POLITECHNIKA WARSZAWSKA
WYDZIAŁ ELEKTRYCZNY
INSTYTUT STEROWANIA I ELEKTRONIKI PRZEMYSŁOWEJ

PRACA DYPLOMOWA INŻYNIERSKA
na kierunku AUTOMATYKA I ROBOTYKA

Stanisław Busza

Nr albumu: 221101

Rok akad.: 2011/2012
Warszawa, 20 czerwca 2011

WYKORZYSTANIE MECHANIZMU WINDOWS SOCKETS DO STEROWANIA ZESPOŁEM MANIPULATORÓW

Zakres pracy:

1. *Wprowadzenie*
2. *Opis proponowanego rozwiązania komunikacji klient-serwer*
3. *Opis interfejsu programowania aplikacji (API) robota w języku C*
4. *Implementacja biblioteki pozwalającej na zdalną kontrolę manipulatorów*
5. *Opracowanie ćwiczenia laboratoryjnego dla zespołu manipulatorów*
6. *Podsumowanie i wnioski*

(*Podpis i pieczątka Kierownika Zakładu Dydaktycznego*)

Kierujący pracą: dr. inż. Witold Czajewski

Termin złożenia pracy: 27 stycznia 2012

WYKORZYSTANIE MECHANIZMU WINDOWS SOCKETS DO STEROWANIA ZESPOŁEM MANIPULATORÓW

Streszczenie

Celem prezentowanej pracy jest stworzenie programu pozwalającego na zdalną kontrolę przez sieć Ethernet lub Internet, manipulatorów SCORBOT-ER 4u znajdujących się w laboratorium. Dotychczas podczas pracy z manipulatorami przed każdym uruchomieniem programu z indywidualnego komputera następowało bazowanie robota. Po wprowadzeniu programu nadrzędnego, programy indywidualne uruchamiane są jeden raz po którym następuje jedno bazowanie. Teraz zmiany i uruchomienia następują tylko w programie nadrzędnym, co nie powoduje ciągłej konieczności bazowań robotów. Skraca to czas testowania i tworzenia aplikacji dla robota, oraz pozwala pominąć czasochłony proces bazowania. Istniejące do tej pory rozwiązanie nie pozwalało również na kontrolę więcej jak jednego robota z jednego komputera i programu. Program stworzony w ramach tej pracy umożliwia centralne sterowanie dowolną liczbą manipulatorów.

Praca zawiera opis mechanizmów sieciowych, wykorzystanych do sterowania poprzez sieć, np. protokół TCP/IP, mechanizm Windows Sockets, oraz projekt i opis protokołu za pomocą którego komunikują się program nadrzędny z podrzędnymi. Opisana jest dokładna budowa programu podrzędnego i mechanizm automatycznego generatora kodu, który powstał aby usprawnić proces implementacji. Podany jest również sposób instalacji i uruchomienia programu w indywidualnym komputerze jednego z robotów.

W ramach pracy powstała dokumentacja funkcji sterujących robotem. Wzbogaca ona poprzednie prace na temat robotów SCORBOT o te funkcje i parametry, które nie zostały dotychczas opisane i udokumentowane. Ich uzyskanie wymagało wielu godzin spędzonych w laboratorium na eksperymentach z robotami. Funkcje zostały pogrupowane względem wykonywanej czynności. Dla każdej funkcji z osobna są opisane parametry i sposób działania. Każda grupa posiada zbiorczy przykład wykorzystania, który pozwala na lepsze zrozumienie ich działania.

W rozdziale szóstym przedstawiono propozycję ćwiczenia laboratoryjnego wykorzystującego napisany program do zdalnej kontroli robota z udziałem kamery do identyfikacji położenia klocków biorących udział w ćwiczeniu. Program określający położenie klocków w przestrzeni za pomocą obrazu, został dostarczony przez Promotora. Ćwiczenie laboratoryjne jest symulacją odcinka linii produkcyjnej, odpowiedzialnej np: za paletyzację bądź przenoszenie elementów. Do pracy dołączona jest płyta CD z filmem, ukazującym działanie robotów na symulowanej linii produkcyjnej w ramach cwiczenia laboratoryjnego. Na płytce znajdują się również wszystkie ujęte w pracy programy.

APPLICATION OF THE WINDOWS SOCKETS MECHANISM TO CONTROL A SET OF MANIPULATORS

Abstract

The presented thesis aims at development of a software tool allowing for remote control of SCORBOT-ER 4u manipulators, located at the laboratory, via the Ethernet or Internet networks. In the existing solutions, in the course of work with manipulators, the automaton has to be homed every time when the programme is started from an individual computer. After implementation of a superior programme, individual programmes are started once and then the automaton is homed. Now, modifications and starts are performed in the superior programme only, what does result in the unnecessity to continuously home the automata. This results in shortening of the time of testing and development of applications for the automaton; it also allows for elimination of homing, which is a time-consuming process. The solution existing so far did not allow for controlling more than one automaton by one computer and one software tool. Software developed within the frames of the presented thesis allows for central control of an arbitrary number of manipulators.

The thesis contains description of network mechanisms applied for remote control, such as TCP/IP protocol, Windows Sockets mechanism and the design and description of a protocol used for communication between the superior and subordinate programmes. The detailed structure of the subordinate programme, as well as the mechanism of automated code generator, which has been developed in order to improve the implementation process, have been also described. The method of the programme installation and implementation in an individual computer of one of the automata, is also presented.

Documentation of functions applied for controlling the automaton has been developed within the presented thesis. It amends previous works concerning SCORBOT automata with such functions and parameters, which have not been described and documented yet. Development of those functions and parameters required many hours of laboratory experiments with the automata. Those functions have been grouped with respect to operations they perform. Parameters and ways of operations have been described individually for each function. For each group, a common example of operations has been presented, what allows for better understanding of performed operations.

Chapter 6 presents a proposal of a laboratory test, which utilises the developed software tool for remote control of an automaton, with the use of a camera, applied to identify locations of blocks, which are also used for the experiment. The programme used for defining locations of blocks in the space has been delivered by the Supervisor of the thesis. The laboratory test simulates a part of a production line, which is responsible, among others, for palletization or transport of elements.

A CD-rom is amended to the thesis, which contains a film, presenting operations of automata at the simulated production line, performed within the frames of a laboratory test. The CD-rom also contains all software tools discussed in the thesis.

Warszawa, dnia 17 stycznia 2012 roku

Politechnika Warszawska
Wydział Elektryczny

OŚWIADCZENIE

Świadom odpowiedzialności prawnej oświadczam, że niniejsza praca dyplomowa inżynierska pt. WYKORZYSTANIE MECHANIZMU WINDOWS SOCKETS DO STEROWANIA ZESPOŁEM MANIPULATORÓW

- została napisana przeze mnie samodzielnie
- nie narusza niczyich praw autorskich
- nie zawiera treści uzyskanych w sposób niezgodny z obowiązującymi przepisami.

Oświadczam, że przedłożona do obrony praca dyplomowa nie była wcześniej podstawą postępowania związanego z uzyskaniem dyplomu lub tytułu zawodowego w uczelni wyższej.
Jestem świadom, że praca zawiera również rezultaty stanowiące własności intelektualne Politechniki Warszawskiej, które nie mogą być udostępniane innym osobom i instytucjom bez zgody Władz Wydziału Elektrycznego.
Oświadczam ponadto, że niniejsza wersja pracy jest identyczna z załączoną wersją elektroniczną.

Imię i Nazwisko dyplomanta: Stanisław Busza
Podpis dyplomanta: .....

Spis treści

1. Wstęp

W dzisiejszych czasach robotyka jest bardzo szybko rozwijającą się dziedziną. Zastosowanie robotów w fabrykach, na liniach produkcyjnych, oraz w innych gałęziach przemysłu i gospodarki, zapewnia zmniejszenie kosztów i zwiększenie mocy produkcyjnej. Pozwala na zastąpienie ludzkiej pracy powtarzalną pracą maszyn. Pewnym problemem występującym przy organizowaniu stanowisk zrobotyzowanych jest przystosowanie układu sterowania do zmiennych warunków otoczenia. Informacji o położeniu przedmiotów i innych parametrach dostarczają czujniki. Jednym z możliwych sposobów rozwiązania tego problemu jest zastosowanie kamery wizyjnej, dzięki czemu na podstawie zarejestrowanego obrazu można określić położenia przedmiotów.

W laboratorium znajdują się dwa manipulatory SCORBOT-ER 4u. Trudnością występującą przy programowaniu tych robotów jest konieczność czekania (~czterech minut) przed każdym uruchomieniem programu, oraz niemożność kontroli kilku robotów z jednego komputera. Zwłoka czasowa przed każdym uruchomieniem programu wynika z konieczności „wybazowania" robotów. Bazowanie polega na obracaniu ramion aż do osiągnięcia wyłączników krańcowych. Dzieje się to przy małej prędkości i jest czasochłonne. Taki stan rzeczy powoduje wydłużenie czasu tworzenia i testowania aplikacji.. Dotychczas ich sterowanie odbywało się poprzez napisanie dwóch niezależnych programów dla każdego z robotów oddzielnie.

1.1 Cel i układ pracy

Celem tej pracy jest napisanie programu który pozwoli na kontrolę dowolnej liczby robotów z jednego centralnego komputera połączonego siecią z innymi indywidualnymi komputerami. Program nadrzędny skróci czas testowania, ponieważ „bazowanie" będzie odbywać się tylko raz na początku, po uruchomieniu programu sterującego w indywidualnym komputerze robota. Komunikacja sieciowa pomiędzy komputerami odbywa się z wykorzystaniem mechanizmu Windows Sockets. Program napisany jest w języku C.

Należy także stworzyć dokumentację funkcji sterujących robotem i jego ruchem. W poprzednich pracach na ten temat nie wszystkie funkcje i ich parametry zostały dokładnie opisane. Dokumentacja która powstanie będzie zawierać przykłady wykorzystania każdego rodzaju funkcji (ruchu, obsługi chwytaka, wejścia-wyjścia).

W zakres pracy wchodzi również opracowanie koncepcji ćwiczenia laboratoryjnego z wykorzystaniem napisanego programu. Będzie to ćwiczenie w ramach którego studenci

napiszą program sterujący pracą robotów na stanowisku. Program symuluje działanie odcinka linii produkcyjnej z wykorzystaniem dwóch kamer wizyjnych do lokalizacji klocków.

Rozdział 2 zawiera ogólny opis robota oraz jego dane techniczne. Jest tam opisany i przedstawiony na zdjęciu sterownik robota wraz z opisem panelu przedniego oraz zrzut ekranowy z oryginalnego programu do kontroli robota.

Rozdział 3 opisuje pokrótce wykorzystane mechanizmy sieciowe, tzn. protokół TCP/IP i mechanizm Windows Sockets oraz zawiera projekt protokołu komunikacji klient-serwer programu do kontroli robotów.

Rozdział 4 przedstawia aplikację klienta i serwera oraz zawiera opis mechanizmu generatora kodu. Są tam zawarte fragmenty kodu źródłowego wraz z komentarzami. Znajduje się tam również opis instalacji aplikacji na komputerze i jej uruchomienia.

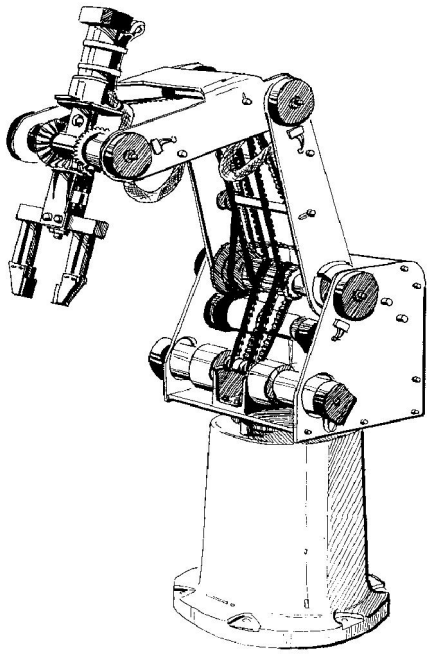
Rozdział 5 stanowi dokumentację zaimplementowanych w programie funkcji służących do kontroli robota. Pogrupowane są one według wykonywanej czynności (funkcje ruchu, kontroli chwytaka, wejścia-wyjścia). Każdy grupa posiada przykład wykorzystania zawartych w niej funkcji.

Rozdział 6 opisuje koncepcje ćwiczenia laboratoryjnego realizowanego na robotach. Zawiera on również opis lokalizacji klocków za pomocą kamer.

Rozdział 7 stanowi podsumowanie i wnioski pracy.

2. Opis robota

Głównym elementem stanowiska jest robot SCORBOT-ER 4U firmy Intelitek, przedstawiony na rys. 1. Składa się on z właściwego manipulatora i sterownika podpiętego do komputera za pomocą kabla USB. Do opcjonalnego wyposażenia wchodzi panel programowania oraz zewnętrzne osie, np. pas transmisyjny.



Rysunek 1: Robot Scorbot-ER 4U firmy Intelitek. Źródło: [9].

