UNIVERSITY OF OSLO
Department of Informatics

Flexinol as Actuator for a Humanoid Finger - Possibilities and Challenges

Master thesis (60 pt)

Øyvind Fjellang Sæther

01.11.2008

Abstract

Robots become more and more common in our every day lives as technology develops. Robots are normally actuated by pneumatics, hydraulics or servo motors. These technologies are mature and widely used, but other less commonly used actuators are also available. Among these we find the artificial muscle fiber Flexinol which belongs to a class of materials known as Shape Memory Alloys.

This thesis aims to implement the artificial muscle fiber Flexinol as actuator for a humanoid finger. The first part of the thesis focuses on testing of single Flexinol wires to determine in what degree these are suitable for long term use as actuators. A test frame is built to investigate contraction speed, force and displacement for wires in different setups. Among these are tests with a small dead weight, a large dead weight, an antagonistic setup and a setup with a spring working as a passive antagonistic force.

The second part of the thesis makes use of Flexinol as actuator when designing and prototyping a humanoid finger. The human finger is used as inspiration in this part, applying tendons and muscles in a human-like way. The finger is designed with CAD-software and then printed in plastic. It is then assembled with tendons and actuated with three Flexinol wires. Finally, an attempt to control the humanoid finger is done.

Specially designed software and hardware is developed through the thesis to implement working experiments. Software for both a laboratory computer and a microcontroller is written to control the system and to collect sensory data respectively.

Preface

This thesis is part of my Master Degree at the University of Oslo, Department of Informatics. The thesis was carried out during 2008 in the research group Robotics and Intelligent Systems (ROBIN). First of all I want to thank my supervisor, Associate Professor Mats Høvin, for creative input, for motivating me to be creative and for valuable feedback both during my practical work and during writing.

I also want to thank the following (sorted by topic): Vegard Friis Ruud, Charlotte Kristiansen, Marie Klemsdahl Eklund and Marte Lødemel Henriksen for fruitful discussions regarding the human anatomy, Kjetil Stiansen for help regarding practical and theoretical electronics and Andreas Gimmestad for his linguistic abilities.

Thanks also go to friends and family for showing interest, specially to my father Dag Henning Sæther for guidance both during my practical work and during writing.

Øyvind Fjellang Sæther
November 2008

Contents

Chapter 1

Introduction

Over the last decades, robotic technology has entered more and more areas in industry and the every day lives of humans. Robots are programmed to do complex tasks which often involve some kind of interaction with the environment. In industrial applications, a robot may be only an arm that performs a special task, or it could also be equipped with wheels that would allow it to move freely in a local environment. In such cases, the hand of the robot would often be a tool that is specially designed for a given task.

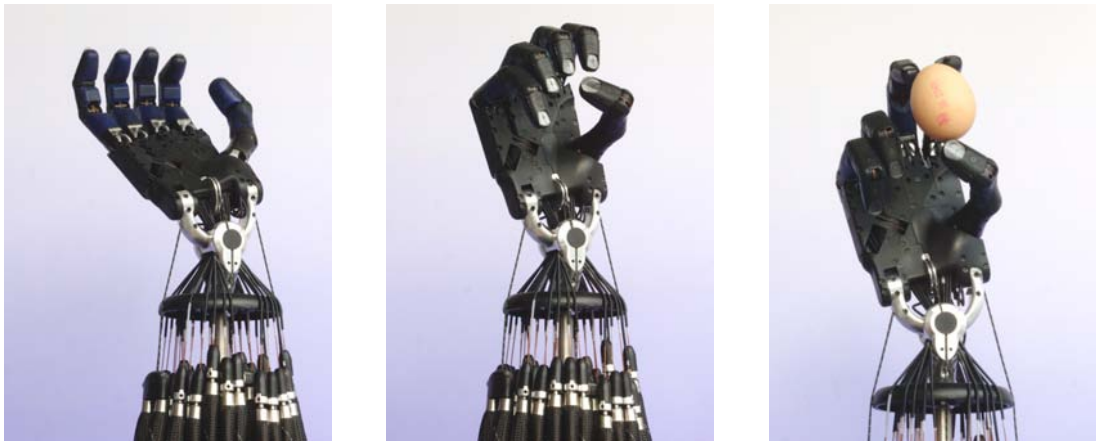
However, robots that are designed to interact with humans in a physical way need human-like hands. Of course a robot could interact with humans using a stick or some other tool, but if the contact is supposed to be interpreted as human-like, hands are necessary. In health care, a robot could be a valuable assistant to a human worker, performing heavy lifts and other routine work that does not need to be done by humans alone. In such settings, a robot as adaptable and flexible as humans would be preferred, but this is still an Utopian setting. The physical adaptability of humans - our ability to use different tools to perform tasks, makes us superior to other animals. Our hands allow us to perform trivial tasks such as gripping around an unknown object while blindfolded, or to hammer in a nail. Of course, these examples depend on a well functioning regulation mechanism - our brain.

1.1 Humanoid Hands

As already mentioned, a robot that is designed to interact with humans in a human-like fashion will often need hands. Robots that perform tasks in a human-like way are referred to as *humanoid*. Analog is a robot hand called humanoid when its design and motion is based on the principles of the human hand.

Principally, there seems to be two main directions in todays research in the field of humanoid hands. The first direction has its main focus on the development of artificial hands for prosthetic applications [1, 2, 3]. These works often have criterias such as light weight, easy control, anatomical design, and in some cases, esthetics. The other branch of researchers focus more on robotic applications such as humanoid robots [4, 5, 6, 7, 8, 9, 10]. These hands have different design criterias depending on the target robot platform, are in general more complex, and possess more advanced control mechanisms than the prosthesis. Moreover, these two branches also seem to have a lot in common, such as the never ending need for adaptability. The ability of the human hand to adapt its grasp to unknown shapes and surfaces is wanted in as good as all hand designs, but is not an easy task to resolve.

One state of the art humanoid robot hand is the Shadow Hand C5 [7] from The Shadow Robot Company (www.shadowrobot.com). This is a commercially developed hand and as a result, no scientific articles have been published. However, an earlier version of the hand is under research at the Bielefeld University [11]. Figure 1.1 shows three pictures of the hand in different positions. In the lower part of figure 1.1a, the actuators of the hand are shown. These are air muscles that provide light weight actuation from the forearm of the application. The hand also has a grid of touch sensors on each fingertip to provide good grasping feedback. A drawback with this design is the physical space needed for the air muscles. As the pictures clearly show, a rather large forearm is needed to fit all the muscles.

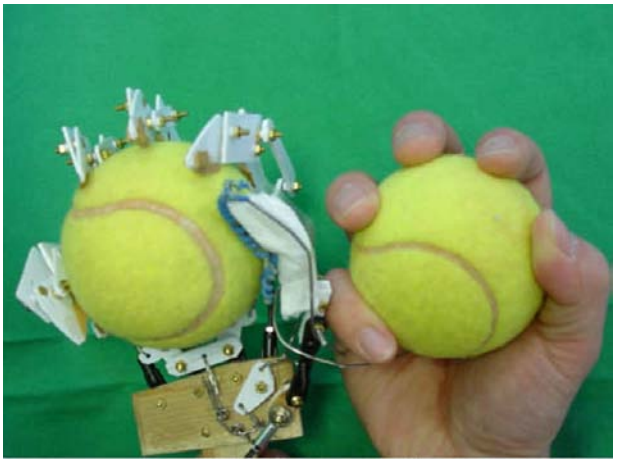


(a) Open hand (b) Half fist (c) Grasping an egg

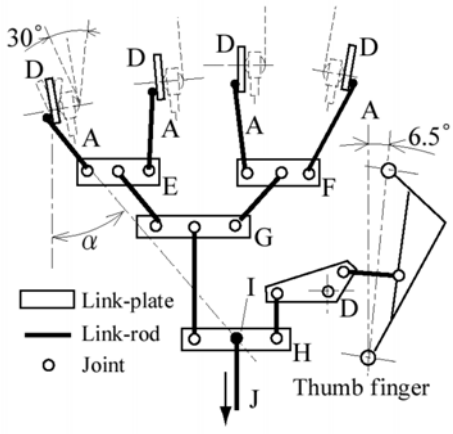
Figure 1.1: *Shadow Hand C5 [7]. The hand is actuated by 40 pneumatic artificial muscles and has 24 degrees of freedom*

1.2 Robot Hand Actuation

By looking at different humanoid robot hands, it soon becomes clear that this is a field under heavy research. No standardized solutions have yet been pointed out regarding materials, actuators or design. One design criteria that seems to be commonly used is the kinematic sketch of the hand, which often is very similar to that of the human hand. However, the way that the hand is actuated varies greatly between projects. Some hands are so-called underactuated hands [1, 3, 6, 12]. This branch of hands have fewer actuators than degrees of freedom. A good example of such an underactuated hand is called the *TUAT/Karlsruhe Humanoid Hand* and can be found in [6, 13, 14]. The hand can be seen in figure 1.2a, and its special link mechanism is depicted in figure 1.2b. The mechanism consists of a number of link plates hierarchically connected with rods. When the actuator is used, the link plates will align such that the hand grasps around whatever object present in the hand. The TUAT/Karlsruhe Humanoid Hand is an example of the similarities between prosthetic and robotic hands, as this hand is designed to be suitable both for a humanoid service robot and for prosthetic purposes. The simple actuation of the hand is its biggest advantage in a prosthetic setting. However, this simplicity also narrows the number of robotic applications where it fits.



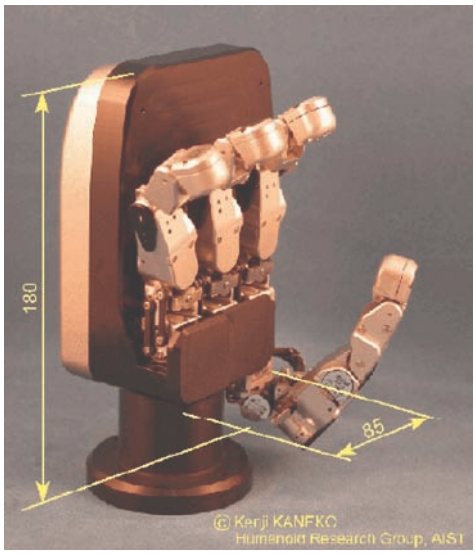
(a) A spherical grasp around a tennis ball



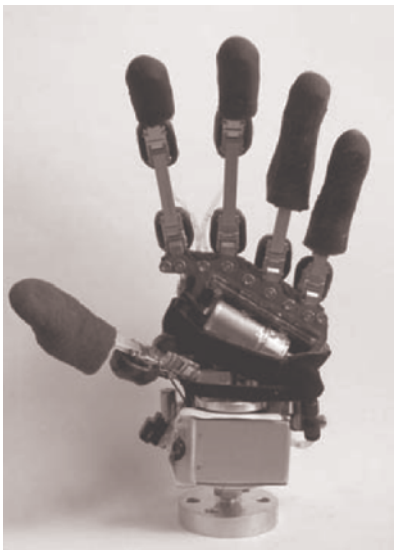
(b) The hand is underactuated, using only one actuator. The link mechanism distributes the grasping force over the different fingers

Figure 1.2: *The TUAT/Karlsruhe Humanoid Hand [6]*

Underactuated hands depend on passive mechanisms to be able to grasp around objects. In contrast to this branch of hands are hands that have their actuators incorporated into each finger joint. However, many hybrids are also available such as the two hands depicted in figure 1.3. Both of these hands are rather large designs caused by the need for integrated motors and pumps.



(a) Multi-fingered Hand for Life-size Humanoid Robots. The hand has 13 active joints and 4 passive joints, actuated with integrated servo motors [4]



(b) Anthropomorphic Hand for a Mobile Assistive Robot. The hand has 8 active joints and 3 passive joints, actuated with miniature hydraulics [8]

Figure 1.3: *State of the art humanoid hands*

1.3 Flexinol - Artificial Muscle Fibers

A group of actuators not so commonly seen are artificial muscles fibers, such as Flexinol. Flexinol looks like a steel wire and has the ability to contract when a current is passed through it. Compared to its size, Flexinol is able to exert rather large forces and can be cut to any length. As the name *artificial muscle fiber* states, Flexinol is very similar to human muscles regarding function. Although human muscles consist of many muscle fibers, a muscle seen as a whole is very much like a Flexinol fiber. When the muscle is contracted, it is shortened and thickened, just like a Flexinol wire.

Flexinol represents a kind of actuator that at first glance seems to fit perfect as actuator for a humanoid robot hand. It is small in size, very powerful and a commercial product, available at a reasonable price. Its biggest advantage is its size, that principally will allow very many Flexinol wires to be fitted inside for example a robot forehand. Although Flexinol has many clear advantages, very little research interest has been shown when it comes to using it as a robot actuator. One attempt to use Flexinol as actuator in a robot hand is proposed in [15]. The authors present a working hand, but do not mention anything regarding regulation or long term properties in the rather brief paper.

Other articles about general use of Flexinol have also been found, like [16, 17, 18, 19, 20, 21, 22]. Common for most of these articles is that difficulties are uncovered, but not often solved. Some of the reported difficulties are hysteretic behaviour, high power dissipation and low strain rates. No information has been found regarding the long term properties of Flexinol. This is highly relevant information if Flexinol is to be used in a robotic application.

All in all, there are several questions that need to be answered in order to determine whether Flexinol is suitable as a robot hand actuator:

- Does Flexinol stand the long time use as a robot hand actuator?
  - Are the properties of Flexinol changing over time?
- Is it possible to regulate the contraction of Flexinol when used as actuator for a humanoid finger?

- Is it possible to improve the strain rate of Flexinol?
- Is it possible to use multiple Flexinol wires in parallel?

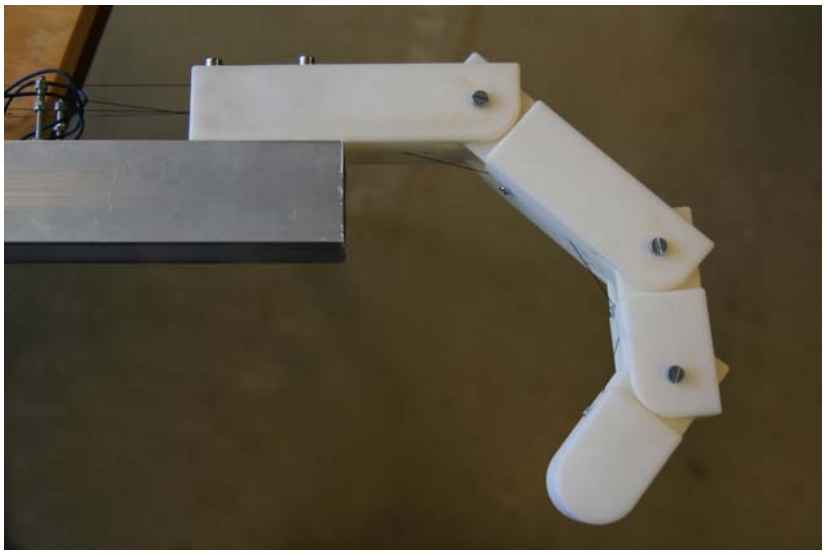
1.4 Thesis Overview

This thesis aims to answer the questions stated above. Chapter 2 contains background information about Flexinol and other actuator technologies. A brief overview of the anatomy of the human hand is given in addition to actuators and feedback sensors. Chapter 3 contains information about two of the tools used in the practical work of the thesis and chapter 4 describes the methods that were developed in order to answer the above questions. Chapter 5 contains an evaluation of the proposed methods and a discussion around regulation is found in chapter 6. Suggestions to future work is given in chapter 7 and finally, a conclusion is given in chapter 8. Following is a list over all the practical work completed during this thesis.

- Long term testing of Flexinol
  - Building test frame
  - Molding weights
  - Design of amplifier circuit for force measurements
  - Calibration of force and displacment transducers
  - Design of driver circuit for Flexinol wires
  - Programming of measurement software
    - Program for controlling Flexinol wires and collecting data
    - Web application for remote surveillance and data browsing
    - Matlab scripts for analyzing data
- PWM-control of Flexinol wire
  - Design of microcontroller circuit with RS232 remote interface
  - Programming microcontroller
    - Command interpreter for RS232 commands
    - Calibration algorithm
    - Flexinol regulation algorithm
  - Programming computer interface
    - Text mode for debugging purposes
    - Command mode for easy operation
    - Continuous mode for data visualization
- Humanoid finger application
  - 3D design of a humanoid finger with a torque free tendon routing scheme
  - Printing in ABS-plastic and assembly of the finger
  - Design, printing and assembly of radial displacement transducers
  - Design, printing, assembly and calibration of force sensor brackets
  - Design of micorcontroller circuit with sensor input and Flexinol driver
  - Expansion of command set for microcontroller to include control and feedback of three wires
  - Programming computer interface
    - Reuse of computer interface from PWM-testing
    - Interface for the Microsoft Robotics Studio framework
- Regulation methods
  - Proportional regulation for PWM-control and finger joints
  - Manual regulation of one wire using a high frequency pwm motor driver

1.5 Short Conclusion

This thesis shows that the use of Flexinol as actuator for a robotic finger is feasible. Flexinol has many disadvantages that have to be overcome such as a limited life, hysteresis behaviour, limited strain rate and degeneration of wires but it also has advantages. Flexinol exerts very large forces compared to its own size and needs very little physical space in an application. Figure 1.4 shows a humanoid finger actuated with Flexinol wires and a test frame for testing Flexinol wires. Both products were developed during this thesis.



(a) Humanoid finger developed in this thesis. The finger is actuated with three Flexinol artificial muscle fibers and controlled with a microcontroller

(b) A test frame was built to investigate the long term properties of Flexinol

Figure 1.4: *Two products of the thesis*

Chapter 2

Background

In this chapter, the background theory for the thesis is presented. First, the human hand is briefly presented to later be used as motivation for the development of a humanoid finger. Secondly, available traditional actuators and actuators based on intelligent materials are presented and compared. Finally, different feedback sensors suitable for displacement- and force measurements in robotic applications are discussed.

2.1 Anatomy of The Human Hand

In this section some basic anatomical principles of the human hand are presented. The human hand consists of 4 fingers and a thumb and is the main organ for physical interaction with the environment surrounding the human body. A schematic of the bones in the human hand can be seen in figure 2.1.

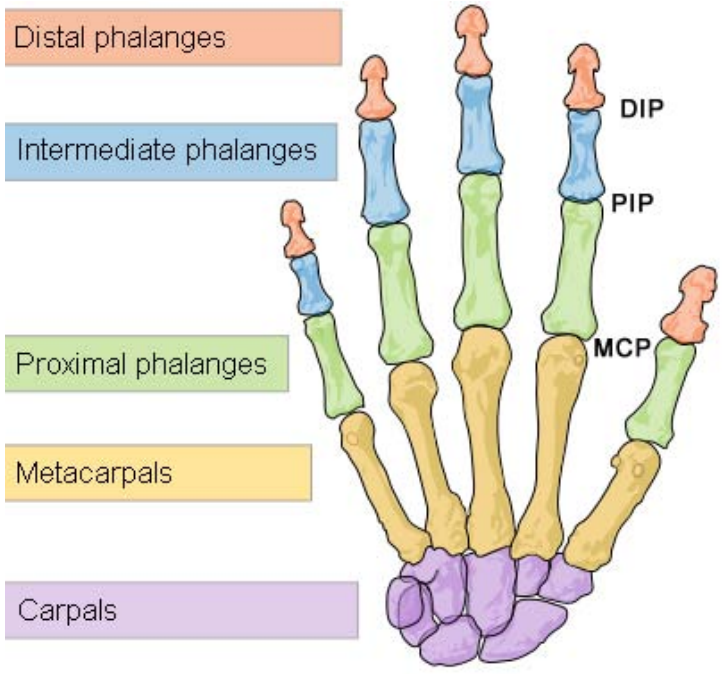


Figure 2.1: *Schematic drawing of a human hand. Carpals and Metacarpals form the wrist and palm of the hand while Proximal, Intermediate and Distal phalanges form the fingers*

2.1.1 Skeleton

In figure 2.1, the carpals are known as the wrist and the metacarpals as the palm. Proximal, intermediate and distal phalanges form the three segments of the finger. The anatomy of the thumb and the wrist are

left out of this thesis as they are complex topics that are not needed for the presented work.

