
**Politechnika Śląska**

**Wydział Automatyki, Elektroniki i Informatyki**

**Kierunek Informatyka**

# Praca dyplomowa magisterska

Projekt i realizacja stanowiska laboratoryjnego do badania zależności czasowych w sieci EtherCAT

Autor: Damian Karbowiak

Kierujący pracą: dr inż. Jacek Stój

Gliwice, czerwiec 2013

# Spis treści

# 1 Wstęp

Tematem projektu, którego dotyczy ta praca jest: „System sterowania i wizualizacji pracy robota 3D". Pomysł na projekt inżynierski pojawił się po zrealizowaniu przez autora projektu semestralnego z przedmiotu Sterowniki PLC. Zagadnienia związane z tworzeniem oprogramowania dla sterowników przemysłowych są dla autora niezwykle interesujące, a zrealizowany projekt miał na celu dalsze pogłębienie jego wiedzy z tego zakresu. Wyboru tego konkretnego tematu autor dokonał, ponieważ magazyny wysokiego składowania są dosyć powszechną problematyką branży informatyki przemysłowej, a praca nad tym tematem wydaje się być pomocna i wartościowa w przyszłej pracy zawodowej lub na dalszym etapie kształcenia.

Głównymi celami pracy było napisanie oprogramowania dla sterownika przemysłowego Siemens S7-300 oraz stworzenie graficznej prezentacji pracy modelu Robota 3D takiego jak na Rysunku **??**. Dodatkowym elementem pracy inżynierskiej zrealizowanej przez autora jest element pozainformatyczny, a mianowicie zbudowanie magazynu, z którym robot będzie współpracował. Wyżej wspomniany magazyn po zakończeniu projektu posłuży jako pomoc dydaktyczna do ćwiczeń laboratoryjnych.

## 1.1 Stanowisko laboratoryjne

Na potrzeby realizacji projektu wykorzystano istniejące stanowisko laboratoryjne, którego schemat przedstawia Rysunek **??**. Składa się ono z:

- Sterownika PLC,
- Komputera,
- Modelu Robota 3D.

Stanowisko to na potrzeby projektu rozbudowano o model magazynu wysokiego składowania. Poszczególne składowe stanowiska zostały bardziej szczegółowo opisane w kolejnych podrozdziałach.

### 1.1.1 Sterownik PLC

Sterownik PLC wykorzystywany do realizacji projektu był wyposażony w następujące moduły:

1. SIMATIC S7-300, Jednostka centralna S7-300 CPU 315F-2 PN/DP,
2. SIMATIC S7-300, Zasilacz PS 307,
3. SIMATIC S7-300, Wejścia/Wyjścia cyfrowe SM 323,
4. SIMATIC S7-300, Wejścia/Wyjścia analogowe SM 334.

Sterownik podłączony jest do sieci lokalnej Ethernet w laboratorium, więc komunikacja z nim odbywa się tak samo jak z każdym innym urządzeniem sieciowym. Podstawy programowania i korzystania ze sterowników autor poznał zapoznając się z odpowiednią literaturą [1, 2, 3, 4, 5]. Konfigurację sterownika wraz z modułami przedstawia Rysunek **??**.

### 1.1.2 Komputer

Projekt w całości był realizowany na laptopie autora, podłączanym do sieci w laboratorium. Na komputerze uruchomiane były dwie maszyny wirtualne. Na jednej zainstalowane było środowisko Step 7 do programowania sterownika, a na drugiej WinCC flexible 2008 do tworzenia i uruchamiania wizualizacji. Wizualizacje tworzone w środowisku WinCC flexible są dedykowane do paneli operatorskich, jednak ta stworzona przez autora na potrzeby projektu była uruchamiana na komputerze za pomocą runtime system.

### 1.1.3 Model Robota 3D

Wszelkie informacje dotyczące modelu robota zostały zebrane na podstawie obserwacji dokonanych przez autora oraz na podstawie udostępnionych przez uczelnię dokumentacji robota i instrukcji laboratoryjnej [**?**, **?**].

Model poprzez odpowiednie wysterowanie potrafi wykonać następujące ruchy: obracać się wokół własnej osi o 180 stopni, opuszczać i podnosić, wysuwać i wsuwać ramię oraz chwytać przedmioty. Każda z wymienionych funkcji jest realizowana przy użyciu odpowiedniego silnika. W dalszej części pracy silniki te będą nazywane odpowiednio: silnik Rotate, silnik Lift, silnik Arm oraz silnik Grab. Każdy element ruchomy robota jest wyposażony w następujące czujniki:

1. „Wyłącznik krańcowy", sygnalizujący pozycję zerową (nazywany w dalszej części pracy dla uproszczenia „krańcówką").
2. „Impulsator", czyli prosty enkoder informujący o pracy danego silnika i generujący 4 sygnały na każdy pełny obrót wału silnika.

Ponadto robot jest wyposażony w cztery pary sygnałów 'Kierunek' oraz 'Praca silnika' po jednej dla każdego silnika. Sygnały wejściowe, pochodzące z zadajnika sygnałów dyskretnych (nazywanego w dalszej części pracy dla ułatwienia „pilotem") służą do sterowania modelem robota w trybie ręcznym. Model jest sterowany poprzez skrzynkę przekaźnikową (dostępną już na stanowisku), która ma za zadanie zmianę polaryzacji napięcia na silniku każdego elementu. Każdy silnik obsługiwany jest przez dwa przekaźniki.

Jak zostało pokazane na Rysunku **??**, linia sygnałów wychodząca z modelu robota jest przekazywana do skrzynki przekaźnikowej, w której sygnały wejściowe (sygnały krańcówek) są przesyłane wprost do stacji I/O (t.j. 8 sygnałów + 1 przewód wspólny, czyli masa dla sygnałów wejściowych). Sygnały wyjściowe (załączające silnik) w skrzynce przekaźnikowej dołączone są do przekaźników, które mają za zadanie zmianę polaryzacji napięcia sterującego silnikami. Ze skrzynki do wyjść modułu I/O dochodzą sygnały, które sterują zmianą polaryzacji na wyjściach przekaźników oraz podaniem zasilania załączającego silnik (t.j. 8 sygnałów + 1 masa). Szczegóły dotyczące tych sygnałów zawierają Tablice 1 oraz 2.

