) ustawienie początkowe, b, c, d) podgląd binarny po 2, 10 i 30 sekundach od uruchomienia trybu autonomicznego.

Na rysunku 6.8 pokazano okno programu nadzorującego, z włączoną funkcją automatycznego przesyłania podglądu obrazu, przetwarzanego przez robota podczas pracy w trybie autonomicznym.



Rysunek 6.8: Program nadzorujący, pracujący w trybie auto-podgląd, automatycznie odbiera od robota wyniki rozpoznawania obrazu.

Podsumowanie

Celem niniejszej pracy było stworzenie od podstaw robota mobilnego wraz z systemem sterowania, działającym w oparciu o wykonywaną przez procesor sterujący analizę obrazu pozyskiwanego z pokładowej kamery. Prace prowadzone były równolegle na pięciu różnych poziomach projektu, których szczegółowymi celami było:

1. Zaprojektowanie i wykonanie elektroniki sterującej.
2. Zaprojektowanie i wykonanie systemu wizji
3. Zaprojektowanie i wykonanie mechaniki.
4. Napisanie oprogramowania sterującego dla mikrokontrolera.
5. Napisanie oprogramowania do zdalnej komunikacji dla komputera PC.

Elektronikę sterującą zdecydowano się zrealizować w oparciu o 32-bitowy mikrokontroler ARM7. Następnie dobrano odpowiednie elementy elektroniczne, w oparciu o które zrealizowano akwizycję obrazu, komunikację bezprzewodową, zasilanie oraz sterowanie podzespołami wykonawczymi. Ostatecznie zaprojektowano i wysłano do produkcji 2 płytki drukowane integrujące elektronikę robota. Gotowe płytki zmontowano, przetestowano i uruchomiono.

System wizji wykonano w oparciu o moduł kamery Pixelplus PO6030K, podłączony bezpośrednio do mikrokontrolera sterującego pracą robota. Dane z kamery przesyłane są ośmiobitowym interfejsem równoległym do pamięci SRAM mikrokontrolera.

Prace nad mechaniką rozpoczęto od dobrania elementów takich jak silniki, akumulatory, serwomechanizm, koła itp. Następnie zaprojektowano i wykonano obudowę zapewniającą zakładaną funkcjonalność – między innymi możliwość zmiany kąta kamery w płaszczyźnie pionowej. Ostatecznie zintegrowano wszystkie elementy mechaniczne i elektroniczne w jednej obudowie.

Tworzenie oprogramowania dla mikrokontrolera rozpoczęto od napisania kodu odpowiedzialnego za konfigurację i sterowanie urządzeń peryferyjnych procesora (PDC, ADC, PWM, UART, I^2C itp.). Następnie oprogramowano obsługę i odbiór danych z kamery oraz stworzono protokół komunikacji realizowany bezprzewodowo w oparciu o system Bluetooth. Po zrealizowaniu sterowania wszystkich podzespołów wykonawczych robota, przystąpiono do zaimplementowania analizy i rozpoznawania obrazu. Zapewniło to możliwość samodzielnego podejmowania decyzji przez robota, w oparciu o informacje pozyskiwane z otoczenia.

Stworzenie oprogramowania do zdalnej komunikacji polegało na zaprojektowaniu aplikacji z interfejsem graficznym, komunikującej się z robotem poprzez stworzony wcześniej protokół, realizowany w oparciu o transmisję bezprzewodową Bluetooth. Dodatkowo w ramach aplikacji stworzono system retransmisji uszkodzonych danych.

Stworzony projekt realizuje w pełni wszystkie cele niniejszej pracy. Zaprojektowana platforma mobilna została zrealizowana w sposób pozwalający na łatwą rozbudowę. Część sprzętowa może zostać wyposażona w nowe czujniki i z powodzeniem posłużyć jako baza do dalszego rozwoju systemu sterowania. Dzięki zaimplementowanej komunikacji bezprzewodowej działanie wszystkich obecnych i przyszłych funkcji może być wygodnie monitorowane i nadzorowane zdalnie.

Wykorzystanie kamery i zaprogramowanie od podstaw algorytmów analizy i rozpoznawania obrazu było pouczające. Należało zmierzyć się z wieloma występującymi po drodze problemami. Jednym z większych problemów okazała się efektywna akwizycja, obróbka i analiza danych z kamery, ze względu na niewystarczające – jak się okazało – zasoby pamięciowe i obliczeniowe zastosowanego mikrokontrolera. Ostatecznie udało się stworzyć system przetwarzający obraz z prędkością ok. 0.5 klatki na sekundę, co można uznać za wynik dobry, biorąc pod uwagę prototypowy charakter systemu.

Dla ewentualnych kolejnych wersji platformy mobilnej, mogącej przetwarzać obrazy o większej rozdzielczości i z większą szybkością, optymalnym byłoby zastosowanie dodatkowej szybkiej logiki dedykowanej do celów akwizycji, przetwarzania i analizy obrazu, która mogłaby być zrealizowana na bazie programowalnych układów FPGA. Dodatkowym ulepszeniem byłoby zastosowanie mikrokontrolera klasy ARM9.

Dodatek A

Oprogramowanie mikrokontrolera SAM7

W załączniku tym przedstawiono oprogramowanie mikrokontrolera AT91SAM7S sterującego praca robota. Załączone pliki zawierają podstawowe makrodefinicje (pio.h, board.h), program główny (main.c), procedury obsługujące i inicjalizujące urządzenia peryferyjne mikrokontrolera (peripherals.c), procedury obliczeniowe i sterujące wyższego poziomu (utils.c) oraz główny program analizujący i rozpoznający obraz (rozpoznawanie.c).

A.1 pio.h

```
/* wyjscia procesora */
#define PIO_PA0     0x1 << 0
#define PIO_PA1     0x1 << 1
#define PIO_PA2     0x1 << 2
#define PIO_PA3     0x1 << 3
#define PIO_PA4     0x1 << 4
#define PIO_PA5     0x1 << 5
#define PIO_PA6     0x1 << 6
#define PIO_PA7     0x1 << 7
#define PIO_PA8     0x1 << 8
#define PIO_PA9     0x1 << 9
#define PIO_PA10    0x1 << 10
#define PIO_PA11    0x1 << 11
#define PIO_PA12    0x1 << 12
#define PIO_PA13    0x1 << 13
#define PIO_PA14    0x1 << 14
#define PIO_PA15    0x1 << 15
#define PIO_PA16    0x1 << 16
#define PIO_PA17    0x1 << 17
#define PIO_PA18    0x1 << 18
#define PIO_PA19    0x1 << 19
#define PIO_PA20    0x1 << 20
#define PIO_PA21    0x1 << 21
#define PIO_PA22    0x1 << 22
#define PIO_PA23    0x1 << 23
#define PIO_PA24    0x1 << 24
#define PIO_PA25    0x1 << 25
#define PIO_PA26    0x1 << 26
#define PIO_PA27    0x1 << 27
#define PIO_PA28    0x1 << 28
#define PIO_PA29    0x1 << 29
#define PIO_PA30    0x1 << 30
#define PIO_PA31    0x1 << 31

/* funkcje wyjśc - plytka */
#define SERWO_PWM   PIO_PA0
#define LED_POWER   PIO_PA1
#define SERWO_POWER PIO_PA2
#define I2C_DATA    PIO_PA3
#define I2C_CLOCK   PIO_PA4
#define UART_RX     PIO_PA5
#define UART_TX     PIO_PA6
#define IN1         PIO_PA7
#define IN2         PIO_PA8
#define IN3         PIO_PA9
#define IN4         PIO_PA10
#define PWM1        PIO_PA11
#define PWM2        PIO_PA12
#define PWM3        PIO_PA13
#define PWM4        PIO_PA14
#define VSYNC       PIO_PA25
#define HSYNC       PIO_PA26
#define MCLK        PIO_PA27
#define CAM_D7      PIO_PA21
#define CAM_D6      PIO_PA20
#define CAM_D5      PIO_PA19
#define CAM_D4      PIO_PA18
#define CAM_D3      PIO_PA17
#define CAM_D2      PIO_PA16
#define CAM_D1      PIO_PA22
#define CAM_D0      PIO_PA24
#define CAM_RESET   PIO_PA23

/* wolne wyjscia PIO */
//2 diodki sygnalizacyjne
#define DIODA1      PIO_PA28
#define DIODA2      PIO_PA29
//nieuzywane
//PIO_PA15 PIO_PA30 PIO_PA31
```

A.2 board.h

```
#ifndef Board_h
#define Board_h

#include "AT91SAM7S256.h"
#define __inline static inline
#include "lib_AT91SAM7S256.h"

#define __ramfunc

#define EXT_OC      18432000   // Exetrnal ocilator MAINCK
//#define MCK       47923200   // MCK (PLLRC div by 2)
#define MCK         95846400   // MCK (PLLRC)
#define MCKKHz      (MCK/1000) //

#endif /* Board_h */
```