Napędy osi robota są wykonane z silników elektrycznych komutatorowych prądu stałego, o wzbudzeniu magnesami trwałymi ze zintegrowanym enkoderem i przekładnią. Posiada on 5 osi oraz chwytak. Robot pozwala na pracę we współrzędnych kartezjańskich oraz wewnętrznych. Nie ma możliwości definiowania własnych układów współrzędnych. Poniżej wymieniono ważniejsze dane techniczne manipulatora [1]:

Liczba stopni swobody: 5 i chwytak
Maksymalna nośność: 2,1 kg
Zakres osi:

- oś 1: 310o
- oś 2: +130 o/ -35 o
- oś 3: +130 o

oś 4: +130 o

oś 5: ±570 o

Maksymalny wysięg: 610 mm

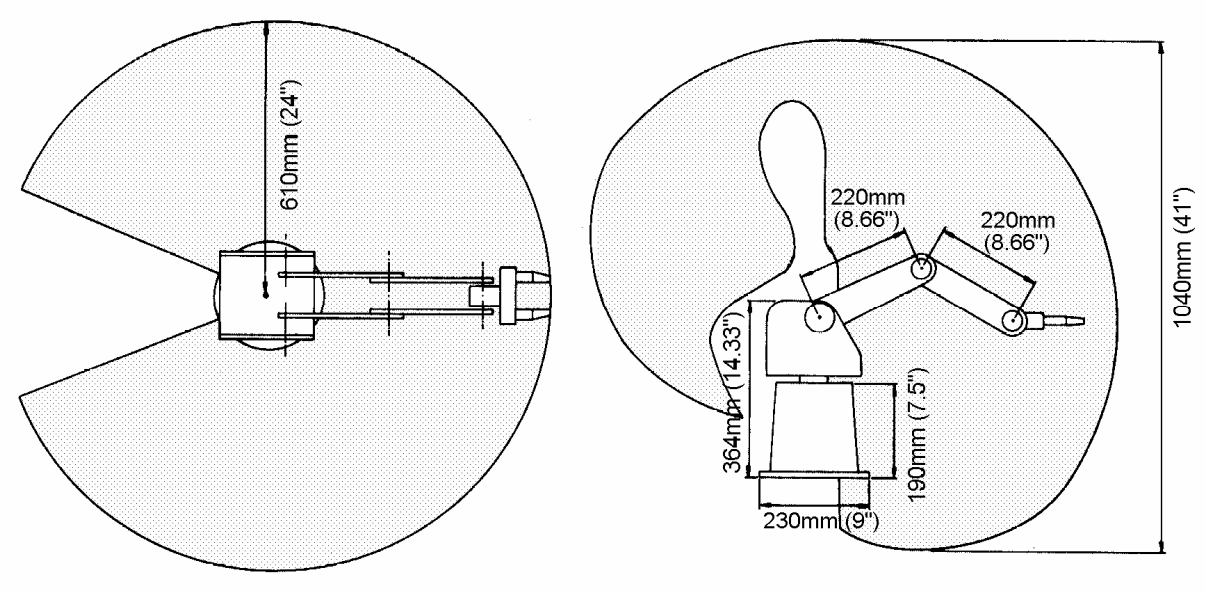
Szybkość: 700 mm/sek.

Powtarzalność: ±0,18 mm

Waga: 10,8 kg

Zasilanie: 230V AC

Dzięki pięciu stopniom swobody można uzyskać dowolne położenie X, Y, Z przy dwóch dowolnych kątach nachylenia chwytaka. Maksymalny wysięg robota wynosi około 60 cm.

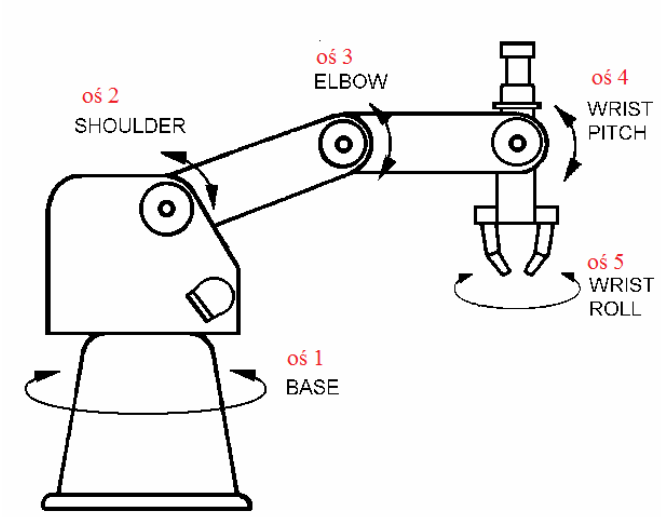


Rysunek 2: Maksymalna przestrzeń robocza manipulatora. Z lewej strony rzut z góry, z prawej strony rzut z boku. Źródło: [9].

Elektronika siłowa i regulatory dla napędu znajdują się w zewnętrznej skrzynce. Posiada ona terminale przyłączeniowe dla wejść i wyjść cyfrowych oraz analogowych i obowiązkowy wyłącznik bezpieczeństwa.

Rysunek 3: Panel przedni sterownika robota. 1 – wyłącznik awaryjny pozwalający zatrzymać robota w każdym momencie, 2 – zaciski wejść i wyjść cyfrowych, 3 – lampki kontrolne sygnalizujące stan wyjść cyfrowych, 4 – lampka sygnalizująca obecność zasilania i zezwolenie na ruch, 5 – zaciski wejść analogowych, 6 – złącza umożliwiające podłączenie zewnętrznych osi. Źródło: własne.

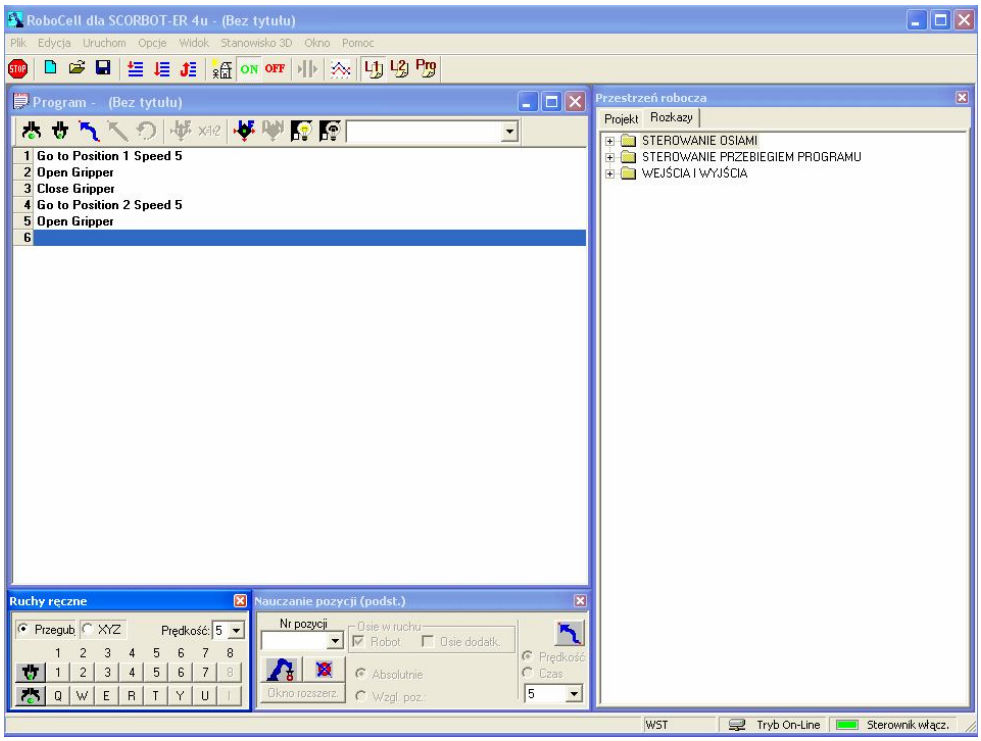
Na panelu tylnym znajdują się dedykowane złącza dla manipulatora, panelu programowania (ang. Teaching pendant) oraz wyłącznik napięcia.



Rysunek 4: numeracja osi robota. Źródło: [9].

Przed rozpoczęciem pracy z robotem należy wybazować wszystkie osie, tzn. odkryć ich fizyczne położenie przez osiągnięcie konkretną osią wyłącznika krańcowego. Dzieje się tak ponieważ w napędach zastosowano enkodery inkrementalne. Dzięki temu osiągamy dość dużą dokładność. Wadą jest konieczność zapamiętywania położenia ramienia robota na czas wyłączenia napięcia zasilania bądź każdorazowe synchronizowanie położenia zapisanego w programie z tym rzeczywistym. Osie należy bazować w konkretnej kolejności ponieważ w przeciwnym wypadku istnieje ryzyko kolizji robota z samym sobą. Ta kolejność to: 1, 2, 4, 3, 0, 5, 7.

Do sterowania robota na komputerze służy oprogramowanie RoboCell dostarczone przez producenta (rys. 4). Pozwala ono na ręczne ustawianie osi, poruszanie się we współrzędnych XYZ, pisanie programów oraz ich wykonywanie, obsługę wejść-wyjść i peryferiów, np. pasa transmisyjnego.



Rysunek 5: Zrzut ekranu z programu RoboCell. W lewym górnym rogu widoczne jest okno programu, na dole znajduje się okno ruchu ręcznego robota i okno programowania pozycji a po prawej okno z rozkazami. Źródło: własne.

Pliki potrzebne do pracy z biblioteką są zawarte na załączonej płytce. Do korzystania z biblioteki są potrzebne 3 pliki i 1 folder które muszą znajdować się razem w jednym katalogu z plikiem wykonywalnym serwera:

usbc.dll – jest to biblioteka dołączana dynamicznie zawierająca implementacje funkcji kontrolujących robota

usbc.ini, er4conf.ini, scbs.ini – pliki konfiguracyjne

folder PAR – zawiera dane dotyczące robota. Zmiany wartości w tym katalogu mogą doprowadzić do jego uszkodzenia

Podczas kompilacji będą nam dodatkowo potrzebne pliki nagłówkowe z definicjami o nazwach usbc.h, usbcdef.h, extern.h, error.h które należy dołączyć na początku pliku źródłowego C\C++ za pomocą dyrektywy preprocesora #include oraz biblioteka LIB usbc.lib.

3. Protokół komunikacji sieciowej

Rozdział ten zawiera szczegółowy opis protokołu komunikacji klient-serwer i mechanizmów sieciowych wykorzystanych w aplikacji. Zaprojektowany protokół stanowi jedną z głównych części pracy.

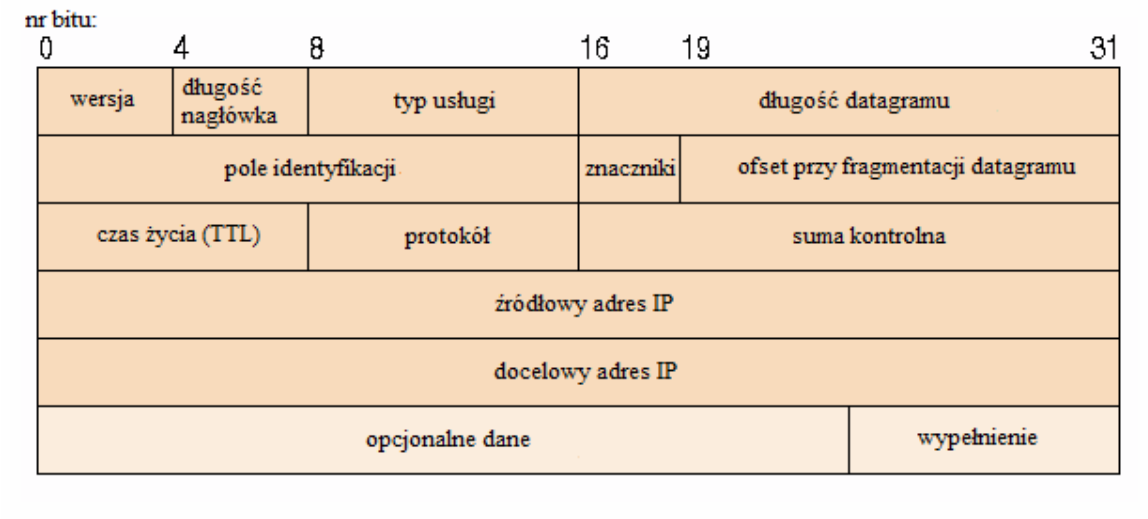
3.1 Mechanizmy sieciowe wykorzystane w projekcie

Do skomunikowania ze sobą aplikacji klienckich i serwera zastosowano protokół TCP/IP. Jest on standardowo wbudowany w system Windows od wersji 95 wzwyż. Tak naprawdę oznaczenie TCP/IP odnosi się do dwóch protokołów działających na różnych poziomach logicznych: protokół IP jest protokołem odpowiedzialnym za adresowanie fizycznych komputerów a protokół TCP pozwala na nawiązywanie połączeń z konkretnymi usługami oraz obsługę kontroli poprawności przesyłanych danych. W niniejszej pracy zdecydowano się na taki wybór z następujących powodów:

1) Popularność wykorzystania TCP/IP.
2) Niezależność od połączenia fizycznego, tzn. może być to technologia ethernet, DSL, itd.
3) Protokół TCP zapewnia niezawodność, tzn.: sprawdza poprawność danych, jeśli są uszkodzone wysyła je jeszcze raz oraz zachowuje ich właściwą kolejność. Jest to jego wyższość na protokołem UDP.

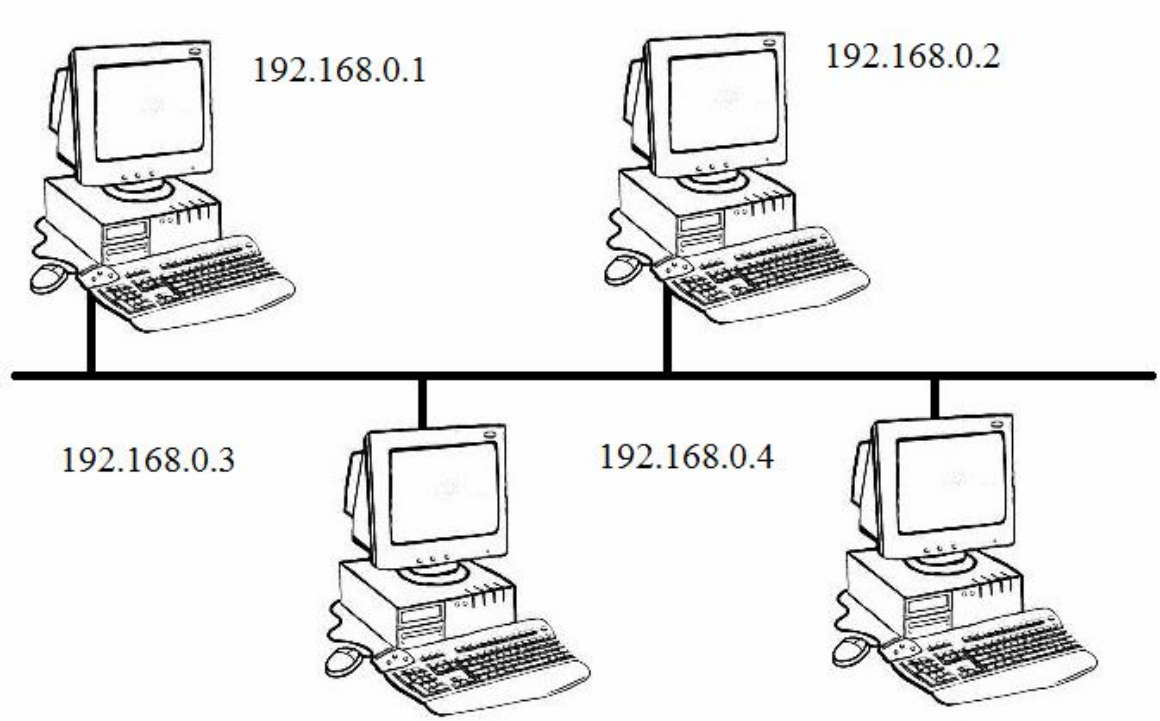
IPv4 (internet protocol version 4):

Jest to protokół sieciowy służący do wymiany danych pomiędzy komputerami. Nadaje on każdemu urządzeniu sieciowemu unikalny 4-bajtowy adres służący do jednoznacznej identyfikacji. Przykładowy zapis takiego adresu wygląda na przykład 192.168.5.100 lub 31.6.135.99. Przesyłanie danych odbywa się za pomocą tzw. ramek lub datagramów, tzn, logicznych ciągów bajtów.



