Designing a control system for the automation of a Festo production line

Integration Project IEM - Bachelor thesis

Maarten Lam - S3822435

Date: 2022-06-17
Course: Integration Project IEM (WBIE900-15.2021-2022.2)
Supervisors: dr. T. Mokabber, dr. G.K.H Larsen

Abstract

For years the course IEM has been giving the course PMS. During this course, students get the unique experience of working with real production line simulations. These simulations are programmed via LabVIEW. However, when a robotic arm is placed in between, which is programmed in MATLAB, a problem arises. As these three robots work together, they must be programmed inside the same application. This research focuses on the complexity of programming three different robots to work as a production line, where these robots are not made for each other. In addition to this, the robotic arm developed by the DTPA is further researched. Research concerning the robotic arm mainly focuses on whether the robotic arm has enough precision and how to increase this. For the robotic arm, a control system has been developed, as well as identifying the necessary motors. Only using these motors makes it possible for the robotic arm to work effectively in between the two Festo stations. In addition to this, the programming of the Festo stations can be easily transferred to MATLAB containing the same educational value as in LabVIEW.

Contents

1 Introduction

The world of automation is constantly progressing. This is due to companies that are always looking at ways to progress. Automation is interesting because for many companies minimizing human intervention means they can optimize their production line and therefore position themselves better in the market [1, 2]. For this, continuous research and improvement of different processes is required. The Discrete Technology & Production Automation (DTPA) is a research group that provides students and researchers a leading education and research environment. A proven way to gain knowledge about automation processes is by simulation [3]. To simulate product processing, a miniature industrial automation process, Modular Production System (MPS), can be used [4]. The production line can be simulated using MPS, which contributes to the educational learning process. Generally, an MPS is implemented using a Programmable Logic Controller (PLC) [5]. These MPS from DTPA are also implemented using a PLC [6, 7].

In this research, two different MPS stations will be used. These have been manufactured by Festo Didactic, which is a daughter company of Festo [8]. Both these stations can complete tasks individually. These stations' tasks will be elaborated on in the coming sections. As these stations bring much educational value, for the PMS (Programming Modelling & Simulation) course, in the 1st year of the bachelor IEM (Industrial Engineering & Management), the students have to do assignments where they learn to program these stations. This programming is done in LabVIEW, a graphical programming application. This language is chosen since LabVIEW has several key features which make it a good fit for an automation environment [9].

This research focuses on automation, and the objective is to minimize the human interference needed for the production line to complete its tasks. As automation is defined: "Automation is the creation and application of technologies to produce and deliver goods and services with minimal human intervention." [10]. In this case, that means minimizing the human intervention between the two stations by transferring the workpieces (cups). In order to achieve this, a robotic arm is placed between the stations that can transfer the cups from the first station, extracting, to the second station, sorting. This robotic arm is constructed by the DTPA research group and is normally programmed using Python. For MATLAB, the DTPA has also developed an interface that can be used to control the robotic arm. Until now, this robotic arm has not yet been used together with any Festo station. The current study aims to achieve a better understanding of the difficulties and opportunities of programming these robots to become an automated production line. This knowledge will contribute to the further development of the PMS course.

In the following sections, the problem standing in the way of achieving automation will be introduced, as well as the system used to tackle this problem. Furthermore, the experimental setup of this research will be discussed. To conclude, the results that have been found during this research will be discussed.

2 Problem context

In this section, the problem that this research aims to solve will be highlighted. First, the general problem of automation will be discussed as well as the errors that can occur in this specific production line. Secondly, the problem that arises when using the needed applications will be discussed. Thirdly the scope of this research will be made clear. Finally, the general problem and research questions following this problem will be stated.

2.1 Automation

As already mentioned shortly in the Introduction, the goal of this research is to gain a better understanding of simulating the automation of a production line. The first problem is that programming the automatic control systems is complex. This is because the robots are expected to work around the clock without interruption. In addition, when communication must take place between sections, multiple interfaces are needed [11]. Because of this, it is crucial that the different robots work smoothly together and that no errors occur.

These errors can occur in two different instances. First, there could be an error in the control system's programming, which means that the control system will not be able to run and will display an error message in the output. This could be the case when for example, the code assumes a certain value to be true while in the simulation, the value is false. The second error that could occur is a mechanical error in the station. This is an error that occurs because something within the station is disrupted. For example, one MPS could get jammed either because a motor is jammed or a workpiece got stuck.

2.2 Applications

The most straightforward solution to the problem of using multiple applications is to study how to combine these two programs to communicate and run parallel to each other. However, this is a complex method and is yet to be discovered fully by academic research [12]. Since this research will mainly focus on writing the program to automate this production line instead of making it possible for MATLAB and LabVIEW to communicate with each other, the goal is to use only one of either applications to program the production line. To specify, at the end of this research, the production line should work automatically while only one application is used for the programming. This can be validated by running the actual simulation. While doing this, insight will be gained in the difficulties that arise when combining the control systems of the three stations into either MATLAB or LabVIEW. Under section System there will be a further discussion about the two applications. However, the main difference between these two applications is that MATLAB is a text based coding program and LabVIEW is a graphical based coding program. The problem that follows from combining these applications is that the way functions, loops, and timers work will be different and have to be 'translated' when transferring the code from one application to the other one [13]. In addition to this, because the code for these robots has to be understandable for first year students, it should be organized and easy to understand, which means that there is a clear order in which the code is executed.

The application that can be used to program a robot is dependent on the PLC that it runs on [14]. For the robotic arm, that the DTPA has developed, the user can choose to run it on MATLAB or Python. However it should be noted here that, normally the robotic arm runs on Python. Meaning that the most advanced control software is also only usable when programming the robotic arm in Python. In addition, for the two stations a PLC that can run MATLAB has been made available [15]. Since the development of a PLC is out of the scope, transferring the LabVIEW code of the two stations into MATLAB code is the how this research will be conducted. The two main difficulties that will be focused on are: the complexity of translating the programming of the two stations into MATLAB and the complexity of letting the three robots work together.

2.3 Scope

In order to make a specific problem context, the scope of the problem has to be defined. If this is not done correctly, the research will have less contribution. In this research, the scope is limited to only the programming of the three robots. When looking at this production line many factors can be influenced. However, for this research, only the 'standard' situation for the MPS stations will be taken into account, because of this, the simulations that will be performed in this research can be easily copied for educational purposes, which means that all the hardware will not be taken into the scope. The diameter of the cups and the speed of the conveyor belt are predetermined and cannot be changed. Besides the programming of the robots, the other thing that can be changed is the speed of the robotic arm. And of course the way the robotic arm moves, but this is part of the programming of the robotic arm.

2.4 Problem

As already mentioned in this section, this research's main problems are minimizing the probability of both programming and mechanical errors, translating the different functions from LabVIEW to MATLAB, and keeping the code organized and clear to follow. In order to solve these problems, the following research questions have been determined.

1. What are the most probable programming errors to occur in the production line?
   (a) How are these error minimized?
2. What are the most probable mechanical errors to occur in the production line?
   (a) How are these errors minimized?

3. Which kind of programming structure should be used in MATLAB?
    (a) Can "while" loops be used in MATLAB as an effective way to stop the process at a certain point
4. Is the robotic arm a suitable robot to put in between the Festo stations to automate the production line?
5. Is MATLAB a suitable replacement for first year students to program the two Festo stations?

3 System

The system used will be made up of three different 'robots'. In addition, the two applications that will be used are LabVIEW and MATLAB, lastly the used controller for the robots will be discussed. First, an overview of the given mechanical properties of the different robots will be given, in this overview only the relevant mechanical properties are discussed. Secondly, a general overview of LabVIEW and MATLAB will be given. Finally, for LabVIEW and MATLAB, some of the general advantages and disadvantages are given. In this section the controllers of the Festo stations will be discussed. As already mentioned altering the mechanical properties of the robots is not in the scope (e.g. adding stronger motors).

3.1 Robots

The setup of this production line is composed of two Festo stations and a robotic arm between them.

3.1.1 Extracting station

The first station is the storage station, this station has its own manual in which all the information about this station is combined [6]. The most important factors about this station for this project, is that it can be used as a store-in and store-out unit. In this project the storing station will be used as an extraction unit. The extracting station has the cups stored on the three levels of racks based on their colour and their material (black, red, metallic) (Fig.1). The first cup is placed on the most right position of the rack and the next one left to the last-placed one. This continuous until a rack is full. During the retrieving sequence the cups are removed from the racks and are transported to the cup holder. For this station, the inputs that have to be given by the user are: from which rack the user wants the station to extract a cup and how many cups are on each rack. This last input is important, since the station does not have sensors which gives the station the information about how many cups are on each rack. Because of this, the user has to give the number of cups on each rack as an input. The output from this station will be how many cups are left on each rack. Another note on this station is that it will place the cups on a cup holder which stays in the same position.

3.1.2 Robotic arm

The robotic arm, created by the DTPA, will pick up the cups from a cup holder and place them on the sorting station. This robotic arm will thus stand between these two stations and will be the cups' transporter from one station to the other. The position and movements of the robotic arm are controlled by 7 different motors, however for the control system there exist only 6 (Fig.2a). This is, because the third and fourth (Fig.2b) motor are combined to only operate as one through the controller (Fig.2c). The inputs for the robotic arm will be the position for each motor in an array format. The input for each motor is the angle in radius, which means that it ranges from -pi to +pi. Also, the clipper will get input on how much it must close and open when it has arrived at the beginning and end positions. An important note on this robotic arm is that the movement it makes must be controlled. Since this robotic arm does not have sensors it can not decide on a good trajectory on it's own. Meaning that it could possibly want to go through the table to reach it is end position. Also, the speed of this robotic arm must be taken into account, since it has no emergency shut off button it could harm students working with this robot.

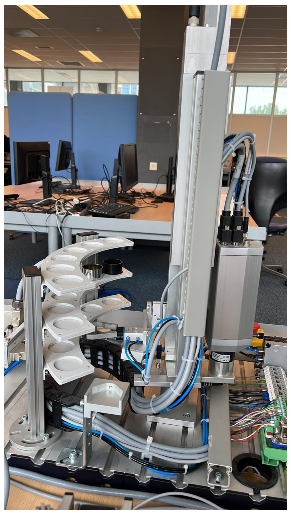
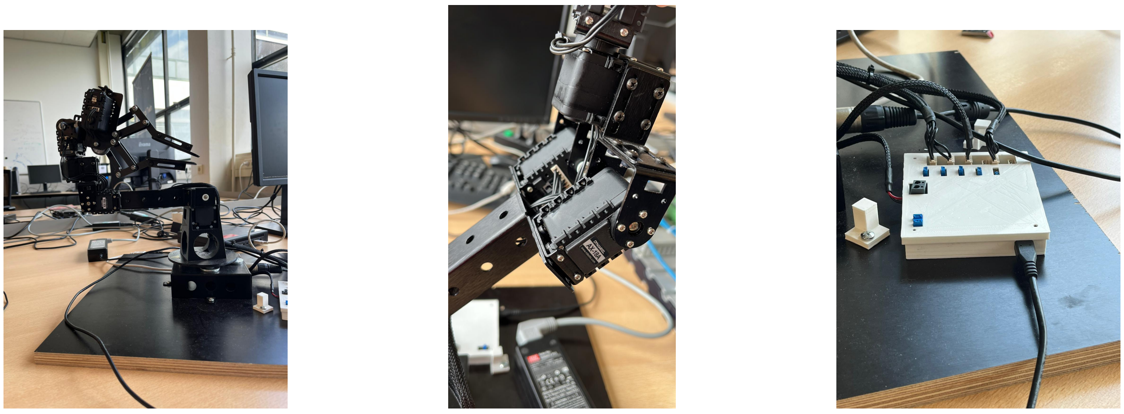


Figure 1: Extracting station


(a) Entire robotic arm

(b) Motor 3

(c) Control system

Figure 2: Robotic arm

3.1.3 Sorting station

The sorting station will sort the cups based on their colour and material (black, red, metallic) (Fig.3). When the cups are inserted at the start of the conveyor, they will pass sensors. These sensors determine the colour and material of the cup, afterwards they will give instructions to the station as to which slide the specific cup should go. The first sensor detects if the cup is metallic or not, and the second determines if the cup is black or not. Through this the correct instructions can be given. There is also a manual for this station [7]. For this station, there are no inputs, through a sensor it detects when a cup is available at the beginning of the conveyor belt and then automatically start the sorting process. In a later stage, an input could be added, where it gets a signal from the robotic arm when the robotic arm has placed a cap on the conveyor belt. This could be done in order to make the process more reliable.

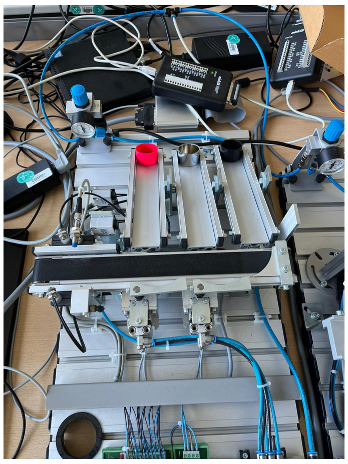


Figure 3: sorting station

3.2 Applications

3.2.1 LabVIEW

As already mentioned LabVIEW is a programming environment well suited for programming automation processes. In addition to this, LabVIEW uses a graphical representation as interface [9]. This makes it easy to overview and change according to the wishes of the user. Because of this, LabVIEW has much educational value for students just learning how to program industrial processes [3, 16]. For both Festo stations there is already a base LabVIEW code that can be worked from, as an example below the IO of the sorting station is shown (Fig.4).