A.3 main.c

```
/**
* Projekt    : Dark Explorer firmware
* Plik       : main.c
* Zawartosc  : Inicjalizacje początkowe, obsługa przerwań systemowych,
             : interpretacja komend, pętla główna
* Autor      : Paweł Kmak
* Data       : 1.08.2009
**/

// Include
#include "board.h"
#include <math.h>
#include "pio.h"
#include "peripherals.c"
#include "utils.c"
#include "rozpoznawanie.c"

//adres kamery do zapisu przez TWI
#define PO6030K_DEVICE_ID 0x6E

//flaga dla timera PIT
int PitState=0;

//Ustawienie polozenia serwa
//1000 - 0, 1500-45, 2000 - 90
//2850 - wieza zlozona
//1750 - postawiona pionowo
//900 - maksymalny wyhyl do przodu
volatile int SerwoAngle=1750;
volatile int SerwoAngleBuffer=1750;

//zmienne do sterowania transmisją UART
char RX_Buffer[2]; //odebrane dane, 0-komenda, 1-dana
int RX_Counter=0; //wskazuje czy byla odebrana komenda

//zmienne konfiguracyjne modulu bluetooth
char bluetooth[] = "ATK?\r";

//Rozmiar ramki do odebrania
volatile char FrameSizeToGet=0;

//Wlacznik trybu autonomicznego
char AutoMode=0, AutoPreview=0;

//pamiec obrazu
char mem[64000];

//tryb auto
```

```
49 //Stos dla SmithFill - mem od 16000
50 int StackIndex=16000;
51 char MaxIndex=0;
52
53 float wzorzec1=0, wzorzec2=0, wzorzec3=0, wzorzec4=0, wzorzec5=0, wzorzec6=0, wzorzec7=0, wzorzec8=0, wzorzec9=0;
54 int xwzorzec=0, ywzorzec=0, WzorzecCount=0, RozpoznanyX, RozpoznanyY;
55
56 char SilnikiEnable=0;
57
58 volatile int xmin, xmax, ymin, ymax;
59
60 ////////////////////////////////////////////////////////////////////////////
61 // Przerwanie od timera PIT
62 ////////////////////////////////////////////////////////////////////////////
63 __ramfunc void SYSTEM_INTERRUPT_irq_handler(void)
64 {
65   //impuls od timera PIT pojawia się co 18-20ms
66
67   unsigned int dummy;
68
69   //odczyt rejestru PIVR (powoduje wyzerowanie flagi żądania przerwania)
70   dummy = AT91C_BASE_PITC->PITC_PIVR;
71
72   //obliczenie pozycji serwa
73   if (SerwoAngleBuffer>SerwoAngle) {SerwoAngle+=5;}
74   if (SerwoAngleBuffer<SerwoAngle) {SerwoAngle-=5;}
75
76   ////////////// STEROWANIE SERWEM ////////////////////////////////////////////////
77   if(PitState==0)
78   {
79     //zmiana interwału na czas 1-2 ms
80     //do wygenerowania impulsu dla serwa
81     //dlugosc impulsu = wychylenie serwa
82     PIT_ChangeInterval(SerwoAngle);
83     PitState=1;
84
85     //wystawnienie jedynki
86     AT91F_PIO_SetOutput( AT91C_BASE_PIOA, PIO_PA0 );
87   }
88   else {
89          //zmiana interwału na czas 18 ms
90          //do wygenerowania odstępu między impulsami
91          PIT_ChangeInterval(18000); //18ms
92          PitState=0;
93
94          //wystawnienie zera
95          AT91F_PIO_ClearOutput( AT91C_BASE_PIOA, PIO_PA0 );
96        }
97   ////////////////////////////////////////////////////////////////////////////
98 }
99
100 ////////////////////////////////////////////////////////////////////////////
101 // Przerwanie od kanalu odbiorczego UART0
102 ////////////////////////////////////////////////////////////////////////////
103 __ramfunc void UART0_DMA_irq_handler(void)
104 {
105 RX_Buffer[RX_Counter] = UART0_Read();
106 RX_Counter++;
107 if (RX_Counter==2) { RX_Counter=0; } //2 bajty na ramke
108
109 //Dekodowanie komend
110 if (RX_Counter==0) //po odebraniu komendy i danych
111 {
112   //Sterowanie serwem
113   if(RX_Buffer[0] == 's')
114   {
115     SerwoAngleBuffer=1065 + ((255-RX_Buffer[1])*7);
116   }
117
118   //Sterowanie kanalem pwm 0, 1
119   if(RX_Buffer[0] == 'a')
120   {
```

```
121     PWM_Set(0, RX_Buffer[1]);
122     PWM_Set(1, RX_Buffer[1]);
123   }
124
125   //Sterowanie kanalem pwm 2, 3
126   if(RX_Buffer[0] == 'b')
127   {
128     PWM_Set(2, RX_Buffer[1]);
129     PWM_Set(3, RX_Buffer[1]);
130   }
131
132   //silniki 1-2 kierunek (lewe)
133   if(RX_Buffer[0] == 'l')
134   {
135     Kierunek(1, RX_Buffer[1]);
136   }
137
138   //silniki 3-4 kierunek (prawe)
139   if(RX_Buffer[0] == 'r')
140   {
141     Kierunek(2, RX_Buffer[1]);
142   }
143
144   //pomiar napiecia beterii
145   if(RX_Buffer[0] == 't')
146   {
147     ADC_StartConversion(AT91C_ADC_CH4);
148      RX_Buffer[0]=ADC_Read_2bit(AT91C_ADC_CH4); //2 starsze bity
149     RX_Buffer[1]=ADC_Read_8bit(AT91C_ADC_CH4); //mlodszy bajt
150      UART0_DMA_Write(&RX_Buffer[0], 2);
151   }
152
153   //wyslanie klatki obrazu 320 x 200
154   if(RX_Buffer[0] == 'f')
155   {
156     if (RX_Buffer[1]==0)
157     {
158       FrameSizeToGet=2;
159     }
160
161     if (RX_Buffer[1]>0 && RX_Buffer[1]<20)
162     {
163       UART0_DMA_Write(&mem[3200*RX_Buffer[1]], 3200);
164     }
165   }
166
167   //wyslanie klatki obrazu 160 x 100
168   if(RX_Buffer[0] == 'p')
169   {
170     if (RX_Buffer[1]==0)
171     {
172       FrameSizeToGet=1;
173     }
174
175     if (RX_Buffer[1]>0 && RX_Buffer[1]<5)
176     {
177       UART0_DMA_Write(&mem[3200*RX_Buffer[1]], 3200);
178     }
179   }
180
181   //wyslanie klatki obrazu 160 x 100 kolorowego
182   if(RX_Buffer[0] == 'x')
183   {
184     if (RX_Buffer[1]==0)
185     {
186       FrameSizeToGet=3;
187     }
188
189     if (RX_Buffer[1]>0 && RX_Buffer[1]<10)
190     {
191       UART0_DMA_Write(&mem[3200*RX_Buffer[1]], 3200);
192     }
```

```
193   }
194
195   //tryb autonomiczny komenda i
196   if(RX_Buffer[0] == 'i')
197   {
198     if (RX_Buffer[1]==10)
199     {
200       AutoMode=1;
201     }
202
203     if (RX_Buffer[1]==11)
204     {
205       AutoMode=0;
206     }
207
208     if (RX_Buffer[1]==12)
209     {
210       AutoPreview=1;
211     }
212
213     if (RX_Buffer[1]==13)
214     {
215       AutoPreview=0;
216     }
217
218     //podglad bufora obrazu - automatyczny
219     if (RX_Buffer[1]<5)
220     {
221       UART0_DMA_Write(&mem[(3200*RX_Buffer[1])+48000], 3200);
222     }
223   }
224
225   //tryb autonomiczny komenda j
226   if(RX_Buffer[0] == 'j')
227   {
228     //podglad bufora obrazu - manualny
229     if (RX_Buffer[1]<5)
230     {
231       UART0_DMA_Write(&mem[(3200*RX_Buffer[1])], 3200);
232     }
233   }
234
235   //zapamietanie wskazanego wzorca - wspolrzedna x
236   if(RX_Buffer[0] == 'm')
237   {
238     xwzorzec = RX_Buffer[1];
239     SilnikiEnable=1;
240   }
241
242   //zapamietanie wskazanego wzorca - wspolrzedna y
243   if(RX_Buffer[0] == 'n')
244   {
245     ywzorzec = RX_Buffer[1];
246   }
247
248   //dioda led
249   if(RX_Buffer[0] == 'd')
250   {
251     //wylaczenie diody
252     if (RX_Buffer[1]==0)
253     {
254       AT91F_PIO_SetOutput( AT91C_BASE_PIOA, LED_POWER );
255     }
256
257     //wlaczenie diody
258     if (RX_Buffer[1]==1)
259     {
260       AT91F_PIO_ClearOutput( AT91C_BASE_PIOA, LED_POWER );
261     }
262   }
263
264   //serwo zasilanie
```

```
265   if(RX_Buffer[0] == 'w')
266   {
267     //wylaczenie
268     if (RX_Buffer[1]==0)
269     {
270       AT91F_PIO_ClearOutput( AT91C_BASE_PIOA, SERWO_POWER );
271     }
272
273     //wlaczenie
274     if (RX_Buffer[1]==1)
275     {
276       AT91F_PIO_SetOutput( AT91C_BASE_PIOA, SERWO_POWER );
277     }
278   }
279 }
280 }
281
282 ////////////////////////////////////////////////////////////////////////////
283 // Main
284 ////////////////////////////////////////////////////////////////////////////
285 int main(void)
286 {
287   // Inicjalizacje
288
289   // Enable User Reset and set its minimal assertion to 960 us
290   AT91C_BASE_RSTC->RSTC_RMR = AT91C_RSTC_URSTEN | (0x4<<8) | (unsigned int)(0xA5<<24);
291
292   // mt: added reset enable to make the board reset-button "useful"
293   AT91F_RSTSetMode( AT91C_BASE_RSTC , AT91C_RSTC_URSTEN );
294
295   // Enable the clock of the PIOA
296   AT91F_PMC_EnablePeriphClock ( AT91C_BASE_PMC, 1 << AT91C_ID_PIOA );
297
298   // Włączenie timera PIT
299   PIT_Configure(100); //pierwsze przerwanie za 100us
300
301   // Wlaczenie PWM
302   PWM_Configure();
303
304   //wlaczenie UART0
305   //UART0_DMA_Configure(115200); //115.2kbit/s
306   //UART0_DMA_Configure(230400); //230.4kbit/s
307   UART0_DMA_Configure(460800); //460.8kbit/s
308
309   //wlaczenie ADC
310   ADC_Configure(1000000); //1MHz
311
312   //wlaczenie TWI
313   //TWI_Configure(400000); //400KHz
314
315   //konfiguracja wyjścia serwa
316   AT91F_PIO_CfgOutput( AT91C_BASE_PIOA, PIO_PA0 );
317   AT91F_PIO_CfgPullup( AT91C_BASE_PIOA, ~PIO_PA0 );
318
319   //konfiguracja zasilania serwa
320   AT91F_PIO_CfgOutput( AT91C_BASE_PIOA, SERWO_POWER );
321   AT91F_PIO_CfgPullup( AT91C_BASE_PIOA, ~SERWO_POWER );
322   //wylaczenie zasilania serwa
323   AT91F_PIO_ClearOutput( AT91C_BASE_PIOA, SERWO_POWER );
324
325   //konfiguracja wyjścia diody led mocy
326   AT91F_PIO_CfgOutput( AT91C_BASE_PIOA, LED_POWER );
327   AT91F_PIO_CfgPullup( AT91C_BASE_PIOA, ~LED_POWER );
328   //zgaszenie diody led mocy
329   AT91F_PIO_SetOutput( AT91C_BASE_PIOA, LED_POWER );
330
331   //konfiguracja wyjsc kierunkowych silnikow (in1-in4)
332   AT91F_PIO_CfgOutput( AT91C_BASE_PIOA, PIO_PA7 );  //in1
333   AT91F_PIO_CfgOutput( AT91C_BASE_PIOA, PIO_PA8 );  //in2
334   AT91F_PIO_CfgOutput( AT91C_BASE_PIOA, PIO_PA9 );  //in3
335   AT91F_PIO_CfgOutput( AT91C_BASE_PIOA, PIO_PA10 ); //in4
336
```

```
337  //konfiguracja linii kamery cam po6030k
338  AT91F_PIO_CfgOutput( AT91C_BASE_PIOA, CAM_RESET ); //reset
339  AT91F_PIO_CfgOutput( AT91C_BASE_PIOA, MCLK ); //mclk
340  AT91F_PIO_CfgInput( AT91C_BASE_PIOA, VSYNC ); //vsync
341  AT91F_PIO_CfgPullup( AT91C_BASE_PIOA, ~VSYNC );
342  AT91F_PIO_CfgInput( AT91C_BASE_PIOA, HSYNC ); //hsync
343  AT91F_PIO_CfgPullup( AT91C_BASE_PIOA, ~HSYNC );
344  AT91F_PIO_CfgInput( AT91C_BASE_PIOA, CAM_D0 ); //D0
345  AT91F_PIO_CfgPullup( AT91C_BASE_PIOA, ~CAM_D0 );
346  AT91F_PIO_CfgInput( AT91C_BASE_PIOA, CAM_D1 ); //D1
347  AT91F_PIO_CfgPullup( AT91C_BASE_PIOA, ~CAM_D1 );
348  AT91F_PIO_CfgInput( AT91C_BASE_PIOA, CAM_D2 ); //D2
349  AT91F_PIO_CfgPullup( AT91C_BASE_PIOA, ~CAM_D2 );
350  AT91F_PIO_CfgInput( AT91C_BASE_PIOA, CAM_D3 ); //D3
351  AT91F_PIO_CfgPullup( AT91C_BASE_PIOA, ~CAM_D3 );
352  AT91F_PIO_CfgInput( AT91C_BASE_PIOA, CAM_D4 ); //D4
353  AT91F_PIO_CfgPullup( AT91C_BASE_PIOA, ~CAM_D4 );
354  AT91F_PIO_CfgInput( AT91C_BASE_PIOA, CAM_D5 ); //D5
355  AT91F_PIO_CfgPullup( AT91C_BASE_PIOA, ~CAM_D5 );
356  AT91F_PIO_CfgInput( AT91C_BASE_PIOA, CAM_D6 ); //D6
357  AT91F_PIO_CfgPullup( AT91C_BASE_PIOA, ~CAM_D6 );
358  AT91F_PIO_CfgInput( AT91C_BASE_PIOA, CAM_D7 ); //D7
359  AT91F_PIO_CfgPullup( AT91C_BASE_PIOA, ~CAM_D7 );
360
361  AT91F_PIO_CfgOutput( AT91C_BASE_PIOA, DIODA1 ); //dioda vsync
362  AT91F_PIO_CfgOutput( AT91C_BASE_PIOA, DIODA2 ); //dioda drop frame
363  //zgaszenie diod
364  AT91F_PIO_SetOutput( AT91C_BASE_PIOA, DIODA1 );
365  AT91F_PIO_SetOutput( AT91C_BASE_PIOA, DIODA2 );
366
367  // INICJALIZACJA KAMERY
368  // kamera w stanie reset
369  AT91F_PIO_ClearOutput( AT91C_BASE_PIOA, CAM_RESET );
370
371  // 100 okresow zegara w stanie resetu
372  int cam_clk;
373  for(cam_clk=0; cam_clk<100; cam_clk++)
374  {
375   AT91F_PIO_ClearOutput( AT91C_BASE_PIOA, MCLK );
376   AT91F_PIO_SetOutput( AT91C_BASE_PIOA, MCLK );
377  }
378
379  // wyjscie ze stanu reset
380  AT91F_PIO_SetOutput( AT91C_BASE_PIOA, CAM_RESET );
381
382  // pojedyncze tykniecie zegara
383  AT91F_PIO_ClearOutput( AT91C_BASE_PIOA, MCLK );
384  AT91F_PIO_SetOutput( AT91C_BASE_PIOA, MCLK );
385
386  //Konfiguracja modulu bluetooth
387  /*waitms(50);
388  UART0_DMA_Write(&bluetooth[0], 1);
389  waitms(50);
390  UART0_DMA_Write(&bluetooth[1], 1);
391  waitms(50);
392  UART0_DMA_Write(&bluetooth[2], 1);
393  waitms(50);
394  UART0_DMA_Write(&bluetooth[3], 1);
395  waitms(50);
396  UART0_DMA_Write(&bluetooth[4], 1);*/
397
398
399  for (;;)
400  {
401    switch (FrameSizeToGet) {
402      case 1://160x100 mono preview
403        GetFrame( 4, 4, 1 );
404        FrameSizeToGet=0;
405        break;
406      case 2: //320x200
407        GetFrame( 2, 2, 1 );
408        FrameSizeToGet=0;
```

```
409         break;
410       case 3://160x100 color
411         GetFrame( 16, 4, 1 );
412         FrameSizeToGet=0;
413         break;
414       }
415
416     if(AutoMode)
417     {
418       MaxIndex = Rozpoznaj();
419     }
420   } //end for
421 } //end main
```