Metacarpal Phalanx

The metacarpal phalanx (yellow) is connected to the first finger segment (proximal phalanx) with a joint called the metacarpophalangeal joint (MCP-joint). The MCP-joint is able to perform to types of movement, flexion/extension and abduction/adduction. Flexion in this case means bending the finger while extension means extending the finger. Abduction denotes the sideways motion of the finger away from the midline of the hand. The opposite movement, adduction, means moving the finger back against the midline of the hand.

Proximal Phalanx

The proximal phalanx (green) is connected to the metacarpal phalanx through the MCP-joint. On the other side of the finger segment it is connected to the second finger segment (intermediate phalanx) with a joint called the proximal interphalangeal joint (PIP-joint). The PIP-joint only has one axis of motion and is therefore called a hinge joint. Flexion and extension of the PIP-joint means bending and stretching the first finger joint.

Intermediate and Distal Phalanx

The intermediate (blue) and distal (red) phalanges are the middle and outer segments of the finger, respectively. They are connected with the distal interphalangeal joint (DIP-joint) which is a hinge joint like the PIP-joint.

2.1.2 Tendons and Muscles

The actuating mechanism of the human body is represented by muscles. However, the forces needed to grip heavy objects are so large that the muscles needed cannot be fitted inside the human hand. Instead, the muscles are placed in the forearm and the forces exerted by the muscles are transferred to the hand using tendons. This type of muscle placement is called *extrinsic* [23]. The efficiency of a human muscle is reported to be between 14% and 27% in the context of rowing and cycling [24].

Human Skeletal Muscles

A skeletal muscle is fastened to a bone in the human body to cause movement and force exertion [24]. Thus can it be seen as an actuator for the body. The muscle itself is a bundle of single, parallel muscle fibers that are built up from muscle cells. A muscle cell consists of plates that are moved relative to each other to generate motion. The cells cause the muscle fibers and the muscle to contract when it receives a neural pulse, called an action potential. The frequency of the neural signal reception decides the contraction rate and force of the muscle. The neural control signals for human skeletal muscle movement can therefore be called pulse frequency modulated.

Finger Movement

The tendons and muscles responsible for flexion and extension of the finger are depicted in figure 2.2. The most important parts for this thesis (marked with red) are the M. lumbricalis that flexes the MCP-joint, the M. flexor digitorum superficialis, Tendo that flexes the PIP-joint, the M. flexor digitorum profundus, Tendo that flexes the DIP-joint, and finally the M. extensor indicis, Tendo that extends the finger. In figure 2.3, the insertion points of all the tendons in the hand can be observed.

2.1.3 Robotic Approach to the Human Hand

There are several aspects of the human hand that makes it very hard to imitate in a robotic application. First of all, the human hand has a very complex kinematic model. The fingers alone possess 21 degrees of freedom (DoF) according to [26]. In addition to the finger, movement of the palm includes 6 DoF, 27 for the whole hand. The movement of the palm itself is rarely seen in robotic hands, a solid palm is often used instead. In addition to the high degree of freedom, the finger is also very sensitive to external

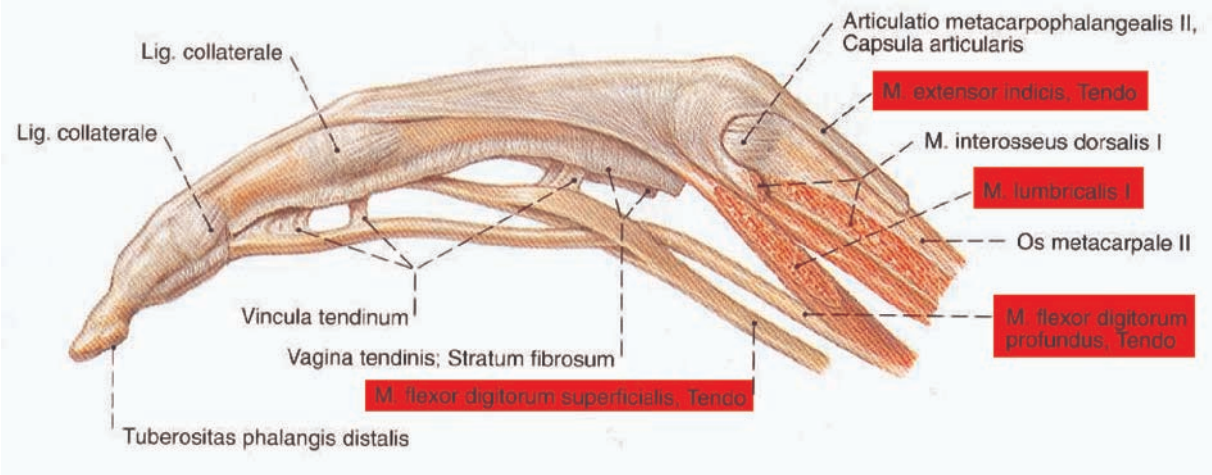


Figure 2.2: *Tendon routing in the human finger [25]. The most important tendons and muscles are marked with red*

input. To imitate all the nerves on the surface of the finger is very complicated. Normally, a number of touch sensors is seen instead.

2.2 Traditional Actuators

This section focuses on traditional actuators suitable for robotic applications.

2.2.1 Hydraulics

A widely used form of general actuation is hydraulic systems. A hydraulic system, which is depicted in figure 2.4, typically consists of a pump, a reservoir, a valve and an actuator. The hydraulic pump is responsible for creating pressure by forcing liquid (typically oil) from the reservoir into the hydraulic system. By opening the valve, the pressurized liquid flows into the actuator, normally a hydraulic cylinder, creating movement. Depending on the physical characteristics of the pump and the actuator, the actuator exerts a force as illustrated in figure 2.5. In this example, the input energy is supplied by the pump, whereas the output force is the actuation of the cylinder.

Hydraulic systems have been used commercially since the industrial revolution wherever great forces have been needed. Traditionally, these systems have been quite large, not suitable for robotic applications. However, with development in technology, small systems including a micropump, valves and actuators now may be integrated into the palm of a robotic hand [28].

2.2.2 Pneumatics

In pneumatics, a pressurized gas is used to create actuation, typically by filling a cylinder with air. In normal pneumatic systems a motion speed of up to 1m/s is obtained, but in high speed pneumatic systems, speeds of 15m/s may be reached [29]. Figure 2.6 shows a pneumatic cylinder half full of air. Pneumatic cylinders can be either two way, as shown in the figure, or one way. A two way cylinder is actuated actively both ways with pressurized air in front of or behind the piston. A one way cylinder is returned passively by a spring mechanism. By controlling the air flowing into the cylinder, the piston can be actuated to either end of the cylinder. Pneumatic systems can be very simple, containing only a compressor, valves and a cylinder. However, the efficiency of the compressor is low, causing an unwanted loss of power in for example mobile applications.

Pneumatic Artificial Muscles (PAMs)

PAMs [30] are special variants of pneumatic actuators. Figure 2.7 depicts a PAM in three different states. The muscle consists of a flexible membrane which can be filled with air. When the muscle is empty (a),

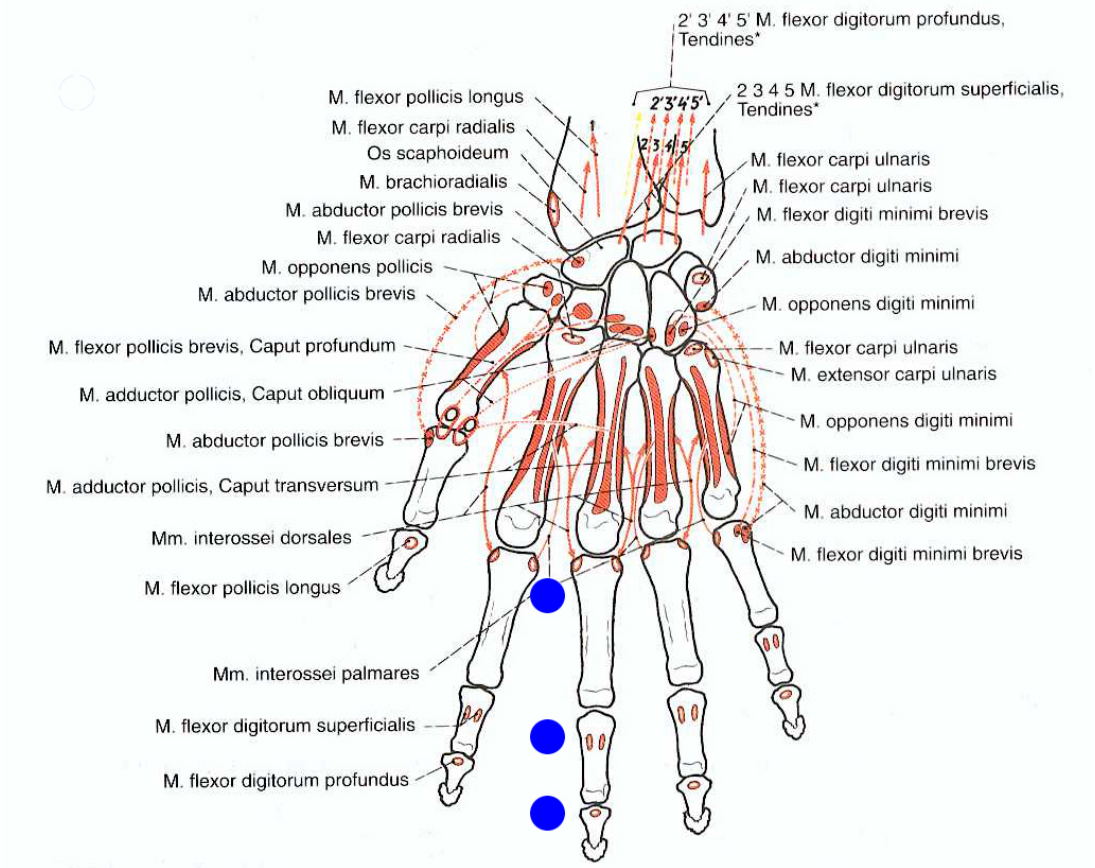


Figure 2.3: *Insertion points on the human hand [25]. The blue dots show the three points essential for finger flexion*

the muscle is at its maximum length. When air is filled into the muscle, a pressure builds up causing the volume to increase. As figure (b) shows, this decreases the length of the muscle. By further increase of the pressure the muscle reaches its maximum volume in (c). At this point the contraction of the muscle can be about 30% of its initial length. Many variants of PAMs are described in [30], but common for all is that they are light weighted. However, the use of PAMs has not been very common due to problems regarding short life of the membrane caused by mechanical stress and friction forces. This seems to have changed over the last decade as for example the Shadow Hand C5 [7] is actuated by a high number of PAMs. Accurate control of the muscle has been reported to be non-trivial.

2.2.3 Servo Motors

A servo motor (figure 2.8) is a closed system which gets input information and performs a motion accordingly. The input is typically a position or a velocity, but may also be information from a force sensor or any other feedback sensor. Servo motors come in a variety of sizes, making them suitable for many different applications with different design considerations such as high force, small size or high speed.

RC-Servo Motors

A subgroup of servo motors are servo motors for radio controlled (RC) applications. These motors typically allow about 180-210 degrees of angular motion and are controlled with a pulse width modulated (PWM) signal with fixed frequency and varying duty cycle. A controller for such a signal can easily be built in hardware or software. RC-servos come in different sizes and types, an example of a servo for robotic applications is given in figure 2.8. This servo has an idle wheel on the opposite side of the drive wheel, making it suitable for stable assembly inside a joint. From leading manufacturer Hitec RCD (www.hitecrcd.com), the smallest available RC-servo (HS-35HD) [31] is $1.85cm \times 0.76cm \times 1.54cm$ and

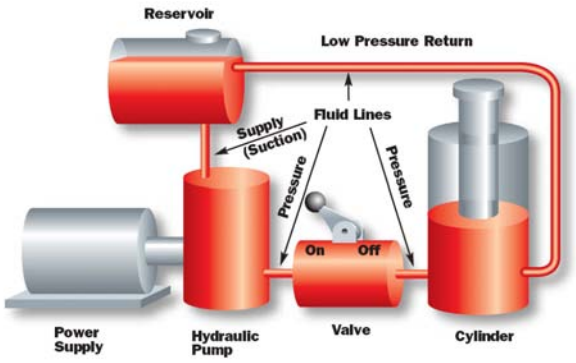


Figure 2.4: *Typical components in a hydraulic system [27]. A cylinder is a typical actuator in such systems*

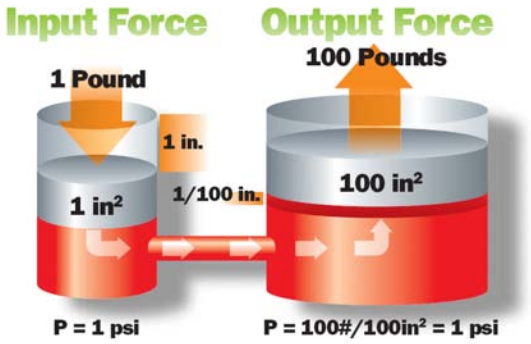


Figure 2.5: *Force multiplier in a hydraulic system [27] is used to generate large forces*

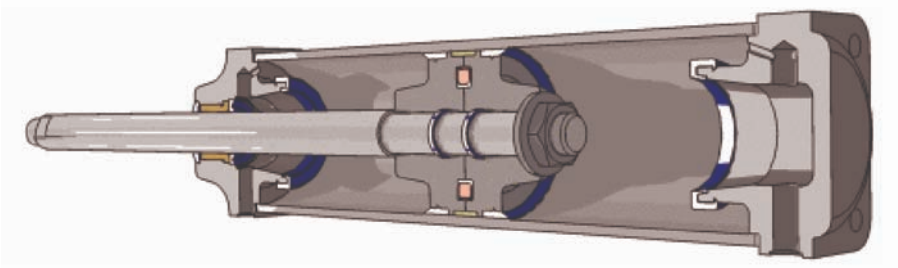


Figure 2.6: *Two way pneumatic cylinder. Air can be pumped in on either side of the piston, causing motion*

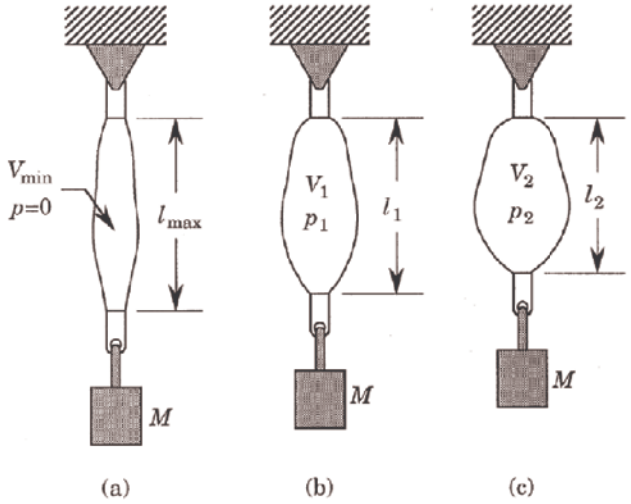


Figure 2.7: *Pneumatic artificial muscle [30]. When air is pumped into the membrane, it is shortened, causing motion*

has a torque of 10Ncm. The biggest servo (HS-815BB) is $6.58cm \times 3.00cm \times 5.74cm$ and has a torque of 242Ncm.

The biggest advantage of servo motors is their very easy control. Typically, a RC-servo takes a PWM-signal with a period of 20ms (50Hz). With a duty cycle of $\frac{1.5ms}{20ms}$ the servo is in its center position. By increasing the duty cycle to $\frac{2ms}{20ms}$ or decreasing it to $\frac{1ms}{20ms}$ the servo is moving respectively $+90°$ or $-90°$.

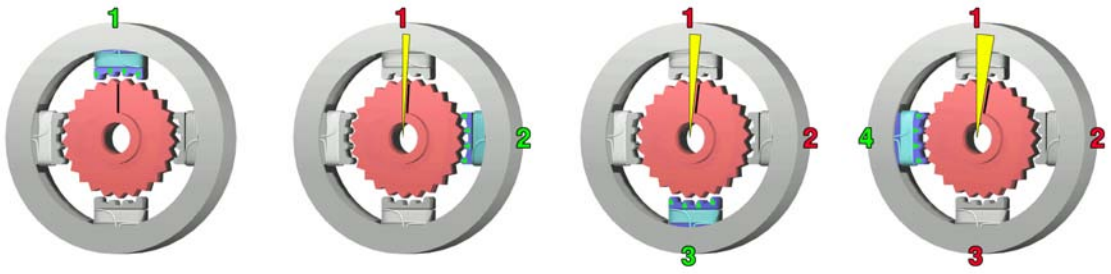
Figure 2.8: *Robot servo motor from Hitec [32]. The servo has an idle wheel for assembly inside for example a robot joint*

2.2.4 Stepper Motors

Stepper motors create angular motion, just as servo motors do. As the servo motor uses feedback to control its position or speed, the stepper motor has no need for this. To eliminate the need for feedback, the stepper motor divides one turn of the motor into a fixed number of steps, varying from model to model. The simplest form of a stepper motor can be seen in figure 2.9. The drive shaft is fastened in the toothed wheel which is surrounded by four electromagnets with toothed surfaces. In step 1, the top magnet is turned on, making the teeth of the wheel align with this magnets teeth. The right magnet is positioned with an angular offset from the top magnet with $\frac{1}{4}$ of the angle between two teeth on the wheel. When turning the top magnet off and the right magnet on (step 2), the teeth of the wheel will align with the teeth of the right magnet, causing a motion with $\frac{1}{4}$ of the tooth angle. Step 3 involves turning the right magnet off and the bottom magnet on, causing another step of motion. In step 4, the bottom magnet is turned off and the left magnet is turned on. The wheel has now turned $\frac{3}{4}$ of the tooth angle and the next time step 1 is enabled, the wheel has moved by 1 tooth angle. In this way, the

motion of the wheel can be controlled very freely. Other variants of step control is more common than this example, involving more than one magnet turned on at a time.

Stepper motors can produce very high torques at low speeds. However, at higher speeds the motor loses some of its torque. At lower speeds, the stepping of the motor can cause mechanical vibrations which in some applications are unwanted. There is also a possibility that the motor can lose steps when heavily loaded. This will cause the controller to believe that the motor has moved one step when it in reality may have stood still. To prevent this, some models include an angular feedback from the motor shaft.



(a) Step 1 (b) Step 2 (c) Step 3 (d) Step 4

Figure 2.9: *Fundamental operation of a stepper motor. After four steps, the drive shaft has turned one tooth clockwise*

2.2.5 Electric Solenoids

Electric solenoid actuators are based on magnetic forces. An example of a commercial solenoid actuator can be seen in figure 2.10. A coil is situated around the stem of the actuator. When alternating current is passed through the coil, a magnetic field builds up, and the stem is attracted towards the center of the coil. When the current stops, the stem is free and can be moved back either by active force or by a return spring.

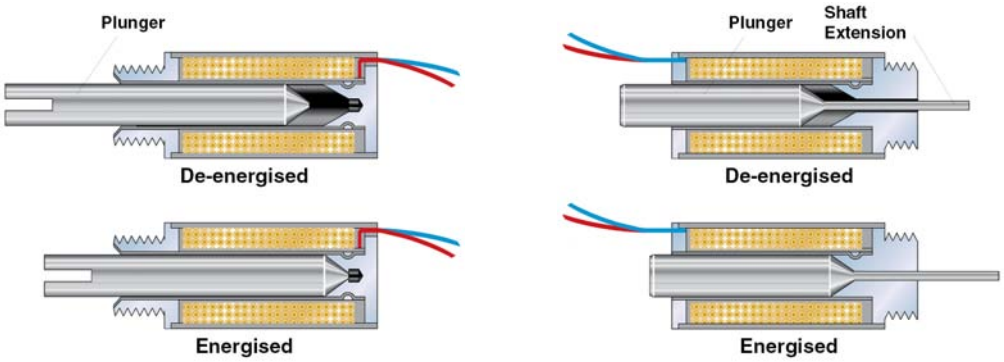
An advantage of solenoid actuators in comparison to pneumatic and hydraulic actuators is easy installation and the fact that no reservoir of fluid or air is needed. A disadvantage of solenoids is that they only have two positions when actuated, either on or off. As the stem is pulled into center of the coil, the force increases, leading to a varying force exertion during the actuation period. By examining datasheets [33, 34] from manufacturers Ledex (`www.ledex.com`) and Dialight BLP (`www.blpcomp.com`) it seems that the maximum force of available actuators is about 650N. However, a solenoid that exerts such a large force has the disadvantage that it weights over 2.2kg. The speed of the solenoid actuator depends greatly on the force it is set to exert. When unloaded, an actuation can be performed in less than 50ms [33, 34] in most cases. When operated, heat is dissipated in the solenoid which modifies the characteristics of it. Typically, the speed of the actuator stroke and the maximum force is at their maximum when the duty cycle of the solenoid is low, meaning the actuator gets to cool down between each actuation.

