March 31, 2015

BACHELOR ASSIGNMENT

PROTOTYPE FOR NON-COLLINEAR WAVE MIXING: SOFTWARE IMPLEMENTATION

J.J.J.W. Dekker

Faculty of Mechanical Engineering,
TM

Exam committee:
A. de Boer, R. Loendersloot, J.M. Jauregui Becker

Documentnumber
TM — 5751

UNIVERSITY OF TWENTE.

Contents

Chapter 1

Introduction

The present work is a collaboration project that was carried out between the University Twente and WETSUS. The objective of the project called Ultrasonic Inspection of Water Distribution Mains was to develop techniques that could be used for the inline inspection of water distribution mains in the Netherlands. The water distribution mains in the Netherlands usually are made of PVC, Asbestos Cement or Cast Iron [1].

The project started with finding an effective technique for detecting material deterioration in the materials used for water distribution mains in laboratory controlled conditions. Research done by Demčenko [2] proved NCWM to be suitable for the identification of physical ageing in PVC. With that technique in mind a prototype has been designed and partially completed by Mainini [3] for testing in laboratory controlled conditions. This prototype itself will not leave the laboratory, but the concept will be used for making a inspection device which will be used in water mains. The prototype itself should also be able to do measurement techniques like diffuse field and pulse-echo however these techniques will not be discussed in detail since NCWM technique is more complex.

Mainini has finished the holding and moving system for the transducers and the casing around it. Furthermore most of the control system was finished. The control system can be split up in a hardware and a software part. For the hardware 1 motor + encoder, 1 motor driver, 12 photo-resistors, an Arduino and a breakout board to connect all the hardware with each other were prepared. A more detailed explanation about the hardware will be given in the chapter 4.

The Arduino is a single-board microcontroller, so a script was programmed to make it able to control things like the motor drivers. Arduino scripting is the same as C++ scripting apart from some simplification which makes it easier to use for people with little programming knowledge. It also has some limitations because of tight memory constraints. The bare Arduino environment itself does not have much functionality. That can be solved by adding libraries in the form of C++ header files. Mainini created and found a couple of libraries which made the Arduino have extra functionality in working with the hardware of the prototype and handling input data. The last component to finish the software part is the user interface which was made in LabVIEW. Through the LabVIEW UI commands can be given to the Arduino which gives a corresponding action command to the other hardware components. The Arduino also returns human readable feedback to the LabVIEW UI. A more detailed explanation of the software will be given in chapter 5.
In figure 1.1 a scheme can been seen on how all elements are connected and communicate with each other for the full system. The solid line blocks are hardware components and the dashed line blocks are software components.

The initial objective of this BSc Assignment was to finalise the prototype assembly and to test if the system works in laboratory controlled conditions. The assignment can be split up in a section prototype assembly, control system and testing. According to Mainini [3] the following things still needed to be done:

- Prototype assembly
  - The installation of 5 of the 6 motors encoders and motor drivers (1 of each has already been installed).

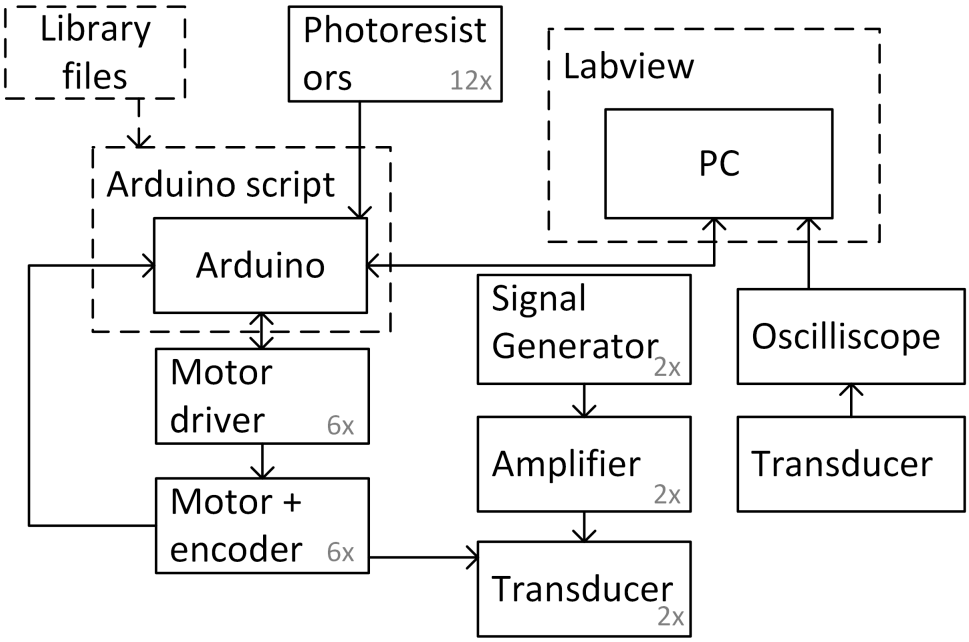


Figure 1.1: The hardware scheme for the system with an addition of the software scheme. The blocks with dashed lines are software components and the blocks with solid lines are the hardware components.

  - The installation of the 3 ultrasonic transmitters.
  - Make the assembly watertight.
- Control system
  - Finalize parts of the Arduino script.
  - Recheck if LabVIEW is truly and fully coherent with the Arduino software.
  - Check if the safety rules are well implemented.
- Testing
  - Test if the assembly works with more than one motor and driver attached.

During the first weeks of orientation it was soon discovered that the control system part lacked useful documentation to fully continue with the testing part. Furthermore the breakout board was seemingly not completed and the way it worked was also unclear. Because of that the main focus of the assignment shifted to fixing and completing to control system part so it would be possible to test the prototype after that. Also proper documentation on this part was necessary for further developments of the system.

The reformulated objective of this assignment is to deliver an operational control system, including proper documentation of the connection and functions of the system in terms of software and hardware. To be able to do that it is first required to get some general knowledge of the Non-Collinear Wave Mixing technique which will be discussed in chapter 2. From the general concept of the NCWM technique the requirements for the prototype concept can be made. Which can be read in chapter 3. After that the control system of the assembly will be discussed in chapter 4 and 5. Finally chapter 6 will contain some tests to check if the control system is working and the report is then finished off with a conclusion and some recommendations for future work.

Chapter 2

Non-Collinear Wave Mixing

Rather than explaining the detailed of the NCWM theory, only the general concept of the NCWM theory will be explained here. More information about the NCWM theory can be found in the reports of Mainini [3] and of Demčenko [2].

Materials generally exhibit non-linear behaviour. Usually these non-linearities tend to be small and are ignored in most cases because of practicality. However these non-linearities are more sensitive to subtle changes in the material than the linear material parameters which makes them very useful for inspecting material degradation or small damages like micro-cracks. A non destructive way to study these non-linearities is by using ultrasonic techniques. The ultrasonics technique can be spilt up in linear ultrasonics and nonlinear ultrasonics. Nonlinear ultrasonics is preferable to usual linear ultrasonic techniques since it is much better at detecting non-linearities in the material [2]. For the nonlinear ultrasonics there are several techniques of which NCWM is also preferable compared to some non-linear ultrasonics like harmonic generation since the interpretation of the measurement are less complex.

Without going too much into the theoretical details of NCWM the techniques working can be seen in figure 2.1. The technique of NCWM works with the principle that two transducers generate linear

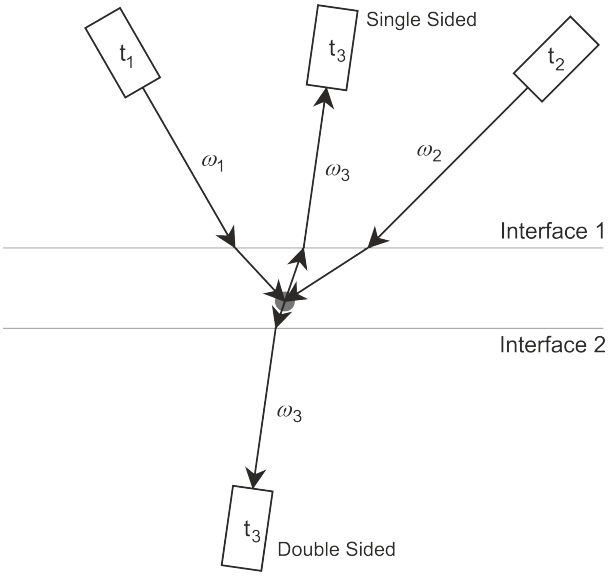


Figure 2.1: The concept of Non-Collinear Wave Mixing. Two generated waves generate under the right conditions a third wave. The area in where they interact is called the interaction volume. Figure taken from Mainini [3].

ultrasonic beam which will interact in a non-linear way in the object under inspection to form a new

wave. From the active volume of the material the new wave will propagate in two directions, one in forward direction and the other one in opposite direction. The wave which propagates forward through the material has a bigger intensity than the one that is going in the opposite direction however in water mains only the single-sided configuration is an option. NCWM is not the most easy technique since for nonlinear interaction of elastic waves a lot of conditions need to be satisfied which means that certain types of interactions cannot exist, “only 10 out of the 54 potential interactions are possible” [2, p.13]. One of the possible interactions to create a new wave is a shear wave and a longitudinal wave: $SV(\omega_1) + L(\omega_2) \rightarrow L(\omega_1 + \omega_2)$. The frequency of the output wave is the sum or the difference of the frequency of the parent waves, which depends on the interaction angle and frequency ratio of the parent waves. Even without knowing the details of the underlying theory it is pretty clear from figure 2.1 that the angle and the position of the transducers with respect to each other and the material is very important for the creation and the reflection of the non-collinear wave. However since the conditions for creation of the new wave are so precise it is relatively easy to extract the resonant scattered waves from the total wave field by narrow-band filtering because you know what kind of waves would be created because of the conditions. Furthermore non-linearities or deviations in the signal from the emitting transducers are not in the new wave, since the do not fulfil the conditions for the generation of the new wave and are thus not contributing to the newly generated wave.

From this chapter it has become clear how important the angle and the position of the transducers with respect to each other and the material for the NCWM method is. The need for that precision has been implemented in the design of the prototype and will be discussed in the next chapter.

Chapter 3

Prototype Assembly

From the previous chapter it has become clear that the angle and the position of the transducers with respect to each other and the material is very important for the NCWM method. The design of the prototype has been highly driven by those precision requirements and the requirement to operate in single-sided configuration. The exact design details will not be discussed in here again as they can be read in the report by Mainini [3]. Mainini took care of the movement issue and made a beginning with making the assembly watertight however he did not have time to finish that last part. Both parts will be discussed below to give a quick view on assembly status and the things that still need to be done.

3.1 Movement

To be able to comply with the high accuracy requirement of the angle and the precision a spindle has been chosen with a small pitch on which the transducers carriages can move. The transducer itself is held by a transducer holder which is attached to a metal rectangular plate as can be seen in figure 3.1. One side of the plate with a hole is attached to the lower carriage on the lower spindle, the other side of the plate with a slit is attached to upper carriage on the other spindle. The reason for that is to make the rotation point of the transducer as close as possible to tip of the transducer to accurately change the angle while maintaining the same position of the tip.

3.2 Watertight

The prototype needs to be watertight for the testing in a water main simulated laboratory setting. With the prototype that has been prepared the construction exist out of two separate compartments, the transducer compartment and the motor compartment. The motor compartment has three things that need to be made watertight. Firstly the holes in the compartment's lid through which the spindles have to go through to get attached to the motor. Secondly the opening of the compartment box where the lid in figure 3.2 goes on and lastly the hole through which the (motor encoder) wires need to go through. For the holes in the compartment's lid a metal ball-bearing holders have been glued into place to make that connection watertight. As for the hole in those holders a rubber ring has been glued in place as can be seen in figure 3.2. When the spindle goes through the ball-bearing holder hole the plastic under the ball-bearing and the rubber ring will make a seal.

As for the lid opening a rubber band has been prepared as can be seen in figure 3.3. The rubber band goes over the inner "edge" of the lid and makes a water and airtight seal when the lid and the motor compartment box are tightly screwed together.

Finally the wire hole solution was a bit more difficult since the wires would also need to go through and it would be preferred if it would still be possible to de-assembly the whole assembly. After some research a possible solution called a 'watertight wire seals' (WWS) was found (in Dutch 'kabelwartel'). It can also be found with the name 'liquid tight cable glands' or something similar (a standard name could not be found). In figure 3.4 a picture can be seen of how they usually look like. Most WWS work with the principle that they have a rubber which wraps around the wires tightly once the WWS has been turned

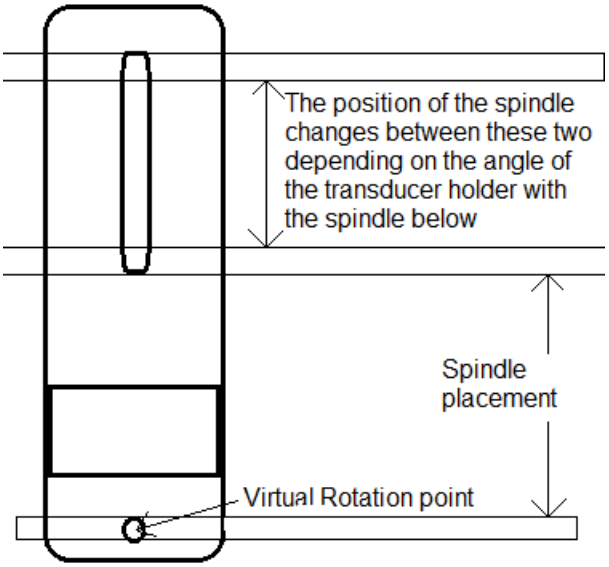


Figure 3.1: The trandsucer holder. If both motors are on the transducer holder will have a translational movement and the spindle position stays the same. If only the motor of the upper spindle is on the holder will move around the virtual rotation point and the spindle position of the upper spindle will change.

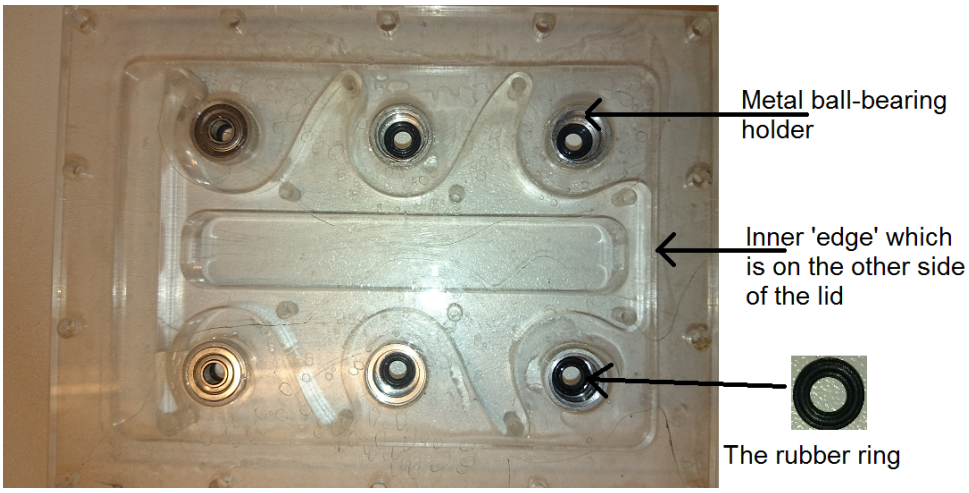


Figure 3.2: The ball-bearing holder with the rubber ring glued into place.

into place. Common WWS on the market have a maximum size of 1 cm in diameter. Unfortunately the current wire hole is 1.95 cm in diameter. The possibilities with the WWS as a watertight solutions could be to find a bigger WWS on specialized websites for wires or WWS. Another option would be to remake the motor compartment so individual holes for WWS can be made for each motor. The last option would also solve the difficulty of having to pack all the 6 encoders wires watertight together.