A.4 peripherals.c

```
1  /**
2  * Projekt    : Dark Explorer firmware
3  * Plik       : perupherals.c
4  * Zawartosc  : Funkcje do obsługi urządzeń peryferyjnych procesora
5  * Autor      : Paweł Kmak
6  * Data       : 1.08.2009
7  **/
8
9  ////////////////////////////////////////////////////////////////////////////
10 // Prototypy funkcji obsługi przerwan
11 ////////////////////////////////////////////////////////////////////////////
12 __ramfunc void SYSTEM_INTERRUPT_irq_handler(void);
13 __ramfunc void UART0_DMA_irq_handler(void);
14
15 ////////////////////////////////////////////////////////////////////////////
16 // Opoznienie ok 1ms przy MCK=48MHz
17 ////////////////////////////////////////////////////////////////////////////
18 void waitms(volatile unsigned long d)
19 {
20   // dla MCK 48MHz
21   d=2000*d;
22   for (;d ;--d);
23 }
24
25 ////////////////////////////////////////////////////////////////////////////
26 // Konfiguracja timera PIT (serwo)
27 ////////////////////////////////////////////////////////////////////////////
28 void PIT_Configure(int czas_us)
29 {
30   unsigned int dummy, piv;
31   float tmp;
32
33   //wyłączenie timera PIT na czas konfiguracji
34   AT91C_BASE_PITC->PITC_PIMR = ~(AT91C_PITC_PITEN | AT91C_PITC_PITIEN);
35
36   //oczekiwanie na wyzerowanie licznika (pole CPIV w rejestrze PIVR)
37   while(AT91C_BASE_PITC->PITC_PIVR & AT91C_PITC_CPIV);
38
39   //wyzerowanie potencjalnego żądania przerwania
40   dummy = AT91C_BASE_PITC->PITC_PIVR;
41
42   //konfiguracja przerwania
43   AT91F_AIC_ConfigureIt ( AT91C_BASE_AIC, AT91C_ID_SYS, AT91C_AIC_PRIOR_LOWEST, AT91C_AIC_SRCTYPE_HIGH_LEVEL, SYSTEM_INTERRUPT_irq_handler);
44   AT91F_AIC_EnableIt (AT91C_BASE_AIC, AT91C_ID_SYS);
45
46   //włączenie timera PIT i ustawienie interwału
47   //piv = (   ( czas_us * ( (MCK/16)/1000 ) ) / 1000   );
48   tmp = ( (((float)MCK)/(16000000.0)) * (float)czas_us ) - 1.0;
49   piv = tmp;
50   AT91C_BASE_PITC->PITC_PIMR = ( piv | AT91C_PITC_PITEN | AT91C_PITC_PITIEN );
51 }
52
```

```
53 ////////////////////////////////////////////////////////////////////////////
54 // Zmiana interwału timera PIT
55 ////////////////////////////////////////////////////////////////////////////
56 void PIT_ChangeInterval(int czas_us)
57 {
58   unsigned int piv;
59   float tmp;
60   tmp = ( (((float)MCK)/(16000000.0)) * (float)czas_us ) - 1.0;
61   piv = tmp;
62   AT91C_BASE_PITC->PITC_PIMR = ( piv | AT91C_PITC_PITEN | AT91C_PITC_PITIEN );
63 }
64
65 ////////////////////////////////////////////////////////////////////////////
66 // Konfiguracja 4-rech kanałów PWM dla silników
67 ////////////////////////////////////////////////////////////////////////////
68 void PWM_Configure()
69 {
70   //wlaczenie zegara dla pwm
71   AT91F_PMC_EnablePeriphClock ( AT91C_BASE_PMC, 1 << AT91C_ID_PWMC );
72
73   //wylaczenie wszystkich kanalow przed zmiana konfiguracji
74   AT91C_BASE_PWMC->PWMC_DIS = (AT91C_PWMC_CHID0 | AT91C_PWMC_CHID1 | AT91C_PWMC_CHID2 | AT91C_PWMC_CHID3);
75
76   //oczekiwanie na wyłączenie wszystkich kanalow
77   while(AT91C_BASE_PWMC->PWMC_SR & (AT91C_PWMC_CHID0 | AT91C_PWMC_CHID1 | AT91C_PWMC_CHID2 | AT91C_PWMC_CHID3) );
78
79   //KONFIGURACJA TRYBU PRACY
80   //    - prescaler = MCK/64
81   //    - alligment = center
82   //    - polarity = high
83   //    - update = duty
84   AT91C_BASE_PWMC->PWMC_CH[0].PWMC_CMR = (0x6 << 0 | 0x1 << 8 | 0x1 << 9 | 0x0 << 10);
85   AT91C_BASE_PWMC->PWMC_CH[1].PWMC_CMR = (0x6 << 0 | 0x1 << 8 | 0x1 << 9 | 0x0 << 10);
86   AT91C_BASE_PWMC->PWMC_CH[2].PWMC_CMR = (0x6 << 0 | 0x1 << 8 | 0x1 << 9 | 0x0 << 10);
87   AT91C_BASE_PWMC->PWMC_CH[3].PWMC_CMR = (0x6 << 0 | 0x1 << 8 | 0x1 << 9 | 0x0 << 10);
88
89   // rozdzielczosc = 255
90   AT91C_BASE_PWMC->PWMC_CH[0].PWMC_CPRDR = 256;
91   AT91C_BASE_PWMC->PWMC_CH[1].PWMC_CPRDR = 256;
92   AT91C_BASE_PWMC->PWMC_CH[2].PWMC_CPRDR = 256;
93   AT91C_BASE_PWMC->PWMC_CH[3].PWMC_CPRDR = 256;
94
95   // wypelnienie poczatkowe = 1
96   AT91C_BASE_PWMC->PWMC_CH[0].PWMC_CDTYR = 2;
97   AT91C_BASE_PWMC->PWMC_CH[1].PWMC_CDTYR = 2;
98   AT91C_BASE_PWMC->PWMC_CH[2].PWMC_CDTYR = 2;
99   AT91C_BASE_PWMC->PWMC_CH[3].PWMC_CDTYR = 2;
100
101  //KONFIGURACJA WYPROWADZENIA PA11, PA12, PA13, PA14
102  AT91F_PIO_CfgPeriph(AT91C_BASE_PIOA, 0, (AT91C_PA11_PWM0 | AT91C_PA12_PWM1 | AT91C_PA13_PWM2 | AT91C_PA14_PWM3) );
103
104  //Uruchomienie wszystkich kanałów PWM
105  AT91C_BASE_PWMC->PWMC_ENA = (AT91C_PWMC_CHID0 | AT91C_PWMC_CHID1 | AT91C_PWMC_CHID2 | AT91C_PWMC_CHID3);
106 }
107
108 ////////////////////////////////////////////////////////////////////////////
109 // Ustawianie wypełnienia dla wybranego kanału
110 ////////////////////////////////////////////////////////////////////////////
111 void PWM_Set(int channel, char duty)
112 {
113   //ustawienie wypelnienia
114   // channel = 0 - 3, duty = 2 - 255
115   AT91C_BASE_PWMC->PWMC_CH[channel].PWMC_CUPDR = duty;
116 }
117
118 ////////////////////////////////////////////////////////////////////////////
119 // Kontroler DMA - wyłączenie kanałów nadawczego i odbiorczego
120 ////////////////////////////////////////////////////////////////////////////
121 void PDC_Disable(AT91PS_PDC pPDC)
122 {
123   //wylaczenie nadajnika i odbiornika
124   pPDC->PDC_PTCR = (AT91C_PDC_TXTDIS | AT91C_PDC_RXTDIS);
```

```
125
126   //zerowanie rejestrow
127   pPDC->PDC_TPR = 0;      // Transmit Pointer Register
128   pPDC->PDC_TCR = 0;      // Transmit Counter Register
129   pPDC->PDC_TNPR = 0;     // Transmit Next Pointer Register
130   pPDC->PDC_TNCR = 0;     // Transmit Next Counter Register
131
132   pPDC->PDC_RPR = 0;      // Receive Pointer Register
133   pPDC->PDC_RCR = 0;      // Receive Counter Register
134   pPDC->PDC_RNPR = 0;     // Receive Next Pointer Register
135   pPDC->PDC_RNCR = 0;     // Receive Next Counter Register
136 }
137
138 ////////////////////////////////////////////////////////////////////////////
139 // Kontroler DMA - włączenie kanału nadawczego
140 ////////////////////////////////////////////////////////////////////////////
141 void PDC_Enable(AT91PS_PDC pPDC)
142 {
143   //wlaczenie nadajnika /*i odbiornika*/
144   pPDC->PDC_PTCR = (AT91C_PDC_TXTEN /*| AT91C_PDC_RXTEN*/);
145 }
146
147 ////////////////////////////////////////////////////////////////////////////
148 // Kontroler DMA - przekazanie danych do nadania
149 ////////////////////////////////////////////////////////////////////////////
150 void PDC_SetTX(char *data, int ile, AT91PS_PDC pPDC)
151 {
152   //ustawienie rejestrów nadawczych
153   pPDC->PDC_TPR = (unsigned int)data;      // Transmit Pointer Register
154   pPDC->PDC_TCR = ile;                       // Transmit Counter Register
155 }
156
157 ////////////////////////////////////////////////////////////////////////////
158 // Kontroler DMA - przekazanie danych do odbioru
159 ////////////////////////////////////////////////////////////////////////////
160 void PDC_SetRX(char *data, int ile, AT91PS_PDC pPDC)
161 {
162   //ustawienie rejestrów odbiorczych
163   pPDC->PDC_RPR = (unsigned int)data;      // Receive Pointer Register
164   pPDC->PDC_RCR = ile;                       // Receive Counter Register
165 }
166
167 ////////////////////////////////////////////////////////////////////////////
168 // Konfiguracja USART0 w trybie DMA (DMA tylko do nadawania)
169 // Odbiór danych za pomocą przerwania
170 ////////////////////////////////////////////////////////////////////////////
171 void UART0_DMA_Configure(unsigned long baudrate)
172 {
173   float podzielnik_float;
174   int calkowity, ulamkowy;
175
176   //Włączenie zegara dla UART0
177   AT91F_PMC_EnablePeriphClock ( AT91C_BASE_PMC, 1 << AT91C_ID_US0 );
178
179   //wylaczenie przyjmowania przerwan
180   AT91C_BASE_US0->US_IDR=0xFFFFFFFF;
181
182   //reset portu
183   AT91C_BASE_US0->US_CR = AT91C_US_RSTRX |          /* Reset Receiver      */
184                           AT91C_US_RSTTX |          /* Reset Transmitter   */
185                           AT91C_US_RXDIS |          /* Receiver Disable    */
186                           AT91C_US_TXDIS;           /* Transmitter Disable */
187   //wylaczenie DMA
188   PDC_Disable(AT91C_BASE_PDC_US0);
189
190   //konfiguracja USART0
191   AT91C_BASE_US0->US_MR = AT91C_US_USMODE_NORMAL |   /* Normal Mode */
192                           AT91C_US_CLKS_CLOCK    |    /* Clock = MCK */
193                           AT91C_US_CHRL_8_BITS   |    /* 8-bit Data  */
194                           AT91C_US_PAR_NONE      |    /* No Parity   */
195                           AT91C_US_NBSTOP_1_BIT;      /* 1 Stop Bit  */
196   //predkosc
```

```
197   //AT91C_BASE_US0->US_BRGR = (MCK/16/baudrate);
198   //wyliczenie podzielnikow zegara MCLK
199   podzielnik_float = ( (float)MCK / 16.0 / (float)baudrate );
200   calkowity = podzielnik_float;
201   podzielnik_float = (podzielnik_float - calkowity) * 8;
202   ulamkowy = podzielnik_float;
203   AT91C_BASE_US0->US_BRGR = ( calkowity ) | ( ulamkowy << 16 );       /* podzielnik calkowity + podzielnik ulamkowy */
204
205   //konfiguracja przerwania
206   AT91F_AIC_ConfigureIt ( AT91C_BASE_AIC, AT91C_ID_US0, 0x3, AT91C_AIC_SRCTYPE_HIGH_LEVEL, UART0_DMA_irq_handler);
207   AT91F_AIC_EnableIt (AT91C_BASE_AIC, AT91C_ID_US0);
208
209   //włączenie USART0
210   AT91C_BASE_US0->US_CR = AT91C_US_RXEN  |          /* Receiver Enable     */
211                           AT91C_US_TXEN;            /* Transmitter Enable  */
212
213   //wlaczenie DMA (nadawanie)
214   PDC_Enable(AT91C_BASE_PDC_US0);
215
216   //konfiguracja wyprowadzen
217   AT91F_PIO_CfgPeriph(AT91C_BASE_PIOA, (AT91C_PA5_RXD0 | AT91C_PA6_TXD0), 0 );
218
219   //wlaczenie przerwania od odbiornika
220   AT91C_BASE_US0->US_IER = (0x1 << 0); //RXRDY
221 }
222
223 ////////////////////////////////////////////////////////////////////////////
224 // USART0 - wysyłanie danych za pomocą kontrolera DMA
225 ////////////////////////////////////////////////////////////////////////////
226 void UART0_DMA_Write(char *data, int ile)
227 {
228   //oczekiwanie na zakonczenie wysylania poprzedniego zestawu
229   while( AT91C_BASE_PDC_US0->PDC_TCR );
230
231   //rozpoczecie wysylania nowego zestawu danych
232   PDC_SetTX(data, ile, AT91C_BASE_PDC_US0);
233
234   //przerwanie gdy transfer sie zakonczy
235   //AT91C_BASE_US0->US_IER = (0x1 << 4);
236 }
237
238 ////////////////////////////////////////////////////////////////////////////
239 // USART0 - odbieranie danych po wystapieniu przerwania RXRDY
240 ////////////////////////////////////////////////////////////////////////////
241 char UART0_Read(void)
242 {
243   return (AT91C_BASE_US0->US_RHR & 0xFF);
244 }
245
246 ////////////////////////////////////////////////////////////////////////////
247 // USART0 - odbieranie danych za pomocą kontrolera DMA - obecnie nie uzywane
248 ////////////////////////////////////////////////////////////////////////////
249 void UART0_DMA_Read(char *data, int ile)
250 {
251   //oczekiwanie na zakonczenie odbierania poprzedniego zestawu
252   while( AT91C_BASE_PDC_US0->PDC_RCR );
253
254   //rozpoczecie odbierania nowego zestawu danych
255   PDC_SetRX(data, ile, AT91C_BASE_PDC_US0);
256
257   //przerwanie gdy transfer sie zakonczy
258   AT91C_BASE_US0->US_IER = (0x1 << 3);
259 }
260
261 ////////////////////////////////////////////////////////////////////////////
262 // Konfiguracja przetwornika Analogowo-Cyfrowego
263 ////////////////////////////////////////////////////////////////////////////
264 void ADC_Configure(int adc_clock)
265 {
266   //Włączenie zegara dla ADC
267   AT91F_PMC_EnablePeriphClock ( AT91C_BASE_PMC, 1 << AT91C_ID_ADC );
268
```

```
269   //zamaskowanie przerwan
270   AT91C_BASE_ADC->ADC_IDR = 0xFFFFFFFF;
271
272   //reset przetwornika
273   AT91C_BASE_ADC->ADC_CR = AT91C_ADC_SWRST;
274
275   //obliczenia parametrów PRESCALER, STARTUP, SHTIM dla zadanego adc_clock
276   ///// PRESCALER
277   int prescaler, adc_clock_real;
278   prescaler = ( MCK / (2 * adc_clock) ) - 1;
279   if (prescaler > 63){prescaler = 63;} //wartosc max
280   adc_clock_real = ( MCK / (2 * (prescaler + 1)) );
281
282   ///// STARTUP
283   int startup;
284   startup = ( (10 * adc_clock_real) / (8*50000) );
285   if ( (startup % 10) >= 1) { startup = (((startup / 10) + 1) - 1); }
286     else { startup = ((startup / 10) - 1); }
287
288   ///// SHTIM
289   int sh;
290   sh = ( (100 * adc_clock_real) / ( 100000 ) );
291   if ( (sh % 100) >= 1 ) { sh = (((sh / 100) + 1) - 1); }
292     else { sh = ((sh / 100) - 1); }
293
294   //parametry pracy ADC
295   AT91C_BASE_ADC->ADC_MR = AT91C_ADC_TRGEN_DIS          |  /* wylaczone wyzwalanie sprzetowe */
296                            AT91C_ADC_LOWRES_10_BIT      |  /* rozdzielczosc 10bit */
297                            AT91C_ADC_SLEEP_NORMAL_MODE  |  /* tryb normalny */
298                            prescaler << 8               |  /* PRESCALER */
299                            startup << 16                |  /* STARTUP */
300                            sh << 24;                       /* SHTIM */
301 }
302
303 ////////////////////////////////////////////////////////////////////////////
304 // ADC - rozpoczecie konwersji analog-digital
305 ////////////////////////////////////////////////////////////////////////////
306 // Dostepne 4 kanały dedykowane
307 // AT91C_ADC_CH4 - Channel 4
308 // AT91C_ADC_CH5 - Channel 5
309 // AT91C_ADC_CH6 - Channel 6
310 // AT91C_ADC_CH7 - Channel 7
311 ////////////////////////////////////////////////////////////////////////////
312 void ADC_StartConversion(int channel_mask)
313 {
314   //disable
315   AT91C_BASE_ADC->ADC_CHDR = 0xFFFFFFFF;
316
317   //enable channel
318   AT91C_BASE_ADC->ADC_CHER = channel_mask;
319
320   //start conversion
321   AT91C_BASE_ADC->ADC_CR = AT91C_ADC_START;
322 }
323
324 ////////////////////////////////////////////////////////////////////////////
325 // ADC - odczytanie wyniku konwersji (8 bit)
326 ////////////////////////////////////////////////////////////////////////////
327 char ADC_Read_8bit(int channel_mask)
328 {
329   //oczekiwanie na zakonczenie konwersji
330   while( AT91C_BASE_ADC->ADC_SR & channel_mask );
331
332   //odczyt wyniku (8 mlodszych bitow)
333   switch (channel_mask)
334   {
335     case AT91C_ADC_CH4: return (AT91C_BASE_ADC->ADC_CDR4 & 0xFF); break;
336     case AT91C_ADC_CH5: return (AT91C_BASE_ADC->ADC_CDR5 & 0xFF); break;
337     case AT91C_ADC_CH6: return (AT91C_BASE_ADC->ADC_CDR6 & 0xFF); break;
338     case AT91C_ADC_CH7: return (AT91C_BASE_ADC->ADC_CDR7 & 0xFF); break;
339     default: return 0;
340   }
```

```
341 }
342
343 ////////////////////////////////////////////////////////////////////////////
344 // ADC - odczytanie wyniku konwersji (2 najstarsze bity)
345 ////////////////////////////////////////////////////////////////////////////
346 char ADC_Read_2bit(int channel_mask)
347 {
348   //oczekiwanie na zakonczenie konwersji
349   while( AT91C_BASE_ADC->ADC_SR & channel_mask );
350
351   //odczyt wyniku (2 najstarsze bity)
352   switch (channel_mask)
353   {
354     case AT91C_ADC_CH4: return (AT91C_BASE_ADC->ADC_CDR4 & 0x300) >> 8; break;
355     case AT91C_ADC_CH5: return (AT91C_BASE_ADC->ADC_CDR5 & 0x300) >> 8; break;
356     case AT91C_ADC_CH6: return (AT91C_BASE_ADC->ADC_CDR6 & 0x300) >> 8; break;
357     case AT91C_ADC_CH7: return (AT91C_BASE_ADC->ADC_CDR7 & 0x300) >> 8; break;
358     default: return 0;
359   }
360 }
361
362 ////////////////////////////////////////////////////////////////////////////
363 // TWI - Konfiguracja
364 // twi_clock min 100kHz dla MCK=48MHz, 200KHz dla MCK=96MHz
365 // max 400KHz
366 ////////////////////////////////////////////////////////////////////////////
367 void TWI_Configure(int twi_clock)
368 {
369   float dzielnik;
370   int dzielnik_calkowity;
371
372   //wlaczenie zegara dla twi
373   AT91F_PMC_EnablePeriphClock ( AT91C_BASE_PMC, 1 << AT91C_ID_TWI );
374
375   //reset
376   AT91C_BASE_TWI->TWI_CR = AT91C_TWI_SWRST;
377
378   // Ustawienie zegara z wypelnieniem 50% na czestotliwosc twi_clock
379   twi_clock = twi_clock<<1;
380   dzielnik = (float)MCK / ( (float)twi_clock );
381   dzielnik_calkowity = dzielnik;
382
383   AT91C_BASE_TWI->TWI_CWGR = (AT91C_TWI_CKDIV & (0 << 16)) |
384                              (AT91C_TWI_CHDIV & (dzielnik_calkowity << 8)) |
385                              (AT91C_TWI_CLDIV & (dzielnik_calkowity << 0));
386
387   //tryb master transfer
388   AT91C_BASE_TWI->TWI_CR = AT91C_TWI_MSEN;
389
390   //konfiguracja wyprowadzen PA3, PA4
391   AT91F_PIO_CfgPeriph(AT91C_BASE_PIOA, (AT91C_PA3_TWD | AT91C_PA4_TWCK), 0 );
392
393   //bez wewnetrznego podciagania
394   AT91F_PIO_CfgPullup( AT91C_BASE_PIOA, ~PIO_PA3 ) ;
395   AT91F_PIO_CfgPullup( AT91C_BASE_PIOA, ~PIO_PA4 ) ;
396 }
397
398 ////////////////////////////////////////////////////////////////////////////
399 // TWI - Wysylanie
400 // TWI_Write( adres slave'a, adres, dane )
401 ////////////////////////////////////////////////////////////////////////////
402 char TWI_Write(int SlaveAddr, int IntAddr, char data)
403 {
404   char end_transmission=0, Return=0;
405
406   AT91C_BASE_TWI->TWI_CR = AT91C_TWI_START;
407   AT91C_BASE_TWI->TWI_MMR = AT91C_TWI_IADRSZ_1_BYTE | (SlaveAddr << 16); //adres sleve'a
408   AT91C_BASE_TWI->TWI_IADR = IntAddr; //adres
409   AT91C_BASE_TWI->TWI_THR = data; //dane
410
411   waitms(1);
412
```

```
413   //oczekiwanie na zakonczenie transmisji
414   while (!end_transmission)
415   {
416     if (AT91C_BASE_TWI->TWI_SR & AT91C_TWI_NACK)
417     {
418       //brak potwierdzenie - blad i zakonczenie
419       Return = 0 ;
420       end_transmission=1;
421     }
422     else if (AT91C_BASE_TWI->TWI_SR & AT91C_TWI_TXRDY)
423     {
424       //jest potwierdzenie, ok i zakonczenie
425       Return = 1;
426       end_transmission=1;
427     }
428   }
429
430   AT91C_BASE_TWI->TWI_CR = AT91C_TWI_STOP;
431
432   //oczekiwanie na zakonczenie transmisji
433   while (!(AT91C_BASE_TWI->TWI_SR & AT91C_TWI_TXCOMP));
434
435   return Return;
436 }
```