Rysunek 6: Nagłówek IP. Adres docelowy jest drugi od dołu. Źródło: własne tłumaczenie z [2].

Ramka składa się z nagłówka i dołączonych do niego danych. Nagłówek zawiera informacje o adresach docelowym i źródłowym, wielkość przesyłanych danych oraz pewne dodatkowe informacje które interesują routery (urządzenia pośredniczące w wymianie danych pomiędzy komputerami, jeśli nie są w jednej sieci lokalnej). Z naszego punktu widzenia należy zwrócić uwagę tylko na sposób adresowania.



Rysunek 7: przykładowy sposób adresowania komputerów. Jedna sieć zawierająca 4 komputery. Źródło: własne.

Ważną cechą adresów jest maska podsieci. Zapisuje się ją np. 255.255.255.0. Wyznacza ona która część adresu oznacza numer sieci, a która numer komputera. Numer sieci otrzymuje się poprzez wykonanie operacji and na masce i adresie, przykład niżej:

```
        192.168.2.100
AND     255.255.255.0
-------------------------------------------------------------
        192.168.2.0
        po przepisaniu bitowo:
        11000000.10101000.00000010.01100100
AND     11111111.11111111.11111111.00000000
-------------------------------------------------------------
        11000000.10101000.00000010.00000000
```

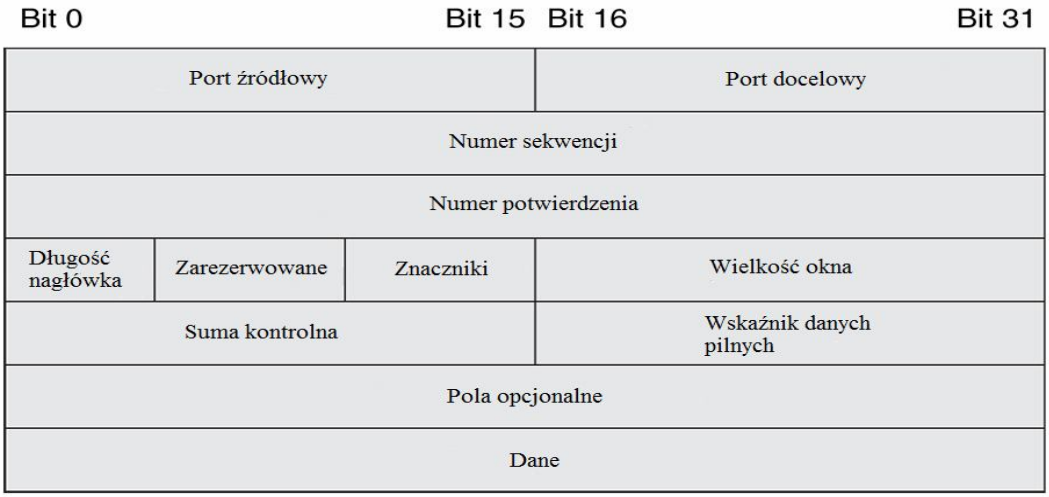
Każdy zakres adresów posiada maskę domyślną. Adresy dzielą się na 3 klasy w zależności od wartości pierwszego bajta. Poniżej dane zaczerpnięte z [3]:

Klasa	Adresy	Maska
A	1-126.X.X.X	255.0.0.0
B	128-191.X.X.X	255.255.0.0
C	192-223.X.X.X	255.255.255.0

gdzie X oznacza dowolną cyfrę

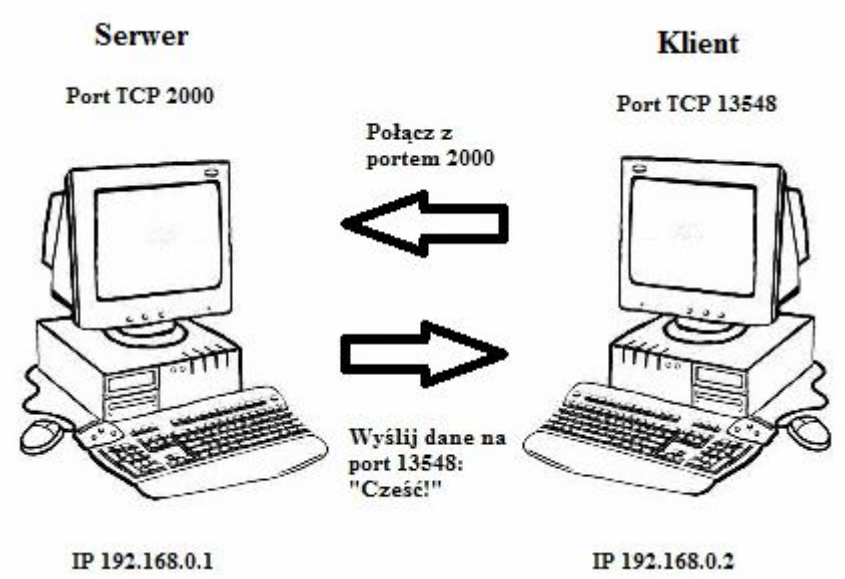
TCP (Transmision control protocol):

Protokół ten służy do nawiązania połączenia z konkretną usługą na zdalnym komputerze. Rozróżnia on stronę klienta która rozpoczyna połączenie i serwera który na nie czeka. Podobnie jak poprzedni protokół posiada on swój nagłówek który jest wstawiany do ramki zaraz za nagłówkiem IP. Poniżej nagłówek TCP pochodzący z [6].



Rysunek 8: Nagłówek protokołu TCP. Źródło: własne tłumaczenie z [6].

Kluczowymi dla nas polami w nagłówku TCP jest port źródłowy (source port) i port docelowy (destination port). To za ich pomocą komputery orientują się które programy łączą się ze sobą. Pola numer sekwencji (sequence number) i numer potwierdzenia (acknowledgment number) pozwalają zachować dobrą kolejność danych w przypadku ich pomieszania.

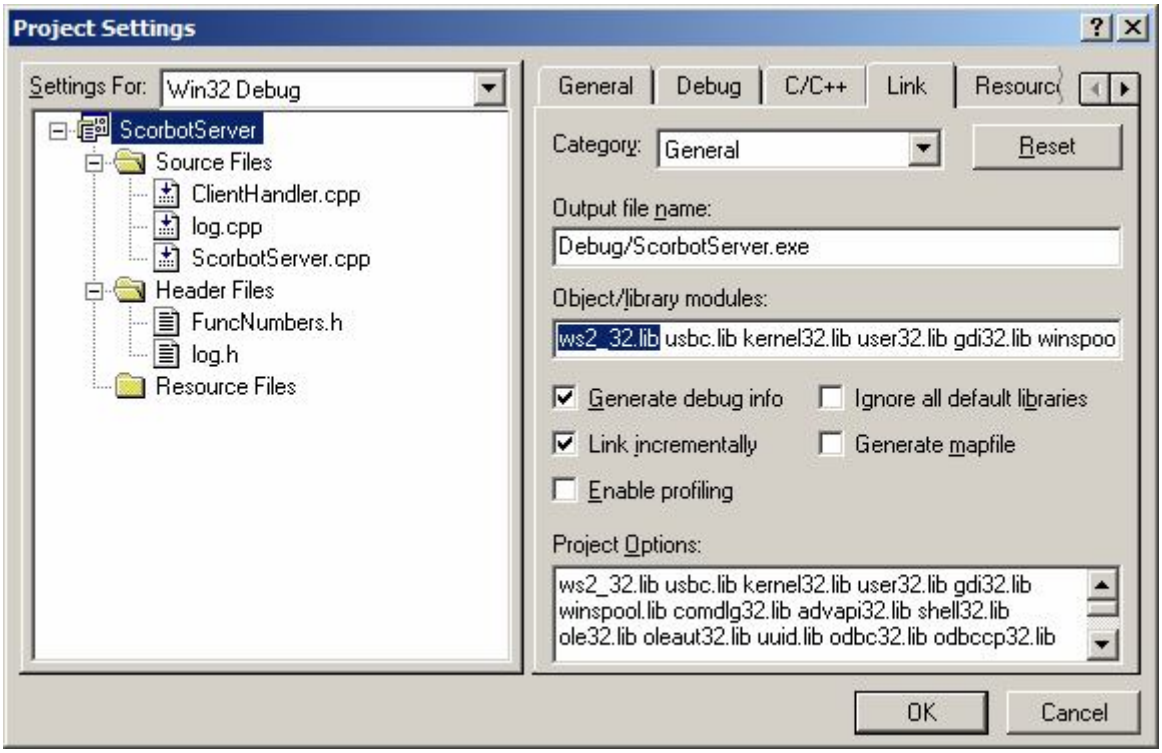


Rysunek 9: Przykładowe połączenie między komputerami. Klient rozpoczyna połączenie, po czym serwer odpowiada mu komunikatem „Cześć!". Źródło: własne.

3.2 Opis API winsock

W poprzednim podrozdziale przedstawiono zasady działania protokołów sieciowych i adresowania bez wzmianki o tym jak z nich skorzystać na konkretnym systemie operacyjnym. Wykorzystanym systemem został MS Windows, najlepiej wersja XP (taki znajduje się w laboratorium). System charakteryzuje się dużą zgodnością wstecz i małymi zmianami w interfejsie programowania. Z naszego punktu widzenia Windows 95 i XP nie różnią się pod tym względem. Mechanizmem zapewniającym łatwy dostęp do funkcji sieciowych w tych systemach jest windows sockets (gniazda windows), w skrócie winsock. Można za jego pomocą stworzyć połączenie a potem je zakończyć oraz wysyłać dane. Dostęp do jego możliwości mamy z poziomu języka C i dowolnego kompilatora. W naszym przypadku jest to MS Visual Studio, zawierający w sobie pełny zestaw bibliotek i plików nagłówkowych więc nie musimy nic instalować dodatkowo.

Aby skorzystać z winsock trzeba w ustawieniach dopisać jego bibliotekę. Poniżej zrzut ekranu ilustrujący postępowanie w przypadku MS Visual Studio 6. Należy wybrać menu Project a potem Settings... aby otworzyć to okno.



Rysunek 10: Ustawienia projektu. Zaznaczony tekst należy dopisać aby korzystać z winsock v2. Źródło: własne.

Na początku pliku źródłowego należy dodać definicję:

```
#include <winsock2.h>
```

która dołącza plik nagłówkowy zawierający definicje funkcji.

Przed wykonaniem jakiejkolwiek funckji należy zainicjować bibliotekę. Poniżej przykład:

```
WSADATA WSAData;

if (WSAStartup(0x2020, &WSAData) != 0)
{
        // tutaj należy zawrzeć obsługę błędów
        //
}
```

Dalsza część rozdziału obejmuje krótki opis odwołań do biblioteki winsock. Służy on do pomocy w zrozumieniu wykorzystanych mechanizmów. Pełen opis w [4].

SOCKET socket (int af, int type, int protocol) – funckja tworzy nowe gniazdo korzystające z zadanego przez parametry protokołu.

int bind(SOCKET s, const struct sockaddr FAR*name, int namelen) – funkcja wiąże gniazdo z konkretnym portem.

int listen(SOCKET s, int backlog) – powoduje przejście gniazda w tryb nasłuchiwania, tzn. oczekiwania na klientów.

SOCKET accept(SOCKET s, struct sockaddr FAR*addr, int FAR*addrlen) – funkcja blokuje wykonanie programu do czasu zgłoszenia się klienta lub wystąpienia błędu.

int send(SOCKET s, const char FAR * buf, int len, int flags) – funkcja wysyła len danych przez gniazdo.

int recv(SOCKET s, char FAR* buf, int len, int flags) – funckja blokuje wykonanie prorgramu do czasu nadejścia danych lub przerwania połączenia, po czym zwraca dane.

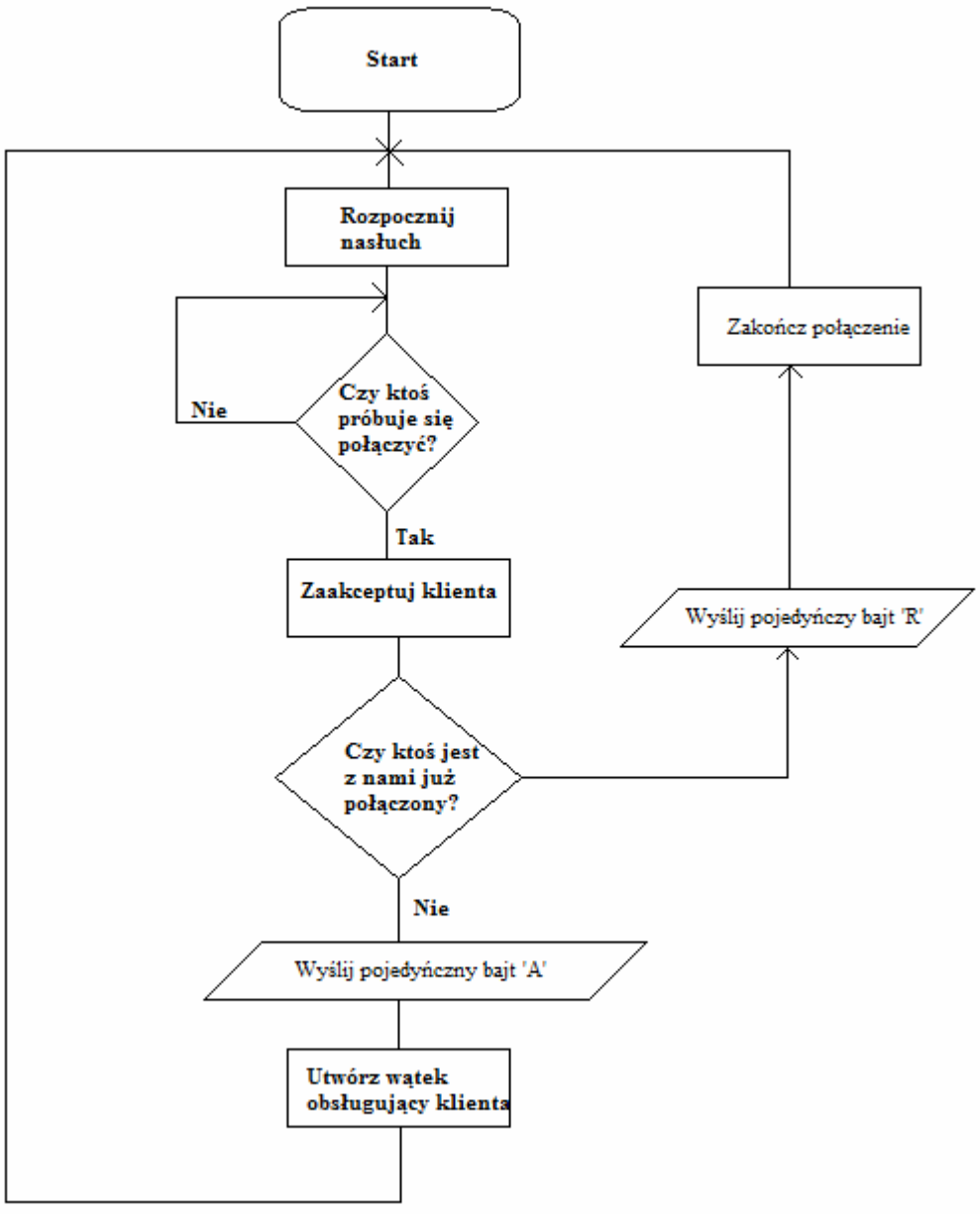
int shutdown(SOCKET s, int how) – funkcja powoduje rozpoczęcie uzgodnionego zamknięcia, i wymianę wszystkich oczekujących w buforach danych.

int closesocket(SOCKET s) – funkcja zamyka gniazdo i zwalnia zasoby.

