Politechnika Poznańska
Wydział Informatyki
Katedra Sterowania i Inżynierii Systemów

PRACA DYPLOMOWA - INŻYNIERSKA

Jan Wiktor Meissner
Wojciech Pazda

Sterowanie robotem manipulacyjnym ze sprzężeniem wizyjnym

Promotor: dr inż. Marcin Kiełczewski

Poznań, 2017

Przedmowa

Zainteresowani robotami manipulacynymi, które poznaliśmy na laboratorium z przedmiotu Robotyka, postanowiliśmy wybrać temat pracy dyplomowej, który pozwoli nam bardziej zgłębić ten obszar wiedzy. Inspirację do zadania jakie ma wykonywać robot zaczerpnęliśmy z reklamy firmy Kuka, która przedstawia Robota Agilus grającego w tenisa stołowego. Wierzymy, że wiedza oraz doświadczenie, jakie nabyliśmy podczas tego projektu pozwoli nam osiągnąć spełnienie zarówno w życiu prywatnym jak i zawodowym.

Podziękowania

W tym miejscu chcielibyśmy serdecznie podziękować promotorowi tej pracy **dr. inż. Marcinowi Kiełczewskiemu** za pomoc w jej realizacji.
Dziekujemy również wszystkim pracownikom Katedry Inżynierii Sterowania i Systemów, którzy przyczynili się do rozwoju naszego projektu, a w szczególności: **mgr. inż. Tomaszowi Jedwabnemu** oraz **mgr. inż. Mateuszowi Przybyle**.

Spis treści

Rozdział 1

Wprowadzenie

1.1 Cel i zakres pracy

Głównym problemem poruszanym w tej pracy jest sterowanie robotem manipulacyjnym w oparciu o sprzężenie zwrotne pochodzące z systemu wizyjnego.

W dobie czwartej rewolucji przemysłowej [1], w której ewidentnie zaciera się bariera człowiek/maszyna przestają wystarczać manipulatory, które realizują stałe ścieżki ruchu po wcześniej zaprogramowanych punktach. Konieczne staje się szybkie reagowanie na zdarzenia zewnętrzne.

Założonym przez autorów efektem końcowym pracy jest robot przemysłowy, który wyposażony w rakietę do tenisa stołowego potrafi odbić w kontrolowany sposób lecącą w jego kierunku piłeczkę, a następnie utrzymywać ją w powietrzu poprzez ciągłe podbijanie. Zagadnienie interakcji narzędzia robota z piłką świetnie oddaje potrzebę ciągłej adaptacji do bieżącego stanu obiektu. Człowiek rzucający piłkę wprowadza do układu zmienną losową, której robot bez sprzężenia wizyjnego nie jest w stanie przewidzieć.

1.2 Podział pracy

Student Jan Meissner zajmował się:

- przygotowaniem narzędzia (rakietki),
- znalezieniem materiału retrorefleksyjnego,
- komunikacją z systemem wizyjnym,
- konfiguracją sieciową,
- filtrem Kalmana,
- regulatorem PID,
- sterownikiem bezpieczeństwa,

- algorytmem ruchu robota realizującym zadanie podbijania.

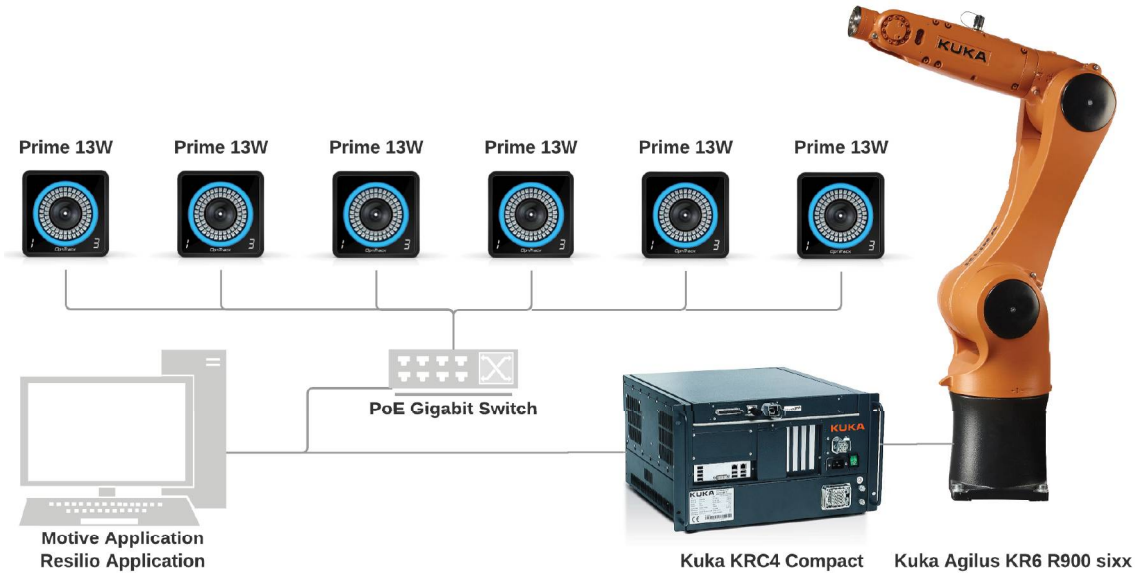
Student Wojciech Pazda zajmował się:

- przygotowaniem piłki,
- komunikacją z manipulatorem,
- oprawą graficzną aplikacji,
- kalibracją układów współrzędnych,
- aproksymacją wielomianową,
- sterownikiem ruchu,
- algorytmem ruchu robota realizującym zadanie stabilizacji piłki w punkcie.

Rozdział 2

Stanowisko laboratoryjne

W tym rozdziale zaprezentowano najważniejsze elementy stanowiska laboratoryjnego, w którego skład wchodzą: manipulator przemysłowy, system wizyjny oraz komputer PC. Ogólny schemat połączeń pomiędzy nimi przedstawia rysunek 2.1



Rysunek 2.1: Struktura sprzętowa systemu.

2.1 Robot KUKA

2.1.1 Manipulator

W projekcie został użyty robot Kuka Agilus KR6 R900 sixx [2] dostępny w laboratorium Katedry Sterowania i Inżynierii Systemów. Jest to manipulator przemysłowy o 6 osiach rotacyjnych, posiadający 6 stopni swobody. Jego maksy-

malny zasięg to 901 mm, a maksymalne obciążenie wynosi ok. 6 kg. W przemyśle robot jest używany do zadań niewymagających dużego udźwigu, lecz wysokiej precyzji i szybkości, np. cięcie, skrawanie, paletyzacja, spawanie, klejenie.

Spośród innych dostępnych robotów w laboratorium wybrany został właśnie ten, ze względu na jego dużą prędkość podczas pracy, dochodzącą nawet do 2 m/s oraz możliwość komunikacji z wykorzystaniem protokołu czasu rzeczywistego oraz pakietu RSI (opisany w rozdziale 3.2).

2.1.2 Szafa sterownicza

Do manipulatora dołączona jest szafa sterownicza KRC4 Compact pracująca z systemem Windows 7 Embedded oraz oprogramowanie Kuka KSS 8.3. Sterownik robota komunikuje się z napędami oraz wejściami i wyjściami logicznymi poprzez protokół komunikacyjny EtherCat. Opis konfiguracji sterownika znajduje się w dodatku B.

2.1.3 Panel smartPAD

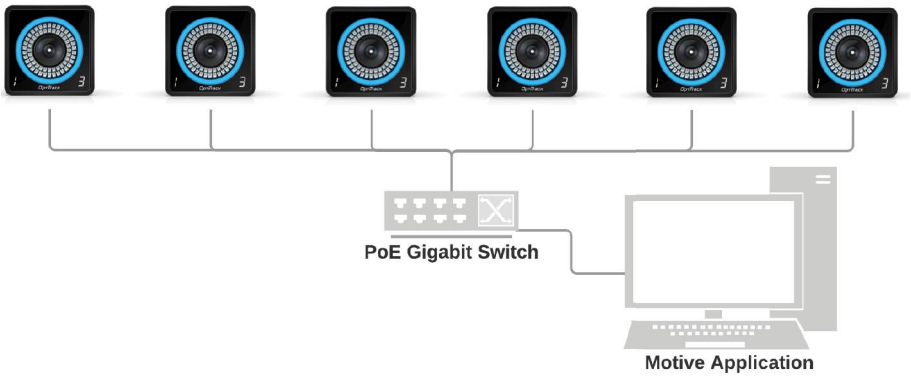
Do szafy sterowniczej dołączony jest panel dotykowy smartPAD, który umożliwia sterowanie robotem, uruchamiane programów oraz programowanie w języku KRL. Na panelu zainstalowany jest system Windows CE, którego zadaniem jest utworzenie sesji zdalnego pulpitu z systemem pracującym w szafie sterowniczej.

2.2 System wizyjny OptiTrack

OptiTrack [3] to komercyjny system wizyjny, który umożliwia lokalizowanie oraz śledzenie obiektów w trójwymiarowej przestrzeni dzięki zastosowaniu specjalnie spreparowanych znaczników. Jego innowacyjność na tle innych systemów wizyjnych polega na wykorzystaniu pasma podczerwonego zamiast widzialnego. Dzięki temu pomiar dokonywany przez kamery jest mniej podatny na zakłócenia pochodzące z otoczenia. Aplikacja Motive systemu OptiTrack dzięki swojemu API pozwala na łatwy dostęp do współrzędnych śledzonego obiektu za pomocą protokołu NatNet (opisany w rozdziale 3.1).

2.2.1 Architektura systemu

System wykorzystany na stanowisku składa się z 6 kamer, gigabitowego switcha oraz komputera, na którym zainstalowana jest aplikacja Motive. Strukturę zobrazowano na rysunku 2.2.



Rysunek 2.2: Struktura sprzętowa systemu wizyjnego.

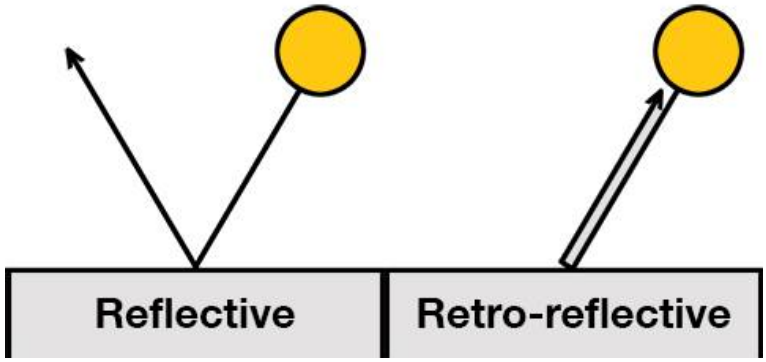
W projekcie wykorzystano kamery PRIME 13W firmy OptiTrack o rozdzielczości 1.3MP, kącie widzenia wynoszącym 82° oraz maksymalnej częstotliwości pracy wynoszącej 240 klatek/s. Każda z kamer wyposażona jest w filtr pasmowo-przepustowy, którego zadaniem jest przepuszczanie tylko promieniowania zbliżonego do długości fali 850nm. Kamery posiadają gigabitowy port ethernet, przez który podłączone są do switcha. Do zasilania kamer wykorzystano technologię Power Over Ethernet, switch oprócz funkcji przełącznika spełnia również funkcję zasilacza. Obie funkcje realizowane są w jednym ekranowanym kablu FTP kategorii 6.

Do switcha podłączony jest również komputer stacjonarny pracujący pod kontrolą systemu Windows 7. Wyposażony jest w dwurdzeniowy procesor Core i5-4690 firmy Intel oraz 8GB pamięci RAM. Na komputerze zainstalowana jest aplikacja Motive, która odczytuje dane o położeniu znaczników w przestrzeni 2D każdej kamery oraz rekonstruuje je do przestrzeni 3D.

Opis konfiguracji programu Motive znajduje się w dodatku A.

2.2.2 Zasada działania

Wspomniany system wizyjny lokalizuje obiekty za pomocą retrorefleksyjnych znaczników. Retrorefleksyjność to cecha obiektów, które zaginają promienie o 180°, promień odbity od znacznika wraca w kierunku swojego źródła. Zobrazowano to na rysunku 2.3. Zjawisko to wykorzystywane jest na co dzień we wszelkich odblaskach oraz znakach i pasach drogowych. Każda z kamer wyposażona jest w zestaw diód LED emitujących fale o długości 850nm (podczerwień). Promienie te są następnie odbijane od znaczników, przechodzą przez filtr pasmowo-przepustowy i padają na matrycę kamery.



Rysunek 2.3: Różnica pomiędzy materiałem refleksyjnym a retrorefleksyjnym [4].

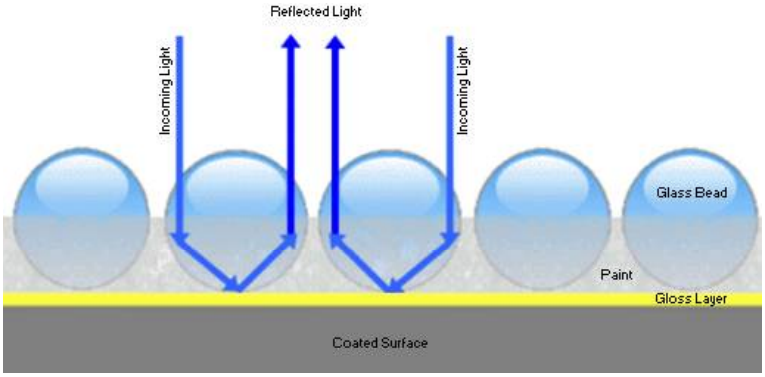
2.3 Piłka do tenisa stołowego

Piłeczka do tenisa stołowego nie posiada właściwości retrorefleksyjnych, dlatego by umożliwić śledzenie jej przez system wizyjny należało pokryć ją odpowiednim materiałem. Ze względu na fakt, iż producent nie umieścił informacji na temat zastosowanego preparatu, konieczne było przeprowadzenie eksperymentów z różnymi materiałami. Do prób użyto preparatów, które wyglądem lub właściwościami przypominają oryginalne znaczniki systemu OptiTrack:

- Tlenek Tytanu (TiO_2, Biel Tytanowa) - dobrze odbija podczerwień,
- Cynk Metaliczny - preparat konserwujący, przypominający kolorem znacznik firmy OptiTrack,
- Mikrogranulki Refleksyjne - używane jako dodatek do farb drogowych, posiadają właściwości retrorefleksyjne,
- Pył odblaskowy - używany do wykonywania odblaskowych nadruków na koszulkach.

Zamierzony rezultat został uzyskany dopiero po zastosowaniu pyłu odblaskowego. Pył ten składa się z szarych mikrokulek, które zaginają promienie świetlne. Zobrazowano to na rysunku 2.4.

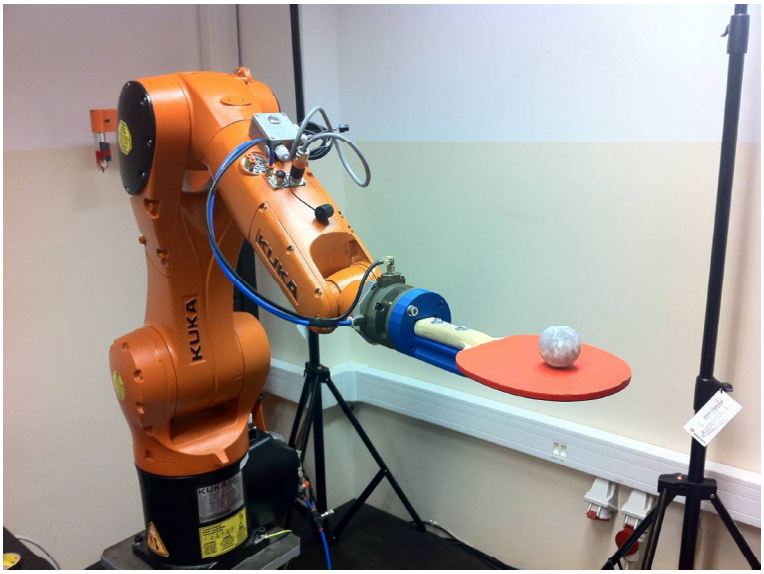
Piłeczka została pokryta proszkiem przyklejonym do jej powierzchni za pomocą cienkiej warstwy lakieru poliuretanowego. Proces nakładania materiału przedstawia rysunek 2.5. Na rysunku 2.6 można zobaczyć właściwości retrorefleksyjne piłeczki oraz pozostałego materiału. Światło lampy błyskowej silnie odbija się od powierzchni pokrytych proszkiem.



Rysunek 2.4: Zasada działania mikrokulek retrorefleksyjnych [5]

Rysunek 2.5: Proces nakładania materiału retrorefleksyjnego na piłeczkę w świetle dziennym.

Rysunek 2.6: Proces nakładania materiału retrorefleksyjnego na piłeczkę w świetle lampy błyskowej, zdjęcie ukazuje silne właściwości retrorefleksyjne.

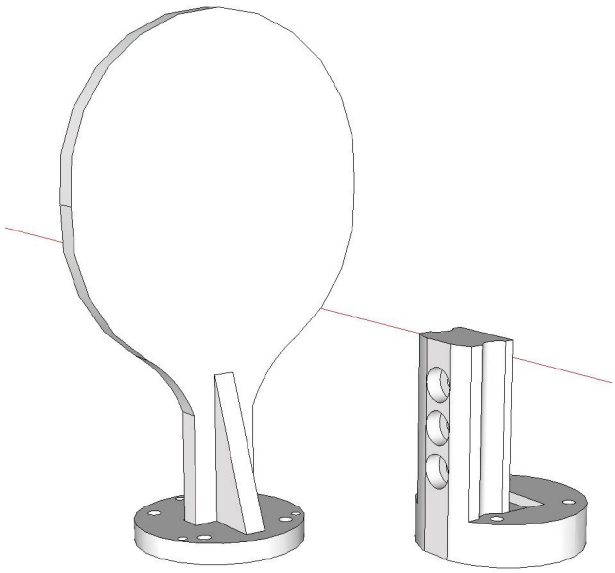


Rysunek 2.7: Rakietka drewniana z plastikowym chwytakiem.

Rysunek 2.8: Rakietka aluminiowa.

2.4 Rakietka do tenisa stołowego

Pierwsza wersja rakietki (przedstawiona na rysunku 2.7) składała się z normalnej rakiety tenisa stołowego oraz wydrukowanego w technologi 3D chwytaka. Ze względu na duże straty energii (piłka spadając na rakietkę powodowała jej wygięcie), zdecydowano się na wykonanie drugiej rakietki zrobionej z aluminium. Rakietka aluminiowa (przedstawiona na rysunku 2.8), ma znacznie większą sztywność, przy takiej samej prędkości odbija piłkę wyżej niż rakietka klasyczna. Oba elementy zostały zaprojektowane w programie SketchUp, na rysunku 2.9 przedstawiono ich wizualizacje. Projekt uwzględnia zastosowanie sprzęgła bezpieczeństwa [6] i montaż do flanszy robota. Drugą wersję rakietki wycięto z aluminiowej blachy o grubości 8mm i pospawano. Na załączonej do pracy płycie CD znajdują się rysunki wykonawcze poszczególnych elementów narzędzia.



Rysunek 2.9: Projekt elementów w programie SketchUp.

2.5 Aplikacja Resilio

Za akwizycję danych z systemu wizyjnego oraz wysyłanie komend ruchu do robota odpowiada napisana przez autorów aplikacja Resilio. Jest to program, który jednocześnie zachowuje się jak klient w przypadku systemu wizyjnego oraz jak serwer w przypadku robota. Oprogramowanie systemu wizyjnego i aplikacja Resilio działają na jednym komputerze. Komunikacja z nimi odbywa się jednak nadal z wykorzystaniem protokołów sieciowych.

2.5.1 Środowisko programistyczne oraz system kontroli wersji

Część programowa pracy została napisana w języku C# oraz środowisku uruchomieniowym .NET Framework firmy Microsoft. Na środowisko programistyczne wybrany został program Visual Studio firmy Microsoft. Dla ułatwienia pracy wykorzystano system kontroli wersji GIT, który umożliwia łatwe współdzielenie kodu pomiędzy twórcami. Jako serwer systemu kontroli wersji wykorzystano prywatne repozytorium w serwisie github.com.