A.5 utils.c

```
1  /**
2  * Projekt    : Dark Explorer firmware
3  * Plik       : utils.c
4  * Zawartosc  : Procedury obliczeniowe i sterujące wyższego poziomu
5  * Autor      : Paweł Kmak
6  * Data       : 1.08.2009
7  **/
8
9  ////////////////////////////////////////////////////////////////////////////
10 // Globalne zmienne zewnetrzne
11 ////////////////////////////////////////////////////////////////////////////
12 extern char mem[];
13 extern int StackIndex;
14 extern char MaxIndex;
15 extern volatile int xmin, xmax, ymin, ymax;
16 #define PO6030K_DEVICE_ID 0x6E
17
18 ////////////////////////////////////////////////////////////////////////////
19 // Sterowanie kierunkiem obrotu silnikow
20 // silnik = 1 - silniki lewe, 2 - silniki prawe
21 // kierunek = 0 - stop, 1 - przod, 2 - tyl
22 ////////////////////////////////////////////////////////////////////////////
23 inline void Kierunek(int silnik, int kierunek)
24 {
25   if(silnik == 1)
26   {
27     switch (kierunek)
28     {
29       case 0:
30         AT91F_PIO_ClearOutput( AT91C_BASE_PIOA, PIO_PA7 );
31         AT91F_PIO_ClearOutput( AT91C_BASE_PIOA, PIO_PA8 );
32         break;
33
34       case 1:
35         AT91F_PIO_SetOutput( AT91C_BASE_PIOA, PIO_PA7 );
36         AT91F_PIO_ClearOutput( AT91C_BASE_PIOA, PIO_PA8 );
37         break;
38
39       case 2:
40         AT91F_PIO_ClearOutput( AT91C_BASE_PIOA, PIO_PA7 );
41         AT91F_PIO_SetOutput( AT91C_BASE_PIOA, PIO_PA8 );
```

```
42          break;
43        }
44    }
45
46    if(silnik == 2)
47    {
48      switch (kierunek)
49      {
50        case 0:
51          AT91F_PIO_ClearOutput( AT91C_BASE_PIOA, PIO_PA9 );
52          AT91F_PIO_ClearOutput( AT91C_BASE_PIOA, PIO_PA10 );
53          break;
54
55        case 1:
56          AT91F_PIO_SetOutput( AT91C_BASE_PIOA, PIO_PA9 );
57          AT91F_PIO_ClearOutput( AT91C_BASE_PIOA, PIO_PA10 );
58          break;
59
60        case 2:
61          AT91F_PIO_ClearOutput( AT91C_BASE_PIOA, PIO_PA9 );
62          AT91F_PIO_SetOutput( AT91C_BASE_PIOA, PIO_PA10 );
63          break;
64      }
65    }
66  }
67
68  ////////////////////////////////////////////////////////////////////////////
69  // Proste sterowanie silnikami
70  // kierunek 0 - stop, 1 - w lewo, 2 - prosto, 3 - w prawo
71  ////////////////////////////////////////////////////////////////////////////
72  inline void go(char kierunek, char pwm1, char pwm2)
73  {
74    switch (kierunek)
75    {
76      case 0:
77        Kierunek(1, 0);
78        Kierunek(2, 0);
79        PWM_Set(0, pwm1);
80        PWM_Set(1, pwm1);
81        PWM_Set(2, pwm2);
82        PWM_Set(3, pwm2);
83        break;
84
85      case 1:
86        Kierunek(1, 2);
87        Kierunek(2, 1);
88        PWM_Set(0, pwm1);
89        PWM_Set(1, pwm1);
90        PWM_Set(2, pwm2);
91        PWM_Set(3, pwm2);
92        break;
93
94      case 2:
95        Kierunek(1, 1);
96        Kierunek(2, 1);
97        PWM_Set(0, pwm1);
98        PWM_Set(1, pwm1);
99        PWM_Set(2, pwm2);
100       PWM_Set(3, pwm2);
101       break;
102
103     case 3:
104       Kierunek(1, 1);
105       Kierunek(2, 2);
106       PWM_Set(0, pwm1);
107       PWM_Set(1, pwm1);
108       PWM_Set(2, pwm2);
109       PWM_Set(3, pwm2);
110       break;
111   }
112 }
113
```

```
114 ////////////////////////////////////////////////////////////////////////////
115 // Odebranie 8 bitów danych z kamery
116 ////////////////////////////////////////////////////////////////////////////
117 inline char CamRead(void)
118 {
119   register char data=0;
120 
121   data = ( AT91F_PIO_GetInput(AT91C_BASE_PIOA) & (CAM_D2 | CAM_D3 | CAM_D4 | CAM_D5 | CAM_D6 | CAM_D7) ) >> 14;
122   if (AT91F_PIO_GetInput(AT91C_BASE_PIOA) & CAM_D0)  {data += 1;}
123   if (AT91F_PIO_GetInput(AT91C_BASE_PIOA) & CAM_D1) {data += 2;}
124 
125  return data;
126 }
127 
128 ////////////////////////////////////////////////////////////////////////////
129 // Odebranie ramki obrazu z kamery // po optymalizacji //
130 // podzielnik = 2 -> klatka 320 x 200
131 // podzielnik = 4 -> klatka 160 x 100
132 ////////////////////////////////////////////////////////////////////////////
133 void GetFrame( register char PodzielnikX, register char PodzielnikY, char transmit )
134 {
135   register unsigned long int x=0, y=0, wsk=0;
136   register char x2=0, y2=0, CamClockEnable=0;
137 
138   // 1-sza klatka do odrzutu
139   CamClockEnable=1;
140   AT91F_PIO_ClearOutput( AT91C_BASE_PIOA, DIODA2 );
141   while (CamClockEnable)
142   {
143     AT91F_PIO_ClearOutput( AT91C_BASE_PIOA, MCLK );
144     AT91F_PIO_SetOutput( AT91C_BASE_PIOA, MCLK );
145     if ( AT91F_PIO_GetInput(AT91C_BASE_PIOA) & VSYNC ) {CamClockEnable=2;}
146       else { if (CamClockEnable==2) {CamClockEnable=0;}}
147   }
148   AT91F_PIO_SetOutput( AT91C_BASE_PIOA, DIODA2 );
149   CamClockEnable=1;
150   wsk=0;
151 
152   // 2-ga klatka dobra
153   while (CamClockEnable)
154   {
155     // 2 tykniecia zegara (lub 1)
156     if (PodzielnikX < 8)
157     {
158       AT91F_PIO_ClearOutput( AT91C_BASE_PIOA, MCLK );
159       AT91F_PIO_SetOutput( AT91C_BASE_PIOA, MCLK );
160     }
161     AT91F_PIO_ClearOutput( AT91C_BASE_PIOA, MCLK );
162     AT91F_PIO_SetOutput( AT91C_BASE_PIOA, MCLK );
163 
164     // VSYNC = 1 (podczas aktywnej czesci klatki)
165     if ( AT91F_PIO_GetInput(AT91C_BASE_PIOA) & VSYNC )
166     {
167       AT91F_PIO_ClearOutput( AT91C_BASE_PIOA, DIODA1 );
168 
169       // HSYNC = 1 (podczas aktywnej czesci linii)
170       if ( AT91F_PIO_GetInput(AT91C_BASE_PIOA) & HSYNC  )
171       {
172 
173         if (PodzielnikX < 8)
174         {
175           if (x<640 && y<400)
176           {
177             if (x2==0 && y2==0) { mem[wsk] = CamRead(); ++wsk;}
178             ++x; ++x2;
179             if(x2==PodzielnikX) {x2=0;}
180           }
181           if (CamClockEnable==1) { CamClockEnable=2; }
182         } else {
183                 if (x<1280 && y<800)
184                 {
185                   if ((x2<3 || x2==9) && y2==0) { mem[wsk] = CamRead(); ++wsk;}
```

```
186                 ++x; ++x2;
187                 if(x2==PodzielnikX) {x2=0;}
188               }
189               if (CamClockEnable==1) { CamClockEnable=2; }
190             }
191
192       } else {
193               // HSYNC = 0
194               if(x!=0){++y; ++y2;}
195               if(y2==PodzielnikY) y2=0;
196               x=0; x2=0;
197             }
198     } else {
199             // VSYNC = 0
200             x=0; x2=0; y=0; y2=0;
201             if(CamClockEnable > 1){ CamClockEnable=0; }
202             AT91F_PIO_SetOutput( AT91C_BASE_PIOA, DIODA1 );
203           }
204
205     //wyslanie pierwszego bloku danych
206     if(wsk==3200 && CamClockEnable==2 && transmit)
207     {
208       UART0_DMA_Write(&mem[0], wsk);
209       CamClockEnable=3;
210     }
211   } // end while (CamClockEnable)
212 } // end GetFrame()
213
214 ////////////////////////////////////////////////////////////////////////////
215 // Funkcje pomocnicze
216 // - obliczanie pozycji piksela w tablicy
217 // - konwersja do zapisu w trybie 2bity na piksel
218 ////////////////////////////////////////////////////////////////////////////
219
220 long int ToWsk(int x, int y)
221 {
222   if(x<320 && y<200 && x>0 && y>0) { return(x+(y*320)); }
223      else return 0;
224 }
225
226 int GetPx2bit(long int wsk)
227 {
228   long int pos_bajt;
229   int pos_bit;
230
231   pos_bajt = wsk>>2; //dzielenie przez 4
232   pos_bit = wsk%4;
233
234   return ( mem[pos_bajt] >> (6-pos_bit-pos_bit) ) & 0x03;
235 }
236
237 void SetPx2bit(long int wsk, char kolor)
238 {
239   long int pos_bajt;
240   int pos_bit, mask[4] = {192, 48, 12, 3};
241
242   pos_bajt = wsk>>2; //dzielenie przez 4
243   pos_bit = wsk%4;
244
245   mem[pos_bajt] &= ~mask[pos_bit]; //zerowanie odpowiednich bitow
246   mem[pos_bajt] |= ( kolor << (6-pos_bit-pos_bit) ); //zapis wyzerowanych bitow
247 }
248
249 ////////////////////////////////////////////////////////////////////////////
250 // Implementacja stosu na tablicy mem dla SmithFill
251 ////////////////////////////////////////////////////////////////////////////
252 void StackPush(int data)
253 {
254   mem[++StackIndex] = (data>>8) & 0xFF;
255   mem[++StackIndex] = data & 0xFF;
256 }
257
```

```
258 int StackPop()
259 {
260   int val1, val2;
261   val1 = mem[StackIndex--];
262   val2 = mem[StackIndex--];
263   val2 = val2<<8;
264   return val1+val2;
265 }
266
267 ////////////////////////////////////////////////////////////////////////////
268 // SmithFill - indeksacja obszarów spójnych
269 ////////////////////////////////////////////////////////////////////////////
270 int SmithFill(int xstart, int ystart, char ChangeColor, char FloodColor)
271 {
272   int x, y, xst, yst, PixelCounter=0;
273   char up, down, kolor;
274
275   //prostokat obcinania inicjalizacja
276   xmin = 500; xmax = 0;  ymin = 500; ymax = 0;
277
278   //punkt poczatkowy na stos
279   StackPush(xstart);
280   StackPush(ystart);
281
282   //dopoki dane na stosie
283   while (StackIndex>16000)
284   {
285     //pobranie punktu startowego
286     yst = StackPop();
287     xst = StackPop();
288     x = xst;
289     y = yst;
290
291     //znaczniki kierunku
292     up=1; down=1;
293
294     //odczyt koloru
295     kolor = GetPx2bit(ToWsk(x,y));
296
297     //malowanie w lewo
298     while (x>=0 && kolor==ChangeColor)
299     {
300       //w gore
301       kolor = GetPx2bit(ToWsk(x,y+1));
302       if (y<199 && kolor==ChangeColor)
303       {
304         if(up)
305         {
306           StackPush(x);
307           StackPush(y+1);
308           up=0;
309         }
310       }
311       else { up=1; }
312
313       //w dol
314       kolor = GetPx2bit(ToWsk(x,y-1));
315       if (y>0 && kolor==ChangeColor)
316       {
317         if(down)
318         {
319           StackPush(x);
320           StackPush(y-1);
321           down=0;
322         }
323       }
324       else { down=1; }
325
326       //zamalowanie piksela
327       SetPx2bit(ToWsk(x,y), FloodColor);
328       ++PixelCounter;
329
```

```
330       //prostokat obcinania
331       if ( x < xmin ) { xmin = x; } //xmin
332       if ( x > xmax ) { xmax = x; } //xmax
333       if ( y < ymin ) { ymin = y; } //ymin
334       if ( y > ymax ) { ymax = y; } //ymax
335
336       //w lewo
337       --x;
338       kolor = GetPx2bit(ToWsk(x,y));
339     }
340
341     //na poczatek
342     x = xst + 1;
343     y = yst;
344
345     //znaczniki
346     up=1;
347     down=1;
348
349     //odczyt koloru
350     kolor = GetPx2bit(ToWsk(x,y));
351
352     //malowanie w prawo
353     while (x<=319 && kolor==ChangeColor)
354     {
355       //w gore
356       kolor = GetPx2bit(ToWsk(x,y+1));
357       if (y<199 && kolor==ChangeColor)
358       {
359         if(up)
360         {
361           StackPush(x);
362           StackPush(y+1);
363           up=0;
364         }
365       }
366       else { up=1; }
367
368       //w dol
369       kolor = GetPx2bit(ToWsk(x,y-1));
370       if (y>0 && kolor==ChangeColor)
371       {
372         if(down)
373         {
374           StackPush(x);
375           StackPush(y-1);
376           down=0;
377         }
378       }
379       else { down=1; }
380
381       //zamalowanie piksela
382       SetPx2bit(ToWsk(x,y), FloodColor);
383       ++PixelCounter;
384
385       //prostokat obcinania
386       if ( x < xmin ) { xmin = x; } //xmin
387       if ( x > xmax ) { xmax = x; } //xmax
388       if ( y < ymin ) { ymin = y; } //ymin
389       if ( y > ymax ) { ymax = y; } //ymax
390
391       //w prawo
392       ++x;
393       kolor = GetPx2bit(ToWsk(x,y));
394     }
395   }
396
397   return PixelCounter;
398 }
```