WSACleanup() – funkcja kończy korzystanie z biblioteki winsock i zwalnia zasoby.

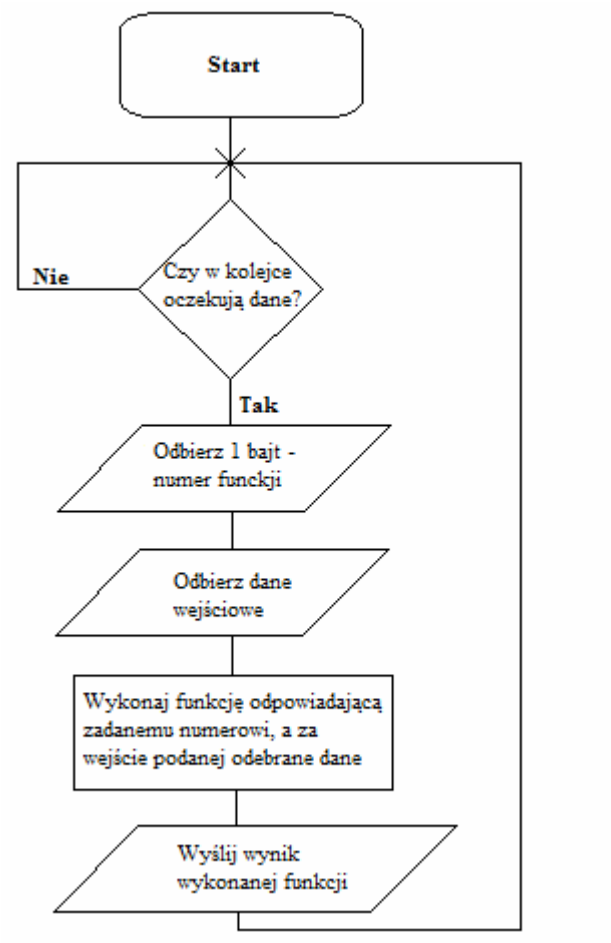
3.3 Opis protokołu

Połączenie rozpoczyna klient, po czym w zależności od tego czy serwer jest wolny następuje jego odrzucenie bądź zaakceptowanie. Serwer może w jednym momencie obsługiwać tylko jednego klienta. Domyślnym portem na którym nasłuchuje serwer jest port 2000.



Rysunek 11: Algorytm obsługi klienta przez serwer. Źródło: własne.

Po zaakceptowaniu klienta i wysłaniu mu literki ‘A’ (skrót od ang. acknowledgment - potwierdzenie) następuje oczekiwanie serwera na wywołanie numer funkcji. Każde wywołanie funkcji blokuje możliwość dalszej pracy aż do zwrócenia wyniku.



Rysunek 12: Algorytm odbioru danych funkcji. Źródło: własne.

Po rozłączeniu się klienta serwer przechodzi w stan oczekiwania na połączenie.

4. Opis aplikacji klienta i serwera

W tym rozdziale została zaprezentowana aplikacja obsługująca robota i jej część biblioteczna od strony technicznej. Wykorzystuje ona mechanizm generatora kodu aby przyspieszyć tworzenie funkcji wysyłających i odbierających dane oraz uniknąć pomyłek.

4.1 Mechanizm automatycznej generacji kodu

Mechanizm automatycznej generacji kodu opiera się na dodatkowym programie napisanym w C który zawiera listę funkcji służących do kontroli robota. Program ten generuje cztery pliki: ClientHandler.cpp, Erlink.cpp, Erlink.h, FuncNumbers.h. Do katalogu serwera należy wgrać ClientHandler.cpp i FuncNumbers.h. Do stworzenia biblioteki posłużą Erlink.cpp oraz Erlink.h i FuncNumbers.h. Plik ClientHandler.cpp należy ręcznie poprawić. Do pracy jest dołączony plik ClientHandler.cpp z naniesionymi poprawkami.

Trzonem aplikacji tworzącej program jest lista funkcji które będą dostępne przez sieć. Cały generator znajduje się w jednym pliku o nazwie main.cpp. Poniżej listing z tego programu zawierający listę.

```
typedef struct              // definicja listy funkcji
{
        char *szName;  // nazwa jaka znajdzie się w pliku
        int iArgs[8];               // lista argumentów, zero na którejś  pozycji // oznacza koniec listy
}FunctionList;

FunctionList FuncList[] =
{
        {"scControl", ARG_CHAR, ARG_BOOL, 0, 0, 0, 0, 0, 0},
        {"scEnterManual", ARG_SHORT, 0, 0, 0, 0, 0, 0, 0},
        {"scMoveManual", ARG_CHAR, ARG_LONG, 0, 0, 0, 0, 0, 0},
        {"scTime", ARG_CHAR, ARG_LONG, 0, 0, 0, 0, 0, 0},
        {"scMoveLinear", ARG_STRING, ARG_SHORT, 0, 0, 0, 0, 0, 0},
        (...)
        {NULL, 0, 0, 0, 0, 0, 0, 0, 0} // NULL -> koniec przetwarzania
};
```

Rekord z polem szName o wartości NULL kończy listę funkcji. Aby dodać jakieś nowe funkcje należy zrobić to przed tym wpisem oraz określić parametry.

Możliwe do przekazania parametry funkcji, zdefiniowane na początku pliku jako wartości liczbowe. Na ich podstawie program wie jak ma wygenerować kod:

ARG_CHAR – parametr jednobajtowy liczbowy
ARG_BOOL – parametr jednobajtowy logiczny (0 lub 1)
ARG_SHORT – parametr dwubajtowy liczbowy
ARG_LONG – parametr czterobajtowy liczbowy ze znakiem
ARG_STRING – łańcuch znaków o maksymalnej długości 20 znaków
ARG_ULONG – parametr czterobajtowy liczbowy bez znaku
ARG_PTR_MIN – powyżej tej wartości wszystkie parametry są przekazywane za pomocą odwołań wskaźnikowych.
ARG_LONG_PTR – wskaźnik do liczby czterobajtowej ze znakiem, fizyczne dane przekazywane są od serwera do klienta
ARG_UCHAR_PTR – wskaźnik do jednobajtowej liczby bez znaku,
fizyczne dane przekazywane są od serwera do klienta
ARG_ROBOTDATA_IN – wskaźnik do informacji o położeniach ramion robota, 8x4 bajty. Dane wędrują w tym przypadku od klienta do serwera
ARG_SHORT_PTR – wskaźnik do liczby dwubajtowej ze znakiem, dane przekazywane od serwera do klienta
ARG_ULONG_PTR – wskaźnik do liczby czterobajtowej bez znaku, dane przekazywane od serwera do klienta
ARG_ROBOTDATA_OUT - wskaźnik do informacji o położeniach ramion robota, 8x4 bajty. Dane przekazywane od serwera do klienta

Przykład definiowania funkcji za pomocą tych parametrów:

```
{"scGetPointInfo", ARG_STRING, ARG_SHORT, ARG_ROBOTDATA_OUT,
ARG_ROBOTDATA_OUT, ARG_ROBOTDATA_OUT, ARG_LONG_PTR, 0, 0}
```

W tym przypadku jest to funkcja o nazwie scGetPointInfo z sześciomia parametrami: pierwszym jest łańcuch znaków, drugim liczba dwubajtowa ze znakiem, trzecim dane wyjściowe (przesyłane do klienta) o stanie robota, czwarty i piąty tak samo a szósty jest przesyłaną do klienta liczbą czterobajtową ze znakiem.

Każdy z generowanych plików podzielony jest na trzy części: nagłówek, cześć tworzoną automatycznie i stopkę. Nagłówek i stopka są stałe i zawarte w jakiejś zmiennej, np:

```
char *ClientHandlerHeader =
"#include <windows.h>\r\n"
"#include \"error.h\"\r\n"
"#include \"usbcdef.h\"\r\n"
"#include \"usbc.h\"\r\n"
"#include \"extern.h\"\r\n"
(...)
```

Nazwa zmiennej odnosi się do konkretnego pliku.

Sekcja generowana zawiera implementacje lub definicje funkcji które wymieniają między sobą dane i służą do kontroli robota. W przypadku pliku Erlink.h są to definicje funkcji:

```
(...)
int scEnterManual(SCON Connection, short Arg0);
int scMoveManual(SCON Connection, char Arg0, long Arg1);
int scTime(SCON Connection, char Arg0, long Arg1);
int scMoveLinear(SCON Connection, char *Arg0, short Arg1);
(...)
```

Plik FuncNumbers.h jest wspólny dla klienta i serwera, ponieważ zawiera on numery funkcji, które są wysyłane przez sieć w celu ich wywołania. Jest to bardzo wygodne, ponieważ można uniknąć pomyłek typu inna numeracja funkcji po stronie klienta, inna po stronie serwera.

```
(...)
#define ERLINK_scDefineVector 6
#define ERLINK_scTeach 7
#define ERLINK_scIsEmergency 8
#define ERLINK_scIsTeachMode 9
#define ERLINK_scIsOnLineOk 10
(...)
```

Plik Erlink.cpp zawiera implementacje funkcji które są używane po stronie klienckiej. Każda funkcja ma ten szablon który wygląda tak (pseudozapis w C):

```
int scNazwaFunkcji(SCON Connection, typ1 Arg0, typ2 Arg1)
{
	char Buffer = ERLINK_ scNazwaFunkcji; // numer fukcji
	int Result; // wynik

	if (send(Connection, &Buffer, 1, 0) != 1) // wysłanie nr.
                                                                    //  funkcji
		return 0;

	/*
		wysyłanie dalszych argumentów
		tutaj program automatycznie generuje funkcje send()
	*/

	// odbiór wyniku wykonania
	if (recv(Connection, (char*)&Result, 4, 0) != 4)
		return 0;

	return Result;
}
```

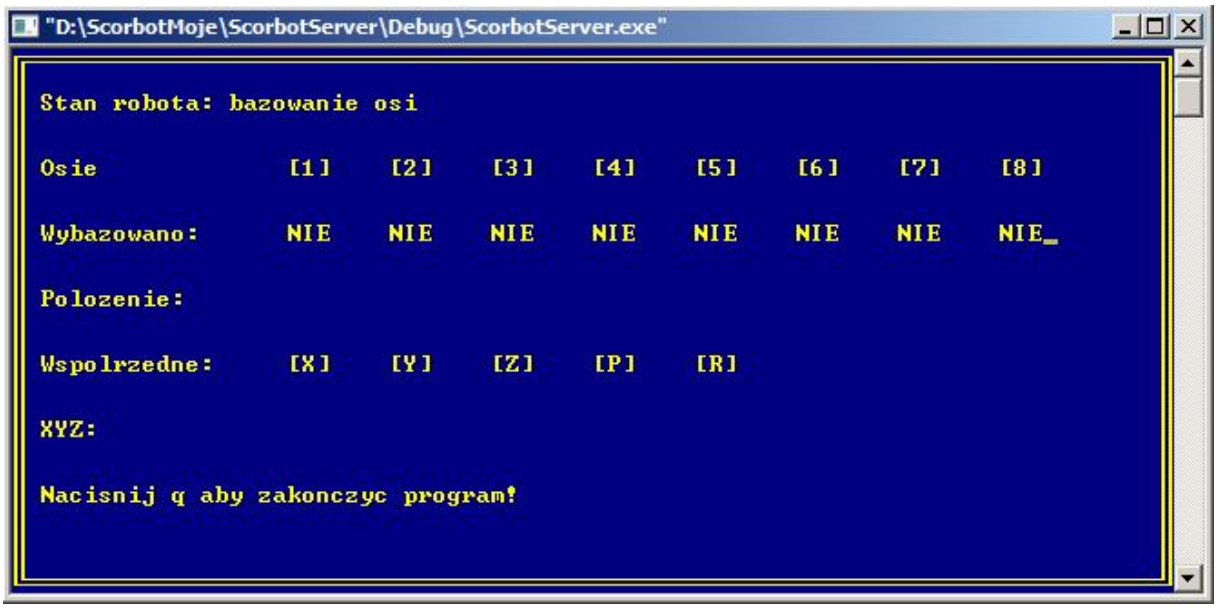
Plik ClientHandler.cpp. działa po stronie serwera i zajmuje się obsługą żądań wywołań funkcji od klientów. Podobnie jak w poprzednim pliku po nagłówku znajduje się sekcja generowana automatycznie która posiada strukturę szablonu. Odbiór danych jest realizowany w jednej dużej procedurze, a wybór konkretnego wywołania odbywa się za pomocą instrukcji switch języka C.

```
// funkcja obsługująca klientów jest osobnym wątkiem
DWORD WINAPI ClientThread(LPVOID lpParam)
{
    SOCKET Sock=(SOCKET)lpParam;
	char Cmd;
    int Result;
	bool bError = false;

	if (send(Sock, "A", 1, 0) != 1) // wysłanie litery ‘A’ na znak
                                                        // akceptacji klienta
	{
		closesocket(Sock);
		ServerBusy = false;
		return 0;
	}

	while(bError != true)
	{
		if (recv(Sock, &Cmd, 1, 0) != 1) // odebranie nr. funkcji
			break;

		switch(Cmd) // wybor procedury obsługi
		{
```

```
		case ERLINK_scControl:
		{
			char Arg0; // deklaracja argumentów
			char Arg1;
			if (recv(Sock, &Arg0, 1, 0) != 1) // odbiór arg. 1
			{
				bError = true;
				break;
			}
			if (recv(Sock, &Arg1, 1, 0) != 1) // odbiór arg. 2
			{
				bError = true;
				break;
			}
			Result = Control(Arg0, Arg1);
			WriteLog("scControl(%d, %d) = %d\n", Arg0, Arg1, Result);
	}
		break;
	(...)
}
```

4.2 Opis aplikacji służącej do kontroli robota

W projekcie zdecydowano się na stworzenie aplikacji serwera bez okienkowego interfejsu użytkownika. Interfejs konsolowy całkowicie wystarcza powodując przy tym bardzo pożądane uproszczenie kodu. Jest to bardzo pozytywna cecha, ponieważ ułatwi rozbudowywanie programu w przyszłości i dodanie do niego nowych funkcji lub możliwości.