### 1.1.4 Magazyn

Jak już zostało wspomniane we wstępie, projekt miał pozostawić po sobie coś więcej niż tylko oprogramowanie i wizualizację, więc powstał model magazynu, który posłuży

| Adres zmiennej w sterowniku | Typ zmiennej | Opis |
|---|---|---|
| I 4.0 | Bool | Sterowanie ręczne Rotate - start |
| I 4.1 | Bool | Sterowanie ręczne Arm - start |
| I 4.2 | Bool | Sterowanie ręczne Lift - start |
| I 4.3 | Bool | Sterowanie ręczne Grab - start |
| I 4.4 | Bool | Sterowanie ręczne Rotate - kierunek |
| I 4.5 | Bool | Sterowanie ręczne Arm - kierunek |
| I 4.6 | Bool | Sterowanie ręczne Lift - kierunek |
| I 4.7 | Bool | Sterowanie ręczne Grab - kierunek |
| I 5.0 | Bool | Krańcówka wyłączająca dany silnik |
| I 5.1 | Bool | Krańcówka licznika impulsów dla obracania |
| I 5.2 | Bool | Krańcówka wyłączająca dany silnik |
| I 5.3 | Bool | Krańcówka licznika impulsów dla ramienia |
| I 5.4 | Bool | Krańcówka wyłączająca dany silnik |
| I 5.5 | Bool | Krańcówka licznika impulsów dla podnośnika |
| I 5.6 | Bool | Krańcówka wyłączająca dany silnik |
| I 5.7 | Bool | Krańcówka licznika impulsów dla chwytaka |

Tablica 1: Zmienne wejściowe do modułów I/O

| Adres zmiennej w sterowniku | Typ zmiennej | Opis |
|---|---|---|
| Q 4.0 | Bool | Sygnalizacja pozycji minimalnej dla silnika Rotate |
| Q 4.1 | Bool | Sygnalizacja pozycji maksymalnej dla silnika Rotate |
| Q 4.2 | Bool | Sygnalizacja pozycji minimalnej dla silnika Arm |
| Q 4.3 | Bool | Sygnalizacja pozycji maksymalnej dla silnika Arm |
| Q 4.4 | Bool | Sygnalizacja pozycji minimalnej dla silnika Lift |
| Q 4.5 | Bool | Sygnalizacja pozycji maksymalnej dla silnika Lift |
| Q 4.6 | Bool | Sygnalizacja pozycji minimalnej dla silnika Grab |
| Q 4.7 | Bool | Sygnalizacja pozycji maksymalnej dla silnika Grab |
| Q 5.0 | Bool | Włączenie/wyłączenie silnika obrotowego |
| Q 5.1 | Bool | Kierunek obrotu (0-counterclock 1-clockwise) |
| Q 5.2 | Bool | Włączenie/wyłączenie silnika wysuwu |
| Q 5.3 | Bool | Kierunek ramienia (0-In 1-Out) |
| Q 5.4 | Bool | Włączenie/wyłączenie silnika podnośnika (0-up 1-down) |
| Q 5.5 | Bool | Kierunek podnoszenia |
| Q 5.6 | Bool | Włączenie/wyłączenie silnika uchwytu |
| Q 5.7 | Bool | Kierunek chwytania (0-open 1-close) |

Tablica 2: Zmienne wyjściowe z modułów I/O

studentom jako pomoc dydaktyczna na laboratoriach z programowania sterownika Siemens do sterowania robotem Fischertechnik.

Magazyn zaprojektowano tak, aby optymalnie wykorzystać konstrukcję robota. Przegrody w modelu magazynu zostały rozmieszczone na półokręgu ze względu na półkolisty

tor poruszania się ramienia w poziomie.

Do budowy magazynu zastosowano płytę MDF o grubości 18 mm oraz szufladki warsztatowe. Na podstawie pomiarów wykonanych na modelu oraz na podstawie wcześniejszego projektu powstał model, który zobaczyć można na Rysunkach **??** oraz **??** lub w sali 544 wydziału AEI na Politechnice Śląskiej.

## 1.2 Analiza tematu

Analiza tematu polegała przede wszystkim na zapoznaniu się z narzędziami programistycznymi do tworzenia oprogramowania sterownika oraz wizualizacji. W wyniku analizy autor poznał podstawy języków: LAD [**?**, 6, **?**], STL [**?**, 6, **?**], FBD [**?**, 6, **?**], GRAPH [**?**], SCL [**?**, **?**, **?**] i AWL do tworzenia programu sterownika oraz VBScript do tworzenia skryptów w wizualizacji. Poznanie tych podstaw pozwoliło dobrać język odpowiedni do realizacji poszczególnych zadań.

## 1.3 Założenia

Oprogramowanie dla Robota Fishertechnik powinno zostać stworzone przy użyciu środowiska Step 7 oraz działać na sterownikach firmy Siemens. Funkcjonalności robota wchodzące w skład projektu, to:

- sterowanie ręczne z pilota podłączonego bezpośrednio do sterownika,
- sterowanie ręczne z wizualizacji,
- sterowanie automatyczne,
- wizualizacja stanu magazynu,
- umożliwienie korzystania z magazynu zarówno poprzez sterowanie ręczne, jak i przy użyciu zautomatyzowanych poleceń dostępnych z poziomu wizualizacji.

Powyżej zostały wymienione założenia podstawowe, jednak autor nie wyklucza zrealizowania dodatkowych zadań, które nie zostały zamieszczone w pierwotnej koncepcji realizacji projektu.

## 1.4 Plan pracy

Realizacja projektu została podzielona na następujące etapy:

- Przygotowanie stanowiska, zebranie odpowiednich materiałów i literatury,
- Analiza wymagań funkcjonalnych aplikacji,
- Projektowanie struktury oprogramowania i interfejsów wymiany danych,
- Implementacja,
- Testowanie i uruchamianie,

- Przedstawienie projektu i ewentualne korekty.

Powyższy plan pracy stanowił dla autora wyznacznik kolejnych działań. Jednak powszechnie wiadomo, że w praktyce poszczególne punkty są wymienne i wpływają na siebie wzajemnie.

# 2 Oprogramowanie sterownika

W niniejszym rozdziale opisane zostało oprogramowanie sterujące modelem. W kolejnych podrozdziałach zostanie przedstawiona specyfikacja zewnętrzna oraz wewnętrzna.

Stworzone przez autora oprogramowanie wraz ze wszystkimi funkcjami systemowymi zajmują w sterowniku 47428 kB z dostępnych 524288 kB. Zajętość pamięci dostępnej w sterowniku obrazuje zrzut ekranu wykonany w środowisku Step 7, widoczny na Rysunku 1.