A.6 rozpoznawanie.c

```
1  /**
2  * Projekt    : Dark Explorer firmware
3  * Plik       : rozpoznawanie.c
4  * Zawartosc  : Analiza i rozpoznawanie obrazu
5  * Autor      : Paweł Kmak
6  * Data       : 1.08.2009
7  **/
8
9  ////////////////////////////////////////////////////////////////////////////
10 // Globalne zmienne zewnetrzne
11 ////////////////////////////////////////////////////////////////////////////
12 extern float wzorzec1, wzorzec2, wzorzec3, wzorzec4, wzorzec5, wzorzec6, wzorzec7, wzorzec8, wzorzec9;
13 extern int xwzorzec, ywzorzec, WzorzecCount, RozpoznanyX, RozpoznanyY;
14 extern volatile int SerwoAngle;
15 extern char SilnikiEnable;
16 extern char AutoPreview;
17
18 ////////////////////////////////////////////////////////////////////////////
19 // Analiza i rozpoznawanie
20 ////////////////////////////////////////////////////////////////////////////
21 inline int Rozpoznaj()
22 {
23   register long int wsk;
24   register int x, y, i, j;
25   long int srednia, ObszarCounter, pole, mx, my;
26   char srednia_255, ZnalezionoObszar;
27   float odleglosc, tmp, obwod, rzut1, rzut2, rzut3, rzut4, roznica=100000;
28   float wspolczynnik1, wspolczynnik2, wspolczynnik3;
29
30   //wlaczenie diody
31   AT91F_PIO_ClearOutput( AT91C_BASE_PIOA, LED_POWER );
32
33   //odebranie ramki 320x200
34   GetFrame( 2, 2, 0 );
35
36   //wylaczenie diody
37   AT91F_PIO_SetOutput( AT91C_BASE_PIOA, LED_POWER );
38
39   //zbadanie sredniej jasnosci
40   srednia=0;
41   for(wsk=0; wsk<64000; ++wsk )
42   {
43     srednia += mem[wsk];
44   }
45   srednia = srednia / 64000;
46   srednia_255 = srednia;
47
48   //progowanie
49   for(wsk=0; wsk<64000; ++wsk )
50   {
51     if(mem[wsk] > srednia_255) {mem[wsk] = 255;}
52       else {mem[wsk] = 0;}
53   }
54
55   //zapis w postaci 2-bitowej
56   for(wsk=0; wsk<64000; ++wsk )
57   {
58     if(mem[wsk] > 0) { SetPx2bit(wsk, 1); }
59       else { SetPx2bit(wsk, 0); }
60   }
61
62   //skopiowanie podgladu na koniec tablicy mem (autopodgląd)
63   if(AutoPreview)
64   {
65     for(wsk=0; wsk<16000; ++wsk )
66     {
67       mem[wsk+48000] = mem[wsk];
68     }
69     //wyslanie info ze gotowe (synchronizacja)
```

```
70    mem[47992]='p'; mem[47993]=1;
71    UART0_DMA_Write(&mem[47992], 8);//wypelniacz - dane bez sensu
72  }
73
74
75  x=0; y=0; ObszarCounter=0; ZnalezionoObszar=0;
76  while (y<200)
77  {
78    //zamalowywanie obszaru zawierajacego piksel (x, y)
79    if ( GetPx2bit(ToWsk(x,y)) == 0 )
80    {
81      //sprawdzenie czy obszar ma wiecej jak 100 pikseli
82      //jesli tak - zamaina wartosci pikseli obszaru na 2
83      if( SmithFill(x, y, 0, 3) > 100 )
84      {
85        SmithFill(x, y, 3, 2);
86        ZnalezionoObszar=1;
87        ++ObszarCounter;
88      } else {SmithFill(x, y, 3, 1);}
89    }
90
91    //Dla kazdego znalezionego obszaru spójnego analiza i rozpoznawanie
92    if (ZnalezionoObszar)
93    {
94      //wyliczenie pola obszaru
95      //oraz momentow 1szego rzedu (srodek ciezkosci)
96      pole=0;
97      mx=0; my=0;
98      for(i=xmin; i<=xmax; ++i)
99      {
100       for(j=ymin; j<=ymax; ++j)
101       {
102         if ( GetPx2bit(ToWsk(i,j)) == 2 )
103         {
104           ++pole;
105           mx = mx + i;
106           my = my + j;
107         }
108       }
109     }
110
111      mx = mx / pole;
112      my = my / pole;
113
114     //suma odleglosci od srodka ciezkosci
115     //dla Blaira Blissa
116     odleglosc=0;
117     for(i=xmin; i<=xmax; ++i)
118     {
119       for(j=ymin; j<=ymax; ++j)
120       {
121         if( GetPx2bit(ToWsk(i,j)) == 2 )
122         {
123           tmp = sqrt( ((i-mx)*(i-mx)) + ((j-my)*(j-my)) );
124           odleglosc = odleglosc + ( tmp * tmp );
125         }
126       }
127     }
128
129     odleglosc = sqrt( odleglosc * 6.28 );
130     wspolczynnik1=0;
131     if(odleglosc!=0) { wspolczynnik1 = ((float)pole / odleglosc)*2.0; }
132
133     //obwod figury
134     rzut1=0, rzut2=0, rzut3=0, rzut4=0;
135     if(xmin>1 && ymin>1 && xmax<318 && ymax<198)
136     {
137       for(i=xmin-1; i<xmax+1; ++i)
138       {
139         for(j=ymin-1; j<ymax+1; ++j)
140         {
141           //dlugosc rzutu 0stopni
```

```
142             if(GetPx2bit(ToWsk(i,j)) != 2 && GetPx2bit(ToWsk(i+1,j)) == 2)
143             {
144               ++rzut1;
145             }
146
147             //dlugosc rzutu 45stopni
148             if(GetPx2bit(ToWsk(i,j)) != 2 && GetPx2bit(ToWsk(i+1,j-1)) == 2)
149             {
150               ++rzut2;
151             }
152
153             //dlugosc rzutu 90stopni
154             if(GetPx2bit(ToWsk(i,j)) != 2 && GetPx2bit(ToWsk(i,j-1)) == 2)
155             {
156               ++rzut3;
157             }
158
159             //dlugosc rzutu 135stopni
160             if(GetPx2bit(ToWsk(i,j)) != 2 && GetPx2bit(ToWsk(i-1,j-1)) == 2)
161             {
162               ++rzut4;
163             }
164           }
165         }
166       }
167
168       obwod = 0.785 * ( (rzut1 + rzut3) + (0.707*(rzut2 + rzut4)) );
169       wspolczynnik2 = (obwod*obwod) / (float)(12.56*pole);
170       wspolczynnik3=1;
171
172       //jesli brak wzorca, to przypisac
173       if(xwzorzec>0 && ywzorzec>0 && GetPx2bit(ToWsk(xwzorzec,ywzorzec)) == 2)
174       {
175         if (WzorzecCount==0)
176         {
177           wzorzec1 = wspolczynnik1;
178           wzorzec2 = wspolczynnik2;
179           wzorzec3 = wspolczynnik3;
180         }
181
182         if (WzorzecCount==1)
183         {
184           wzorzec4 = wspolczynnik1;
185           wzorzec5 = wspolczynnik2;
186           wzorzec6 = wspolczynnik3;
187         }
188
189         if (WzorzecCount==2)
190         {
191           wzorzec7 = wspolczynnik1;
192           wzorzec8 = wspolczynnik2;
193           wzorzec9 = wspolczynnik3;
194         }
195         xwzorzec=0;
196         ywzorzec=0;
197         ++WzorzecCount;
198
199         if (WzorzecCount==3){ WzorzecCount=0;}
200       }
201
202
203       //sprawdzanie podobienstwa obiektu ze wzorcem (jesli jest wzorzec)
204       if(wzorzec1>0 && wzorzec2>0 && wzorzec3>0)
205       {
206         tmp = sqrt( ((wspolczynnik1-wzorzec1)*(wspolczynnik1-wzorzec1)) + ((wspolczynnik2-wzorzec2)*(wspolczynnik2-wzorzec2))
207         + ((wspolczynnik3-wzorzec3)*(wspolczynnik3-wzorzec3)) );
208         if( tmp < roznica) { roznica = tmp; RozpoznanyX = mx; RozpoznanyY = my;}
209       }
210
211       if(wzorzec4>0 && wzorzec5>0 && wzorzec6>0)
212       {
213         tmp = sqrt( ((wspolczynnik1-wzorzec4)*(wspolczynnik1-wzorzec4)) + ((wspolczynnik2-wzorzec5)*(wspolczynnik2-wzorzec5))
```

```
214         + ((wspolczynnik3-wzorzec6)*(wspolczynnik3-wzorzec6)) );
215         if( tmp < roznica) { roznica = tmp; RozpoznanyX = mx; RozpoznanyY = my;}
216       }
217
218       if(wzorzec7>0 && wzorzec8>0 && wzorzec9>0)
219       {
220         tmp = sqrt( ((wspolczynnik1-wzorzec7)*(wspolczynnik1-wzorzec7)) + ((wspolczynnik2-wzorzec8)*(wspolczynnik2-wzorzec8))
221         + ((wspolczynnik3-wzorzec9)*(wspolczynnik3-wzorzec9)) );
222         if( tmp < roznica) { roznica = tmp; RozpoznanyX = mx; RozpoznanyY = my;}
223       }
224
225       //zapamietanie wymiarow rozpoznanego obiektu w mem
226       if(RozpoznanyX==mx && RozpoznanyY==my)
227       {
228         mem[47994] = xmin>>8;
229         mem[47995] = xmin&0xFF;
230         mem[47996] = xmax>>8;
231         mem[47997] = xmax&0xFF;;
232         mem[47998] = ymin;
233         mem[47999] = ymax;
234       }
235
236       //zmiana wartosci obszaru na 3
237       for(i=xmin; i<=xmax; ++i )
238       {
239         for(j=ymin; j<=ymax; ++j )
240         {
241           if( GetPx2bit(ToWsk(i,j)) == 2) //gdy piksel nalezy do obszaru
242           {
243             SetPx2bit(ToWsk(i,j), 3);
244           }
245         }
246       }
247       ZnalezionoObszar=0;
248     } //end znaleziono obszar
249
250     ++x;
251     if (x>319) {x=0; ++y;}
252
253   } // end while (y<200)
254
255   //Zaznaczenie rozpoznanego obszaru na 2
256   SetPx2bit(ToWsk(RozpoznanyX,RozpoznanyY), 2);
257   SetPx2bit(ToWsk(RozpoznanyX,RozpoznanyY+1), 2);
258   SetPx2bit(ToWsk(RozpoznanyX,RozpoznanyY+2), 2);
259   SetPx2bit(ToWsk(RozpoznanyX,RozpoznanyY+3), 2);
260   SetPx2bit(ToWsk(RozpoznanyX+1,RozpoznanyY), 2);
261   SetPx2bit(ToWsk(RozpoznanyX+1,RozpoznanyY+1), 2);
262   SetPx2bit(ToWsk(RozpoznanyX+1,RozpoznanyY+2), 2);
263   SetPx2bit(ToWsk(RozpoznanyX+1,RozpoznanyY+3), 2);
264   SetPx2bit(ToWsk(RozpoznanyX+2,RozpoznanyY), 2);
265   SetPx2bit(ToWsk(RozpoznanyX+2,RozpoznanyY+1), 2);
266   SetPx2bit(ToWsk(RozpoznanyX+2,RozpoznanyY+2), 2);
267   SetPx2bit(ToWsk(RozpoznanyX+2,RozpoznanyY+3), 2);
268   SetPx2bit(ToWsk(RozpoznanyX+3,RozpoznanyY), 2);
269   SetPx2bit(ToWsk(RozpoznanyX+3,RozpoznanyY+1), 2);
270   SetPx2bit(ToWsk(RozpoznanyX+3,RozpoznanyY+2), 2);
271   SetPx2bit(ToWsk(RozpoznanyX+3,RozpoznanyY+3), 2);
272
273   //wyslanie lokalizacji rozpoznanego obiektu
274   if(AutoPreview)
275   {
276     mem[47992]='p'; mem[47993]=2;
277     UART0_DMA_Write(&mem[47992], 8);
278   }
279
280   //sterowanie silnikami
281   if(SilnikiEnable)
282   {
283     if( RozpoznanyX > 120 && RozpoznanyX < 200)
284     {
285       //obiekt na srodku - jazda na wprost
```

```
286       //srodek = 320/2 = 160
287       go(2,215,225);
288     }
289
290     if( RozpoznanyX >= 200)
291     {
292       //obiekt po lewej
293       //ostry zakret w lewo przez odpowiedni czas
294       go(1,255,255);
295       waitms(250);
296       //potem stop do nastepnej klatki
297       go(0,255,255);
298     }
299
300     if( RozpoznanyX <= 120)
301     {
302       //obiekt po prawej
303       //ostry zakret w prawo przez odpowiedni czas
304       go(3,255,255);
305       waitms(250);
306       //potem stop do nastepnej klatki
307       go(0,255,255);
308     }
309   }
310
311   //sterowanie serwem
312   if( RozpoznanyY > 120) {SerwoAngle = SerwoAngle + 10;}
313   if( RozpoznanyY < 80) {SerwoAngle = SerwoAngle - 10;}
314
315   return ObszarCounter;
316 }
```