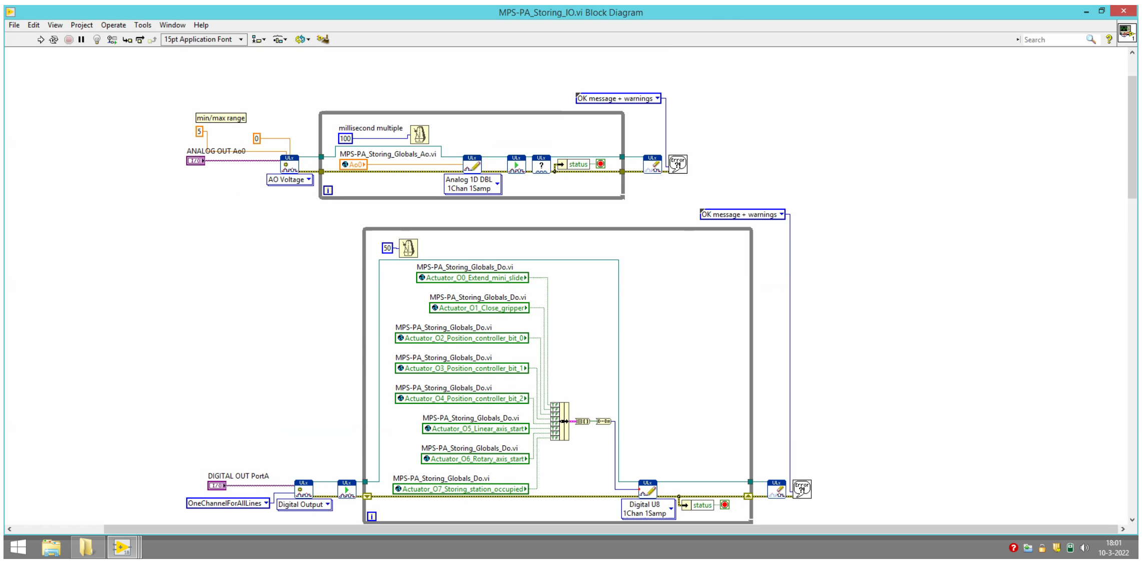
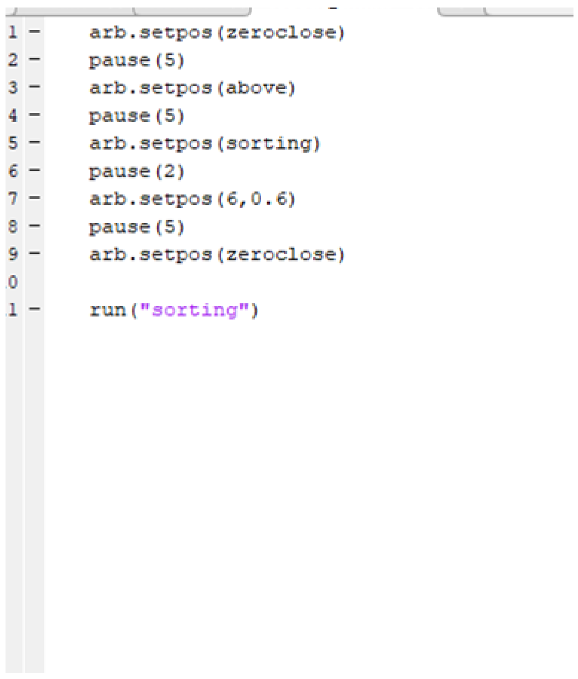

MPS-PA_Storing_IO.vi Block Diagram
File Edit View Project Operate Tools Window Help
15pt Application Font
Search
OK message + warnings
min/max range
5
0
ANALOG OUT Ao0
millisecond multiple
100
MPS-PA_Storing_Globals_Ao.vi
Ao0
AO Voltage
Analog 1D DBL 1Chan 1Samp
status
OK message + warnings
50
MPS-PA_Storing_Globals_Do.vi
Actuator_00_Extend_mini_slide
MPS-PA_Storing_Globals_Do.vi
Actuator_01_Close_gripper
MPS-PA_Storing_Globals_Do.vi
Actuator_02_Position_controller_bit_0
MPS-PA_Storing_Globals_Do.vi
Actuator_03_Position_controller_bit_1
MPS-PA_Storing_Globals_Do.vi
Actuator_04_Position_controller_bit_2
MPS-PA_Storing_Globals_Do.vi
Actuator_05_Linear_axis_start
MPS-PA_Storing_Globals_Do.vi
Actuator_06_Rotary_axis_start
MPS-PA_Storing_Globals_Do.vi
Actuator_07_Storing_station_occupied
DIGITAL OUT PortA
OneChannelForAllLines
Digital Output
Digital U8 1Chan 1Samp
status
18:01
10-3-2022

Figure 4: Screenshot of the Storing.IO.vi layout

As these Festo stations are programmed often (each PMS course) through LabVIEW, there will not be much attention to the actual programming in LabVIEW. The only aspect that will be discussed is the process of the extracting station, since this is a bit more complex than the sorting station. This is, because the already given LabVIEW lay-out is not designed for this task and thus must be altered

3.2.2 MATLAB

As already mentioned, the robotic arm normally programmed in MATLAB, this means that this application will also be researched. The programming style of MATLAB is structured text, which means that the commands are given via text commands. MATLAB is a programming application that is used mainly for matrices and other mathematics [17]. This is convenient for the robotic arm, since the position of the different sections is expressed in a matrix format. For the robotic arm the controller is already set up (Fig.2c). However, as already mentioned, when the input for the robotic arm is the position that the user wants the robotic arm to finish at the robotic arm will behave chaotically. To specify the robotic arm will move all the motors at the same time, without taking its environment into account. This means that for the robotic arm a code must be written which makes sure that the movement of the robotic arm is as smooth and controlled as possible. Because the safety is of upmost importance. When the safety of the robotic arm cannot be guaranteed, this research will lose its value. An example of such a code is given below in Fig.5, where the robotic arm moves from its starting position to the position of the cup, closes its clipper, moves back to the starting position, in order to make sure the trajectory is safe, and then moves to the station where it drops the cup. In Fig.5, it can be seen that the robotic arm gets the following position commands: (line 1) go to the initial position, (line 3) go above the sorting station, (line 5) go to the position where the cup touches the conveyor belt, (line 7) open the clipper to release the cup, (line 9) go back to the initial position. This is however a simplified example of the code. The actual code that is used for the robotic arm is discussed later on.



```
1 -     arb.setpos(zeroclose)
2 -     pause(5)
3 -     arb.setpos(above)
4 -     pause(5)
5 -     arb.setpos(sorting)
6 -     pause(2)
7 -     arb.setpos(6,0.6)
8 -     pause(5)
9 -     arb.setpos(zeroclose)
0
1 -     run("sorting")
```

Figure 5: Example of MATLAB code for the robotic arm

3.2.3 Overview

As mentioned above, the biggest difference between LabVIEW and MATLAB is that LabVIEW is a graphical programming application and MATLAB structured text based. As the code for LabVIEW has to be transferred to MATLAB multiple decisions have to be made regarding the MATLAB code. The following arise: will all the code be in one MATLAB script, will each task be a separate function or loop, how will the functions or loops be activated, how will the code communicate between the different stations, how will the code keep track of the amount of cups on each rack. These are important questions on which decisions have to be made. The decisions have to be made during the programming and simulating of these robots. These answers will contribute to the knowledge of transferring the LabVIEW code to MATLAB.

3.3 Festo controller

As already mentioned the Festo stations are programmed via LabVIEW. However in order to be able to program them via MATLAB, a new controller had to be developed. This was done by the DTPA research group. This controller was made on an Arduino board which then could be programmed via Arduino software. In order to be able to control the stations via MATLAB an add-on had to be installed, which could communicate between MATLAB and the Arduino board controlling the station. An important note that must be made here is that for this research only digital in and outputs will be assumed. Meaning that all the in and outputs to the Festo stations will either be a 1 or a 0. However, since the development of the Arduino board is out of this scope, this board will not be discussed further. Under Code the programming of the Arduino board is discussed thoroughly.

4 Experimental setup

For this research to be successful, the production line should be able to run without any errors for 5 iterations. In order to validate this, the entire production line has to be simulated. This is done by an experimental setup. However, in order to make sure that each robot worked, the two Festo stations and the robotic arm were set up and tested individually. The code was written for the Festo stations in LabVIEW, this was tested. Afterwards, the robotic arm was tested multiple times to get an insight in how its movements corresponded to the inputs given. These tests were successful, meaning that the three different robots could perform their tasks individually. Finally it was all tested together, to see if the production line would work when driven by different applications that run at the same time (Fig.6). This was successful, where the cup could go through the entire process without any human interference.

For the next part of the research, the Festo stations had to be run via MATLAB, for this a new controller had to be designed. As already mentioned, this was done by using an Arduino board which could be connected to the station instead of the controller that was run on LabVIEW. To obtain a better understanding of the Arduino software, first it was tested if it was possible to program the two stations through Arduino software, as the necessary MATLAB add-on was not available at first. Meaning that for both stations Arduino code was written and tested. Because Arduino code is written in C++, which is very similar to MATLAB, this already gave some insights in how to program the two Festo stations in MATLAB. Through the Arduino board and software the different actuators and sensors can be either "read" or "written". Meaning that the data from the sensors can be read, and the movement of the actuators can be given as input. An example of this Arduino test code can be seen in Appendix. Afterwards to connect MATLAB and the Arduino board the *"MATLAB Support Package for Arduino Hardware"* was installed. With this add-on the two Festo stations could be programmed in MATLAB.

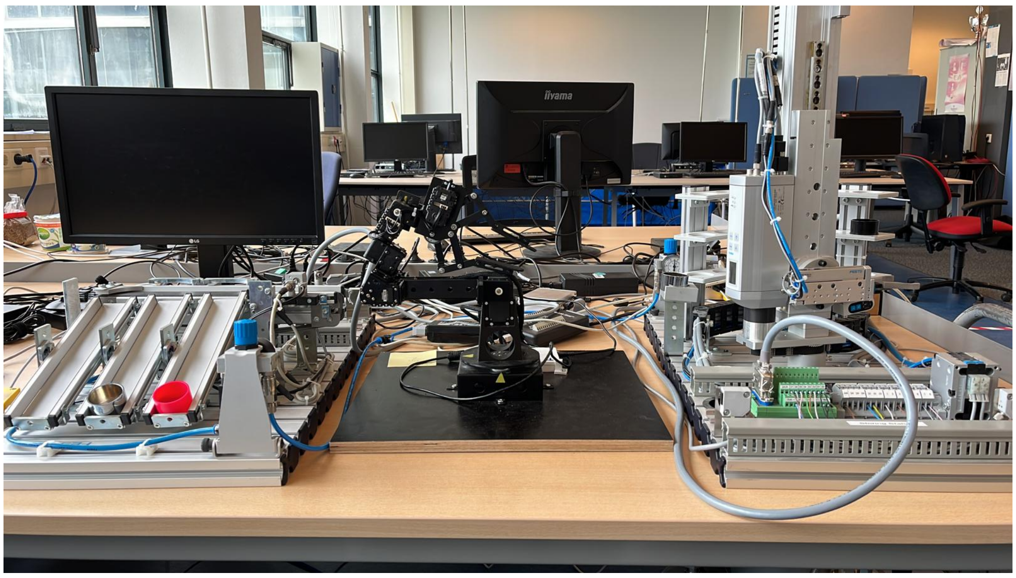


Figure 6: Experimental setup of the production line

5 Results

A process diagram has been constructed to understand what the tasks are for each robot and how this should be programmed/set up. This was the first step in this research, and this process diagram will be explained later. The second step in this research was to program the two stations in LabVIEW. Thirdly the robotic arm was placed in between, and the MATLAB code for the robotic arm was written. Afterward, these three codes were run together to see if the simulation would be successful. Lastly, the two stations were programmed in Arduino software, after which they were programmed in MATLAB. After each of these developments, multiple test runs were carried out. In order to obtain results from each development, these runs could either be carried out by one specific robot or by the entire production line. The results obtained during the programming and tests will be discussed in this section. This will happen in two parts. First the knowledge obtained regarding the mechanical aspects of the robots will be discussed. This will mainly focus on the relevant mechanic factors that came into play during the tests. Secondly, the knowledge and alterations regarding the programming of the robots will be analyzed, in this part, the process diagram that has been made for the entire production line and for each robot individually will also be explained. First, the information regarding the robotic arm will be reviewed for both parts. Afterwards the results regarding the extracting and the sorting station will be discussed, respectively. For Code first, the knowledge that was obtained during the research will be discussed. Afterwards, the process diagram and the code will be explained for each part of the production line.

5.1 Mechanical aspects of the robots

First, the results relevant to the mechanical aspects of the robots will be discussed. Throughout the research some noteworthy mechanical aspects were found, however the alteration of these properties is not in the scope, and these have thus not been performed. During the tests, some robots showed inaccuracies or errors, in order to fix these, minor "reparations" had to be done, these will be discussed. To conclude, the most significant mechanical note is that to increase the production line's reliability, the different robots should be attached to a position that cannot be changed. As this was not the case for this research, for every test the robotic arm had to be calibrated to the locations of the extracting and sorting station.

