Group 567

PLC Miniproject

Anders Flodgaard
Nicklas Krogh Andersen
Carsten Larsen
Laura Montesdeoca Fenoy

1. Introduction

The goal of this assignment is to develop a PC program that determines the behaviour of a physical system, and a PLC program in charge of controlling the physical system. This assignment is focused around Smart Production Lab, an automated assembly line designed to produce smartphones. The production line behaves as followed: When a pallet arrives to the desired module, sensors will read it's RFID tag. This information is then sent to a PC via TCP/IP. The TCP/IP server, in this case the PC program, will then decide how to process the pallet based on its ID tag. Then the decided action is sent to the PLC program via TCP/IP and the action will be executed on the PLC.

Details of necessary hardware and software components

This project focuses around the industry 4.0, where this setup is operated as a research facility, with all the modern equipment for a production line. This system allows for full reconfiguration of the line as all units are mobile and modular.

Hardware description:

The Smart Production Lab in AAU, is a state of the art modular cyber physical assembly line of PLC's, which are connected in order to simulate industry 4.0 production line for research purposes. This line contains several components like: magazine for feeding components, drilling unit, assembly robot, inspection camera, a PLC festo line, manual rework or packing station, mobile robot, conveyor belt and drones to transport the items in the line. This system can either be set up in a simulation tool or as a real system. In this line, when a product is under development, a shop floor control number (SFC), is used by the RFID scanners. This will allow the system to track the product throughout the assembly line. All this information is stored in the System, Application and Products Manufacturing Execution (SAP ME) system. When a production order is created in the SAP ERP system, the order is then transferred to the SAP ME system for execution. The order contains information about the material, quantity, Bill of

Materials (BOM), and routing for each order. The system contains visualization of defects logging, picking of packing box and touch mobile User nterface (UI) for the user.

Software description

Codesys is a software for PLC programming and is developed by 3S-Smart solutions in Germany. The program is free of charge and it is a tool used for industrial automation. Codesys can be used with several different languages: IL, ST, LD, FBD, SFC and CFC. Instruction List (IL) is assembler like programming. Structured Text (ST), is like pascal and C. Ladder Diagram (LD), is used as a virtual simulation. Function Block Diagram (FBD), is used for boolean and analogue expression. Sequential Function Chart (SFC), for programming sequential processes and Continuous Function Chart (CFC), is an open, less strict version of SFC. The Codesys compiler will translate the code into binary machine code. This platform can enable a developer to write plugins for the Codesys, which means that technicians can connect several modules to form a system, without the need of advanced programming skills. For this, the programmer can use visualization, integrated motion controller, where he or she can create animated masks in Codesys, where these masks are used for testing machines or plants.

2. Description and documentation of the project

Purpose of the program

The solution for this mini project consists of two programs communicating with each other. One program being the PC, functioning as the server and the other one PLC functioning as the client. When a pallet arrives at the stop block on the PLC, the RFID on the pallet will be read and the PLC will send the information to the PC. The PC will process the information and then tell the PLC what to do with the pallet. In this project some of the pallets will be held back for 3 seconds while the others will be let through, depending on the ID tag on the pallet.