## 2.1 Specyfikacja zewnętrzna

Specyfikacja zewnętrzna przedstawiona w dalszej części podrozdziału zawiera opis, jak korzystać z oprogramowania wgranego do sterownika przez jego autora. Opisane zostało, jak ustawiać odpowiednie zmienne, aby uzyskać żądany efekt.

Lista zmiennych wejściowych i wyjściowych wymieniana między sterownikiem a modelem została już opisana w pierwszym rozdziale, w Tablicach 1 oraz 2. Pozostałe zmienne znajdują się w wewnętrznej pamięci sterownika.

Oprogramowanie może sterować modelem w sposób automatyczny lub ręczny. Tryb automatyczny w trybie obsługi magazynu zostanie opisany w podrozdziale 2.1.2. Tryb ręczny może być realizowany przy pomocy zadajnika podpiętego do sterownika lub przy pomocy przycisków umieszczonych na odpowiednim ekranie wizualizacji. W trybie tym o pracy robota decydujący jest stan przycisków. Dopuszczalne są wszystkie możliwe ruchy w przedziale od wyłącznika krańcowego do wartości maksymalnej.

Sterowanie przy pomocy pilota podłączonego do sterownika odbywa się za pomocą 4 przycisków monostabilnych do załączania silników oraz 4 przełączników bistabilnych do wybierania kierunku. Podobne do sterowania pilotem jest sterowanie ręczne z poziomu wizualizacji, polegające na odpowiednim modyfikowaniu bitów: M11.0 - M11.7.

W automatycznym trybie pracy kluczową rolę odgrywają 4 zmienne: *LiftEndPos*, *GrabEndPos*, *ArmEndPos* oraz *RotateEndPos* typu INT, które wskazują na pozycje docelowe silników. Są one podawane do bloków FB1 odpowiadających kolejnym silnikom. O tym, czy wybrany silnik może się w danym momencie poruszać czy nie decydują flagi odpowiednio ustawione przez blok FB9.

Ważnym elementem jest kolejka obsługiwana w trybie automatycznym. Indeksy przechowywane są w przestrzeni od DB6.DBW0 do DB6.DBW202, a związane z nimi bezpośrednio zmienne typu bool - w przestrzeni od DB6.DBX202.0 do DB6.DBX216.0. Z kolejką tą związana jest dodatkowo zmienna *CurrentIndex* określająca jej długość.

Kolejną przestrzenią adresową jest blok DB5, który jest reprezentacją w pamięci sterownika modelu magazynu i informacji o nim. Od adresu DB5.DBX0.0 znajduje się 26-elementowa tablica określająca zajętość poszczególnych komórek magazynu. Następnie od adresu DB5.DBW4 dostępna jest dwuwymiarowa tablica 26x3 przechowująca pozycję poszczególnych komórek magazynu. W przestrzeni zaczynającej się pod adresem DB5.DBB160 znajduje się 26-elementowa tablica zmiennych typu DATE_AND_TIME przechowująca datę ostatniego dostępu do komórki. Dodatkowo w przestrzeni tej mamy dostępną pod adresem DB5.DBB368 zmienną przechowującą aktualną datę oraz godzinę w sterowniku, odświeżaną przy każdym przebiegu bloku OB1.

## 2.2 Specyfikacja wewnętrzna

Podrozdział specyfikacja wewnętrzna opisuje sposób rozwiązania przez autora kwestii sterowania modelem przy użyciu dostępnego na stanowisku sterownika oraz poszczególnych trybów sterowania. W tworzeniu oprogramowania zostały wykorzystane następujące języki programowania:

- język drabinkowy (Ladder), wykorzystany do stworzenia głównych elementów programu,
- S7-SCL, który został zastosowany do korzystania z tablic. Niestety do korzystania z nich nie można zastosować języka LAD, ponieważ nie da się w nim odwoływać do elementów tablicy przez indeksy będące zmiennymi, a jedynie przez stałe. Po zapoznaniu się z dokumentacją okazało się, że taka możliwość istnieje w języku STL, ale jest to metoda skomplikowana w implementacji. Właśnie dlatego najlepszym i najprostszym rozwiązaniem okazuję się S7-SCL, który jest kompilowany do kodu w języku STL.

Istotne fragmenty programu aplikacyjnego:

- blok funkcyjny FB1 - blok sterujący pracą poszczególnych silników,
- blok funkcyjny FB8 - blok zawierający implementację kolejki FIFO,
- blok funkcyjny FB9 - blok analizujący stan modelu i ustawiający zmienne zezwalające na ruch silników,
- blok funkcyjny FB10 - blok przetwarzający zadania z kolejki na odpowiednie dane (operuje na odpowiednich tablicach),
- blok funkcyjny FB11 - blok zwracający indeks i zmienną bool najbliższego lub najdalszego elementu,
- blok funkcyjny FB12 - blok zwracający indeks i zmienną bool najmłodszego lub najstarszego elementu.

Wszystkie wymienione bloki zostaną szczegółowo opisane w kolejnych podrozdziałach.

### 2.2.1 Blok FB1

Najważniejszą częścią oprogramowania jest blok funkcyjny FB1. Wszystkie zmienne wejściowe oraz wyjściowe dla tego bloku zostały zebrane w Tablicah 3 oraz 4. Do działania wykorzystuje on zestaw danych wewnętrznych. Jako parametry wejściowe bloku, oprócz zmiennych z modułu I/O sterownika, podajemy typ aktualnego trybu, maksymalną dopuszczalną wartość wewnętrznego licznika, pozycję docelową w trybie automatycznym oraz zmienne *Enable* i *ResetSequence*. Ostatnie dwie zmienne odpowiadają za to, aby ruchy w trybie automatycznym i resetowanie wykonywane były w odpowiedniej kolejności. Na wyjściu mamy tylko połączenia dla zmiennych z tablicy Symbols oraz flagę osiągnięcia pozycji docelowej przez dany silnik.