5.1.1 Robotic arm

For the robotic arm, the first insights came when running the test movements. The first thing that was noted was the safety, since the robotic arm has no sensors, it does not know what is in its surroundings. This means that when a position is given to the robotic arm to go to, it does not take its surroundings in account. This could for example lead to the robotic arm wanting to move through the table. Because of this it is very important to calibrate and test each step of the robotic arm before letting it run in the production line. By first testing each step, it can be made sure that the robotic arm does not collide with its surroundings. In addition to this it was noted that the robotic arm was not accurate enough to complete its task more than two times in a row. It was observed that this was caused by two factors. Firstly when the robotic arm moves all motors at the same time this gives a chaotic movement which in turn increases the chance of the cup flying out of the clipper. This chance must be minimized for the process to be automatic, the solution that was found for this is discussed under Code. Secondly the robotic arm moves too fast. When it moved to a position it overshoots almost every time because of which it must correct. Due to this overshooting and correcting the precision is decreased. This leads to issues when the robotic arm is not able to go to the same position to pick up and release the cup each time.

To solve this, firstly the speed of the DC motors has been decreased, this gives the robotic arm overall more precision. To give the robotic arm even more precision and to control it more refined, only the motors that are needed will be used. This means that only three motors will be used, these are 1, 3, 5, 6 (Fig.7). Only using these motors gives the robotic arm still enough freedom to complete the task of moving the cups. The motors that have been selected are all needed to complete the task. Firstly motor 1 is used for the rotational movement, the robotic arm must rotate from the extracting to the sorting station. Motor 5 is used to keep the clipper close to the rest of the robotic arm, making sure that it does not collide with the stations or other objects surrounding the production line (Fig.8). Motor 6 is the clipper, which is obviously used to grab and release the cup. The most interesting choice regarding motors has been the choice between motor 2 and 3. Both motors are able to move the robotic arm vertically, this is needed because the output of the extracting station is located lower than the conveyor belt of the sorting station. As already mentioned motor 3 is made up of two smaller motors, because of these smaller motors, motor 3 has the most trouble with handling the torque and overheating. This means that when the robotic arm is used, motor 3 has the most problems with maintaining its set position. Because of this motor 3 has been chosen to move during the process. This has been done, because now motor 3 is set to specific positions during the process, which decreases the chance of motor 3 to differ from its set position. During the tests it became clear that after multiple movements from motor 1, the bolts that attached the robotic arm to the plate became loose. Because of this during the research these bolts were screwed tighter again, while doing this it is important not to screw them too tight, since the arm still must be able to rotate.

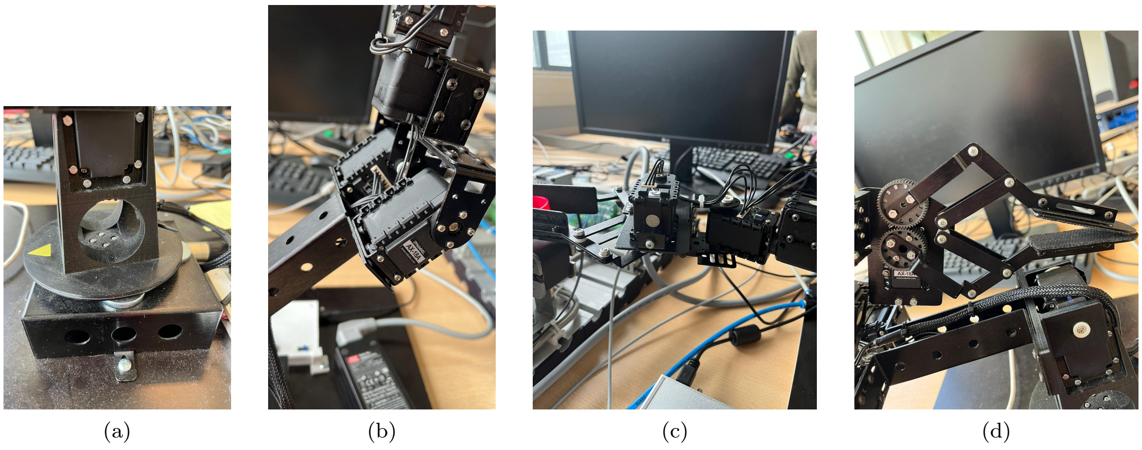


Figure 7: (a) Motor 1 rotary movements (b) Motor 3 vertical movements (c) Motor 5 horizontal movements (d) Motor 6 the clipper

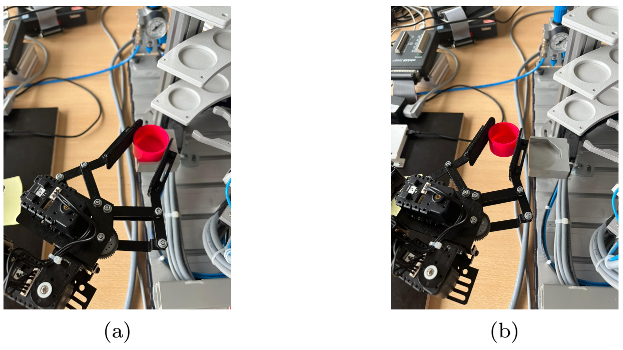


Figure 8: (a) Clipper extended (b) Clipper tucked in

5.1.2 Festo stations

For the two Festo stations only a small amount of new mechanical information has been obtained. This is because these stations have been developed for educational purposes, meaning they have been researched a lot. The new information was obtained when testing the entire production line. The reason for this is that, the two stations are developed to be in a production line with the same kind of stations, which means that the stations are adjusted and calibrated to each other. However, since the same company did not develop the robotic arm, the two stations were not made to work together with the robotic arm. This difference was the cause for the new knowledge that was obtained.

5.1.2.1 Extracting station

As already mentioned the extracting station is actually developed and normally used as a storing station. Where instead of extracting the cup from the rack to the output, it stores the cup from the output in to the rack. However as reversing this task is not too complex, the station operates well. It was found however that the extracting station is not consistent with placing the cup in the output. As sometimes the cup would not slide into the output, but ended skew on the surroundings of the output (Fig.9). As the clipper of the robotic arm was bigger than the cup less precision was needed, because of this it did not cause any problems. A mechanical property that will be further discussed under Code is that the first movement the extracting station must do when powered on is the homing. During this movement the clipper goes all the way up and all the way to the right, doing this makes sure that the positions of the station are set correctly.

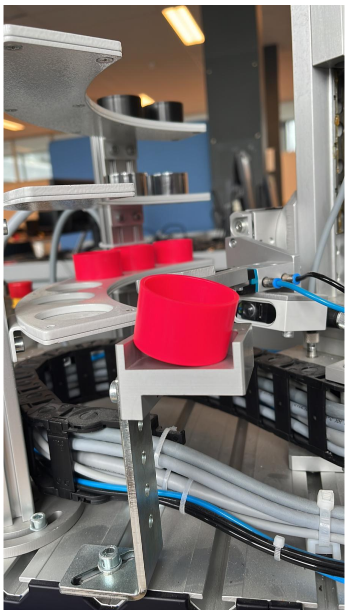


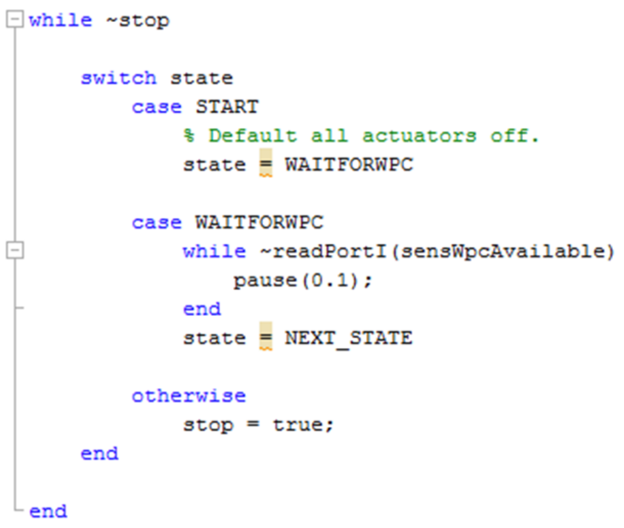
Figure 9: Cup standing skew on the output

5.1.2.2 Sorting station

For the sorting station also most mechanical aspects were not new. However, because in this production line, the robotic arm places the cup on the conveyor belt, in stead of another conveyor belt attached to the sorting station, some thought had to go into placing the cup on the conveyor belt. First of all the sorting station and robotic arm needed to be placed in such a way that when the robotic arm released the cup it would be placed at the start of the conveyor belt. This is important, because at the start of this belt a sensor indicates if there is a cup available. When the entire production line is controlled from one code, however this is easily countered by the fact that the sorting station will get a signal from the robotic arm when it has released a cup.

5.2 Code

As all the mechanical aspects relevant to this research have now been highlighted, the knowledge obtained relating to the programming will be discussed. In this section the same order will be used as in the previous section. First for each robot the process diagram will be discussed. In this process diagram all the steps that the robot must take to complete its task are visualised. The process diagram is a good visualizer to see what functions have to be written. Each task that must be done can be written as a function, that can be called to complete certain tasks. In addition to this case structures can also be used in MATLAB. Where a certain section of code will be run when a certain condition is met. An example of this can be seen in Fig.10. Afterwards specific results concerning the robot will be discussed. In total four MATLAB scripts have been developed, which work in hierarchical order (Fig.11). As can be seen in Fig.11, the process starts with the waiting code. Where MATLAB waits until the user gives as input how much cups they want to go through the production line. At this same moment the user must give the amount of cups on each rack as input. Because of this two other codes have to give input back, Extracting and Sorting. The reason for this is that there are two possible reasons, apart from an error, that the production line must come to a stop. This is when there are no cups for the extracting station to extract or when the sorting station has a slide that is full.



```
while ~stop

    switch state
        case START
            % Default all actuators off.
            state = WAITFORWPC

        case WAITFORWPC
            while ~readPortI(sensWpcAvailable)
                pause(0.1);
            end
            state = NEXT_STATE

        otherwise
            stop = true;
    end

end
```

Figure 10: Case structure example in MATLAB

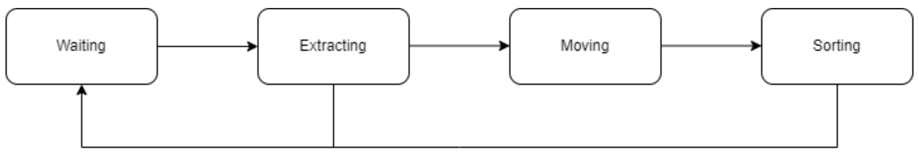


Figure 11: Simplified process diagram of the entire production line

This all happens in Waiting (Fig.12). The reason that in waiting the initial states of the different robots are not checked, is because at the end of the task of each robot there is a check if the robot has gone back to it is initial state. If this is not the case the next task will not be called, and thus the process will stop. Also after a certain amount of time/checks an error message will appear to the user. As can be seen in the process diagram of waiting the two possible errors are shown, where the user gets the choice to continue (restock the racks or empty a slide) or the option to shut the production line down. If the user decides to continue, by restocking a rack or emptying a slide the use again has to put in the amount of cups it wants to go through the production line. This code is also called upon after the sorting station is done with sorting a cup, the reason that this code is called each iteration is to check if one of the slides is full. As the sorting station does not keep track of how many cups are in each slide (there are no sensors to do this) this has to be checked each iteration. As already mentioned, the user has to give the number of cups on each rack as input to the extracting station. The first lines of waiting.m do this (Fig.13 line 1-4). These lines ask the user for how many cups are on each rack. Also, the user is asked how many cups they want to extract, this determines the number of iterations that the production line will do. The code that has been written in MATLAB can be found in the Appendix.

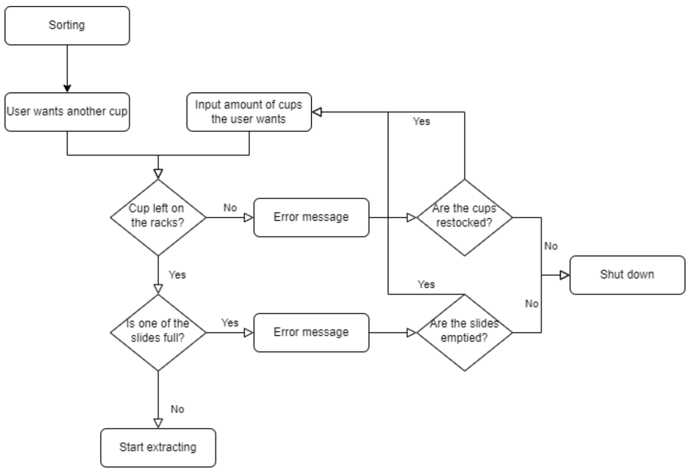
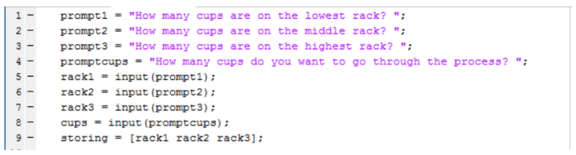


Figure 12: Process diagram of waiting



```
prompt1 = "How many cups are on the lowest rack? ";
prompt2 = "How many cups are on the middle rack? ";
prompt3 = "How many cups are on the highest rack? ";
promptcups = "How many cups do you want to go through the process? ";
rack1 = input(prompt1);
rack2 = input(prompt2);
rack3 = input(prompt3);
cups = input(promptcups);
storing = [rack1 rack2 rack3];
```

Figure 13: Asking input from the user in waiting.m

5.2.1 Robotic arm

As this robotic arm is normally not programmed and controlled via MATLAB, there were some issues that had to be fixed. Firstly during the first tests it became clear that the speed of the robotic arm was too high. This high speed caused precision issues, as the robotic arm overshot almost every time when given a new position. This was countered in two ways. Firstly the robotic arm was updated. In this update a control system was added for controlling from MATLAB. This controller made the acceleration more smooth. Meaning that it adjusted the speed to increase and slowly decrease slowly. Secondly, the maximum speed of the different motors was limited. This maximum speed was different for each robot and has to be set again whenever the robotic arm is powered on. To counter the chaotic moving of the robotic arm, and the consequence of the cup flying out of the clipper, the motors were moved separately, meaning that for a desired position, each motor goes to that position in hierarchical order. Since the robotic arm has no sensors, it has to get an input that it can start. This was resolved by programming all the robots together, since now the extracting station could give this input when it was done with its task. To conclude, it was found that for this robotic arm "while" functions worked effectively to check if a certain position is reached. However since, the robotic arm is not a 100% accurate, the condition could not be for the robotic arm to precisely reach the desired position. In order to fix this, the desired position has to be a range that is acceptable for the production line to work without errors. The task of the robotic arm is carried out in seven distinctive steps, as can bee seen in the process diagram (Fig.14). As can be seen in the process diagram, a lot of the steps are a single motor moving (either 1,3,5,6). The only steps that include more motors are: "Move to extracting station" (1,3,5) "Putting cup above conveyor belt" (motor 3,5) and "Return to middle" (1,3,5). This is, because a movement of a motor is so small, that it does not increase the chance of the cup falling, or when the cup is already placed at the sorting station. When the robotic arm has dropped the cup at the sorting station there is a wait time of 5 seconds. This has been inserted, because if the cups does not move while the robotic arm tries to go back to its initial position, there is a chance that the clipper collides with the cup. After each movement a condition has to be met, which is a range of the desired position. This is checked by a "while" function which holds the process at a stop until the condition is met. Unless after a certain time the condition is still not met, in that case, the robotic arm shuts down and an error message appears. The MATLAB code that has been written can be found in the Appendix. Before the process is started, a file is run that gives all the variables their corresponding values.