3.3 Concluding remarks

Since the focus of this assignment was shifted to completing the control system the construction of the prototype has not been finish. This chapter was meant to give an introduction about the construction of the prototype and to give some input for making the assembly watertight to enable quick future

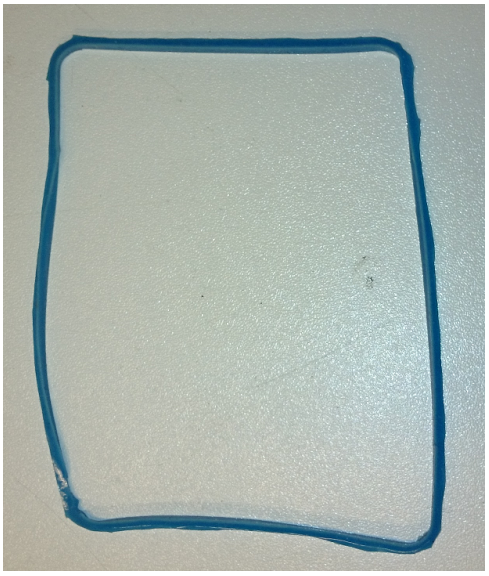


Figure 3.3: Rubber sealing band for the compartment lid.

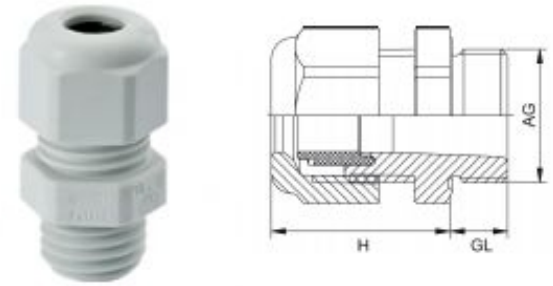


Figure 3.4: Watertight wire seals.

continuation. The only visible change of the assembly status from the starting point of this assignment is that all 6 motors and encoders are now available and some minor adjustments and assembly things have been done. For the transducers the ones already available in the testing laboratory will be used, however for the future it is possible to look into getting traducers with the specifications that Mainini gave in his report [3].

Chapter 4

Hardware

The hardware components of the prototype consist of the motors and the attached encoders, motor drivers, photo resistors which work like switches and the Arduino board. In figure 4.1 a diagram can be seen with arrows indicating in which direction information flows and how the components are connected. The Arduino itself is connected with the PC and communication goes both ways between the PC and the Arduino.

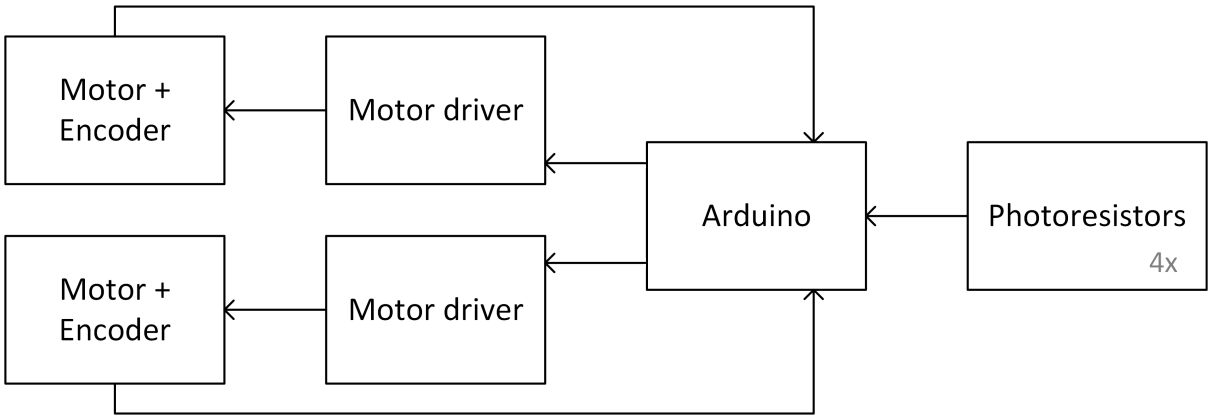


Figure 4.1: A schematic of the hardware connections.

In figure 4.1 the hardware connection for the control of one transducer can be seen. The Arduino gives information to the motor drivers on how to move their motors. The motor drivers are also connected to the motors to tell them to turn. The encoders subsequently tell the Arduino information about the rotation of the motors. The photo resistors are placed on both ends of the rails and give a signal to the Arduino to indicate if the carriages have moved over the photo resistors.

4.1 Switches

The switches/photoresistors on the rails work to prevent the carriages which hold the transducers holders from moving too far and off the rails and they also function as a home reference point. The principle can be seen in figure 4.2.

R_1 and R_3 are normal resistors and R_2 is the variable resistor which resistance decreases with light intensity.
When $R_3 \gg R_1$ is taken, then V_{out} will be almost the same as V_{in} if the photoresistor gets completely covered according to the formula 4.1.

$$V_{out} = V_{in} \cdot \frac{R_2 \cdot R_3}{R_1 \cdot (R_2 + R_3) + R_2 \cdot R_3} \tag{4.1}$$

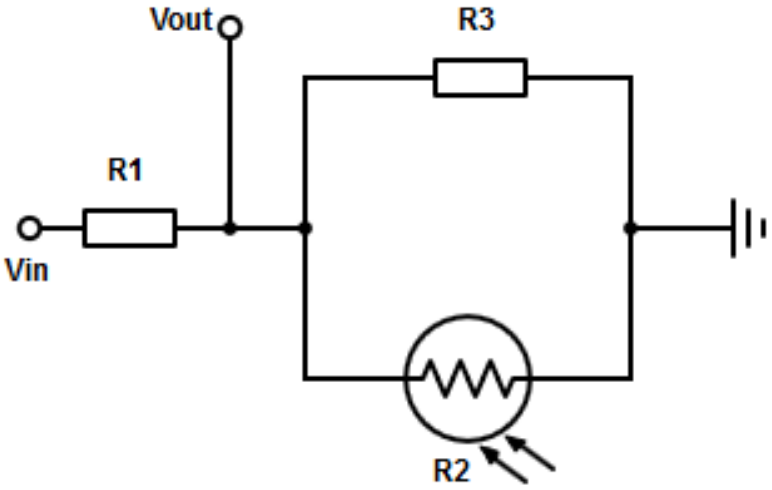


Figure 4.2: Photo resistor circuit.

The resistance of R_2 will increase with the 'blockage' of the light intensity by the carriage which results in V_{out} getting closer to the V_{in} value. With a specified threshold value of V_{out} the 'blockage' of the light intensity by the carriage is detected by the Arduino as a sign that the motor carriage is on the home position. On the breakout board the photo resistor circuit looks like figure 4.3 .

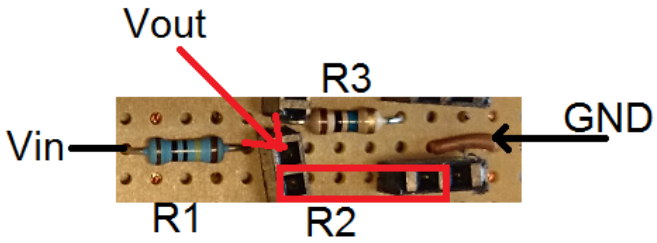


Figure 4.3: Photo resistor circuit on breakout board.

4.2 Encoders

The encoder is a two-channel hall effect encoder. The output of both encoders are square waves from 0 V to V_{cc}, both waves are about $90°$ out of phase. By counting both the rising and falling of the waves from E_A and E_B it is possible to get 48 counts per revolution of the motor shaft. Using only one output results in 12 counts per revolution of the motor shaft [7]. For reading an encoder output an interrupt (pin) is needed. In this prototype only the output of encoder E_A is read by an interrupt pin since the Arduino Mega 2560 only has 6 interrupt enabled pins. How exactly the encoder can be used to read the speed of the motor can be found on wikipedia [13].
The encoder has 6 wires with the following function:

Red Motor power A

Black Motor power B

Green Encoder ground

Blue Encoder Vcc (3.5-20V)

Yellow E_A/ Encoder A output

White E_B/ Encoder B output

The red and the black wire need to be connected to the M_a and M_b output of the motor driver respectively. The other wires need to be connected to pins on the Arduino which will be discussed in the breakout board section.

4.3 Motor drivers

The motor drivers main component is the h-bridge which leads to several pin part of the logical interface. An H-bridge is an electronic circuit that allows the motor to be able to run forwards and backwards by changing the value of the EN,DI and DIR pins according to figure 4.4. It enables a voltage to be applied across the motor in either direction. The H-bridge can be seen in figure 4.5.

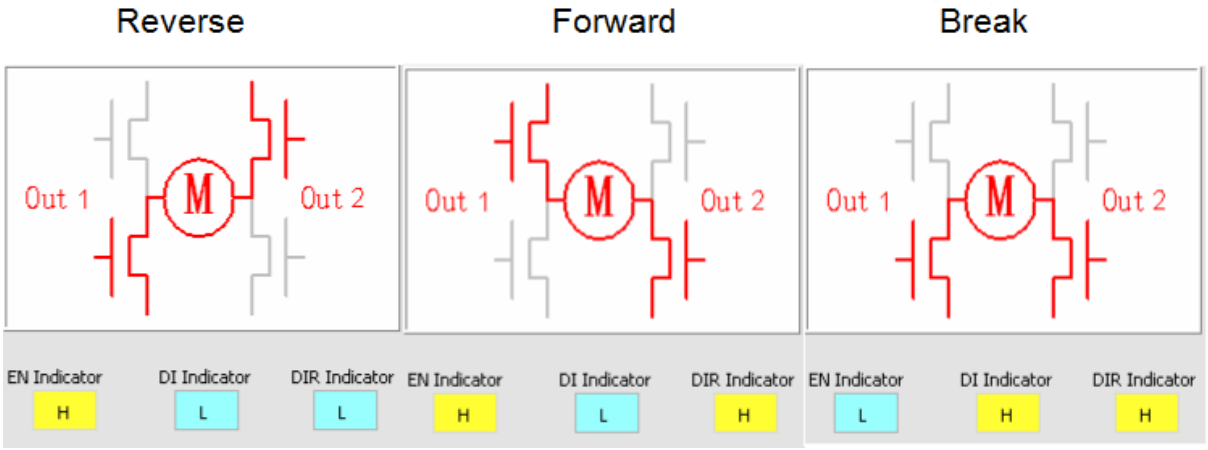


Figure 4.4: EN/DI/DIR with the H-bridge [4].

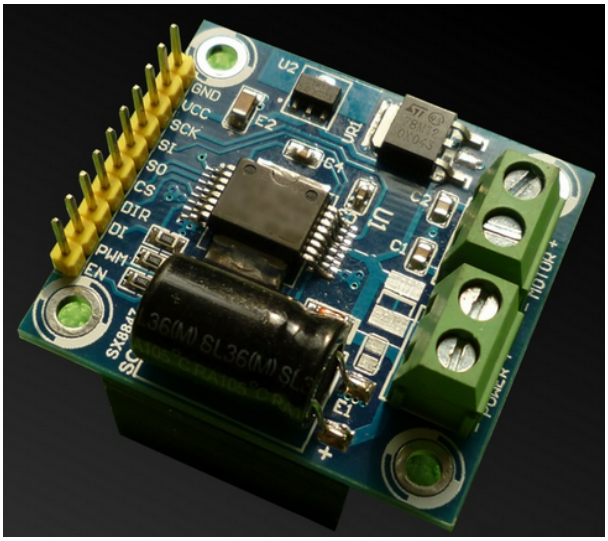


Figure 4.5: Motor driver [16]. The H-bridge is the black chip in the middle.

The yellow pin array in figure 4.5 has the following pins:

Ground (GND) The ground of the chip.

Power (Vcc) The power input of the chip.

Serial Clock (SCK) It is part of the Serial Peripheral Interface (SPI), a synchronous serial communication interface with a Master-Slave structure. On the SCK pin the output is done by the master [6].

Master Output, Slave Input (SI/MOSI) It is part of the SPI. On the SI pin the output is done by the master [6].

Master Input, Slave Output (SO/MISO) It is part of the SPI. On the SO pin the output is done by the slave [6].

Slave/Chip Select (CS) It is part of the SPI. On the CS pin the output is done by the master if the pin value is put on LOW [6].

Direction input (DIR) . The HIGH/LOW value on this pin decides the direction of the motor.

Disable (DI) Shutdown. With this pin you can shut-down the motor if the value is put on HIGH.

Pulse-width modulation (PWM) It is used to get analogue results with digital means. It changes the speed of the motor by creating a square wave between 0V and 5V depending on the PWM frequency value which can be given from 0 to 255 with 0=0V and 255=5V.

Enable (EN) With this pin you can turn on the motor if the value is set to HIGH and the DI pin is set to LOW.

In figure 4.6 it can be seen that DIR, PWM, DI and EN are part of the control logic of the motor driver. That means that the motor is controlled by the input the motor driver gets from those pins. The remaining pins apart from the GND and Vcc are for the SPI. SPI can be used to send data between micro controllers and peripherals. However since the SPI functionality was not implemented in the software by Mainini so its functionality will not be discussed in detail. From figure 4.6 it can be seen that the SPI is linked to, current limitation, over temperature and over- and undervoltage. With the SPI it is possible to monitor those values and set limitations for them. The Arduino board has a special pin cluster called ICSP (In-Circuit Serial Programming, which is a protocol for programming micro controller devices [17]) (figure 4.7) on which the SCK, SI and the SO pins of the motor drivers can be connected to. The remaining pins are connected to other pins of the Arduino as specified in the PinDef.h file.

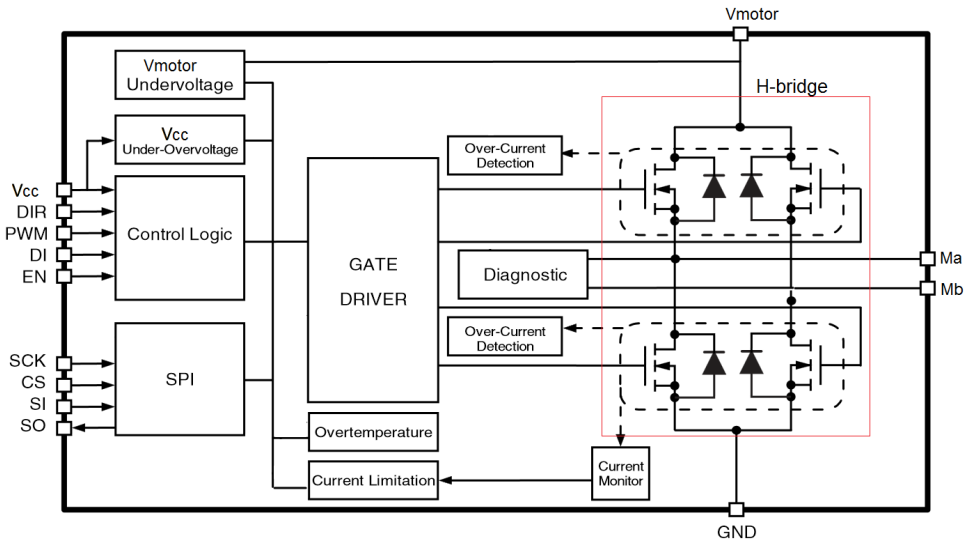


Figure 4.6: Motor drivers pin functions [5].

Figure 4.7: ICSP

4.4 Breakout board

To be able to connect these components in a more easy and clear way a breakout board has been prepared. In figure 4.8 the breakout board schematic can be seen and figure 4.9 shows the finished product.
Each box indicates a pin. The dotted lines within some pin boxes are an indication that it does not matter if they are connected, the solid lines mean that the pins are not connected.