Blok FB1 jest wywoływany w bloku OB1 z odpowiednimi parametrami, niezależnie dla poszczególnych 4 silników. Przykładowe wywołanie znajduje się na Rysunku 2. Na

| Nazwa zmiennej | Typ zmiennej | Opis |
|---|---|---|
| StopSensor | Bool | Sygnał z odpowiedniej krańcówki |
| EngineCounter | Bool | Sygnał z impulsatora obrotów |
| CounterMax_I | Int | Maksymalna wartość licznika typu Int |
| EndPosition | Int | Pozycja docelowa |
| Automat | Bool | Tryb automatyczny |
| ManualWorkDirection | Bool | Start w trybie ręcznym z pilota |
| ManualWorkStart | Bool | Start w trybie ręcznym z wizualizacji |
| VisualWorkDirection | Bool | Kierunek w trybie ręcznym z pilota |
| VisualWorkStart | Bool | Kierunek w trybie ręcznym z wizualizacji |
| CounterForState | Counter | Licznik wewnętrzny z aktualną pozycją |
| Enable | Bool | Sygnał zezwalający na inkrementację |
| ResetSequence | Bool | Zmienna pozwalająca na sekwencyjny reset modelu |
| VisualPilot | Bool | Zmienna określająca tryb pracy manualnej z pilota lub wizualizacji |

Tablica 3: Zmienne wejściowe do bloku FB1

| Nazwa zmiennej | Typ zmiennej | Opis |
|---|---|---|
| EngineOnOff | Bool | Włączenie/wyłączenie silnika |
| EngineDirection | Bool | Kierunek pracy silnika |
| MinValue | Bool | Sygnał osiągnięcia minimum |
| MaxValue | Bool | Sygnał osiągnięcia maksimum |
| ReachPosition | Bool | Sygnał osiągnięcia pozycji docelowej |
| ResetFinish | Bool | Sygnał zakończenia resetu danego silnika |

Tablica 4: Zmienne wyjściowe z bloku FB1

rysunku łatwo można zaobserwować, że do bloku podane są wszystkie niezbędne informacje dla danego silnika oraz 2 wspólne dla wszystkich silników sygnały informujące o aktualnym trybie pracy modelu.

Istotnymi fragmentami bloku FB1 są 3 gałęzie programu: licznik wewnętrzny, decyzja o kierunku oraz decyzja o załączeniu. Bardzo istotnym warunkiem wpływającym na pracę silnika jest jego położenie. Jeśli silnik osiąga wartość maksymalną lub minimalną to przerywa swoją pracę. Warunkiem koniecznym do określenia tej pozycji jest wykorzystanie licznika do zliczania impulsów z modelu i jego odpowiednia inkrementacja lub dekrementacja. Tak działający licznik przedstawia Rysunek 3.

Decyzja dotycząca kierunku jest podejmowana zależnie od trybu pracy, co zaobserwować można na Rysunku 4. W przypadku trybu automatycznego decydująca jest bieżąca pozycja oraz pozycja docelowa. Kierunek jest ustawiany tak, aby załączony silnik zmierzał do pozycji docelowej. W przypadku trybów manualnych decydujące są stany przycisków przy sterowaniu z pilota lub zmienne przy sterowaniu z wizualizacji. Jeżeli została podjęta decyzja o załączeniu silnika to stan przycisku (zmiennej) jest wpisywany do zmiennej

statycznej *EngineDirectionManual*, która jest następnie zsumowana logicznie ze zmienną *EngineDirectionAuto*, dając w wyniku decyzję o kierunku.

Część decydująca o załączeniu silnika jest bardzo rozbudowana. Jest to przykład załączenia z podtrzymaniem. Zależnie od trybu pracy sprawdzane są inne warunki. W przypadku trybu automatycznego ważna jest zmienna *Enable*, która decyduje o załączeniu wybranego silnika, co pozwala na częściową pracę sekwencyjną. Silnik pracuje w kierunku krańcówki aż do jej osiągnięcia, a w kierunku od krańcówki aż do osiągnięcia swojego maksimum, chyba że wcześniej zostanie osiągnięta pozycja docelowa. W ręcznych trybach pracy głównym warunkiem decydującym o załączeniu zależnie od kierunku są wartość maksymalna licznika lub krańcówka. Wspólnym dla obu trybów warunkiem pozwalającym zatrzymać pracę silnika jest ustawienie zmiennej *EmergencyStop* na wartość *true*. Tą rozbudowaną i skomplikowaną gałąź prezentuje Rysunek 5.

### 2.2.2 Blok FB8

W trybie automatycznym podczas obsługi magazynu praca odbywa się na zasadzie zadań do wykonania. Na zadanie takie składa się indeks komórki w magazynie oraz zmienna typu bool, która decyduje o ruchu chwytaka (zaciskanie lub zwalnianie). Indeks jest następnie przetwarzany w odpowiednim bloku na położenie komórki w modelu magazynu.

Bardzo ważnym blokiem funkcyjnym jest blok zarządzający kolejką zadań do wykonania w trybie automatycznym. Na wejściu znajdują się: dodawany do kolejki indeks oraz zmienna decydująca o zamknięciu lub otwarciu chwytaka. Na wyjściu mamy zmienne takie same jak na wejściu, ale przepuszczone już przez kolejkę. Zmienne wejściowo-wyjściowe to 2 flagi żądania, odpowiednio: dodania do kolejki lub pobrania z niej elementu oraz 2 flagi potwierdzające wykonanie żądania. Dodatkowo w bloku tym występuje zmienna wewnętrzna stanowiąca indeks w tablicy.

```
FUNCTION_BLOCK FB8

VAR_INPUT
    AddedBool: BOOL;
    AddedIndex: INT;
END_VAR

VAR_OUTPUT
    WarehouseBool: BOOL;
    WarehouseIndex: INT;
END_VAR

VAR_IN_OUT
    Request: BOOL;
    Add: BOOL;
    RequestAck: BOOL;
    AddAck: BOOL;
END_VAR

VAR
    Index: INT ;
END_VAR

BEGIN
IF Add = true & Queue.CurentIndex <= 100 THEN
    IF AddedIndex >= 0 AND AddedIndex < 26 THEN
        Queue.QueueIndex[Queue.CurentIndex] := AddedIndex;
        Queue.QueueBool[Queue.CurentIndex] := AddedBool;
        Queue.CurentIndex := Queue.CurentIndex + 1;
        Add := false;
        AddAck := true;
    ELSE
        Error.IndexError := 'Indeks spoza zakresu';
        Error.IndexErrorPres := true;
    END_IF;
ELSIF Queue.CurentIndex >= 100 THEN
    Error.QueueError := 'Przepelniona Kolejka';
    Error.QueueErrorPres := true;
END_IF;

IF Request = true & Queue.CurentIndex > 0 THEN
    IF Queue.QueueIndex[0] <> -1 THEN
        WarehouseIndex := Queue.QueueIndex[0];
        WarehouseBool := Queue.QueueBool[0];
            FOR Index := 0 TO Queue.CurentIndex DO
                Queue.QueueIndex[Index] := Queue.QueueIndex[Index+1];
                Queue.QueueBool[Index] := Queue.QueueBool[Index+1];
            END_FOR;
        Queue.QueueIndex[Queue.CurentIndex] := 0;
        Queue.QueueBool[Queue.CurentIndex] := false;
        Queue.CurentIndex := Queue.CurentIndex - 1;
        Request := false;
        RequestAck := true;
    ELSE
        Error.IndexError := 'Bledny indeks';
```

```
56          Error.IndexErrorPres := true;
57      END_IF;
58  ELSIF Queue.CurentIndex = 0 THEN
59      Request := false;
60      RequestAck := false;
61      Error.QueueError := 'Kolejka jest pusta';
62      Error.QueueErrorPres := true;
63  END_IF;
64  END_FUNCTION_BLOCK
```