2.3 Intelligent Materials used as Actuators

In this section, several different actuators, based on intelligent materials, are presented. Intelligent materials used as actuators are subject for research and not yet commonly seen in real life applications. Common for most of these technologies is that they often have at least one big drawback, making them non-trivial in use. This section gives a presentation of different actuators and references to research performed in the area.

2.3.1 Shape Memory Alloys in General

Shape memory alloys (SMAs) are a group of alloys that possesses the shape memory effect (SME) [19]. This section focuses on the nickel-titanium (NiTi) alloy, which appears to be the most commonly used alloy. The shape memory effect is a temperature dependent transformation between to phases, *martensite*



(a) Push type (b) Pull type

Figure 2.10: *Ledex ® Electric Solenoid Actuator [33] Operation. Magnetic force is used to drive the stem into or out of the housing*

and *austenite*, which is illustrated in figure 2.11. The martensite phase is also known as the cold phase of the material, normally occurring at room temperature. Although this phase may appear as the original phase of the material, it is really the phase in which the material is deformed. This is according to the internal crystalline structure of the material, which at this point is stretched (figure 2.11c). By applying heat to the material (figure 2.12a), a transformation from martensite to austenite starts at the temperature A_{start}. Further heating of the material leads to a straightening of the crystalline structure until the temperature A_{finish} is reached. The material is now in its austenite phase, with its crystalline structure in its original state (figure 2.11a). When cooling the material, the transformation from austenite back to the martensite phase starts at temperature M_{start} and ends at temperature M_{finish} (figure 2.11b). Cooling of the material alone does not make the material go back to its martensite shape, stress also has to be applied to the material to stretch the internal crystalline structure (figure 2.11c). Figure 2.12b shows the same transformation curve as figure 2.12a, but also has stress as a parameter. As can be seen, applying stress to the SMA-material leads to a shifting of the transformation curve. Higher stress means that a higher temperature is needed to reach the austenite phase, and vice versa is the martensite phase reached at a higher temperature than when no stress is applied. A hysteresis behaviour can also be observed between the heating and cooling phase in both cases. More on hysteretic behaviour can be found in [16].

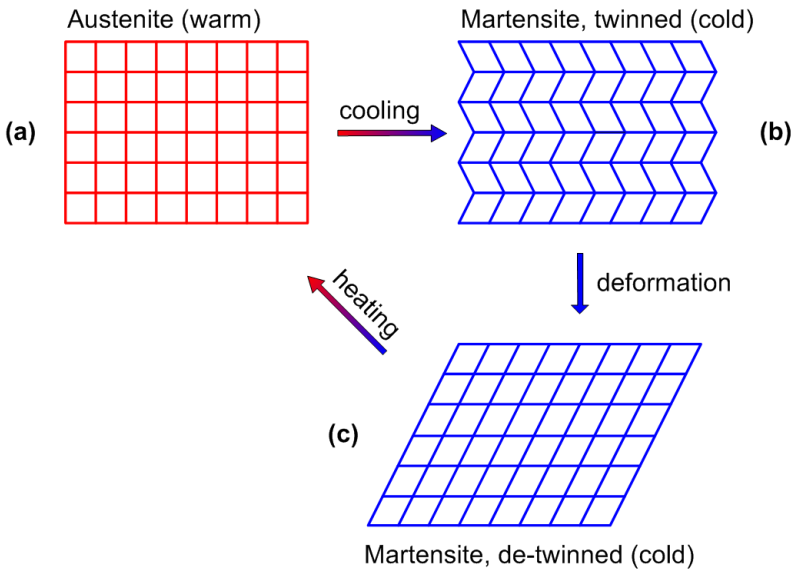
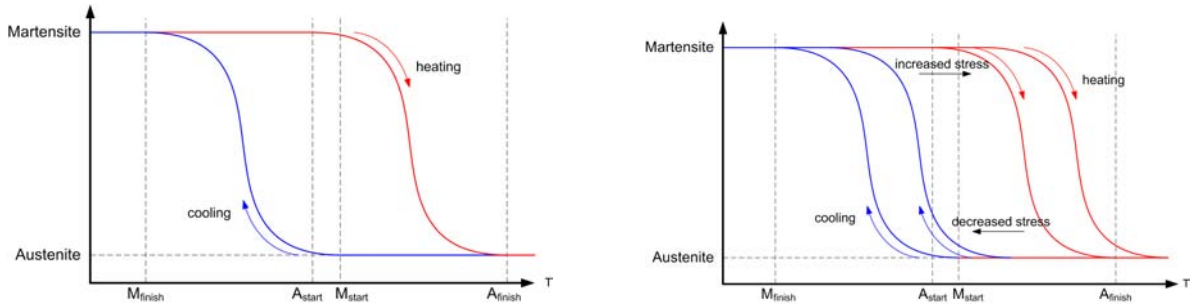


Figure 2.11: *Cycle of the Shape Memory Effect [35]. The grid symbolizes the atomic crystalline structure of the material*



(a) Temperature hysteresis

(b) Temperature hysteresis and stress

Figure 2.12: *Transformation curve of shape memory alloys [36, 37]*

2.3.2 Flexinol

Flexinol is the trade name of a SMA NiTi actuator wire, produced by Dynalloy, Inc. (www.dynalloy.com). Actuator wires are often referred to as artificial muscle fibers because of their similarity to anatomical muscle fibers.

Diameter [mm]	Res [Ω/cm]	Force [N]	Current [mA]	T_{off} (70°C)	T_{off} (90°C)	Stretch [N]
0.0254	17.72	0.07	20	0.1	0.06	0.07
0.0381	8.27	0.17	30	0.25	0.09	-
0.0508	4.72	0.34	50	0.3	0.1	0.34
0.0762	1.97	0.78	100	0.5	0.2	0.78
0.1016	1.18	1.47	180	0.8	0.4	1.47
0.127	0.71	2.26	250	1.6	0.9	2.26
0.1524	0.51	3.24	400	2	1.2	3.24
0.2032	0.31	5.79	610	3.5	2.2	5.79
0.254	0.20	9.12	1000	5.5	3.5	9.12
0.3048	0.13	12.26	1750	8	6	-
0.381	0.08	19.61	2750	13	10	-
0.508	0.06	34.93	4000	17	14	-

Table 2.1: *Technical and electrical characteristics of Flexinol artificial muscle fibers given by the manufacturer [37]*

Physical Properties

Flexinol comes in many different diameters, the first column of table 2.1 shows all available sizes. Two versions of Flexinol with different transformation curves are available, one with $A_{start} = 70°C$ and another with $A_{start} = 90°C$. The first wire has the advantage that it needs a smaller increase in temperature to contract. This is reflected by a lower power consumption. However, the lower temperature of the wire makes the relative temperature difference between the wire and the surrounding environment smaller. This again leads to a slower cooling of the wire below the M_{start} temperature (by a factor of 1.3-3.0 according to [37]).

According to the technical specifications for Flexinol [37], a contraction of ca 4.5% can be expected upon heating of the wires. In figure 2.13, the transformation curve for Flexinol ($A_{finish} = 90°C$) is depicted. When heated, the martensite to austenite transformation (contraction) begins around 70°C (A_{start}) and ends around 110°C (A_{finish}). When cooled, the austenite to martensite transformation starts at around 100°C (M_{start}) and ends around 40°C (M_{finish}). These transformation temperatures make the wires suitable for most real life applications, except for those concerning too high temperatures (cooling of the wire is slowed down or completely prevented). Heating of the wire is normally done by passing a current through it (joule heating), but any method that makes the wire reach a temperature of A_{finish}, without damaging it, can be used.

Table 2.1 also shows some technical data from [37] that has to be taken into account when working

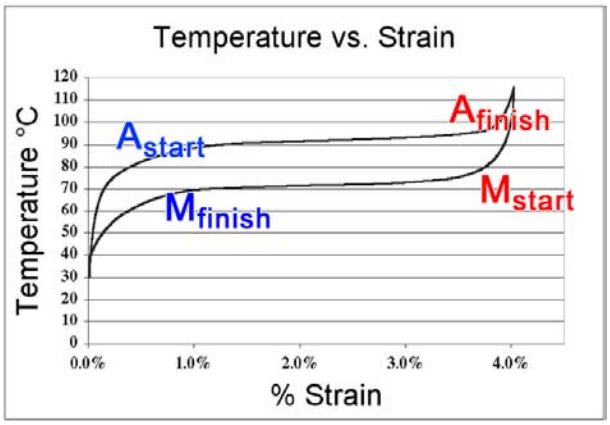


Figure 2.13: *Transformation curve of Flexinol [37]. Relatively large changes in strain are observed between* $85°C$ *and* $95°C$

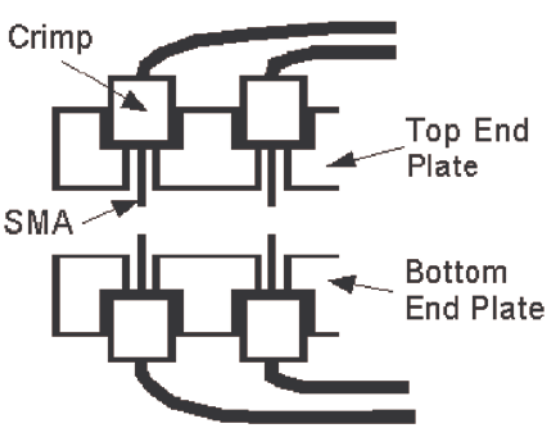
with Flexinol. The first column of table 2.1 contains the different wire diameters. The diameter of the wire determines its resistance and maximal pull force, found in column 2 and 3. A Flexinol wire has a specified resistance according to the length and diameter of it. The specified current in the table is the current needed to heat the wire above its transition temperature in one second. More about the time response of Flexinol can be found in [20]. The voltage U, needed to create such a current can easily be calculated using Ohm's law, $U = RI$, where R is the resistance of the wire, measured or calculated from its length, and where I is the wanted current. Generally, resistive materials have an increase in resistance when heated. This does not apply to Flexinol in the same way due to the fact that the shape of the wire changes as a function of temperature. The volume of the wire is constant at all times, leading to a larger diameter and smaller length in the austenite phase compared to in the martensite phase. The formula for a materials resistance R is given as $R = \frac{\rho \cdot l}{A}$, where ρ is the specific electrical resistivity of the material, l is the length of the material and A is the cross-sectional area of the material. From the formula it is clear that by shortening and thickening the wire, the resistance is decreased. This assumption holds because the increase in ρ caused by temperature is negligible compared to the decrease in resistance caused by deformation.

The pull force found in column 3 of table 2.1 is the maximum guaranteed pull force reported by the manufacturer. This means, that by exceeding the maximum pull force, a stable contraction rate over time cannot be guaranteed. This should specially be taken in concern when designing the mechanical parts of a system. An overloaded wire will most likely cause the need for a replacement - a drawback in most applications.

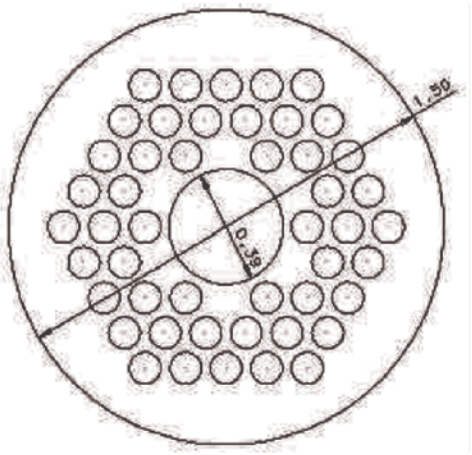
As already mentioned, the current given in column 4 is the amount needed to contract the wire in one second, surrounded by air at room temperature. Of course, a smaller current than specified in the table can be used. This leads to a slower contraction but also to a higher power consumption caused by the increase in time that the warm material is exposed to its surrounding air. From the manufacturer of Flexinol, it is reported that any current waveform can be used to heat the wires. This makes the wires suitable for embedded solutions, using for example a widely common PWM module to control the contraction speed. Column 5 and 6 in table 2.1 show the cooling time for Flexinol wires with $A_{start} = 70°C$ and $A_{start} = 90°C$ surrounded by air at room temperature. The differences in cooling time solely depend on the surface to volume ratio of the wires. The surface area of the wire has a linear growth when written as function of the diameter ($A = \pi \cdot D$). In comparison, the volume of the wire has a quadratic growth ($V = \pi \cdot (D/2)^2$) and as a result, doubling the wire diameter results in more than a doubling of the cooling time. In applications where great force is needed, faster cooling times may be achieved by coupling multiple wires in parallel. This solution depends on that multiple Flexinol wires are able to work together in parallel and has the disadvantage that more space is needed. In [21], a design for a Flexinol bundle is proposed (depicted in figure 2.14).

To shorten the cooling time, a number of different actions can be taken as shown in table 2.2. The improvement ratios in the table are given in the technical specifications for Flexinol [37]. Another cooling technique is discussed in [38].

The rightmost column in table 2.1 shows the force that is recommended to stretch a wire that has



(a) Crimping of the wires to the end plates



(b) End plate for parallel wires. The number of electrical parallel connected wires can be used to control the electrical operating point of the wires

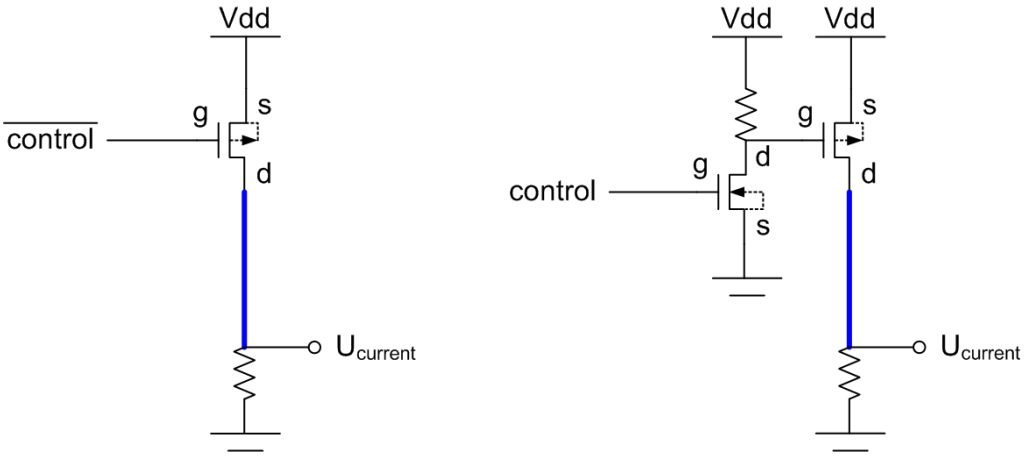
Figure 2.14: *A SMA Bundle Actuator [21]. By using many parallel wires, very large forces can be exerted*

Method	**Improvement**
Still air	1:1
Increasing stress	1.2:1
Using higher temperature wire	2:1
Using solid heat sink materials	2:1
Forced air	4:1
Heat conductive grease	10:1
Oil immersion	25:1
Water with glycol	100:1

Table 2.2: *Cooling techniques [37]. The improvement is the increase in cooling rate when the corresponding cooling technique is applied*

been contracted. The numbers are provided by the manufacturers of Flexinol.

In figure 2.15a, a drive circuit schematic for a Flexinol wire can be seen. The current flowing through the wire is controlled by the power MOSFET, directly connected to a power supply. A precision resistor connected between the wire and ground is used for current measurements. The resistor should be small enough to build up only a minimal voltage, relative to the drive voltage of the wire. The use of a pMOS transistor may seem strange because a nMOS normally has better driving characteristics. However, for the small signal driver it is desirable with a positive control signal for safety reasons. An undriven or uninitialized control signal should not cause a wire contraction or a current flow as this could be dangerous in for example a robot application that interacts with humans.



(a) Normal driver circuit. Charging of the transistor may draw currents $\geq$ 20mA

(b) Small signal driver circuit. Suitable in for example microcontroller applications

Figure 2.15: *Circuits for driving Flexinol wires (blue)*

Precautions

The contraction speed of the wire is reported to be proportional to the current used to heat it. However, concern should be taken, not to overheat or overload the wire. A too high temperature or a too high stress will cause permanent damage to the wire, reducing its ability to contract [37]. The manufacturer has not reported any detailed specifications other than a general warning. Stretching of the material is normally done using a spring or a dead weight. Although the strain percentage of Flexinol is reported to be maximal 5%, the maximal strain percentage of the NiTi alloy is 8% [39]. In practice, this means that from its austenite phase, the wire can be stretched 8% without causing any damage to its crystalline structure. By violating this limit, permanent damage is done to the wire, decreasing its maximum strain rate.

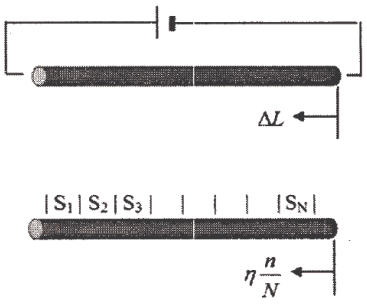
Control

In many applications, it is desirable to control the contraction of the wire somewhere between no contraction and maximum contraction. As figure 2.12b shows, the transformation curve shifts according to applied stress, making such a control of the contraction rate non-trivial. Theoretically this could be done by measuring the current passing through the wire, as mentioned above. However, no research has been found confirming this assumption.

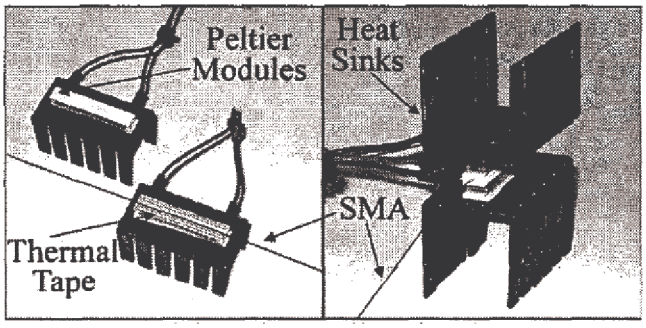
In [17], a method for controlling the degree of contraction is proposed. The authors heat and cool the Flexinol wire with the use of Peltier elements. Peltier elements are electrically driven heaters and coolers. When passing a current through the element, a temperature difference builds up between the warm and cold side of the element. By applying a heat sink to the cold side, high temperatures may be built up on the warm side of the element. By reversing the current, the warm side becomes cold and vice versa [40]. The authors use several elements to heat the different segments of a Flexinol wire, as illustrated in figure 2.16a. Every single Peltier element is controlled in a binary manner, set to either heat or cool its corresponding wire segment. In this way, by deciding how many Peltier elements that cool the wire and

how many that heat the wire, the degree of contraction of the wire can be controlled. The assembly of Peltier elements on a Flexinol wire is depicted in figure 2.16b.

By looking at commercial Peltier elements [41, 42], it can be concluded that heating and cooling requires more power than normal joule heating does when controlling a Flexinol wire. What has to be taken into consideration when applying Peltier elements to an application is the increase in system complexity. The biggest disadvantage by introducing Peltier elements to control Flexinol wires, is the increased physical space needed for each wire. By increasing the space need for an application, the biggest advantage of Flexinol, its weight to force ratio, is removed.