Dodatek B

Oprogramowanie nadzorujące

W załączniku tym przedstawiono oprogramowanie nadzorujące pracą robota. Zostało ono napisane w języku C++, przy użyciu środowiska Borland C++ Builder. Ze względu na ograniczoną objętość pracy załącznik ten nie zawiera całości kodów źródłowych, a jedynie przykłady najważniejszych funkcji.

Plik DarkExplorerControll.h zawiera deklaracje funkcji, zmiennych globalnych i obiektów z biblioteki vcl. W pliku DarkExplorerControll.cpp zawarto definicje najważniejszych funkcji.

B.1 DarkExplorerControll.h

```
1  #ifndef DarkExplorerControllH
2  #define DarkExplorerControllH
3
4  #include <Classes.hpp>
5  #include <Controls.hpp>
6  #include <StdCtrls.hpp>
7  #include <Forms.hpp>
8  #include <ExtCtrls.hpp>
9  #include "CGAUGES.h"
10 #include <jpeg.hpp>
11 #include <Menus.hpp>
12
13 class TForm1 : public TForm
14 {
15 __published:    // IDE-managed Components
16   TLabel *status;
17   TImage *Obraz;
18   TScrollBar *SerwoKat;
19   TScrollBar *PrawySilnik;
20   TScrollBar *LewySilnik;
21   TButton *StopButton;
22   TImage *pad;
23   TButton *NapiecieCheck;
24   TLabel *WartoscNapiecia;
25   TButton *Preview3;
26   TTimer *TimeOutTimer;
27   TCGauge *PasekPostepu;
28   TLabel *DownloadStatus;
29   TLabel *DownloadErrors;
30   TLabel *RXBuffer;
31   TCGauge *RXPasek;
32   TCGauge *EnergiaPasek;
```

```
33  TButton *Preview1;
34  TButton *Preview2;
35  TButton *AutoWlacz;
36  TButton *AutoWylacz;
37  TButton *AutoPreviewOnButton;
38  TButton *DiodaWlacz;
39  TButton *DiodaWylacz;
40  TButton *SerwoOn;
41  TButton *SerwoOff;
42  TLabel *LewyStatus;
43  TLabel *PrawyStatus;
44  TLabel *SerwoStatus;
45  TComboBox *WyborPortu;
46  TButton *ComConnect;
47  TButton *ComDisconnect;
48  TButton *Klawiatura;
49  TTimer *AutoTimeOutTimer;
50  TButton *AutoPreviewOffButton;
51  TImage *Image2;
52  TImage *Image3;
53  TCheckBox *AutoPreviewCheck1;
54  TCheckBox *AutoPreviewCheck2;
55  TCheckBox *AutoPreviewCheck3;
56  TButton *AutoPrevievDisable;
57  TButton *PojedynczyPodglad;
58  TImage *Image4;
59  TImage *Image5;
60  TImage *Image6;
61  TImage *Image7;
62  TMainMenu *MainMenu1;
63  TMenuItem *Program1;
64  TMenuItem *Informacje1;
65  TMenuItem *Wylacz1;
66  TTimer *SterowanieTimer;
67  TImage *Image1;
68  void __fastcall FormCreate(TObject *Sender);
69  void __fastcall FormDestroy(TObject *Sender);
70  void __fastcall SerwoKatChange(TObject *Sender);
71  void __fastcall PrawySilnikChange(TObject *Sender);
72  void __fastcall LewySilnikChange(TObject *Sender);
73  void __fastcall StopButtonClick(TObject *Sender);
74  void __fastcall padMouseMove(TObject *Sender, TShiftState Shift,
75    int X, int Y);
76  void __fastcall NapiecieCheckClick(TObject *Sender);
77  void __fastcall Preview3Click(TObject *Sender);
78  void __fastcall TimeOutTimerTimer(TObject *Sender);
79  void __fastcall Preview1Click(TObject *Sender);
80  void __fastcall Preview2Click(TObject *Sender);
81  void __fastcall AutoPreviewOnButtonClick(TObject *Sender);
82  void __fastcall AutoWlaczClick(TObject *Sender);
83  void __fastcall AutoWylaczClick(TObject *Sender);
84  void __fastcall SerwoOnClick(TObject *Sender);
85  void __fastcall SerwoOffClick(TObject *Sender);
86  void __fastcall DiodaWlaczClick(TObject *Sender);
87  void __fastcall DiodaWylaczClick(TObject *Sender);
88  void __fastcall ObrazMouseMove(TObject *Sender, TShiftState Shift,
89    int X, int Y);
90  void __fastcall ObrazClick(TObject *Sender);
91  void __fastcall ComConnectClick(TObject *Sender);
92  void __fastcall ComDisconnectClick(TObject *Sender);
93  void __fastcall KlawiaturaKeyDown(TObject *Sender, WORD &Key,
94    TShiftState Shift);
95  void __fastcall KlawiaturaKeyUp(TObject *Sender, WORD &Key,
96    TShiftState Shift);
97  void __fastcall AutoTimeOutTimerTimer(TObject *Sender);
98  void __fastcall AutoPreviewOffButtonClick(TObject *Sender);
99  void __fastcall Wylacz1Click(TObject *Sender);
100 void __fastcall AutoPreviewCheck1Click(TObject *Sender);
101 void __fastcall AutoPreviewCheck2Click(TObject *Sender);
102 void __fastcall AutoPreviewCheck3Click(TObject *Sender);
103 void __fastcall AutoPrevievDisableClick(TObject *Sender);
104 void __fastcall PojedynczyPodgladClick(TObject *Sender);
```

```
105   void __fastcall padMouseDown(TObject *Sender, TMouseButton Button,
106     TShiftState Shift, int X, int Y);
107   void __fastcall padMouseUp(TObject *Sender, TMouseButton Button,
108     TShiftState Shift, int X, int Y);
109   void __fastcall SterowanieTimerTimer(TObject *Sender);
110   void __fastcall Informacje1Click(TObject *Sender);
111
112 private:  // User declarations
113
114 public:    // User declarations
115   __fastcall TForm1(TComponent* Owner);
116   DCB dcb1;
117   DWORD dwErrors;
118   COMSTAT comStat;
119   HANDLE handlePort_;
120   byte mem[64000]; //pamiec obrazu
121   int TimeOut, AutoTimeOut;
122   DWORD RS_ile;      //ilosc bitow wyslanych
123
124   int AutoPreview, AutoPreview1, AutoPreview2, AutoPreview3;
125   int LButtonDown, RButtonDown;
126   int pwm_left, pwm_right, kierunek_left, kierunek_right, kat, kat255;
127   int xwzorzec, ywzorzec;
128   float PI;
129   Graphics::TBitmap *Bmp = new Graphics::TBitmap;  //obraz
130   Graphics::TBitmap *rysunek = new Graphics::TBitmap;  //sterowanie
131 };
132 extern PACKAGE TForm1 *Form1;
133 #endif
```