Rysunek 13: Widok aplikacji serwera przed wybazowaniem osi. Źródło: własne.

Program jest napisany w języku C. Plikiem w którym rozpoczyna się wykonanie programu po jego uruchomieniu jest ScorbotServer.cpp. Znajduje się tam funkcja main() która jak wiadomo jest punktem wejścia dla wszystkich programów napisanych w tym języku. Otrzymuje ona również argumenty przekazane do programu za pomocą linii komend. W naszym programie zdecydowano się jednak z nich nie korzystać na rzecz konfiguracji przy pomocy pliku ustawienia.ini. Postąpiono tak ponieważ upraszcza to wykorzystanie programu przez niewyszkolonego użytkownika i pozwala na edycję parametrów w Windows-owym notatniku który jest jak wiadomo aplikacją okienkową. W pewnym sensie pozwala to też na zapanowanie nad aplikacją za pomocą zewnętrznego programu lub skryptu, co w przyszłości może się okazać bardzo pomocne. Na następnej stronie zamieszczono listing funkcji main() programu z wyszczególnieniem wykonywanych operacji i ich omówieniem. Kluczowe momenty zaznaczono numerkami, do których odniesienia znajdują się w tekście. Poprzez jej analizę czytelnik będzie w stanie szybciej zrozumieć szczegółowe i techniczne funkcjonowanie aplikacji.

```
int main(int argc, char **argv)
{
	WSADATA WSAData;			// 1
	HANDLE  hThread;
	DWORD   dwThreadId;

	LoadConfig(); // 2

	if ((LogName[0] != 0) && (LogName[0] != '0')) // 3
		OpenLog(LogName);

	WriteLog("Uruchomilem program");
	DrawMainScreen(); // 4
	InitializeScorbot(); // 5

	if (WSAStartup(0x2020, &WSAData) != 0) // 6
	{
      WriteLog("Nie można wczytać biblioteki Winsock! ");
      return 1;
   }
// 7
	hThread = CreateThread(NULL, 0, AcceptingThread,
				(LPVOID)NULL, 0, &dwThreadId);
	if (hThread == NULL)
	{
		WriteLog("Funkcja CreateThread() zakończona błędem: %d ",
				GetLastError());
	}
	else
		CloseHandle(hThread);
// 8
	while(getch() != 'q');
// 9
	Stop('A');
   WSACleanup();
	return 0;
}
```

Przy punkcie 1) zawarte są wszystkie definicje zmiennych i struktur danych potrzebnych w naszym programie. WSADATA jest strukturą zdefiniowaną w pliku winsock2.h i zwrócone w niej informacje nie są w jakikolwiek sposób wykorzystane, podobnie zresztą dzieje się ze dwThreadId. Są wymagane przez wywoływane w ciele main() funkcje i nie sposób tego obejść. hThread po otrzymaniu uchwytu wątku należy zamknąć jak jest to pokazane na przykładzie w [4].

W punkcie 2) jest wywoływana funkcja LoadConfig(). Ma ona za zadanie wczytać konfigurację programu z pliku ustawienia.ini. Narazie są to dwie wartości – numer portu na którym nasłuchuje serwer i nazwa pliku log'a. W przyszłości być może będzie to rozbudowane. Przez wpisanie 0 lub pozostawienie nazwy log'a pustej wyłączamy logowanie do pliku. Nie będą one wtedy wyświetlane wogóle. Dzieje się to w punkcie 3) gdzie otwieramy plik log'a. W punkcie 4) główny interfejs jest tworzony za pomocą znaków

dostępnych w konsoli. Funkcja ta rysuje obramowanie wokół ekranu, narożniki i dzieli ekran na kolumny w których będą wyświetlane dane liczbowe. W punkcie 5) jest wywoływana procedura InitializeScorbot(), która ma za zadanie wczytać bibliotekę USBC obsługującą robota i wybazować wszystkie jego osie. W punkcie 6) są inicjowane usługi sieciowe Winsock. Bez wywołania tej funkcji nie będzie możliwe jakiekolwiek korzystanie z sieci. W punkcie 7) rozpoczynamy nowy wątek oczekujący na zgłoszenia się klientów. Dzięki takiemu rozwiązaniu w punkcie 8) sprawdzane jest czy ktoś nacisnął klawisz ‘q’ chcąc w ten sposób zakończyć program. Dalej w punkcie 9) robot zostaje zatrzymany i kończymy prace z biblioteką winsock po czym kończymy aplikacje i powracamy do systemu operacyjnego.

W programie zdecydowano się na obsługę pliku konfiguracyjnego ustawienia.ini za pomocą natywnych funkcji Windows. Format INI jest wbudowanym w Windows formatem który dawniej służył do konfiguracji samego systemu i aplikacja a obecnie wychodzi z użycia. Jednak zgodnie z polityką Microsoftu funkcje te będą dalej dostępne, ponieważ wszystkie jego produkty zachowują zgodność wstecz. W projekcie użyto dwóch funkcji GetPrivateProfileString() i GetPrivateProfileInt(). Pierwsza z nich pobiera z pliku konfiguracyjnego łańcuch znaków a druga wartość liczbową. Poniżej listing częsci programu odpowiedzialnej za wczytanie konfiguracji.

```
void LoadConfig()
{
	GetCurrentDirectory(MAX_PATH+1, IniFilePath);
	strcat(IniFilePath, "\\ustawienia.ini");

	GetPrivateProfileString("ustawienia", "log", "0", LogName, MAX_PATH+1, IniFilePath);
	port = GetPrivateProfileInt("ustawienia", "port", 2000, IniFilePath);

	CreateConfigFile();
}
```

Na końcu funkcji plik konfiguracyjny jest nadpisywany za pomocą wczytanych wartości, tak aby zawsze miał oryginalny format i znajdowały się w nim dwie wartości. Nawet jeśli nie znaleziono pliku jest on tworzy w wersji defaultowej, tzn. nasłuch na porcie 2000 i brak logowania.

Funkcje służące do kontroli robota są dostępne w bibliotece w formacie LIB. Jest to format statyczny, tzn. że biblioteka ta jest dołączana fizycznie do kompilowanego programu. Aby z niej skorzystać należy się posłużyć plikiem nagłówkowym sclink.h w obrębie którego znajdują się definicje poszczególnych funkcji.

Aby uruchomić aplikacje na swoim komputerze należy:

1. Na komputerze na którym mamy zamiar uruchomić serwer należy zainstalować pełne oprogramowanie dostarczone przez producenta dla robota.
2. Trzeba skompilować ScorbotServer.
3. Należy stworzyć nowy folder o dowolnej nazwie i umieścić w nim plik ScorbotServer.exe. Do tego folderu należy wgrać również pliki dołączone na płycie o nazwach: usbc.dll, usbc.ini, er4conf.ini oraz folder o nazwie Par.
4. Na płycie dołączony program testowy o nazwie ScorbotClient. Należy go skompilować.
5. Należy uruchomić plik ScorbotServer i poczekać na koniec bazowania.
6. Można uruchomić program ScorbotClient, i jeśli robot zacznie wykonywać ruchy wszystko jest ok.

5. Opis funkcji udostępnianych przez bibliotekę kliencką

W tym rozdziale znajduje się dokładny opis funkcji udostępnianych przez bibliotekę kliencką. Większość funkcji jako pierwszy argument przyjmuje SCON Connection, Jest to uchwyt do aktywnego połączenia z serwerem uzyskany za pomocą funkcji scConnect(). Wszystkie funkcje (oprócz scConnect(), scClose(), scShutdown) zwracają wartość 1 gdy wykonanie zakończy się powodzeniem i 0 w przeciwnym wypadku.

5.1 Funkcje inicjujące i pomocnicze

BOOL scInitializeNetwork()

Funkcje tą należy wywołać na początku programu przed korzystaniem z innych funkcji. Zwraca ona TRUE w przypadku powodzenia i FALSE w przeciwnym wypadku.

SCON scConnect(char *HostName, unsigned short usPort)

Funkcja ta łączy się z serwerem którego adres jest zawarty w HostName. Może to być adres w postaci numeru IP, tzn. np. „192.168.0.1” lub nazwa domenowa „komputer1.pw.edu.pl”. W argumencie usPort należy podać numer portu na którym nasłuchuje serwer. Domyślny numer portu to 2000.

void scClose(SCON Connection)

Funkcja kończy połączenie z serwerem i zwalnia zasoby. Uchwyt połączenia podajemy w argumencie Connection.

void scShutdown()

Funkcja ta zwalnia zasoby zajęte przez bibliotekę. Po jej wywołaniu wszystkie funkcje zakończą się błędem. Jeśli chcemy ponownie skorzystać z biblioteki musimy wywołać scInitializeNetwork(). Zwyczajowo tą funkcję wywołuje się na końcu programu.

int scIsEmergency(SCON Connection)

Funkcja zwraca 1 gdy jest wciśnięty przycisk bezpieczeństwa, 0 w przeciwnym wypadku.

int scIsOnLineOk(SCON Connection)

Funckja zwraca 1 gdy istnieje funkcjonujące połączenie z robotem, 0 w przeciwnym wypadku.

Przykład użycia wymienionych funkcji:

```
#include "sclink.h"
#include <stdio.h>

int main()
{
	SCON Con;

	if (scInitializeNetwork() != TRUE)
	{
		printf("Nie mozna uruchomic biblioteki sclink!\n");
		return 0;
	}

	if ((Con = scConnect("localhost", 2000)) == SOCKET_ERROR)
	{
		printf("Nie moge sie polaczyc z serwerem!\n");
		scShutdown();
		return 0;
	}

	if (scIsOnLineOk(Con))
	{
		if (scIsEmergency(Con) == 0)
		{
			// tutaj wykonujemy swój program
		}
	}

	scClose(Con);
	scShutdown();
	return 0;
}
```

5.2 Funkcje operujące na punktach i wektorach

Biblioteka do przechowywania współrzędnych przestrzeni wykorzystuje wektory. Każdy wektor składa się z dowolnie zdefiniowanej wcześniej ilości punktów. Do dyspozycji w programie mamy 4 rodzaje współrzędnych które są przekazywane do procedury w postaci argumentu o typie long:

- ABS_XYZ_A – współrzędne X, Y, Z absolutne
- REL_XYZ_CRNT – współrzędne X, Y, Z relatywne do poprzedniego punktu
- ABS_JOINT – współrzędne wewnętrzne absolutne
- REL_JOINT_CRNT - współrzędne wewnętrzne relatywne do poprzedniego punktu

int scDefineVector(SCON Connection, char Arg0, char *Arg1, short Arg2)

Funkcja definiuje wektor o nazwie Arg1 i ilości punktów składowych Arg2. Parametr Arg0 decyduje dla jakiej grupy osi będą obowiązywały współrzędne. Możliwe wartości to: ‘A’ dla osi robota, ‘B’ dla osi zewnętrznych i ‘&’ dla wszystkich osi. Przed zdefiniowaniem wektora należy wykasować z pamięci jego poprzednie instancje poprzez funkcje scDeletePoint() w sposób scDeletePoint(Connection, „NazwaWekt”, -1).

int scTeach(SCON Connection, char *Arg0, short Arg1, RobotData *Arg2, short Arg3, long Arg4)

Funkcja ta zapisuje do punktu numer Arg1 w wektorze Arg0 współrzędne podane w Arg2 (typ RobotData to tablica long[8]). Arg3 decyduje o ilości uzupełnionych współrzędnych w Arg2, a Arg4 to rodzaj punktu tak jak to opisano we wstępie do tego rozdziału.

int scClearPointsAttributes(SCON Connection)

Funkcja ta kasuje wszystkie wektory i punkty z pamięci sterownika i robota. Po jej wywołaniu trzeba definiować wszystko na nowo.

int scRenameVector(SCON Connection, char *Arg0, char *Arg1)

Funkcja zmienia nazwę wektora z Arg0 na Arg1.

int scDeletePoint(SCON Connection, char *Arg0, short Arg1)

Funkcja kasuje punkt o numerze Arg1 z wektora o numerze Arg0. Numeracja punktów ulega zmianie.

int scHere(SCON Connection, char *Arg0, short Arg1, long Arg2)

Zapisuje bieżącą pozycje robota w punkcie o numerze Arg1 w wektorze Arg0. Punkt jest zapisany we współrzędnych o rodzaju Arg2, tak jak opisano we wstępie.

int scIsPointExist(SCON Connection, char *Arg0, short Arg1)

Funkcja zwraca 1 gdy punkt o numerze Arg1 istnieje w wektorze Arg0, 0 w przeciwnym wypadku.

int scGetPointInfo(SCON Connection, char *Arg0, short Arg1, RobotData *Arg2, RobotData *Arg3, RobotData *Arg4, long *Arg5)

Funkcja zwraca informacje na temat punktu o numerze Arg1 zawartego w wektorze Arg0. Argument Arg2 zwraca informacje o danych enkoderów (wartości inkrementów), Arg3 zwraca informacje o kątach poszczególnych przegubów a Arg4 zwraca informacje o położeniu w współrzędnych X,Y,Z. Argument Arg5 zwraca które pole jest ważne.

int scGetCurrentPosition(SCON Connection, RobotData *Arg0, RobotData *Arg1, RobotData *Arg2)

Funkcja zwraca informacje o aktualnej pozycji robota. Argument Arg0 zwraca informacje o danych enkoderów (wartości inkrementów), Arg1 zwraca informacje o kątach poszczególnych przegubów a Arg2 zwraca informacje o położeniu w współrzędnych X,Y,Z.