(a) Segmentation of the wire. By regulating each segment to a hot or cold state, the total amount of contraction can be controlled



(b) Assembly of a Peltier element on a Flexinol wire. The result is a large increase in physical size

Figure 2.16: *Binary control of segmented Flexinol wire [17]*

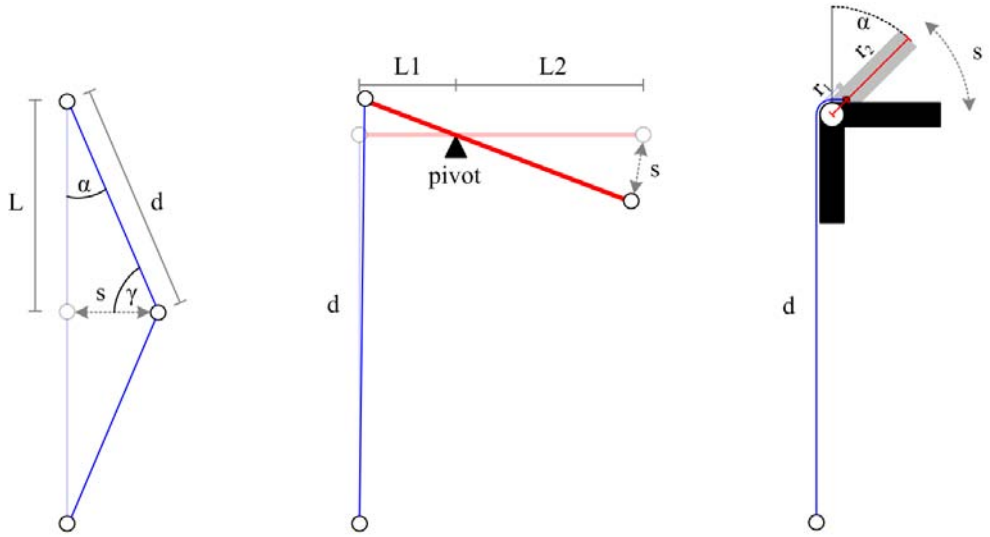
Movement

One drawback of Flexinol wires is their limited strain rate. Under normal conditions, a strain rate of 4.5% can be expected, which in many applications means that very long wires have to been used in order to generate the needed amount of movement. The most elegant and easy way to use Flexinol is to connect one wire that generates exact the amount of desired force and movement. In many real life application, this cannot be expected due to space limitations and force needs. Therefore, different mechanical gear mechanisms have been proposed to overcome this problem. The gear mechanisms shown in figure 2.17 are from the makers of Flexinol [37]. Figure 2.17a shows an angle pull of a Flexinol wire (blue line), where the displacement ratio is given by the formula $\frac{\delta d}{\delta s} = \frac{1}{sin(\alpha)}$. Here, δd is the contraction of the wire and δs is the output movement of the gear. The formula shows that a decrease of the angle α leads to a higher ratio, the ratio is in other words variable over the working range of the gear. The force transmission is also affected by the gear and can be written as $F_s = F_d \cdot cos(\gamma)$, where F_s is the output force of the gear and F_d is the force generated by Flexinol. As expected, an increase in stroke length leads to a decrease in force. Figure 2.17b shows a second gear variant which makes use of a lever mechanism. The lever (red bar) has a defined pivot point $L1/L2$ from one end. The displacement ratio of the gear is linearly dependent on the relationship between $L1$ and $L2$ and can be written as $\frac{\delta d}{\delta s} = \frac{L1}{L2}$. The force ratio is inverse of the displacement ratio and is written as $F_s = \frac{F_d \cdot L1}{L2}$. Figure 2.17c shows a radial pull which is similar to the anatomical principals of tendon fastening in the human body. The displacement of the gear is written as $\frac{\delta d}{\delta s} = \frac{r_1}{r_2}$ and the output force as $F_s = F_d \cdot cos(\alpha)$.

Another gear mechanism is reported in [22] and can be seen in figure 2.18 and 2.19. The author introduces a special routing scheme in a pattern around multiple gear plates (figure 2.18b and 2.19a) and makes use of the angle pull from figure 2.17a. A loss of force due to friction and deformation is observed, but in exchange the author is able to get a strain rate of 18%. However, the size of the gear makes the application gigantic compared to a single Flexinol wire.

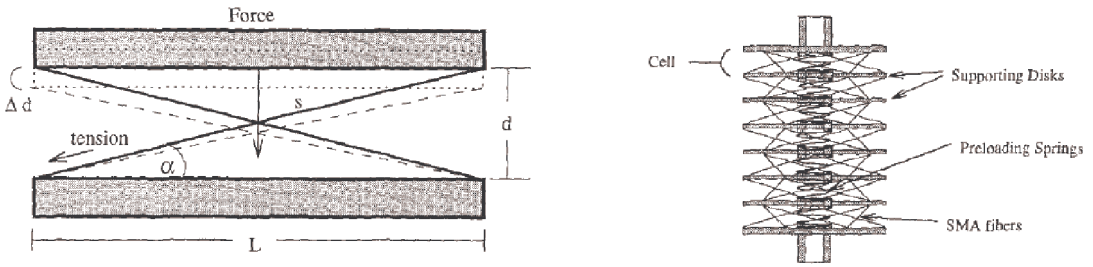
Commercial Applications

Although Flexinol and similar products have been available on the market for several years, it does not seem like a particular brand of applications have been pointed out as very suitable. Not many examples



(a) Angle pull (b) Lever (c) Radial pull

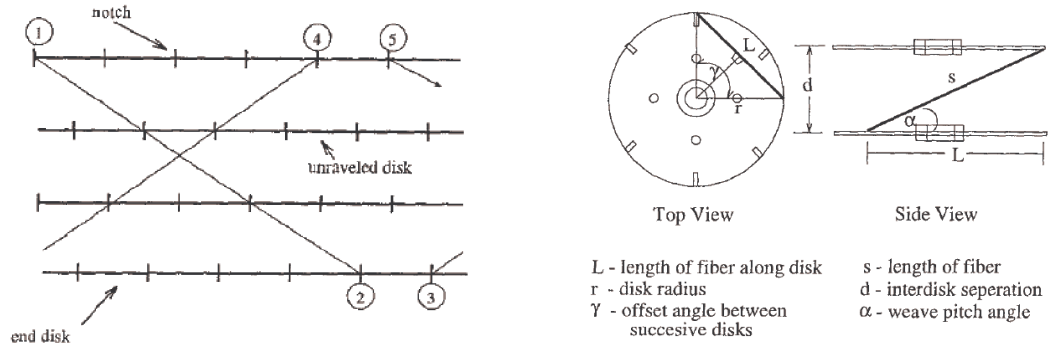
Figure 2.17: *Gear mechanisms for Flexinol [37]. The gears result in larger strains and lower forces*



(a) Gear plates are pulled against each other

(b) Gear assembly. The number of gear plates decides the strain rate

Figure 2.18: *SMA actuator with high strain [22]. Proposition unfortunately results in a large increase in physical size*



(a) Wire routing schematic. The wires are routed in a diagonal pattern around the gear structure

(b) Top view and side view of the gear. The routing tracks can be seen on the top of the gear plate

Figure 2.19: *SMA actuator with high strain [22]*

have been found on commercial applications that use Flexinol, and the ones found may appear to be produced by Dynalloy, Inc. An example for one of these products is the Electrostem^{TM} air valve [43]. The air valve has been made to give a practical example of the possibilities of Flexinol. The rather simple design of the air valve can be seen in figure 2.20.

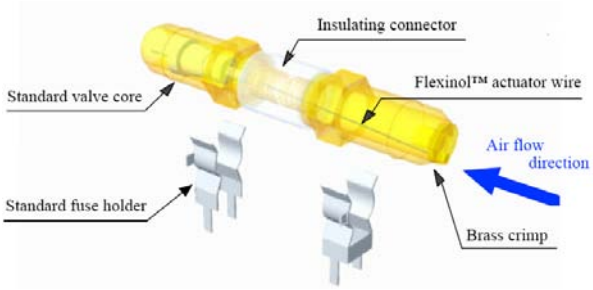


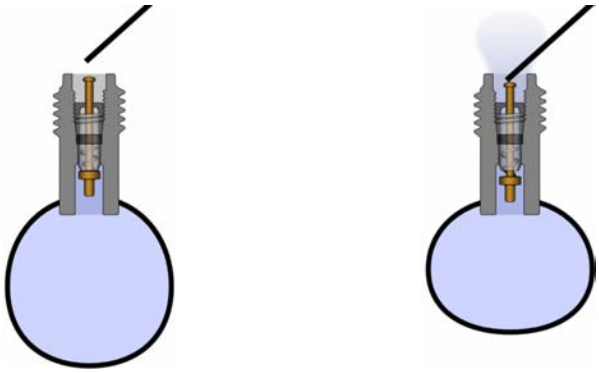
Figure 2.20: *The Flexinol Electrostem^{TM} air valve [43]. The valve uses the thermal properties of Flexinol to regulate air flow*

The valve is said to be able to proportionally control the air flowing through it. The valve itself is a standard car tyre valve (Schrader valve) which is controlled by its internal stem cap as seen in figure 2.21. As figure 2.20 shows, the Flexinol wire is fastened internally on the stem cap of the Electrostem^{TM} valve. When contracting the wire, the valve opens and air flows through it. This operation itself is not very special compared to other electrically driven air valves except for the small size of the valve. However, as air flows through the valve, one of the special characteristics of Flexinol is revealed. The air flow cools the Flexinol wire down, working against the Joule heating of the wire. With an increasing pressure difference between the inside and the outside of the valve, the amount of air flowing through the valve also increases. The wire is further cooled causing the valve to decrease the air flow until a balance between heating and cooling of the wire stabilizes.

The wire needs about 750mA to operate, but if a higher air flow is wanted, the current may be increased. However, if the current is increased and the airflow suddenly stops, the wire and the valve could very fast overheat causing permanent damage. The danger of overheating is generally a reoccurring challenge when using Flexinol, and should always be taken into concern when designing applications.

Non-Commercial Applications

The small six-legged robot bug *Stiquito* can be found in [44, 45]. The robot bug is able to walk 10 centimeters per minute and carry a load of 50g. It uses Flexinol wires of 100μm to move its legs. Other scientific applications are presented in [46, 47].



(a) Schrader valve closed

(b) Schrader valve open

Figure 2.21: *Schrader valve operation which can be handled by a Flexinol wire*

2.3.3 Electroactive Polymers

Electroactive polymers (EAP) are a group of polymer materials that change their shape as a result of an applied voltage [48]. Electroactive polymers can be divided into two groups, whereas the first group are electroactive polymers that can be operated in a dry environment. The other group consists of polymers that need to be wet in order to function. Common for all EAP-materials is that large strain rates can be expected when compared to for example shape memory alloys. However, the exerted force is much lower than that of SMAs - one of the biggest drawbacks of EAPs. Examples of EAP applications can be found in [49, 50, 51, 52, 53, 54]. Figure 2.22 and 2.23 show two underwater applications actuated with fins built from EAP materials.

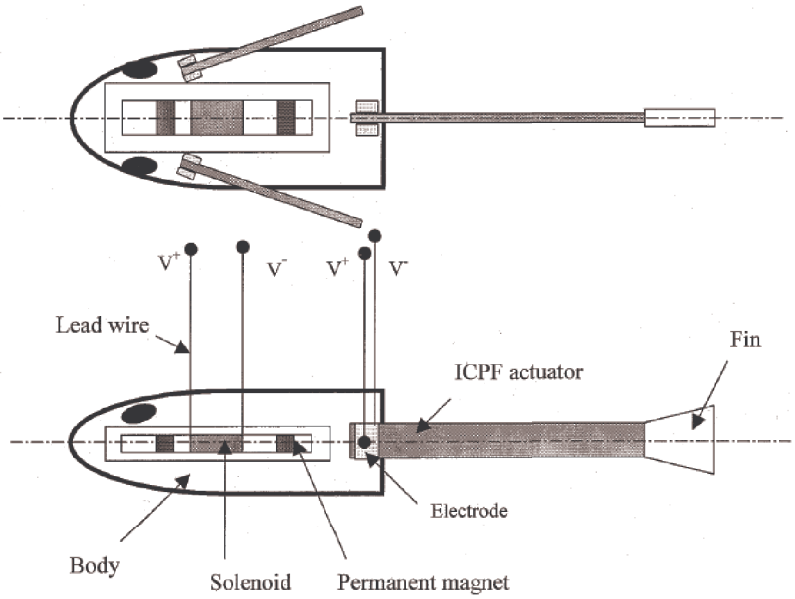
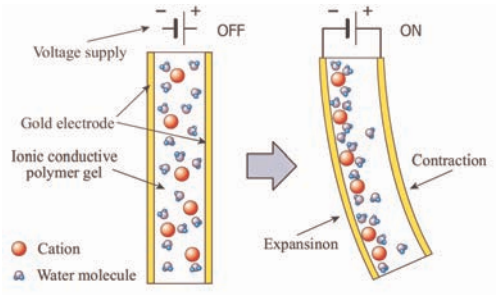
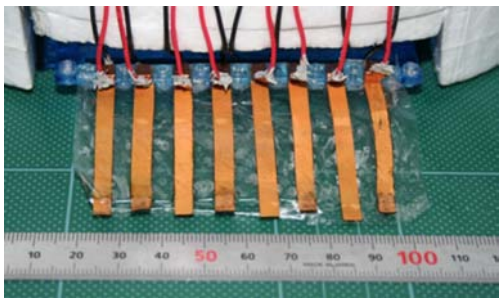


Figure 2.22: *Underwater micro robot that is actuated with a ICPF (Ionic conducting polymer film) fin [55].*



(a) Deformation mechanics of IPMC (Ionic polymer-metal composite) [50]. When a voltage is applied, the water in the material moves to one side, causing a bending of the material.



(b) A fin construction for a Rajiform Swimming Robot [50]. EAP technologies that need a wet environment are well suited for underwater applications.

Figure 2.23: *EAP technology*

2.4 Actuator Comparison

In this section a comparison between the mentioned actuators is done. Table 2.3 shows a summary of the results. As more and more manufacturers enter the market, it gets harder and harder to get a general view of the available products. In many applications it is not easy to decide what actuator technology

to use, as many technologies share some of their properties. A deep analysis of the actual actuator need is therefore crucial to make a good decision.

Flexinol and electroactive polymers differ from the other actuators in being very small in size and in being non-mature technologies. Flexinol is able to exert great forces and has a high power to weight ratio [56], but suffers from small strain rates, that can be compensated with gears. Electroactive polymers can be said to have opposite properties with large strain rates but only exert small forces. Electroactive polymers in some cases need drive voltages of $\geq$ 1000V.

Hydraulics have been used for a long time and a wide number of parts in different sizes are therefore available. Lately, very small systems have also been developed. Tremendous forces can be achieved using hydraulics, at the cost of heavy systems. As a result of technology development, pneumatics are more and more used in robotics in the form of PAMs. These systems are in general lighter than hydraulic systems, but the exerted forces are lower when compared against the system weight.

Radial movement is often achieved by RC servos or stepper motors. For example are RC servos ideal for actuating a joint with a given degree of movement, while a stepper motor would better be fitted in an application driving for example a multi-turn wheel. Both technologies seem to create about the same amount of torque, but stepper motors have the disadvantage that they lose power as the motor speed increases. On the other hand, RC servos tend to be rather slow compared to stepper motors, caused by their internal gears. Stepper motors normally do not need gears.

Solenoids are normally used to operate valves, but may also be used in robotic applications. However, as only to positions are available when actuating, the number of suitable applications are rather low.

Actuator	**Motion**	**Force**	**Size**	**Res.**	**Control**	**Strain**	**Ref**
Flexinol (SMA)	Linear	40N	Small	high	Non-trivial	4.5%	[37]
Electroactive polymers	Linear	Weak	Small	high	Non-trivial	$\leq$ 120%	[35]
Hydraulics	Linear	Strong	Med/Large	high	Controller	-	[27, 28]
Pneumatics	Linear	Medium	Med/Large	On/Off	Controller	-	[29]
PAM	Linear	Medium	Medium	high	Controller	30%	[30]
RC Servo motors	Radial	ca 250Ncm	Medium	ca 1°	PWM	$\pm$90°	[31]
Stepper motors	Radial	ca 300Ncm	Medium	ca 0.5°	Controller	-	[57]
Electric Solenoids	Linear	150N	Medium	On/Off	Binary	-	[33]

Table 2.3: *Comparison of different actuators and some of their properties*

2.4.1 Power to Weight Ratio

Figure 2.24 shows the power-to-weight ratio [58] of the above mentioned actuator technologies. Values for SMAs, hydraulics, pneumatics and DC-motors were found in [35, 59, 60]. Values for EAP actuators are based on numbers found in [61, 62]. For electronic solenoids, no exact numbers were found. The values in the figure is therefore based on one of the smallest and one of the largest solenoids from Ledex [63, 64]. To calculate the effect, an assumption was made that the actuation time for the solenoid is one second which is considered a reasonable actuating time in robotic applications.

The actuation time that is used to calculate the effect is a factor that has to be considered when comparing the power-to-weight ratio of all technologies. Lately, very small DC-motors and hydraulics have become available. This comparison does not include such actuators.

2.5 Feedback Sensors

This section describes different types of feedback sensors suitable in a humanoid robot finger. Two main types are mentioned, sensors for displacement measurements and sensors for force measurements.

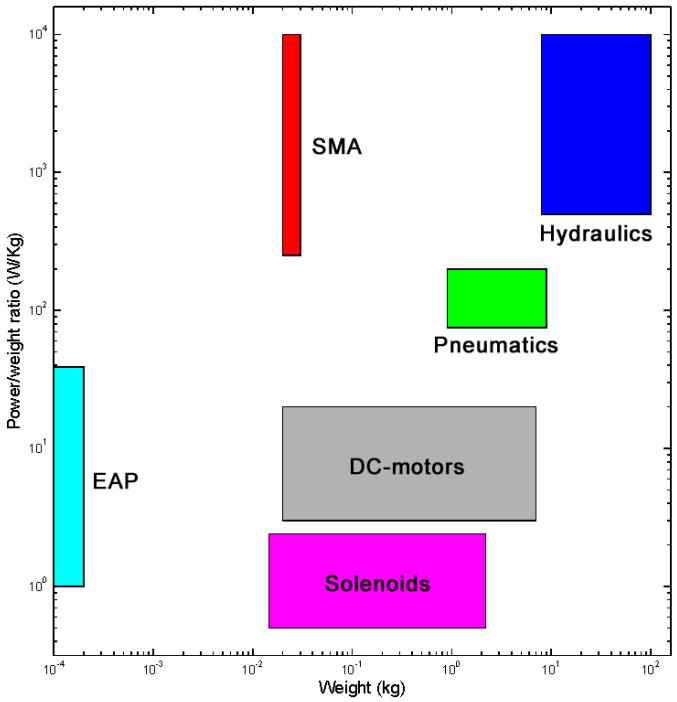


Figure 2.24: *Power-to-weight ratio of different actuator technologies [35, 59, 60, 61, 62, 63, 64]*

2.5.1 Displacement Transducers

In robotic applications, it is often desirable to know how much mechanical parts have been moved. Such tasks are often solved with a displacement transducer.

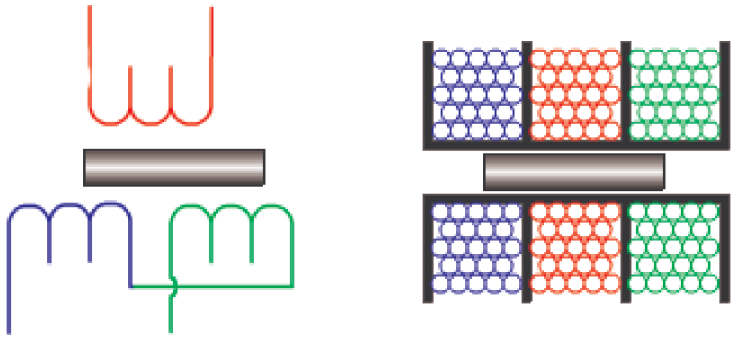
Linear Variable Differential Transformers (LVDT)

A LVDT is in general a simple construction consisting of three coils and a sliding magnetic core. Figure 2.25 shows a schematic and a layout view of a LVDT. The single red coil is driven with a fixed frequency and fixed amplitude AC signal. The two secondary coils (green and blue) are connected in opposite directions so that when the metal core is in the center position, the induced currents from the coils cancel each other. However, by moving the metal core away from the center position, a difference in the two signals is measured according to the position of the core. By looking at the phase of the differential signal according to the signal present at the center coil, it can be determined whether the metal core is moved to the left or to the right of the center position.