Description of the programs

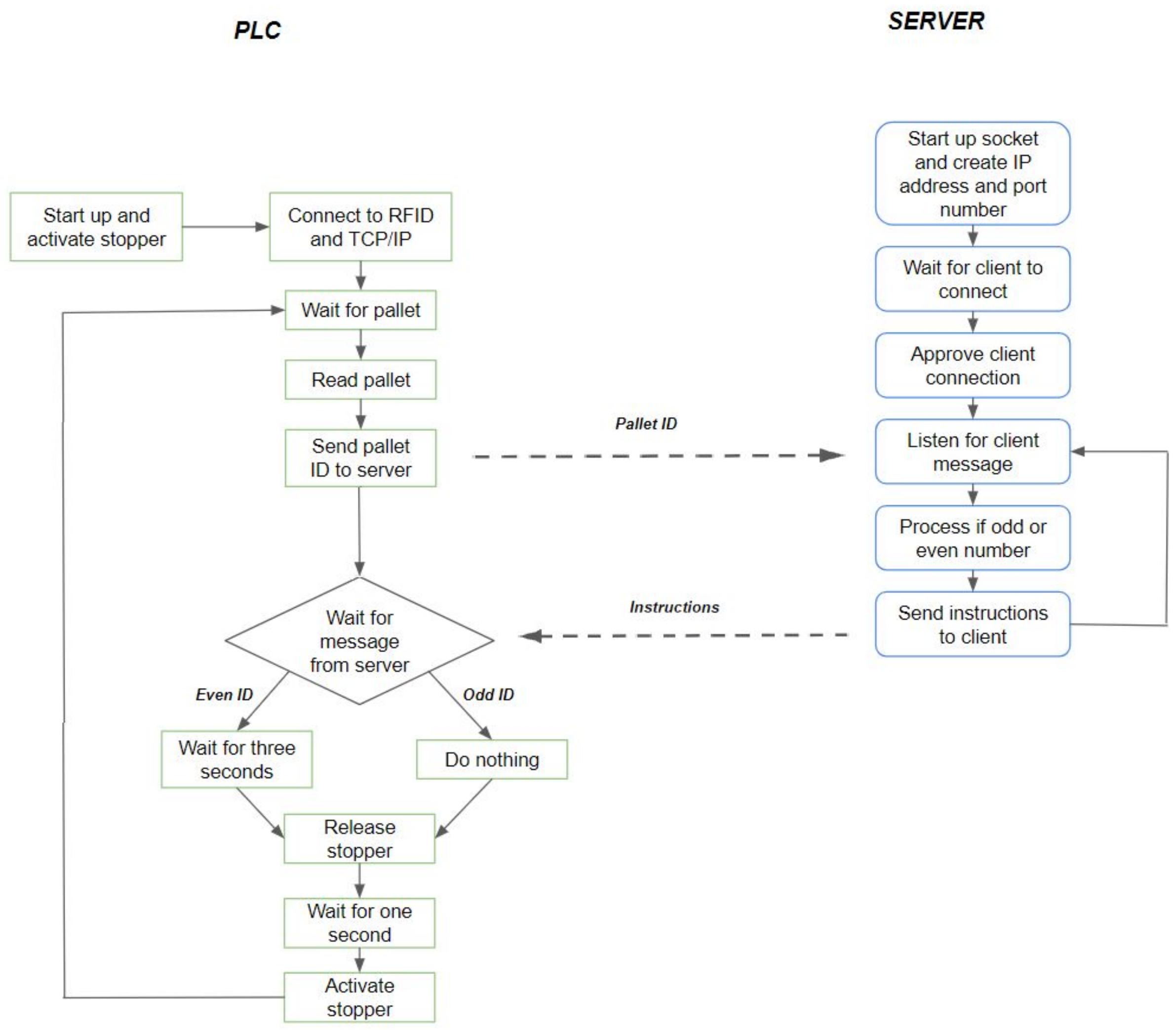
In this part the program and how it works will be explained. The full source code can be found in the appendix.

PLC

The PLC program is written in Codesys and is running on a Festo PLC. When the program starts up, it turns on the conveyors, clears errors followed by establishing connection with the RFID and PC server using the TCP/IP. When the connection is established the PLC will wait until a pallet arrives. When a pallet arrives the RFID tag will be read by the RFID and the pallets ID will be sent to the server on the PC. The PC will then send instructions to the PLC to either wait 3 seconds before releasing the stopper or releasing it right away. If the pallet's ID is an even number, it will be held back. If it is an odd number, it will be released right away. After releasing the pallet the PLC will wait for the next pallet.