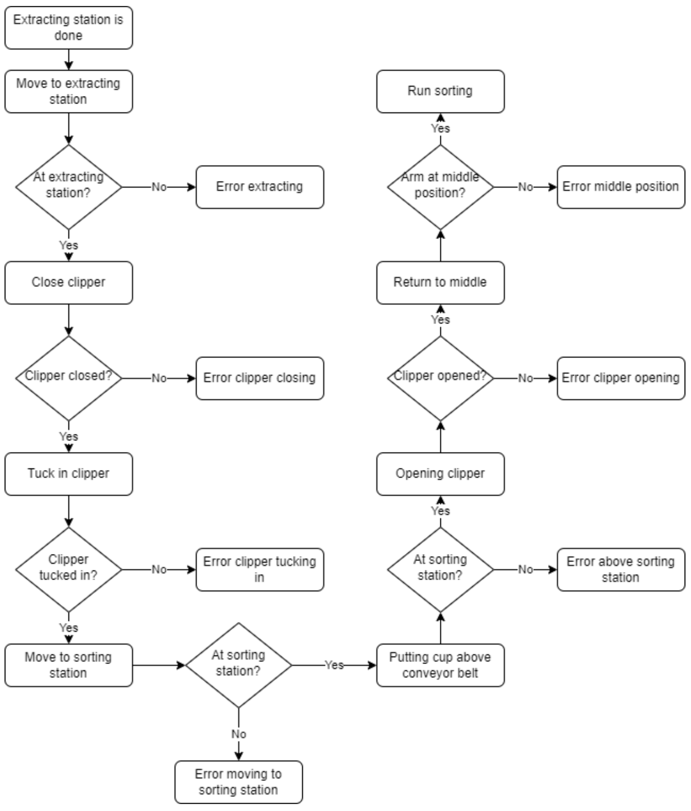


Figure 14: Process diagram of the robotic arm

5.2.2 Festo stations

For transferring the LabVIEW code to MATLAB the following insights have been gained. Firstly, because the two stations were first programmed successfully in LabVIEW, this can be taken over. In LabVIEW a case construction is used, which is basically a function that at its end calls another function. Furthermore, all these functions form a loop, where in each function, different tasks are carried out. Thus for the MATLAB code also functions have been chosen. These functions perform a part of the task in the desired order. An insight that has been obtained for the both Festo stations is that, when being programmed from MATLAB is that it is important to take a good look at the sensors. As these were sometimes inverted, where a sensor gave a true value, MATLAB saw it as false.

5.2.2.1 Extracting station

The programming of the extracting station in LabVIEW already gave some new insights, this is due to the fact that the process layout had to be reversed in some way, to extract the cups instead of storing them. The interesting part was how the station will decide which cup from which rack to take. There are a couple of ways for the station to make this choice: The station could extract all the cups from a certain rack and then move to the next rack, until all the cups are extracted, the station could choose to extract a cup from the rack with the most cups on them until all the racks are empty, let the user choose the rack from which the station has to extract cups. In the LabVIEW code it was chosen to let the user give the rack to take a cup from as input. This was done, because this was the easiest to program, since the station would not have to make the decision on which rack to go to itself. The position of the cup was however determined by the station, since the user gives the number of cups on each rack as input this makes sure that the user does not have to keep track of each cup on each rack. However, since MATLAB is a mathematical based coding application, the possibility for letting the station make its own decisions is bigger. This is, because in MATLAB it is easier to change, compare, subtract certain elements in an array. Because of this, it was chosen to let the Extracting station decide which cup to extract. The method that was chosen is that the extracting station would extract a cup from the rack with the most cups on, if all the racks had the same amount of cups, it would take a cup from the first rack and so on. As already mentioned the only inputs into the Festo station digital. However for the gripper on the extracting station there are multiple coordinates. Both the rotary and the vertical axis can have a value from 0..7. This means that when the extracting station decides on which rack it will take a cup, and on which place this cup is stored (e.g. 2,4). These two integers have to be transferred into a digital input, in this case that means into a binary number. This can then be given as input to the extracting station. Where the station has three inputs for the movement of the gripper, the first input represents the first number in binary and so on. For the extracting station the process diagram has been divided in five different steps (Fig.15). For the function "Move to desired position" the desired position is thus determined by "Choose which rack". For "Extract cup" the code is not dependent on which rack the cup is taken from, as the output is in the same position and the same movements are needed to reach the output. The entire code can be found in the Appendix. As already mentioned in this section there are three parts of the code that should be highlighted. First off in MATLAB a function move has been developed that moves the gripper (Fig.16). This function has been called pos and has the desired x and y position as input. In this function the integer of the rack and cup has to be changed to a binary value, this can easily be done in MATLAB by function dec2bin (line 11, 23). Then by a for loop, the binary code is transferred into the position controller bytes (line 12-18, 24-30). In MATLAB the station decides which cup from which rack to take, as already explained MATLAB looks at which rack has the most cups and then the station takes the cup most left on that rack (Fig.17). The function max, gives two outputs, first the highest value of the array and second the position of that value in the array (line 54). Which translates to where the cup is and on which rack respectively. From these values then the linear position that corresponds to each rack is given (line 55-61). The variable sorting is obtained in waiting.m. Whenever the extracting station is powered on (the first run it does), first the homing position should be moved to (Fig.18). As can be seen in the figure the homing is only carried out when i = 1, i is defined in waiting.m where after the first run it is increased by 1 with every run.

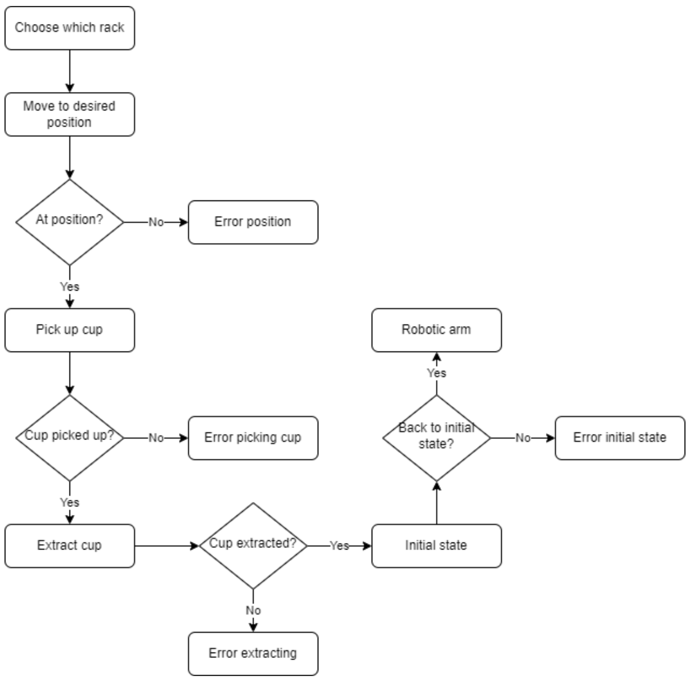
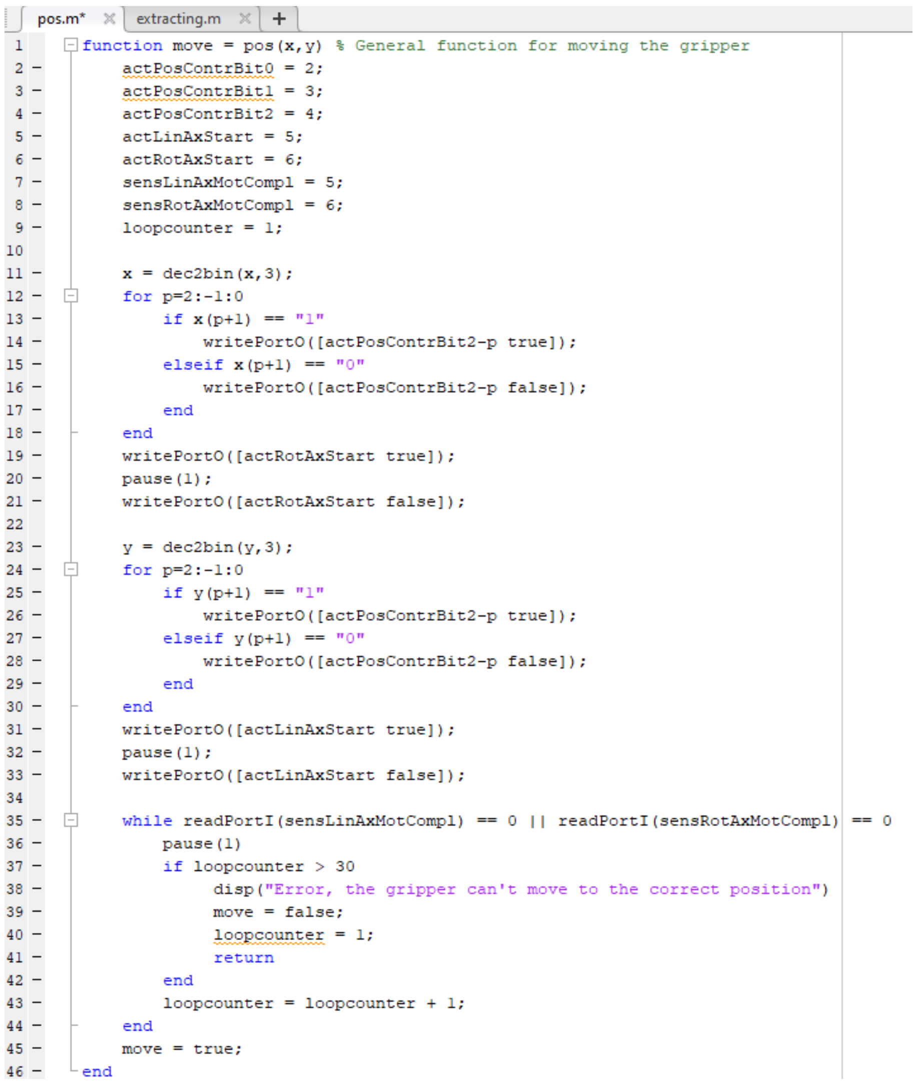
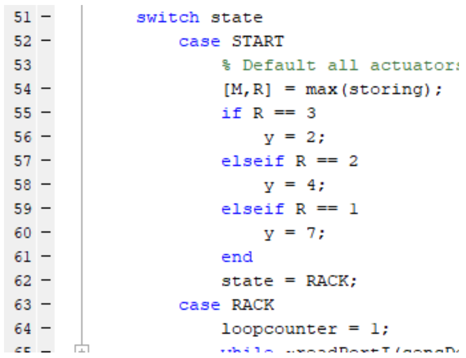


Figure 15: Process diagram of the extracting station


pos.m* extracting.m

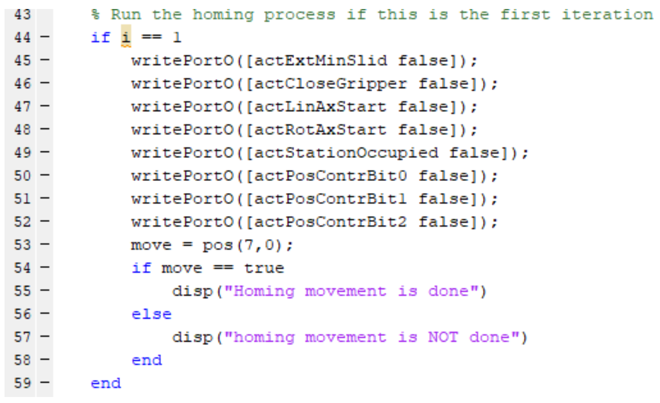
```
function move = pos(x,y) % General function for moving the gripper
    actPosContrBit0 = 2;
    actPosContrBit1 = 3;
    actPosContrBit2 = 4;
    actLinAxStart = 5;
    actRotAxStart = 6;
    sensLinAxMotCompl = 5;
    sensRotAxMotCompl = 6;
    loopcounter = 1;

    x = dec2bin(x,3);
    for p=2:-1:0
        if x(p+1) == "1"
            writePortO([actPosContrBit2-p true]);
        elseif x(p+1) == "0"
            writePortO([actPosContrBit2-p false]);
        end
    end
    writePortO([actRotAxStart true]);
    pause(1);
    writePortO([actRotAxStart false]);

    y = dec2bin(y,3);
    for p=2:-1:0
        if y(p+1) == "1"
            writePortO([actPosContrBit2-p true]);
        elseif y(p+1) == "0"
            writePortO([actPosContrBit2-p false]);
        end
    end
    writePortO([actLinAxStart true]);
    pause(1);
    writePortO([actLinAxStart false]);

    while readPortI(sensLinAxMotCompl) == 0 || readPortI(sensRotAxMotCompl) == 0
        pause(1)
        if loopcounter > 30
            disp("Error, the gripper can't move to the correct position")
            move = false;
            loopcounter = 1;
            return
        end
        loopcounter = loopcounter + 1;
    end
    move = true;
end
```