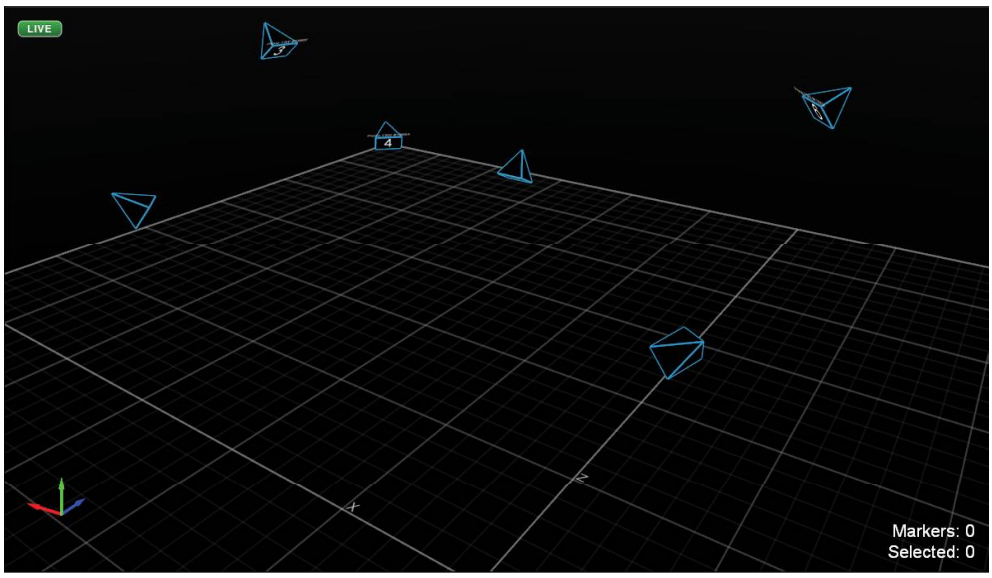
2.6 Przygotowanie stanowiska laboratoryjnego

Kamery zostały przymocowane do zewnętrznych ścian klatki robota. Dzięki wykorzystaniu profili aluminiowych możliwa jest swobodna zmiana położenia ka-

Rysunek 2.10: Jedna z kamer systemu OptiTrack.

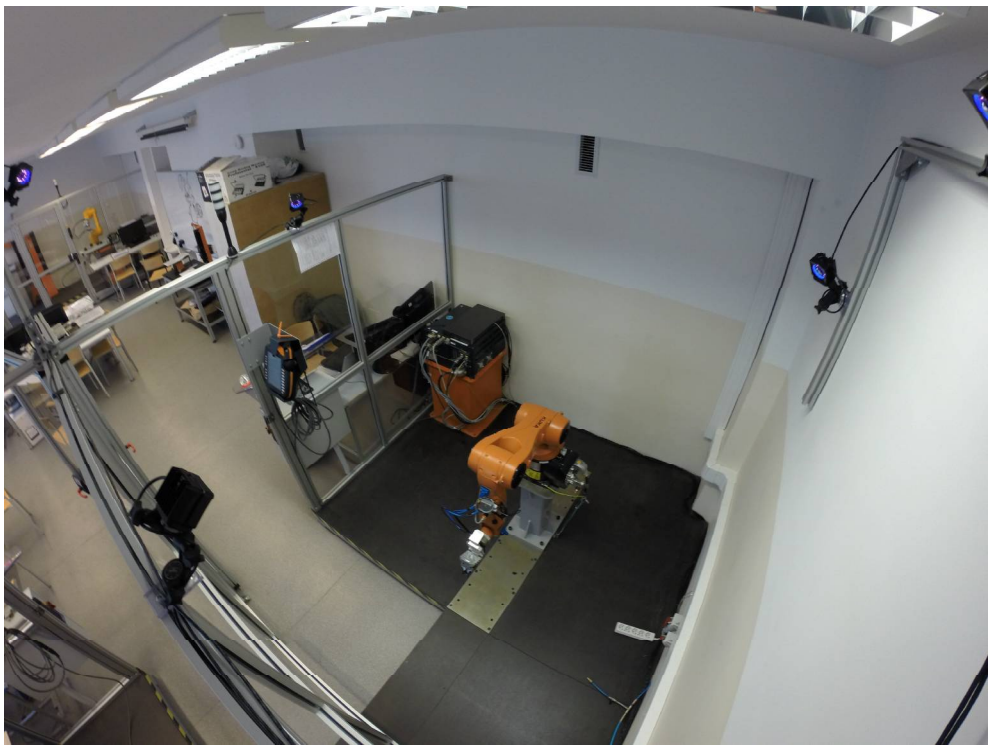
mer, co ułatwia kalibrację systemu oraz dobór pozycji i orientacji kamer. Sposób ich mocowania został przedstawiony na rysunku 2.10, a wizualizacja przestrzeni roboczej w aplikacji Motive na rysunku 2.11.

W celu zwiększenia obszaru roboczego (rysunek 2.13) oraz zapewnienia bezpieczeństwa konieczne było przemieszczenie szafy sterującej robotem (rysunek 2.12). Dzięki temu można również szybko wnieść lub wynieść stół wykorzystywany podczas zajęć laboratoryjnych, co znacznie zwiększa obszar pracy manipulatora. W dodatku zamieszczonym na płycie CD dołączonej do tej pracy znajdują się zdjęcia przewodów poprowadzonych pod podłogą klatki.



Rysunek 2.11: Wizualizacja przestrzeni roboczej w aplikacji Motive.

Rysunek 2.12: Położenie szafy po zmianach.



Rysunek 2.13: Widok całego obszaru roboczego.

Rozdział 3

Komunikacja pomiędzy systemami

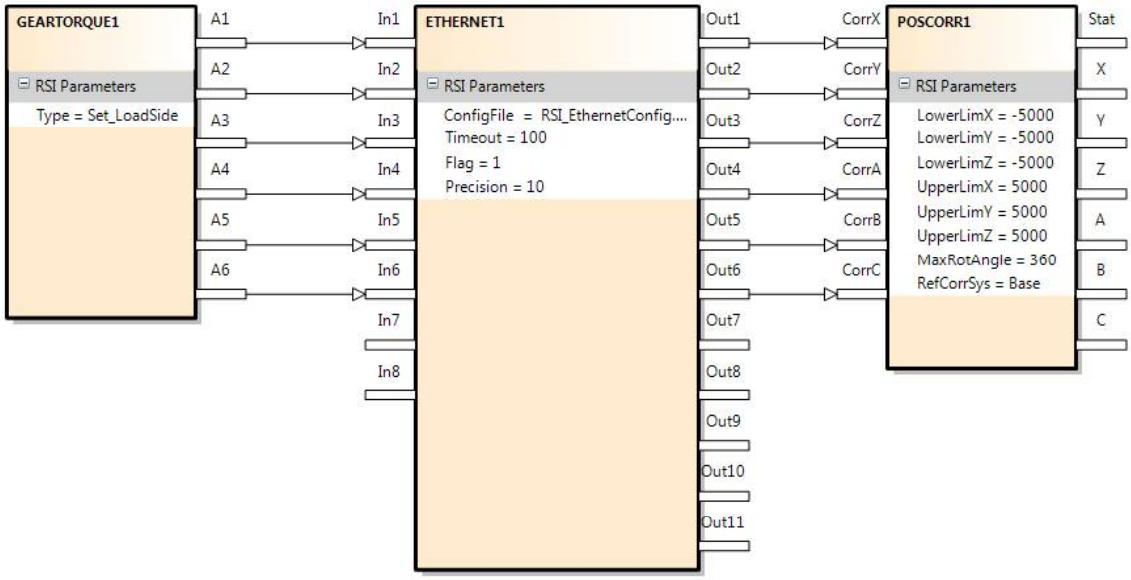
W tym rozdziale zostały omówione protokoły wykorzystane do połączenia poszczególnych części systemu oraz konfiguracja urządzeń odpowiadających za komunikację sieciową.

3.1 OptiTrack - PC, protokół NATNET 1.8.0

Do połączenia systemu wizyjnego z aplikacją wykorzystano protokół NatNet w wersji 1.8. Firma OptiTrack udostępnia SDK protokołu oraz skromną implementację w środowisku .NET. Dzięki temu możliwe było zaimplementowanie modułu odbierania danych pomiarowych w czasie rzeczywistym. Działanie modułu klienta sieci sprowadza się do odebrania danych, sprawdzenia ich zgodności oraz zapisania ich w przygotowanym obiekcie klasy BallData. Dodatkowo sprawdzane są znaczniki czasowe poszczególnych ramek danych, dzięki czemu dokładnie wiadomo, ile pakietów zostało utraconych (np. poprzez zbyt wolne przetwarzanie danych).

3.2 Kuka - PC, protokół RSI 3.2

Do komunikacji aplikacji pracującej na komputerze PC z robotem użyto protokół czasu rzeczywistego - Kuka Robot Sensor Interface [7, 8] w wersji 3.2. Jest to komunikacja typu klient - serwer. Wymiana danych następuje poprzez połączenie UDP co 4 lub 12ms (w zależności od konfiguracji). Protokół jest dedykowany do aplikacji przemysłowych, w których robot wykonujący program dostaje informacje od zewnętrznych czujników. Na podstawie tych informacji dokonywane



Rysunek 3.1: Konfiguracja komunikacji RSI w programie RSI Visual Shell.

są korekty konfiguracji poszczególnych złączy lub w przypadku korekt kartezjańskich, pozycji i orientacji.

Aby uruchomić komunikację, należy zainstalować pakiet RSI na sterowniku robota, skonfigurować interfejs połączenia IP oraz napisać aplikacje serwera na komputerze. Manipulator jako klient łączy się do serwera.

Pakiet RSI ma bardzo wiele opcji i rozległą konfigurację. W zależności od ustawień system może dokonywać korekt pozycji w układzie kartezjańskim lub w poszczególnych złączach. Korekty pozycji robota można wysyłać w dwóch trybach: relative - zadaje się wartość, o jaką robot powinien się przemieścić, absolute - zadaje się pozycję, w której robot powinien się znajdować po wykonaniu korekty. W komunikacji można również zdefiniować system współrzędnych, względem którego wykonywany będzie ruch, np. world, base, tool. Wszystkie informacje o konfiguracji systemu zapisane są w plikach z rozszerzeniem .rsi, które należy umieścić w pamięci sterownika. Aby uprościć proces konfiguracji skorzystano z dedykowanego oprogramowania RSI Visual (wygląd programu przedstawia rysunek 3.1), w którym poprzez utworzenie bloków i połączeń między nimi następuje automatyczna generacja większości potrzebnych plików:

- RSIEthernet.rsi,
- RSIEthernet.rsi.xml,
- RSIEthernet.rsi.diagram.

Przedstawiony blok ETHERNET1 odpowiada za komunikację sieciową sterownika z komputerem. Do lewej strony bloku podłącza się sygnały, które będą wysyłane przez robota. W tym przypadku są to dane o aktualnych momentach

sił na przekładniach. Po prawej stronie podłączony jest blok odczytujący dane z serwera i realizujący korekty kartezjańskie - POSCORR1. W jego ustawieniach można dodatkowo wybrać limity przesunięć oraz układ współrzędnych, względem którego będzie następowało przemieszczenie. Obrót narzędzia wykonywany jest względem osi wybranego układu przesuniętego do TCP (Tool Center Point) narzędzia.

W konfiguracji bloku ETHERNET1 możliwe jest ustawienie dokładności reprezentacji wysyłanych danych oraz liczby utraconych pakietów, po której następuje zerwanie komunikacji z serwerem. Do poprawnej konfiguracji bloku konieczne jest jeszcze podanie ścieżki do pliku zawierającego ustawienia komunikacji. Ze względu na fakt, iż jest to jedyny plik, który nie jest generowany przez aplikację RSI Visual, załączono go w listingu 3.1.

```
<ROOT>
  <CONFIG>
    <IP_NUMBER>192.168.3.1</IP_NUMBER>
    <PORT>8081</PORT>
    <SENTYPE>Resilio_Project</SENTYPE>
    <ONLYSEND>FALSE</ONLYSEND>
  <SEND>
    <ELEMENTS>
      <ELEMENT TAG="DEF_RIst" TYPE="DOUBLE" INDX="INTERNAL" />
      <ELEMENT TAG="DEF_RSol" TYPE="DOUBLE" INDX="INTERNAL" />
      <ELEMENT TAG="DEF_AIPos" TYPE="DOUBLE" INDX="INTERNAL" />
      <ELEMENT TAG="DEF_ASPos" TYPE="DOUBLE" INDX="INTERNAL" />
      <ELEMENT TAG="DEF_MACur" TYPE="DOUBLE" INDX="INTERNAL" />
      <ELEMENT TAG="DEF_Delay" TYPE="LONG" INDX="INTERNAL" />
      <ELEMENT TAG="GEARTORQUE1.A1" TYPE="DOUBLE" INDX="1" />
      <ELEMENT TAG="GEARTORQUE1.A2" TYPE="DOUBLE" INDX="2" />
      <ELEMENT TAG="GEARTORQUE1.A3" TYPE="DOUBLE" INDX="3" />
      <ELEMENT TAG="GEARTORQUE1.A4" TYPE="DOUBLE" INDX="4" />
      <ELEMENT TAG="GEARTORQUE1.A5" TYPE="DOUBLE" INDX="5" />
      <ELEMENT TAG="GEARTORQUE1.A6" TYPE="DOUBLE" INDX="6" />
    </ELEMENTS>
  </SEND>
  <RECEIVE>
    <ELEMENTS>
      <ELEMENT TAG="DEF_EStr" TYPE="STRING" INDX="INTERNAL" />
      <ELEMENT TAG="RKorr.X" TYPE="DOUBLE" INDX="1" HOLDON="1" />
      <ELEMENT TAG="RKorr.Y" TYPE="DOUBLE" INDX="2" HOLDON="1" />
      <ELEMENT TAG="RKorr.Z" TYPE="DOUBLE" INDX="3" HOLDON="1" />
      <ELEMENT TAG="RKorr.A" TYPE="DOUBLE" INDX="4" HOLDON="1" />
      <ELEMENT TAG="RKorr.B" TYPE="DOUBLE" INDX="5" HOLDON="1" />
```

```
31         <ELEMENT TAG="RKorr.C" TYPE="DOUBLE" INDX="6" HOLDON="1" />
32       </ELEMENTS>
33     </RECEIVE>
34 </ROOT>
```

Listing 3.1: Plik konfiguracyjny bloku. Ethernet - RSI_EthernetConfig

Do uruchomienia komunikacji konieczne jest napisanie programu w języku KRL, który wywoła funkcje pakietu RSI oraz zestawi połączenie zgodnie z wcześniej przedstawioną konfiguracją. Kod programu znajduje się w listingu 3.2.

```
1  DEF RSI_Ethernet( )
2  ; ===========================================
3  ;
4  ; RSI EXAMPLE: ETHERNET communication
5  ; Realtime UDP data exchange with server application
6  ;
7  ; ===========================================
8
9  ; Declaration of KRL variables
10 DECL INT ret ; Return value for RSI commands
11 DECL INT CONTID ; ContainerID
12
13 INI
14
15 PTP P Vel=100 \% PDAT9 Tool[15]:rakietka Base[0];
16
17 ; Create RSI Context
18 ret = RSI_CREATE("RSI_Ethernet.rsi",CONTID,TRUE)
19 IF (ret <> RSIOK) THEN
20   HALT
21 ENDIF
22
23 ; Start RSI execution
24 ret = RSI_ON(#RELATIVE)
25 IF (ret <> RSIOK) THEN
26   HALT
27 ENDIF
28
29 ; Sensor guided movement
30  RSI_MOVECORR()
31
32 ret = RSI_OFF()
33 IF (ret <> RSIOK) THEN
34   HALT
35 ENDIF
```

```
36
37 END
```

Listing 3.2: Plik RSIethernet.src wykonywany na robocie podczas działania. aplikacji

Dane wysyłane i odbierane przez robota, zapisywane w reprezentacji XML, muszą mieć składnię zgodną z wcześniej zaprezentowanym plikiem konfiguracyjnym (listing 3.1). Ramka żądania danych wysłana z robota (klienta) została przedstawiona na listingu 3.3, ramka odpowiedzi wysłana z aplikacji Resilio (serwera) została przestawiona na listingu 3.4.

```
1 <Rob Type="KUKA">
2 <RIst
3 X="0.0022054811" Y="850.0036621094" Z="100.0022964478"
4 A="-23.7704410553" B="88.7474975586" C="-113.7855911255"/>
5 <RSol
6 X="0.0024411387" Y="850.0020141777" Z="100.0003051426"
7 A="-23.7758750814" B="88.7472152550" C="-113.7909774554"/>
8 <AIPos
9 A1="-89.9931884800" A2="-15.4559783917" A3="103.2046875000"
10 A4="0.0169254106" A5="-87.2435902573" A6="1.1454798561"/>
11 <ASPos
12 A1="-89.9931935482" A2="-15.4559171980" A3="103.2048714565"
13 A4="0.0168721200" A5="-87.2436131906" A6="1.1456941081"/>
14 <MACur
15 A1="0.0115871429" A2="0.0046730042" A3="0.0022411346"
16 A4="-0.0049591064" A5="-0.0147342682" A6="-0.0056266785"/>
17 <Delay D="0"/>
18 <GEARTORQUE1
19 A1="-1.2612520556" A2="-0.3860683516" A3="-0.2305452309"
20 A4="-0.1086477616" A5="-0.4281812277" A6="-0.1235963759"/>
21 <IPOC>3331134</IPOC>
22 </Rob>
```

Listing 3.3: Ramka żądania danych wysyłana z robota (klienta).

```
1 <Sen Type="Resilio_Project">
2 <EStr>Resilio Project</EStr>
3 <RKorr X="0" Y="0" Z="0" A="0" B="0" C="0" />
4 <IPOC>3331138</IPOC>
5 </Sen>
```

Listing 3.4: Ramka odpowiedzi wysyłana z PC (serwera).

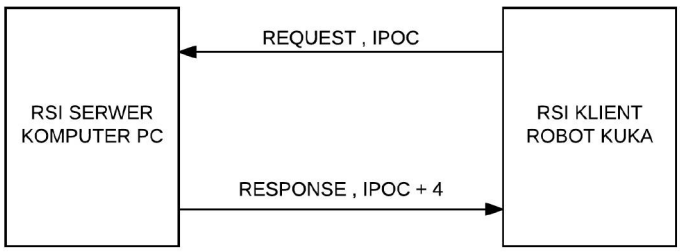
W ramce żądania danych wysyłane są aktualne dane z manipulatora, takie jak:

- RIst - aktualna pozycja i orientacja w wybranym układzie kartezjańskim,
- RSol - pozycja i orientacja w jakiej byłby robot gdyby nie wykonano korekt,
- AIpos - aktualna pozycja poszczególnych złączy robota (A1-A6),
- ASpos - pozycja w jakiej byłyby złącza gdyby nie wykonano korekt,
- MACur - aktualny prąd płynący przez poszczególne silniki osi,
- Delay - ilość utraconych pakietów,
- GEARTORQUE1 - aktualne momenty sił na poszczególnych złączach.

W ramce odpowiedzi wysyłany jest rozkaz wykonania korekty kartezjańskiej (RKorr) oraz numer IPOC.

Protokół UDP nie ma zaimplementowanej kontroli potwierdzenia przesłania informacji. Dlatego w komunikacji RSI występuje zmienna IPOC. Robot wysyła jej wartości i w ciągu czasu 1 cyklu (w tym przypadku 4 ms) czeka na odpowiedź z wartością IPOC powiększoną o 4 (rysunek 3.2). W przypadku odesłania złego numeru IPOC lub spóźnienia się z odpowiedzią, inkrementowany jest licznik ramek niedostarczonych. Gdy wartość licznika przekroczy wcześniej zadaną wartość, w tym przypadku 100, połączenie zostaje zerwane, a na panelu manipulatora pojawia się komunikat RSITimeout.

W protokole RSI występuje tzw. opóźnienie transportowe, które wynosi 32ms (równowartość 8 ramek). Robot buforuje wysłane korekty w celu zachowania ciągłości ruchu pomiędzy kolejnymi ramkami. Sprowadza się to do tego, że robot wykona korektę położenia 32 ms po faktycznym wysłaniu danych.



Rysunek 3.2: Cyklogram komunikacji RSI klient - serwer.

3.3 Konfiguracja sieciowa

Połączenia sieciowe zostały skonfigurowane zgodnie z ustawieniami zamieszczonymi w tabeli 3.1. Program Motive samodzielnie wykrywa podłączone do sieci kamery. Ze względu na to, że serwer oraz klient protokołu NatNet znajdują się na tej samej maszynie, ich adresy sieciowe to lokalny adres hosta (loopback).

Tabela 3.1: Ustawienia sieciowe

Adresy IP oraz porty poszczególnych komponentów		
Komponent	IP	PORT
NatNet Client	127.0.0.1	1510 oraz 1511
NatNet Server	127.0.0.1	1510 oraz 1511
RSI Client	192.168.3.1	8081
RSI Server	192.168.3.2	8081

Rozdział 4

Aplikacja Resilio

W tym rozdziale została omówiona aplikacja Resilio (z łac. *resilio* - skakać) - główny program odpowiadający za odczyt pozycji piłki z systemu wizyjnego oraz wysyłanie komend sterujących do robota.