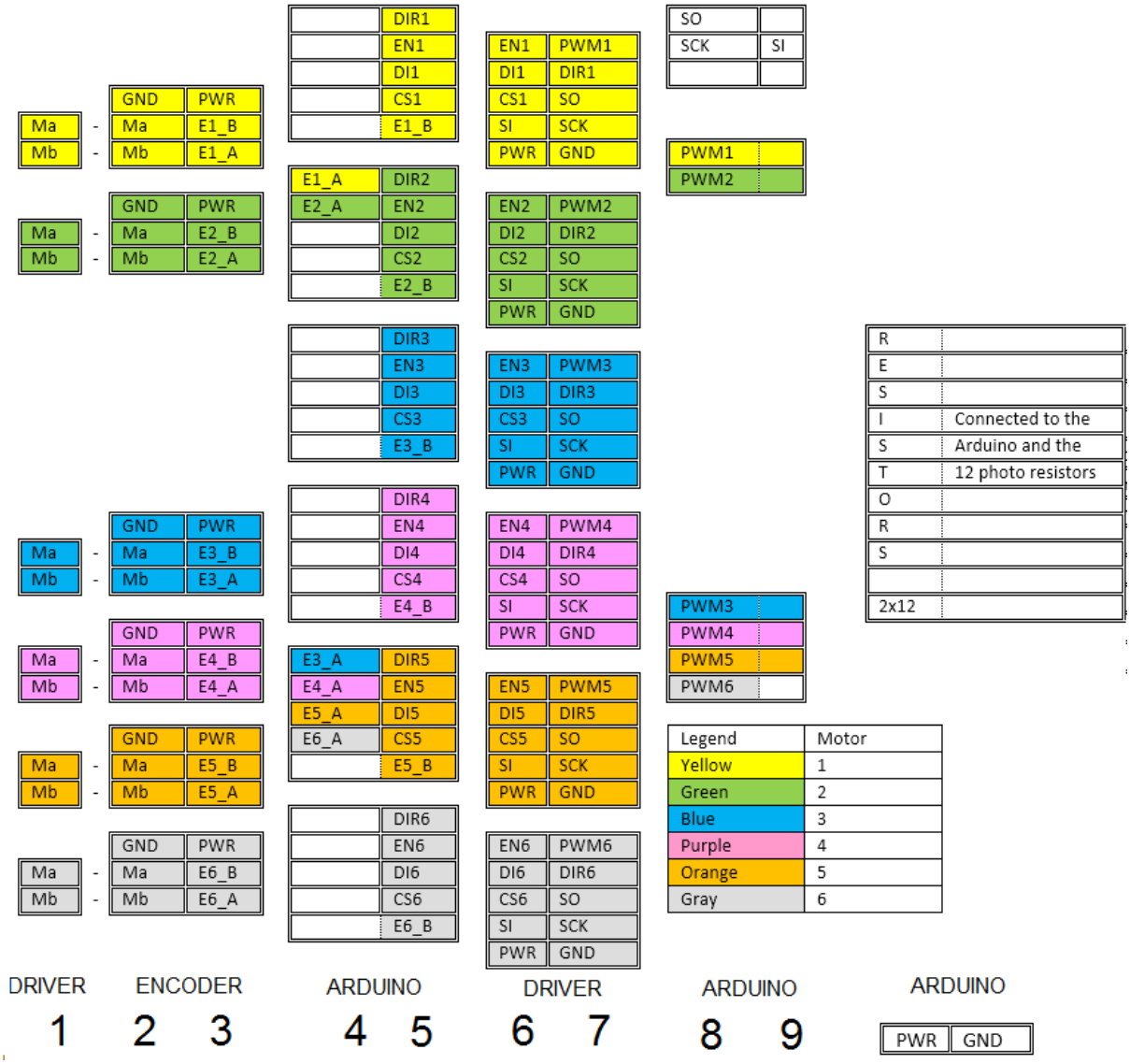


Figure 4.8: Breakout board schematic. In the legend can be seen which pins are for which motor. Two pins with corresponding names mean that those two pins are connected.

The breakout board has several rows of pins as can be seen. Most pins have a name/abbreviation, the pins with the same name in them are connected with each other. Below each set of columns the

hardware component it is connected to is written. Pins that belong to one motor control system are highlighted in the same colour as the legend in the same figure also shows. The resistor block on the side is 12 times the switch circuit mentioned in section 4.1 below each other. The block on the bottom is the power and ground connector, they should be attached to the 5V and the ground of the Arduino respectively. A more detailed explanation can be found in Appendix A.

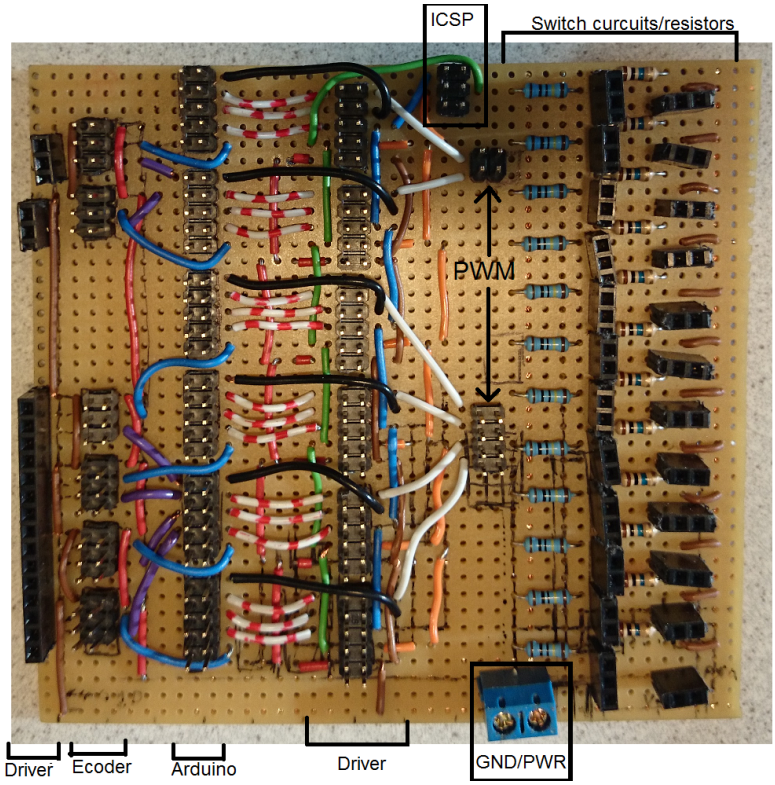
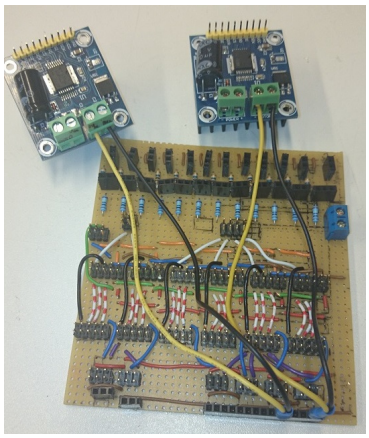


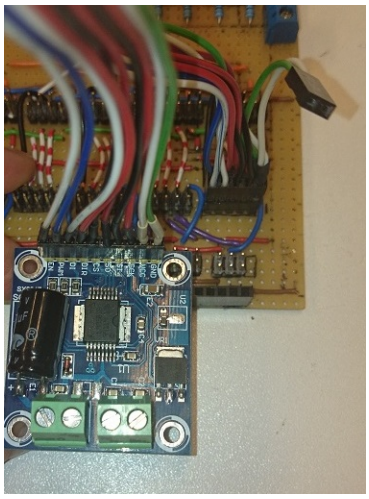
Figure 4.9: Breakout board.

4.5 Concluding remarks

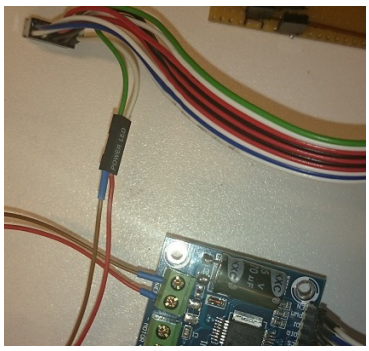
This chapter gave a clear as possible explanation on how to connect the hardware components and what their functions are. Apart from only the breakout board also the connection wires for the hardware components were prepared to complete the hardware part of the prototype. The wires can be seen in Appendix A. The full assembly of the hardware with two motors (without the switch wires) can be seen in figure 4.10.



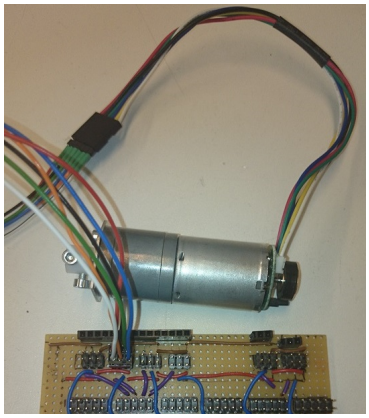
(a) Connecting the motor power.



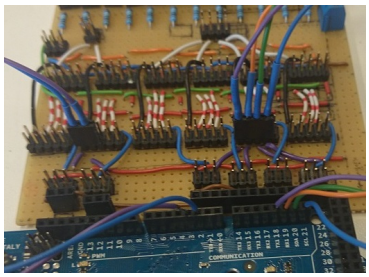
(b) Connecting the yellow in array of motor driver 6. The connection on the breakout board should be on the pin row behind the current one.



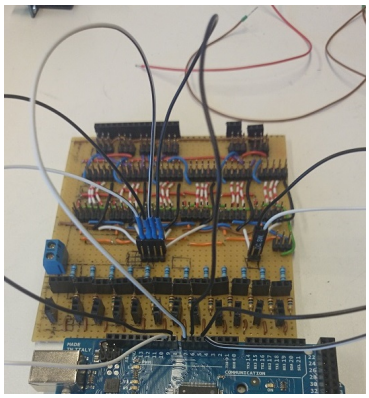
(c) Connecting the motor driver power (, do this two times).



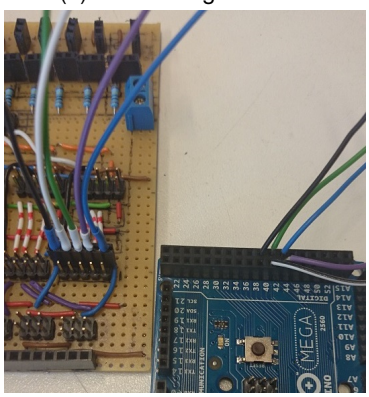
(d) Connecting motor 5.



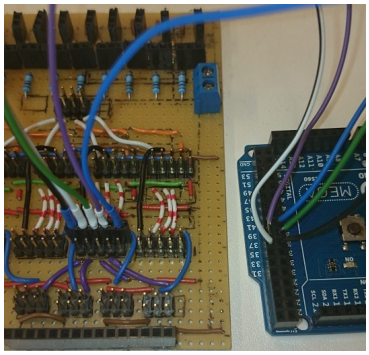
(e) Connecting the interrupt pins.



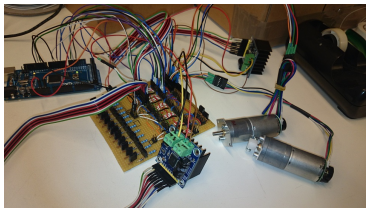
(f) Connecting pwm pins.



(g) Connecting the motor 6 pins to the Arduino.



(h) Connecting the motor 5 pins to the Arduino.



(i) The total assembly of 2 motors.

Figure 4.10: The assembly of two motors.

Chapter 5

Software

The software of the control system of the prototype consist of several components based on C++, LabVIEW scripting and Arduino scripting.

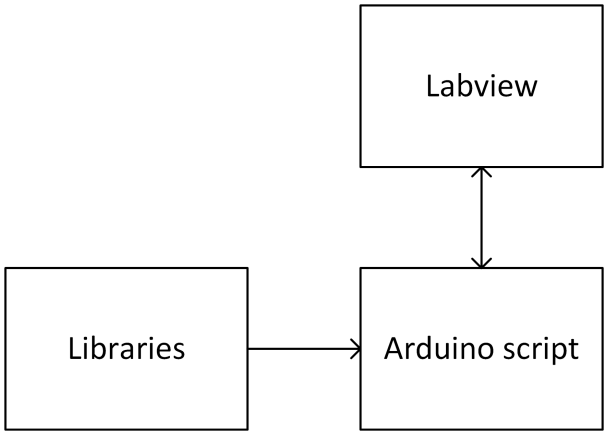


Figure 5.1: A schematic of the software components.

In figure 5.1 a schematic of the software components can be seen. The LabVIEW component is the human interface. Through the LabVIEW UI it is possible to adjust parameters, positions, angles and communicate things with the Arduino. The C++ scripts are libraries to the Arduino in this case. The library contains classes, variables and functions that are called on in the Arduino script.

5.1 C++

As mentioned in the introduction the Arduino script uses libraries as an extension on the Arduino environment to be able connect other hardware to the Arduino. The libraries contain functions and classes which can be used in the Arduino script once they are included. These libraries will need to be installed in the Arduino folder on the computer that is used to control the prototype.

The Arduino is controlled by the user interface which has been made in LabVIEW. To be able to process the commands given to the Arduino the library BUFFER2 and COMMAND have been created. The first library processes the string of data it gets from the Serial and divides them into individual smaller chunks. The COMMAND library then inspects those chunks and makes a usable command of them. These command types can be split up into two types. One for controlling the prototype movement and the other one for data storage.

To make the Arduino control the system first every pin has to be defined, that is done in the PinDef library.

The prototype itself has several parameters and variables which are used to control the motor. That data is stored on the Arduino's internal memory (EEPROM), the MEMORY and the EEPROMEx library are both made for storing the data on the Arduino. The MEMORY library specifies how many bytes every variable and parameter is allowed to be and the EEPROMEx library is an extension on the standard EEPROM library to store data in a better and more efficient way.
With each action of the prototype or command given to the Arduino a corresponding board status exist. The STATUS library was made to define and save that status.

Finally there are some libraries needed to control the position of the system. The TRANSDUCER, MOTOR2 and PID library are used for that. The TRANSDUCER library is used to set the positions of the transducers and the corresponding carriage positions. Those carriage positions are converted the amount of turns the motor would have to make by the TRANSDUCER library and that data is then given to the MOTOR2 library. The MOTOR2 library is used for turning the motors and specifies the properties of each motor. When the system is not on manual mode the PID library calculates the speed for the motors with the PID methode.

Only two libraries now remain the SWITCH library and the ONOFF. The SWITCH library is used monitoring the switch circuit to know when the carriage is homed. The ONOFF library has as function to give ON the value 1 and OFF the value 2.
In Appendix B more details can be found on what the libraries do.

5.2 Arduino

The Arduino script is a combination of several functions each with their own purpose. They are all are part of a main file for controlling the motor called NDT_controller_b_0_3.ino .

The main file includes the libraries that are used. It defines some of the first variables and it contains the script below which will be explained bit by bit.

First a serial port is opened and the data rate is set to 19200 bps. Then the SPI bus is initialized by setting SCK, MOSI, and SS to outputs, pulling SCK and MOSI low, and SS high.

```
1 void setup() {
    Serial.begin(19200);
3   SPI.begin();
```

Now the Interupt pins are initialized so that they are associated with the corresponding interrupt functions. The attachInterrupt are meant for making things happen automatically in micro controller programs and for helping solve timing problems. They are now used to read the encoder A output.

```
1   attachInterrupt(0, Inter1, intMode);
    attachInterrupt(1, Inter2, intMode);
3   attachInterrupt(2, Inter3, intMode);
    attachInterrupt(3, Inter4, intMode);
5   attachInterrupt(4, Inter5, intMode);
    attachInterrupt(5, Inter6, intMode);
```

Finally it is checked if it is the first time of start-up. If it is the EEPROM, the internal memory is initialized else the EEPROM will be read by the readEEPROM function.

```
    if(EEPROM.readByte(FIRST_TIME)==false){
2     startUp();
    };
```

```
4    readEEPROM();
}
```

Once the void setup() part is finished the script will loop the void loop() part.

```
1 void loop(){
```

Through the LabVIEW UI it is possible to communicate with the Arduino by giving it commands. That string is sent to the Serial (port).
The BUFFER2 library reads out the Serial (port) and makes sure that any command messages are recorded. Then the command is read and the proper action is induced.

```
1     BUFF.update();
      command.writeNew(BUFF.buffInExtract());
3       action();
```

The possible actions can be divided in 2 groups. One group is meant for controlling the prototype. The other group is meant for data saving, manipulation and getting information.