Figure 16: Pos function to move gripper



```
        switch state
            case START
                % Default all actuator
                [M,R] = max(storing);
                if R == 3
                    y = 2;
                elseif R == 2
                    y = 4;
                elseif R == 1
                    y = 7;
                end
                state = RACK;
            case RACK
                loopcounter = 1;
```

Figure 17: Rack decision in extracting.m



```
% Run the homing process if this is the first iteration
if i == 1
    writePort0([actExtMinSlid false]);
    writePort0([actCloseGripper false]);
    writePort0([actLinAxStart false]);
    writePort0([actRotAxStart false]);
    writePort0([actStationOccupied false]);
    writePort0([actPosContrBit0 false]);
    writePort0([actPosContrBit1 false]);
    writePort0([actPosContrBit2 false]);
    move = pos(7,0);
    if move == true
        disp("Homing movement is done")
    else
        disp("homing movement is NOT done")
    end
end
```

Figure 18: Homing in extracting.m

5.2.2.2 Sorting station

To start the knowledge about the sorting station that has been obtained through the simulations will be discussed. As programming the sorting station in LabVIEW is part of the PMS course, the programming has not given any new insights. However, when the simulation of the entire process was done some points of interest came to the light. Firstly the two sensors that determine the colour and the material of the cup take more time to detect the colour and material than the sensor that detects if a cup is available. This means that the stopper has to stay extracted a bit longer for the station to sort the cup in the right slide. See Fig.3 for a detailed picture. Secondly, it was noted that when the cup was placed on the conveyor belt by the robotic arm, sometimes the station would not start its process. This was due to the fact that the cup would be placed too far ahead on the conveyor belt, in such a way that the sensor that detects if a cup is available could not detect the cup. A solution for this is that when the production line is set to run automatically, the sorting station begins to run when the robotic arm is done. For the sorting station also a process diagram has been made (Fig.19). In this diagram it is clearly visible that the sorting station is the least complex. Dependent on the colour and material of the cup a certain switch is extended after which the stopper is retracted (Fig.20, line 52-59). After 5 seconds this switch is retracted and the task is done (line 61-73). The check that has to be done if any of the slides is full is, as already mentioned, done during the waiting process. The entire code for sorting.m can be seen in the Appendix.

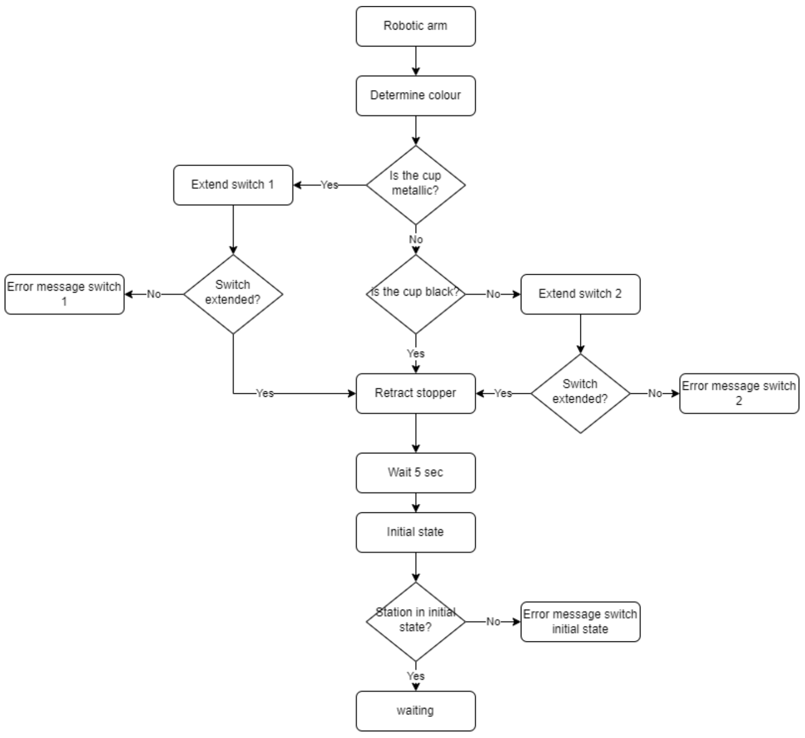
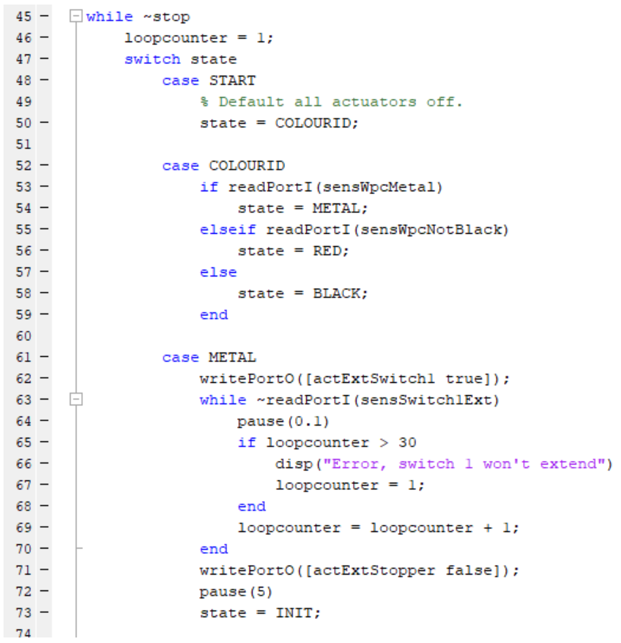


Figure 19: Process diagram of the sorting station



```
while ~stop
    loopcounter = 1;
    switch state
        case START
            % Default all actuators off.
            state = COLOURID;

        case COLOURID
            if readPortI(sensWpcMetal)
                state = METAL;
            elseif readPortI(sensWpcNotBlack)
                state = RED;
            else
                state = BLACK;
            end

        case METAL
            writePortO([actExtSwitch1 true]);
            while ~readPortI(sensSwitch1Ext)
                pause(0.1)
                if loopcounter > 30
                    disp("Error, switch 1 won't extend")
                    loopcounter = 1;
                end
                loopcounter = loopcounter + 1;
            end
            writePortO([actExtStopper false]);
            pause(5)
            state = INIT;
```

Figure 20: Identifying the colour and material in sorting.m

6 Discussion

The following section provides a further insight into the hurdles that had to be overcome during this research. First off the transfer of the robotic arm from Putty in Python to MATLAB was difficult. Being able to control the robot via MATLAB was easy, as only a switch had to be turned. However no control systems for the robotic arm existed for the MATLAB application. This meant that for quite a long time, the robotic arm was too chaotic to work in the production line effectively. It took some time and tests to see which motors could be used that would make the movements not too chaotic, as with the new control system, it is less likely that the cup is lost during a movement. However still it was chosen only to use the four necessary motors, as a minimization of the chaotic movement is desired. In addition to this, it was found that certain functions did not work the same way or at all in MATLAB, firstly the functions that were solved by an updated robotic arm during the research will be discussed. Firstly the function where the maximum speed could be set did not work. Normally for each movement a certain speed could be set, however this thus did not work. Secondly the function were instead of a certain position the user could give a path to the robotic arm to follow was not functional. Finally as already mentioned was a speed controller added to the robotic arm after the update. This controller made sure that the acceleration and deceleration of the robotic arm became more smooth, which in turn made the robotic arm less chaotic. The function that stopped working after the update was the relax function, where the motors of the robotic arm would become 'relaxed'. In this state the motors would have zero torque, meaning that the robotic arm could be moved freely. A problem that will continue for the robotic arm is the overheating of the different motors, this can be solved by using stronger motors that can handle more torque. Finally for the production line to be really reliable and functional for multiple iterations, two things have to be solved. Firstly the robotic arm has to become even more precise, this could be done by using stronger motors or by having a more accurate control system. Secondly, the positions of the stations have to stay the same, meaning that they should be bolted to a certain position in order not to move. By doing this the positions that are given to the robotic arm could stay the same for each iteration. Instead this was now solved by calibrating the robotic arm each iteration.

As already mentioned were the Festo stations first programmed in LabVIEW. This code was tested and optimized to serve as a desired outcome. As this was already done multiple times and a layout was provided did this not pose big problems. Only the fact that the LabVIEW license expired during the research posed a problem. As after the expiration date, which input and output correlated to which sensor or actuator could not be seen anymore. This was solved by taking screenshots of which input and output correlated to which sensor and actuator the day before the expiration date. These screenshots made sure that in MATLAB was easy to name each input and output after the sensor or actuator that is correlated to. When developing the homing process for the extracting station (the homing process has to be done after powering up the station), some new insights were obtained. As at the beginning of this research the homing setting was not clearly described, and code already existed in LabVIEW to run this process. This means that for some time it was unclear how to set the right homing setting for the extracting station. When not given the exact right commands for the homing process the station would shut down after a couple of seconds. This was solved by experimenting with different coordinates and also with different orders of giving the commands. As already mentioned there was trouble with downloading the necessary add-on for MATLAB, this meant that it took a long time before the entire production line could be tested via MATLAB. The reason that the MATLAB add-on could not be downloaded, is that the computers at the DTPA lab did not have the rights to download an add-on. This was solved by using a laptop that did not run on the server of the University of Groningen. For further research, there are multiple opportunities. First of all, as LabVIEW was originally chosen as an application to program the Festo stations because of its graphical interface, which ads educational value. It could be possible to look for a graphical interface within MATLAB. As MATLAB supports another graphical interface, SIMULINK. A next research could determine if this is a valid replacement of LabVIEW. If this is the case, it would be possible for students to still learn to program in a graphical interface, and also use the robotic arm in-between stations. Another field of research that would add value is trying to get a better understanding of how to improve the accuracy of the robotic arm. As every time an error occurred during the production line, the robotic arm's lack of accuracy was the cause. This could be either be at picking up the cup (not holding it firmly), or at releasing the cup (not releasing precisely above the conveyor belt).

7 Conclusion

Throughout this study different results relevant to mechanical aspects and the programming of the three robots have been discussed. These results have given solutions to the problems that have been observed at the beginning of this research. Firstly it has been found that indeed by using a mathematical based application more data can be obtained and used for the extracting station. In MATLAB it was found to be easier to let the extracting station decide which cup to extract. Also just as in LabVIEW while functions are an effective way to stop the process at a certain point and to wait until a sensor gives the desired input, either true or false. Meaning that MATLAB is a good alternative to program the Festo stations. This is even more the case since for students the same educational value can be obtained. In MATLAB, a student can also make use of case structures, or choose to use functions that represent the cases. This is the same as in LabVIEW, however the student should have a basic environment where from they can start the programming. The expected benefit of the three robots being programmed from the same application have also been found to be true. Where it became clear that the communication between the different robots was easy, as the global variables made in one script were automatically transferred to the other script. This meant that for each robot a separate MATLAB script could be developed. A note here is that when operating multiple Festo stations from the same application, it is important to define the different stations very precise to the application. Otherwise one station might behave as the other station, by listening to their commands. Finally the last problem that is fixed using only MATLAB is that now the robots can depend on each other. The robotic arm can start operating as soon as the extracting station gives a command to the robotic arm that it has done its task. The same applies for the robotic arm and the sorting station. To conclude, this research the deliverable, the code in MATLAB for all three robots, has been delivered to DTPA. In addition to this the knowledge obtained about the working of the Festo station programmed via MATLAB has contributed to the knowledge about these Festo stations. Also the updates for the robotic arm were tested, which in turn gave new insights into the working of the software of the robotic arm.

References

[1] Goldratt, E. M. (1990). Theory of constraints. Crotonon-Hudson: North River

[2] Hansen, R. C. (2001). Overall equipment effectiveness: a powerful production/maintenance tool for increased profits. Industrial Press Inc.