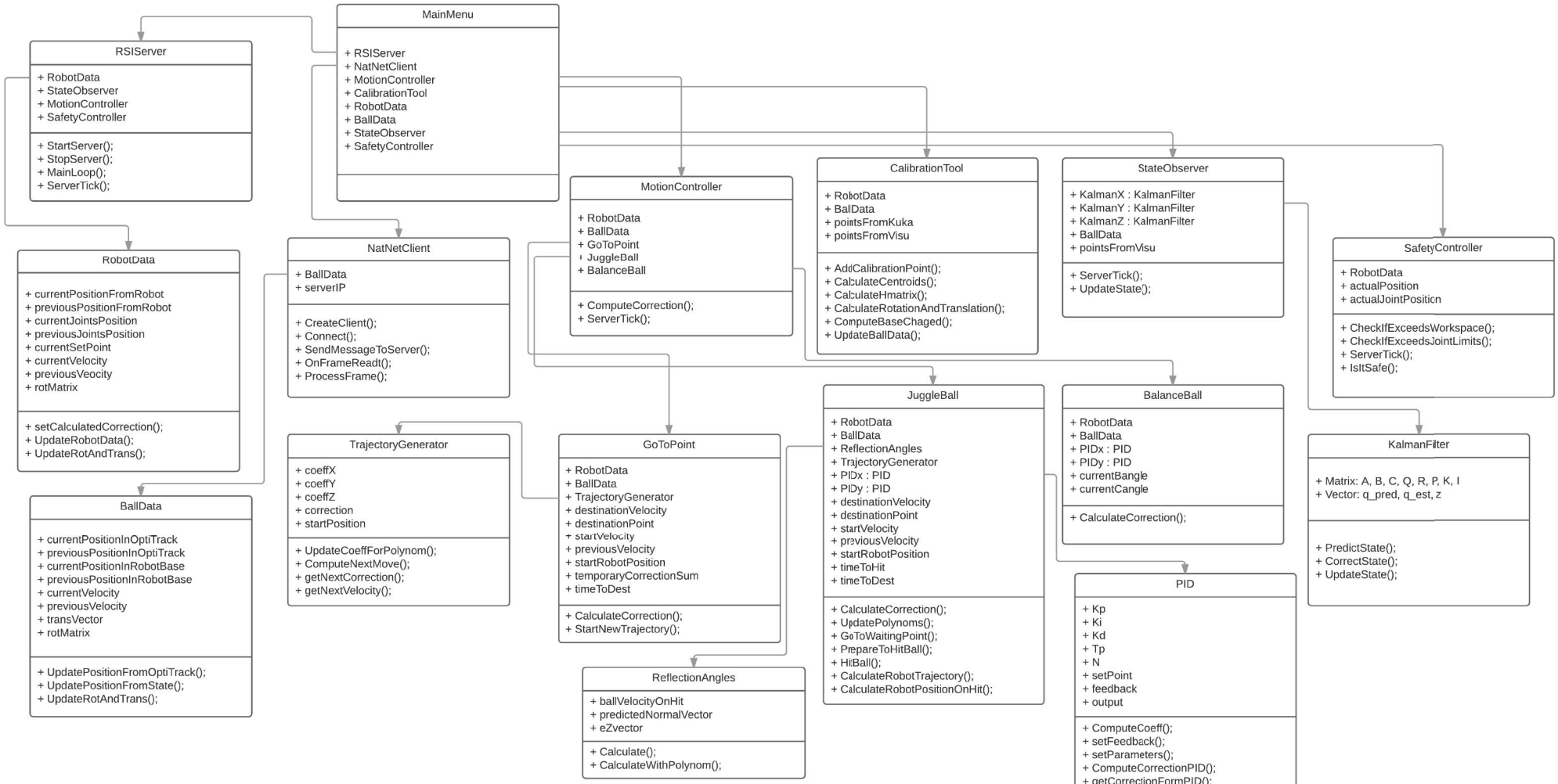
4.1 Struktura aplikacji

Struktura całej aplikacji została przedstawiona na rysunku 4.1. Program można podzielić na 3 równolegle działające wątki:

- wątek serwera, obiekt RSIServer - odpowiedzialny za wysyłanie komend ruchu oraz wywoływanie wszystkich obiektów, które taktowane są z częstotliwością działania RSI (4ms),
- wątek klienta, obiekt NatNetClient - odpowiedzialny za akwizycję danych, które pochodzą z systemu wizyjnego,
- wątek widoku - odpowiedzialny za obsługę graficzną interfejsu użytkownika oraz obsługę pochodzących z niego zdarzeń.

Oprócz obiektów odpowiedzialnych za obsługę logiki aplikacji powstały klasy, których zadaniem jest przechowywanie danych o stanie układu regulacji. Są to następujące obiekty:

- BallData - przechowuje dane dotyczące piłki,
- RobotData - przechowuje dane dotyczące manipulatora.



Rysunek 4.1: Struktura aplikacji Resilio.

4.2 Komunikacja z robotem - RSIServer

Obiekt RSIServer działa w ciągłej pętli i oczekuje na pakiet danych przychodzących z robota. Po ich otrzymaniu następuje przepisanie danych z łańcucha znaków XML do odpowiadających danym zmiennych. Następnie wywoływana jest funkcja ServerTick(), która taktuje wszystkie zależne od czasu moduły aplikacji. Dzięki temu program, pomimo wykonywania na systemie Windows, działa w czasie rzeczywistym, ponieważ ma stałe, zewnętrzne taktowanie. Dalej wywoływany jest kontroler bezpieczeństwa. Na tym etapie obliczone są już korekty wysyłane do robota. Za ich ponowne zapisanie w łańcuchu danych XML odpowiada funkcja ComposeResponse(). Należy pamiętać, że mimo zewnętrznego zegara (RSI), to system Windows przydziela zasoby procesora. Mogą więc wyniknąć opóźnienia w transmisji danych, które są obsługiwane po stronie kontrolera robota. Po zapisaniu łańcuch zostaje wysłany do kontrolera robota. Wygląd głównej pętli serwera został zaprezentowany na listingu 4.1.

```
1    while (serverRunning)
2    {
3        var received = await udpListener.Receive();
4        ParseRequest(received.Message);
5        ServerTick();
6        CheckSafety();
7        String response = ComposeResponse();
8        udpListener.Reply(response, received.Sender);
9    }
```

Listing 4.1: Główna pętla serwera.

4.3 Komunikacja z systemem wizyjnym - NatNetClient

Obiekt NatNetClient został zaimplementowany w oparciu o bibliotekę NatNetML.dll udostępnioną przez producenta systemu wizyjnego. Klasa podczas podłączania, dzięki wykorzystaniu tzw. delegatów dostępnych w C#, wysyła wskaźnik na funkcję obsługującą komunikację do serwera. Dzięki temu za każdym razem, gdy pojawiają się nowe dane pomiarowe w systemie wizyjnym, wywoływana jest funkcja OnFrameReady(). Funkcja ta sprawdza obecność oraz pozycję markerów, sprawdza czy nie zostały utracone ramki danych oraz oblicza czas, co jaki napływają nowe dane. Odebrane dane są następnie zapisywane w obiekcie BallData.

4.4 Obiekt RobotData

Informacje dotyczące stanu manipulatora są zapisywane w instancji klasy RobotData. We wszystkich klasach programu, których obiekty wchodzą w interakcję z robotem, przekazywany jest obiekt RobotData w postaci referencji. Ten sposób odwołania zapewnia, że zmienne stanu robota są spójne we wszystkich modułach aplikacji. W obiekcie znajdują się informacje o:

- aktualnej, kartezjańskiej pozycji robota,
- aktualnych pozycjach osi robota,
- aktualnych momentach sił na osiach,
- przestrzeni roboczej manipulatora,
- ilości opóźnionych i zgubionych pakietów,
- aktualnym stemplu czasu - IPOC.

Dodatkowo obiekt RobotData przechowuje historię o:

- korektach wysyłanych do robota,
- momentach sił.

Dane te wyświetlane są na wykresach, które pozwalają na szybszą diagnostykę ewentualnych problemów w działaniu.

4.5 Obiekt BallData

Obiekt przechowuje wszystkie dane związane ze stanem piłeczki. Między innymi zawiera informacje o:

- aktualnej kartezjańskiej pozycji piłki w bazie systemu wizyjnego,
- poprzedniej kartezjańskiej pozycji piłki w bazie systemu wizyjnego,
- aktualnej kartezjańskiej pozycji piłki w bazie robota,
- poprzedniej kartezjańskiej pozycji piłki w bazie robota,
- aktualnej kartezjańskiej pozycji piłki obliczonej przez filtr Kalmana w bazie robota,
- poprzedniej kartezjańskiej pozycji piłki obliczonej przez filtr Kalmana w bazie robota,
- aktualnej prędkości piłki obliczonej przez filtr Kalmana w bazie robota,
- poprzedniej prędkości piłki obliczonej przez filtr Kalmana w bazie robota,
- macierzy translacji z układu systemu wizyjnego do układu robota,
- macierzy rotacji z układu systemu wizyjnego do układu robota,
- poprzednich pozycjach piłeczki (historia).

Podobnie, jak klasa RobotData, obiekt BallData przekazywany jest do wszystkich

modułów aplikacji za pomocą referencji.

4.6 Kalibracja układów współrzędnych - CalibrationTool

Zarówno system wizyjny, jak i robot posiadają oddzielne układy współrzędnych. Robot podaje pozycję paletki (tool'a) względem swojego układu *World*, natomiast system wizyjny podaje pozycję piłeczki w układzie zdefiniowanym podczas jego kalibracji. Przejście pomiędzy tymi dwoma systemami było jednym z pierwszych problemów napotkanych podczas realizacji pracy. Zdefiniowanie układu narzędzia w robocie metodą 3 punktową, który pokrywa się z układem współrzędnych robota okazało się nieprecyzyjne. Zdecydowano się na wyznaczenie macierzy transformacji z układu systemu OptiTrack do układu *World* robota. Za pomocą tej macierzy możliwe jest każdorazowe przeliczenie punktu, w którym znajduje się piłka do układu współrzędnych zdefiniowanego na manipulatorze.

Problem sprowadza się do policzenia macierzy rotacji $\mathbf{R}$ i wektora translacji $\vec{t}$, takich, że [9]:

$$\mathbf{B} = \mathbf{R} * \mathbf{A} + \vec{t}, \tag{4.1}$$

gdzie $\mathbf{A}$ to macierz zawierająca współrzędne punktów w układzie systemu wizyjnego, zaś $\mathbf{B}$ to współrzędne tych punktów, po przeliczeniu, w układzie robota.

Aby obliczyć wspomniane wyżej wartości $\mathbf{R}$ i $\vec{t}$, wybrano metodę statystyczną. Proces kalibracji polega na zgromadzeniu zestawu punktów odpowiadających sobie w dwóch układach współrzędnych. W tym celu dokonuje się przejazdu kalibracyjnego z położoną na środku rakietki piłką. Następnie obliczane są centroidy punktów dla układu A oraz B, czyli współrzędne punktu średniego dla wybranych miejsc:

$$P = \begin{bmatrix} x \\ y \\ z \end{bmatrix}, \tag{4.2}$$

$$centroid_A = \frac{1}{N} \sum_{i=1}^{N} P_A^i, \tag{4.3}$$

$$centroid_B = \frac{1}{N} \sum_{i=1}^{N} P_B^i. \tag{4.4}$$

P_A^i i P_B^i to współrzędne kolejnych punktów zawarte w macierzach $\mathbf{A}$ i $\mathbf{B}$.

Następnym krokiem jest wyznaczenie macierzy kowariancji **H**, według wzoru:

$$\mathbf{H} = \sum_{i=1}^{N} (P_A^i - centroid_A)(P_B^i - centroid_B)^T. \tag{4.5}$$

Dokonując dekompozycji SVD macierzy **H** na jej wartości osobliwe uzyskano macierze **U**, **S**, **V**:

$$[\mathbf{U}, \mathbf{S}, \mathbf{V}] = SVD(\mathbf{H}). \tag{4.6}$$

Konieczne jest sprawdzenie wyznacznika macierzy **V**. Jeżeli jest on ujemny, należy przemnożyć trzecią kolumnę tej macierzy przez -1.

Aby uzyskać macierz **R**, przemnaża się uzyskaną macierz **V** i transponowaną macierz **U**:

$$R = \mathbf{V}\mathbf{U}^T. \tag{4.7}$$

Po wyznaczeniu macierzy rotacji **R** można przystąpić do wyznaczania wektora translacji między dwoma układami:

$$\vec{t} = -\mathbf{R} \times centroid_A + centroid_B. \tag{4.8}$$

Znajomość macierzy rotacji **R** oraz wektora translacji $\vec{t}$ pozwala na zapisywanie współrzędnych pochodzących z systemu wizyjnego w układzie współrzędnych manipulatora. Każdy nowy zestaw danych podstawiany jest jako macierz **A** do równania 4.1.

Dokładność algorytmu kalibracji wzrasta wraz ze wzrostem ilości punktów kalibracyjnych. Zadowalającą dokładność (błąd rzędu dziesiętnych części milimetra), otrzymuje się po 25 punktach.

Narzędzie kalibracji zaimplementowane w aplikacji powstało w celu szybkiego wyznaczania macierzy rotacji i translacji. Wyniki algorytmu są automatycznie zapisywane w obiekcie BallData i wykorzystywane podczas zapisywania w nim stanu piłki w przestrzeni.

4.7 Regulator PID

W każdym układzie regulacji trzeba zaimplementować regulator. Ze względu na brak modelu obiektu, zdecydowano się na regulator PID. Składa się on z trzech członów:

- proporcjonalnego (P),
- całkującego (I),
- różniczkującego (D).

Wejściem regulatora jest uchyb, czyli różnica między wartością zadaną a faktycznym stanem obiektu. Wyjścia członów regulatora, które działają na podstawie uchybu, są sumowane i podawane na wyjście regulatora. Każdy z członów regulatora działa inaczej. Człon proporcjonalny skaluje obliczony uchyb. Człon całkujący akumuluje informacje o uchybach w przeszłości i na tej podstawie oblicza swoje wyjście. Różniczkujący przewiduje przyszłe uchyby na podstawie aktualnej prędkości zmian zachodzących w układzie. W przypadku, gdy mierzoną wartością jest pozycja jakiegoś obiektu (np. naszej piłki), człon różniczkujący bada jego prędkość.

Zaimplementowany regulator posiada 4 nastawy:

- K_p - wzmocnienie członu proporcjonalnego,
- K_i - wzmocnienie członu całkującego,
- K_d - wzmocnienie członu różniczkującego,
- N - współczynnik tłumienia dla członu różniczkującego.

Do wyprowadzenia wzorów na dyskretną postać regulatora PID, potrzebna była jego ciągłą postać w dziedzinie Laplace'a:

$$C(s) = K_p + \frac{K_i}{s} + K_d s. \tag{4.9}$$

Ponieważ człon różniczkujący jest bardzo podatny na wszelkiego rodzaju zakłócenia i szumy, postanowiono dodać do niego filtr dolnoprzepustowy (LPF):

$$C(s) = K_p + \frac{K_i}{s} + \frac{NK_d}{1 + \frac{N}{s}}. \tag{4.10}$$

Posiadając transmitancję regulatora w dziedzinie ciągłej, można dokonać transformacji do dziedziny dyskretnej. Przejście można zrealizować np. metodami:

- Eulera w przód,
- Eulera w tył,
- Tustina.

Transmitancje w dziedzinie Z, przybliżone metodą Eulera w tył, wynoszą odpowiednio:

- człon całkujący
$$\frac{K_i T_p z}{z - 1}, \tag{4.11}$$
- człon różniczkujący
$$\frac{K_d N(z - 1)}{(1 + NT_p)z - 1}. \tag{4.12}$$

Cały regulator PID w dziedzinie dyskretnej prezentuje równanie:

$$C(z) = K_p + \frac{K_i T_p z}{z - 1} + \frac{K_d N(z - 1)}{(1 + NT_p)z - 1}. \tag{4.13}$$

Transmitancję w dziedzinie transformaty Z można zapisać za pomocą równań różnicowych [10]:

$$C(z) = \frac{U(z)}{E(z)} = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{a_0 + a_1 z^{-1} + a_2 z^{-2}}, \tag{4.14}$$

gdzie $U(z)$ i $E(z)$ to odpowiednio wyjście regulatora i wartość uchybu. Przekształcając równanie 4.13 do postaci 4.14 otrzymuje się zmienne a i b równe:

$$\begin{aligned} b_0 &= K_p(1 + NT_p) + K_i T_p(1 + NT_p) + K_d N \\ b_1 &= -(K_p(2 + NT_p) + K_i T_p + 2K_d N \\ b_2 &= K_p + K_d N \\ a_0 &= 1 + NT_p \\ a_1 &= -(2 + NT_p) \\ a_2 &= 1 \end{aligned} \tag{4.15}$$

Dalej, przemnażając równanie 4.14 otrzymujemy:

$$a_0 U(z) = -a_1 z^{-1} U(z) - a_2 z^{-2} U(z) + b_0 E(z) + b_1 z^{-1} E(z) + b_2 z^{-2} E(z) \tag{4.16}$$

Na podstawie definicji transformaty Z można zapisać powyższe przekształcenie jako:

$$u[k] = -\frac{a_1}{a_0} u[k-1] - \frac{a_2}{a_0} u[k-2] + \frac{b_0}{a_0} e[k] + \frac{b_1}{a_0} e[k-1] + \frac{b2}{a0} e[k-2], \tag{4.17}$$

gdzie wartości w nawiasach kwadratowych to numery poprzednich próbek.

4.8 Aproksymacja wielomianowa - Polyfit

Do predykcji wektora prędkości piłki podczas zderzenia z rakietką (więcej w sekcji 5.2.1), została użyta funkcja Polyfit. Na podstawie zbioru próbek położenia piłki potrafi ona wygenerować wielomian dowolnego stopnia, którego przebieg będzie zbliżony do położenia tych punktów. Argumentami funkcji są:

- tablica wartości - $y[]$,
- tablica argumentów - $x[]$,
- stopień wielomianu - n,

Polyfit zwraca tablice współczynników wielomianu (od współczynnika przy najniższej potędze do najwyższej). Implementacji funkcji dokonano na podstawie kodu znalezionego w internecie [11].

4.9 Obserwator stanu - filtr Kalmana

Do estymacji stanu układu $\hat{\mathbf{x}} = [x \quad y \quad z \quad \dot{x} \quad \dot{y} \quad \dot{z}]^T$, gdzie x, y oraz z to poszczególne pozycje w osiach, postanowiono użyć filtr Kalmana [12]. Jest to rekursywny algorytm, który pozwala na estymowanie stanu procesu w taki sposób, aby zminimalizować średnią z jej kwadratu błędu. Zakłada się, że szum mierzonego procesu ma rozkład gaussowski.

Filtr składa się z dwóch grup równań dyskretnych: aktualizacji w czasie, która przewiduje stan w chwili k na podstawie chwili k-1 oraz aktualizacji pomiaru, która nakłada korektę na wcześniejszą estymację. Obie grupy równań działają naprzemiennie, z okresem czasu próbkowania.

Równania aktualizacji w czasie:

$$\hat{\mathbf{x}}_k^- = \mathbf{A}\hat{\mathbf{x}}_{k-1} + \mathbf{B}\mathbf{u}_{k-1}, \tag{4.18}$$

gdzie $\hat{\mathbf{x}}_k^-$ to predykcja stanu w chwili k, $\mathbf{u}$ to wejście układu, $\mathbf{A}$ to macierz przejścia, $\mathbf{B}$ to macierz wejścia,

$$\mathbf{P}_k^- = \mathbf{A}\mathbf{P}_{k-1}\mathbf{A}^T + \mathbf{Q}, \tag{4.19}$$

gdzie $\mathbf{P}_k^-$ to macierz kowariancji błędu w chwili k, $\mathbf{Q}$ to macierz kowariancji szumu przetwarzania. Na jej podstawie stwierdzane jest, czy bardziej ufa się danym z pomiaru czy wartościom estymowanym. Macierz $\mathbf{Q}$ została wyznaczona doświadczalnie.

Równania aktualizacji pomiaru:

$$\mathbf{K}_k = \mathbf{P}_k^-\hat{\mathbf{H}}^T(\hat{\mathbf{H}}\mathbf{P}_k^-\hat{\mathbf{H}}^T + \hat{\mathbf{R}})^{-1}, \tag{4.20}$$

gdzie $\mathbf{K}_k$ to macierz wzmocnień Kalmana, $\hat{\mathbf{H}}$ to macierz wyjścia, $\hat{\mathbf{R}}$ to macierz kowariancji błędu pomiaru,

$$\hat{\mathbf{x}}_k = \hat{\mathbf{x}}_k^- + \mathbf{K}_k(\mathbf{z}_k - \hat{\mathbf{H}}\hat{\mathbf{x}}_k^-), \tag{4.21}$$

gdzie $\hat{\mathbf{x}}_k$ to stan układu w chwili k, $\mathbf{z}_k$ to wartość pomiaru,

$$\mathbf{P}_k = (\mathbf{I} - \mathbf{K}_k\hat{\mathbf{H}})\mathbf{P}_k^-, \tag{4.22}$$

gdzie $\mathbf{P}_k$ to macierz kowariancji błędu w chwili k po uwzględnieniu pomiaru.