The use of electromagnetism makes mechanical contact between the metal core and the coils unnecessary. This has two great advantages. This means that very little wearing can be observed, causing a long life. It also leads to clean data and almost infinite resolution. On the other hand, the use of LVDTs requires a fixed frequency for the center coil. If a dc output is wanted, some simple signal processing also has to be done externally. Some products like [65] have internal signal processing and frequency generation and only need DC power to be operated.

Resistive (potentiometric) Displacement Transducers

Compared to LVDTs, resistive displacement transducers are in general cheaper. They consist of a slider that is moved back and forth over the surface of a resistive material as figure 2.26 shows. The resistive material acts like a voltage divider, causing the output voltage to be proportional to the position of the slider. Resistive displacement transducers are widely used, but because of the contact between the sliding mechanism and the resistive material, their lifetime is in general shorter than that of a LVDT. A great advantage of resistive displacement transducers is the simple three wire connection. Any common supply voltage can be used as excitation and they can therefore easily be fitted into most applications.



(a) Schematic view. The blue and green coils are connected symmetrically

(b) Layout view. The red coil induces a magnetic field that is transferred to the other coils via the iron core

Figure 2.25: *Principal of LVDT [66]*

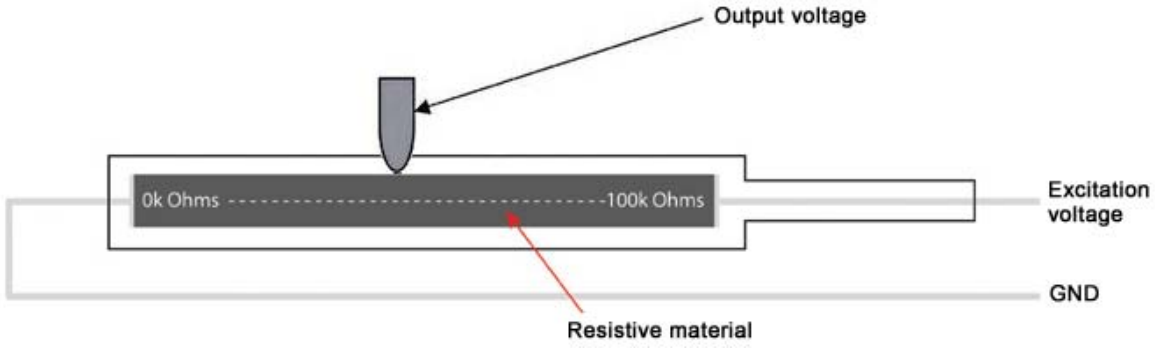


Figure 2.26: *Resistive displacement transducer. The slider is moved over a resistive material that is functioning as a voltage divider*

Merlin Elastic Stretch Sensor

The stretch sensor is made from a smart material which increases its resistance when it is stretched. The sensor is depicted in figure 2.27 and looks and behaves just like a normal rubber cord. However, the rubber mix is a conducting material and its resistance can easily be measured by fastening a terminal in both ends. A great advantage of the sensor is its limited need for space.

2.5.2 Force Transducers

In robotic applications it is often desirable to measure mechanical forces. This may for example be to restrict the gripping force of a robot hand carrying a glass. In other applications the sensor would only need to report whether a contact force is present.

Strain Gauge Load Cell

A strain gauge is a long conducting wire, arranged in a pattern like the one seen in figure 2.28. The wire is mounted on a flexible base. When the strain gauge is bent in a vertical direction, only a small change to the geometry of the wire occurs. However, if the strain gauge is bent in a horizontal direction, the majority of the wire length will be stretched. The stretching of the material means that it gets longer and thinner. According to the formulas for the resistance of a material, this leads to an increase of resistance in the strain gauge.

By placing a strain gauge on the surface of a material, the strain of the material can be measured as a change in the strain gauge resistance. The strain of the material is caused by an external force. Therefore, by placing a strain gauge on a material with a stiffness proportioned to the expected force

Figure 2.27: *The elastic stretch sensor from Merlin increases its resistance when it is stretched*

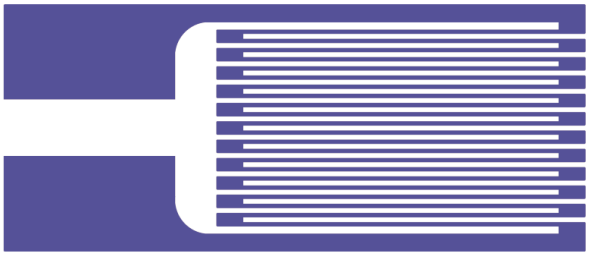


Figure 2.28: *Strain gauge. The zig-zag pattern makes the wire change its resistivity when stretched in a horizontal direction*

range, the exerted force can be measured as a change in strain. The material that the small strain gauge is fastened on is in general much larger than the strain gauge itself. To measure the small changes in strain gauge resistance, a Wheatstone bridge is often used. Figure 2.29 depicts four strain gauges on the surface of a material, connected as a Wheatstone full-bridge. As the figure shows, an increase of resistance in R2 and R4 occurs when the material is bent. This unbalances the two voltage dividers of the Wheatstone bridge and causes a voltage difference between their two center nodes. The force causing a material strain is represented by this voltage.

Strain gauge load cells are specified with a maximum load and a sensitivity. The sensitivity of the load cell is reported as mV/V. This denotes the amount of millivolts per excitation volt the output of the Wheatstone bridge will be when the load cell is loaded to its maximum. For a load cell specified to 100N and 2mV/V, an excitation voltage of 1V would cause an output of 2mV when the cell is loaded with 100N. In the same way, with an excitation voltage of 10V the output would be 20mV.

Because of these low voltages, the use of a Wheatstone bridge is required. The differential measurement will cancel noise that is present on both channels. Also, if only one voltage divider would have been used, the output voltage would be $\frac{V_{dd}}{2} \pm 20mV$. If a V_{dd} of 5V would be used, the output would therefore be about 2.5V $\pm 20mV$. This would waste almost the hole measuring range of the data acquisition unit. Instead, by introducing two balanced voltage dividers and measuring their difference, a much more efficient use of the measuring range is achieved ($0V \pm 20mV$). Another great advantage of the Wheatstone bridge is its ability to cancel temperature drifts. This is caused by the mirrored placement of resistors like R2 and R4 in figure 2.29.

Interlink Force Sensing Resistor [67]

The Interlink Force Sensing Resistor (FSR) is not designed for accurate force measurements, but can be used for indications. The FSR decreases its resistance as stress is applied to its polymer material. This behaviour is known as the piezoresistive effect. Although the accuracy of the FSR is reported to be rather poor, its physical size of ca $8mm \cdot 1mm$ (figure 2.30) makes it ideal for robotic use. In contrast, strain gauge load cells tend to be very large. A great advantage of the FSR is its high sensitivity. A small change in pressure causes the resistance of the sensor to change with many hundred Ohm. This eliminates the need for an amplifier.

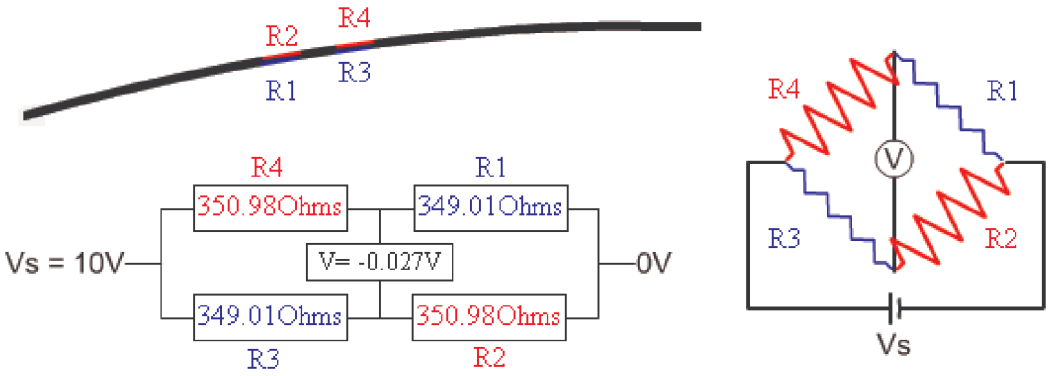


Figure 2.29: *Strain gauge connected as Wheatstone bridge. Very small changes in resistivity can be measured*

Figure 2.30: *Interlink Force Sensing Resistor [67]. Small changes in pressure result in large changes in resistivity*

Chapter 3

Used Tools

In this chapter, three essential development tools are presented. First, the Atmel ATMega32 microcontroller is described. The ATMega32 will later be used to implement a control unit for a humanoid finger. Secondly, the data acquisition unit for the later described test frame is described and finally, an overview of the robotic software framework, Microsoft Robotics Studio, is given. An interface for Robotics Studio will later be developed for a humanoid finger.

3.1 Atmel AVR Microcontrollers

The 8bit AVR microcontrollers from manufacturer Atmel (www.atmel.com) are a series of RISC-controllers with high performance and low power consumption. They have a Harvard architecture (figure 3.1) and can be operated at clock frequencies up to 20MHz. Code development, compiling and simulation can be done in Atmels own AVR Studio, which also handles uploading the compiled microprogram to the flash memory of the microcontroller. This can be done via the microcontrollers In System Programming (ISP) interface, Jont Test Action Group (JTAG)interface or High Voltage Serial Programming (HVSP) interface. This section focuses on the AVR ATMega32 device [68], but the information is also highly relevant for other AVR devices as all of these are built upon the same AVR hardware platform.

The ATMega32 has a wide range of integrated functions for internal and external operations. This is a presentation of the most important functions used in this thesis. Generally, all functions in the microcontroller are controlled by setting bits in the corresponding control registers of the function.

3.1.1 I/O-Ports

The microcontroller has many different I/O-ports, the number depends on the chosen housing. All ports can be used as general digital inputs or outputs and in addition most ports can be configured to perform one or more other functions. The pinout of the ATMega32 in a PDIP-40 housing can be seen in figure 3.2. The pin names written within a parenthesis are the alternate functions of each pin.

When a port is used as a normal digital input or output pin, the direction of the signal has to be configured. This is done by writing respectively a '1' or a '0' in the data direction register (DDR) of the port. When used as input, an internal pull-up of the pin is done, making the use of external active low signals convenient. The alternate functions of a port is configured through the control register of the function. Some of these functions are described below.

3.1.2 Memory

The ATMega32 has 32kB of internal non-volatile flash memory for program code. Additionally, the microcontroller has 2kB of SRAM used for data variables and 1kB of EEPROM for data storage. The SRAM is volatile, meaning its state gets lost on a shutdown of the microcontroller. The EEPROM is non-volatile and supports 100,000 write cycles, making it very suitable for storing device setup or medium sized datasets.

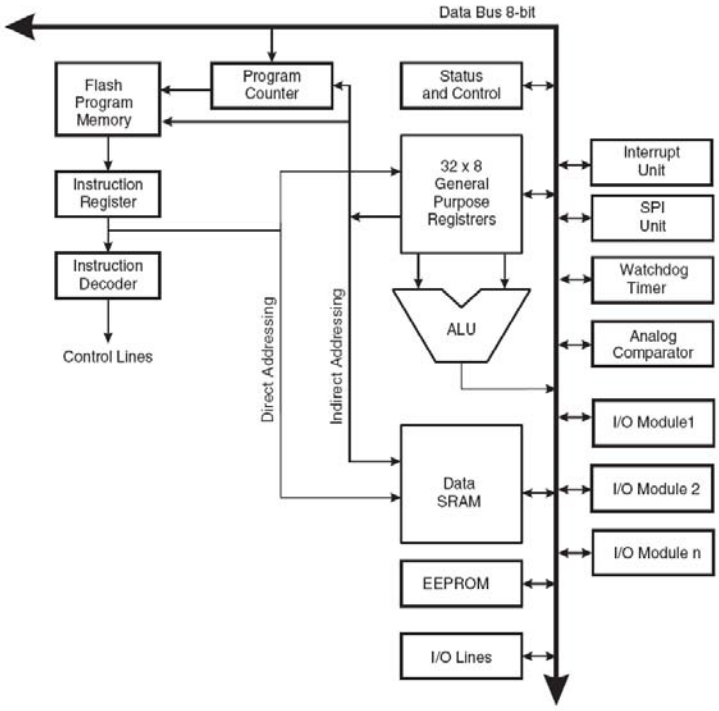


Figure 3.1: *Harvard architecture of ATMega32 [68] with separate memory and buses for data and program*

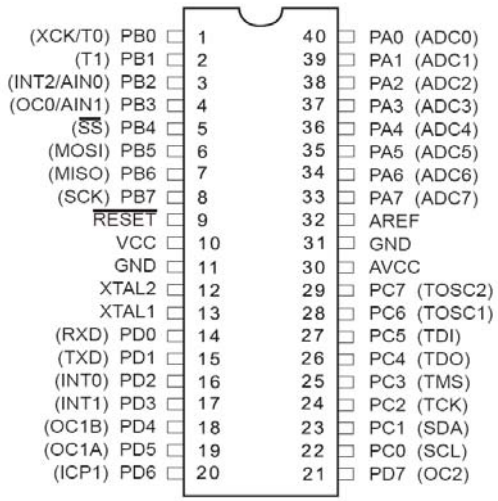


Figure 3.2: *Pinout of the ATMega32 microcontroller [68] in a PDIP-40 housing. Alternate port functions are inside the parenthesis*

3.1.3 Interrupts

Different asynchronous events may occur during the execution of a microprogram. Interrupts are used to prevent the programmer from having to poll different status registers constantly, waiting for events to happen. An interrupt can for example be caused by a level change on an external interrupt port, a counter overflow or a complete reception of a serial data byte. The first 40 bytes of the program memory contains 20 interrupt vectors. For each vector a jump command tells the microcontroller where in the program code the triggered interrupt is to be handled. If a microprogram runs when an interrupt is triggered, the execution of the program is halted. Then the interrupt is handled according to the interrupt vector. The handling should of course not take too long, or else the occurrence of other interrupts may be overseen. When the handling of the interrupt is done, execution of the main program continues from where it was halted.

3.1.4 Counters and Pulse Width Modulation (PWM)

Two 8bit counters and one 16bit counter are available on the microcontroller. In general, these consist of a counter register, a prescaler, control register and compare registers. The prescaler divides the global clock signal of the microcontroller with either 1, 8, 64, 256 or 1024, allowing different counter speeds. The counters can be set to operate in three different modes. In normal mode, the counter starts counting from 0 and counts upwards until the counter register overflows (8bit/16bit). The overflow will trigger an interrupt if the corresponding bit is set in the Timer/Counter Interrupt Mask Register (TIMSK) of the microcontroller.

The second mode is called Clear Timer on Compare Match (CTC) and does, as the name suggests, a reset of the counter when a compare value is reached. As the counter is reset, an interrupt is triggered if set up in the TIMSK-register. An output pin on the microcontroller can be set to toggle on each compare match, creating a clock signal with programmable frequency. By correct setup of the CTC-mode, an interrupt can be triggered on a regular basis and thus update for example a clock register. The frequency of the counter (F_{CTC}) is written as function of the system clock (F_{CPU}), the prescaler (N) and the reset compare value (OCR):

$$F_{CTC} = \frac{F_{CPU}}{N \cdot (1 + OCR)}$$

The third mode of the counter is the Pulse Width Modulation (PWM). This mode can again have three different modes of operation, called Fast PWM, Phase correct PWM and Phase and Frequency correct PWM. This section describes the Fast PWM mode.

Fundamentally, the PWM mode works a lot like the CTC-mode. First, a reset value is set for the counter, deciding the frequency of the generated PWM-signal. In addition an output compare value is set. When the counter counts from 0 to the reset value, it compares its value with the output compare value. Depending on an inverting or non-inverting mode of operation, the output pin of the microcontroller corresponding to the counter is toggled from 1 to 0 or from 0 to 1 when the counter value matches the output compare value. In this way, by choosing an output compare value between 0 and the reset value of the counter, the duty cycle of the PWM-signal is controlled. The frequency of the Fast PWM signal is written as function of the system clock (F_{CPU}), the prescaler (N) and the reset compare value (OCR):

$$F_{FastPWM} = \frac{F_{CPU}}{N \cdot (1 + OCR)}$$

Figure 3.3 illustrates the operation of the Fast PWM mode, where $TCNTn$ is the counter register, $OCRnx/TOP$ is the reset compare value, and $OCnx$ is the output pin for the PWM signal. $OCnx$ and $\overline{OCnx}$ represent respectively inverting and non-inverting mode of operation.

3.1.5 Universal Synchronous and Asynchronous Serial Receiver and Transmitter (USART)

For serial communication with its surroundings, the ATMega32 contains an USART. This function unit can be set up to send and/or receive data on the TX and RX pins of the microcontroller allowing it to communicate with other microcontrollers, serial analog to digital converters, computers (with level shifter like the MAX232 from maxim), GPS modules or any other device with an USART interface. The

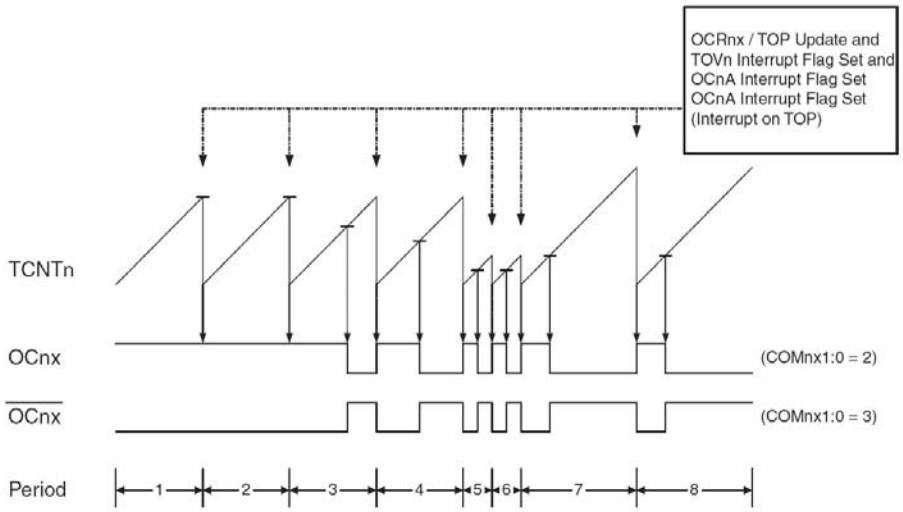


Figure 3.3: *Fast PWM mode (single slope) of ATMega32 [68]. The output is toggled each time the counter passes the compare value*

USART operates in a full duplex mode, supports 5, 6, 7, 8 or 9 data bits, 1 or 2 stop bits and odd, even or no parity check and generation.

The USART can be set up to trigger an interrupt on arrival of a new data byte or on a complete transmission of the current output data. By handling these interrupts correctly, the programming of buffers for sending and receiving data can easily be implemented. The Baud rate of the USART can be written as a function of the system clock (F_{CPU}) and the Baud Rate Register ($UBRR$) of the microcontroller:

$$BAUD = \frac{F_{CPU}}{16 \cdot (1 + UBRR)}$$

3.1.6 Analog to Digital Converter (ADC)

The ATMega32 has an Analog to Digital Converter (ADC) with 8 single ended channels and a resolution of 10bit. The ADC uses a successive approximation which supports conversion times between 13 and 260μs. Only one conversion unit is available and the 8 single ended channels are therefore connected to the ADC through an 8-channel analog multiplexer. In addition to single ended measurements, the three first channels (ADC0, ADC1 and ADC2) can be used in combination with the rest of the channels to perform differential voltage measurements.

3.1.7 Watchdog Timer

A watchdog timer is situated in the ATMega32 and is responsible for controlling that the microprogram does not hang in an indefinite loop. The watchdog timer works a lot like the other counters on the microcontroller, except for its wider range of prescaler values. When the watchdog timer reaches an overflow, a reset of the microcontroller is performed. To prevent unwanted resets, the programmer has to clear the watchdog timer regularly, before it reaches its maximum value. The watchdog timer can also be used to perform a software reset of the ATMega32. This is the only way to do a proper reset of the system without cutting the power supply lines.