[3] Chaos, D., Chacón, J., Lopez-Orozco, J., & Dormido, S. (2013). Virtual and Remote Robotic Laboratory Using EJS, MATLAB and LabVIEW. Sensors, 13(2), 2595–2612. https://doi.org/10.3390/s130202595

[4] Systems, M., & Training, M. (2018). Learning Systems Modular Systems for Mechatronics Training Automation Training with MPS : Modular Production Systems

[5] Sartika, E. M., Sarjono, T., & Christian, K. (2021). Modular Production System Control Using Supervisory Control Theory Method. Journal of Physics: Conference Series, 1858(1), 012095. https://doi.org/10.1088/1742-6596/1858/1/012095

[6] Ebel, F., Pany, M. (2006). Handbuch Station Lagern

[7] Ebel, F., Pany, M. (2006). Handbuch Station Sortieren

[8] Festo Didactic. (z.d.). Copyright Festo - All Rights Reserved. Consulted on 23 March 2022, from https://www.festo-didactic.com/int-en/?fbid=aW50LmVuLjU1Ny4xNy4xMi4zNDU2

[9] Elliott, C., Vijayakumar, V., Zink, W., & Hansen, R. (2007). National Instruments LabVIEW: A Programming Environment for Laboratory Automation and Measurement. JALA: Journal of the Association for Laboratory Automation, 12(1), 17–24. https://doi.org/10.1016/j.jala.2006.07.012

[10] Techopedia. (2021, 8 juni). Automation. Techopedia.Com. Consulted on 23 March 2022, from https://www.techopedia.com/definition/32099/automation

[11] Sisbot, S. (2011). Execution and evaluation of complex industrial automation and control projects using the systems engineering approach. Systems Engineering, 14(2), 193–207. https://doi.org/10.1002/sys.20171

[12] Jun Liu, Khiang Wee Lim, Weng Khuen Ho, Kay Chen Tan, Arthur Tay, & Srinivasan, R. (2005). Using the OPC Standard for Real-Time Process Monitoring and Control. IEEE Software, 22(6), 54–59. https://doi.org/10.1109/ms.2005.168

[13] John, K. H., & Tiegelkamp, M. (2014). IEC 61131–3: Programming Industrial Automation Systems: Concepts and Programming Languages, Requirements for Programming Systems, Decision-Making Aids (2nd ed. 2010 ed.). Springer.

[14] Sartika, E. M., Sarjono, T. R., & Saputra, D. D. (2019). Prediction of PID control model on PLC. TELKOMNIKA (Telecommunication Computing Electronics and Control), 17(1), 529. https://doi.org/10.12928/telkomnika.v17i1.11589

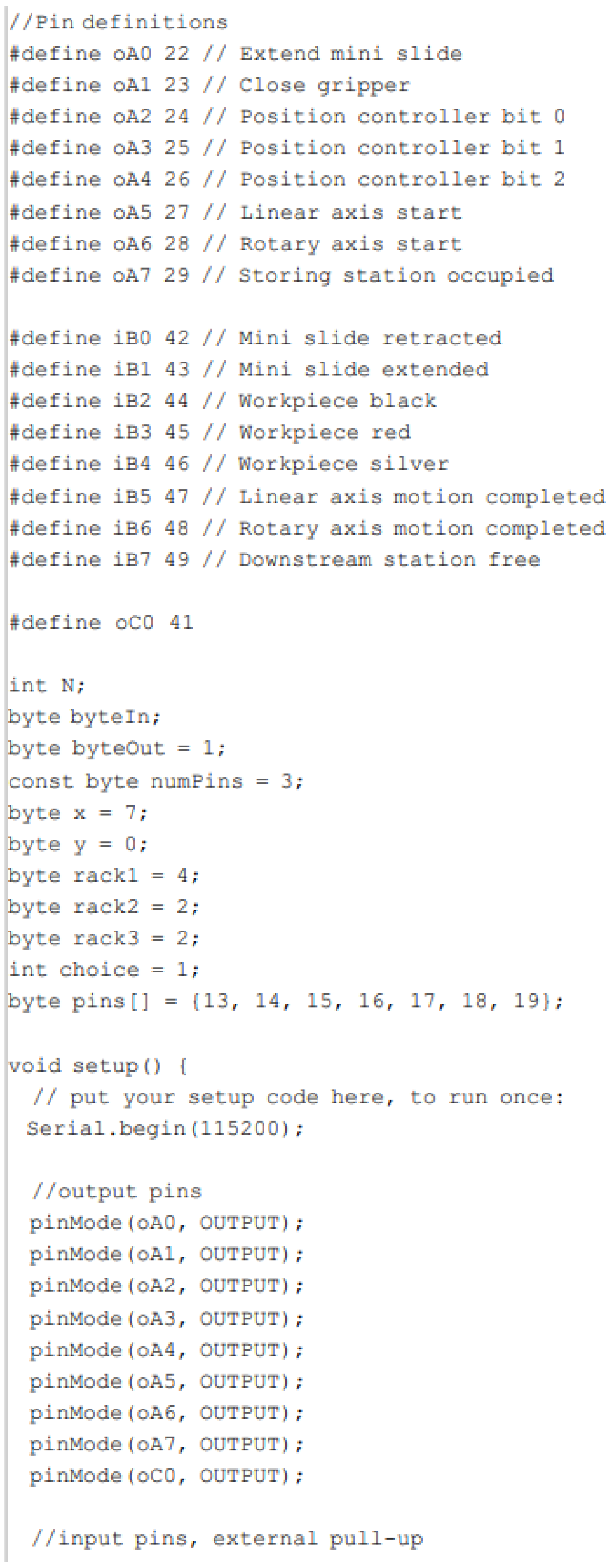
[15] Festo. (2021, February). Factory Automation Learning solutions for basic and advanced training. `https://www.festo.com/net/SupportPortal/Files/468161/PG-FA_en_2021-02_56826_Screen.pdf`

[16] Bai, Jianghua & La Rosa, Andres. (2017). Essentials of Building Virtual Instruments with LabVIEW and Arduino for Lab Automation Applications. International Journal of Science and Research (IJSR). 604. 10.21275/ART20173325.

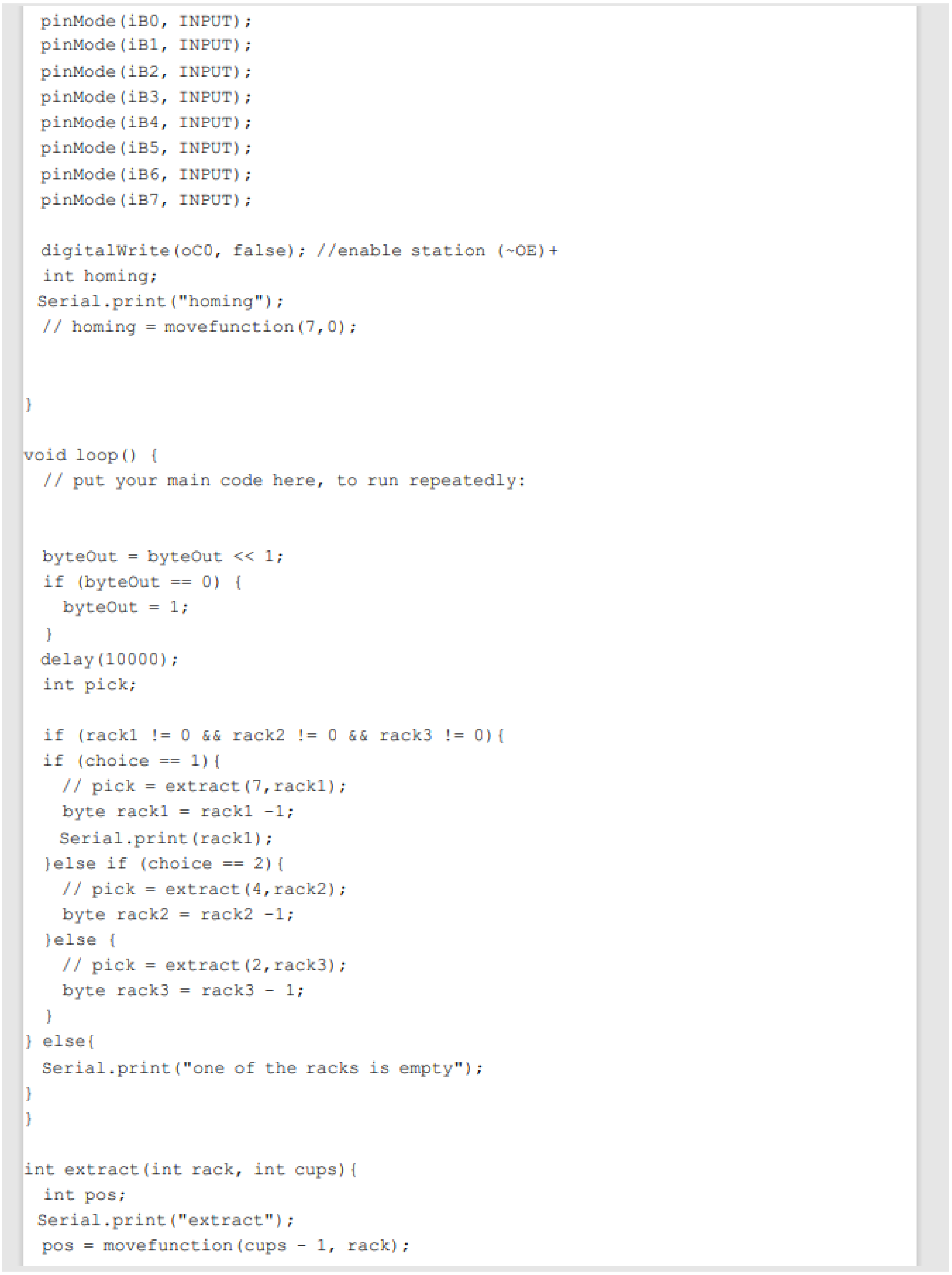
[17] MATLAB - MathWorks. (z.d.). MATLAB & Simulink. Geraadpleegd op 9 mei 2022, van https://nl.mathworks.com/products/matlab.html