Kod źródłowy 1: FB8 - Zarządzanie kolejką

### 2.2.3 Blok FB9

Kolejnym istotnym blokiem jest blok decydujący o zezwoleniu na pracę poszczególnych silników, pozwalając przez to wykonywać ruchy ramienia w odpowiedniej kolejności. Blok na wyjściu posiada 4 zmienne stanowiące właściwe zezwolenie na ruch. Decyzja jest podejmowana na podstawie położenia silnika Arm oraz osiągnięcia przez poszczególne silniki swoich pozycji docelowych.

```
1   FUNCTION_BLOCK FB9
2
3   VAR_INPUT
4       ArmCounterl: COUNTER;
5   END_VAR
6
7   VAR_OUTPUT
8       ArmEn: BOOL;
9       LiftEn: BOOL;
10      RotateEn: BOOL;
11      GrabEn: BOOL;
12  END_VAR
13
14  BEGIN
15      IF AutoTest THEN
16          ArmEn := true;
17          LiftEn := true;
18          RotateEn := true;
19          GrabEn := true;
20      ELSE
21          IF ArmData.CounterValue_I <= 25 THEN
22              ArmEn := true;
23              LiftEn := true;
24              RotateEn := true;
25              GrabEn := false;
26          ELSE
27              IF LiftPositionReach  AND RotatePositionReach THEN
28                  ArmEn := true;
29                  LiftEn := false;
30                  RotateEn := false;
31                  GrabEn := false;
32              ELSE
33                  IF ArmData.CounterValue_I > 25 THEN
34                      ArmEn := true;
```

```
35                      LiftEn := false;
36                      RotateEn := false;
37                      GrabEn := false;
38                  else
39                      ArmEn := false;
40                      LiftEn := true;
41                      RotateEn := true;
42                      GrabEn := false;
43                  END_IF;
44              END_IF;
45          END_IF;
46          IF LiftPositionReach & RotatePositionReach & ArmPositionReach
               THEN
47              ArmEn := false;
48              LiftEn := false;
49              RotateEn := false;
50              GrabEn := true;
51          END_IF;
52      END_IF;
53  END_FUNCTION_BLOCK
```

Kod źródłowy 2: FB9 - Zezwolenie na ruch

### 2.2.4 Blok FB10

Blok funkcyjny FB10 zarządza operacjami wykonywanymi na tablicach związanych z obsługiwanym magazynem. Na wejściu mamy zmienne stanowiące wyjście z kolejki oraz zmienną określającą, że wykonane zostało ostatnie podzadanie uruchomione przez ten blok. Na wyjściu mamy zmienne określające, do jakich pozycji mają dojechać silniki.

Zmienne wejściowo-wyjściowe to żądanie następnego zadania z kolejki oraz potwierdzenie obsłużenia żądania przez kolejkę. Zmienna tymczasowa jest to wartość zwrócona przez funkcję systemową odczytu bieżącej daty oraz godziny. Zmienna wewnętrzna *DoneCount* przechowuje informację o tym, ile podzadań zostało wykonanych.

```
1  FUNCTION_BLOCK FB10
2
3  VAR_INPUT
4      Done: BOOL;
5      WarehouseIndex: INT;
6      WarehouseBool: BOOL;
7  END_VAR
8
9  VAR_OUTPUT
10     ArmTo: INT;
11     LiftTo: INT;
12     RotateTo: INT;
13     GrabTo: INT;
14 END_VAR
15
16 VAR_IN_OUT
17     NextRequest: BOOL;
18     NextRequestACK: BOOL;
19 END_VAR
```

```
20
21 VAR_TEMP
22     SFC1_Ret_val: INT;
23 END_VAR
24
25 VAR
26     DoneCount: INT;
27     LoadNext: BOOL;
28 END_VAR
29
30 BEGIN
31
32 IF NextRequestACK = true THEN
33     NextRequest := false;
34     NextRequestACK := false;
35     DoneCount := 0;
36     ArmTo := 22;
37     IF WarehouseBool THEN
38         LiftTo := Warehouse.WarehousePosition[WarehouseIndex, 2];
39     ELSE
40         LiftTo := Warehouse.WarehousePosition[WarehouseIndex, 2]-3;
41     END_IF;
42     RotateTo := Warehouse.WarehousePosition[WarehouseIndex, 3];
43     LoadNext := true;
44 END_IF;
45
46 IF DoneCount = 1 THEN
47     ArmTo := Warehouse.WarehousePosition[WarehouseIndex, 1];
48     IF WarehouseBool THEN
49         LiftTo := Warehouse.WarehousePosition[WarehouseIndex, 2];
50     ELSE
51         LiftTo := Warehouse.WarehousePosition[WarehouseIndex, 2]-3;
52     END_IF;
53     RotateTo := Warehouse.WarehousePosition[WarehouseIndex, 3];
54     GrabTo := BOOL_TO_INT(WarehouseBool) * 19;
55 END_IF;
56
57 IF DoneCount = 2 THEN
58     Warehouse.WarehouseBool[WarehouseIndex] := NOT WarehouseBool;
59     SFC1_Ret_val := SFC1(CDT := Warehouse.WarehouseDateTime[
           WarehouseIndex]);
60     DoneCount := 0;
61     IF Queue.CurentIndex > 0 THEN
62         NextRequest := true;
63         LoadNext := false;
64     ELSE
65         Error.QueueError := 'Kolejka jest pusta';
66     END_IF;
67 END_IF;
68
69 IF Done = true & LoadNext THEN
70     DoneCount := DoneCount + 1;
71 END_IF;
72 END_FUNCTION_BLOCK
```

Kod źródłowy 3: FB10 - Operacje na tablicah

### 2.2.5 Blok FB11

Blok FB11 w wyniku działania zwraca indeks komórki w magazynie oraz powiązaną z nim zmienną typu bool, które pozwalają wybrać najdalej lub najbliżej położony element. Wskazuje on pustą lub zajętą komórkę magazynu, zależnie od zmiennych wejściowych. Zależnie od wartości zmiennej *FarNear* wybierana jest najdalej lub najbliżej położona komórka, natomiast zmienna *BoolGrab* decyduje o tym, czy szukamy wolnej czy zajętej komórki magazynu.