Powyższe równania zostały zaimplementowane w klasie KalmanFilter. Jej obiekty utworzono dla każdej z osi układu współrzędnych. Dla wszystkich 3 obiektów przyjęto macierze o następujących wartościach:

$$\mathbf{A} = \begin{bmatrix} 1 & dt & dt^2/2 \\ 0 & 1 & dt \\ 0 & 0 & 1 \end{bmatrix}, \tag{4.23}$$

$$\mathbf{B} = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}, \tag{4.24}$$

$$\hat{\mathbf{H}} = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}, \tag{4.25}$$

$$\mathbf{Q} = \begin{bmatrix} 0.5 & 0 & 0 \\ 0 & 1000 & 0 \\ 0 & 0 & 1000 \end{bmatrix}, \tag{4.26}$$

gdzie dt to czas próbkowania (w tym przypadku 4ms). Ze względu na to, że dla każdej osi istnieje osobny filtr Kalmana, wektor stanu wygląda następująco:

$$\hat{\mathbf{x}} = \begin{bmatrix} q_x \\ \dot{q_x} \\ \ddot{q_x} \end{bmatrix}, \tag{4.27}$$

gdzie q_x to pozycja w odpowiadającej osi. Zależności pomiędzy zmiennymi stanu uwzględniono w macierzy przejścia $\mathbf{A}$. Pozostałe macierze $\mathbf{R}$, $\mathbf{I}$, na początku działania algorytmu są macierzami jednostkowymi.

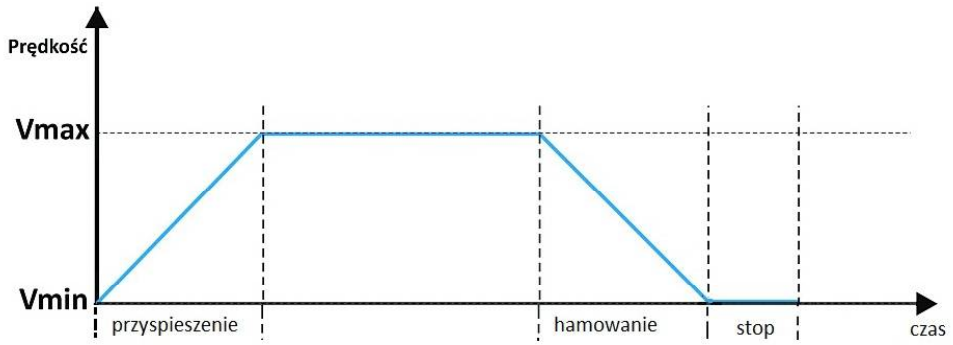
4.10 Sterownik ruchu

Pakiet RSI firmy Kuka pozwala na wysyłanie rozkazów ruchu bezpośrednio do kontrolera napędów. W tak niskopoziomowej realizacji trzeba samemu zapewnić, że wysłane do robota korekty ruchu nie wpłyną na uszkodzenia jego mechaniki np. poprzez wprowadzenie go w drgania. W przypadku niepoprawnego sterowania, zadania za dużej prędkości lub przyspieszenia, kontroler robota gwałtownie zatrzymuje się, zaciskając hamulce na każdym ze złączy oraz wyświetla komunikat przeciążenia momentu siły na silniku lub przekładni (Motor Torque Error lub Gear Torque Error)

Pierwszym sposobem kontroli ruchu było generowanie tzw. ramp. Prędkość końcówki robota była stopniowo zwiększana, aby uzyskać małą stromość na wykresie prędkości, który zamieszczono na rysunku 4.2.

Taki sposób rozpędzania manipulatora okazał się jednak niezadowalający w działaniu. Problematyczne okazało się wyznaczanie stromości i długości rampy.

Zdecydowano się napisać bardziej złożony sterownik ruchu, który generowałby trajektorię, po której ma poruszać się robot za pomocą wielomianu 3 stopnia. Stopień wielomianu został dobrany tak, aby zawsze jego pierwsza pochodna



Rysunek 4.2: Rampa prędkości podczas przyspieszania i hamowania [13].

(prędkość) była ciągła. Pomysł na zastosowanie trajektorii wielomianowych zaczerpnięto z literatury [14].

Obiekt w programie odpowiedzialny za liczenie trajektorii zapisany jest w klasie TrajectoryGenerator. Jako argumenty wejściowe, potrzebne do wygenerowania funkcji pozycji, przyjmuje:

- pozycję startową - x_p,
- czas ruchu - t,
- początkową prędkość - V_{xp},
- końcową prędkość - V_{xk},
- pozycje końcową - x_p.

Na podstawie tych danych obliczane są współczynniki wielomianu (a_0 ... a_3) przy jego kolejnych potęgach:

$$a_0 = x_p, \tag{4.28}$$

$$a_1 = V_{xp}, \tag{4.29}$$

$$a_2 = \frac{3 * x_k - 2 * V_{xp} * t - 3 * x_p - V_{xk} * t}{t^2}, \tag{4.30}$$

$$a_3 = \frac{V_{xk} * t + V_{xp} * t - 2 * x_k + 2 * x_p}{t^3}. \tag{4.31}$$

Znając współczynniki wielomianu oraz aktualny numer próbki (czas od rozpoczęcia) możliwe jest wyznaczenie miejsca, gdzie powinno się znaleźć narzędzie robota (Tool Center Point) po następnym kwancie czasu (4 ms):

$$x(t) = a_3 t^3 + a_2 t^2 + a_1 t + a_0. \tag{4.32}$$

W celu uzyskania korekty relatywnej, od docelowej pozycji robota odejmuje się pozycję aktualną, którą następnie wysyła się do robota:

$$correction_x = x(t) - x_p. \tag{4.33}$$

Oprócz obliczenia pozycji robota konieczne jest wyznaczenie jego prędkości w punkcie. Otrzymany wielomian 4.32 różniczkuje się otrzymując funkcję 2 stopnia:

$$V_x(t) = 3a_3t^2 + 2a_2t + a_1. \tag{4.34}$$

Podstawiając za jej argument czas próbkowania uzyskuje się prędkość w kolejnej korekcie ruchu robota.

Współczynniki wielomianu są przeliczane co cykl biblioteki RSI, jest to konieczne ze względu to, że próbki sterujące ruchem robota nie są ciągłą krzywą, lecz odcinkami składającymi się na krzywą (spróbkowany wielomian). Za każdym razem, gdy wyznaczany jest wielomian za pozycję startową x_p podstawiana jest aktualna pozycja robota.

Przedstawiony powyżej sposób generowania trajektorii został zastosowany dla każdej z osi kartezjańskiego układu współrzędnych oraz dla kątów A, B oraz C, które odpowiadają za orientację narzędzia robota wokół odpowiednio osi Z, Y oraz X.

Zastosowanie funkcji wielomianowej sprawiło, że ruchy robota stały się bardziej płynne. Przyśpieszenie oraz hamowanie rozkładają się w sposób jednakowy na cały ruch. Ponadto możliwe jest zadanie dokładnej prędkości narzędzia manipulatora w określonej pozycji co jest kluczowe w zagadnieniu odbijania piłki. Dodatkowo rozwiązanie wyeliminowało problem przekroczeń momentów sił na poszczególnych złączach.

4.11 Sterownik bezpieczeństwa

Ze względu na to, że pakiet RSI, z którego korzystamy działa w bardzo niskiej warstwie programowej kontrolera (przełożenie wysłanych korekt bezpośrednio na sterownik napędów), większość z funkcji bezpieczeństwa dostępnych w systemie KRC4 nie jest w stanie zareagować na potencjalnie niebezpieczne sytuacje. Z tego względu powstał obiekt SafetyController, który przed każdym wysłaniem korekty do robota sprawdza czy manipulator będzie znajdował się w położeniu bezpiecznym. Przyjęto następujące limity w przestrzeni kartezjańskiej (względem układu bazowego) oraz kątowe dla poszczególnych złączy:

$$550mm > X > -550mm$$
$$1300mm > Y > 550mm$$
$$750mm > Z > -100mm$$
$$-20^\circ > A1 > -150^\circ$$
$$40^\circ > A2 > -130^\circ$$
$$185^\circ > A4 > -185^\circ$$
$$115^\circ > A5 > -100^\circ$$

Limity w osiach X, Y oraz Z zapewniają, że manipulator nie wyjedzie poza swój określony obszar roboczy, natomiast limity na złączach (A1 ... A6) chronią robota przed uderzeniem narzędziem w samego siebie. Ze względu na brak zagrożenia, na złączach A3 oraz A6 nie ustawiono limitów.

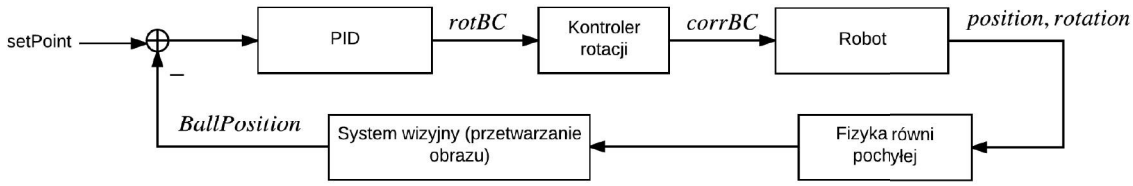
Rozdział 5

Realizacja założonych zadań

W tym rozdziale został omówiony sposób realizacji zadania utrzymywania piłki w punkcie płaszczyzny oraz zadania utrzymywania piłki w powietrzu poprzez ciągłe podbijanie.

5.1 Utrzymywanie piłki w punkcie płaszczyzny

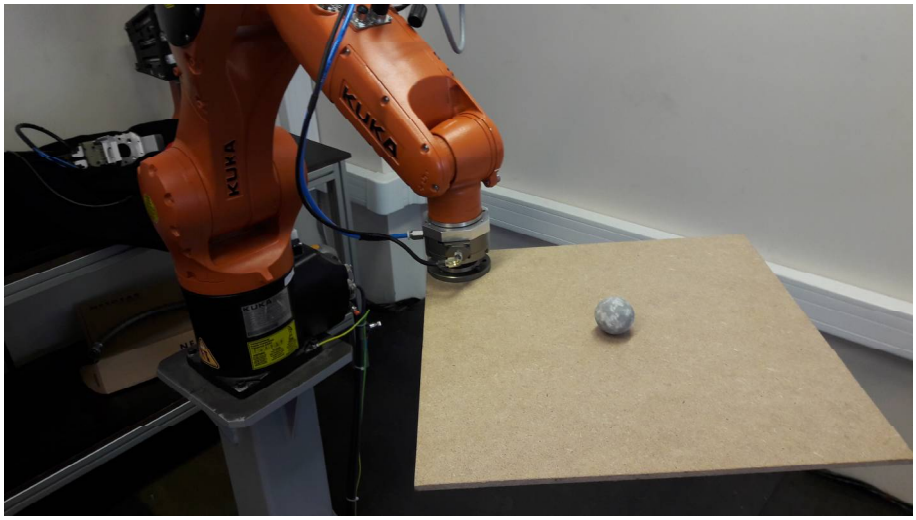
Na rysunku 5.1 przedstawiono schemat układu regulacji automatycznej dla zadania utrzymywania piłki w punkcie na płaszczyźnie.



Rysunek 5.1: Schemat układu regulacji.

5.1.1 Algorytm ruchu

Zagadnienie utrzymywania piłki w punkcie na płaszczyźnie potraktowane zostało jako układ dwóch niezależnych równi pochyłych. Obracając płaszczyznę wokół dwóch osi X, Y, można zadać miejsce, w które przemieści się piłka. W systemie rotacyjny ruch narzędzia (w tym przypadku jest to deska o wymiarach 400mm x 400mm przedstawiona na rysunku 5.2) realizowany jest poprzez wysyłanie do manipulatora korekt dwóch zmiennych kartezjańskich: B oraz C, które określają odpowiednio obrót wokół osi Y i X układu światowego, którego początek znajduje się w TCP (Tool Center Point).

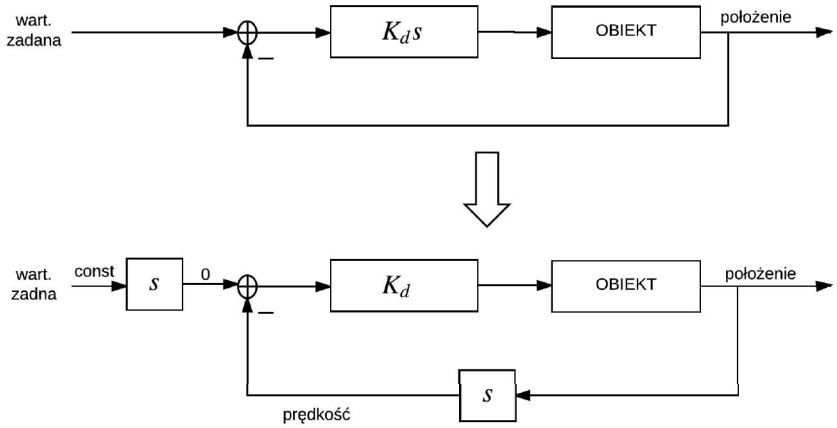


Rysunek 5.2: Robot realizujący zagadnienie stabilizacji piłki w punkcie płaszczyzny.

Funkcja utrzymująca piłkę zrealizowana została wykorzystując wcześniej zaimplementowany regulator PID. Utworzono w programie dwie instancje tego obiektu. Zadaniem każdej była regulacja położenia piłki w jednej osi. Wyjściem regulatorów są kąty, o które należy obrócić płaszczyznę, aby sprowadzić piłkę do ustalonego punktu. Jako, że wynik otrzymany z kontrolera PID powinien być realizowany natychmiastowo, nie można tu wykorzystać generatora trajektorii opisanego wcześniej, ponieważ wymaga on podania czasu dla realizacji ruchu. Należy pamiętać, że wysłanie zbyt dużej korekty rotacyjnej do manipulatora może spowodować wystąpienie błędów przeciążeń przekładni i silników. Sprawdzono więc maksymalne wartości korekty wokół osi narzędzia, które można wysłać do nieporuszającego się robota. Następnie napisano prosty algorytm, który porównuje wynik regulatora z poprzednią, wysłaną korektą. Jeżeli różnica jest większa od wartości maksymalnej (co sugeruje możliwość wystąpienia błędów), algorytm zmniejsza wynik regulatora do wartości maksymalnej i dopiero wtedy wysyła go do robota. Algorytm działa na zasadzie regulatora z nasyceniem. Cały opisany algorytm został zaimplementowany w klasie - *BalanceBall*.

5.1.2 Strojenie nastaw regulatora PID

Dobór nastaw regulatora PID dla sterownika utrzymującego piłkę na płaszczyźnie został ustalony na podstawie wielu przeprowadzonych eksperymentów. Początkowo przeprowadzono testy dla regulatora proporcjonalnego. Zaobserwowano, że bez względu na wielkość wzmocnienia K_p, amplituda oscylacji układu się zwiększała.



Rysunek 5.3: Regulator prędkości.

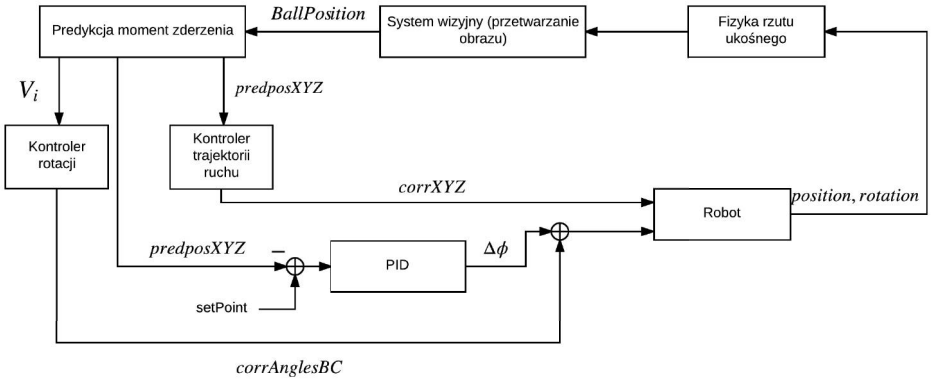
W kolejnych próbach użyto jedynie regulatora różniczkującego, który ustawia wyjście na podstawie szybkości zmian położenia piłki, czyli jej aktualnej prędkości. Otrzymane wyniki były o wiele lepsze. Przy nastawie K_d rzędu 0.09, piłka stabilizowała się po wychyleniu z punktu równowagi, tzn. zatrzymywała w miejscu na płaszczyźnie. Punkt, w którym stawała, był jednak oddalony od punktu podanego jako wartość zadana. Powodem, był brak wzmocnienia w członie proporcjonalnym. Regulator obserwował jedynie prędkość piłki a nie jej pozycje (regulator prędkości - rysunek 5.3). Niewielkie zmniejszenie wzmocnienia członu różniczkującego i zwiększenie wzmocnienia członu proporcjonalnego znacznie poprawiło sytuacje. Piłka po paru oscylacjach stabilizuje się w obszarze zadanego punktu. Część całkująca regulatora pozostawała wyłączona przez cały czas. Ostatnia nastawa - N, czyli tłumienia dla członu różniczkującego została dobrana tak, aby ruchy robota były jak najbardziej płynne, z zachowaniem poprawnego działania tego członu.

Ostatecznie przyjęte nastawy dla zadana stabilizowania piłki na płaszczyźnie:

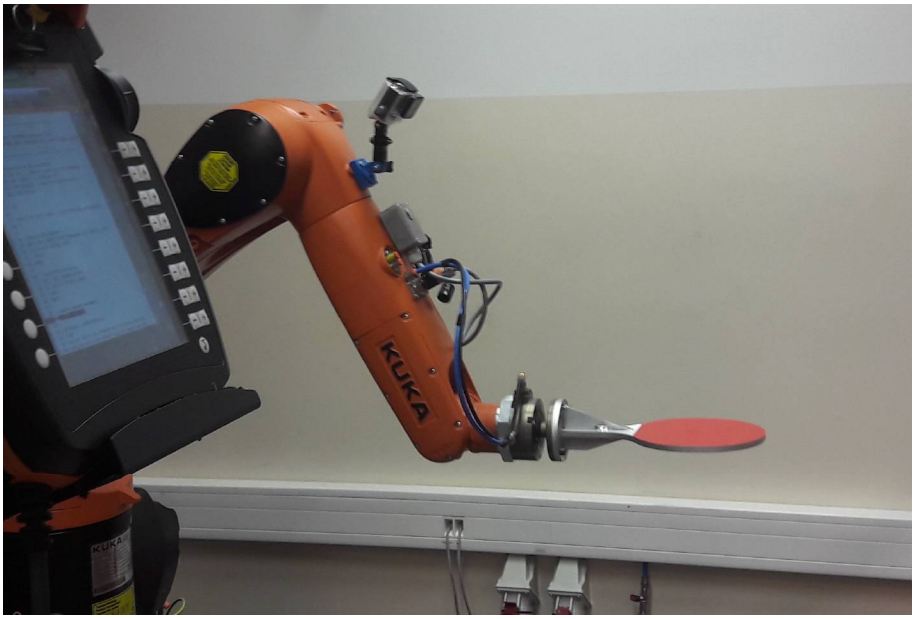
- wzmocnienie członu proporcjonalnego - $K_p = 0.05$,
- wzmocnienie członu całkującego - $K_i = 0$,
- wzmocnienie członu różniczkującego - $K_d = 0.0718$,
- współczynnik tłumienia N = 0.159.

5.2 Utrzymywanie piłki w powietrzu poprzez ciągłe podbijanie

Na rysunku 5.4 przedstawiono schemat układu regulacji automatycznej dla zadania utrzymywania piłki w powietrzu poprzez ciągłe podbijanie. Rysunek 5.5 przedstawia robota z dołączoną aluminiową rakietką w miejscu narzędzia.



Rysunek 5.4: Schemat układu regulacji dla zadania podbijania piłki.



Rysunek 5.5: Robot realizujący zagadnienie odbijania piłki.

5.2.1 Predykcja pozycji i prędkości piłki w momencie zderzenia z rakietką

W aplikacji założono, że robot będzie zawsze odbijał piłkę na ustalonej wysokości. Do obliczenia pozycji i prędkości piłki, w momencie osiągnięcia zadanej wysokości, wykorzystano dwie zaimplementowane przez nas metody:

- obserwator Kalmana,
- wpasowywanie w trajektorię lotu wielomianu n-tego stopnia.

Obserwator Kalmana

Przy wykorzystaniu filtru Kalmana znany jest wektor prędkości i pozycja w aktualnej chwili. Pozostałe potrzebne wartości można uzyskać wykorzystując prawa fizyki działające w rzucie ukośnym:

$$crossZ = V_z * timeToHit - \frac{g * timeToHit^2}{2}. \tag{5.1}$$

Najpierw należy obliczyć czas:

$$timeToHit = \frac{V_z + \sqrt{V_z^2 + 2g * pos_z - crossZ}}{g}, \tag{5.2}$$

$timeToHit$ - czas do zderzenia,
V_z - aktualna prędkość w osi Z,
pos_z - aktualna pozycja w osi Z,
$crossZ$ - wysokość na której odbijamy piłkę,
g - przyśpieszenie ziemskie.