8 Appendix

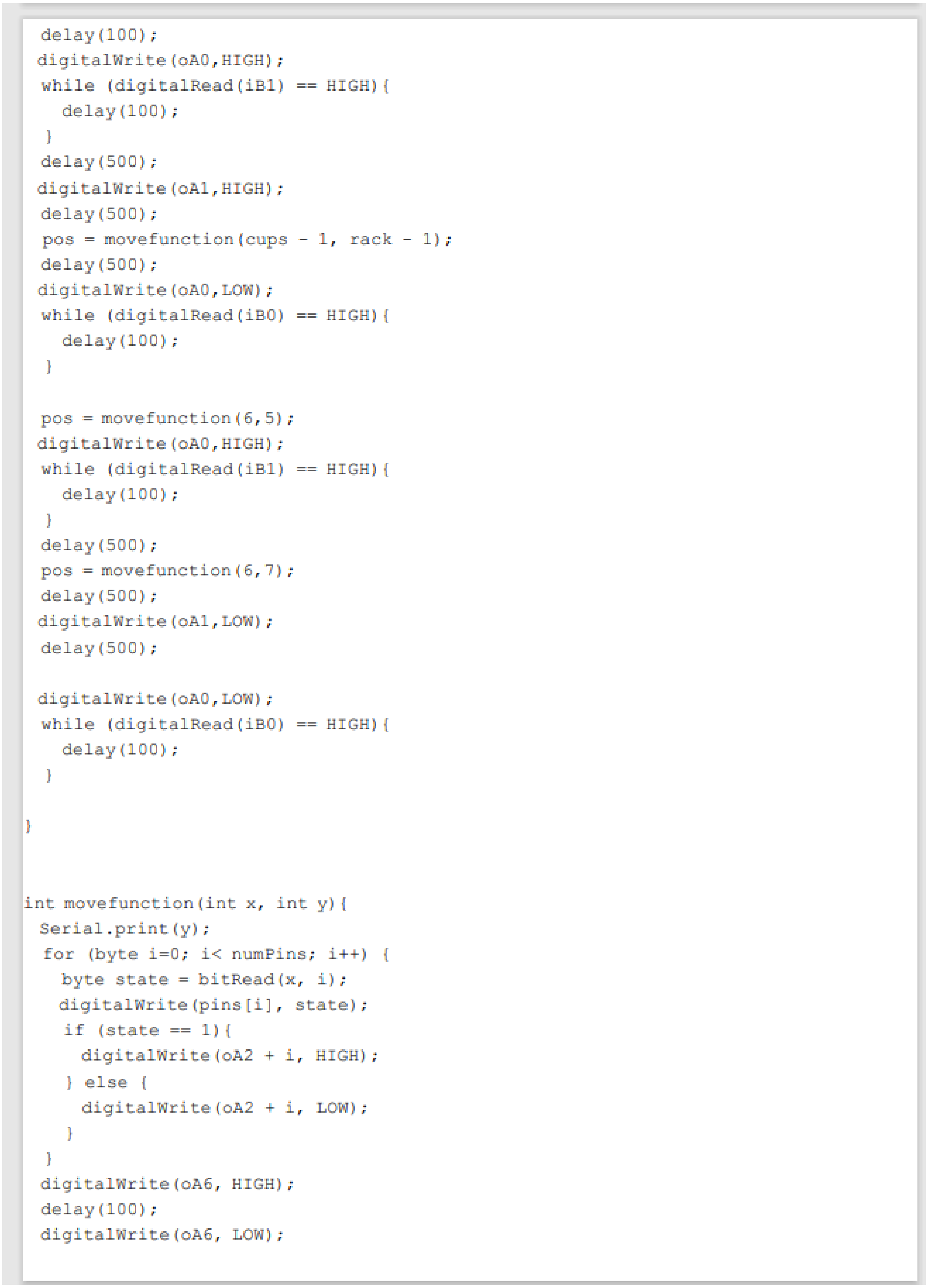
8.1 Appendix A. Arduino test code



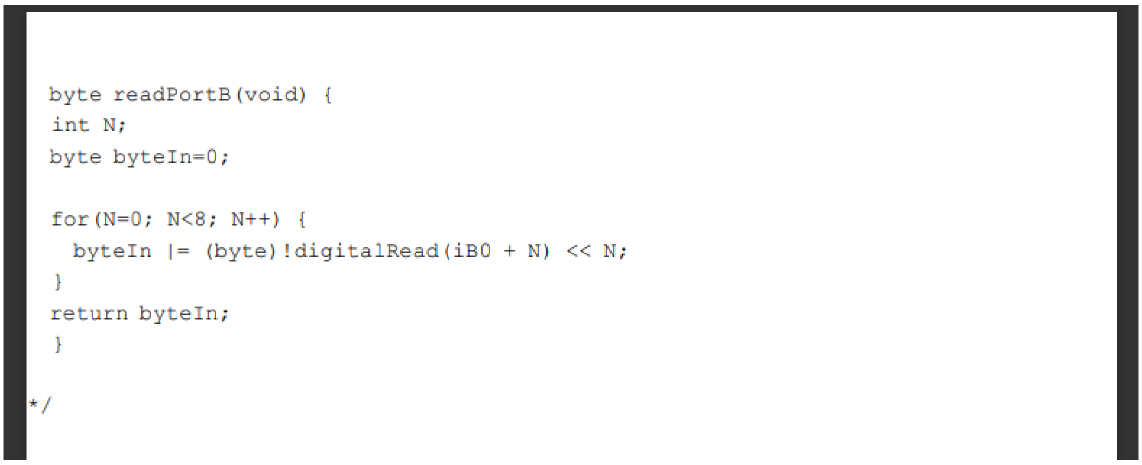
```
//Pin definitions
#define oA0 22 // Extend mini slide
#define oA1 23 // Close gripper
#define oA2 24 // Position controller bit 0
#define oA3 25 // Position controller bit 1
#define oA4 26 // Position controller bit 2
#define oA5 27 // Linear axis start
#define oA6 28 // Rotary axis start
#define oA7 29 // Storing station occupied

#define iB0 42 // Mini slide retracted
#define iB1 43 // Mini slide extended
#define iB2 44 // Workpiece black
#define iB3 45 // Workpiece red
#define iB4 46 // Workpiece silver
#define iB5 47 // Linear axis motion completed
#define iB6 48 // Rotary axis motion completed
#define iB7 49 // Downstream station free

#define oC0 41

int N;
byte byteIn;
byte byteOut = 1;
const byte numPins = 3;
byte x = 7;
byte y = 0;
byte rack1 = 4;
byte rack2 = 2;
byte rack3 = 2;
int choice = 1;
byte pins[] = {13, 14, 15, 16, 17, 18, 19};

void setup() {
  // put your setup code here, to run once:
 Serial.begin(115200);

  //output pins
  pinMode(oA0, OUTPUT);
  pinMode(oA1, OUTPUT);
  pinMode(oA2, OUTPUT);
  pinMode(oA3, OUTPUT);
  pinMode(oA4, OUTPUT);
  pinMode(oA5, OUTPUT);
  pinMode(oA6, OUTPUT);
  pinMode(oA7, OUTPUT);
  pinMode(oC0, OUTPUT);

  //input pins, external pull-up
```



```
  pinMode(iB0, INPUT);
  pinMode(iB1, INPUT);
  pinMode(iB2, INPUT);
  pinMode(iB3, INPUT);
  pinMode(iB4, INPUT);
  pinMode(iB5, INPUT);
  pinMode(iB6, INPUT);
  pinMode(iB7, INPUT);

  digitalWrite(oC0, false); //enable station (~OE)+
  int homing;
 Serial.print("homing");
  // homing = movefunction(7,0);


}

void loop() {
  // put your main code here, to run repeatedly:


  byteOut = byteOut << 1;
  if (byteOut == 0) {
    byteOut = 1;
  }
  delay(10000);
  int pick;

  if (rack1 != 0 && rack2 != 0 && rack3 != 0){
  if (choice == 1){
    // pick = extract(7,rack1);
    byte rack1 = rack1 -1;
    Serial.print(rack1);
  }else if (choice == 2){
    // pick = extract(4,rack2);
    byte rack2 = rack2 -1;
  }else {
    // pick = extract(2,rack3);
    byte rack3 = rack3 - 1;
  }
} else{
  Serial.print("one of the racks is empty");
}
}

int extract(int rack, int cups){
  int pos;
 Serial.print("extract");
  pos = movefunction(cups - 1, rack);
```

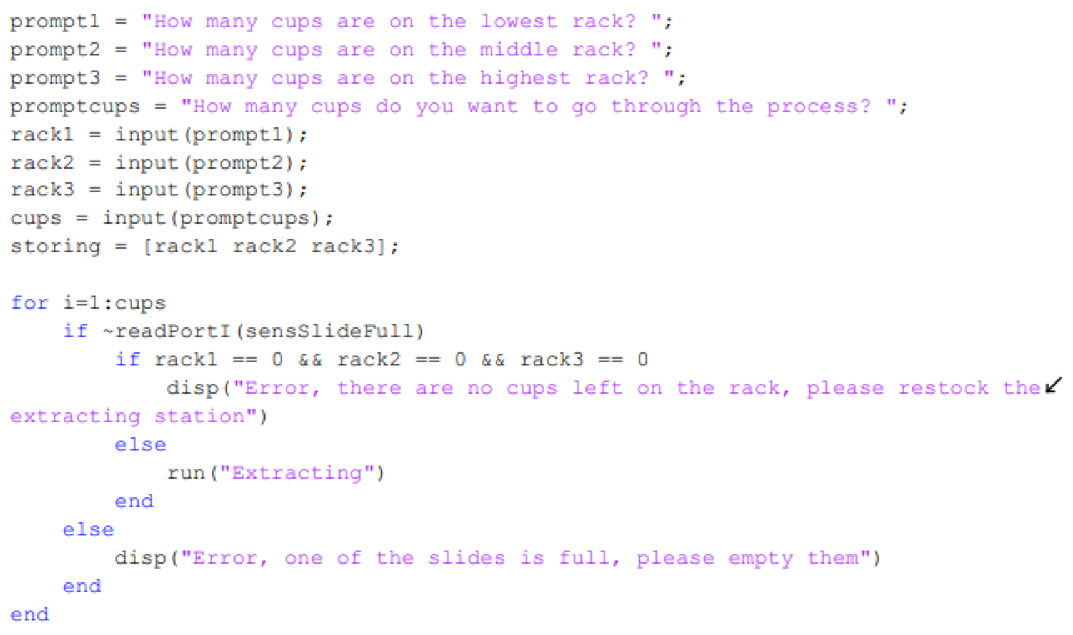


```
  delay(100);
  digitalWrite(oA0,HIGH);
  while (digitalRead(iB1) == HIGH){
    delay(100);
  }
  delay(500);
  digitalWrite(oA1,HIGH);
  delay(500);
  pos = movefunction(cups - 1, rack - 1);
  delay(500);
  digitalWrite(oA0,LOW);
  while (digitalRead(iB0) == HIGH){
    delay(100);
  }

  pos = movefunction(6,5);
  digitalWrite(oA0,HIGH);
  while (digitalRead(iB1) == HIGH){
    delay(100);
  }
  delay(500);
  pos = movefunction(6,7);
  delay(500);
  digitalWrite(oA1,LOW);
  delay(500);

  digitalWrite(oA0,LOW);
  while (digitalRead(iB0) == HIGH){
    delay(100);
  }

}


int movefunction(int x, int y){
  Serial.print(y);
  for (byte i=0; i< numPins; i++) {
    byte state = bitRead(x, i);
    digitalWrite(pins[i], state);
    if (state == 1){
      digitalWrite(oA2 + i, HIGH);
    } else {
      digitalWrite(oA2 + i, LOW);
    }
  }
  digitalWrite(oA6, HIGH);
  delay(100);
  digitalWrite(oA6, LOW);
  for (byte i=0; i< numPins; i++) {
    byte state = bitRead(y, i);
    digitalWrite(pins[i], state);
    if (state == 1){
      digitalWrite(oA2 + i, HIGH);
    } else {
      digitalWrite(oA2 + i, LOW);
    }
    Serial.print(state);
  }
  digitalWrite(oA5, HIGH);
  delay(100);
  digitalWrite(oA5, LOW);
  delay(100);
  while (digitalRead(iB5) == HIGH){
    delay(100);
    Serial.print("Linear axis busy");
  }
  while (digitalRead(iB6) == HIGH){
    delay(100);
    Serial.print("Rotary axis busy");
  }
  Serial.print("Done moving");
}


/* To do: create functions to write/read all in- and outputs at once.

  void enMPA(int enable) {
  if (enable != 0) {
    digitalWrite(oC0, false);
  }
  else {
    digitalWrite(oC0, true);
  }
  }




  void writePortA(byte byteOut) {
  int N;

  for(N=0; N<8; N++) {
    digitalWrite( oA0+N, byteOut&(1<<N) );
  }
  }
```



```
byte readPortB(void) {
int N;
byte byteIn=0;

for(N=0; N<8; N++) {
  byteIn |= (byte)!digitalRead(iB0 + N) << N;
}
return byteIn;
}

*/
```

8.2 Appendix B. waiting.m



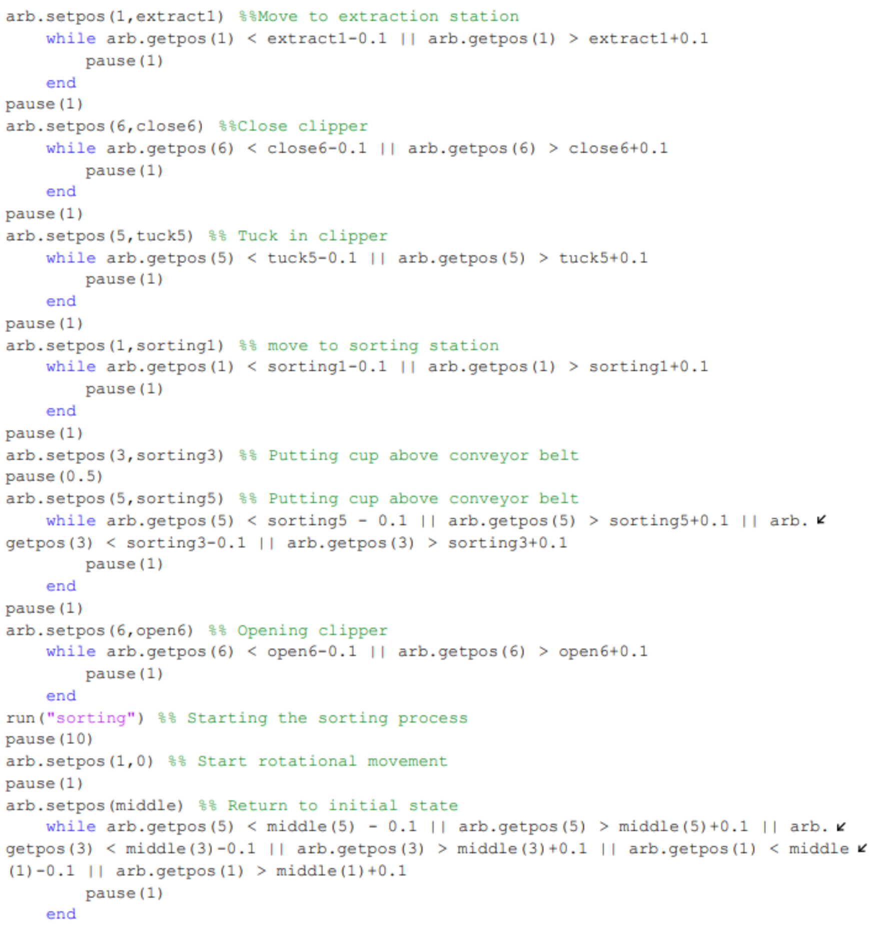
```
prompt1 = "How many cups are on the lowest rack? ";
prompt2 = "How many cups are on the middle rack? ";
prompt3 = "How many cups are on the highest rack? ";
promptcups = "How many cups do you want to go through the process? ";
rack1 = input(prompt1);
rack2 = input(prompt2);
rack3 = input(prompt3);
cups = input(promptcups);
storing = [rack1 rack2 rack3];

for i=1:cups
    if ~readPortI(sensSlideFull)
        if rack1 == 0 && rack2 == 0 && rack3 == 0
            disp("Error, there are no cups left on the rack, please restock the↙
extracting station")
        else
            run("Extracting")
        end
    else
        disp("Error, one of the slides is full, please empty them")
    end
end
```

Figure 21: Waiting.m

8.3 Appendix C. robotic_arm.m

```
close6 = 0.720970970306251
extract1 = 1.119805975156518
extract3 = 0.0972
extract5 = -1.447055209905455
middle =  [    0.0102   -1.14    0.0972   -0.0205   -1.447055209905455    0.2]
open6 =  0.200000000000000
sorting1 =  -1.636246173744684
sorting3 = -0.010000000000000
sorting5 =  -0.470420774951597
test =  [-0.005113269292952,-1.176051937378992,0.107378655151995,-0.112491924444947,↙
-1.487961364249072,-0.010226538585904]
tuck = -1.774304444654392
tuck5 =  -1.774304444654392
zero = [0,0,0,0,0,0]
arb.writedata2(5, Arbotix.ADDR_SPEED, 150)
arb.writedata2(6, Arbotix.ADDR_SPEED, 150)
arb.writedata2(3, Arbotix.ADDR_SPEED, 10)
arb.writedata2(1, Arbotix.ADDR_SPEED, 50)
```