PC server

The server program is written in Visual Studio 2015 as C++ code using the "#include <WinSock2.h>" library. The server starts up the socket and if any error occurs, the program will send an error message that it failed to startup. After the socket has started the server creates a fixed IP address and port number for the server that the client(s) can connect to.
After a connection with a client is established, the server gives the client a connection ID and starts listening for messages received from the client, which will be the pallet ID from the PLC. When the server receives the pallet ID from the PLC, it will process it and calculate if it's an even or odd number and then send instructions back to the PLC depending on the number.


PLC
SERVER
Start up and activate stopper
Connect to RFID and TCP/IP
Wait for pallet
Read pallet
Send pallet ID to server
Pallet ID
Wait for message from server
Instructions
Even ID
Odd ID
Wait for three seconds
Do nothing
Release stopper
Wait for one second
Activate stopper
Start up socket and create IP address and port number
Wait for client to connect
Approve client connection
Listen for client message
Process if odd or even number
Send instructions to client

The picture above shows a block diagram of the two programs.

3. Project evaluation and discussion

A manufacturing process, is a designed procedure where a material is transformed in order to increase its value. This process however, requires a system to organize the production, which in modern society is computer integrated. These manufacturing systems are expected to access, analyze, and manage vast volumes of data, and depending on their applications, they can be classified as: ERP, MES or PLC among the most common.

Enterprise Resource Planning (ERP) is an information system which organises and integrates data through a single central database. This system does not only handle manufacturing operations; but it also deals with sales, marketing, purchasing, logistics, distribution, inventory control, finance, and human resources. Therefore it makes information accessible for all the sectors across the company in a useful standardized format and common for the company. Thereby increasing efficiency and productivity.

Manufacturing Execution Systems (MES) is a system that gives manufacturers an overview of how present conditions are in the line and can thereby be used to improve and/or optimize the production line. It does this by tracking and documenting every state, that the raw material goes through until it is transformed into a product. MES can be implemented in various production areas, such as resource scheduling, product quality, production analysis, etc.

A Programmable Logic Controller (PLC) is a form of microprocessor that uses functions such as logic, sequencing, timing, counting and arithmetic to control machines and processes. With input devices that provide data, and output devices which are being controlled and it is mostly used for automation.

ERP and MES have mutual data dependencies. They both feed each other information that in turn, increases on time quality products that are flexible and meet the demands from customers, regulators, suppliers and internal staff. The MES feeds the ERP informations about, inventory, material, time spent on a part during production and the status of how far it is in production. The ERP feeds the MES with receiving sale orders and in general what is going be put in production next. Therefore a combination of MES and ERP will provide the manufacturer with a good digital ecosystem, that can help to adjust the company's business plan for better and higher production rate.

In conclusion the ERP knows “why”, while the MES knows “how to” and together they give a manufacture overview of their production and solutions on how to improve or optimize their production line.

ERP Pro’s

- Saves time and expenses.
- Faster decision making with fewer errors.
- Organization data is visible for all.

ERP Con’s

- Customization can be problematic
- High price
- Extensive training requirements

MES Pro’s

- Reduce waste.
- Quick setup.
- Increased uptime
- Incorporate Paperless Workflow Activities
- Reduced inventory.

Con’s

- It can be costly to maintain and improve the system
- It requires IT personnel
- It lacks portability if the company decides to sell the system

The developed solution features a PC and PLC program which cooperate to create a system which controls a production line. This specific solution is focused around the production only, not taking into consideration aspects such as inventory control, sales, finances or scheduling.

For this project, the PLC functions as the controller of the production line, whereas the PC program emulates a manufacturing system. It organises data through a single central database, in the same way ERP does, however this data is limited to the manufacturing process.

The RFID tags in the pallets, allow the product to be tracked in every process of the production line, enabling the users to gain a better overview of the process, as well as the possibility to document the transformation of the product in a similar way as done in MES.

In conclusion, this project shares some of the positive aspects of the modern manufacturing systems, as stated above. However, in order to make a more efficient system, certain ERP and MES features would be more desirable, such as the possibility of making a communication between the shop floor and the top floor, which would translate as a more flexible and well rounded system.