Przykład użycia będzie wspólny razem z przykładem dla funkcji ruchu.

5.3 Funkcje kontroli ruchu robota

int scControl(SCON Connection, char Arg0, BOOL Arg1)

Funkcja włączą kontrolę nad serwomechanizmami dla danej grupy osi. Parametr Arg0 może przyjmować wartość ‘A’ dla osi robota, ‘B’ dla osi zewnętrznych i ‘&’ dla wszystkich osi. Gdy chcemy włączyć kontrolę parametr Arg1 ustawiamy na TRUE, FALSE w przeciwnym wypadku.

int scEnterManual(SCON Connection, short Arg0)

Funkcja włącza możliwość ręcznej kontroli ruchu poszczególnych osi za pomocą funkcji scMoveManual(). Parametr Arg0 może przyjmować wartość MANUAL_TYPE_ENC gdy chcemy poruszać poszczególnymi osiami lub MANUAL_TYPE_XYZ gdy chcemy poruszać się wzdłuż poszczególnych współrzędnych.

int scMoveManual(SCON Connection, char Arg0, long Arg1)

Porusza osią o numerze Arg0 (numeracja od 0) z prędkością Arg1. Ruch trwa 7,5 sekundy chyba że zatrzymamy go funkcją scStop().

int scCloseManual(SCON Connection)

Funkcja wyłącza ręczna możliwość kontroli osi za pomocą funkcji scMoveManual().

int scTime(SCON Connection, char Arg0, long Arg1)

Funkcja ustawia prędkość robota tak aby zrealizować następny ruch w czasie zadanym za pomocą Arg1 w milisekundach. Arg0 określa grupę osi dla których ten czas obowiazuje. 0-7 dla konkretnej osi, ‘A’ dla osi robota, ‘B’ dla osi zewnętrznych i ‘&’ dla wszystkich osi. Nie jest polecane użycie tej funkcji ponieważ przy źle dobranym czasie (za krótkim bądź za długim generuje błędy).

int scMoveBelt(SCON Connection, long Arg0, long Arg1)

Funkcja przemieszcza robota po torze jezdnym do położenia Arg0 z prędkością Arg1. Położenie podaje się w inkrementach enkodera.

int scMoveLinear(SCON Connection, char *Arg0, short Arg1)

Funkcja przemieszcza TCP robota po linii prostej od aktualnego punktu do punktu o numerze Arg1 w wektorze Arg0.

int scMoveJoint(SCON Connection, char *Arg0, short Arg1)

Funkcja przemieszcza TCP robota za pomocą ruchu PTP od aktualnego punktu do punktu o numerze Arg1 w wektorze Arg0.

int scMoveCircularVect(SCON Connection, char *Arg0, short Arg1, short Arg2)

Funkcja przemieszcza TCP robota po łuku. Łuk wyznaczają trzy punkty. Pierwszy z nich to punkt w którym aktualnie znajduje się ramię robota a dwa pozostałe to punkty o numerach Arg1 i Arg2 w wektorze Arg0. Punkt o numerze Arg2 to punkt końcowy.

int scSpeed(SCON Connection, char Arg0, long Arg1)

Funkcja ustawia w procentach prędkość robota dla ruchu liniowego. Arg1 to prędkość a Arg0 może przyjmować wartości 0-7 dla konkretnej osi, ‘A’ dla osi robota, ‘B’ dla osi zewnętrznych i ‘&’ dla wszystkich osi.

int scSpeedLin(SCON Connection, char Arg0, long Arg1)

Funkcja ustawia w procentach prędkość robota. Arg1 to prędkość a Arg0 może przyjmować wartości 0-7 dla konkretnej osi, ‘A’ dla osi robota, ‘B’ dla osi zewnętrznych i ‘&’ dla wszystkich osi.

int scSetJoint(SCON Connection, RobotData *Arg0)

Ustawienie poszczególnych osi w zadany kąt.

int scStop(SCON Connection, char Arg0)

Zatrzymanie ruchu dla grupy osi określonej w Arg0. 0-7 dla konkretnej osi, ‘A’ dla osi robota, ‘B’ dla osi zewnętrznych i ‘&’ dla wszystkich osi.

int scVelocity(SCON Connection, char Arg0, short Arg1)

Ustawienie prędkości bezwględnej z jaką ma być wykonywany ruch.

int scGetMotionStatus(SCON Connection)

Zwraca 1 gdy trwa ruch, 0 w przeciwnym wypadku.

int scWaitUntilMotionEnd(SCON Connection)

Funkcja zatrzymuje wykonanie programu do momentu kiedy skończy się ruch robota.

Przykład użycia funkcji ruchu i funkcji operacji na wektorach:

```
#include "sclink.h"
#include <stdio.h>

int main()
{
	SCON Con;
	RobotData CoorArray[3] =
	{{431140, 286870, 206130, -63550, 0, 0, 0, 0},
	 {323660, 215350, 453310, -63550, 0, 0, 0, 0},
	 {166450, -351320, 453310, -63550, 0, 0, 0, 0}};

	if (scInitializeNetwork() != TRUE)
	{
		printf("Nie mozna uruchomic biblioteki sclink!\n");
		return 0;
	}

	if ((Con = scConnect("localhost", 2000)) == SOCKET_ERROR)
	{
		printf("Nie moge sie polaczyc z serwerem!\n");
		scShutdown();
		return 0;
	}

	// usuwamy z pamięci wszystkie punkty
	scClearPointsAttributes(Con);

	// wlaczamu kontrole robota i wykonujemy ruch wzdluz osi x przez 1s z
	// predkoscia 100mm/s
	scControl(Con, '&', TRUE);
	scEnterManual(Con, MANUAL_TYPE_XYZ);
	scMoveManual(Con, 0, 100);
	Sleep(1000);
	scStop(Con, 0);
	scCloseManual(Con);

	// definiujemy wektor "wektorVS" i zapisujemy w nim biezaca pozycje
	scDefineVector(Con, 'A', "wektorVS", 6);
	scHere(Con, "wektorVS", 1, ABS_XYZ_A);

	// definiujemy 3 swoje pozycje
	scTeach(Con, "wektorVS", 2, &CoorArray[0], 5, ABS_XYZ_A);
	scTeach(Con, "wektorVS", 3, &CoorArray[1], 5, ABS_XYZ_A);
	scTeach(Con, "wektorVS", 4, &CoorArray[2], 5, ABS_XYZ_A);

	// zmieniamy nazwe wektora i kasujemy 5 punkt
```

```
        scRenameVector(Con, "wektorVS", "wektor1");
        scDeletePoint(Con, "wektor1", 5);

        // ruch 100 % predkosci max liniowo
        scSpeedLin(Con, 'A', 100);
        scMoveLinear(Con, "wektor1", 1);
        scWaitUntilMotionEnd(Con);

        // ruch 100% predkosci max PTP
        scSpeed(Con, 'A', 100);
        scMoveJoint(Con, "wektor1", 2);
        scWaitUntilMotionEnd(Con);

        // ruch trwajacy 3s
        scTime(Con, 'A', 3000);
        scMoveCircularVect(Con, "wektor1", 4, 1);
        scWaitUntilMotionEnd(Con);

        scClose(Con);
        scShutdown();
        return 0;
}
```

5.4 Funkcje kontroli chwytaka robota

Na zakończenie operacji związanych z chwytakiem należy oczekiwać za pomocą instrukcji Sleep(), w przeciwnym wypadku pojawią się błędy.

int scOpenGripper(SCON Connection)

Funkcja otwiera chwytak.

int scCloseGripper(SCON Connection)

Funkcja zamyka chwytak.

int scJawMetric(SCON Connection, short Arg0)

Funkcja rozsuwa chwytak na Arg0 milimetrów.

int scGetJaw(SCON Connection, short *Arg0, short *Arg1)

Funkcja pobiera szerokość na jaką jest rozsunięty chwytak. Arg0 zwraca rozsunięcie w procentach rozsunięcia maksymalnego a Arg1 w milimetrach.

Przykład użycia wymienionych funkcji:

```
#include "sclink.h"
#include <stdio.h>

int main()
{
	SCON Con;
	short JawPercent, JawMM;
	RobotData CoorArray =
	{431140, 286870, 206130, -63550, 0, 0, 0, 0};

	if (scInitializeNetwork() != TRUE)
	{
		printf("Nie mozna uruchomic biblioteki sclink!\n");
		return 0;
	}

	if ((Con = scConnect("localhost", 2000)) == SOCKET_ERROR)
	{
		printf("Nie moge sie polaczyc z serwerem!\n");
		scShutdown();
		return 0;
	}

	scDefineVector(Con, 'A', "wektorVS", 1);
	scTeach(Con, "wektorVS", 1, &CoorArray, 5, ABS_XYZ_A);

	// otwarcie chwytaka
	scOpenGripper(Con);
	Sleep(3000);

	scMoveLinear(Con, "wektorVS", 1);
	scWaitUntilMotionEnd(Con);


	// ustawienie 10mm rozwarcia chwytaka
	scJawMetric(Con, 10);
	Sleep(3000);

	// pobranie danych chwytaka i ich wyświetlenie
	// w naszym przypadku to wielkosc przedmiotu
	// na jakim zacisnal sie chwytak
	scGetJaw(Con, &JawPercent, &JawMM);
	printf("%d %%, %d mm\n", JawPercent, JawMM);

	scClose(Con);
	scShutdown();
	return 0;
}
```

5.5 Obsługa wejść-wyjść cyfrowych i analogowych

int scGetDigitalInputs(SCON Connection, unsigned long *Arg0)

Funkcja zwraca a Arg0 maskę bitową świadczącą o stanie wejść cyfrowych. Najmłodszy bit odpowiada wejściu o najniższym numerze.

int scGetDigitalOutputs(SCON Connection, unsigned long *Arg0)

Funkcja zwraca a Arg0 maskę bitową świadczącą o stanie wyjść cyfrowych. Najmłodszy bit odpowiada wejściu o najniższym numerze.

int scSetDigitalOutput(SCON Connection, short Arg0, BOOL Arg1, BOOL Arg2)

Funkcja ustawia wyjście cyfrowe o numerze Arg0 na wartość Arg1. Parametr Arg2 mówi o tym czy wartość ma być ustawiona natychmiast. Zalecana wartość to TRUE (ustawienie natychmiast).

int scEnableDigitalInput(SCON Connection, short Arg0)

Funkcja włącza wejście cyfrowe o numerze Arg0.

int scDisableDigitalInput(SCON Connection, short Arg0)

Funkcja wyłącza wejście cyfrowe o numerze Arg0.

int scGetDigitalInputEnabledStatus(SCON Connection, unsigned long *Arg0)

Funkcja pobiera status wejść cyfrowych do bitmaski Arg0.

int scGetAnalogInput(SCON Connection, short Arg0, unsigned char *Arg1)

Funkcja pobiera stan wejścia analogowego o numerze Arg0 do zmiennej Arg1.

int scSetAnalogOutput(SCON Connection, short Arg0, char Arg1)

Funkcja ustawia wyjście analogowe o numerze Arg0 na wartość Arg1.

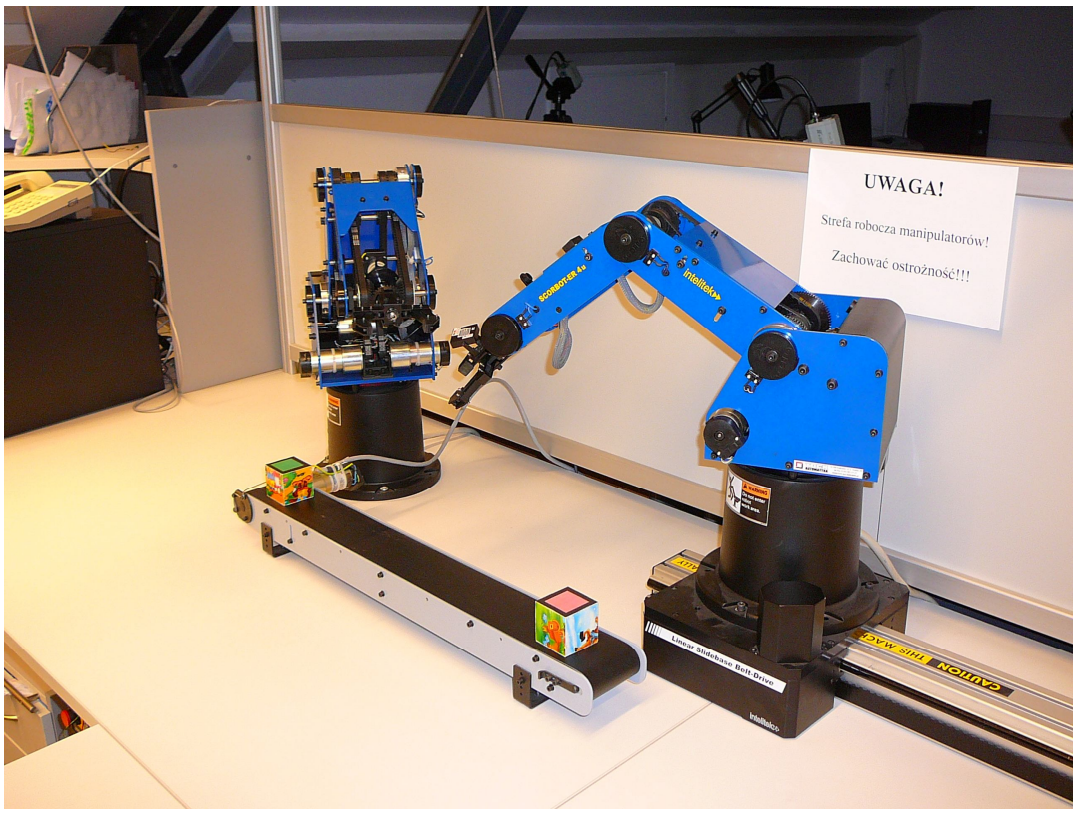
Przykład użycia wymienionych funkcji:

```
#include "sclink.h"
#include <stdio.h>

int main()
{
	SCON Con;
	unsigned long inputs;
```

```
	if (scInitializeNetwork() != TRUE)
	{
		printf("Nie mozna uruchomic biblioteki sclink!\n");
		return 0;
	}

	if ((Con = scConnect("localhost", 2000)) == SOCKET_ERROR)
	{
		printf("Nie moge sie polaczyc z serwerem!\n");
		scShutdown();
		return 0;
	}
	// wlaczenie wejscia cyfrowego 1
	scEnableDigitalInput(Con, 1);
	// jeśli wejscie numer 1 jest włączone
	scGetDigitalInputs(Con, &inputs);
	if (inputs & 1)
	{
		char AnalogInput;
		// ustawienie wyjscia cyfrowego 4 na wartosc 1
		scSetDigitalOutput(Con, 4, TRUE, TRUE);
		// pobranie i wyswietlenie wartosci wejscia analogowego
		scGetAnalogInput(Con, 1, &AnalogInput);
		printf("%d\n", AnalogInput);
		// ustawienie wyjscia analogowego 1 na wartosc z wejscia
		scSetAnalogOutput(Con, 1, AnalogInput);
	}

	scClose(Con);
	scShutdown();
	return 0;
}
```

6. Ćwiczenie laboratoryjne z wykorzystaniem manipulatorów

Ćwiczenie laboratoryjne będzie polegało na stworzeniu programu kontrolującego ruchy dwóch manipulatorów z czego jeden jest zamocowany na torze jezdnym a drugi posiada pas transmisyjny. Zadaniem takiego stanowiska jest wyselekcjonowanie klocków z odpowiednim znacznikiem i przetransportowanie ich na drugą stronę stołu. Do rozpoznawania klocków i określania ich pozycji w przestrzeni zostanie wykorzystana kamera.