The action group for controlling the prototype:

ABT It switches off all motors.

HOM It puts the status on HOMEING. Which will cause the motors to be sent to the homing position.

POS It calculates the carriage positions and gives the information to the MOTOR2 library.

CNF It can configure the final positions of the carriages by moving the motor a little bit.

The action group for data saving, manipulation and getting information:

BRN It burns information describing the configurations to the EEPROM of the Arduino.

STS It prints the actual status to the Serial port.

PAR If the command is a question then it will print the motor state of the motor and print all other parameters. If it is not a question then it sets the parameters of a specified motor.

RPT It returns the previous used command. It does not do anything else apart from that.

UPD It updates the information the LabVIEW UI has on the system.

After that it is checked if the status was put on HOMEING or CONFIG.
If the status is HOMEING the motors will be moved to their home position, if the status is CONFIG the user is able to configure the motor positions. If the status was not HOMEING or CONFIG the script will check if the motors are on the right locations and will move the motors to the right location if needed.

```
1     if(STATUS==HOMEING){
          homeSet();
3   }else if(STATUS==CONFIG){
       configuration();
5     }else{
        switchOnMotors();
7     };
```

After that it will print the time to the Serial (port) as human-readable ASCII text.

```
1     Serial.print(millis()-time);
      Serial.print('\n');
3     time=millis();
}
```

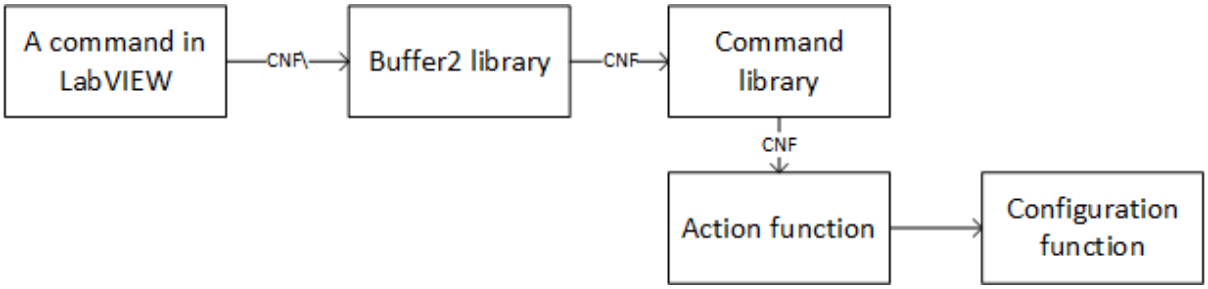


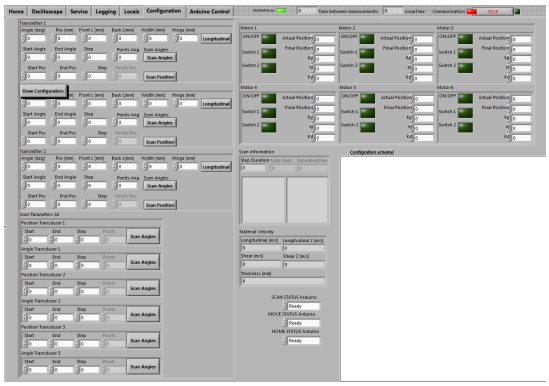
Figure 5.2: The route of a command CNF given in the LabVIEW UI.

So for example if the configuration command is given to adjust the position of one of the carriages slightly then the command given in the LabVIEW UI will follow the route which can be seen in figure 5.2. If the command string is longer than one command then it splits it into individual commands else only the backslash will be removed with the BUFFER2 library. The COMMAND library writes the current command to be the newCommand and then the action function will check which case corresponds to the commmand string it is given. After the case CNF is selected the configuration function is started.

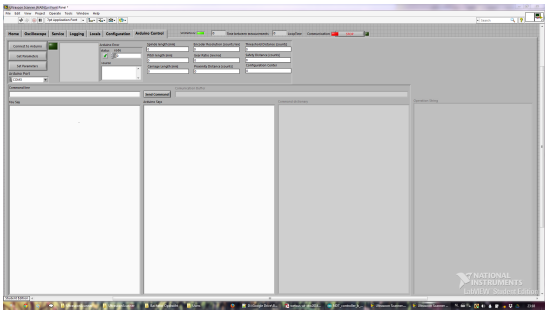
The Arduino script functions will be discussed in more detail in the Appendix C.

5.3 LabVIEW

The LabVIEW UI is based on a script which was already available for the XY traverse system available in the laboratory of the chair of Production Technology. The main differences are that Mainini added a Configuration panel (figure 5.3a) and an Arduino Control panel (figure 5.3b) to the already existing panels. He also added a small section to the main panel for changing the configuration of the transducers (figure 5.6). A bigger figure of figure 5.3a and 5.3b can be found in Appendix D. The current LabVIEW UI does only work on the computer in the lab because of the disk location of some files. However this problem will not be looked into during this project and further functioning of the LabVIEW script is beyond the scope of this assignment.



(a) The Configuration panel in the LabVIEW UI.

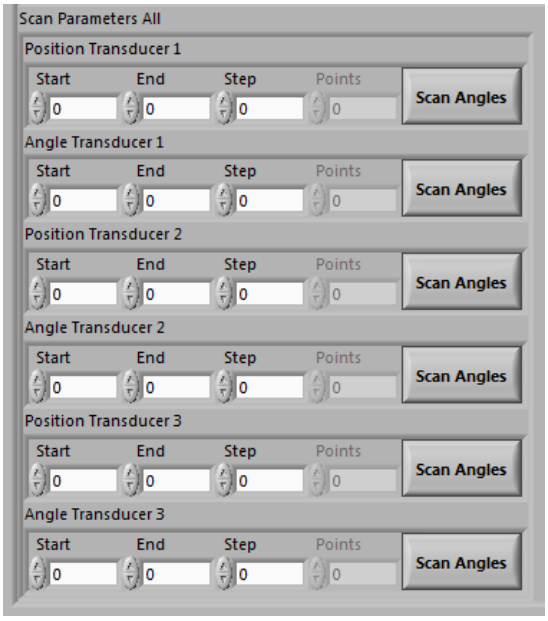


(b) The Arduino Command panel in the LabVIEW UI.

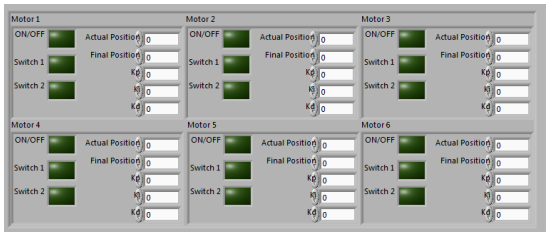
5.4 Concluding remarks

This chapter explained how the basics of the control system work and how the prototype is controlled by it. The Arduino script and the Library scripts have been debugged, edited and added where it was needed. The script is compiling without error which means that the Arduino script and the Library script are now free of bugs(, maybe apart from some hard to find ones). However a few problems have

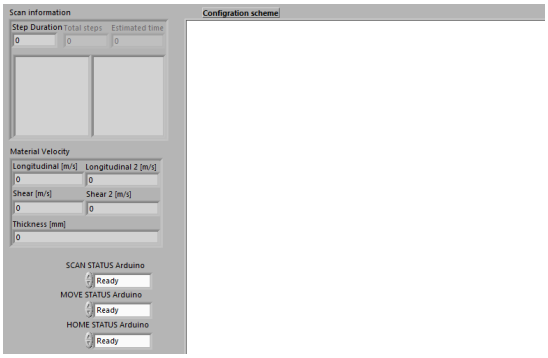
(a) The section to change parameters of the transmitters and change the angles and positions.



(b) The section to change the angles and positions of the transducers.



(c) The section with information about the motor being homed, being on or off and also the section to change the PDI values.

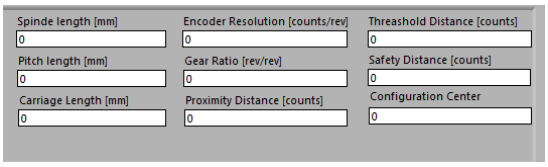


(d) The section with scanning information and the configuration scheme.

Figure 5.4: The Configuration panel in the LabVIEW UI in parts.

been found within the finished script. The problems concern the way the motors are moved which will be discussed in the next chapter and the method of vector calculations which will be discussed in Appendix B. Even with those problems the script is functioning, however solving those problems would improve the script.

(a) The section initialize the communication and connection with the Arduino and receives Arduino errors.



(b) The section to change the default script parameters of the prototype.

(c) The section with the command window. With and output of what has been send to the Serial by the user and what the Arduino replies in return.

(d) The section with the command dictionary and the operation string.

Figure 5.5: The Arduino Command panel in the LabVIEW UI in parts.

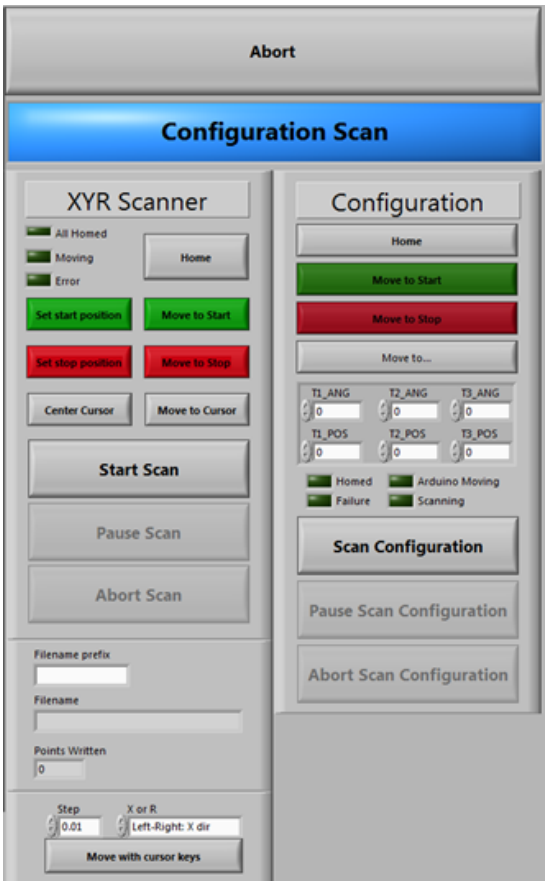


Figure 5.6: A section of the Home panel in the LabVIEW UI for the configuration scan.

Chapter 6

Testing

To finalize the assignment it is needed to know if the hardware and the software are functioning like they should be since it is not known if the prototype works for certain. To test that the following three tests will be done:

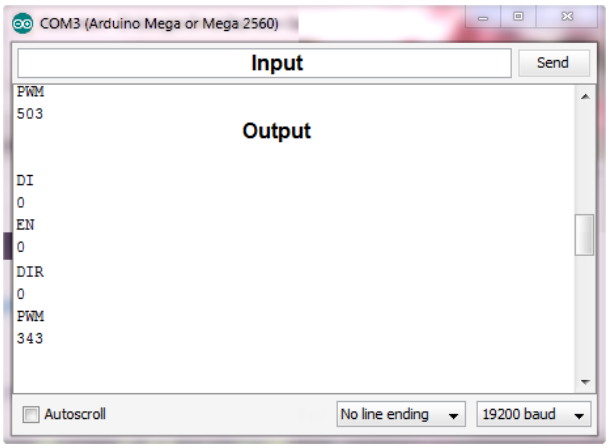
- Test the breakout board and the hardware by controlling one motor.
- Test the possibility of the system to move two motors at the same time since the current Arduino script is written in such a way that it will only move one motor at a time.
- Test the Arduino script to see if it works correctly and to find and solve any hidden bugs.

To prepare the Arduino program to upload the scripts (for the tests) the following things have to be done:

- Install the Libraries on the computer that is connected to the Arduino. The guide for that is on the Arduino website.
- Select the right board in the Arduino program through “Tools-Board-Arduino Mega or Mega2560”.

Communication with the Arduino without using the LabVIEW UI can be done use the Serial monitor (figure 6.1). Before any communicating can be done it is necessary to select the same data rate in the bottom left corner as for the Serial.begin() command else only gibberish (,letters and symbols of random length and order) will appear automatically in the output window.

(a) Serial monitor button.



(b) Serial monitor.

Figure 6.1: The Serial Monitor.

To communicate with the Arduino through the Serial monitor the commands need to have a certain format. This format can be seen in table 6.1. To make a command like PAR a question (,in this case to ask the parameters instead of setting them) put a question mark in front of it: ?PAR. For a command like POS1ANG50 that changes the angle of transducer 1 to 50 degrees the root consists of the first 3 letter and the branch is everything behind the root. The root can only consist of letters the branch can also contain numbers. To end a command a '\' is required to be put behind the command. This is needed for the script to understand that the information before the '\' is a command and it can also be used to separate multiple command written after each other before sending them.

Q	Root	Branch	End char.
"?"	ABC	123DEF	"\"

Table 6.1: Table with the command structure.

For testing the whole system the pin defining list of Mainini can be seen in a table in table 6.2. In figure 6.2 the front side of the Arduino can be seen with the pin (holes) and the corresponding

	DI	EN	DIR	PWM	SP_SELECT	E_A	E_B	SWITCH_1	SWITCH_2
M 1	10	11	12	4	13	2	14	0	1
M 2	15	16	17	5	22	3	23	2	3
M 3	24	25	26	6	27	18	28	4	5
M 4	29	30	31	7	32	19	33	6	7
M 5	34	35	36	8	37	20	38	8	9
M 6	39	40	41	9	42	21	43	10	11

Table 6.2: The pin definitions put in a table.

numbers/names.

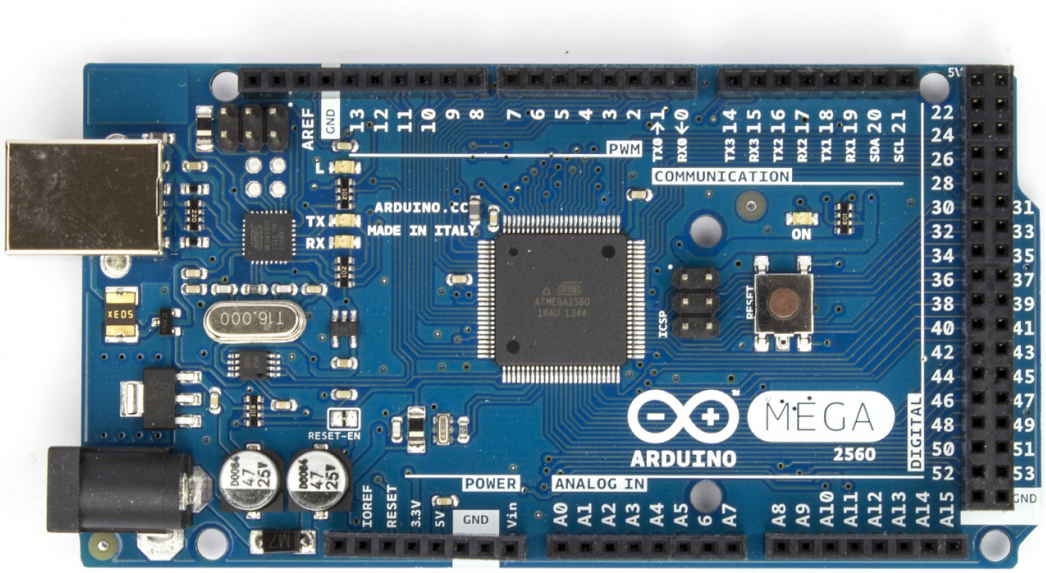


Figure 6.2: The Arduino front side [9].