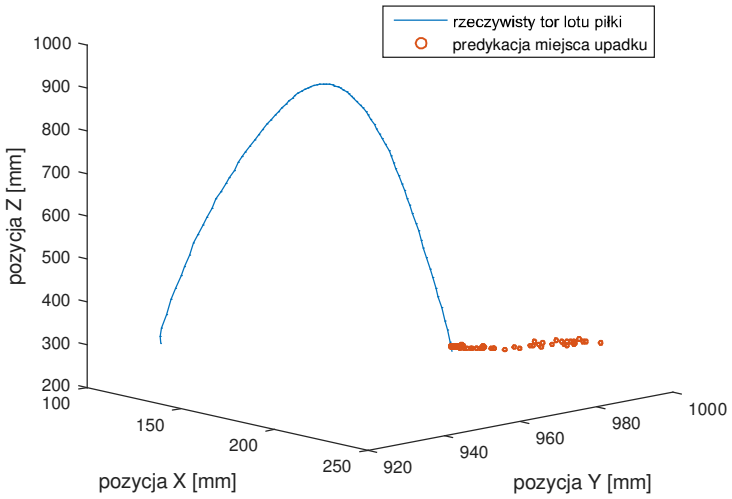
Znając $timeToHit$ i wykorzystując wzory na drogę w ruchu przyśpieszonym można obliczyć miejsce zderzenia:

$$predpos_x = timeToHit * V_x + pos_x, \tag{5.3}$$

$$predpos_y = timeToHit * V_y + pos_y, \tag{5.4}$$

$predpos_x$ - przewidywana współrzędna X miejsca zderzenia,
$predpos_y$ - przewidywana współrzędna Y miejsca zderzenia.

Przewidywane pozycje piłki na początku lotu dość mocno odbiegają od miejsca jej faktycznego upadku. Dlatego powinno się je przeliczać dla każdego nowego pomiaru. Im piłka jest bliżej ustalonej wysokości ($crossZ$), tym predykcje są dokładniejsze. Na rysunku 5.6 niebieskim kolorem zaznaczony jest rzeczywisty tor lotu piłki, a czerwonym predykowane punkty na płaszczyźnie X-Y, w których piłka się znajdzie, gdy osiągnie ustaloną wysokość ($crossZ$).



Rysunek 5.6: Predykcja miejsca upadku piłki na płaszczyźnie przy pomocy filtru Kalmana.

Prędkości V_x i V_y w rzucie ukośnym są stałe, dlatego za prędkości w tych osiach w momencie zderzenia można podstawić wartości aktualne, a V_z^i (prędkość w osi Z w chwili zderzenia) oblicza się następująco:

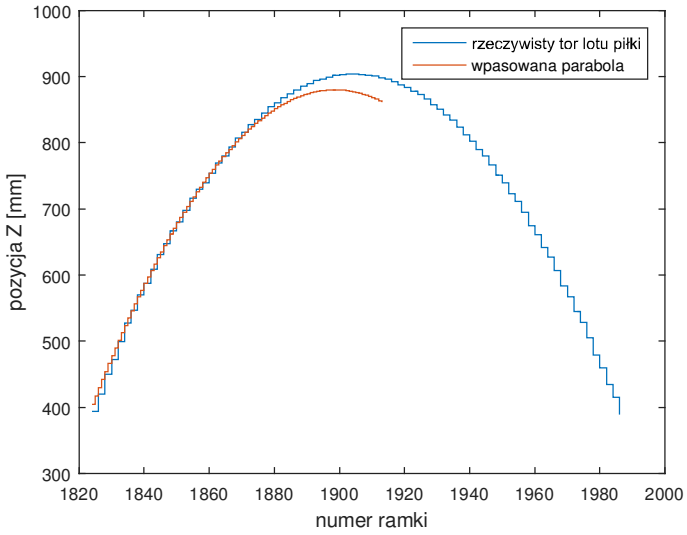
$$V_z^i = V_z - g * timeToHit. \tag{5.5}$$

Uzyskane prędkości są potrzebne do ustawiania kątów paletki podczas odbicia.

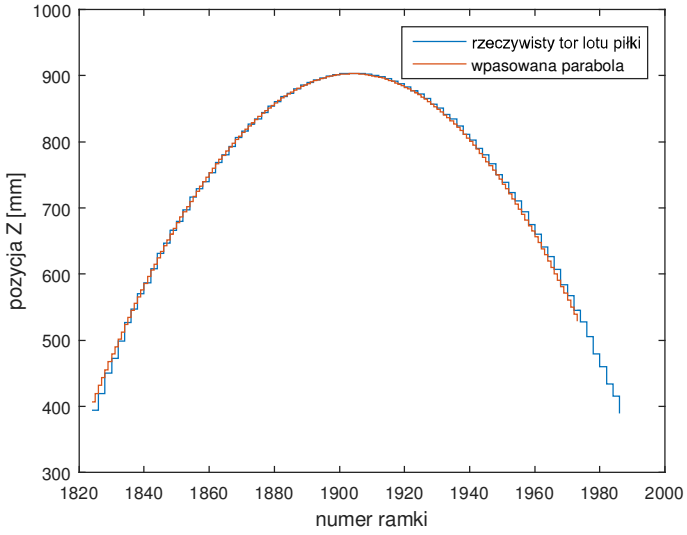
Wpasowywanie wielomianu n-tego stopnia

Inna metoda predykcji pozycji i prędkości piłki polega na wpasowaniu wielomianu w kolejne próbki, pochodzące z systemu wizyjnego, zawierające informacje o pozycji piłki. Ze względu na to, że tor lotu w osi Z przypomina parabolę, zastosowano wielomian 2-go stopnia. W aplikacji, po zgromadzeniu więcej niż 25 próbek, co ramkę danych, wywoływana zostaje funkcja polyfit, która zwraca współczynniki dopasowanej paraboli.

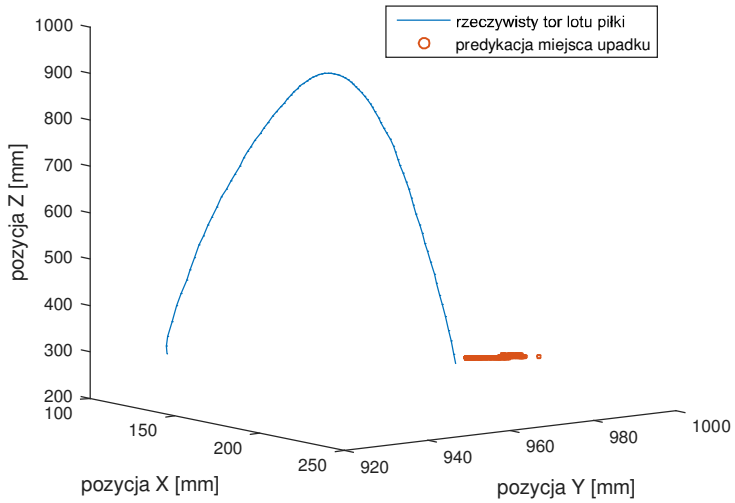
Jak widać na wykresach (Rysunki 5.7 i 5.8), im więcej zgromadzonych pomiarów pozycji piłki, tym wpasowana parabola, przedstawiona kolorem czerwonym, jest bardziej zbliżona do faktycznego toru lotu piłki, zaznaczonego kolorem niebieskim.



Rysunek 5.7: Wykres wpasowanej paraboli dla współczynników wielomianu obliczonych po zgromadzeniu 35 próbek.



Rysunek 5.8: Wykres wpasowanej paraboli dla współczynników wielomianu obliczonych po zgromadzeniu 125 próbek.



Rysunek 5.9: Predykcja miejsca upadku piłki na płaszczyźnie przy pomocy wpasowanego wielomianu.

Na podstawie otrzymanej funkcji można obliczyć kiedy piłka osiągnie ustaloną wysokość:

$$\Delta = b^2 - 4a(c - crossZ), \tag{5.6}$$

$$frameNum = \frac{-b - \sqrt{\Delta}}{2a}, \tag{5.7}$$

a, b i c - współczynniki wpasowanej paraboli.

Wynikiem ($frameNum$) jest numer ramki, w której nastąpi zderzenie, gdyż to on jest argumentem obliczonej paraboli. W odczytane pozycje piłki w osiach X i Y wpasowywane są wielomiany, 1-go stopnia, ponieważ tor lotu piłki w nich przypomina linię prostą. Aby obliczyć pozycję X i Y ($predpos_x$ i $predpos_y$), w chwili osiągnięcia zadanej wysokości, należy podstawić wartość $frameNum$ do uzyskanych (za pomocą funkcji polyfit) funkcji liniowych. Rysunek 5.9 przedstawia rzeczywisty tor lotu piłki (kolor niebieski) oraz predykowane punkty (kolor czerwony) na płaszczyźnie X-Y, w których piłka będzie na wysokości równej $crossZ$.

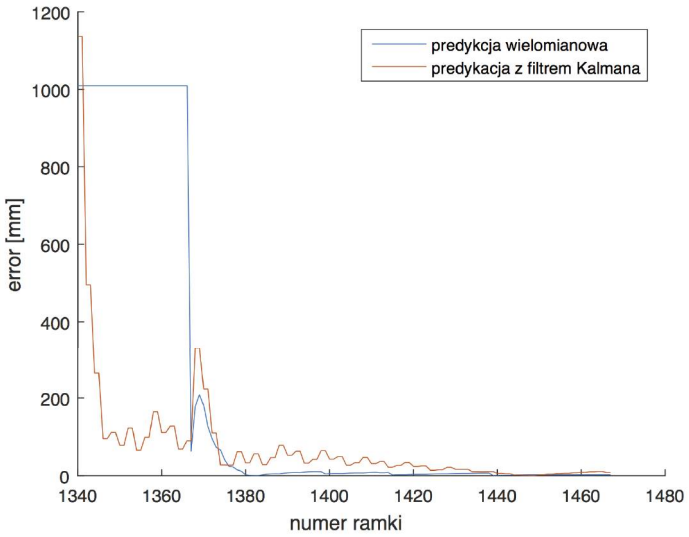
Kolejnym krokiem jest obliczenie prędkości w momencie zderzenia. Do tego potrzebne są pochodne trzech otrzymanych wielomianów (funkcji kwadratowej i dwóch funkcji liniowych). Za argument podstawiana jest obliczona w równaniu (5.6) wartość $frameNum$.

Porównanie metod predykcji pozycji i prędkości

Obie metody predykcji miejsca upadku i prędkości zostały przeanalizowane w środowisku MatLab. Na rysunku 5.10 oraz 5.11 pokazano przebieg zmian błędu predykcji miejsca upadku piłki podczas jej lotu. W pierwszej części lepsze przybliżenie uzyskuje się stosując metodę wpasowywania wielomianu (kolor niebieski). Przedstawiony wykres ma mniejsze wartości błędu i posiada mniejsze oscylacje. W początkowych kilkunastu próbkach jest stały ponieważ prawidłowe działanie tego sposobu predykcji wymaga zgromadzenia około 25 próbek danych.

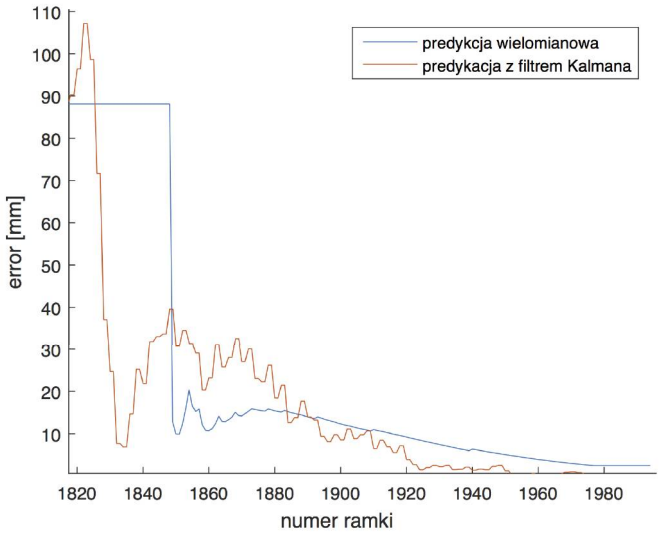
W końcowej fazie lotu piłki sytuacja zmienia się. Przybliżenie z zastosowaniem obserwatora Kalmana (kolor czerwony) jest dokładniejsze (błąd pokazany na wykresie jest mniejszy). Średnica paletki w najwęższym miejscu wynosi 151 mm, czyli jest parokrotnie większa od błędu mniej dokładnego sposobu predykcji. Użycie obu metod spowoduje że robot trafi w piłkę rakietką, jednak przy użyciu filtru Kalmana piłka będzie się odbijać bliżej środka narzędzia.

Porównując wykresy można również zauważyć, że dla pierwszego rzutu początkowe aproksymacje cechują się dużym błędem, rzędu tysięcy mm. W przypadku n-tego odbicia predykowane miejsce upadku piłki w pierwszej fazie lotu nie odbiega już tak mocno od miejsca rzeczywistego.



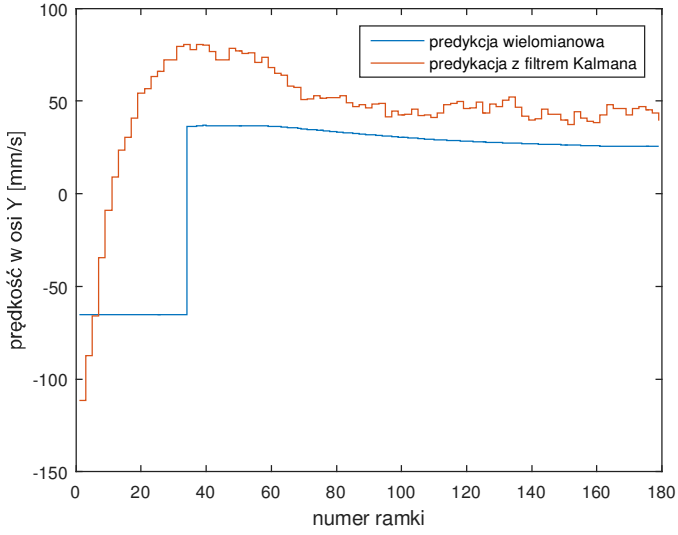
Rysunek 5.10: Porównanie dwóch metod predykcji miejsca upadku piłki dla pierwszego rzutu.

Na rysunku 5.12 przedstawiona jest predykowana wartość prędkości piłki w osi Y w momencie zderzenia z rakietką dla dwóch metod aproksymacji. Wartości uzyskane przy pomocy filtru Kalmana (kolor czerwony) mocno oscylują i są większe. Z kolei przebiegi otrzymane z wykorzystaniem wpasowanego wielomianu

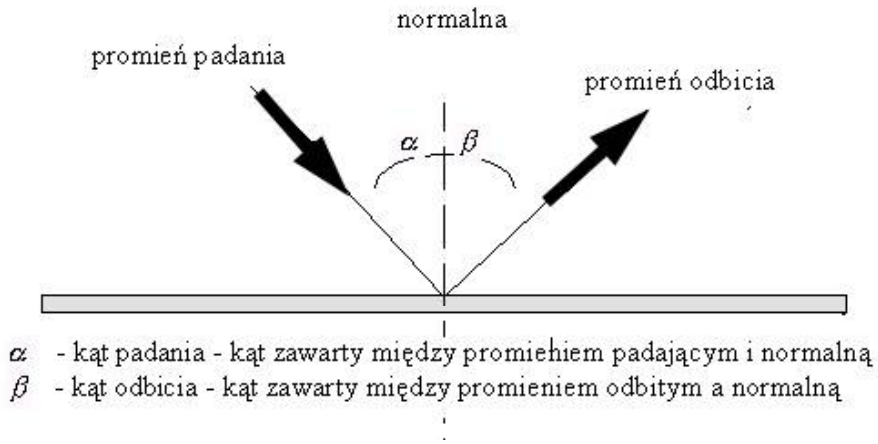


Rysunek 5.11: Porównanie dwóch metod predykcji miejsca upadku piłki dla n-tego odbicia.

(kolor niebieski) są łagodniejsze. Na podstawie testów, gdy robot podbijał piłkę, zauważono, że gdy do predykcji prędkości wykorzystuje się metodę wielomianową robot lepiej ustawia orientacje rakietki podczas odbicia (piłka nie wylatuje poza obszar roboczy). Dlatego metoda ta została uznana za dokładniejszą.



Rysunek 5.12: Wartość prędkości piłki w osi Y w chwili zderzenia dla dwóch metod aproksymacji.



Rysunek 5.13: Prawo odbicia [16].

5.2.2 Obliczanie nastaw kątów paletki

Podczas każdego odbicia kontroler stara się obliczyć i ustawić orientacje paletki (narzędzia robota) tak, aby odbiła się idealnie ku górze (bez prędkości w osiach X i Y). Do wyliczenia potrzebnych kątów, skorzystano z prawa odbicia twierdzącego [15], że kąt padania równy jest kątowi odbicia (rysunek 5.13)

Zasadę tą można opisać równaniem:

$$\vec{d} = \vec{V_i} - 2(\vec{V_i}^T * \vec{n})\vec{n}, \tag{5.8}$$

gdzie $\vec{d}$ - wektor odbicia, $\vec{V_i}$ - wektor prędkości piłki w chwili zderzenia, $\vec{n}$ - wektor normalny paletki. Aby wyznaczyć wzór na wektor normalny paletki, przekształcono równanie 5.8 do postaci:

$$\vec{n} = \frac{\vec{d} - \vec{V_i}}{\sqrt{2(\vec{V_i}^T * \vec{V_i} - \vec{V_i}^T \vec{d})}}. \tag{5.9}$$

Do wyznaczenia wartości wektora $\vec{n}$

$$\vec{n} = \begin{bmatrix} n_x \\ n_y \\ n_z \end{bmatrix}, \tag{5.10}$$

przy założeniu, że piłka powinna się odbić prosto w górę, za $\vec{d}$ i $\vec{V_i}$ przyjęto:

$$\vec{d} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}, \vec{V_i} = \begin{bmatrix} V_x \\ V_y \\ V_z \end{bmatrix}. \tag{5.11}$$

Należy pamiętać, że wektor $\vec{V_i}$ powinien zostać podany w postaci znormalizowanej, czyli jego długość powinna być równa 1. Dalej, żeby wyznaczyć kąty rotacji rakietki, wokół osi X i Y układu narzędzia, skorzystano z funkcji $atan2$:

$$B = atan2(n_x, n_z), \tag{5.12}$$

$$C = atan2(n_y, n_z). \tag{5.13}$$

5.2.3 Strojenie nastaw regulatora PID

Proces podbijania zaczyna się od rzutu piłki w kierunku robota przez osobę stojącą poza jego obszarem roboczym. Podczas każdego odbicia robot stara się ustawić orientacje paletki tak, aby piłka odbiła się prosto ku górze. Niedokładności obliczeń (spowodowane np. brakiem dokładnego modelu ruchu) powodują, że po kilku takich odbiciach piłka wylatuje poza obszar roboczy manipulatora. Zdecydowano się na zastosowanie regulatora PI dla osi X oraz Y, który ma za zadanie sprowadzać piłkę za każdym odbiciem w kierunku środka obszaru pracy (czyli punktu o współrzędnych: x = 0, y = 910). Każdy z regulatorów wpływa na zmianę rotacji narzędzia wokół innej osi (kąty B i C). Uchybem sterowania jest różnica między współrzędnymi punktu, w którym chcemy odbijać piłkę, a przewidywanym miejscem jej upadku.

Nastawy regulatorów zostały dobrane na drodze dużej ilości testów. Ich ostateczne wartości są następujące:

- wzmocnienie członu proporcjonalnego $K_p = 0.0075$,
- wzmocnienie członu całkującego - $K_i = 0.001$,
- wzmocnienie członu różniczkującego - $K_d = 0$.

Wzmocnienie dla członu całkującego jest niewielkie, ale odgrywa duże znaczenie. Człon ten gromadzi w sobie uchyby i gdy piłka odbija się dłuższy czas z dala od zadanego miejsca, coraz bardziej stara się ją sprowadzić do środka obszaru roboczego.

5.2.4 Algorytm ruchu robota

Algorytm ruchu robota podczas podbijania piłki składa się z 3 stanów, które wykonywane są w pętli (Rysunek 5.14):

- jugglestate = 1 - stan oczekiwania,
- jugglestate = 2 - ruch do piłki,
- jugglestate = 3 - uderzenie piłki.

Dodatkowo zadeklarowane zostały następujące zmienne globalne (tzw. zmienne ruchu):

- destVelocityXYZ - końcowa prędkość narzędzia,
- destinationPoint - końcowa pozycja narzędzia,
- timeToDest - czas do końca ruchu.

W zależności od tego, w którym stanie kontroler się znajduje, wykonywane są inne funkcje. Każda z nich w inny sposób oblicza wartości dla zmiennych ruchu.