3.1.8 Clock Source

The ATMega32 can be operated with different clock sources, making it fit to different applications. An internal RC oscillator is able to generate a clock signal of 1, 2, 4 or 8MHz and is in many applications the best choice because no external components are required. The internal RC oscillator is specified to give a frequency within ±3% of the nominal frequency. By adjusting the calibration byte of the oscillator, a frequency within ±1% is guaranteed. If an other frequency, a higher frequency or a higher accuracy is

needed, an external RC oscillator or an external crystal may be used. This may be the case in timing critical applications. The last clock source option of the microcontroller is the operation with an external clock signal. This option could for example be used in a system with a global clock signal and multiple synchronous microcontrollers.

3.2 Keithley KUSB-3100

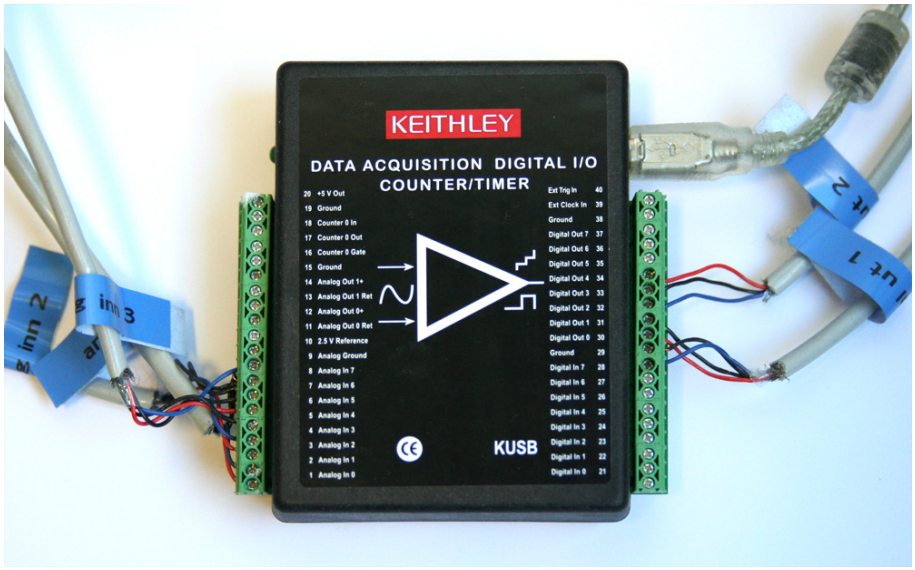


Figure 3.4: *The Keithley KUSB-3100 has 12bit resolution and 50kS/s sample rate*

The Keithley KUSB-3100 (figure 3.4) is a data acquisition (DAQ) unit with an USB interface. It also comes with an application programming interface (API) for the Microsoft .NET Framework. The DAQ unit has an internal 500V isolator barrier which protects the computer from external electrical disturbance. Table 3.1 shows the technical specifications of the unit. The KUSB-3100 is divided into several subsystems responsible for handling analog to digital conversion, digital to analog conversion, digital input, digital output and timer operation. These subsystems are exposed through the API.

Property	**Value**
Resolution	12Bit
Throughput	50kS/s
Analog input channels	8 Single ended
Analog output channels	2
Digital I/O channels	16
Counters/timers	1
Gain	1, 2, 4, 8
Connectivity	Built in screw terminals

Table 3.1: *Specifications of Keithley KUSB-3100 [69]*

3.3 Microsoft Robotics Studio

In robotic applications, dealing with I/O is a very natural operation. This is the interface between the robot and the real world. As robotic systems become more advanced, multi-threaded software is a natural leap towards imitation of the parallel processing capabilities of the human brain. However, the concurrency that multi-threaded systems possess also introduces some difficulties. Traditional I/O systems normally have one connection to a control unit and multiple I/O channels. How would this be

handled in a concurrent system if one I/O channel is used to control a leg and another used to control a hand? If the concurrent system has one thread for computation of leg movement and one for hand movement, there is a possibility that these two threads will try to access the single hardware connection simultaneously.

In a normal application, such settings will have to be avoided using software programming techniques for multi-threading such as semaphores. This functionality is an integrated part of Robotics Studio which aims to handle such settings hidden from the programmer of the leg and hand movement. This presumes that the hardware process has been designed to handle concurrent requests properly. The following list summarizes how Robotic Studio solves the challenge of I/O concurrency.

- Hardware is abstracted and wrapped in a Robotics Studio Service
- The hardware service can be reached from any other service
- The hardware can be polled via messages to the hardware service
  - The message handler decides if concurrent message handling is allowed
  - Robotics Studio queues requests if no concurrency is allowed
- Result
  - The degree of allowed hardware concurrency is defined when writing the hardware service
  - User does not need to worry about concurrent hardware request when writing services that access the hardware service

3.3.1 Overview

Microsoft Robotics Studio is a free framework for robotic application on the Windows platform and was released in December 2006. The idea behind Robotics Studio is to create an environment of concurrently working services which communicate with each other through different types of messages. There are two types of messages that can be sent between services. The first type is service operations, which are requests for a certain action to be performed. Such an action can for example be a request to return the current state of the service, a request to return a HTML-representation of the state, a request to acquire data from hardware or simply a request to update values in the state of the service. This way of system design, with independent modules, is also known as loosely coupled. The second message type is notifications, which are sent from a service when a predefined event occurs.

3.3.2 Concurrency and Coordination Runtime (CCR)

The CCR is a programming model for Robotics Studio and can be said to be an abstraction from the details considering multi threaded computation like semaphores, locks and critical sections. Behind the scenes, it handles all the work that normally would be done by hand in a system of concurrently working modules. Therefore, a service developer doesn't have to worry about the execution order of other cooperating services. The CCR itself is a dynamic linked library (dll) for .NET framework v2.0 or later, which means it is available through all languages targeting the .NET Runtime.

3.3.3 Decentralized Software Services (DSS)

The DSS is built on top of the CCR and is an application model which builds on the REST model. The principal of its operation is that a service is exposed to its environment by providing a state object. Expansion of the REST model includes structured data manipulation, event notification and service composition. DSS is built to support applications which are sewn together from multiple services, which can run on different computers and communicate over a network.

The main component of DSS is services. A service can be seen as an object instance in traditional object oriented programming. Figure 3.5 shows a graphical representation of a service component. The service identifier is a unique URI address for the service, which all other services use to communicate with it. The service is assigned this address when it is constructed. The service contract is a short description of the behaviour of the service which enables the composition or reuse of a service with a given contract.

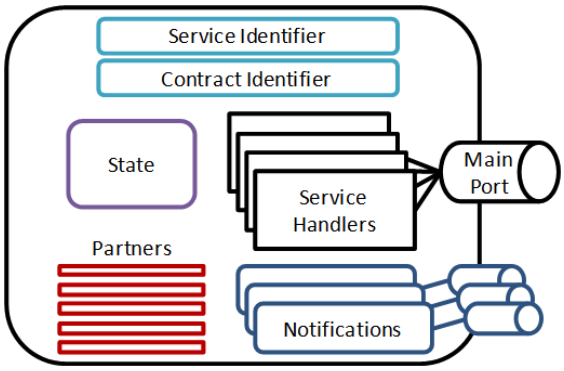


Figure 3.5: *DSS service model [70]. All communication with the service is done via its main port and notification port*

The service state is representing the current state of the service. It can for example contain the latest acquisition data from a hardware module, the settings for a robot test drive or any other stored information. The state can be seen as the public domain of a service, defining what information that can be retrieved and what information that can be modified. The service partner list contains all the other services that the service is cooperating with through messages. In order to be able to talk to other services, they are required to be defined in the service partners list.

The main port of the service is a FIFO data structure, which handles all received messages from other services. Each service defines what type of messages it can handle on its main port, which forwards the message to its corresponding service handler. The service handler handles the message and returns a confirmation or, in the case of a data request, the requested data.

3.3.4 Visual Programming Language (VPL)

In contrast to conventional textual programming of DSS-services, Microsoft Robotics Studio also offers the ability to do graphical programming with the Visual Programming Language (VPL). VPL is on one hand meant for beginners, with lack of experience in textual programming. On the other hand, it can also be used by experienced programmers to prototype, to define services and for code generation. VPL programming is a description of the dataflow between services or service internal actions like calculations, variable assignments or other functions. In Figure 3.6, one can see how data values, variables and calculations are used to construct a dataflow which finally is sent to a text-to-speech service. This simple program first initializes a variable called *Test* with the value 1. The value is then sent to the text-to-speech service, which tells the user the actual value. A boolean comparison is then done between *Test* and the value 10. If *Test* is unequal to 10, it is incremented by 1 and then sent to the speech service again. This is repeated until the value of *Test* is 10 and then the text string "All done!" is sent to the speech service before the program is terminated.

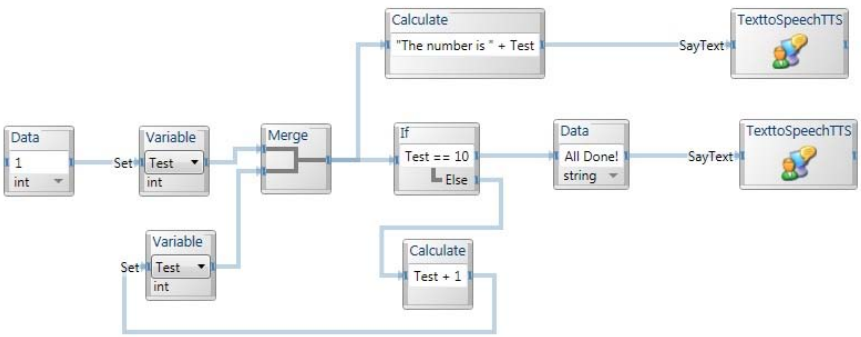


Figure 3.6: *Microsoft Visual Programming Language (VPL) [70]. A tool for beginners and for rapid prototyping*

Chapter 4

Own Methods

In this chapter, the methods developed during this thesis are described. First, the methods for testing Flexinol wires are presented. This is to gain an insight in the physical capabilities of Flexinol, as described in the thesis introduction. After this test, a humanoid finger design is developed, and finally the humanoid finger is assembled with sensors and a control mechanism is developed. Figure 4.1 shows an overview schematic of the developed methods.

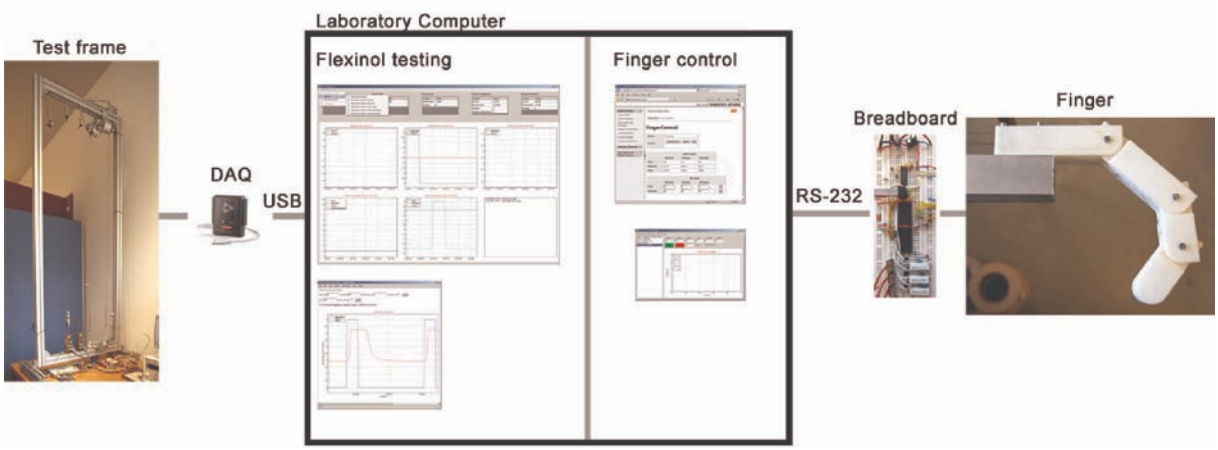


Figure 4.1: *Overview of the developed system. The test frame is used for long term testing, controlled through a Keithley KUSB DAQ-unit. The self-developed finger model is controlled with a custom made microcontroller program, communicating with the laboratory computer via RS232.*

4.1 Testing of Flexinol

Research in the field of shape memory alloys has been ongoing for many years. Interesting when reading articles is that it seems like very little work has been done regarding investigations on the physical properties and the long-time properties of SMAs. Attempts to do intelligent regulation of SMAs have been found in [22, 71, 72, 73, 74], but none of these papers actually use SMA wires as robot actuators. Because of its commercial availability, Flexinol is chosen as the SMA actuator under test.

This section describes the different tests that are to be performed on the Flexinol wires. The mentioned test frame is described in section 4.2.

4.1.1 Fixation Test

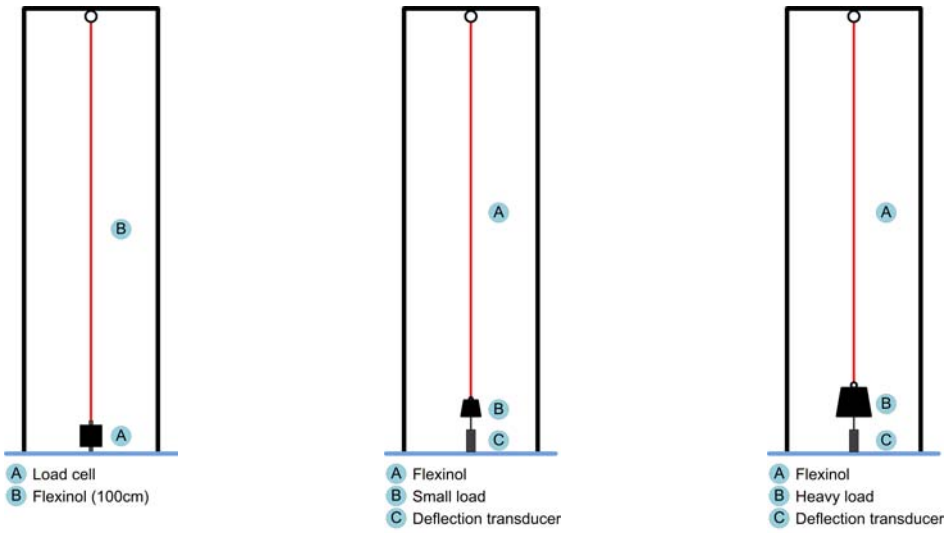
In the fixation test (figure 4.2a), the Flexinol wire is fixated inside the test frame. The wire is fastened directly to the frame in the top, and in a strain gauge load cell in the bottom. When contracting the wire, a force is exerted to the load cell and measured. In this way, it should be possible to determine the maximum pull force of the wire, and how the contraction curve looks in this case. Another interesting

property to investigate is how the contraction changes over time. To be able to use Flexinol or similar products in real life applications, enough knowledge should be available to perform a proper scaling of the wires that act as a part of a bigger system.

4.1.2 Degeneration Test

This setup is designed to examine if a Flexinol wire suffers from degeneration when actuated continuously over time. One setup will be done with a small load (figure 4.2b), and a second setup (figure 4.2c) will be done with a load, slightly smaller than the maximal pull force of Flexinol [37]. The parameter to be examined is the strain rate of the wire, which optimally should not change too much. A wire that changes its working characteristics over time would in many applications lead to the need for a replacement of the wire. This complicates both the application design and the service routines. In the article *Low-mass muscle actuators using electroactive polymers (EAP)* [60], the author contends that shape memory alloys have a life limit of 1000 cycles when actuated to maximal displacement. This should also be investigated.

The contraction rate is measured with two displacement transducers, located underneath the small load and the big load as seen in figure 4.2b and 4.2c.



(a) Fixation. Can Flexinol stand this test, and how large forces will be exerted?

(b) Long term testing with a small load applied

(c) Long term testing with a heavy load applied

Figure 4.2: *Test designs for Flexinol*

4.1.3 Flexinol Antagonist

In this test, two Flexinol wires act together as antagonistic muscles (figure 4.3a). When one wire is fully contracted (agonist), power is cut and cooling of the wire begins. Simultaneously, the second wire (antagonist) contracts, causing a stretching of the first wire. By measuring the displacement and actuation force over time, it can be examined if such an antagonistic use of two Flexinol wires causes degeneration. With this setup it should also be investigated at what maximal frequency the antagonistic muscles can be actuated.

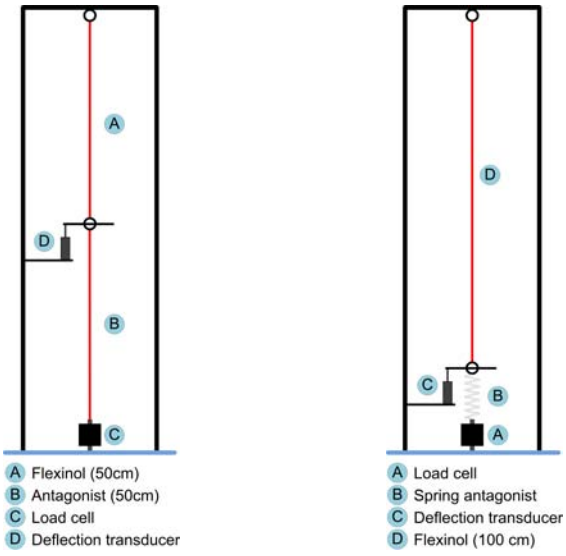
The contraction rate of the degeneration tests is measured with normal linear displacement transducers. When testing antagonists, the center point of the generated motion is located in the middle of the frame and is not easily measured with a linear displacement transducer. Instead, a radial displacement transducer is fastened on the top of the frame. A thread is fastened in the connection point of the two Flexinol wires, pulled over the radial displacement transducer and then fastened in a nut, working as a dead weight to keep the thread straight.

A strain gauge load cell is used to measure the force of each contraction. The forces exerted to the wires when one wire is turned off and the other on, are interesting because they show in what degree the opposite forces of the wires are able to overload eachother.

4.1.4 Spring Antagonist

This test (figure 4.3b) is similar to the test in section 4.1.3, except for the nature of the antagonist. A spring is used as a passive antagonist instead of the active Flexinol antagonist in section 4.1.3. The test frame holds 100cm of Flexinol wire and with 4.5% strain rate, the spring antagonist will be stretched about 4.5cm. It has to be taken into concern, that a certain force is needed to stretch the wire, and therefore the spring should be preloaded. A preloading of the spring will of course be a force working against the wire, but this is the only way to utilize the maximum strain rate of the wire.

The deflection and forces caused by the wire is measured in the same way as for the Flexinol antagonist, described above.



(a) Flexinol wire performing an active antagonistic movement

(b) Spring performing passive antagonistic movement

Figure 4.3: *Antagonistic test designs for Flexinol*

4.1.5 PWM-controlled

As described in section 5.2, the results from the above mentioned tests revealed that the use of an antagonistic Flexinol wire is possible but complicated. The PWM-controlled setup therefore uses a spring antagonist, as described above. When controlling a finger, it is desirable to be able to control each joint to any degree of flexion. To be able to do this, the Flexinol wire has to be contracted to different points between no contraction and full contraction. With most actuators this is an easy task, but as the degree of contraction for a Flexinol wire depends solely on the wire temperature and the stress of the wire, this is a non-trivial task.

Some sort of regulation has to be used in order to control the wire. For the ease of design, the regulation loop should have as few sensor inputs as possible. For a regulation of the wire contraction, two variants will therefore be tested. The first variant uses sensor input from a deflection transducer to control the wire contraction to a predefined set value. The second variant uses current measurements to control the contraction. This variant may be difficult to get working, as the PWM-control signal also causes the current to be pulse width modulated.

With these two tests, it will become clear if it is possible to control the wires as wanted. If the methods are successful, they will also be used to control the humanoid finger.

4.2 Test Frame

In order to be able to measure different physical characteristics of Flexinol, a mechanical test frame is built. The frame is assembled from aluminium parts and is high enough to hold 100cm of Flexinol wire, and wide enough to hold 5 wires in parallel. The aluminium construction is very stiff and guarantees minimal measurement errors due to mechanical deformation. To be able to measure the contraction rate of the wires, additional vertical space was left inside the frame, leaving room for displacement transducers and load cells at the end of the wires. A picture of the test frame is seen in figure 4.4.