Bibliography

Bolton, W. (2015). *Programmable Logic Controllers (Sixth Edition).* Newnes.

Groover, M. P. (u.d.). *Fundamentales of Modern Manufacturing.* John Wiley and sons .

http://www.smartproduction.aau.dk/digitalAssets/221/221376_aau-smart-lab.pptx

https://www.moodle.aau.dk/pluginfile.php/1001322/mod_resource/content/2/ROB5-15%202016.pdf

Appendix

PLC code:

```
PROGRAM PLC_PRG
VAR
	UI_State: INT := 0;
	RFID: dc_ecp.Dtm425;
	DataSt: dc_ecp.st_CPF_RfidData;
	TCPClient: dc_ecp.TcpClientString;
	IP_Addr: STRING := '172.20.100.67';
	Port_Nr: UINT := 11111;
	SizeMsg: INT := 256;
	Msg: STRING := '';
	RMsg: STRING := '';
	CarrierID: DWORD;
	sCommand: INT;
	tonHolder: TON; // TIMER DELAY
END_VAR

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

CASE UI_STATE OF
	0:
		RFID.ClearError();
		GVL.xQA1_RIGHT := TRUE; //Turn on conveyor
		UI_State := 1;
		GVL.xMB20 := FALSE; //Activating the stopper
	1:
		IF RFID.xReady THEN
			UI_State:=2;
		END_IF
	2:
		//Connecting to RFID
		RFID.Connect(usiNodeId:=32, usiNetworkId:=0, usiChannel:=1);
		UI_State:=3;
	3:
		//Checking if connected
		IF RFID.xConnected THEN
			UI_State:=4;
		END_IF
```

```
        4:
                //Preparing to connect to the PC server over TCP/IP
                TCPClient.Reset();
                UI_State:=5;
        5:
                IF TCPClient.xReady THEN
                        UI_State:=6;
                END_IF
        6:
                //Connecting to the PC server over TCP/IP
                TCPClient.Connect(sIP:=IP_Addr, uiPort:=(Port_Nr));
                UI_State:=7;
        7:
                //Waiting for pallet and ready to read ID
                IF xBG21 THEN
                        RFID.ReadTag(uiStartAddress:= 0, uiDataLength := SIZEOF(DataSt),
ADR(DataSt));
                        UI_State:=8;
                END_IF
        8:
                //Save pallet ID
                IF RFID.xReady THEN
                        CarrierID := dc_ecp.SwapWORD(DataSt.uiCarrierID);
                        UI_State:=9;
                END_IF
        9:
                IF TCPClient.xConnected THEN
                        UI_State:=10;
                END_IF
        10:
                //Converting the CarrierID to string and send it to the server
                IF TCPClient.xReady THEN
                        Msg := DWORD_TO_STRING(CarrierID);
                        TCPClient.SendAndReceive(Msg);
                        UI_State:=11;
                END_IF
        11:
                //Receiving message from the server
                IF TCPClient.xReady THEN
                        RMsg := TCPClient.sReceived;
                        UI_State:=12;
                END_IF
        12:
```

```
        //Handling the instructions from the server
        sCommand := STRING_TO_INT(RMsg);
        UI_State:=sCommand;
13:
        //Case one with the odd number pallets
        UI_State:=30;
14:
        //Case two with the even number pallets
        UI_State:=20;
15:
        GVL.xMB20 := FALSE; //Activating the stopper
        UI_State:=7;



20:
        //Turning timer on
        tonHolder(IN := TRUE, PT := T#3S);
        UI_State:=21;
21:
        //Wait 3 seconds because of even pallet ID
        tonHolder();
        IF tonHolder.Q THEN
                UI_State:=22;
        END_IF
22:
        //Turning timer off
        tonHolder(IN := FALSE);
        GVL.xMB20 := TRUE; //Releasing the stopper
        UI_State:=30;



30:
        //Release stopper
        GVL.xMB20 := TRUE; //Releasing the stopper
        UI_State:=40;
        40:
                //Turning timer on
                tonHolder(IN := TRUE, PT := T#1S);
                UI_State:=41;
        41:
                //Wait for 1 second to let pallet through
                tonHolder();
                IF tonHolder.Q THEN
                        UI_State:=42;
                END_IF
        42:
                //Turn timer off
                tonHolder(IN := FALSE);
                UI_State:=15;


END_CASE


RFID();
TCPClient();
```