6.1 Test1: 1 Motor

To quickly test if the hardware is functioning correctly it is possible to run the Arduino script below which will move one motor if the hardware is functioning like it should. Through the Serial Monitor mentioned before it is possible to adjust the speed of the motor.

First the pins on the Arduino where DI, EN, DIR and PWM pins of the motor driver are attached to are given a name.

```
// only the DI,EN, DIR and PWM pin are necessary to
//control the motors rotation
int DI_1 = 10;
int EN_1 = 11;
int DIR_1 = 12;
int PWM_1 = 4;
```

Then the pins are initialized to be an OUTPUT or INPUT pin.

```
void setup(){
pinMode(DI_1,OUTPUT);
pinMode(EN_1,OUTPUT);
pinMode(DIR_1,OUTPUT);
pinMode(PWM_1,OUTPUT);
```

After that the pins except the PWM pin are given a HIGH or LOW state to turn on the motor.

```
digitalWrite(DI_1,LOW);
digitalWrite(EN_1,HIGH);
digitalWrite(DIR_1,LOW);
```

Now the data rate will be set to 9600 bits per second for serial data transmission and ”speed 0 to 255” is printed to the Serial (monitor).

```
Serial.begin(9600);
Serial.println("Speed 0 to 255");
};
```

Finally if any input has been send to the Serial then Serial.parseInt() returns the first valid input which is then attached to the speed integer. If the speed value is between 0 and 255 then its value will be written to be the output value of the PWM pin and the motor will start turning.

```
void loop(){
if (Serial.available()){
int speed = Serial.parseInt();
if (speed >= 0 && speed <= 255){
analogWrite(PWM_1, speed);
};
};
};
```

6.2 Conclusion test 1

The motor did move so the hardware is working however sometimes the one row wide pin holes of the connector wires do not make contact with the pins on the breakout board. This is solvable by wiggling the connector a bit on the pins till it makes contact, however that freedom of motion is probably also the cause of the connection problem. Since this problem did not occur that often especially not at the end of the testing phase (probably because the connectors were not moved anymore) the connectors were left like that. If the problem returns or if this connection problem is deemed unacceptable then the solution would be to get crimp connectors with housing (figure 6.3) for the connectors instead of using single row female pin headers. The crimp connectors make better contact with the pins on the breakout board and have less movement freedom once they are pinned on the pins.

Figure 6.3: The crimp connectors can be seen on the right side and the housing on the left side.

6.3 Test2: 2 Motor

It has been noticed that the Arduino script delivered by Mainini only moves one motor at a time when changing its position or angle and when homing the motors. This is rather inconvenient since there is a maximum possible distance between two carriages of one transducer holder when the holder is attached. If one of the new carriage position is further away from the current position than the maximum possible distance then the movement would have to be in smaller steps or the transducer holders would have to be removed every time. Neither is a good option.

To test if it is even possible to move to motors at the same time the script of the test 1 will be adapted for moving two motors. Basically it means taking most of the script of test 1 times two with different names for the new pins. In theory the motors are switched on after each other since script and hardware wise it is not possible to execute two commands at the same time. To find out how much time it takes to turn on both motors a stopwatch function was built since it was very difficult to find the speed of executing code of an Arduino.

As can be seen in figure 6.4 the integer for starting time end elapsed time is made first for the stopwatch function. Then the script will initialize a few other things and the speed for the motors will be asked. After that the stopwatch is started and the speed of the motors is written to the drivers and the motors are turned on. After some time the motors are turned off. The turning on and off can be repeated several times to be able to calculate an accurate average time it takes to turn the motors on and off.

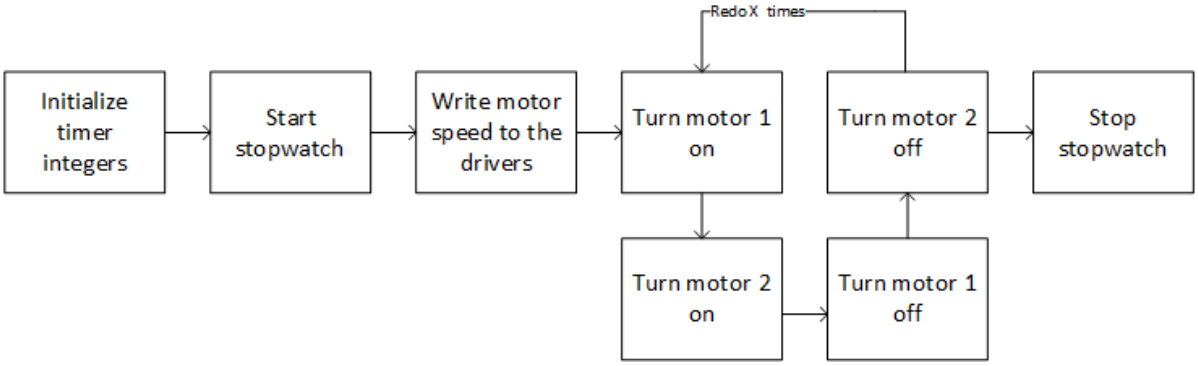


Figure 6.4: The stopwatch function.

First the pins are given a name.

```
//Motor 1
```

```
int DI_1 = 10;
int EN_1 = 11;
int DIR_1 = 12;
int PWM_1 = 4;
//Motor 2
int DI_2 = 15;
int EN_2 = 16;
int DIR_2 = 17;
int PWM_2 = 5;
```

The integer types for the stopwatch function are made.

```
int xtime=1; //starting count for the while loop
long startTime;// start time for stop watch
long elapsedTime;// elapsed time for stop watch
```

Then the pins are initialized to be an OUTPUT or INPUT pin.

```
void setup(){

pinMode(DI_1,OUTPUT);
pinMode(EN_1,OUTPUT);
pinMode(DIR_1,OUTPUT);
pinMode(PWM_1,OUTPUT);
pinMode(DI_2,OUTPUT);
pinMode(EN_2,OUTPUT);
pinMode(DIR_2,OUTPUT);
pinMode(PWM_2,OUTPUT);
```

The data rate is set to 19200 bits per second for serial data transmission and the script prints “speed 0 to 255” to the Serial (monitor) as long as nothing has been send to the Serial.

```
Serial.begin(19200);
while (! Serial);
Serial.println("Speed 0 to 255");
```

The DI and DIR pins are initialized to be an OUTPUT or INPUT pin.

```
digitalWrite(DI_1,LOW);
digitalWrite(DIR_1,LOW);
digitalWrite(DI_2,LOW);
digitalWrite(DIR_2,LOW);
```

The script asks the speed for motor 1. While no value has been chosen it will wait till an input has been given other than ‘a’. That input will be attached to the variable speed1.

```
void loop(){
Serial.print('\n');
Serial.println("Motor1");
int speed1 = 'a';
while(speed1=='a'){
if (Serial.available()){
speed1=Serial.parseInt();
Serial.print("speed1=");
Serial.println(speed1);
};
};
```

The script asks the speed for motor 2. While no value has been chosen it will wait till an input has been given other than ‘a’. That input will be attached to the variable speed2.

```
Serial.println("Motor2");
int speed2 = 'a';
while(speed2=='a'){
if (Serial.available()){
speed2=Serial.parseInt();
Serial.print("speed2=");
Serial.println(speed2);
};
};
```

The stopwatch is started. startTime will become the current time since the Arduino board began running the current script with micros() in microseconds.

```
startTime = micros();
```

If both speed values are between 0 and 255 then they are written to be to output value of the respective PWM pins. After that the motor is turned on for half a second and turned off for half a second and that is repeated 100 times.

```
if (speed1 >= 0 && speed1 <= 255){
if (speed2 >= 0 && speed2 <= 255){
analogWrite(PWM_1,speed1);
analogWrite(PWM_2,speed2);
while(xtime <=100){
digitalWrite(EN_1,HIGH);
digitalWrite(EN_2,HIGH);
delay(500);
digitalWrite(EN_1,LOW);
digitalWrite(EN_2,LOW);
delay(500);
xtime++;
};
};
};
```

Finally the stopwatch is ended and the elapsedTime time is the time since the Arduino board began running the current script at that moment minus the startTime from before. It then prints the elapsed-Time to the Serial (monitor).

```
elapsedTime = micros() - startTime;
Serial.print("elapsedTime=");
Serial.println(elapsedTime);
Serial.print('\n');
};
```

6.4 Conclusion test 2

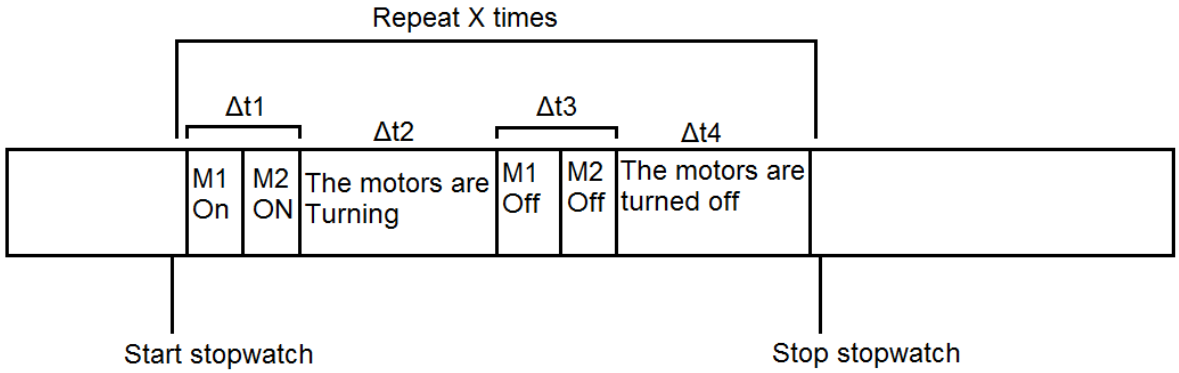


Figure 6.5: A graph of the stopwatch.

With the analogue value of the speeds being set to 70 of the 255 the total elapsed time is 100004116 microseconds. With figure 6.5 the following equation can be made to determine the time $\Delta t_1 + \Delta t_3$ and how much the carriage moves in that amount of time.

$$\Delta t_{tot} = \Delta t_{1tot} + \Delta t_{2tot} + \Delta t_{3tot} + \Delta t_{4tot} = 100004116 \mu s \tag{6.1}$$

$$\Delta t_2 = \Delta t_4 = 500000 \mu s \tag{6.2}$$

If X is 100 then the total amount of time that the motors were turning and were not turning is:

$$\Delta t_{(2+4)tot} = 100 \cdot (\Delta t_{2tot} + \Delta t_{4tot}) = 100000000\mu s \tag{6.3}$$

Then the total time it takes to turn the motors on and off is:

$$\Delta t_{(1+2)tot} = \Delta t_{tot} - \Delta t_{(2+4)tot} = 100004116 - 100000000 = 4116\mu s \tag{6.4}$$

So for turning both motors on and off only once the time is:

$$\Delta t_{1+2} = 4,116\mu s = 0.04116ms \tag{6.5}$$

To know how much the carriages could have moved in that time it is necessary to know a few things. The motor has a maximum speed of 120 revolutions per minute (RPM) when attached to a 6V output [7].

$$v_{max,motor} = 120RPM = 2RPs = 0,002RPms \tag{6.6}$$

The amount of revolutions that is possible in the Δt_{1+2} time is:

$$Rev_{t_{1+2}} = v_{max,motor} \cdot \Delta t_{1+2} = 0,002 \cdot 0,04116 = 0,00008232 \tag{6.7}$$

The spindle has a spindle pitch of 0,3 mm/revolution

$$spindlepitch = 0,3mm/rev \tag{6.8}$$

The total displacement for turning motor 1 and motor 2 on and off in this case would be:

$$Displacement = spindlepitch \cdot Rev_{t_{1+2}} = 0,3 \cdot 0,00008232 = 0,000024696mm = 24.7nm \tag{6.9}$$

That is only about 0,008 % of 0,3 mm and thus can be considered negligible.

This shows that is possible to start two motors at almost the same time. To implement this in Mainini's script means that it should be rewritten which will take some time. The most likely reason Mainini choose to move the motors individually is because he uses the PID methode to calculate the speeds for the motors. In the controlMotors function script (which is not in use because it should do the same as the already existing switchOnMotor function) the following was commented by Mainini: "With the PID approach it is necessary to make sure that only one motor per time is ON". The PID method is useful for automatically calculating the speed for the motors however it is not necessary since the user can also manually select a speed in the same way as in the script above for example. To implement the two motors turning at the same time at least the MOTOR2 library, homeSet Arduino function and the switchOnMotor function would need to be adapted. Apart from that it is possible with the script above to have a transducer translation, rotation or a combination by changing the speed of motor 1 and 2.

6.5 Test 3: The Arduino script

While working on completing the script some bugs were found in the Arduino and library scripts with using the script verifying option in the Arduino program. This does suggest that Mainini did not fully test if the Arduino script was working correctly. From that the idea came to test the prototype Arduino script to see if it works correctly for as far as that is currently possible to test.

The Arduino scripts have partially been tested using the Serial.println() function that prints the value between the bracket to the Serial monitor. It can be used to print the state or the value of a pin that is read with digitalRead() or analogRead() or it can be used to print a letter or word to show the script user that the script is executed completely/correctly by the Arduino. Also some values like position or motor status were manually written to the corresponding integer to see its effects.

The script has been tested in general with the action function because that one contains all the commands that will be used. To execute all actions after each other all of them have been filled in the command window which can be seen in figure 6.6.

In the Serial monitor all commands that have been send to the Serial have returned their action, status or something similar (except NAC because it is the “not a command” action which returns nothing to the Serial monitor) which indicates that the action function is working correctly. Also the short functions ABT, STS and RPT are working correctly since they give all the information they should give.

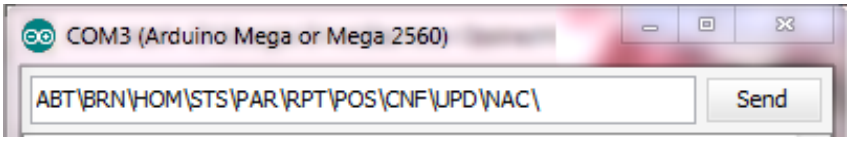


Figure 6.6: All the commands available in the action function will be send to the Serial so the action() function will execute them one by one.

The only longer command functions that have been tested are the configuration function,the homeSet function and the setParameter function. The configuration function has been tested with two motors. In figure 6.7 the Serial monitor can be seen during the execution of the configuration function. According to the pin states which can be compared with figure 4.4 the script gives the correct information for the motor to turn forward and backward.

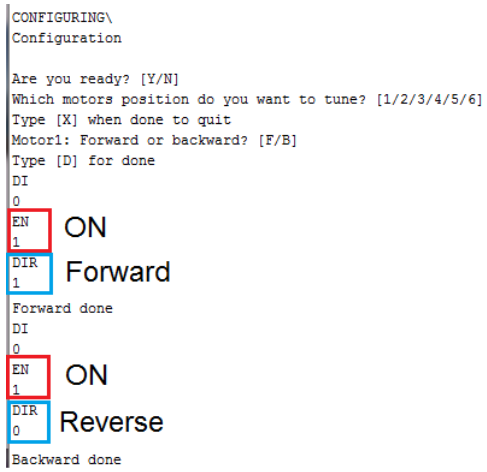


Figure 6.7: The Serial Monitor with the configuration function executed. It shows the states of the DI,EN,DIR,PWM pins of motor 1 after the function is executed.

The testing of the homeSet function has been done without the feedback of the photo resistors so it is not sure if the photo resistor feedback works correctly. When comparing the read pin values in figure 6.8 with the information from figure 4.4 it can be seen that the HIGH and LOW values of the pin correspond the turning and the breaking of the motor which meaning that the motors receive the correct signals to turn and to shut off once the motor driver gets the signal that the carriage is at the homing position.