B.2 DarkExplorerControll.cpp

```
1  #include <vcl.h>
2  #pragma hdrstop
3
4  #include "DarkExplorerControll.h"
5  #include <Windows.h>
6  #include <commctrl.h>
7  #include <stdio.h>
8  #include <math.h>
9
10 #pragma package(smart_init)
11 #pragma link "CGAUGES"
12 #pragma resource "*.dfm"
13 TForm1 *Form1;
14
15 ////////////////////////////////////////////////////////////////////////////
16 float fi_rad(float x, float y)
17 {
18   if(x>0 && y>=0) {return atan(y/x);}
19   if(x>0 && y<0) {return atan(y/x) + (2.0*PI);}
20   if(x<0) {return atan(y/x) + PI;}
21   if(x==0 && y>0) {return PI/2.0;}
22   if(x==0 && y<0) {return (3.0*PI)/2.0;}
23 }
24 ////////////////////////////////////////////////////////////////////////////
25
26 void GetFrame(char komenda, int MaxCounter)
27 {
28   char data;
29   int counter=0, err=0, cleared=0;
30   PurgeComm(handlePort_, PURGE_RXCLEAR);
31   ClearCommError(handlePort_, &dwErrors, &comStat);
32
33   //wyslanie prosby o pierwszy blok danych (10 linii)
34   data=komenda;
35   WriteFile(handlePort_, &data, 1, &RS_ile, NULL);
36   data=counter;
```

```
37  WriteFile(handlePort_, &data, 1, &RS_ile, NULL);
38
39  while (counter < MaxCounter && err < 10)
40  {
41    //ustawienie TimeOutu oczekiwania (milisekundy)
42    TimeOut=0;
43    if(counter==0) { TimeOutTimer->Interval=1800; }
44      else{ TimeOutTimer->Interval=400; }
45    TimeOutTimer->Enabled=true;
46
47    //oczekiwanie na blok danych
48    while (comStat.cbInQue < 3200 && TimeOut==0)
49    {
50      ClearCommError(handlePort_, &dwErrors, &comStat);
51      RXBuffer->Caption="RXBuffer: " + IntToStr(comStat.cbInQue);
52      RXPasek->Progress=(comStat.cbInQue / 3200.0)*100;
53      Application->ProcessMessages();
54    }
55
56  //zresetowanie timeoutu
57  TimeOutTimer->Enabled=false;
58  TimeOut=0;
59
60  //jesli odebrano poprawna ilosc danych
61  if (comStat.cbInQue == 3200)
62  {
63    //wyslanie prosby o kolejny blok
64    data=komenda;
65    WriteFile(handlePort_, &data, 1, &RS_ile, NULL);
66    data=counter+1;
67    WriteFile(handlePort_, &data, 1, &RS_ile, NULL);
68
69    //odebranie poprzednich danych z bufora
70    ReadFile(handlePort_, &mem[3200*counter], 3200, &RS_ile, NULL);
71    counter++;
72  }
73  else {
74          //gdy blad - czyszczenie złych danych
75          err++;
76          TimeOutTimer->Interval=1000;
77          TimeOutTimer->Enabled=true;
78          ClearCommError(handlePort_, &dwErrors, &comStat);
79          while (comStat.cbInQue > 0 || TimeOut==0)
80          {
81            if(comStat.cbInQue > 0)
82            {
83              PurgeComm(handlePort_, PURGE_RXCLEAR);
84              TimeOutTimer->Enabled=false;
85              TimeOutTimer->Interval=1000;
86              TimeOutTimer->Enabled=true;
87            }
88            ClearCommError(handlePort_, &dwErrors, &comStat);
89            RXBuffer->Caption="RXBuffer: " + IntToStr(comStat.cbInQue);
90            RXPasek->Progress=(comStat.cbInQue / 3200.0)*100;
91            Application->ProcessMessages();
92          }
93        TimeOutTimer->Enabled=false;
94        TimeOut=0;
95
96        //ponowna prosba o ten sam blok danych
97        data=komenda;
98        WriteFile(handlePort_, &data, 1, &RS_ile, NULL);
99        data=counter;
100       WriteFile(handlePort_, &data, 1, &RS_ile, NULL);
101     }
102
103 //statusy
104 PasekPostepu->Progress=((float)counter/MaxCounter)*100;
105 DownloadStatus->Caption="Odbieram pakiet nr: " + IntToStr(counter+1) + " z " + IntToStr(MaxCounter);
106 DownloadErrors->Caption="Błędy: " + IntToStr(err);
107 ClearCommError(handlePort_, &dwErrors, &comStat);
108 RXBuffer->Caption="RXBuffer: " + IntToStr(comStat.cbInQue);
```

```
109   RXPasek->Progress=(comStat.cbInQue / 3200.0)*100;
110   Application->ProcessMessages();
111
112   //jesli nie odebrano poprawnie danych zostanie wyslana ponowna prosba
113   //o przeslanie tego samego bloku
114   } // end   while (counter < MaxCounter && err < 10)
115   if(err<50){ DownloadStatus->Caption="Transfer zakonczony sukcesem"; }
116     else {DownloadStatus->Caption="Błąd - zbyt duże zaklócenia transmisji";}
117   DownloadErrors->Caption="Błędy: " + IntToStr(err);
118 }
119 ////////////////////////////////////////////////////////////////////////////
120
121 int GetPx2bit(int wsk)
122 {
123   int pos_bajt, pos_bit;
124   pos_bajt = wsk>>2;
125   pos_bit = wsk%4;
126
127   return ( mem[pos_bajt] >> (6-pos_bit-pos_bit) ) & 0x03;
128 }
129 ////////////////////////////////////////////////////////////////////////////
130
131 __fastcall TForm1::TForm1(TComponent* Owner)
132         : TForm(Owner)
133 {
134   AutoPreview=0; AutoPreview1=0; AutoPreview2=0; AutoPreview3=0;
135   LButtonDown=0; RButtonDown=0;
136   PI = 3.14159265;
137 }
138 ////////////////////////////////////////////////////////////////////////////
139
140 void __fastcall TForm1::ComConnectClick(TObject *Sender)
141 {
142   char port[15][5]={
143   {"COM1"},{"COM2"},{"COM3"},{"COM4"},{"COM5"},{"COM6"},{"COM7"},{"COM8"},{"COM9"},
144   {"COM10"},{"COM11"},{"COM12"},{"COM13"},{"COM14"},{"COM15"} };
145   handlePort_ = CreateFile(port[WyborPortu->ItemIndex],  // Specify port device: default "COM1"
146   GENERIC_READ | GENERIC_WRITE,       // Specify mode that open device.
147   0,                                  // the devide isn't shared.
148   NULL,                               // the object gets a default security.
149   OPEN_EXISTING,                      // Specify which action to take on file.
150   FILE_ATTRIBUTE_NORMAL,              // default.
151   NULL);                              // default.
152
153   //dcb1.BaudRate = CBR_115200;  // Specify buad rate of communicaiton.
154   //dcb1.BaudRate = 230400;  // Specify buad rate of communicaiton.
155   dcb1.BaudRate = 460800;  // Specify buad rate of communicaiton.
156   dcb1.StopBits = ONESTOPBIT;  // Specify stopbit of communication.
157   dcb1.Parity = NOPARITY;      // Specify parity of communication.
158   dcb1.ByteSize = 8;  // Specify  byte of size of communication.
159
160   if ( SetCommState(handlePort_,&dcb1) == 0)
161   {
162     status->Caption="Status: blad inicjalizacji portu COM" + IntToStr(WyborPortu->ItemIndex+1);
163   }  else {
164             status->Caption="Status: Połączono via COM" + IntToStr(WyborPortu->ItemIndex+1);
165             ComConnect->Enabled=false;
166             ComDisconnect->Enabled=true;
167             WyborPortu->Enabled=false;
168             AllControlsEnable();
169           }
170
171   // instance an object of COMMTIMEOUTS.
172   COMMTIMEOUTS comTimeOut;
173   // Specify time-out between charactor for receiving.
174   comTimeOut.ReadIntervalTimeout = 200;
175   // Specify value that is multiplied
176   // by the requested number of bytes to be read.
177   comTimeOut.ReadTotalTimeoutMultiplier = 100;
178   // Specify value is added to the product of the
179   // ReadTotalTimeoutMultiplier member
180   comTimeOut.ReadTotalTimeoutConstant = 200;
```

```
181   // Specify value that is multiplied
182   // by the requested number of bytes to be sent.
183   comTimeOut.WriteTotalTimeoutMultiplier = 300;
184   // Specify value is added to the product of the
185   // WriteTotalTimeoutMultiplier member
186   comTimeOut.WriteTotalTimeoutConstant = 200;
187   // set the time-out parameter into device control.
188   SetCommTimeouts(handlePort_,&comTimeOut);
189
190   ClearCommError(handlePort_, &dwErrors, &comStat);
191 }
192 ////////////////////////////////////////////////////////////////////////////
193
194 void __fastcall TForm1::ComDisconnectClick(TObject *Sender)
195 {
196  ClearCommError(handlePort_, &dwErrors, &comStat);
197  CloseHandle(handlePort_);
198  status->Caption="Status: Rozłączono - COM" + IntToStr(WyborPortu->ItemIndex+1);
199 }
200 ////////////////////////////////////////////////////////////////////////////
201
202 void __fastcall TForm1::padMouseMove(TObject *Sender, TShiftState Shift,
203       int X, int Y)
204 {
205   Graphics::TBitmap *rysunek = new Graphics::TBitmap;
206   int x, y, promien;
207   float r, fi;
208   char data;
209
210   rysunek->Width = 300;
211   rysunek->Height = 300;
212   rysunek->Canvas->Brush->Color = clBlack;
213   rysunek->Canvas->FillRect(Rect(0,0,300,300));
214   rysunek->Canvas->Pen->Color=clRed;
215   rysunek->Canvas->Ellipse(100,100,200,200);
216
217   x = -1*(150 - X);
218   y = (150 - Y);
219   r = sqrt((x*x) + (y*y));
220   fi = fi_rad(x,y);
221   if (r>50) {r=50;}
222
223   //promien = r;
224   promien = 255.0 * (r/50.0);
225   kat=fi * (180.0 / PI);
226
227   if (!(X==150 && Y==150))
228   {
229     rysunek->Canvas->MoveTo(150,150);
230     rysunek->Canvas->LineTo( ((r*cos(fi))+150), -1*(r*sin(fi))+150);
231   }
232
233   if(kat>=0 && kat <= 45)
234   {
235     kierunek_left = 1; //przod
236     kierunek_right = 2; //tyl
237     pwm_left = promien;
238     pwm_right = promien - ( promien *  ((float)kat/45.0) );
239   }
240
241   if(kat>45 && kat <= 90)
242   {
243     kierunek_left = 1; //przod
244     kierunek_right = 1; //przod
245     pwm_left = promien;
246     pwm_right = promien *  ((float)(kat-45)/45.0);
247   }
248
249   if(kat>90 && kat <= 135)
250   {
251     kierunek_left = 1; //przod
252     kierunek_right = 1; //przod
```

```
253     pwm_left = promien - ( promien *  ((float)(kat-90)/45.0) );
254     pwm_right = promien;
255   }
256
257   if(kat>135 && kat <= 180)
258   {
259     kierunek_left = 2; //tyl
260     kierunek_right = 1; //przod
261     pwm_left = promien *  ((float)(kat-135)/45.0);
262     pwm_right = promien;
263   }
264
265   if(kat>180 && kat <= 225)
266   {
267     kierunek_left = 2; //tyl
268     kierunek_right = 1; //przod
269     pwm_left = promien;
270     pwm_right = promien - ( promien *  ((float)(kat-180)/45.0) );
271   }
272
273   if(kat>225 && kat <= 270)
274   {
275     kierunek_left = 2; //tyl
276     kierunek_right = 2; //tyl
277     pwm_left = promien;
278     pwm_right = promien *  ((float)(kat-225)/45.0);
279   }
280
281   if(kat>270 && kat <= 315)
282   {
283     kierunek_left = 2; //tyl
284     kierunek_right = 2; //tyl
285     pwm_left = promien - ( promien *  ((float)(kat-270)/45.0) );
286     pwm_right = promien;
287   }
288
289   if(kat>315 && kat <= 360)
290   {
291     kierunek_left = 1; //przod
292     kierunek_right = 2; //tyl
293     pwm_left = promien *  ((float)(kat-315)/45.0);
294     pwm_right = promien;
295   }
296
297   if (pwm_left<2) {kierunek_left=0; pwm_left=2;}
298   if (pwm_right<2) {kierunek_right=0; pwm_right=2;}
299
300   pad->Picture->Bitmap->Assign(rysunek);
301 }
302 ////////////////////////////////////////////////////////////////////////////
303
304 void __fastcall TForm1::PrawySilnikChange(TObject *Sender)
305 {
306   //r - silniki prawe kierunek
307   //b - silniki prawe pwm
308   char data;
309
310   if(PrawySilnik->Position >= -2 && PrawySilnik->Position <= 2)
311   {
312     //stop silniki
313     data='r';
314     WriteFile(handlePort_, &data, 1, &RS_ile, NULL);
315     data=0;
316     WriteFile(handlePort_, &data, 1, &RS_ile, NULL);
317     PrawyStatus->Caption="STOP";
318   }
319
320   if(PrawySilnik->Position < -2)
321   {
322     //prawe silniki do przodu
323     data='r';
324     WriteFile(handlePort_, &data, 1, &RS_ile, NULL);
```

```
325     data=1;
326     WriteFile(handlePort_, &data, 1, &RS_ile, NULL);
327
328     data='b';
329     WriteFile(handlePort_, &data, 1, &RS_ile, NULL);
330     data=(PrawySilnik->Position)*-1;
331     WriteFile(handlePort_, &data, 1, &RS_ile, NULL);
332     PrawyStatus->Caption="/|\\ "+ IntToStr(PrawySilnik->Position*-1);
333   }
334
335   if(PrawySilnik->Position > 2)
336   {
337     //prawe silniki do tylu
338     data='r';
339     WriteFile(handlePort_, &data, 1, &RS_ile, NULL);
340     data=2;
341     WriteFile(handlePort_, &data, 1, &RS_ile, NULL);
342
343     data='b';
344     WriteFile(handlePort_, &data, 1, &RS_ile, NULL);
345     data=(PrawySilnik->Position);
346     WriteFile(handlePort_, &data, 1, &RS_ile, NULL);
347     PrawyStatus->Caption="\\|/ "+ IntToStr(PrawySilnik->Position);
348   }
349 }
350 ////////////////////////////////////////////////////////////////////////////
351
352 void __fastcall TForm1::padMouseDown(TObject *Sender, TMouseButton Button,
353       TShiftState Shift, int X, int Y)
354 {
355   if (Button==0){ LButtonDown=1; SterowanieTimer->Enabled=true;}
356   if (Button==1){ RButtonDown=1; SterowanieTimer->Enabled=true;}
357 }
358 ////////////////////////////////////////////////////////////////////////////
359
360 void __fastcall TForm1::padMouseUp(TObject *Sender, TMouseButton Button,
361       TShiftState Shift, int X, int Y)
362 {
363   if (Button==0)
364   {
365     LButtonDown=0;
366     if( Preview1->Enabled==true ){ Preview1->Click(); }
367   }
368   if (Button==1){ RButtonDown=0; }
369 }
370 ////////////////////////////////////////////////////////////////////////////
371
372 void __fastcall TForm1::SterowanieTimerTimer(TObject *Sender)
373 {
374   if(LButtonDown==1)
375   {
376     //silniki lewe
377     if(kierunek_left==0){ LewySilnik->Position=0; }
378     if(kierunek_left==1){ LewySilnik->Position=pwm_left*-1; }
379     if(kierunek_left==2){ LewySilnik->Position=pwm_left; }
380
381     //silniki prawe
382     if(kierunek_right==0){ PrawySilnik->Position=0; }
383     if(kierunek_right==1){ PrawySilnik->Position=pwm_right*-1; }
384     if(kierunek_right==2){ PrawySilnik->Position=pwm_right; }
385   }
386   else {
387         //silniki lewe
388         LewySilnik->Position=0;
389
390         //silniki prawe
391         PrawySilnik->Position=0;
392       }
393
394   if (RButtonDown==1)
395   {
396     kat255 = kat;
```

```
397     if(kat255>180 && kat255<=270) { kat255=180; }
398     if(kat255>270) { kat255=0; }
399     kat255 = ( ((float)kat255)/180.0 ) * 255;
400     SerwoKat->Position=255-kat255;
401   }
402
403   if(LButtonDown==0 && RButtonDown==0)
404   {
405     //autowylaczenie
406     SterowanieTimer->Enabled=false;
407   }
408 }
409 ////////////////////////////////////////////////////////////////////////////
410
411 void __fastcall TForm1::SerwoKatChange(TObject *Sender)
412 {
413   char data;
414   data='s';
415   WriteFile(handlePort_, &data, 1, &RS_ile, NULL);
416   data=255-SerwoKat->Position;
417   WriteFile(handlePort_, &data, 1, &RS_ile, NULL);
418   SerwoStatus->Caption="Kąt: "+IntToStr( (int)(((float)(255-SerwoKat->Position)/255)*180) );
419 }
420 ////////////////////////////////////////////////////////////////////////////
421
422 void __fastcall TForm1::NapiecieCheckClick(TObject *Sender)
423 {
424   char data;
425   byte val[2];
426   int val_rx, energia;
427   float vref=3.3, napiecie;
428
429   // czyszczenie bufora
430   ClearCommError(handlePort_, &dwErrors, &comStat);
431   PurgeComm(handlePort_, PURGE_RXCLEAR);
432
433   data='t';
434   WriteFile(handlePort_, &data, 1, &RS_ile, NULL);
435   data=0;
436   WriteFile(handlePort_, &data, 1, &RS_ile, NULL);
437
438   ReadFile(handlePort_, &val[0], 2, &RS_ile, NULL);
439
440   val_rx = (val[0]<<8) + val[1];
441   napiecie = vref / 1023;
442   napiecie = napiecie * val_rx;
443   napiecie = napiecie * 3.0;
444
445   //przyblizona pozostala energia w akumulatorach w %
446   //napiecie max = 8,5v (4.25 na cele)
447   //napiecie min = 6v (3.0 na cele)
448   //max - min = 2.5v
449   energia = ((napiecie-6.0)/2.5) * 100;
450   EnergiaPasek->Progress=energia;
451   WartoscNapiecia->Caption="Napiecie = " + FormatFloat("0.00", napiecie);
452 }
453 ////////////////////////////////////////////////////////////////////////////
454
455
456 void __fastcall TForm1::Preview3Click(TObject *Sender)
457 {
458   do
459   {
460     //odebranie klatki 320x200 przez bluetootha (20 pakietow)
461     GetFrame('f',20);
462
463     int counter=0;
464     int *linia = new int[400];
465     int *linia2 = new int[400];
466
467     for(y=199; y>=0; y--)
468     {
```

```
469       linia = (int*)Bmp->ScanLine[y*2];
470       linia2 = (int*)Bmp->ScanLine[(y*2)+1];
471         for(x=319; x>=0; x--)
472         {
473           linia[x*2] = RGB(mem[counter],mem[counter],mem[counter]);
474           linia[(x*2)+1] = RGB(mem[counter],mem[counter],mem[counter]);
475           linia2[x*2] = RGB(mem[counter],mem[counter],mem[counter]);
476           linia2[(x*2)+1] = RGB(mem[counter],mem[counter],mem[counter]);
477           counter++;
478           Application->ProcessMessages();
479         }
480     }
481     Obraz->Picture->Bitmap->Assign(Bmp);
482
483   }
484   while(AutoPreview3==1);
485
486   // czyszczenie bufora
487   ClearCommError(handlePort_, &dwErrors, &comStat);
488   PurgeComm(handlePort_, PURGE_RXCLEAR);
489 }
490 ////////////////////////////////////////////////////////////////////////////
491
492 void __fastcall TForm1::AutoPreviewOnButtonClick(TObject *Sender)
493 {
494   int r,g,b;
495   int counter=0;
496   int *linia = new int[400];
497   int *linia2 = new int[400];
498   int xmin, xmax, ymin, ymax;
499   char data;
500
501   AutoPreview=1;
502   AutoTimeOut=0;
503   AutoTimeOutTimer->Interval=5000;
504   AutoTimeOutTimer->Enabled=true;
505
506   //uruchomienie wysylania informacji synchronizacyjnych
507   //po stronie robota
508   data='i';
509   WriteFile(handlePort_, &data, 1, &RS_ile, NULL);
510   data=12;
511   WriteFile(handlePort_, &data, 1, &RS_ile, NULL);
512
513   while(AutoPreview==1 && AutoTimeOut==0)
514   {
515     Application->ProcessMessages();
516     ClearCommError(handlePort_, &dwErrors, &comStat);
517     if (comStat.cbInQue >= 8)
518     {
519       ReadFile(handlePort_, &mem[60000], 8, &RS_ile, NULL);
520       PurgeComm(handlePort_, PURGE_RXCLEAR);
521       if(mem[60000]=='p' && mem[60001]==2)
522       {
523         xmin=mem[60002]; xmin = xmin<<8; xmin+=mem[60003];
524         xmax=mem[60004]; xmax = xmax<<8; xmax+=mem[60005];
525         ymin=mem[60006];
526         ymax=mem[60007];
527         xmin = (319-xmin)*2;
528         xmax = (319-xmax)*2;
529         ymin = (199-ymin)*2;
530         ymax = (199-ymax)*2;
531
532         //rysowanie zaznaczenia rozpoznanego obiektu
533         Bmp->Canvas->Brush->Color = clRed;
534         Bmp->Canvas->FrameRect(Rect(xmax, ymax, xmin, ymin));
535         Bmp->Canvas->FrameRect(Rect(xmax+1, ymax+1, xmin-1, ymin-1));
536         Bmp->Canvas->FrameRect(Rect(xmax+2, ymax+2, xmin-2, ymin-2));
537         Bmp->Canvas->FrameRect(Rect(xmax+3, ymax+3, xmin-3, ymin-3));
538         Obraz->Picture->Bitmap->Assign(Bmp);
539       }
540       if(mem[60000]=='p' && mem[60001]==1)
```

```
541       {
542         //zresetowanie timera
543         AutoTimeOut=0;
544         AutoTimeOutTimer->Enabled=false;
545         AutoTimeOutTimer->Interval=5000;
546         AutoTimeOutTimer->Enabled=true;
547         //odbieranie podgladu
548         GetFrame('i',5);
549
550         counter=0;
551         for(y=199; y>=0; y--)
552         {
553           linia = (int*)Bmp->ScanLine[y*2];
554           linia2 = (int*)Bmp->ScanLine[(y*2)+1];
555
556           for(x=319; x>=0; x--)
557           {
558             r=g=b=150;
559             if (GetPx2bit(counter)==0) {r=g=b=0;}
560             if (GetPx2bit(counter)==1) {r=g=b=255;}
561             if (GetPx2bit(counter)==2) {r=g=b=150;}
562             if (GetPx2bit(counter)==3) {r=200; g=b=0;}
563
564             linia[x*2] = RGB(b,g,r);
565             linia[(x*2)+1] = RGB(b,g,r);
566             linia2[x*2] = RGB(b,g,r);
567             linia2[(x*2)+1] = RGB(b,g,r);
568             counter++;
569             Application->ProcessMessages();
570           }
571         }
572         Obraz->Picture->Bitmap->Assign(Bmp);
573
574       }// end odbieranie podgladu
575     }// end if (comStat.cbInQue >= 8)
576   }// end while
577
578   //wylaczenie wysylania informacji synchronizacyjnych
579   //po stronie robota
580   data='i';
581   WriteFile(handlePort_, &data, 1, &RS_ile, NULL);
582   data=13;
583   WriteFile(handlePort_, &data, 1, &RS_ile, NULL);
584
585   // czyszczenie bufora
586   ClearCommError(handlePort_, &dwErrors, &comStat);
587   PurgeComm(handlePort_, PURGE_RXCLEAR);
588 }
589 ////////////////////////////////////////////////////////////////////////////
590
591 void __fastcall TForm1::ObrazMouseMove(TObject *Sender, TShiftState Shift,
592       int X, int Y)
593 {
594   xwzorzec=319-(X/2);
595   ywzorzec=199-(Y/2);
596 }
597 ////////////////////////////////////////////////////////////////////////////
598
599 void __fastcall TForm1::ObrazClick(TObject *Sender)
600 {
601   char data;
602
603   data='m';
604   WriteFile(handlePort_, &data, 1, &RS_ile, NULL);
605   data=xwzorzec;
606   WriteFile(handlePort_, &data, 1, &RS_ile, NULL);
607
608   data='n';
609   WriteFile(handlePort_, &data, 1, &RS_ile, NULL);
610   data=ywzorzec;
611   WriteFile(handlePort_, &data, 1, &RS_ile, NULL);
612 }
```

```
613 ////////////////////////////////////////////////////////////////////////////
614
615 void __fastcall TForm1::TimeOutTimerTimer(TObject *Sender)
616 {
617   TimeOut=1;
618 }
```