W wyniku działania blok ustawia na wyjściu zmienne związane z wybraną komórką magazynu oraz flagę żądania dodania uzyskanego wyniku do kolejki zadań. Zmienna *MyEn* decyduje o tym, czy ten blok zostaje aktywowany czy nie. Zmienna index jest zmienną wewnętrzną wykorzystywaną w pętlach FOR.

```
FUNCTION_BLOCK FB11

VAR_INPUT
    FarNear: BOOL;
END_VAR

VAR_OUTPUT
    IndexSel: INT;
    AddQueue: BOOL;
END_VAR

VAR_IN_OUT
    MyEn: BOOL;
    BoolGrab: BOOL;
END_VAR

VAR
    Index: INT;
END_VAR

BEGIN
IF MyEn = true THEN
    IF FarNear = true & BoolGrab = true THEN
        FOR Index := 1 TO 24 DO
            IF Warehouse.WarehouseBool[Index] = true THEN
                IndexSel := Index;
                EXIT;
            ELSE
                IndexSel := -1;
            END_IF;
        END_FOR;
    END_IF;

    IF FarNear = false & BoolGrab = true THEN
        FOR Index := 24 TO 1 BY -1 DO
            IF Warehouse.WarehouseBool[Index] = true THEN
                IndexSel := Index;
                EXIT;
            ELSE
                IndexSel := -1;
            END_IF;
```

```
42          END_FOR;
43      END_IF;
44
45      IF FarNear = true & BoolGrab = false THEN
46          FOR Index := 1 TO 24 DO
47              IF Warehouse.WarehouseBool[Index] = false THEN
48                  IndexSel := Index;
49                  EXIT;
50              ELSE
51                  IndexSel := -1;
52              END_IF;
53          END_FOR;
54      END_IF;
55
56      IF FarNear = false & BoolGrab = false THEN
57          FOR Index := 24 TO 1  BY -1 DO
58              IF Warehouse.WarehouseBool[Index] = false THEN
59                  IndexSel := Index;
60                  EXIT;
61              ELSE
62                  IndexSel := -1;
63              END_IF;
64          END_FOR;
65      END_IF;
66      MyEn := false;
67      AddQueue := true;
68  END_IF;
69  END_FUNCTION_BLOCK
```

Kod źródłowy 4: FB11 - Funkcja wybiera najdalszą lub najbliższą komórkę

### 2.2.6 Blok FB12

Blok FB12 w wyniku działania zwraca indeks komórki w magazynie oraz powiązaną z nim zmienną typu bool, które pozwalają wybrać najstarszy lub najmłodszy element tablicy. Wskazuje on zajętą komórkę magazynu. Zależnie od wartości zmiennej *Smallest-Greatest* wybierana jest najmłodsza lub najstarsza zajęta komórka.

W wyniku działania blok ustawia na wyjściu zmienne związane z wybraną komórką magazynu oraz flagę żądania dodania uzyskanego wyniku do kolejki zadań. Zmienna *MyEn* decyduje o tym, czy ten blok zostaje aktywowany czy nie. Zmienna index jest zmienną wewnętrzną wykorzystywaną w pętlach FOR. *IndexTemp* jest zmienną pomocniczą stanowiącą tymczasowy indeks przy wybieraniu elementu końcowego.

```
1   FUNCTION_BLOCK FB12
2
3   VAR_INPUT
4       SmallestGreatest: BOOL;
5   END_VAR
6
7   VAR_OUTPUT
8       IndexSelected: INT;
9       BoolSelected: BOOL;
10      AddQueue: BOOL;
```

```
END_VAR

VAR_IN_OUT
    MyEn: BOOL;
END_VAR

VAR
    Index: INT;
END_VAR

VAR_TEMP
    IndexTemp: INT;
END_VAR

BEGIN
IF MyEn = true THEN

    IF SmallestGreatest = false THEN
        FOR Index := 1 TO 24 DO
            IF Warehouse.WarehouseBool[Index] = true AND FC28(DT1 :=
               Warehouse.WarehouseDateTime[Index], DT2 := DT
               #1990-01-01-12:00:00.00) THEN
                IndexTemp := Index;
                EXIT;
            ELSE
                IndexTemp := -1;
            END_IF;
        END_FOR;
        IF IndexTemp <> -1 THEN
            FOR Index := 1 TO 24 DO
                IF Warehouse.WarehouseBool[Index] = true then
                    IF FC14(DT1 := Warehouse.WarehouseDateTime[Index],
                       DT2 := Warehouse.WarehouseDateTime[IndexTemp])
                       THEN
                        IndexTemp := Index;
                    END_IF;
                END_IF;
            END_FOR;
        END_IF;
    END_IF;

    IF SmallestGreatest = true THEN
        FOR Index := 1 TO 24 DO
            IF Warehouse.WarehouseBool[Index] = true AND FC28(DT1 :=
               Warehouse.WarehouseDateTime[Index], DT2 := DT
               #1990-01-01-12:00:00.00) THEN
                IndexTemp := Index;
                EXIT;
            ELSE
                IndexTemp := -1;
            END_IF;
        END_FOR;
        IF IndexTemp <> -1 THEN
            FOR Index := 1 TO 24 DO
                IF Warehouse.WarehouseBool[index] = true then
```

```
60                      IF FC23(DT1 := Warehouse.WarehouseDateTime[Index],
                            DT2 := Warehouse.WarehouseDateTime[IndexTemp])
                            THEN
61                           IndexTemp := Index;
62                      END_IF;
63                  END_IF;
64              END_FOR;
65          END_IF;
66      END_IF;
67      MyEn := false;
68      IndexSelected := IndexTemp;
69      BoolSelected := true;
70      AddQueue := true;
71  END_IF;
72  END_FUNCTION_BLOCK
```

Kod źródłowy 5: FB12 - Funkcja wybiera najmłodszą lub najstarszą zajętą komórkę

# 3 Uruchamianie i testowanie

Rozdział ten zawiera generalne podsumowanie przebiegu prac nad projektem. Opisane zostaną tu wszelkie poważne problemy, które wystąpiły w czasie realizacji projektu. Ponadto zawarto tu opis przebiegu procesu testowania.