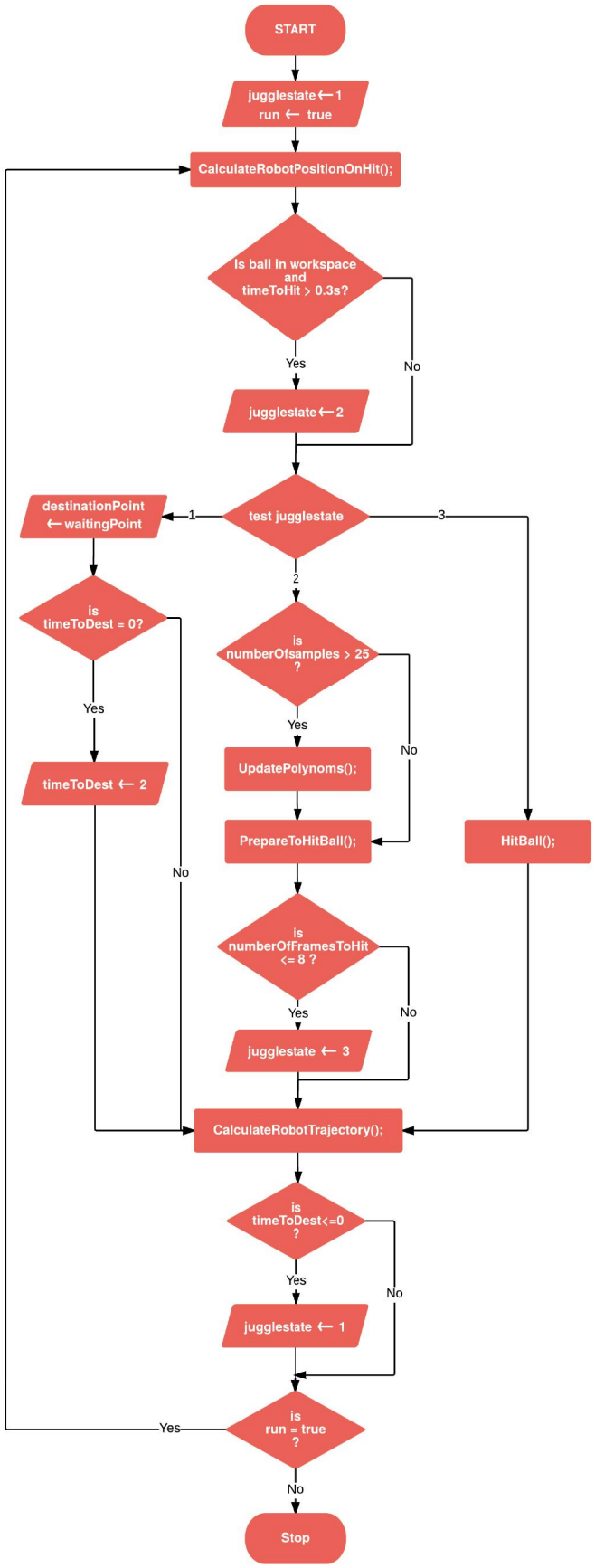
Na starcie każdej iteracji pętli wywoływana jest metoda *CalculateRobotPositionOnHit()*, z kolei na końcu - *CalculateRobotTrajectory()*. Pierwsza oblicza, przy pomocy danych z filtru Kalmana, pozycje i orientacje narzędzia robota w momencie uderzenia (metoda liczenia została opisana w sekcji 5.2.1). Druga, na podstawie zmiennych ruchu oblicza ścieżkę, po której robot będzie się poruszał i korekty wysyłane na sterownik manipulatora.

W momencie startu algorytmu znajduje się on w stanie pierwszym. Zmienne ruchu są tak ustawiane, aby robot jechał do ustalonej pozycji oczekiwania. Jeżeli się w niej znajdzie, będzie stał w bezruchu.

Warunkiem przejścia do stanu drugiego (ruch do piłki) jest pojawienie się piłki w obszarze roboczym robota oraz czas dojazdu do miejsca odbicia (obliczony przez *CalculateRobotPositionOnHit()*) większy niż 0.3 s. Zakłada się, że gdyby czas ten był mniejszy nie udałoby się jej skutecznie odbić. Kiedy warunki zostaną spełnione, algorytm przechodzi do stanu drugiego, zostaje wywołana funkcja *UpdatePolynoms()*. W przypadku, gdy tor lotu piłki, od momentu zmiany stanu został zarejestrowany przez więcej niż 25 próbek, wygenerowany zostaje wielomian wpasowany w ruch piłki, na podstawie którego liczony jest wektor prędkości piłki w chwili zderzenia (podrozdział 5.2.1) oraz orientacja rakietka w chwili odbicia. Wyliczone wartości nadpisują wyniki obliczone przez funkcje *CalculateRobotPositionOnHit()*. Jeżeli próbek jest mniej, wcześniejsze wartości pozostają bez zmian. Na podstawie przeprowadzonych testów i analiz w środowisku MatLab zauważono, że gdy zgromadzone jest więcej niż 20-25 próbek, metoda wpasowywania wielomianu w tor lotu dokładniej przybliża prędkości piłki niż filtr Kalmana. Dlatego, gdy próbek jest mniej, algorytm korzysta z filtru do predykcji orientacji rakietki w chwili zderzenia z piłką, a jeżeli więcej, to przełącza się na predykcje na podstawie wpasowanych wielomianów. Przewidywane miejsce upadku piłki i czas, po którym to nastąpi, przez cały czas brane są z filtru Kalmana. Kolejną metodą wywoływaną w stanie drugim jest *PrepareToHit()*. Podstawia ona za zmienne ruchu, przewidywane czasy i miejsce upadku piłki. Ze względu na to, że niemożliwe obliczenie dokładnie kiedy piłka uderzy w rakietkę, na osiem ramek danych przed planowanym uderzeniem kontroler przechodzi do stanu trzeciego. Nie analizuje wtedy lotu piłki, zamraża wcześniej wyliczone wartości i przez

kolejne 16 ramek danych porusza narzędzie w górę z założoną prędkością odbicia równą 600 mm/s. Wartość prędkości została dobrana na podstawie wielu przeprowadzonych testów. Zbyt duża prędkość powodowała podbijanie piłki na co raz większe wysokości, system wizyjny gubił ją wtedy z pola widzenia. Za mała zaś sprawiała, że piłka zbyt szybko opadała na wysokość odbicia i robot nie miał wystarczająco dużo czasu, aby wykonać do niej ruch. Podczas eksperymentów zauważono, że istnieje pewna strefa nieczułości w zależności prędkości odbicia od wysokości, na którą piłka się wznosi. Przy prędkościach odbicia rzędu 500-600 mm/s piłka osiągała przy każdym odbiciu podobne wysokości. Jej energia ani nie rosła ani nie malała.

Po szesnastu ramkach danych, kiedy piłka powinna się odbić, algorytm powraca do stanu pierwszego, a jeżeli spełnione są warunki obszaru roboczego i obliczonego czasu do uderzenia, to automatycznie następuje przejście do stanu drugiego.



Rysunek 5.14: Schemat blokowy przedstawiający działanie algorytmu podbijania piłki.

Rozdział 6

Obsługa aplikacji Resilio

6.1 Ekran startowy - Menu

Po uruchomieniu aplikacji na ekranie pojawia się okienko startowe podzielone na dwie kategorie. Pierwsza - *Main Menu* zawiera funkcje konfiguracyjne systemu. Za ich pomocą można kolejno (rysunek 6.1):

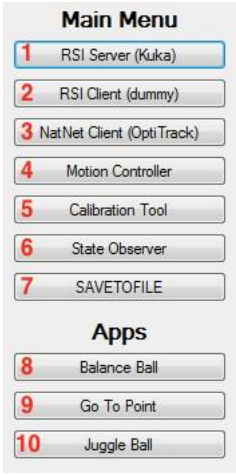
- utworzyć serwer dla połączenia RSI z robotem (1),
- nawiązać połączenie z symulatorem klienta RSI (2),
- nawiązać połączenie z systemem wizyjnym (3),
- podejrzeć wykresy aktualnych korekt oraz prądów i momentów na silnikach (4),
- podejrzeć wykresy pozycji rzeczywistej piłki i pozycji obliczonej za pomocą filtru Kalmana (5),
- skalibrować ze sobą system współrzędnych wizji i robota (6),
- zapisać logi aplikacji do pliku tekstowego (7).

W drugiej części okna startowego znajdują się przyciski, którymi uruchamia się aplikacje realizujące zadania:

- utrzymywanie piłki na paletce (8),
- jazda do punktu (9),
- podbijanie piłki (10).

6.2 Nawiązanie połączenia z robotem

Po kliknięciu przycisku *RSI Server (Kuka)* pojawia się okienko ustawień połączenia RSI (rysunek 6.2). Można tutaj włączyć/wyłączyć serwer, ustawić limity obszaru roboczego dla robota, oraz odczytać jego aktualną i relatywną pozycję. Na dole ekranu znajduję się także informacja o czasach odpowiedzi serwera oraz



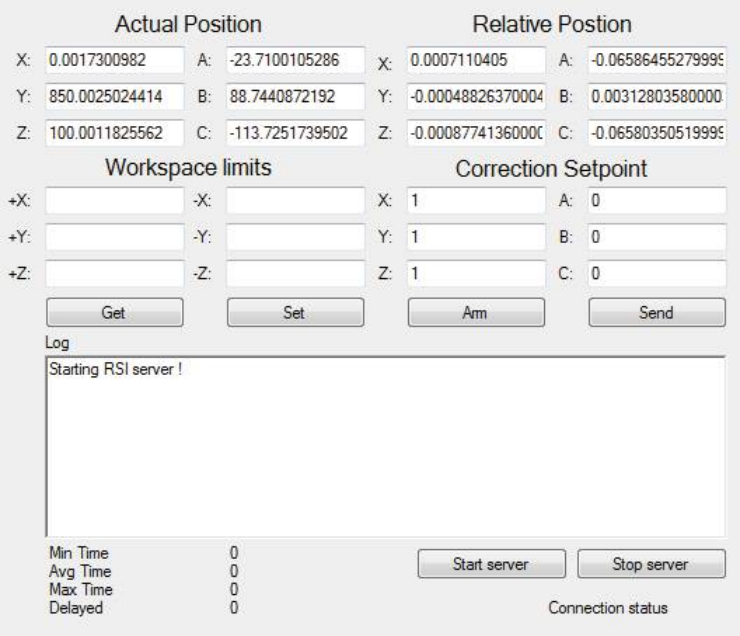
Rysunek 6.1: Ekran startowy aplikacji Resilio.

ilość ramek, które zostały wysłane za późno do robota.

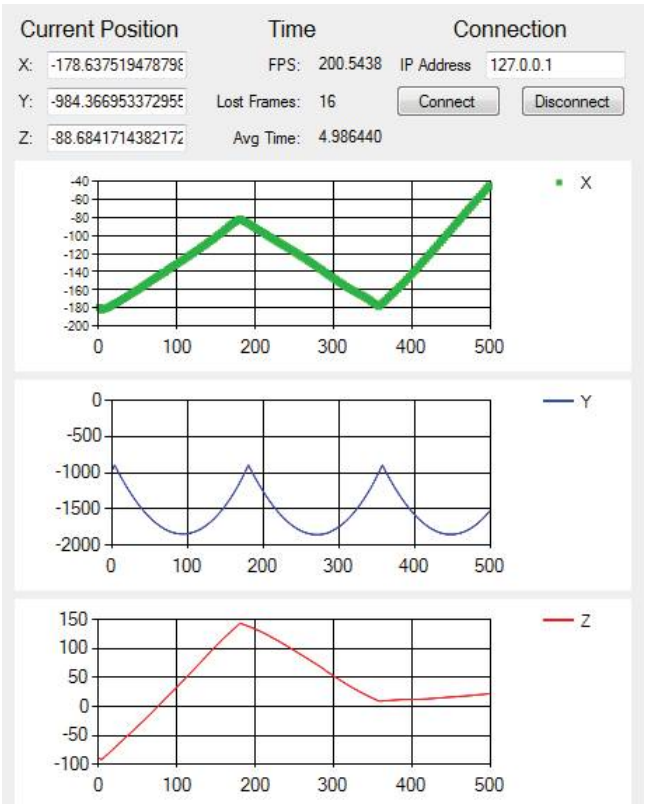
Aby nawiązać połączenie RSI między komputerem PC a manipulatorem, należy najpierw włączyć serwer w aplikacji, a potem, poprzez program *RSI_Ethernet.src* znajdujący się w kontrolerze robota, utworzyć klienta, który połączy się z serwerem. Wspomniany program uruchamia się poprzez wybranie go opcją *Select*. Po dojeździe do pierwszego punktu PTP następuje utworzenie wątka RSI i zestawienie komunikacji z serwerem.

6.3 Nawiązanie połączenia z systemem wizyjnym

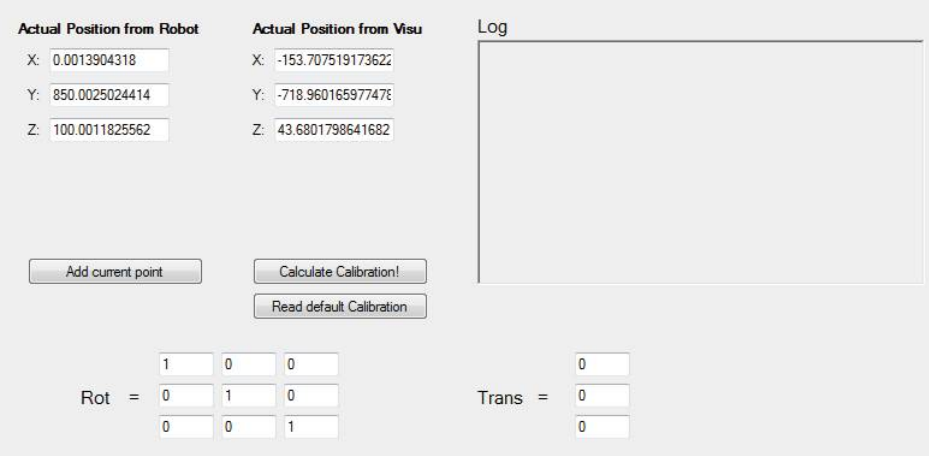
Do współpracy z systemem wizyjnym OptiTrack potrzebna jest uruchomiona aplikacja - Motive. W ustawieniach transmisji powinna być włączona opcja przesyłania danych do lokalnego hosta (loopback) poprzez protokół NatNet. Aby aplikacja Resilio mogła odbierać informacje z systemu wizyjnego, trzeba uruchomić z głównego menu opcje - *NatNet Client* i kliknąć przycisk nawiązujący połączenie. W oknie (rysunek 6.3), które się pojawi można odczytać aktualną pozycje obiektu widzianego przez system wizyjny oraz za pomocą wykresów przeanalizować zmiany pozycji w każdej z osi. Znajduje się tutaj także informacja o częstotliwości pracy systemu OptiTrack oraz ilości ramek danych, które z różnych powodów nie dotarły do klienta.



Rysunek 6.2: Ekran połączenia RSI z robotem.



Rysunek 6.3: Ekran połączenia aplikacji z systemem wizyjnym (osie pionowe wykresów pokazują położenie piłki [mm] względem odpowiedniej osi układu systemu wizyjnego, osie poziome przedstawiają numer ramki danych.

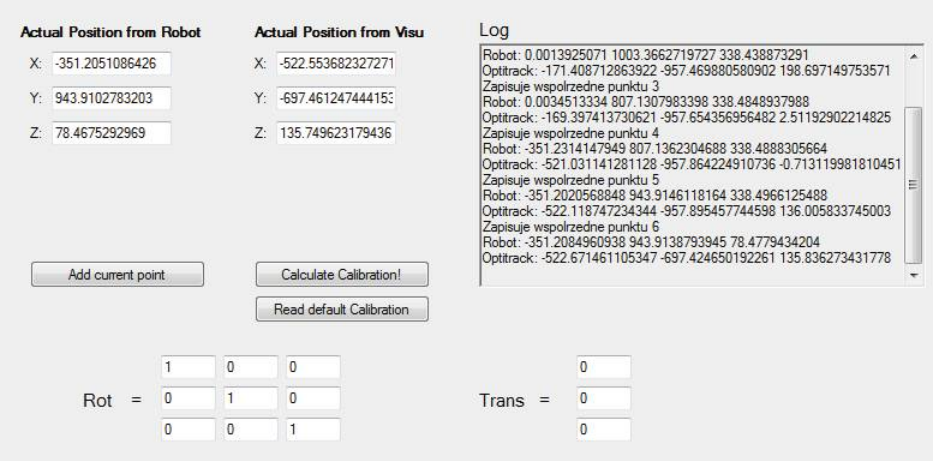


Rysunek 6.4: Ekran kalibracji. Na rysunku widać, że współrzędne w układzie systemu wizyjnego różnią się od tych w układzie robota.

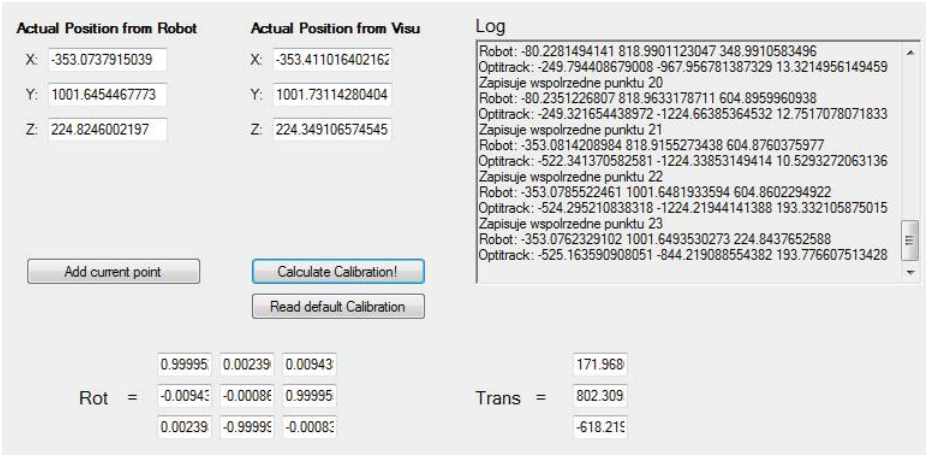
6.4 Kalibracja układów współrzędnych

Kalibracja układów współrzędnych polega na przemieszczaniu ramienia manipulatora z piłką położoną na środku paletki (Tool Center Point) i zapisywanie losowych punktów w układzie współrzędnych systemu wizyjnego i robota. Po kliknięciu przycisku *Calibration Tool* na ekranie startowym aplikacji pojawia się okno kalibracji. Znajdują się tam informacje o położeniu piłki w każdym z systemów (rysunek 6.4). Aby zapisać punkt kalibracyjny należy kliknąć przycisk *Add current point* (rysunek 6.5). Gdy zostanie zgromadzona odpowiednia ilość pomiarów, użycie przycisku *Calculate Calibration* wygeneruje wartości w macierzy rotacji ($\mathbf{R}$) i wektorze translacji ($\vec{t}$), które zostały pokazane u dołu okna (rysunek 6.6). Od tego momentu pozycja piłki w układzie robota i systemu wizyjnego powinna być bardzo zbliżona do siebie (akceptowalne są niedokładności rzędy dziesiątej części milimetra). Trzeba pamiętać, że większa ilość punktów zapewni lepszą dokładność. W przypadku dużych rozbieżności miedzy współrzędnymi, można dodać kilka punktów, a następnie jeszcze raz obliczyć macierz $\mathbf{R}$ i wektor $\vec{t}$.

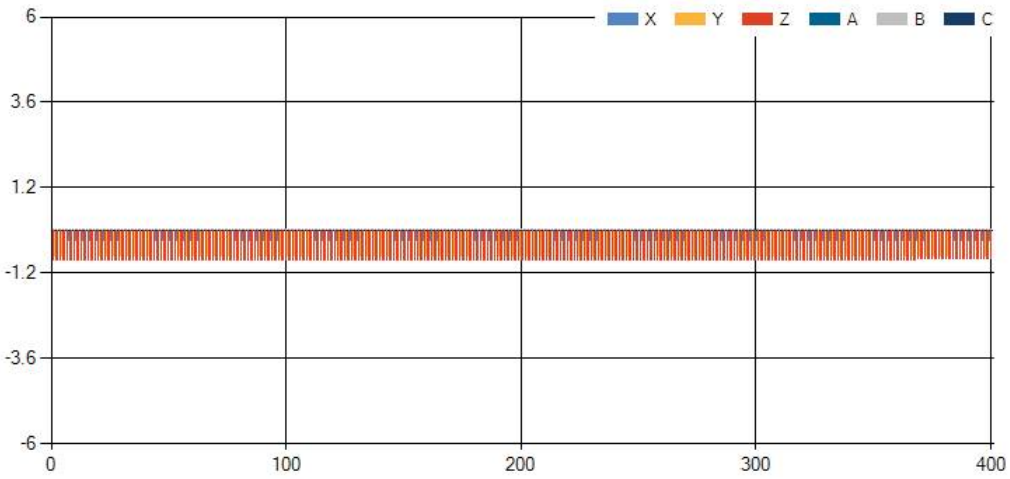
Zadowalającą dokładność, uzyskano przy 30 punktach pomiarowych.