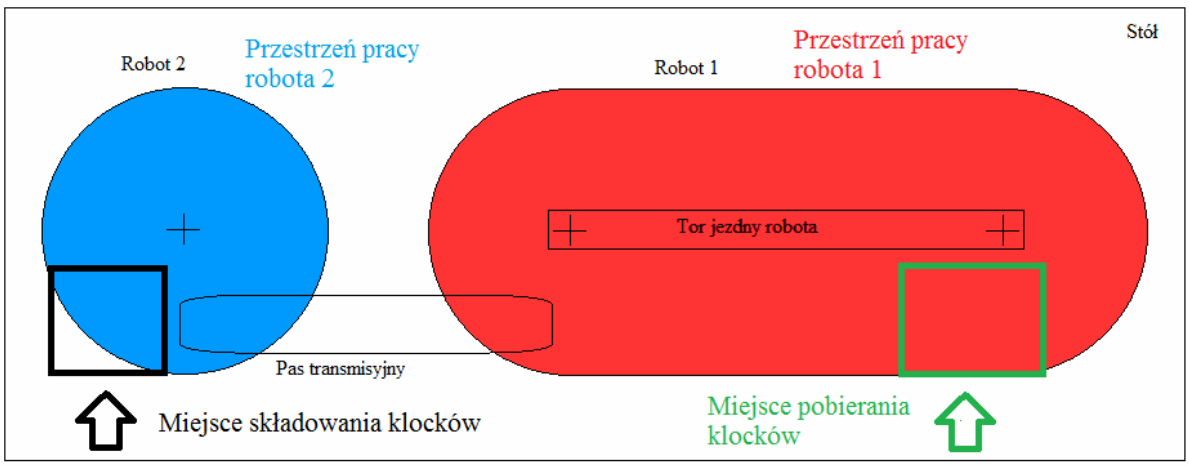
Rysunek 14: Widok boczny na stanowisko pracy robotów. Źródło: własne.

Wszystkie elementy stanowiska (roboty i osie zewnętrzne) są przytwierdzone do stołu co uniemożliwia zmianę ich położeń. Nad stanowiskiem znajdują się profile aluminiowe do których przytwierdzone są kamery. Każdy robot posiada własny sterownik i komputer do którego jest podłączony. Komputery te są połączone za pomocą sieci ethernet.

Dokładne zadanie programu kontroli robotów będzie polegało na:

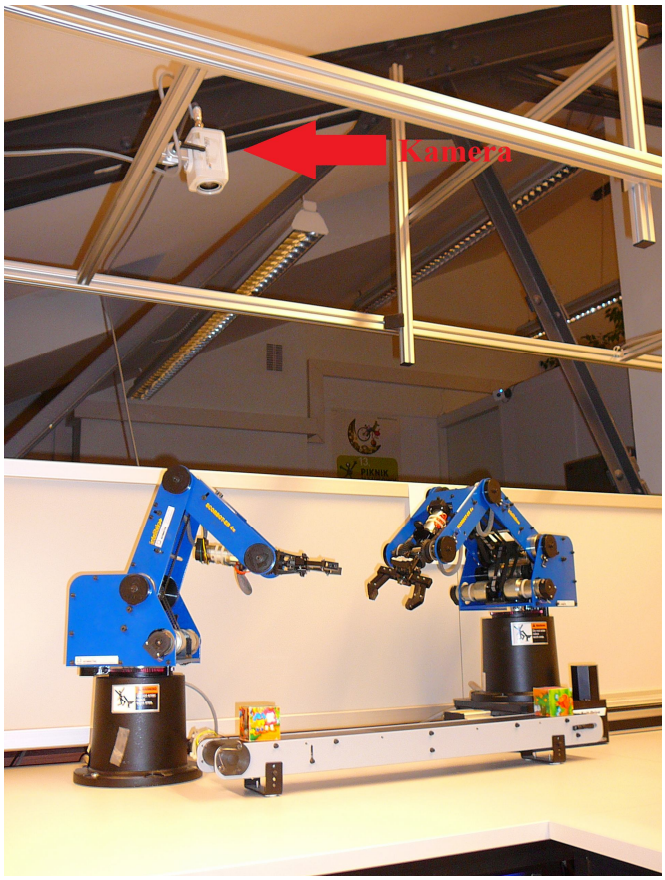
- zidentyfikowaniu klocków z odpowiednim znacznikiem znajdujących się w strefie „Miejsce pobierania klocków”
- przesunięcie robota numer 1 wzdłuż toru jezdnego aby jego przestrzeń robocza pokryła się z miejscem pobierania klocków

- pobrania jednego z klocków z odpowiednim znacznikiem
- przesunięcie robota numer 1 wzdłuż toru jezdnego aby jego przestrzeń robocza pokryła się z pasem transmisyjnym
- odłożenie klocka na pas transmisyjny
- równoczesne włączenie pasa transmisyjnego i powrót robota 1 w miejsce pobierania klocków
- wyłączenie pasa transmisyjnego po wykryciu przez drugą kamerę klocka
- pobranie przez robota 2 klocka z pasa transmisyjnego i odłożenie go w miejsce składowania klocków
- kontynuacje algorytmu aż do wyczerpania klocków

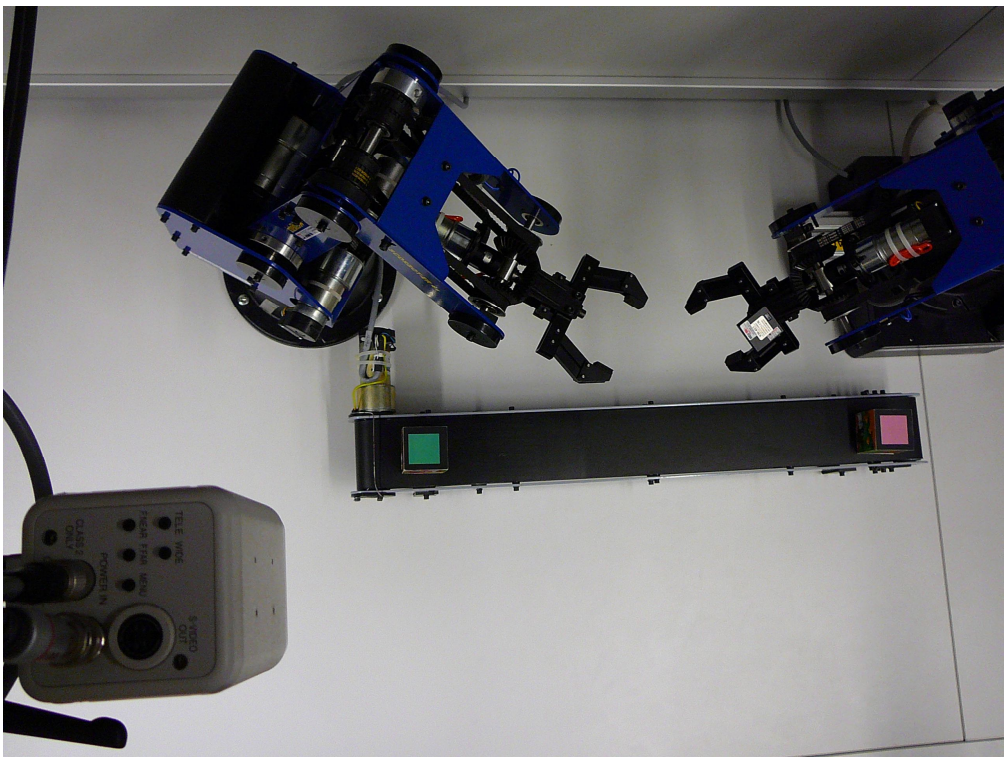


Rysunek 15: schemat organizacji stanowiska pracy robotów. Źródło: własne.

Na stanowisku znajdują się 2 kamery. Obserwują one stół od góry. Kamera nr 1 obserwuje prawą stronę stanowiska, czyli miejsce pobierania klocków i część przestrzeni roboczej robota 1, a kamera nr 2 obserwuje lewą stronę stołu czyli przestrzeń roboczą robota nr 2 i miejsce składowania klocków.



Rysunek 16: kamera nr 2 obserwująca stanowisko. Źródło: własne.



Rysunek 17: Widok zza kamery nr 2 na stanowisko. Źródło: własne.

Program sterujący pracą robotów będzie uruchomiony na jednym z komputerów. Za pomocą sieci Ethernet będzie on łączył się z serwerami uruchomionymi na obu komputerach podłączonych do robotów. Obydwie kamery będą również podłączone do głównego komputera z uruchomionym programem.

Lokalizacja klocków za pomocą kamery składa się z 3 etapów. Pierwszym z tych etapów jest kalibracja kamery i wyznaczenie jej parametrów wewnętrznych oraz parametrów zniekształceń soczewkowych które posłużą do ich korekcji. Parametry wewnętrzne kamery to c_x, c_y, f_x, f_y. Wynikają one z modelu kamery otworkowej. c_x i c_y są przesunięciami w osi X i Y środka matrycy światłoczułej względem osi optycznej obrazu a f_x i f_y są ogniskowymi odpowiednio dla osi X i osi Y. Rzutowanie punktu 3D na płaszczyznę matrycy światłoczułej w takim modelu przedstawia równanie:

$$q = MQ,\ gdzie\ q = \begin{bmatrix} x \\ y \\ w \end{bmatrix}, M = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}, Q = \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$$

Zniekształcenia soczewkowe dzielą się na zniekształcenia radialne i tangencjalne. Zniekształcenia radialne wynikają ze sferycznego kształtu soczewki. Są one opisane zazwyczaj za pomocą trzech współczynników wielomianu k_1, k_2, k_3. Równania służące do korekcji położenia pikseli wyrażają się:

$$x_k = x(1 + k_1 r^2 + k_2 r^4 + k_3 r^6)$$

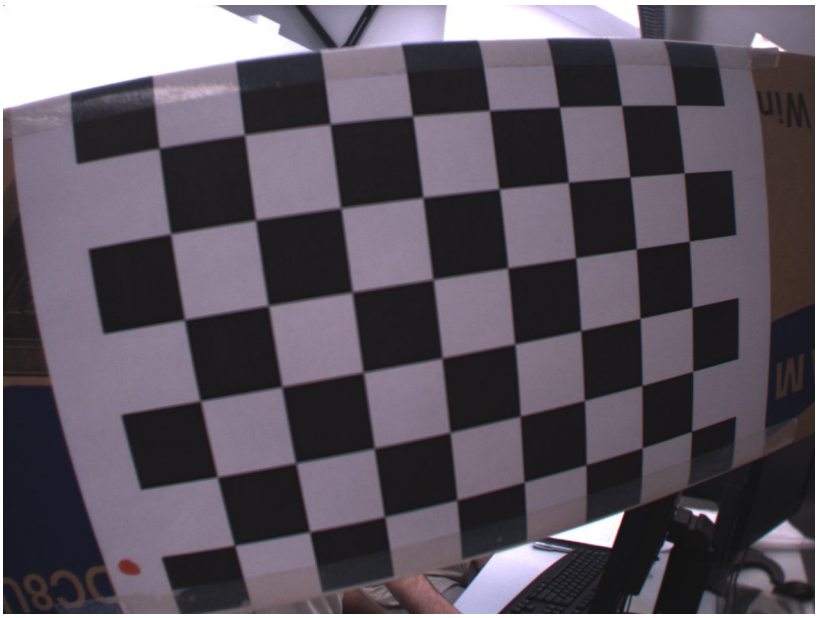
$$y_k = y(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \text{ gdzie } r = \sqrt{x^2 + y^2}$$

Zniekształcenia tangencjalne wynikają z nierównoległości płaszczyzny soczewki do płaszczyzny matrycy światłoczułej. Są one opisane za pomocą dwóch współczynników p_1 i p_2. Równania służące do korekcji położenia pikseli są następujące:

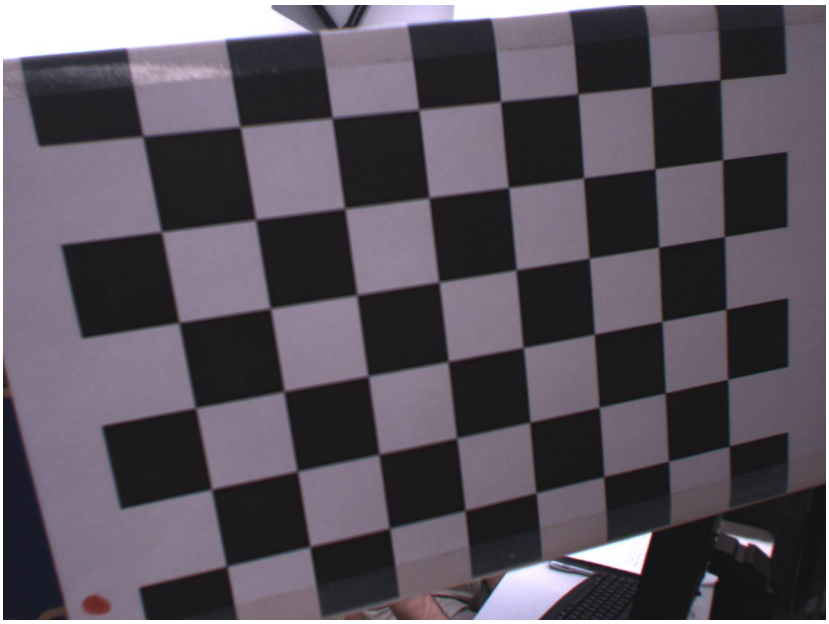
$$\begin{aligned} x_k &= x + 2p_1 xy + p_2(r^2 + 2x^2) \\ y_k &= y + p_1(r^2 + 2y^2) + 2p_2 xy \end{aligned} \text{ gdzie } r = \sqrt{x^2 + y^2}$$

Wyznaczenie tych parametrów odbywa się automatycznie za pomocą znacznika. Program służący do kalibracji został dostarczony przez promotora. Jego użycie polega na

uruchomieniu, wprowadzeniu parametrów i zrobieniu odpowiedniej ilości zdjęć znacznika. Rezultatem działania programu jest plik out_camera_data.txt zawierający odpowiednie parametry. Kalibracja jest wykonywana raz na początku, potem dane są wczytywane z dysku.