## 3.1 Przebieg testowania

W procesie weryfikacji poprawności działania projektu zastosowano testowanie wstępujące. Na początku powstał blok FB1 sterujący pojedynczym silnikiem, który następnie został wywołany dla wszystkich 4 silników z odpowiednimi parametrami. W kolejnym etapie działające poprawnie sterowanie silnikami zostało wykorzystane do bardziej zaawansowanych funkcji i tak aż do osiągnięcia prawidłowych wyników testów wszystkich funkcji.

Głównym testerem był autor projektu więc większość testów przebiegała na zasadzie białej skrzynki (ang. *white box*), bardzo często z użyciem podglądu stanu w środowisku Step 7. Takie testowanie pozwala stosunkowo łatwo wyszukać źródło błędu i je wyeliminować.

Autor kilka razy przeprowadzał testy stosując metodę czarnej skrzynki (ang. *black box*), nie biorąc pod uwagę zależności wykonywanych czynności, od realizowanego przez sterownik kodu. Kilkukrotnie w czasie realizacji projektu do testów zgłaszały się osoby trzecie, które były nim zaciekawione. Testy wykonane przez takie osoby są niezwykle cenne ze względu na dużą nieprzewidywalność oraz całkowitą niezależność działań od rozwiązań ze względu na brak ich znajomości.

W kilku testach ze względu na destrukcyjny wpływ błędów na model robota, testy odbywały się bez podłączonych silników z ręcznym wymuszaniem zmiany stanów krańcówek oraz impulsatorów. Takie działanie pozwoliło wykryć ewentualne błędy zanim kod je zawierający mógłby doprowadzić do uszkodzenia robota. Taka metoda przypomina klasyczne testowanie zstępujące, gdzie elementy niżej położone w hierarchii są zastępowane przez zaślepki.

W czasie realizacji autor stosował testowanie oparte na dwóch metodach analizy. Testowanie oprogramowania można wykonywać pod kątem analizy statycznej i dynamicznej. Analiza statyczna polega na sprawdzaniu kodu źródłowego i znajdowaniu w nim błędów bez uruchamiania sprawdzanego kodu. Ta metoda była stosowana poza laboratorium, gdzie brak był dostępu do sterownika i modelu. Podczas analizy dynamicznej oprogramowanie jest uruchamiane i badane pod kątem ścieżki przebiegu i czasu wykonywania. Ta metoda z kolei była najważniejsza i często wyniki tych testów były zaskakujące w stosunku do przeprowadzonych wcześniej z zastosowaniem analizy statycznej.

Ostatnim etapem testów były te przeprowadzone w obecności promotora oraz te wykonane przez niego. Ostatecznie oprogramowanie zostało zatwierdzone i uznane za spełniające wszystkie wstępne założenia przedstawione w podrozdziale 1.2.

## 3.2 Napotkane problemy

Podczas tworzenia projektu napotkane i przeanalizowane zostały następujące problemy:

- Adresy w sterowniku a zmienne symboliczne:

  Po zdefiniowaniu zmiennej symbolicznej w tablicy Symbols zostaje ona użyta w bloku stworzonym w języku Ladderu. Następnie zostaje dokonana zmiana adresu tej zmiennej, co niestety powoduje, że we wspomnianym wcześniej bloku zostaje pozostawiony pierwotny adres, zamiast zostać automatycznie zaktualizowany do nowej wartości. Niestety o spójność adresów i nazw symbolicznych programista musi zadbać samodzielnie.

- Brak przenośności adresów i nazw symbolicznych ze Step 7 do WinCC flexible:

  Niestety nie udało się odnaleźć opcji pobrania adresów ze sterownika lub wyeksportowania w środowisku Step 7 i zaimportowania w WinCC flexbile. Z tego powodu dopisywanie lub edytowanie adresów zmiennych w sterowniku wymagało pamiętania o zrobieniu tego samego w środowisku do tworzenia wizualizacji.

- Bezwładność silników:

  Podczas pozycjonowania silnika do wybranej pozycji dochodziło do oscylacji w okolicach wybranej wartości. Silnik po wyłączeniu, siłą bezwładności wykonał jeszcze ruch powodujący naliczenie jednego lub dwóch impulsów, więc oprogramowanie próbowało skorygować tę wartość do tej zadanej wcześniej. Czasami udało się uzyskać żądaną pozycję, a czasami silnik pracował nieprzerwanie w obie strony naprzemiennie.

- Problem ze zliczaniem impulsów:

  Silnik inaczej nalicza impulsy podczas ruchu w kierunku wyłącznika krańcowego a inaczej w kierunku przeciwnym. Przykładowo ustawienie silnika Rotate na 100 i powrót do 0 powodowało zatrzymanie silnika gdzieś w okolicach pozycji odpowiadającej wartości 15. Problem związany był z wykorzystaniem zmiennych tymczasowych do przechowywania zmiennych pomocniczych związanych z ustawianiem kierunku pracy silnika. Błędne użycie danych powodowało, że pojedyncze impulsy były naliczane w sposób nieprawidłowy. Ze względu na zastosowane rozwiązanie trzeba było przechowywać te wartości dla kolejnych wywołań bloków, dlatego rozwiązaniem okazało się przeniesienie tych zmiennych z tymczasowych do statycznych.

Wszystkie problemy zostały rozwiązane i w ostatecznej wersji oprogramowania nie wpływają one w negatywny sposób na pracę modelu. Niestety dwa pierwsze problemy nie posiadają dobrego rozwiązania, więc należało statycznie analizować tworzone oprogramowanie celem uniknięcia błędów z nich wynikających.

# 4 Wnioski

Tworzenie rozbudowanego oprogramowania do obsługi robota 3D pracującego w magazynie wysokiego składowania wymaga rozwiązania wielu problemów programistycznych. Autor w pierwszej kolejności zapoznał się z kursami programowania w środowisku Step 7 oraz WinCC flexible [7, **?**, **?**] dostępnymi w sieci.

W wyniku realizacji projektu inżynierskiego, którego dotyczy niniejsza praca powstało oprogramowanie dla sterownika przemysłowego Siemens S7-300 sterujące modelem robota firmy Fischertechnik oraz wizualizacja stanu robota wraz ze stanem obsługiwanego magazynu wysokiego składowania.