Figure 22: Setting the variable for the robotic arm



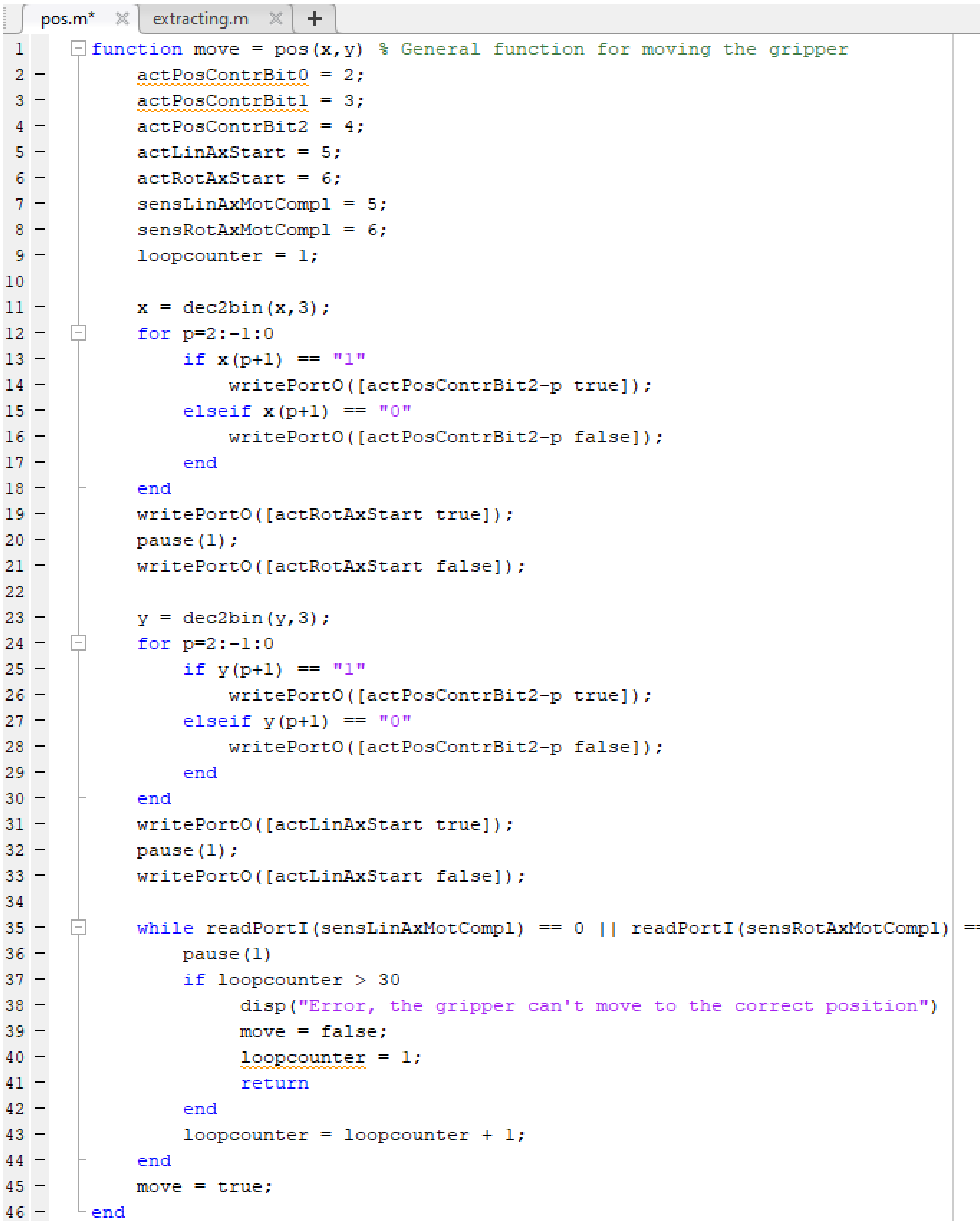
```
arb.setpos(1,extract1) %%Move to extraction station
    while arb.getpos(1) < extract1-0.1 || arb.getpos(1) > extract1+0.1
        pause(1)
    end
pause(1)
arb.setpos(6,close6) %%Close clipper
    while arb.getpos(6) < close6-0.1 || arb.getpos(6) > close6+0.1
        pause(1)
    end
pause(1)
arb.setpos(5,tuck5) %% Tuck in clipper
    while arb.getpos(5) < tuck5-0.1 || arb.getpos(5) > tuck5+0.1
        pause(1)
    end
pause(1)
arb.setpos(1,sorting1) %% move to sorting station
    while arb.getpos(1) < sorting1-0.1 || arb.getpos(1) > sorting1+0.1
        pause(1)
    end
pause(1)
arb.setpos(3,sorting3) %% Putting cup above conveyor belt
pause(0.5)
arb.setpos(5,sorting5) %% Putting cup above conveyor belt
    while arb.getpos(5) < sorting5 - 0.1 || arb.getpos(5) > sorting5+0.1 || arb.↙
getpos(3) < sorting3-0.1 || arb.getpos(3) > sorting3+0.1
        pause(1)
    end
pause(1)
arb.setpos(6,open6) %% Opening clipper
    while arb.getpos(6) < open6-0.1 || arb.getpos(6) > open6+0.1
        pause(1)
    end
run("sorting") %% Starting the sorting process
pause(10)
arb.setpos(1,0) %% Start rotational movement
pause(1)
arb.setpos(middle) %% Return to initial state
    while arb.getpos(5) < middle(5) - 0.1 || arb.getpos(5) > middle(5)+0.1 || arb.↙
getpos(3) < middle(3)-0.1 || arb.getpos(3) > middle(3)+0.1 || arb.getpos(1) < middle↙
(1)-0.1 || arb.getpos(1) > middle(1)+0.1
        pause(1)
    end
```

Figure 23: MATLAB code for the robotic arm

8.4 Appendix D. extracting.m

```
% Declare vars here to show up in the workspace (example for separating station).

% Sensors | bit number
sensMinSlidRetr = 0;
sensMinSlidExt = 1;
sensWpcBlack = 2;
sensWpcRed = 3;
sensWpcSilver = 4;
sensLinAxMotCompl = 5;
sensRotAxMotCompl = 6;
sensDownStatFree = 7;

% Actuators | bit number
actExtMinSlid = 0;
actCloseGripper = 1;
actPosContrBit0 = 2;
actPosContrBit1 = 3;
actPosContrBit2 = 4;
actLinAxStart = 5;
actRotAxStart = 6;
actStationOccupied = 7;

% Program states, numbers are arbitrary
START = 1;
RACK = 2;
PICK_CUP = 3;
EXTRACT = 4;
INIT = 5;

% Local variables
state = START;   % initial state
stop = false;   % stop flag


% INIT, clear outputs and enable the station
writePortO([actExtMinSlid false]);
writePortO([actCloseGripper false]);
writePortO([actLinAxStart false]);
writePortO([actRotAxStart false]);
writePortO([actStationOccupied false]);
enableFesto(true);

% Run the homing process if this is the first iteration
if i == 1
    writePortO([actExtMinSlid false]);
    writePortO([actCloseGripper false]);
    writePortO([actLinAxStart false]);
    writePortO([actRotAxStart false]);
    writePortO([actStationOccupied false]);
    writePortO([actPosContrBit0 false]);
    writePortO([actPosContrBit1 false]);
    writePortO([actPosContrBit2 false]);
    move = pos(7,0);
    if move == true
        disp("Homing movement is done")
    else
        disp("homing movement is NOT done")
    end
end

% PROGRAM LOOP, loop until stop = true
while ~stop

    switch state
        case START
            % Default all actuators off.
            [M,R] = max(storing);
            if R == 3
                y = 7;
            elseif R == 2
                y = 4;
            elseif R == 1
                y = 4;
            end
            state = RACK;
        case RACK
            loopcounter = 1;
            while ~readPortI(sensDownStatFree)
                pause(0.1);
                if loopcounter > 30
                    disp("Error, the down stream station is not free")
                    loopcounter = 1;
                end
                loopcounter = loopcounter + 1;
            end
            move = pos(M-1,y);
            if move == true
                state = PICK_CUP;
            else
                stop = true;
            end
        case PICK_CUP
            writePortO([actExtMinSlid true]);
            while ~readPortI(sensMinSlidExt)
                pause(0.1)
                if loopcounter > 50
                    disp("Error, the mini slide can't extract")
                    loopcounter = 1;
                end
                loopcounter = loopcounter + 1;
            end
                writePortO([actCloseGripper true]);
                pause(0.5)
                state = EXTRACT;

        case EXTRACT
            move = pos(M-1,y-1);
            if move == true
                pause(0.1)
                writePortO([actExtMinSlid false]);
```

```
                  while ~readPortI(sensMinSlidRetr)
                  pause(0.1)
                  if loopcounter > 50
                      disp("Error, the mini slide can't retract")
                      loopcounter = 1;
                  end
                  loopcounter = loopcounter + 1;
                  end
                  move = pos(6,5);
                  if move == true
                      writePortO([actExtMinSlid true]);
                      while ~readPortI(sensMinSlidExt)
                          pause(0.1)
                          if loopcounter > 50
                              disp("Error, the mini slide can't extract")
                              loopcounter = 1;
                          end
                          loopcounter = loopcounter + 1;
                      end
                      move = pos(6,7);
                          if move == true
                              pause(0.1)
                              writePortO([actCloseGripper false])
                              state = INIT;
                          end
                  end
              end
          case INIT
              storing(R) = M-1;
              writePortO([actExtMinSlid false]);
              writePortO([actCloseGripper false]);
              writePortO([actLinAxStart false]);
              writePortO([actRotAxStart false]);
              writePortO([actStationOccupied false]);
              while ~readPortI(sensMinSlidRetr)
                  pause(0.1)
                  if loopcounter > 50
                      disp("Error, the mini slide can't retract")
                      loopcounter = 1;
                  end
                  loopcounter = loopcounter + 1;
               end
          otherwise
              stop = true;
      end
  end

  % STOP, disable the station (unpower)
  enableFesto(false);



run("robotic_arm")
```



```
function move = pos(x,y) % General function for moving the gripper
    actPosContrBit0 = 2;
    actPosContrBit1 = 3;
    actPosContrBit2 = 4;
    actLinAxStart = 5;
    actRotAxStart = 6;
    sensLinAxMotCompl = 5;
    sensRotAxMotCompl = 6;
    loopcounter = 1;

    x = dec2bin(x,3);
    for p=2:-1:0
        if x(p+1) == "1"
            writePortO([actPosContrBit2-p true]);
        elseif x(p+1) == "0"
            writePortO([actPosContrBit2-p false]);
        end
    end
    writePortO([actRotAxStart true]);
    pause(1);
    writePortO([actRotAxStart false]);

    y = dec2bin(y,3);
    for p=2:-1:0
        if y(p+1) == "1"
            writePortO([actPosContrBit2-p true]);
        elseif y(p+1) == "0"
            writePortO([actPosContrBit2-p false]);
        end
    end
    writePortO([actLinAxStart true]);
    pause(1);
    writePortO([actLinAxStart false]);

    while readPortI(sensLinAxMotCompl) == 0 || readPortI(sensRotAxMotCompl) ==
        pause(1)
        if loopcounter > 30
             disp("Error, the gripper can't move to the correct position")
             move = false;
             loopcounter = 1;
             return
        end
        loopcounter = loopcounter + 1;
    end
    move = true;
end
```

8.5 Appendix E. sorting.m

```
% Declare vars here to show up in the workspace (example for separating station).

% Sensors | bit number
sensWpcAvailable = 0;
sensWpcMetal = 1;
sensWpcNotBlack = 2;
sensSlideFull = 3;
sensSwitch1Retr = 4;
sensSwitch1Ext = 5;
sensSwitch2Retr = 6;
sensSwitch2Ext = 7;

% Actuators | bit number
actConveyerMotor = 0;
actExtSwitch1 = 1;
actExtSwitch2 = 2;
actExtStopper = 3;
actStationOccupied = 7;

% Program states, numbers are arbitrary
START = 1;
COLOURID = 2;
METAL = 3;
RED = 4;
BLACK = 5;
INIT = 6;

% Local variables
state = START;   % initial state
stop = false;   % stop flag



% INIT, clear outputs and enable the station
writePortO([actConveyerMotor false]);
writePortO([actExtSwitch1 false]);
writePortO([actExtSwitch2 false]);
writePortO([actExtStopper true]);
writePortO([actStationOccupied false]);
enableFesto(true);


% PROGRAM LOOP, loop until stop = true
while ~stop
    loopcounter = 1;
    switch state
        case START
            % Default all actuators off.
            state = COLOURID;

        case COLOURID
            if readPortI(sensWpcMetal)
                state = METAL;
            elseif readPortI(sensWpcNotBlack)
```

```
                state = RED;
            else
                state = BLACK;
            end

        case METAL
            writePortO([actExtSwitch1 true]);
            while ~readPortI(sensSwitch1Ext)
                pause(0.1)
                if loopcounter > 30
                    disp("Error, switch 1 won't extend")
                    loopcounter = 1;
                end
                loopcounter = loopcounter + 1;
            end
            writePortO([actExtStopper false]);
            pause(5)
            state = INIT;

         case RED
            writePortO([actExtSwitch2 true]);
            while ~readPortI(sensSwitch2Ext)
                pause(0.1)
                if loopcounter > 30
                    disp("Error, switch 2 won't extend")
                    loopcounter = 1;
                end
                loopcounter = loopcounter + 1;
            end
            writePortO([actExtStopper false]);
            pause(5)
            state = INIT;

        case BLACK
            writePortO([actExtStopper false]);
            pause(5)
            state = INIT;

        case INIT
            writePortO([actConveyerMotor false]);
            writePortO([actExtSwitch1 false]);
            writePortO([actExtSwitch2 false]);
            writePortO([actExtStopper true]);
            writePortO([actStationOccupied false])
            while ~readPortI(sensSwitch2Retr) && ~readPortI(sensSwitch1Retr)
                pause(0.1)
                if loopcounter > 30
                    disp("Error, the station won't return to its initial state")
                    loopcounter = 1;
                end
                loopcounter = loopcounter + 1;
            end

        otherwise
            stop = true;


    end

end

run("waiting")

% STOP, disable the station (unpower)
enableFesto1(false);
```