Bibliografia

[1] J. Augustyn, *Projektowanie systemów wbudowanych na przykładzie rodziny SAM7S z rdzeniem ARM7TDMI*, Wydawnictwo IGSMiE PAN, 2007.

[2] R. Tadeusiewicz, P. Korohoda, *Komputerowa analiza i przetwarzanie obrazów*, Wydawnictwo Postępu Telekomunikacji, 1997.

[3] R. Tadeusiewicz, M. Flasiński, *Rozpoznawanie obrazów*, PWN, 1991.

[4] K. Łabanowski, *Programowanie kart graficznych*, Wydawnictwo Lupus, 1994.

[5] Atmel, *Nota aplikacyjna mikrokontrolera AT91SAM7S, http://www.atmel.com/dyn/resources/prod_documents/doc6175.pdf*, 2007.

[6] Pixelplus, *Nota aplikacyjna kamery PO6030k, http://download.maritex.com.pl/pdfs/op/PO6030K-2.pdf*, 2008.

[7] Rayson, *Nota aplikacyjna układu BTM-222, http://download.maritex.com.pl/pdfs/wi/BLU_BTM222.pdf*, 2005.

[8] Motorola, *Nota aplikacyjna układu LM2575*, 1999.

[9] Macroblock, *Nota aplikacyjna układu MBI1801, http://download.maritex.com.pl/pdfs/op/MBI1801.pdf*, 2006.

[10] ST Microelectronics, *Nota aplikacyjna układu L293D*, 1996.

[11] Microchip, *Nota aplikacyjna układu TC1185, TC1015, http://ww1.microchip.com/downloads/en/DeviceDoc/21335e.pdf*, 2002.

[12] Maxim, *Nota aplikacyjna układu MAX3232, http://datasheets.maxim-ic.com/en/ds/MAX3222-MAX3241.pdf*, 2007.

[13] Propox, *Dokumentacja modułu MMsam7s, EVBsam7s, http://www.propox.com/*, 2006.

[14] Atmel, *http://www.atmel.com/products/at91/*, 2009.

[15] International Telecommunication Union, *Specyfikacja standardu ITU-R BT.601 YCbCr, http://inst.eecs.berkeley.edu/~cs150/Documents/ITU601.PDF*, 1994.

[16] W. Zwierychlejski, M. Stołowski, J. Kruk, *Loty kosmiczne, http://astro.zeto.czest.pl/*, 2009.

[17] T. Buratowski, *Teoria robotyki, http://www.robotyka.com/teoria_spis.php*, 2009.

[18] Wortal robotyki, *http://www.asimo.pl/teoria.php*, 2009.

[19] NASA, *Dokumentacja misji MER, http://marsrovers.jpl.nasa.gov/*, 2009.

[20] Mars Society Polska, *http://www.marssociety.pl/*, 2009.

[21] General Atomics Aeronautical, *http://www.ga-asi.com/*, 2009.

[22] ASIMO, *http://asimo.honda.com/*, 2009.

Spis rysunków

Spis tabel