Figure 4.4: *Developed test frame. Two radial displacement transducers are seen in the top and to linear displacement transducer are seen in the bottom. The frame holds 5 Flexinol wires in parallel*

4.2.1 Electronics Design

The schematic of the test frame can be seen in figure 4.5. In general, the system can be divided into four parts. Firstly, the control unit of the system is realized with a data acquisition (DAQ) unit from manufacturer Keithley. The model is called Keithley KUSB3100 and has 8 analog inputs with 12bit resolution ($\pm 10V$), 8 digital output ports and other functions that are not relevant. The KUSB3100 has an USB interface and comes with an API for the Microsoft .NET Framework. The DAQ unit controls the contraction of the Flexinol wires in a binary fashion, using its digital outputs. The contraction signals and force signals are then measured, using the analog inputs of the unit.

The second part of the system is the force measurement. The force signals from the load cells are in the range of $\pm 5mV$ and has to be amplified before the KUSB3100 can measure them with its measuring range of $\pm 10V$. The third part of the system is measurement of deflection signals and the fourth part is the driver circuit for the Flexinol wires. The different parts are described in detail in the succeeding sections.

Force Measurements

As described in section 2.5.2, a strain gauge load cell is built up like a Wheatstone bridge (figure 2.29). When a force is exerted to the load cell, a voltage in the range of $\pm 5mV$ is built up between the two outputs of the bridge. To amplify this signal before the KUSB3100 reads it, a differential amplifier is used (figure 4.7). The output of the differential amplifier is given as

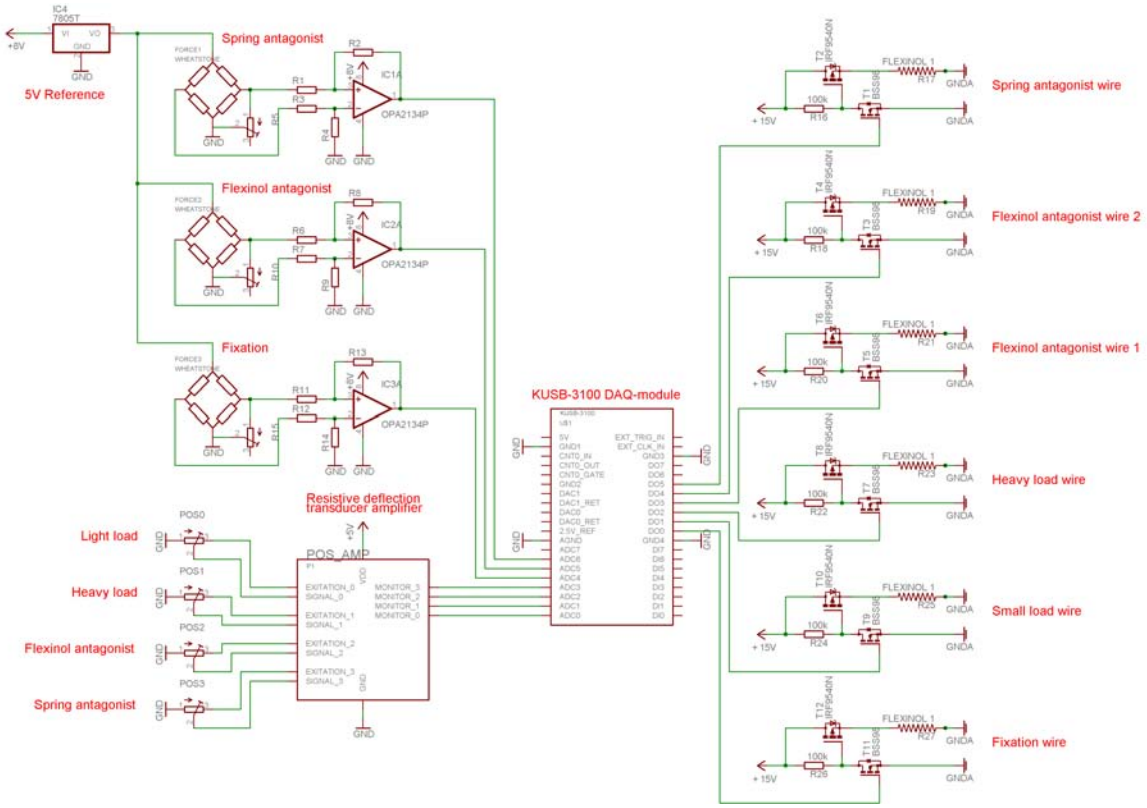


Figure 4.5: *Electrical schematic of the test frame. Overview of the whole system with control unit, force measurements, displacement measurements and Flexinol driving circuits*

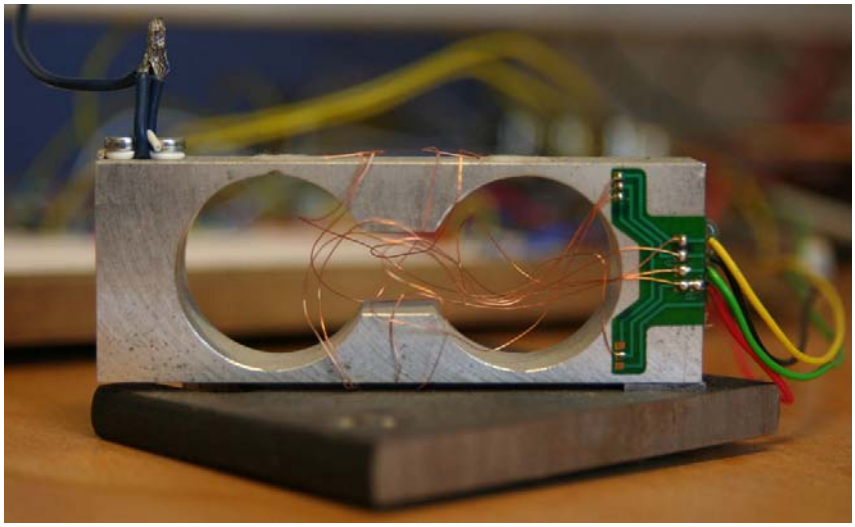


Figure 4.6: *Load cell in use. The four connections points of the Wheatstone bridge are seen on the right side. The loose wire is connected to a Flexinol wire*

$$V_{out} = \frac{R2}{R1} \cdot (V_{in+} - V_{in-})$$

in the special case when $R1 = R3$ and $R2 = R4$. The load cells in use (figure 4.6) are rather cheap units that are specified to about 200N. The load cells are not specified with a given mV/V, but the maximal output will be found when calibrating it. By measuring the mV output of the Wheatstone bridge and expecting a maximum force of 30N, the gain factor is chosen to be ca 6800. This is done by setting $R1 = R3 = 120\Omega$ and $R2 = R4 = 820k\Omega$. An excitation voltage of 5V is regulated by a MAX603 from MAXIM, supplying the Wheatstone bridge. An 8V power supply is used for the operational amplifier. The choice of an 8V supply for the operational amplifier is to make the maximum output signal use a large as possible part of the input range of the KUSB3100. The operational amplifier is an OPA2134 from Burr Brown and was chosen because of its low offset.

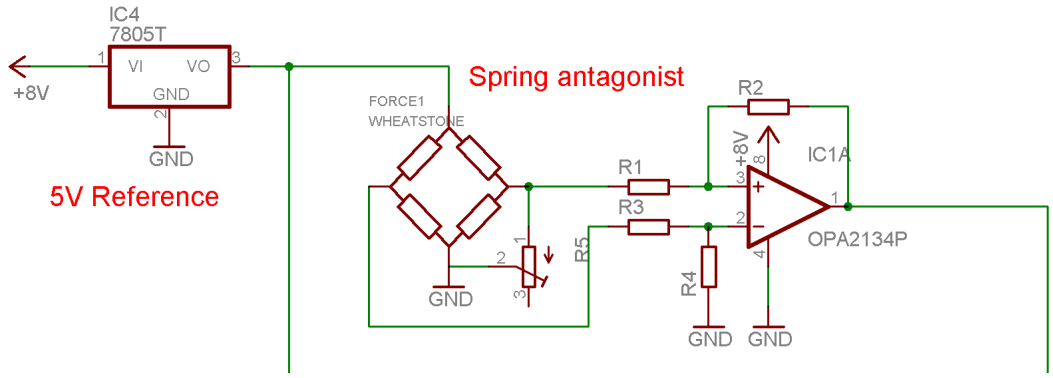


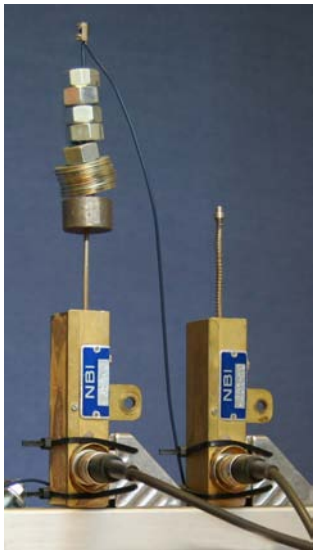
Figure 4.7: *Force signal amplification circuit with potentiometer used for balancing of the load cell*

Measurement of the load cells when unloaded revealed that these suffer from an offset. This results in an bridge output of as much as 5mV when no load is applied to the load cell. The offset is amplified by the differential amplifier, which in the case of a 5mV input creates a theoretical output of ca 34V, which can not be output because the supply voltage of the op-amp is lower. To cancel the offset, the potentiometer $R5$ is added to the circuit. $R5$ has a resistance of $0 - 1M\Omega$ and is connected in parallel with one of the resistors in the Wheatstone bridge. The chosen resistor should be on the side of the bridge with the highest potential. By adding a potentiometer in parallel, the chosen resistor can be lowered, thus adjusting the output of the bridge. With a potentiometer with a maximum resistance of $1M\Omega$, the resulting resistance can be adjusted from near the value of the bridge resistor down to 0Ω and thus, balancing the bridge.

Displacement Signals

For the measurement of linear displacement, a *linear variable differential transformer* (LVDT) transducer can be considered. LVDTs operate with magnetic fields and can be produced to slide with almost no friction. However, they need alternating currents (AC) to operate, which complicates the design. A set of resistive displacement transducers with removable return springs is used instead and are depicted in figure 4.8a. These work as a voltage dividers, making them perfect for applications using mainly direct current (DC). However, some friction might occur while sliding the stem through the housing. The main goal of measuring forces in this setup is to see how the forces change over time, not to make a perfect absolute force measurement. The forces exerted by the displacement transducers are therefore considered to be small enough to be neglected.

For the two antagonist tests, the use of linear displacement transducers is a bit problematic. As mentioned above, two radial displacement transducers are used instead. These are connected in the exact same way as the linear transducers, like voltage dividers and are depicted in figure 4.8b. All displacement signals are connected through a monitor for resistive displacement transducers (figure 4.10). The monitor has four inputs and internally generates the excitation voltage needed. It has the ability to generate an excitation voltage of 2.5V, 5V or 10V, and in this case 10V is used to take advantage of the whole input range of the KUSB3100 DAQ-board. The monitor also has the ability to correct for offset, but this is better done in software to prevent unwanted adjustments.



(a) Two displacement transducers. To the left, the small load can be seen. In the top of the image, the connection point to a Flexinol is visible

(b) Radial displacement transducer. A thread is routed over to wheel to measure displacements

Figure 4.8: *Displacement transducers*

Flexinol Driver Circuit

The last part of the system schematic for the test frame is the driver circuit for the Flexinol wires. The circuit is depicted in figure 4.11 and only has one input through the small signal nMOS transistor T1 (BSS98 [75]). The BSS98 has a maximum gate threshold voltage ($V_{GS(th)}$) of 1.6V and should therefore easily be driven by the 5V digital outputs of the KUSB3100. The drain-source breakdown voltage is at 50V which is more than enough to control the 15V that are used for the Flexinol wire. The drain pin of T1 is connected to the gate of T2, a pMOS hexfet power transistor (IRF9540N [76]), able to control a continuous drain current of 23A. This amount of current will require a heat sink when the transistor is operated, but to control a 100cm of Flexinol wire (ca 15Ω), a current of ca 1A can be expected and thus should the transistor not need a heat sink. The gate of T2 is driven actively low by T1 and pulled up by R16 when undriven. The Flexinol wire is connected to the drain of T2, illustrated as a resistor in the schematic.

4.2.2 Calibration

To ensure correct measurements, the displacement and force setups must be calibrated.

Displacement Calibration

The output from the displacement transducers are assumed to be highly linear, but nevertheless, a calibration routine is used to ensure this. A ruler is used to measure the amount of displacement while the corresponding output voltage is noted. This is done at eleven evenly distributed points between 0mm and 50mm of displacement. The generated table can then be plotted and the linearity can be controlled against a straight line. If the output proves to be linear, a linear gain factor should be calculated, or else it must be decided what polynom degree to use. However, before calculating polynoms, it should be considered whether a non-linearity will cause problems or not in an overall view of the system. For the radial transducers, calibration is done between 0mm and 100mm.

Force Calibration

Strain gauge load cells will react to both positive and negative forces, and the easiest way of calibration when the gauges are situated inside the frame would be to position different weights directly on top of

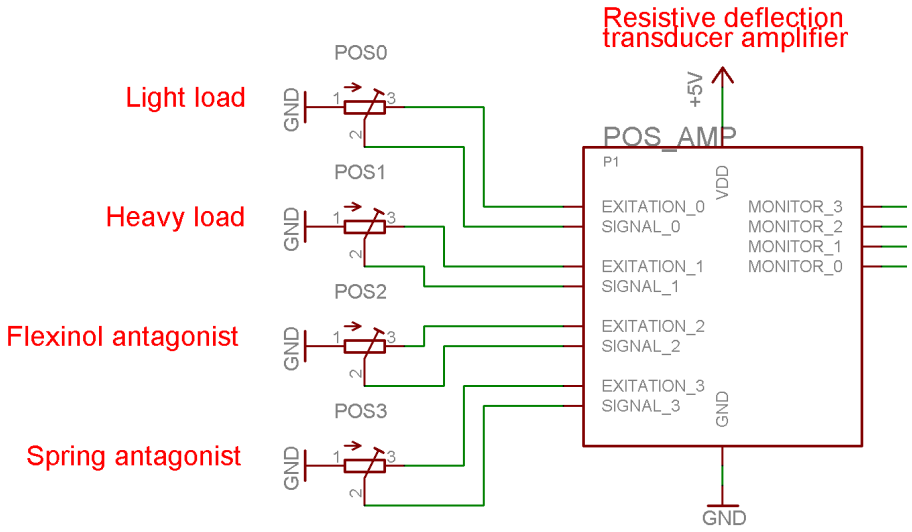


Figure 4.9: *Displacement signal circuit with monitor that generates the needed excitation voltage*

Figure 4.10: *Monitor for resistive displacement transducers*

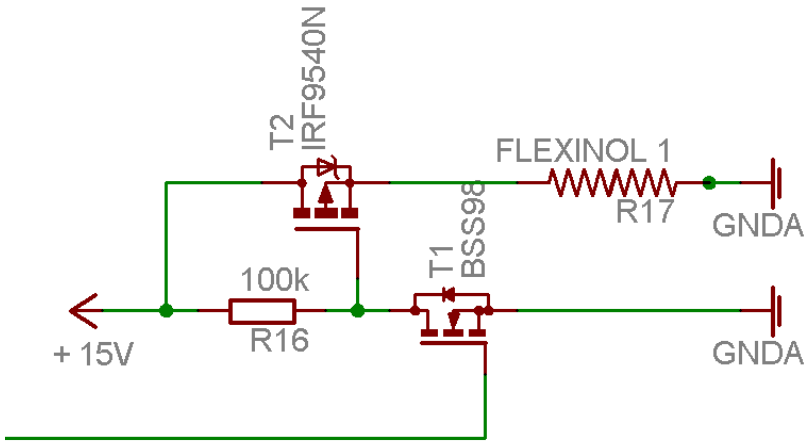


Figure 4.11: *Flexinol driver circuit. A small signal transistor makes the circuit suitable for the DAQ-unit*

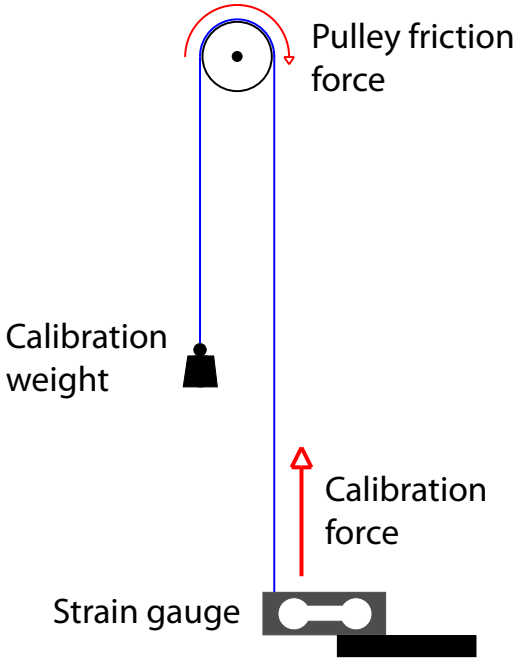


Figure 4.12: *Calibration setup for force measurements. The pulley introduces a small error in the force measurements*

the load cells. However, the output may not be exactly proportional when calibrating this way. Instead, the wheel of a radial displacement transducer is used as a pulley. A thread is fastened in the load cell, pulling in upward direction. The thread is then pulled over the wheel as depicted in figure 4.12. This way of calibrating will contain some error caused by friction when rolling the wheel of the radial displacement transducer as illustrated in the figure. However, this error is considered small enough to be neglected. As already mentioned above, the main goal of the force measurements is to uncover relative changes over time. To investigate the linearity of the load cells, calibration weights between 100g and 1500g with 100g steps are used (figure 4.13). Analogous to the displacement calibration, a linear gain factor is calculated if possible.

Figure 4.13: *Calibration weights. Combinations of the weights are used to calibrate at 100g steps*

4.3 Test Software

For the purpose of testing Flexinol wires, two programs are developed, one for controlling the test frame and one web application for remote surveillance.

4.3.1 Software for the Test Frame

For signal control, data collection and graphical presentation, a measurement program is written. The program is developed in Microsoft Visual Studio 2008 and is written in the programming language C# for the Microsoft .Net Framework Runtime. A screenshot of the user interface can be seen in figure 4.14. The user interface consists of a set of tables continuously showing the last values for input and output, while the graphs show the same information graphically over time. The program has a simple menu where the measurements can be started and stopped, and while the program is running, a new setup for output values can be loaded (described later). Figure 4.15 shows a block diagram of the program architecture. Each block is described in the next sections.

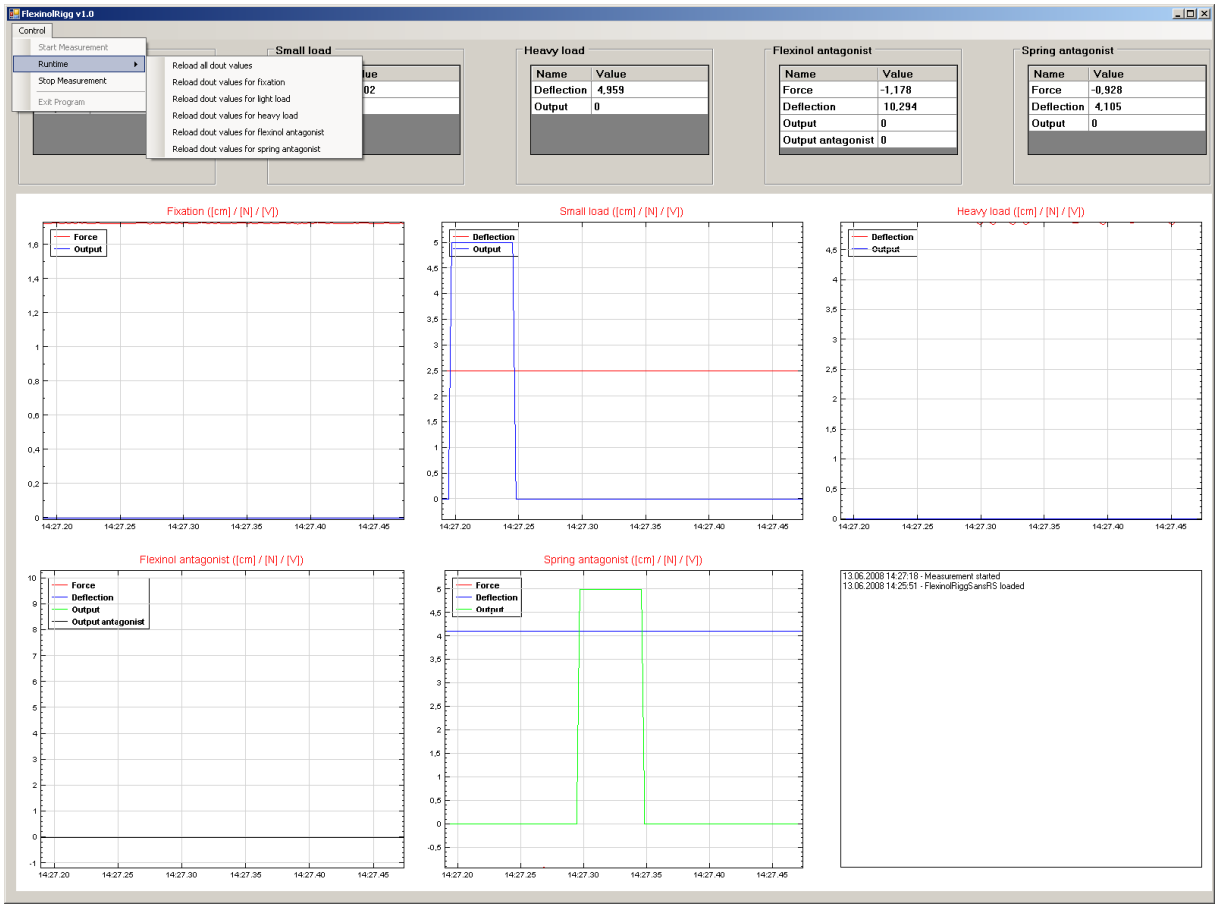


Figure 4.14: *User interface of measurement software. Measurement values are seen in the top, plots in the bottom of the screen*

Data Acquisition (DAQ) Module

The data acquisition module is responsible for communication with the KUSB3100 USB-device [69, 77]. The device is divided into so called subsystems, one for each integrated function. In this module, analog inputs and digital outputs are handled. Therefore, the two subsystems *AnalogInputSubsystem* and *DigitalOutputSystem* are used. The digital output system is used at its simplest form by outputting one byte of data. The 8 bits in the byte represent the digital state of the output channels. The control of the digital outputs is described in the next section.