Server code:

```
#define _WINSOCK_DEPRECATED_NO_WARNINGS
#pragma comment(lib,"ws2_32.lib")
#include <WinSock2.h>
#include <string>
#include <iostream>

SOCKET Connections[100];
int ConnectionCounter = 0;
int clients[10];
std::string rmsg;
int PalletID;


void ClientHandlerThread(int index) //index = the index in the SOCKET Connections array
{
	ConnectionCounter += 1; //Increment total # of clients that have connected

	while (true)
	{
		char buffer[256] = ""; //Buffer to receive and send out messages from/to the
clients
		recv(Connections[index], buffer, sizeof(buffer), NULL); //get message from client
		rmsg = buffer;
		std::cout << buffer << std::endl;
		//send(Connections[index], "I dunno", sizeof(buffer), NULL); (used to make sure
something is send, dont use it anymore)


		PalletID = atoi(rmsg.c_str());

		if (PalletID % 2 == 0)
		{
			send(Connections[index], "14", sizeof(buffer), NULL);
			std::cout << "Is even" << std::endl;
		}

		else if (PalletID % 2 == 1)
		{
			send(Connections[index], "13", sizeof(buffer), NULL);
```

```
				std::cout << "Is odd" << std::endl;
			}

		}
}

int main()
{
		//Winsock Startup
		WSAData wsaData;
		WORD DllVersion = MAKEWORD(2, 1);
		if (WSAStartup(DllVersion, &wsaData) != 0) //If WSAStartup returns anything other than
0, then that means an error has occured in the WinSock Startup.
		{
			MessageBoxA(NULL, "WinSock startup failed", "Error", MB_OK |
MB_ICONERROR);
			return 0;
		}

		SOCKADDR_IN addr; //Address that we will bind our listening socket to
		int addrlen = sizeof(addr); //length of the address (required for accept call)
		addr.sin_addr.s_addr = inet_addr("172.20.100.67"); //Broadcast locally
		addr.sin_port = htons(11111); //Port
		addr.sin_family = AF_INET; //IPv4 Socket

		SOCKET sListen = socket(AF_INET, SOCK_STREAM, NULL); //Create socket to listen
for new connections
		bind(sListen, (SOCKADDR*)&addr, sizeof(addr)); //Bind the address to the socket
		listen(sListen, SOMAXCONN); //Places sListen socket in a state in which it is listening
for an incoming connection. Note:SOMAXCONN = Socket Oustanding Max Connections

		SOCKET newConnection; //Socket to hold the client's connection
		int ConnectionCounter = 0; //# of client connections
		for (int i = 1; i < 100; i++)
		{
			newConnection = accept(sListen, (SOCKADDR*)&addr, &addrlen); //Accept a
new connection
			if (newConnection == 0) //If accepting the client connection failed
			{
				std::cout << "Failed to accept the client's connection." << std::endl;
			}
			else //If client connection properly accepted
			{
				std::cout << "Client Connected!" << std::endl;
				Connections[i] = newConnection; //Set socket in array to be the newest
connection before creating the thread to handle this client's socket.
				ConnectionCounter += 1; //Increment total # of clients that have
connected
				CreateThread(NULL, NULL,
(LPTHREAD_START_ROUTINE)ClientHandlerThread, LPVOID(i), NULL, NULL); //Create
Thread to handle this client. The index in the socket array for this thread is the value (i).
			}
	}

	system("pause");
	return 0;
}
```