When wanting to set the position of the motors and thus the transducer the actions in the flowchart of figure 6.9 happen. In figure 6.9 transducer 1 has to be set to an angle of 15 degrees. The BUFFER2 library will split that command from any other possible commands in the Serial buffer and the COMMAND library will split the command in the main command POS and the branch that specifies what the position change should be. The action function recognizes the POS command and calls the setPosition function which will figure out which transducer the new configuration is for and if it is for the angle or the position. The new configuration will be written to the newAngle of the newPos variable in the TRANSDUCER library so that it can calculate the new configuration. After that the checkRules function will check the new configuration for any problems. These problems are collision with the outer walls or with the other transducers and exceeding the maximum distance between two carriages of one transducer.

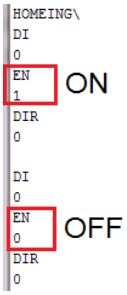


Figure 6.8: The Serial Monitor with the homeSet function executed and the states of the DI,EN,DIR,PWM pins printed to check if the motors get a signal to turn.

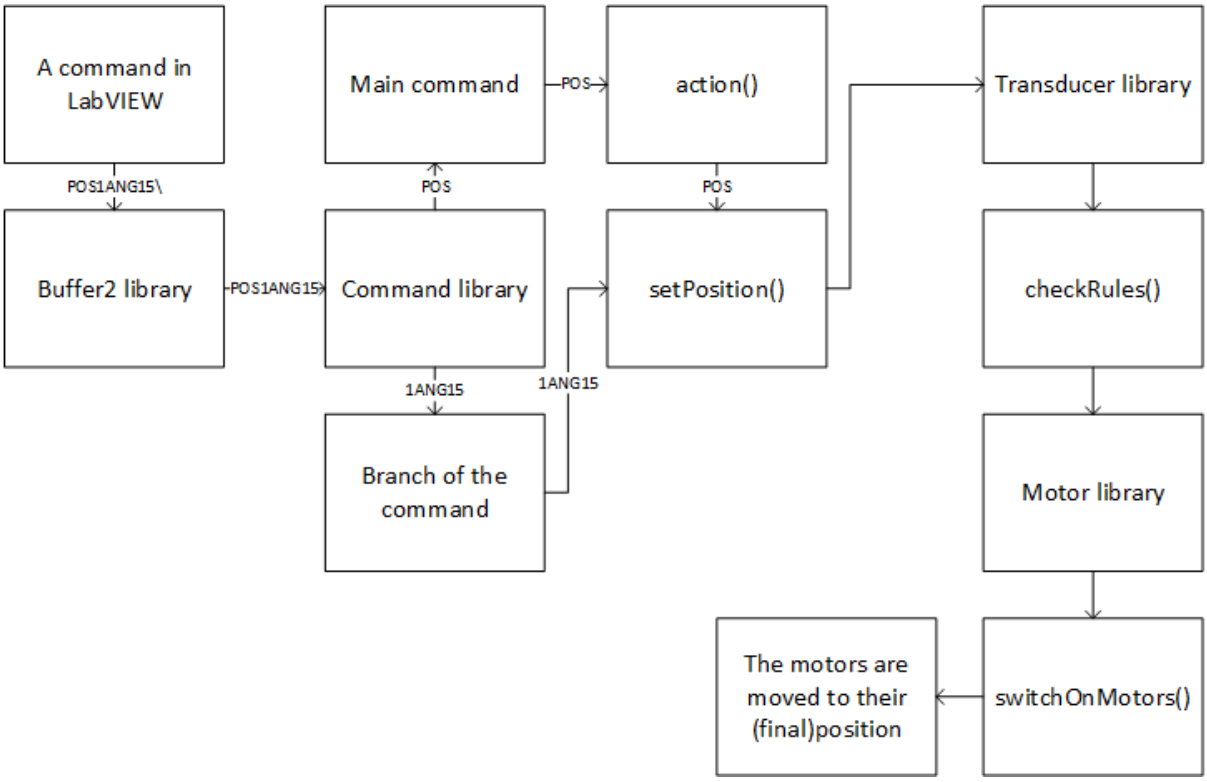


Figure 6.9: A flowchart about setting the position of the motors and transducers.

If the new configuration is correct the carriage positions from the TRANSDUCER library will be given to the MOTOR2 library. The MOTOR2 library will save that information to calculate to direction and speed for the motor once the switchOnMotor functions is executed and the motors are turned on.

6.6 Conclusion test 3

From the previous test it can be concluded that at least the basics of all the command functions work. Some command functions like the configuration function, the homeset function and the set position function are more complicated and longer than other functions that only have to save parameters and return asked values. These three functions have been tested more extensively and they now work correctly like they should for as far as it is possible to test without any automatic position and homing feedback.

6.7 Concluding remarks

From all three tests it can be concluded that the prototype is functioning as it currently is however there is still room left for improvements to the system. The possible hardware and software improvements have been mentioned before. If it is worth the time to improve the prototype depends on if this is (almost) the final prototype product or if it is the first one in a series of many more to come. One thing to keep in mind during the testing is that the connection between the Arduino and the computer can easily get lost. Usually symptoms are that the Serial monitor does not seem to work/renew any more or that the commands that are send are not displayed on the Serial monitor screen or that it is not possible to upload the (new) script. This disconnecting can also happen when a pin or a cable is removed or attached. The connection is restored again after the USB cable is disconnected and connected again however this is only solves the problem temporary. For a final solution the real cause of this problem has to be discovered.

Chapter 7

Conclusion and Recommendation

7.1 Conclusion

The objective of this assignment was to deliver an operation control system, including proper documentation of the connections and functions of the system in terms of software and hardware. The work described in this report has resulted in getting the prototype started by Mainini a step closer to test phase ready. Small details regarding the size of several (aluminium) components of the prototype which were only noticed at the last moment made it difficult to assemble the whole prototype. However most of them are solvable by adjusting the size with the cutter in the mechanical engineering workplace.
The starting point of this assignment was a prototype which should have worked however nobody knew how and any useful documentation for continuation of the project was difficult to find. Some of the hardware components were missing and the software needed some additions and adjustments.
The main assignment goal was to finish the basic control system of the prototype. For that the following has been done:

- The Arduino script has been completed and debugged
- The libraries have been completed and debugged.
- The breakout board has been made.
- The connection wires/ connectors that are needed to connect the hardware components together have been prepared.

To check if everything worked properly three tests were done:

The first test was to check if the breakout board and the connectors were working correctly. This was done by attaching the needed hardware and connectors for controlling one motor and running a simple script to see if the motor would turn. The result was that the made hardware worked correctly to turn one motor.

Test two was to see if it is possible to turn two motors at the same time since the current Arduino script by Mainini moves the other motor after the first one has been turned of. In theory it is not possible to switch on both motors at the same time. With using a stopwatch function and calculating the displacement during turning one motor on and then the other the offset was tested. From the test it can be concluded that a displacement of 24.7 nm during turning both motors on and off can be considered a negligible offset, so it is possible to start both motors at almost the same time.

Test three was done to check is the script contained any remaining harder to find bugs/errors. This was done by running several functions and comparing the output shown in the Serial monitor window with what should have been done/returned according to the script. The remaining bugs that were found were solved and the final test showed that the current command functions of the Arduino script work correctly.

During the testing it was noticed that the system had some problems with connectors/cables/wires that disconnected. It is needed to find a solution for that to improved the system so that it can function fluently without any interruptions. Any further recommendations and points for attention will be mentioned below in the recommendation section.

The last goal of this assignment was to make this prototype project ready for easy future continuation. For that extensive documentation of the hardware components has been done and the software scripts have been provided with additional comments. From this point on it should be possible to quickly understand the workings of the system for further testing of the prototype. Looking back at the final objective of this assignment and at what has been realized it can be concluded that the objective of this assignment has been reached.

7.2 Recommendation

While spending some time on the prototype made by Mainini several comments can be given regarding the prototype.

The first thing that has been noticed is that the motor driver capabilities are too extensive for what it is actually used for. The drivers have SPI functionality to communicate with the Arduino however in the script nothing can be found on using this capability apart from starting the SPI. The SPI of the h-bridge can be used for detailed (failure) diagnostics on each channel and the h-bridge itself. Such diagnostics include short circuit to battery, short circuit to ground, short circuit overload and about temperature. SPI is also used to set things like voltage and current slew-rate control for low EMI, SPI current regulation threshold and the thermal warning bit [4, 5]. In theory it would also be possible to control the motor drivers through the SPI pins (,exact documentation on how has not been found yet). From the general examples of the usage of SPI for basic things like blinking LEDs it was noticed that the Arduino writes the information about the blinking to the SPI pin. Currently the motors are controlled by the values given to the EN,DIR,DI and PWM pins of the drivers. To implement the SPI method to control the motors the Arduino script would needs to be rewritten. At the moment controlling the system works fine as it is and no added value is seen into implementing the SPI for controlling the system.

Furthermore during the testing phase with two motors lots of hardware components and wires were present including the wires of the switch systems. It gets very messy and unclear. It is the best thing to omit wires for functions that are not used (like the SPI wires) and maybe think of another way to connect all the components (without a breakout board).

Another thing that has been noticed is that the motors need to be homed every time for having a reference point so that they can accurately be moved to the new position. The only place where it is possible to home the motors is the ends of the rails of the carriage which is not convenient. This is because the transducer holders need to be remove else the transducers would collide. It would be better to have several reference points along the moving axis of the carriages by placing more photo resistors in the rails. Having more reference points would take less time with referencing the position of the carriages for the next position. It would also take away the hassle of removing the transducer holders every time the motors need to home. It is also possible to choose another way of referencing/measuring the carriage positions like attaching a linear displacement sensor to the carriages.

The comments above are all things to consider for later since it would require the current prototype to be adjusted. There are also several things that should be considered before assembling the prototype completely.

The first things is the ball-bearings which are too big for the spindles. Currently plastic rings and isolation tape is put over the spindle ends to keep the ball-bearings in place. It is not sure how watertight that temporary solution is and getting the right sized ball-bearings should be considered.

Another problem with those plastic rings and isolation tape is that they are the ones that make a seal with the rubber rings that are placed in the ball-bearing holders seen in figure 3.2. It is not sure if this seal is watertight enough. Getting smaller rubber rings that do make a seal with the spindle itself would be more logical and would also probably be better to assure a watertight seal instead of the current seal.

In addition to the rubber-rings is has also been noticed that there are other another problem with the motor compartment lid of figure 3.2. The ball bearing holders have been glued into place however they were not glued properly which caused openings between the hole in the plastic for the ball-bearing holders and the exterior of the ball-bearing holder. It is even possible to put a wire through that hole so water should also be able to go through. Furthermore the two ball bearing holders on the right side of the lid do not have the right size causing the ball-bearing not to fit in.
Possibilities to fix this could be to use the cutter in the workplace to adjust the size of the two ball-bearing holders which are too small. Then try to remove all the rubber rings and replacing them by smaller ones and to add some glue to the holes between the holders and the holder openings in the lid. However it is not sure if that is easily done. Especially the rubber rings changing part may be difficult. Remaking this lid might possibly be easier.

Appendix A

Hardware

The hardware components of the prototype consist of:

- switches/photoresistors
- encoders
- motor drivers
- Arduino

All the components are connected to the breakout board with connectors and wires which have been prepared and can be seen in figure A.4. The details of the individual components and the layout of the breakout board are discussed below.

A.1 Switches

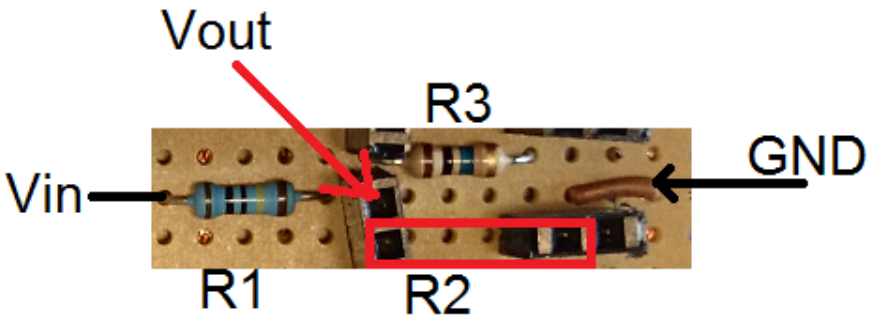


Figure A.1: Photo resistor circuit on breakout board.

From left to right the 1MOhm resistors front side can be seen. The vertical row next to that is the Vin line on the other side of the breakout board. After that at the place of the resistors back side the pin below and above the pin hole of the resistors back side are soldered together. From there it splits up in a 10MOhm resistor, the pin hole for the Vout which goes to a corresponding Arduino pin and the pin hole to connect the one of the wires of a photo resistor with the hole for the other wire opposite to it. The two resistors in parallel then come together to be connected to the ground by the brown wire.

A.2 Pins

Something to keep in mind while looking at the pin definition file is that the Arduino mega 2560 has a few functions pre-assigned pins, like the interrupts and the PWM. The external interrupts are located at

pin 2 (interrupt 0), 3 (interrupt 1), 18 (interrupt 5), 19 (interrupt 4), 20 (interrupt 3), and 21 (interrupt 2). The PWM is located on pin 2 to 13 and 44 to 46. That is the reason for the strange placements of some motor pins.

A.3 Breakout Board

With the old breakout board it was unclear what each pin should be connected to and which wires connected with which pins. To avoid the same problem a schematic has been prepared for the pins and also a schematic has been prepared for some of the wires, they can be seen in figure A.2 and figure A.3 respectively.

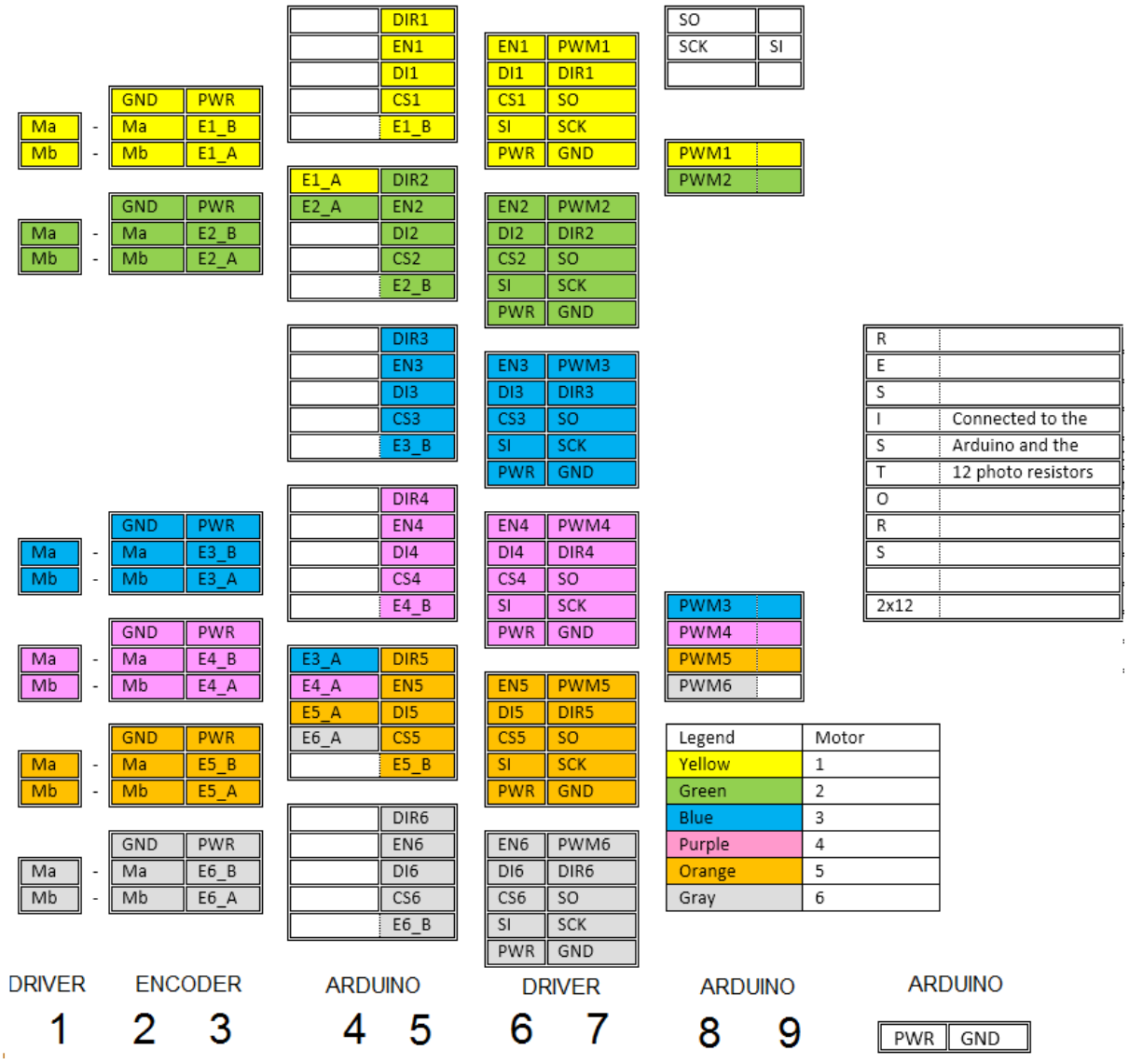


Figure A.2: Schematic of the breakout board.