Rysunek 6.5: Ekran kalibracji po zapisaniu sześciu punktów.



Rysunek 6.6: Ekran kalibracji po przeliczeniu macierzy rotacji i wektora translacji. Współrzędne piłki w obu układach są bardzo zbliżone do siebie.



Rysunek 6.7: Wykres korekt wysyłanych do robota(oś pionowa - wartość korekty [mm], oś pozioma - numer ramki danych) .

6.5 Wykresy analizy ruchu

Po kliknięciu *Motion Controller* w oknie startowym aplikacji na ekranie pokażą się dwa wykresy:

- wykres korekt wysyłanych do robota (Rysunek 6.7)
- wykres momentów sił na przekładniach napędów (Rysunek 6.8)

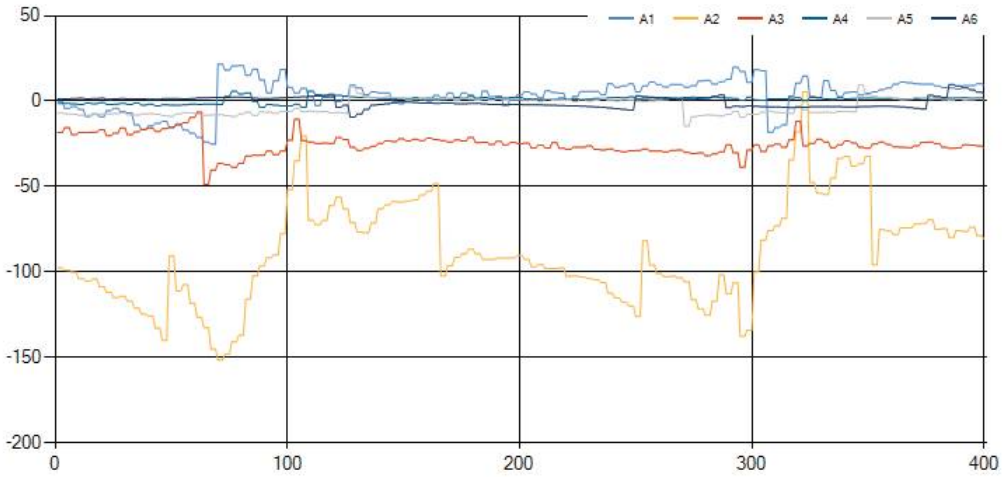
6.6 Funkcje programu

6.6.1 Ruch do punktu

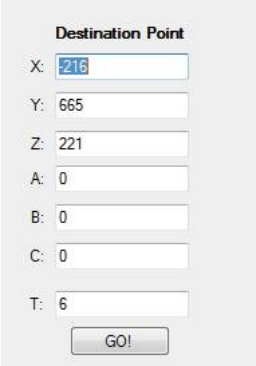
Ruch końcówki robota, od punktu do punktu, realizowany jest po włączeniu aplikacji *Go To Point*. Na ekranie, który się pojawia (rysunek 6.9), trzeba wpisać współrzędne (X,Y,Z) punktu, do którego narzędzie ma dojechać oraz orientacje (A,B,C), która ustali się na końcu ruchu. Ostatnim parametrem jest czas. Ruch do punktu odbywa się po wyliczonej trajektorii wielomianowej, od podanej zmiennej czasowej zależy, jak szybko robot osiągnie cel. Ruch następuje w momencie naciśnięcia przycisku *GO!*. Należy pamiętać, że podanie małego czasu spowoduje obliczenia trajektorii o bardzo dużych przyśpieszeniach. W takiej sytuacji mogą wystąpić błędy przeciążenia osi.

6.6.2 Utrzymywanie piłki w punkcie na płaszczyźnie

Aplikacja *Balance Ball* realizuje zadanie utrzymywania piłki w punkcie na płaszczyźnie. Po jej uruchomieniu, na ekranie (rysunek 6.10) pokazuje się okno z



Rysunek 6.8: Wykres momentów sił na przekładniach napędów (oś pionowa - procentowa wartość obciążenia złącza, oś pozioma - numer ramki danych).

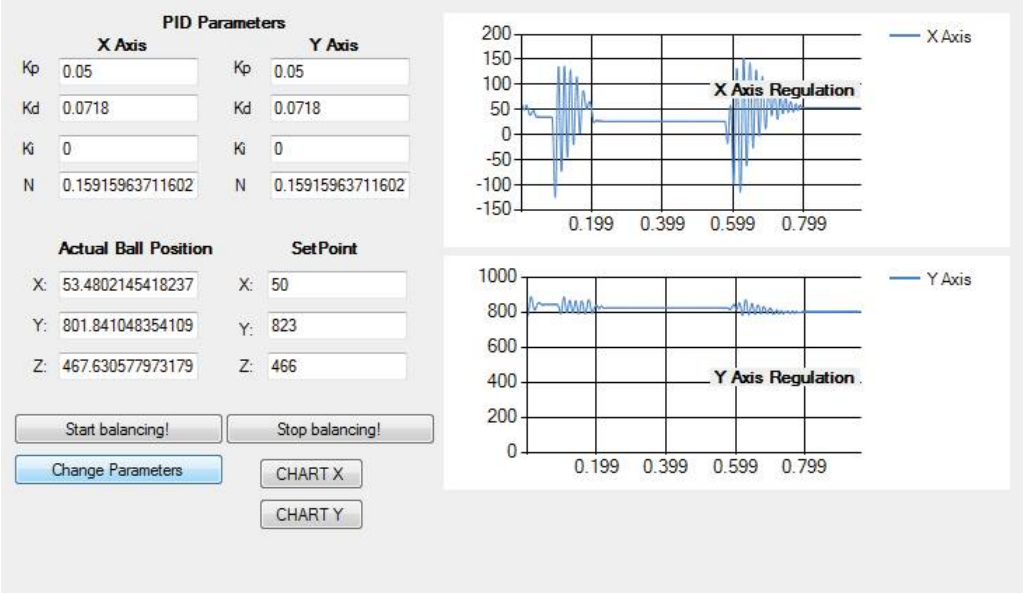


Rysunek 6.9: Ekran jazdy do punktu aplikacji Resilio.

parametrami ustawień regulatorów PID dla balansowania w osiach X i Y. Wykresy znajdujące się po prawej stronie obrazują zmiany uchybów w czasie. Uruchomienie aplikacji następuję w momencie kliknięcia przycisku *Start balancing*. Nastawy regulatorów można zmieniać podczas działania. Wpisane wartości należy potwierdzać przyciskiem *Change Parameters*.

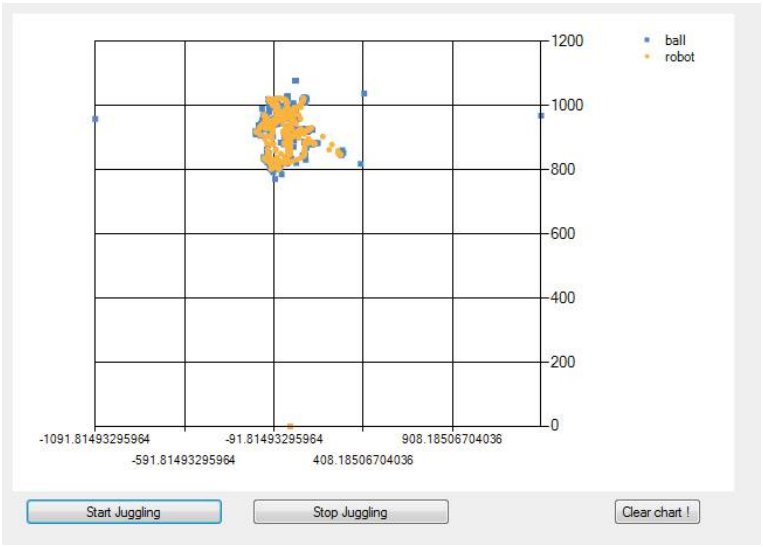
6.6.3 Podbijanie piłki

Aplikacja odpowiedzialna za realizacje podbijania piłki nazywa się *Juggle Ball*. Po jej uruchomieniu (poprzez ekran startowy aplikacji Resilio), pojawia się nowe okno z przyciskami sterującymi oraz wykresem obrazującym położenie (w mm) piłki i narzędzia robota na płaszczyźnie X-Y (rysunek 6.11). W momencie kliknięcia przycisku *Start* robot przesunie się do pozycji oczekiwania na piłkę i stanie w bezruchu. Kiedy nadejdzie informacja od systemu wizyjnego o obecności piłki



Rysunek 6.10: Okno sterujące utrzymywaniem piłki w punkcie na płaszczyźnie.

aplikacja zacznie przeliczać jej trajektorie ruchu. Jeżeli zostaną spełnione warunki odbicia (więcej o nich w sekcji 5.2.2), robot wykona ruch, aby odbić piłkę. Rzut piłki w kierunku robota powinno się wykonać z miejsca, w którym jest ona niewidoczna dla systemu wizyjnego, ponieważ ruchu piłki w ręce nie można opisać za pomocą równań rzutu ukośnego.



Rysunek 6.11: Okno sterujące zadaniem podbijania piłki (oś pionowa wykresy - położenie w osi X [mm], oś pozioma - położenie w osi Y [mm]).

Rozdział 7

Osiągnięte rezultaty

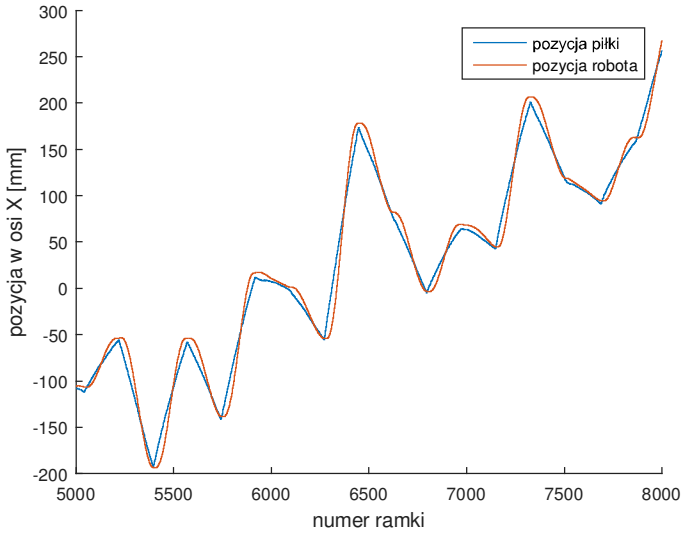
7.1 Utrzymywanie piłki w punkcie na płaszczyźnie

Problemem w realizacji tego zadania było zaprojektowanie kontrolera rotacji przetwarzającego wartości wychodzące z regulatora PID. Korekty wysyłane do robota nie mogą być zbyt duże bo grozi to pojawieniem się błędów przeciążeń silników i przekładni. Jednocześnie nie można zaprojektować trajektorii dla tego ruchu gdyż wartości pojawiające się na wyjściu regulatora PID powinny być natychmiast podawane na wejście obiektu (robota z płaską deską) aby układ doprowadzić do stabilności. Zaimplementowany regulator wraz z kontrolerem stabilizują piłkę w pobliżu punktu. Zdarza się jednak, że ruchy robota nie są płynne, następują szarpnięcia podczas wykonywania korekt.

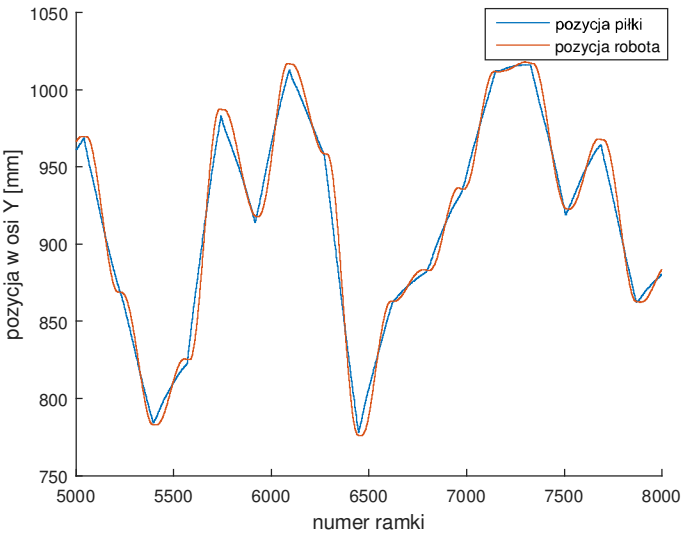
7.2 Podbijanie piłki

Kluczowym etapem projektu było zaimplementowanie generatora trajektorii, który zapewnia płynny ruch i gwarantuje, ze narzędzie robota znajdzie się na zadanej pozycji w dokładnie określonym momencie czasu. Na rysunkach 7.1, 7.2, 7.3 przedstawiona została pozycja piłki (kolor niebieski) i robota (kolor czerwony) w każdej osi. Ich pozycje są bardzo zbliżone do siebie. Manipulator śledzi cały czas piłkę i stara się dostosować do jej współrzędnych. Miejsca, w których piłka zmienia kierunek lotu, to miejsca odbić. Pomimo, że zmiana ruchu piłki jest natychmiastowa, droga robota jest w tych miejscach wygładzona.

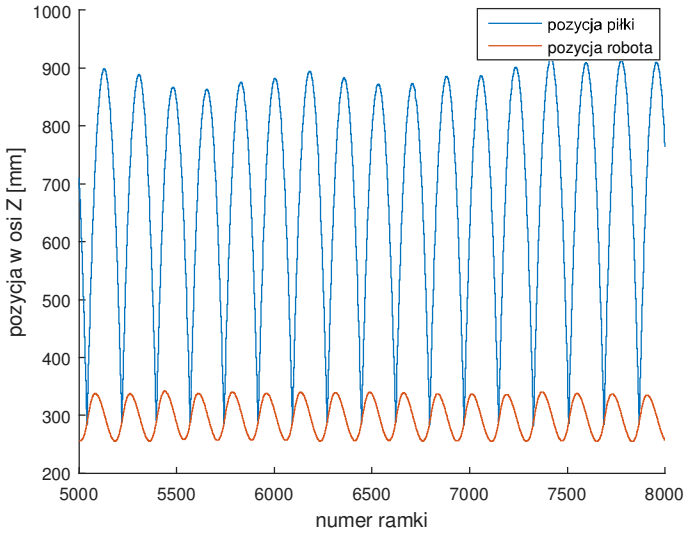
Z rysunku 7.3 można odczytać dokładnie, na jakiej wysokości piłka się odbiła. Teoretycznie powinno to nastąpić zawsze na wysokości ($crossZ$) równej 250 mm.



Rysunek 7.1: Pozycja piłki i robota w osi X.



Rysunek 7.2: Pozycja piłki i robota w osi Y.

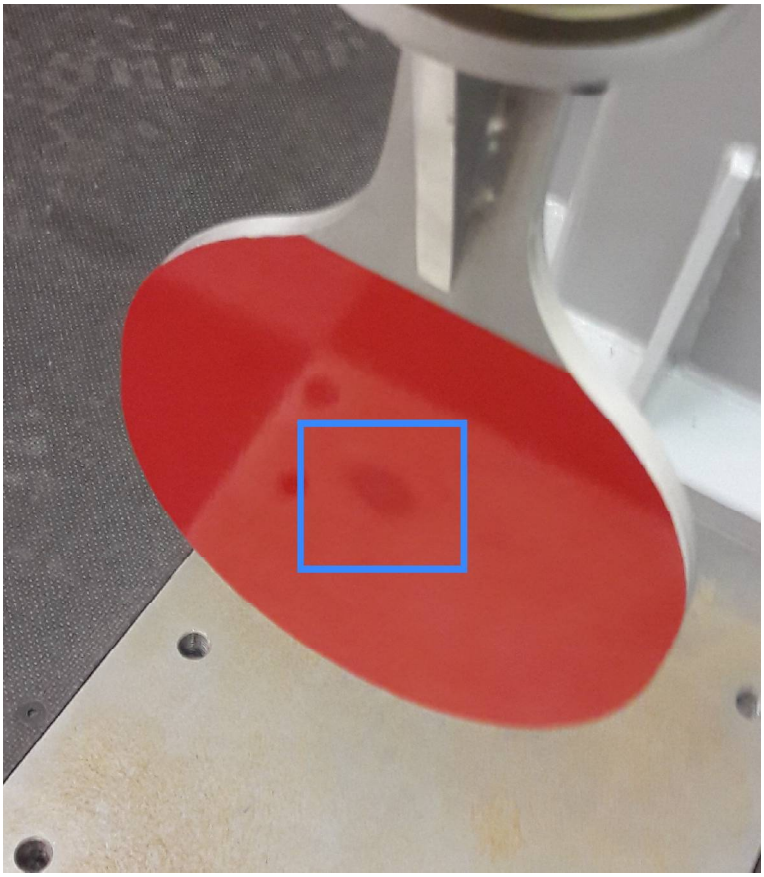


Rysunek 7.3: Pozycja piłki i robota w osi Z.

Rzeczywista wartość różni się nieznacznie. Jest to spowodowane niedokładnością pomiarów pozycji piłki i braku jej dokładnego modelu ruchu. Aby ten problem rozwiązać, algorytm ruchu jest tak zaprojektowany, żeby zadana prędkość paletki (600 mm/s) utrzymywana była przez 16 ramek (8 próbek przed teoretycznym odbiciem i 8 po). Dzięki temu zwiększono tolerancje na błąd związany ze złą predykcją momentu osiągnięcia przez piłkę określonej wysokości.

Warto również zauważyć (rysunek 7.3), że po każdym odbiciu piłka wznosi się na bardzo podobną wysokość. Moment zderzenia w niewielkim stopniu wypływa na zmianę jej energii całkowitej.

Na rysunku 7.4 widać rakietkę, na której jest ślad po odbijaniu. Jego średnica wynosi 20mm, czyli jest dwa razy mniejsza od średnicy piłki. Pomimo braku idealnych pomiarów oraz dokładnego modelu piłki, odbicie następuje zawsze bardzo blisko punktu środka rakietki. Całkowity rozmiar narzędzia można więc zmniejszyć siedmiokrotnie i nie powinno to stanowić problemu dla realizacji zadania.



Rysunek 7.4: Ślad na rakietce po odbijaniu piłki.

Rozdział 8

Podsumowanie pracy

8.1 Podsumowanie

Praca nad systemem prowadzona była na przestrzeni sześciu miesięcy. Podzielona została na etapy składające się z coraz trudniejszych i ambitniejszych zadań. Początkowo najważniejsze było zintegrowanie ze sobą dostępnych urządzeń, czyli komunikacja między robotem, systemem wizyjnym i aplikacją na komputerze PC. Dopiero kiedy to było gotowe, można było zacząć realizację zadania utrzymywania piłki na płaszczyźnie, a następnie jej podbijania.

Ostatecznie udało się zrealizować wszystkie wyznaczone cele. Trzeba przyznać, że praca była wymagająca i poświęcono na nią dużo czasu. Ostatecznie osiągnięte wyniki przerosły oczekiwania twórców. Rezultat rekompensuje czas oraz wysiłek zainwestowany w projekt.

Pomimo, że realizowane zadanie nie jest przydatne z punktu widzenia świata przemysłu, to porusza bardzo ważną kwestię. Roboty mogą działać jeszcze efektywniej. Ich zadaniem nie musi być jedynie powielanie takiego samego ruchu wiele razy, ale także realizowanie prac bardziej wymagających, w których zachodzi potrzeba dostosowania się do bieżącej sytuacji.

8.2 Dalszy rozwój projektu

Zrealizowane zadania cechuje stabilność. Piłka podczas podbijania utrzymywana jest w obszarze roboczym, lecz nie jest odbijana zawsze w jednym punkcie. Robot czasem musi wykonać większy ruch aby ją odbić. Sytuacja prawdopodobnie polepszyłaby się po zaimplementowaniu dokładnego modelu piłki. Obliczenia rotacji paletki w chwili zderzenia byłyby wtedy dokładniejsze. Aby uzyskać lepsze rezultaty, konieczna byłaby zmiana piłki na niepokrytą pyłem refleksyjnym.

Wiąże się to jednak ze zmianą systemu wizyjnego.

Poza tym, projekt posiada duży potencjał i może być rozwijany o wiele funkcji, na przykład:

- współpraca dwóch robotów - przerzucanie piłki ponad ścianą celi,
- odbijanie piłki o ścianę,
- rytmiczne odbijanie piłki (metronom),
- odbijanie piłki z człowiekiem,
- utrzymywanie piłki na zadanej wysokości.

W dalszych pracach można też zbadać zależność wysokości lotu piłki w funkcji prędkości uderzenia. Jest to konieczne w przypadku problemu celowania piłką w punkt w przestrzeni.