W pierwszej kolejności powstał kod sterowania pojedynczym silnikiem, który po gruntownym testowaniu podlegał dalszemu rozwojowi. Wszystkie silniki mają możliwość pracowania w trybach: automatycznym, ręcznym z dołączonego do sterownika zadajnika sygnałów dyskretnych oraz ręcznym z poziomu stworzonej wizualizacji. Tak przygotowane oprogramowanie zostało wykorzystane do stworzenia kodu umożliwiającego obsługę zbudowanego przez autora modelu magazynu z wykorzystaniem kolejki zadań do realizacji. Wszystkie operacje wykonywane na magazynie są realizowane z wykorzystaniem zaimplementowanej kolejki FIFO, od momentu kliknięcia przycisku w wizualizacji aż do jej opróżnienia.

Podczas realizacji zostały rozwiązane problemy, z których najtrudniejszym do wyeliminowania zdaniem autora była bezwładność silników. W jej wyniku silnik po wyłączeniu dalej poruszał się przez nieokreślony czas aż do samoistnego zatrzymania. Wyeliminowanie negatywnych efektów tego zjawiska pozwoliło stworzyć w pełni funkcjonalne oprogramowanie spełniające wszelkie wymagania użytkownika.

W procesie tworzenia oprogramowania autor zapoznał się z wieloma zagadnieniami typowymi dla sterowników firmy Siemens. Przykładowo, bloki wywoływane podczas startu sterownika (OB100 / OB101 / OB102), z czego na sterowniku S7-300 dostępnym w laboratorium dostępny jest tylko blok OB100, czyli tzw. gorący restart (ang. *warm restart*). Wywoływany jest on przy każdym przejściu ze stanu STOP do RUN/RUN-P oraz podczas uruchamiania po zaniku zasilania. Kolejne charakterystyczne bloki to te wywoływane jako cykliczne przerwania (OB30 do OB38). Tutaj niestety również występuje ograniczenie i dostępny jest jedynie blok OB35, wywoływany domyślnie co 100 ms. Częstotliwość wywołania można jednak łatwo zmienić w konfiguracji sprzętowej jednostki centralnej.

Wykonanie pojedynczego cyklu sterownika stworzonego przez autora oprogramowania w czasie testowania wynosiło od 1 do 3 ms. Czas ten jest zadowalający biorąc pod uwagę, że sterownik bez funkcji użytkownika oraz z pustym blokiem OB1 ma czas wykonania cyklu równy 1 ms oraz to, że oprogramowanie przynajmniej zdaniem autora jest rozbudowane.

Wizualizacja stanu robota stanowi bardzo atrakcyjną formę korzystania z urządzenia, która jest jednocześnie wyjątkowo przystępna dla osób nie znających zagadnień związanych z programowaniem sterowników przemysłowych. Wizualizacja będzie pełniła ważną rolę w prezentacji projektu na potrzeby koła naukowego Industrum.

Bardzo ciekawymi zagadnieniami poznanymi przez autora w czasie realizacji projektu były kwestie bezpieczeństwa. Ważne jest, aby komercyjne i praktyczne projekty realizować ze szczególnym naciskiem na bezpieczną pracę robotów i innych obiektów przemysłowych ze względu na pracujących w ich otoczeniu ludzi oraz na bardzo wysokie koszty zakupu

i naprawy takich urządzeń. Przykładowym rozwiązaniem z tego zakresu jest zastosowanie przycisków monostabilnych do załączania silników, co pozwala na natychmiastowy i automatyczny powrót do położenia neutralnego z chwilą ich zwolnienia przez operatora (funkcja „dead-man-control“). Innym rozwiązaniem podnoszącym bezpieczeństwo jest zastosowanie kilku przycisków do awaryjnego zatrzymania pracy modelu. Eliminuje to konieczność precyzyjnego działania w sytuacji kryzysowej.

Autor planuje kontynuować pracę nad projektem na potrzeby koła naukowego Industrum oraz ewentualnie jako projekt magisterski. Ciekawym rozwinięciem byłoby rozbudowanie modelu o gąsienice albo koła umożliwiając mu tym samym poruszanie. Umożliwiłoby to zastąpienie półkolistego magazynu typowym szeregowym magazynem wysokiego składowania. Magazyn taki mógłby być zdecydowanie większy, a umożliwiając całemu modelowi poruszanie się, również w osi pionowej, zwiększyłoby te możliwości jeszcze bardziej. Dodanie tego elementu wymaga zastosowania technologii bezprzewodowej do komunikacji między modelem, a sterownikiem. Dobrym pomysłem zdaniem autora byłoby rozbudowanie magazynu o czujniki oraz platformy wejściowe i wyjściowe magazynu np. taśmociągi. Biorąc pod uwagę inne modele dostępne w laboratorium bardzo interesująca wydaje się perspektywa połączenia ich wszystkich w jedną całość, uzyskując w ten sposób dość rozbudowaną linię produkcyjną.

# 5 Bibliografia

Literatura, która została wykorzystana przez autora w czasie powstawania projektu, którą opisuje niniejsza dokumentacja.


[1] Jerzy Kasprzyk: *"Programowanie sterowników przemysłowych"*, Wydawnictwa Naukowo-Techniczne WNT, Warszawa, 2007

[2] *"Programowalne sterowniki PLC w systemach sterowania przemysłowego"*, Politechnika Radomska, Radom, 2001

[3] Andrzej Maczyński: *„Sterowniki programowalne PLC. Budowa systemu i podstawy programowania"*, Astor, Kraków, 2001

[4] Zbigniew Seta: *„Wprowadzenie do zagadnień sterowania. Wykorzystanie programowalnych sterowników logicznych PLC."*, MIKOM Wydawnictwo, Warszawa, 2002

[5] Janusz Kwaśniewski: *„Programowalne sterowniki przemysłowe w systemach sterowania"*, Wyd. AGH, Kraków, 1999

[6] Dokumentacja producenta: *„SIMATIC Working with STEP 7 - Getting Started Edition"*, marzec 2006.

[7] Materiały szkoleniowe: „SIMATIC S7 - Kurs podstawowy"

# 6 Spis rysunków, tablic i kodów źródłowych

## 6.1 Spis rysunków



## 6.2 Spis tablic



## 6.3 Spis kodów źródłowych

# 7 Załączniki

- Oświadczenie o autorstwie,
- Płyta CD, na której znajdują się:
  - Kod oprogramowania wewnętrznego oraz pliki projektu Step7,
  - Kod wizualizacji oraz pliki projektu WinCC flexible,
  - Plik wykonywalny wizualizacji typu WinCC flexible RT document,
  - Projekt magazynu wykonany w programie Blender,
  - LaTeXowe pliki pracy inżynierskiej,
  - Zdjęcia magazynu oraz robota,
  - Filmy prezentujące działanie projektu.