The wire schematic does not show most wire connection because that would make the picture only messy and most of the connections should be relatively clear from the previous scheme. The lines in the background are a representation of the backside of the breakout board. The vertical solid lines (after every 3 holes) indicate that the two holes on both side are not connected by the same copper strip.

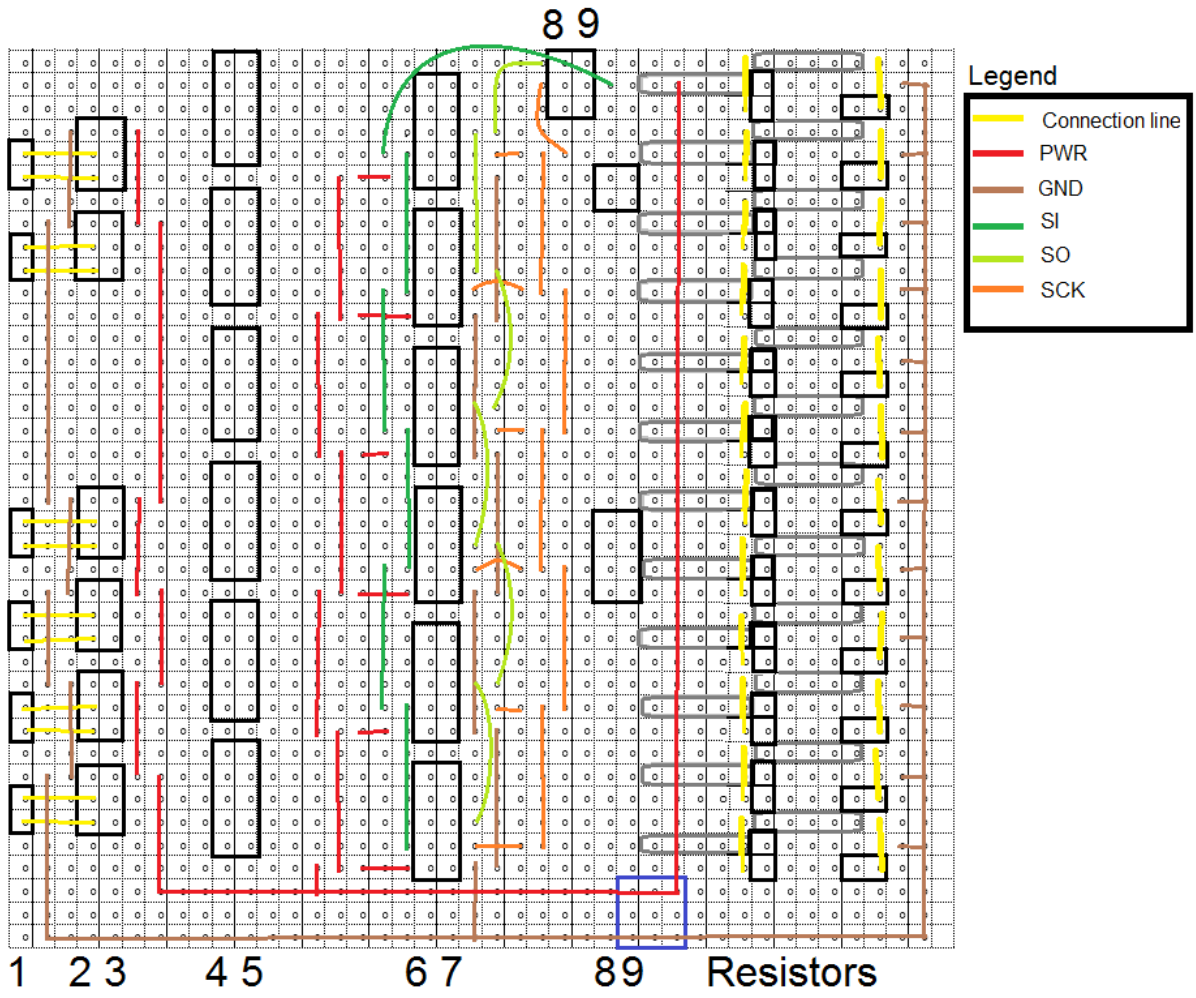


Figure A.3: Breakout board wire schematic.

The rows as you can see are numbered from 1 to 9 from left to right.

Row 1 Contains all the Ma and Mb, they are connected to the Motor power output of the motor drivers. The small line to the right of the boxes of Ma and Mb indicate that they should be connected to the respective pins of row 2.

Row 2 and 3 Connected to the motor encoder. Ma and Mb of row 2 are connected to the respective pins of the motor encoder. All the E__B pins are connected to the encoder output B pins and all the E__A pins are connected to the encoder output A of the respective motor encoder.

Row 4 and 5 Connected to other corresponding pins on the breakout board and the Arduino.

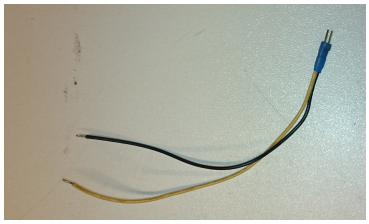
Row 6 and 7 Meant for the motor drivers. The cable in figure A.4b is provided for the connection of the motor driver to the breakout board.

Row 8 and 9 Cconnected to the Arduino again. The top pin row of 2x3 is connected to the ICPS of the Arduino. Keep in mind to put SO on the SO and etc.

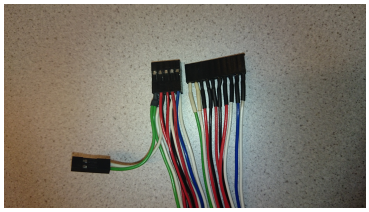
Apart from the straight line in the drawing we have three other things, the black boxes which are pin headers or pin connectors, the blue box on the bottom which is the power connector and the gray rounded boxes on the right which are resistors. Along the top and the bottom of the figure the numbering that corresponds with the schematic drawing has been added to be able to compare.

A.4 Wires and connectors

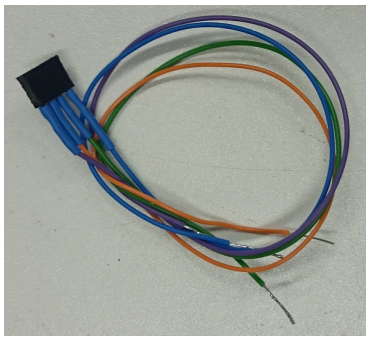
Several wires in figure A.4 have been prepared to connect the breakout board pins to the other hardware components and to the Arduino. The heat shrink on the connectors for the motor drivers in figure A.4b have a different colour for the power and ground wire to make it easier to connect to the pins. For the cables for the encoders the wire colour have been kept the same as the colours of the wires of the encoder except for the yellow wire which has been made orange in the prepared cable in figure A.4d.



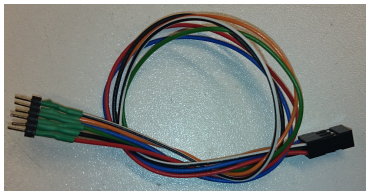
(a) The wire that goes in M_ a and M_ b.



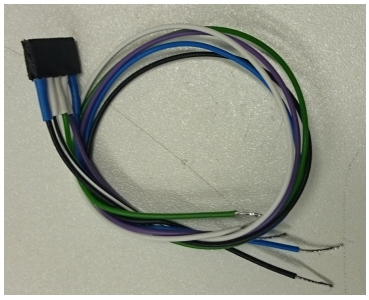
(b) Connector for the motor driver to the breakout board.



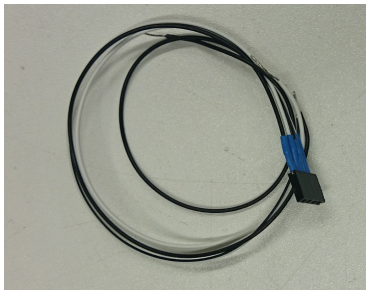
(c) Connector for four of the six interrupt pins.



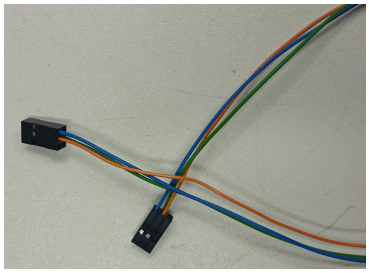
(d) The wire for the encoder cable.



(e) One of the six connector for the pins in row 5 of figure 4.8.



(f) Connector for four of the six pwm pins.



(g) Wire for the ICSP connection.

Figure A.4: Wires

Appendix B

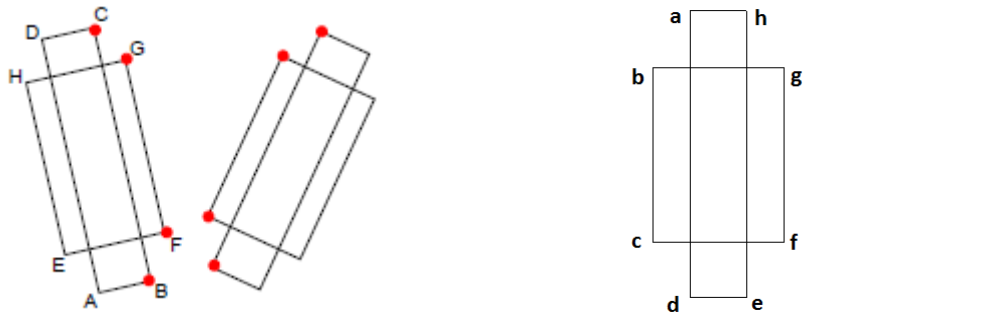
C++

The libraries used for this prototype are extensions of the basic Arduino environment to be able to work better with the hardware components and to do the calculations. To be able to use them they first will need to be installed in the Arduino folder on the computer that is used to control the prototype. On the Arduino library page can be found how to install them [8] .
For the prototype the following libraries are used. A small explanation of the function of each library is included below.

BUFFER2 It initializes the buffer for for the input the serial port gives. It also splits the buffer string into the individual commands.

COMMAND It initializes the possible command types.

TRANSDUCER It contains the transducer movement parameters script. It also contains the rules for the movement restrictions which are: transducer-wall interactions, transducer-transducer interactions and the carriage distance of one transducer. Figure B.1a describes the transducer outlining that was included in the report by Mainini, however this picture is not correctly lettered when compared with the script. Figure B.1b contains the correct lettering.



(a) Old outlining.

(b) New outlining.

Figure B.1: Transducer outlining.

MOTOR2 It initializes each motor and it contains the motor controlling parameters script which works according the switch map which can be seen in figure B.2. The script also links the PinDef.h information to a script variable so it can be read and written.

SWITCH The switches/photo resistors are the homing positions for the motors so the switch script contains functions to check if the carriage is on the homing positions or not by comparing V_{out} to the threshold. Since there probably would be an error with this library if the function checkStatus() was ran for the first time the script was edited a bit.

Switch	EN	DI	DIR
Forward	EN Indicator H	DI Indicator L	DIR Indicator H
Break	EN Indicator L	DI Indicator H	DIR Indicator H
Reverse	EN Indicator H	DI Indicator L	DIR Indicator L
Break	EN Indicator L	DI Indicator H	DIR Indicator L

Figure B.2: EN/DI/DIR switch map [4].

PID It is the PID controller script which is a basic Arduino PID library. More detailed information can be read on the page of the maker of this library [10].

ONOFF It is a enumeration of ON and OFF, making ON 1 and OFF 2.

MEMORY The memory library is only used to specify how much memory space each variable is allowed to take, it contains the registry addresses.

PinDef It defines to which pin of the Arduino the pins of the breakout board should be connected.

EEPROMEx It is an extension of the standard Arduino EEPROM library. It writes and reads basic types like bytes, longs, ints, floats & doubles. It can also read/write single bits, arbitrary data-formats and arrays. It adds debug functionality to identify and stop writing outside of the EEPROM memory size and excessive writing to prevent memory wear. More information can be found on the website of the maker [11]. There is also a new version available of the EEPROMEx code with some small changes, it has not been implemented in the others scripts yet.

STATUS It saves the current status of the board and returns the current status.

As a side note: Currently pointers are being used to do vector calculations in a few libraries. (At least in TRANSDUCER,SWITCH and MOTOR2.) Pointers shortly explained are variables that store the address of another variable, so not its value but its address. Pointer can be used to access the variable they point towards directly to change its value indirect however they are difficult to understand and to work with. For better understanding of how pointers work it is advised to read a tutorial about pointers. The main problem however is that Mainini seems to mistakenly calculate values in the libraries using the address of the variable itself instead of the variable. Mainini also switches a lot between attaching a pointer to a variable and using the address of that variable to calculate something which will result in an address, he then takes the value of that address to calculate a value to take the address of that value again etc. Probably from the previous sentence can be understood how confusing Mainini has used the pointer. Currently it is not known if the pointers work correctly for the vector calculations and that gives another thing wrong with pointer.
When pointers have been used wrong no warning of the bug will be given, “debugging will have to be done manually and it will be very difficult because the pointer is connected to the rest of the program as

it contains the memory locations" [12]. The easiest solutions would be to use a Vector class for those vector calculations. However it is not sure how much that would influence the other scripts therefore an example will be provided in section B.1 if implementation is deemed necessary.