Bibliografia

[1] Federal Ministry for Economic Affairs and Energy. What is industrie 4.0? http://www.plattform-i40.de/I40/Navigation/EN/Industrie40/WhatIsIndustrie40/what-is-industrie40.html, 2017. data dostępu: 2017-01-01.

[2] KUKA Roboter GmbH. *KR Agilus sixx - Specification*. KUKA, Augsburg, 2015.

[3] Inc. NaturalPoint. OptiTrack opis sprzętowy systemu. http://wiki.optitrack.com/index.php?title=Hardware_Setup, 2017. data dostępu: 2017-01-01.

[4] NextStage Pro. Rysunek 2.3. http://www.nextstagepro.com/retroreflective-markers.html, 2017. data dostępu: 2017-01-01.

[5] glowtec.co.uk. Rysunek 2.4. http://www.glowtec.co.uk/reflective-paint-info.htm, 2017. data dostępu: 2017-01-01.

[6] SCHUNK GmbH & Co. KG. *OPS 080 - Collision Protection with a low Triggering Torque starting at 0.4 Nm*. SCHUNK, 2016.

[7] KUKA Roboter GmbH. *KUKA.RobotSensorInterface 3.1*. KUKA, Augsburg, 2010.

[8] Mateusz Owczarczak Krzysztof Łakomy. Projekt. *ISA Raport*, 2016.

[9] Nathan N. Liu Weike Pan, Evan W. Xiang and Qiang Yang. Transfer learning in collaborative filtering for sparsity reduction. *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence*, pages 230–235, 2010.

[10] Jacek Konopacki Jacek Izydorczyk. *Filtry analogowe i cyfrowe*. Pracownia Komputerowa, 2003.

[11] Vili Petek. Polynomial fitting in c#. http://www.vilipetek.com/2014/12/13/polynomial-fitting-in-c/, 2017. data dostępu: 2017-01-04.

[12] Gary Bishop Greg Welch. An introduction to the kalman filter. http://www.cs.unc.edu/~welch/media/pdf/kalman_intro.pdf, 2017. data dostępu: 2017-01-01.

[13] silniki krokowe.com.pl. Rysunek 4.2. http://silniki-krokowe.com.pl/wp-content/uploads/2014/10/Rampa-start-stopu-silnika-krokowego.jpg, 2017. data dostępu: 2017-01-01.

[14] John J. Craig. *Wprowadzenie do robotyki, mechanika i sterowanie*. Wydawnictwa Naukowo-Techniczne, 1995.

[15] H. H. Rapp. A ping-pong ball catching and juggling robot: A real-time framework for vision guided acting of an industrial robot arm. *5th International Conference on Automation, Robotics and Applications*, 2011.

[16] Anna Bek. Rysunek 5.13. http://kompendiumzfizyki.prv.pl/optyka.html, 2017. data dostępu: 2017-01-01.

Spis rysunków

Dodatek A

Konfiguracja programu Motive

A.1 Ustawienia kamer

Każda z kamer posiada 5 podstawowych nastaw:

- FPS - ilość klatek na sekundę, częstotliwość pracy kamery,
- EXP - ekspozycja,
- THR - wartość progowa dla rozpoznawania znaczników, odnosi się do jasności odbitych promieni podczerwonych,
- LED - moc promiennika IR zamontowanego na kamerze,
- Gain - parametr jasności obrazu kamery.

Do zadania wykrycia piłki doświadczalnie dobrano następujące nastawy dla każdej z kamer:

- FPS = 200,
- EXP = 250,
- THR = 200,
- LED = 15,
- Gain = 4:Medium (Mid Range).

A.2 Rekonstrukcja znaczników

Algorytm rekonstrukcji znaczników w przestrzeni posiada następujące nastawy:

- Residual = 2,73mm,
- Minimum Angle = 5°,
- Minimum Rays = 3,
- Minimum Ray Length = 0,20m,
- Maximum Ray Length = 4,24m,

- Pixel Gutter = 0px,
- Maximum 2D Objects = 512,
- Maximum Calculation Time = 50ms,
- Gain = 4:Medium (Mid Range),
- Filter Type - Size & Roundness,
- Min Thresholded Pixels = 20px,
- Max Thresholded Pixels = 2000px,
- Circularity = 0,7.

A.3 Ustawienia serwera NatNet

Serwer protokołu NatNet ma następujące ustawienia:

- Local Interface - Local Loopback,
- Stream Markers - True,
- Stream Rigid Bodies - False,
- Up Axis - Y Up (default),
- Remote Trigger - False,
- Type - Multicast,
- Command Port = 1510,
- Data Port = 1511,
- Multicast Interface = 239.255.42.99,
- VRPN - Broadcast Frame Data - False,
- Trackd - Broadcast Frame Data - False.

Dodatek B

Konfiguracja Robota Kuka

B.1 Katalog z ustawieniami biblioteki RSI

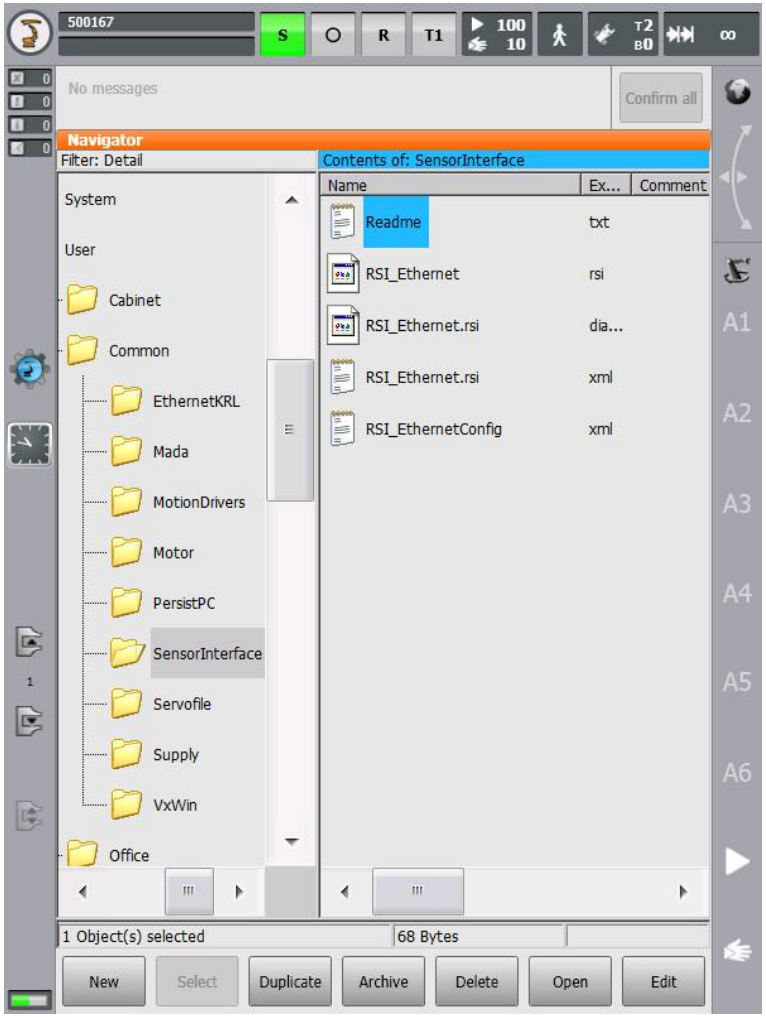
Pliki konfiguracyjne wygenerowane za pomocą programu RSIVisual powinny znaleźć się w folderze: `C:\KRC\ROBOTER\Config\User\Common\SensorInterface` przedstawionym na rysunku B.1.

B.2 Ustawienia sieciowe biblioteki RSI

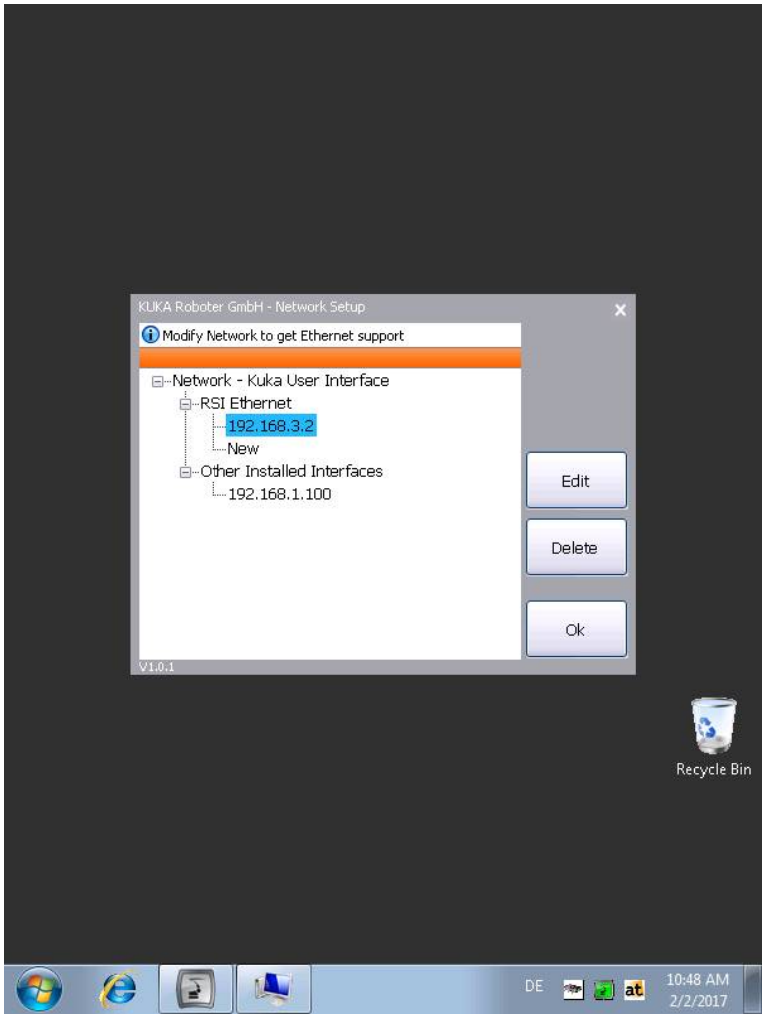
Zrzuty ekranu ustawień sieciowych przedstawiają rysunki B.2, B.3 oraz B.4.

B.3 Definicja narzędzia - Rakietka Aluminiowa

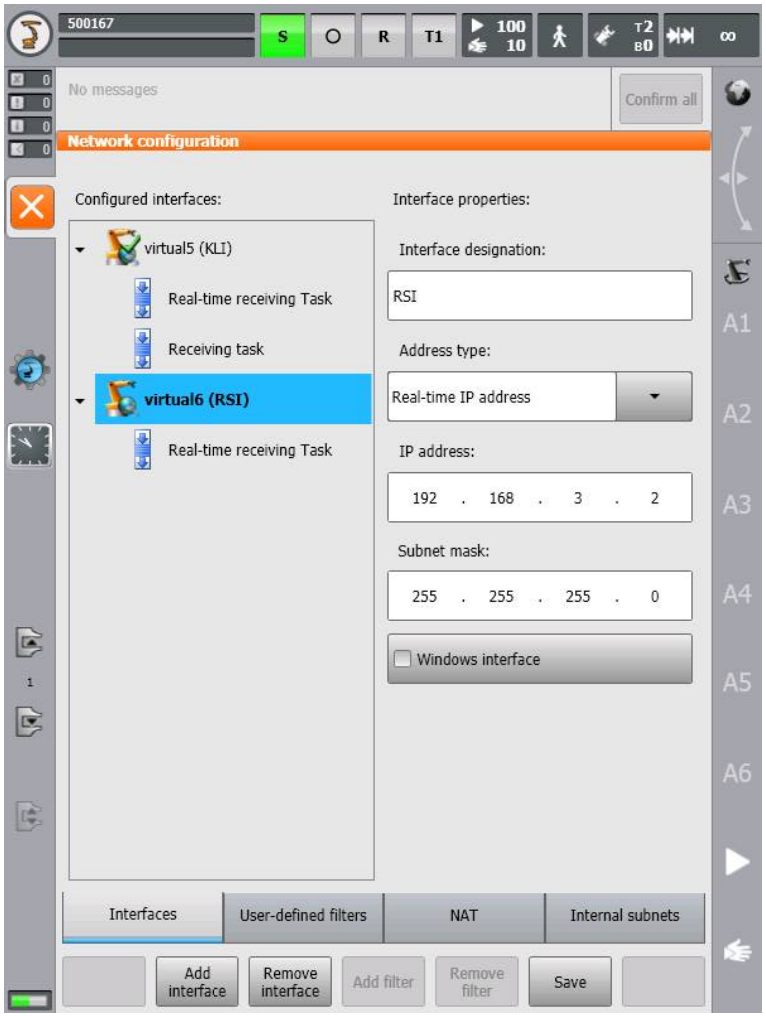
Definicja narzędzia (rakietki) została przedstawiona na rysunkach B.5 i B.6.



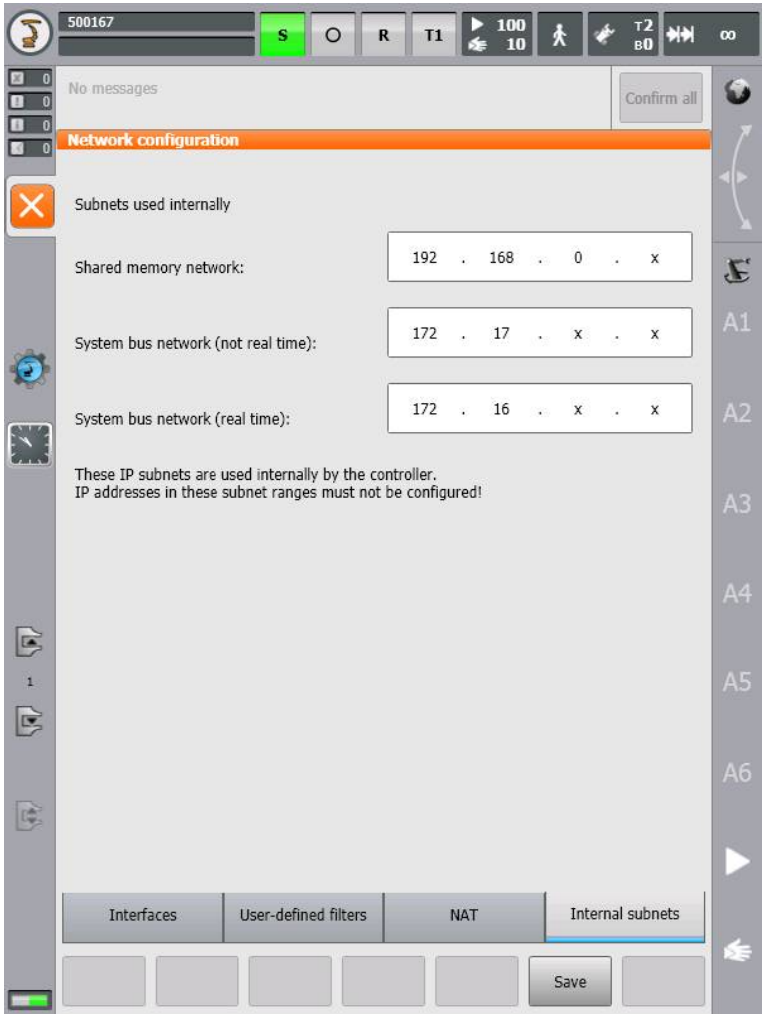
Rysunek B.1: Folder w którym znajdują się pliki konfiguracyjne pakietu RSI.



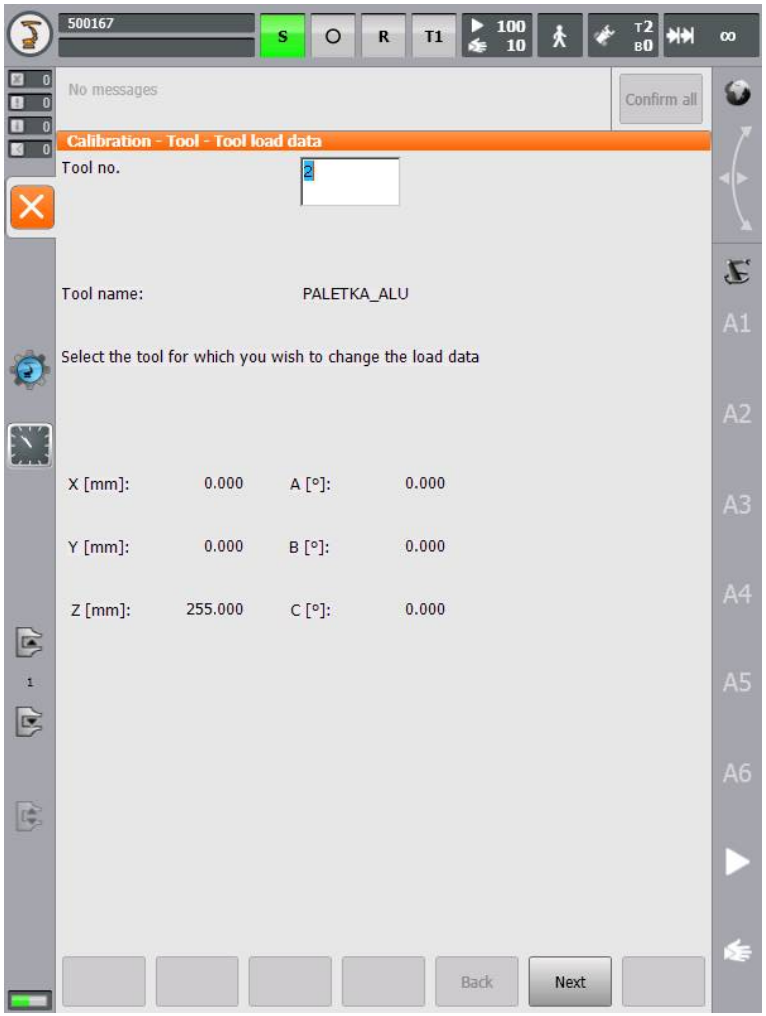
Rysunek B.2: Program do ustawiania adresu IP dla protokołu RSI.



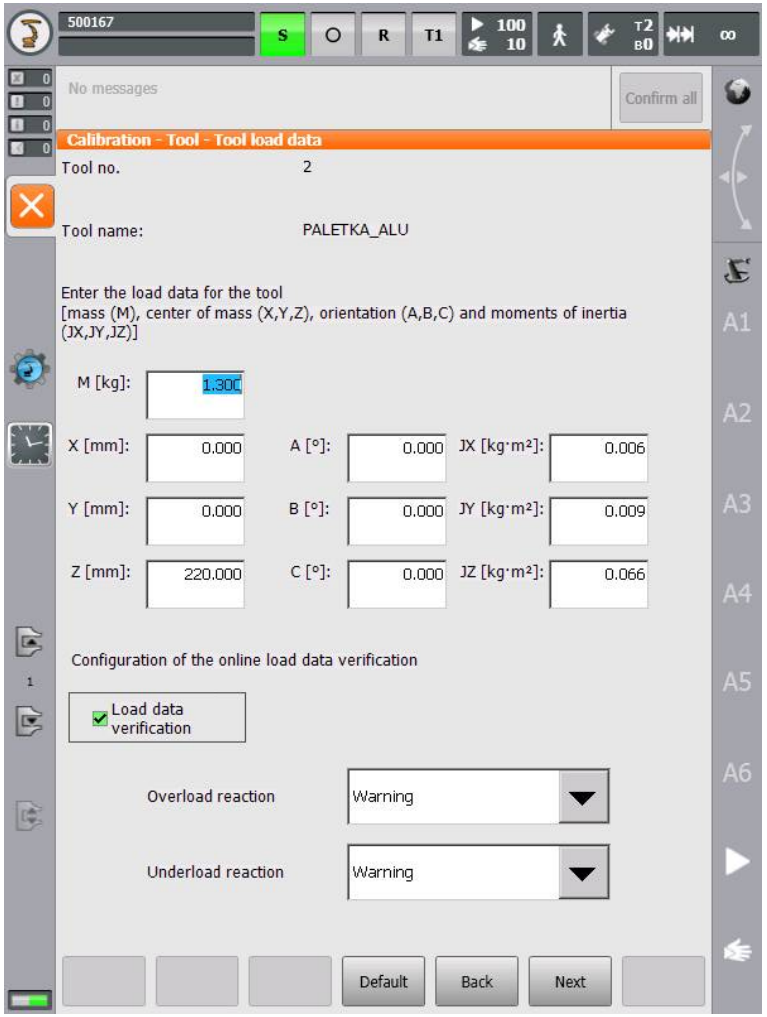
Rysunek B.3: Konfiguracja interfejsu sieciowego RSI.



Rysunek B.4: Konfiguracja wewnętrznych sieci sterownika.



Rysunek B.5: Definicja narzędzia - pozycja i orientacja.



Rysunek B.6: Definicja narzędzia - masa i środek ciężkości.