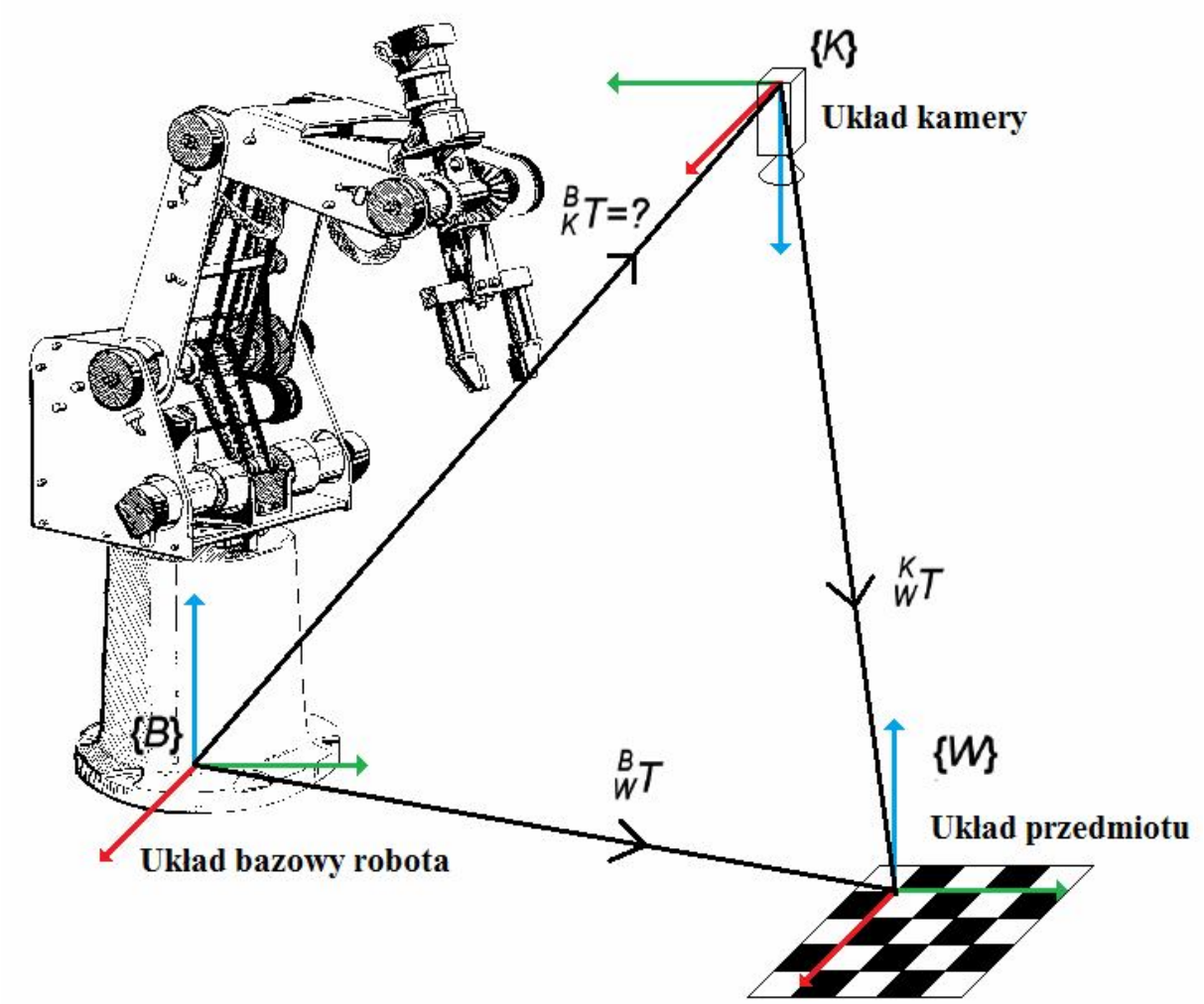


Rysunek 18: Zdjęcie szachownicy za pomocą kamery. Zniekształcenia radialne widać gołym okiem. Szachownica w rzeczywistości jest prostokątem, a w kamerze widać owal. Źródło [8].



Rysunek 19: Obraz z kamery po kompensacji zniekształceń. Linie proste w rzeczywistości są teraz także proste na obrazie.

Kolejnym etapem przygotowującym jest tzw. kalibracja oko-ręka. Służy ona do określenia przestrzennego położenia kamery względem bazowego układu współrzędnych robota.



Rysunek 20: Położenie układu kamery względem układu bazowego robota. Źródło: własne.

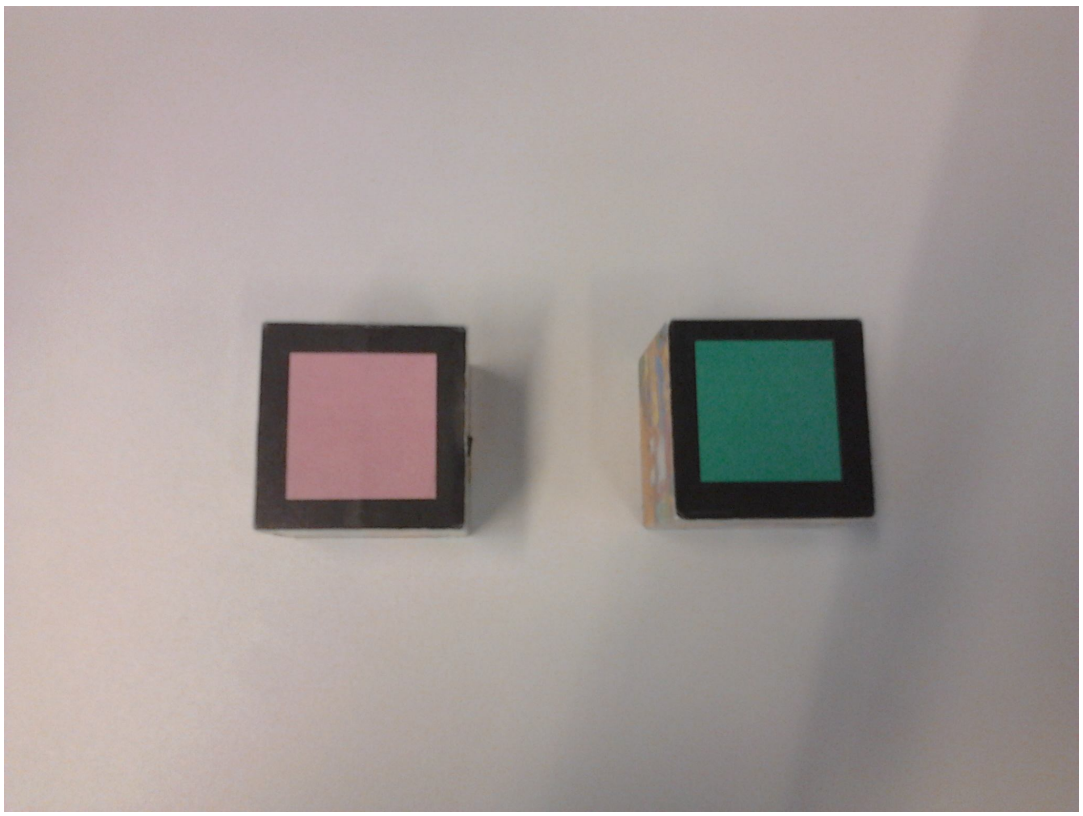
W czasie kalibracji oko ręka wyznacza jest macierz $^{B}_{K}T$. Pozwala ona określić położenie obiektu w bazowym układzie współrzędnych robota na podstawie położenia w układzie współrzędnych kamery. Macierz $^{B}_{K}T$ program oblicza na podstawie wzoru:

$$^{B}_{K}T = {}^{B}_{W}T \cdot {}^{K}_{W}T^{-1}$$

Macierz $^{B}_{W}T$ jest wyznaczana przez dojechanie punktem TCP robota do odpowiedniego znacznika i zanotowanie współrzędnych a macierz $^{K}_{W}T$ jest wyznaczana przez wyznaczenie położenia w przestrzeni znacznika widzianego przez kamerę.

Trzecim i ostatnim etapem wyznaczania położenia obiektu w przestrzeni jest etap wykonywany na bieżąco w każdej iteracji programu. Polega on na rozpoznaniu znacznika znajdującego się na klocku i uchwyceniu jego punktów charakterystycznych. Punkty te

zostają poddane transformacji z układu współrzędnych matrycy kamery do układu 3D po czym do układu bazowego robota. Do lokalizacji w przestrzeni potrzeba czterech punktów. Ich oddalenie od siebie decyduje o dokładności pomiarów a co za tym idzie o dokładności wyznaczania położenia klocka względem kamery. W naszym przypadku znacznik znajdujący się na obiekcie (klocku) jest kolorowym prostokątem na czarnym tle. Każdy z rogów tego prostokąta jest punktem charakterystycznym służącym do wyznaczania położenia w przestrzeni. Rozpoznanie znacznika odbywa się za pomocą biblioteki OpenCV. Obraz jest poddawany obróbce i są na nim wyszukiwane obszary o określonym natężeniu barw (natężenie to podaje się w zapisie kolorów HSV).



Rysunek 21: Widok na znaczniki znajdujące się na klockach. Źródło: własne.

7. Podsumowanie i wnioski

W ramach pracy zrealizowano wszystkie postawione zadania. Przedmiotem pracy były dwa manipulatory SCORBOT-ER 4u firmy Intelitek z indywidualnymi komputerami sterującymi. Mankamentem programowania robota było czasochłonne bazowanie przy każdym uruchomieniu programu. Zastosowano komunikację pomiędzy indywidualnymi komputerami co pozwoliło na stworzenie wspólnego programu dla 2 lub więcej manipulatorów, oraz uniknięcie każdorazowego bazowania robotów. Wynika to z faktu, że indywidualne programy są uruchamiane jednorazowo a następnie polecenia otrzymują z programu nadrzędnego. Wspólny program do sterowania pracą kilku robotów na linii produkcyjnej bardzo upraszcza współpracę manipulatorów.

Tematem sprawiającym dużo kłopotu było wykonanie spisu funkcji do sterowania robotem. Z powodu braku wsparcia technicznego producenta robota i braku dokumentacji, funkcje i parametry należało określić doświadczalnie w laboratorium. Polegało to na wielokrotnym uruchamianiu robota z różnymi parametrami i funkcjami. Zajęło to wiele czasu, lecz w efekcie tej pracy powstał spis sprawdzonych funkcji.

Uwieńczeniem wyżej wymienionych prac jest opracowanie ćwiczenia laboratoryjnego dla stanowiska składającego się z taśmociągu, 2 manipulatorów sterowanych poprzez sieć, oraz 2 kamer do rozpoznawania klocka. Program sterowania kamerami zwraca, do programu sterującego robotami, położenie klocków w przestrzeni. Umożliwiało to precyzyjne uchwycenie klocka przez manipulator. W mojej pracy napisałem program wykorzystujący kamery, taśmociąg i manipulatory. Przebieg pracy robotów na linii technologicznej został sfilmowany i nagrany na płytę CD.

W przyszłości można dodać modyfikację programu, o manualne wykonywanie ruchów otwierania/zamykania chwytaka za pomocą aplikacji serwera. Czasami podczas pracy zachodzi potrzeba zatrzymania programu nadrzędnego i robot zostaje w niewygodnej dla nas pozycji, bądź w jego szczękach znajduje się chwycony element. Możliwość sterowania ręcznego ruchem w takim przypadku pozwala nam szybko doprowadzić robota do żądanej pozycji lub uwolnienia przedmiotu bez pisania dedykowanego programu. Zadaniem studentów podczas ćwiczeń laboratoryjnych będzie napisanie programu sterującego linią, z wykorzystaniem funkcji zawartych w bibliotece.

Bibliografia

[1] Broszura reklamowa robota SCORBOT-ER 4u:
http://www.intelitek.com/admin/Products/uploads/File/File1_17.pdf, ostatni dostęp 19.01.2012

[2] Opis struktury nagłówka IP:
http://www.freesoft.org/CIE/Course/Section3/7.htm, ostatni dostęp 19.01.2012

[3] Joe Habraken, ABC sieci komputerowych, HELION, 2002

[4] Anthony Jones, Jim Ohlund, Programowanie sieciowe Microsoft Windows, Wydawnictwo RM, 2000

[5] Opis modelu OSI:
http://pl.wikipedia.org/wiki/Model_OSI, ostatni dostęp 19.01.2012

[6] Nagłówek TCP:
http://ciscoskills.net/2011/02/25/understanding-tcp/tcpheader/, ostatni dostęp 19.01.2012

[7] Michał Macias, Mariusz Rylski, Sterowanie manipulatora Scorbot-er 4u, Praca przejściowa, Politechnika Warszawska, Wydział elektryczny

[8] Witold Czajewski, Marcin Iwanowski, Maciej Sławinski,Inteligentne Maszyny i Systemy, preskrypt, Politechnika Warszawska, 2011

[9] SCORBOT-ER 4u User Manual, Intelitek 2001