B.1 Vector library

```
Vector2D.h
_____________________________
#ifndef Vector2D_h
#define Vector2D_h

#include <math.h>
#include <vector>

class Vector2D{
public:
  double X,Y;

  Vector2D();
  Vector2D (double x, double y);
  double getLength ();
  double getLengthSquared ();
};

#endif

Vector2D.cpp
_____________________________
#include "math.h"
#include "vector"
#include "Vector2D.h"

//If no arguments are given, it is an empty vector
Vector2D::Vector2D(){
  X=0;
  Y=0;
};

//Creates Vector2D with given input
Vector2D::Vector2D(double x, double y){
  X=x;
  Y=y;
};

//Gives the length of the vector
double Vector2D::getLength(){
  return sqrt(X*X+Y*Y);
};

//Gives the length squared of the vector
double Vector2D::getLengthSquared(){
  return X*X+Y*Y;
};

//Addition + Subtraction
Vector2D operator+(Vector2D a, Vector2D b ){
  return{a.X+b.X,a.Y+b.Y};
};
Vector2D operator-(Vector2D a, Vector2D b ){
  return{a.X-b.X,a.Y-b.Y};
};

//Multiplication, both sides
Vector2D operator*(double d, Vector2D v ){
  return{d*v.X,d*v.Y};
};
Vector2D operator*(Vector2D v, double d ){
```

```
  return{d*v.X,d*v.Y};
};

//Division by number
Vector2D operator/(Vector2D v, double d ){
  return{v.X/d,v.Y/d};
};

//Negate the vector
Vector2D operator-(Vector2D v){
  return{-v.X,-v.Y};
};

//Give the dot product
double dot(Vector2D a, Vector2D b){
  return{a.X*b.X+a.Y*b.Y};
};
```

Appendix C

Arduino

The Arduino script consist of a main file with several supporting files with functions to control the prototype. Only the functions that have been edited will be discussed below the other ones can be read in the report by Mainini [3].

action
It regulates the commands that are given and executes the corresponding actions associated with that command.

checkRules
It checks the rules for the new configuration and gives a Boolean (true false) feedback confirming or rejecting the new configuration.
The rules are written in the Transducer library and are:
Rule 1: The transducers do not touch the outer walls.
Rule 2 and 3: The transducers do not touch each other.
Rule 4: The carriages of a transducer do not move apart more than the safety distance.

configuration
It puts the Arduino in configuration mode and makes it possible to manually modify parameters on the board from LABVIEW. It is meant to be a back-door to have direct and full control over the tool. The implementation of this function was a bit difficult without knowing what parameters would want to be changed because the setPar function to set parameters already existed. The configuration function currently only allows to tune the motor positions however more configuration options could be added for more possibilities.

homeSet
It drives the motors to their home condition. It does this task for each motor individually which is not convenient if the transducer holders are still attached. Currently a safety rule has been implemented to make sure that to motors of one transducer do not move too far apart because of the transducer holders length causing the motor to stop the homing movement if the safety distance is met. To actually home the motors with this script it would be needed to run it several times which takes more time than should be necessary. The best thing is to rewrite the function so the motors of a transducer move at almost the same time to the homing position. For that it is useful to know that M1 and M2 are for transducer 1, M3 and M4 for transducer 2 and M5 and M6 for transducer 3.

setPar
It sets a parameter to a value. The parameter and the value should be contained in the branch of the command. The command itself should be in the form of PAR1POS. . . / PAR1ANG. . . / PAR1CT1. . . / PAR1CT2. . . for the position, angle and the positions of motor1 and motor2 of transducer. Changing the first number to 2 or 3 would be the commands for transducer 2 and 3
By ommitting the number and everything what comes behind it is possible to adjust other parameters by adding one of the 3 letter combinations below to the PAR.

ENR encoder resolution

SPL spindle length

SPP spindle pitch

CAL carriage length

SAD safety distance

PRD proximity distance

THD threshold distance

WAC wait cycle

MAP max pulse

Appendix D

LabVIEW

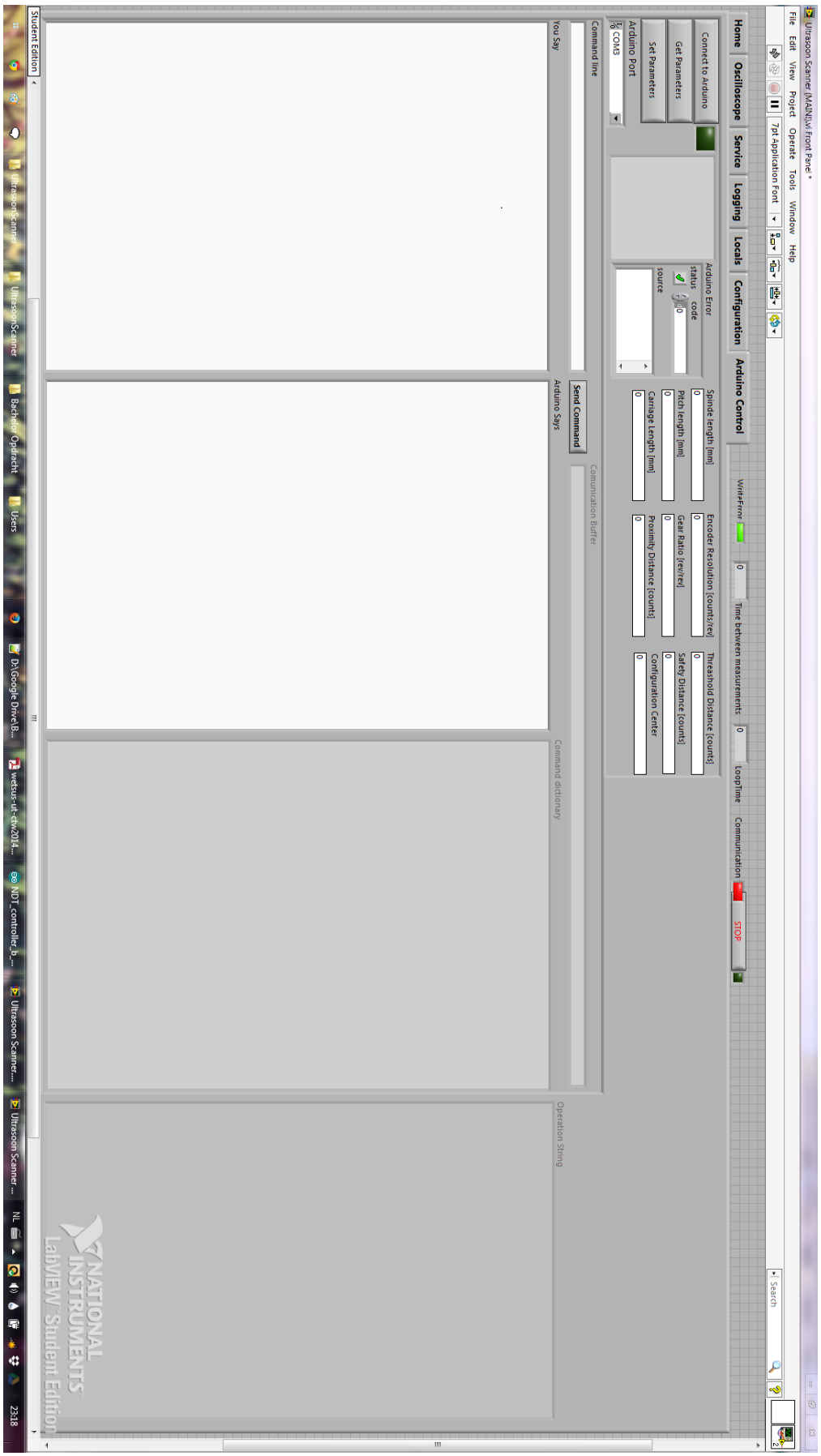


Figure D.1: Arduino Command LabVIEW UI panel.

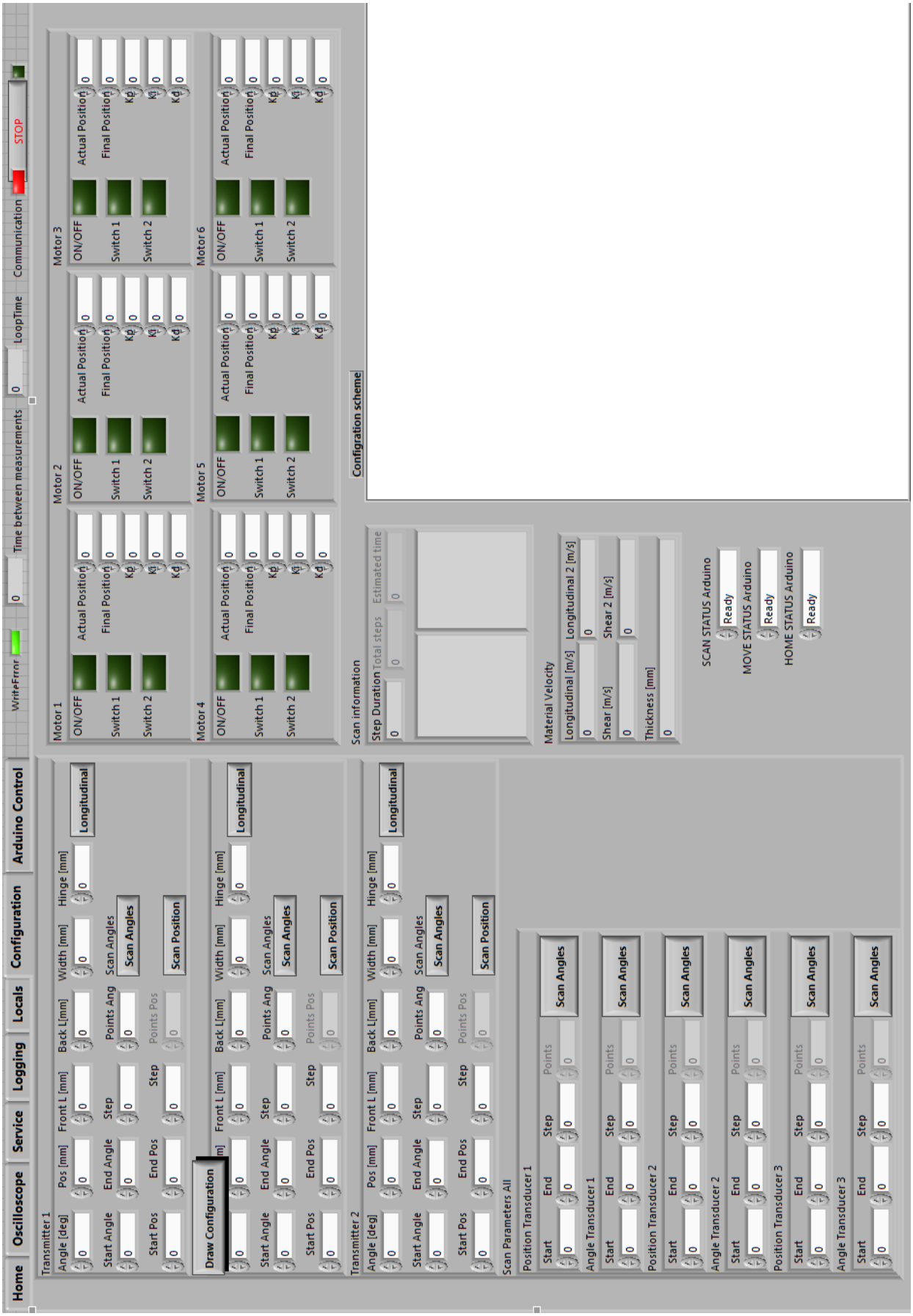


Figure D.2: Configuration LabVIEW UI panel.

Appendix E

Testing

E.1 Test1 script

```
int DI_1 = 10;
int EN_1 = 11;
int DIR_1 = 12;
int PWM_1 = 4;

void setup(){
pinMode(DI_1,OUTPUT);
pinMode(EN_1,OUTPUT);
pinMode(DIR_1,OUTPUT);
pinMode(PWM_1,OUTPUT);

digitalWrite(DI_1,LOW);
digitalWrite(EN_1,HIGH);
digitalWrite(DIR_1,LOW);

Serial.begin(9600);

Serial.println("Speed 0 to 255");

};
void loop(){
if (Serial.available()){
int speed = Serial.parseInt();
if (speed >= 0 && speed <= 255){
analogWrite(PWM_1, speed);
};
};

};
```

E.2 Test2 script

```
//Motor 1
int DI_1 = 10;
int EN_1 = 11;
int DIR_1 = 12;
int PWM_1 = 4;

//Motor 2
int DI_2 = 15;
int EN_2 = 16;
int DIR_2 = 17;
int PWM_2 = 5;
int xtime=1;
```

```
long startTime;                    // start time for stop watch
long elapsedTime;                  // elapsed time for stop watch
void setup(){

pinMode(DI_1,OUTPUT);
pinMode(EN_1,OUTPUT);
pinMode(DIR_1,OUTPUT);
pinMode(PWM_1,OUTPUT);
pinMode(DI_2,OUTPUT);
pinMode(EN_2,OUTPUT);
pinMode(DIR_2,OUTPUT);
pinMode(PWM_2,OUTPUT);

Serial.begin(19200);
while (! Serial);
Serial.println("Speed 0 to 255");
digitalWrite(DI_1,LOW);
digitalWrite(DIR_1,LOW);
digitalWrite(DI_2,LOW);
digitalWrite(DIR_2,LOW);

};

void loop(){

Serial.print('\n');
Serial.println("Motor1");
int speed1 = 'a';
while(speed1=='a'){
if (Serial.available()){
speed1=Serial.parseInt();
Serial.print("speed1=");
Serial.println(speed1);
};
};

Serial.println("Motor2");
int speed2 = 'a';
while(speed2=='a'){
if (Serial.available()){
speed2=Serial.parseInt();
Serial.print("speed2=");
Serial.println(speed2);
};
};
startTime = micros();
if (speed1 >= 0 && speed1 <= 255){
if (speed2 >= 0 && speed2 <= 255){
analogWrite(PWM_1,speed1);
analogWrite(PWM_2,speed2);
while(xtime <=100){
digitalWrite(EN_1,HIGH);
digitalWrite(EN_2,HIGH);
delay(500);
digitalWrite(EN_1,LOW);
digitalWrite(EN_2,LOW);
delay(500);
xtime++;
};
};
};

elapsedTime =  micros() - startTime;
Serial.print("elapsedTime=");
Serial.println(elapsedTime);
Serial.print('\n');
};
```

Reference

[1] Vewin, *Drinkwater statistieken*. PhD thesis, University of Twente, Enschede, The Netherlands http://www.vewin.nl/SiteCollectionDocuments/Publicaties/Vewin%20Drinkwaterstatistieken%202012%20lowres.pdf , 2012

[2] Andriejus Demčenko, *Development and analysis of non collinear wave mixing techniques for material properties evaluation using immersion ultrasonics*. PhD thesis, University of Twente, Enschede, The Netherlands, 2014

[3] L. Mainini, H.A. Visser, R. Loendersloot, R. Akkerman, *Prototype for non-collinear wave mixing: design & software implementation*. University of Twente, Enschede, The Netherlands, 2014

[4] UM0759 User manual, http://www.st.com/st-web-ui/static/active/en/resource/technical/document/user_manual/CD00242798.pdf , Last seen on 15-03-2015

[5] L9958: Low R_{DSON} SPI controlled H-Bridge, http://www.st.com/web/catalog/sense_power/FM1965/SC1039/PF246497, Datasheet: http://www.st.com/st-web-ui/static/active/en/resource/technical/document/datasheet/CD00268302.pdf, Last seen on 15-03-2015

[6] Serial Peripheral Interface Bus, http://en.wikipedia.org/wiki/Serial_Peripheral_Interface_Bus, Last seen on 17-03-2015

[7] Pololu Robotics & Electronics, http://www.pololu.com/product/2275, Last seen on 03-03-2015

[8] Arduino: Libraries, http://arduino.cc/en/Guide/Libraries, Last seen on 03-03-2015

[9] Arduino Mega 2560, http://arduino.cc/en/Main/ArduinoBoardMega2560, Last seen on 17-03-2015

[10] Improving the Beginners PID Introduction, Brett Beauregard, http://brettbeauregard.com/blog/2011/04/improving-the-beginners-pid-introduction/, Last seen on 03-03-2015

[11] Extended EEPROM library for Arduino, Thijs Elenbaas, http://thijs.elenbaas.net/2012/07/extended-eeprom-library-for-arduino/, Last seen on 03-03-2015

[12] Disadvantages of pointers and errors while using pointers, http://www.moreprocess.com/c-and-c/disadvantages-of-pointers-and-errors-while-using-pointers, Last seen on 04-03-2015

[13] Rotary encoder, http://en.wikipedia.org/wiki/Rotary_encoder#Incremental_rotary_encoder, Last seen on 10-03-2015

[14] analogRead() http://arduino.cc/en/Reference/analogRead Last seen on 18-03-2015

[15] H-bridge, http://en.wikipedia.org/wiki/H_bridge, Last seen on 10-03-2015

[16] The motor driver that is used for this prototype., http://nl.aliexpress.com/item/Free-shipping-New-240W-H-bridge-Motor-Driver-Board-Motor-Controller-SPI-for-Arduino-Smart-Car/984874066.html?recommendVersion=1, Last seen on 10-03-2015

[17] In-system programming, http://en.wikipedia.org/wiki/In-system_programming, Last seen on 10-03-2015