The analog signals from the test frame may contain some noise from the surroundings - mechanical vibrations as well as electrical noise. To prevent this from influencing the measurement results, a simple mean filtering is used. When measuring analog values, the KUSB3100 can be used in either single mode or continuous mode. In continuous mode, the *AnalogInputSubsystem* is first configured with a conversion frequency, a channel list and a data buffer. The conversion frequency is the raw frequency of the digital to analog converter. If the subsystem is configured with a frequency of 1000Hz, a channel list with one

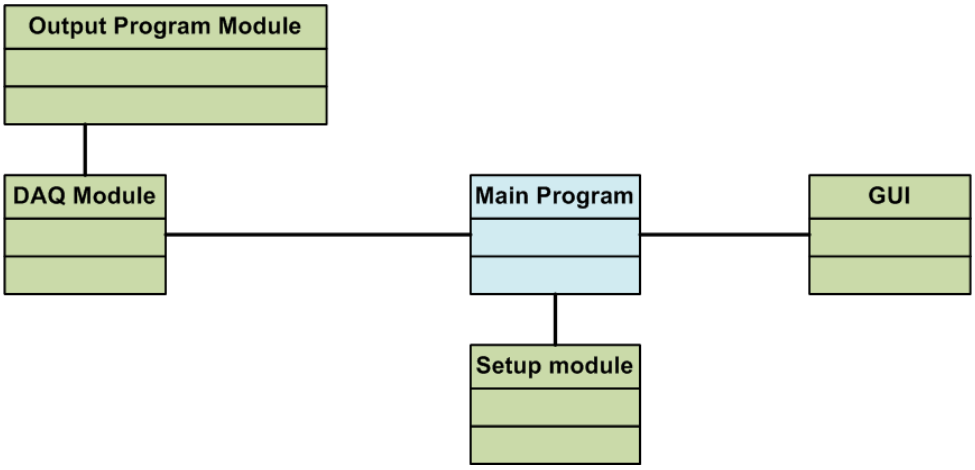


Figure 4.15: *Block diagram of the program architecture. DAQ module communicates with the hardware and the main program handles inputs from the GUI*

channel and a buffer for 1000 samples, the resulting buffer will contain 1000 samples of the specified channel with 1ms delay between samples. However, if two channels are specified and the same frequency and buffer is used, the resulting buffer will contain 500 samples of each channel with 2ms delay between samples.

When the DAQ-module receives an analog measurement request, the subsystem is configured with a suitable frequency, channel list and buffer. After the measurement is started, the module waits until the buffer is filled, and then the mean value of the samples is calculated and returned.

Output Program Module

The program is supposed to control the Flexinol wires mounted in the test frame continuously for several weeks. It must be expected that during this period, single wires will have to be replaced. It may also be necessary to change the output signal before mounting a new wire. To be able to perform this on the fly, without stopping the program, an output program module is developed. The output program tells the DAQ-module what control signals to output to the different Flexinol wires at all times. Each output has its own array in the program module. Each time the DAQ-module sends a request to the output program module, the next value in the array is returned. The module can be replaced at runtime through the user interface.

```
1 [dout_tick]=1000
2 [wire_0]=10*false;5*true;20*false
3 [wire_1]=15*false;5*true;15*false
```

The code above shows an example of an output program for two wires. The first line tells the DAQ-module how often it should request a new output value, in this case 1000ms. Wire 0 will in this case have a low output for $10 \cdot 1000ms = 10s$, then a high output for $5 \cdot 1000ms = 5s$ and finally a low output for 20 seconds. Because the driving of each Flexinol wire requires about 1A of current, it will be desirable to not have all wires contract at the same time. The output waveform of wire 1 from the above code will be exactly equal to that of wire 0, with a delay of 5 seconds. When wire 0 turns off, wire 1 turns on, resulting in a 10 second period of 1A current instead of 5 seconds with 2A current. With only 2A of current flowing this will normally not be a problem, but with five wires driven at the same time, a current of 5A would be expected. 5A is the upper limit for continuous current from the laboratory power supply in use. When operated continuously for a longer period of time, this could cause problems because of temperature variations in the surroundings.

Setup Module

To be able to change parameters for the program such as gain factors, mapping of outputs etc. without having to change the program code, a setup module is developed. The setup module opens a text file

with lines formatted as

```
[key]=value
```

and stores these key-value pairs in a hash table. By providing the name of the key and the data format, the value corresponding to the key is returned. A global setup file is loaded on program startup, containing names of the five wires, corresponding output and input channels and gain factors. The output program module mentioned above also uses the setup module to read the file containing the output values.

Graphical User Interface

The graphical user interface (GUI) was briefly presented above and is depicted in figure 4.14. The plotting of the signal data is done with the free scientific charting library NPlot (www.nplot.com). NPlot is a large library which supports many different types of plots, but is unfortunately not so well documented. A few tutorials are available from the website, but the class descriptions are very short and lack examples. However, compared to other free plotting libraries like Chart FX Lite (www.softwarefx.com/SFXNETProducts/CFXLiteforNet) which contain limitations like maximum one data series, NPlot is considered to be a very good free solution. A small wrapper class is developed to hold track of the data series to be plotted with NPlot. The wrapper class has a list of Y-data for all data series and a list of timestamps for the X-values. Each time the wrapper class gets new data, it deletes all data that is older than the amount of seconds specified in the setup file. Then the new dataset is passed to NPlot and displayed. In this way, a charting functionality is created.

4.3.2 Web Application for Remote Surveillance

As mentioned above, the testing of Flexinol wires has a duration of many weeks and during this time, broken wires and other challenges are expected to occur. In order to remotely monitor the testing, a web application is developed. The main menu of the web application can be seen in figure 4.16 and consists of a webcam picture, links to data from the five different wires and a survey of all wires.

The webcam picture is captured by the open source webcam capture program Dorgem, which is available from http://dorgem.sourceforge.net. It has a built in webserver listening on a selectable port, in this case port 8080. The webpage itself only contains an image reference to the url http://localhost:8080 and each call to this address causes Dorgem to return the latest captured image.

Each of the data links in the main menu lead to a list of available measurement data from the corresponding wire, as seen in figure 4.17a. A new data file is created every day, and each file is available through a link to a page that displays detailed information from the file. This page can be seen in figure 4.17b and shows a couple of controls at the top and a plot of the selected data. Alternatively, the data used to create the plot can also be displayed underneath it. The main goal of the web application is to be able to browse the measurement data, going back to see for example when a wire snapped. By using the controls at the top, a section of the file can be extracted and displayed. The first line of controls enables the user to choose any part of the file to display, alternatively only show every nth line of the selected section. The second line of controls makes it possible to display the n last lines of the data file.

Because special user rights are desirable, the computer used for measurements is not a part of the wired network at the University of Oslo (UiO). It therefore doesn't possess a static IP address and instead, the address http://flexinoltesting.dyndns.org was assigned to the computer through the dynamic DNS provider www.dyndns.org. This makes the webserver easy to reach from inside the UiO network or via VPN.

4.4 Humanoid Finger Design

This section focuses on the design of a humanoid finger, actuated with Flexinol wire. The design is influenced by some principles from the human anatomy, but is also an attempt to create a finger that can be actuated with Flexinol wires without making the design too complex. The thoughts of how too design the finger is highly influenced by the results from the testing of Flexinol (section 5.2) that revealed some limitations and challenges. All presented 3D models are developed in the CAD software SolidWorks.

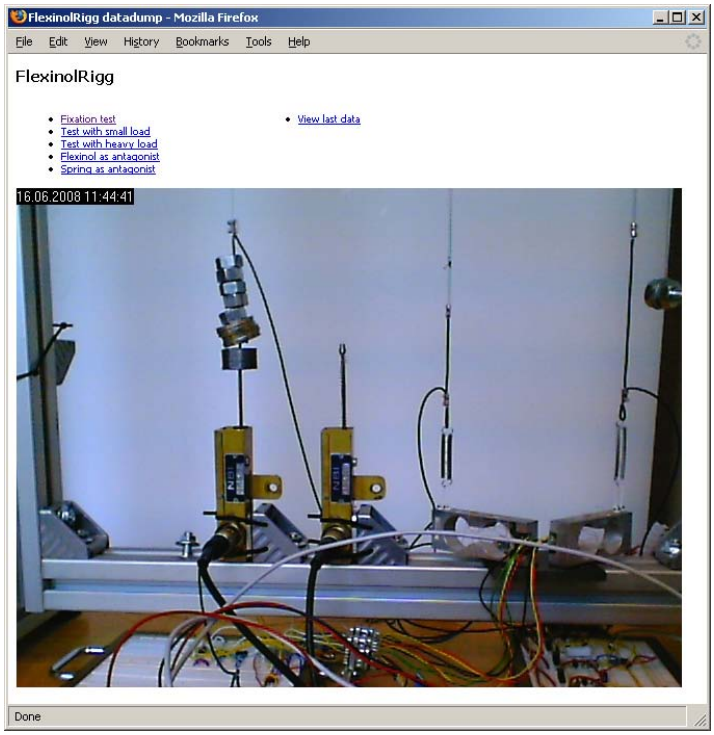
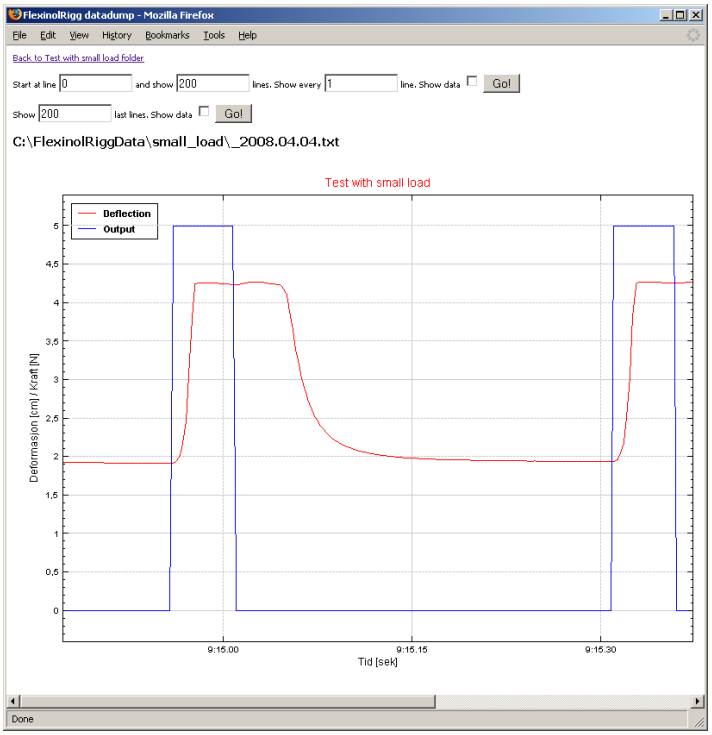


Figure 4.16: *Menu for web surveillance. Web camera picture is updated each time the page is loaded*

(a) Listing of available data. Filename, file size and the last modification date are shown in the listing



(b) Presentation of one single data file from the listing. By using the controls on the top of the page, different parts of the data file can be inspected

Figure 4.17: *Data view from web surveillance*

4.4.1 Anatomical Model

The presented finger is partially built up like the human finger. Mechanically, it consists of four phalanges connected with three hinge joints. The joint between the metacarpal phalanx and the proximal phalanx is able to perform adduction and abduction by humans in addition to flexion and extension. Adduction and abduction is not a part of this mechanical design to keep it as simple as possible. If a good controlling of flexion and extension is achieved, this should be easily adaptable to adduction and abduction in later designs.

As presented in section 2.1, the human anatomy uses muscles to create motions and tendons to transfer the motions. Tendon routing is done highly sophisticated in the human hand, causing for example that a flexion of the DIP-joint will automatically result in a flexion of the PIP-joint. This design makes the human hand highly suitable to grasp around objects. A good control of the contraction of the Flexinol wires would make such a design possible, but if not, it could make the finger impossible to control. This function of the human finger was therefore not used.

The extension of the finger is done in the same way as in the human hand. A single extensor tendon is straightening the joints on the upper side of the finger. The size of the finger segments are based on measurements of the hand of a 24 year old male.

4.4.2 3D Design

The 3D design is an attempt to combine the anatomical aspects from the last section with the abilities and limitations of the 3D printer Dimension SST 768. Figure 4.18 shows the end result of the designed parts. As the figure shows, most edges in the design have been rounded. This is done because straight edges tend to be rather weak when printed in ABS-plastic. Each finger segment has a height and a width of 4cm and a varying length. As mentioned above, the finger sizes are based on measurements of a human hand. To make a mechanical analysis easier, the finger is designed in a ratio of 3:1 to the measures. The oversized design also makes more space available for tendon routing.

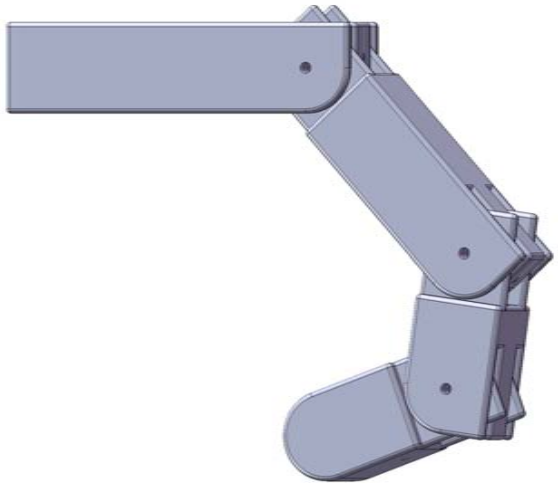


Figure 4.18: *3D design of a humanoid finger. The finger is designed in SolidWorks and then printed in ABS plastic*

Metacarpal Phalanx (MCP)

The metacarpal bone of the human hand is inside the palm and therefore doesn't appear as a part of the finger. In this design the metacarpal bone is used as a base for the hole finger construction and is depicted in figure 4.19. On the back end of the finger segment, four holes can be seen in the top view of the drawing. These are for mounting the finger and are positioned in the back to prevent obstruction of the finger movement. The joint at the other end of the segment is the first finger joint, namely the metacarpal phalangeal joint (MCP-joint). As can be seen from the top and bottom view in the drawing,

two slots are cut in from the end of the finger segment. These two slots will be connected with the next finger segment as described later. The radial part of the joint can be seen in the side view and ensures that the joint will be able to move freely over 90°. As described later, the corresponding end of the connected joint is also designed to fit exactly with the shape of the metacarpal phalanx, ensuring exactly 90° of movement.

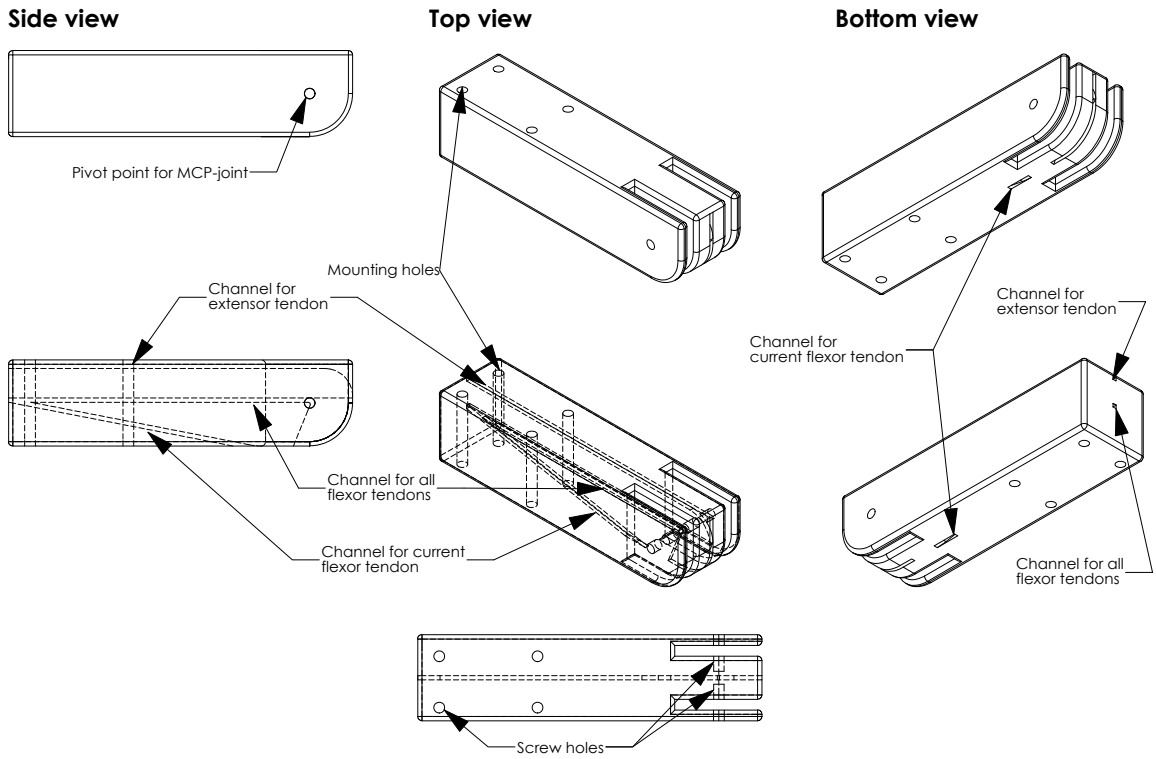


Figure 4.19: *Drawings of the metacarpal phalanx from different angles. This is the first finger segment and is fastened to the forearm*

As mentioned above, the human finger is designed in such a way that if the DIP-joint of the finger is flexed, this will also cause a flexion in the PIP-joint. To prevent this kind of movement in the 3D model, the tendons will be routed through the pivot point of each joint. When for example the end joint (DIP-joint) of the finger is flexed, the corresponding tendon will be tightened, exerting forces on each joint it is routed through. By routing the tendon as near as possible to the pivot point of the other joints, the exerted forces will be minimal, causing movement only in the correct joint. To make room for such a construction, the center of each joint has to be kept free. As can be seen in the lowest part of figure 4.19, this is done by splitting the joint axle in two parts, each one entering from its own side of the joint. The two axles are normal M5 screws with a length of 16mm. The outer screw holes have a diameter of 5mm, allowing the screws to slide in without much friction. The inner holes have a diameter of 4.8mm, which is small enough to fasten the screws.

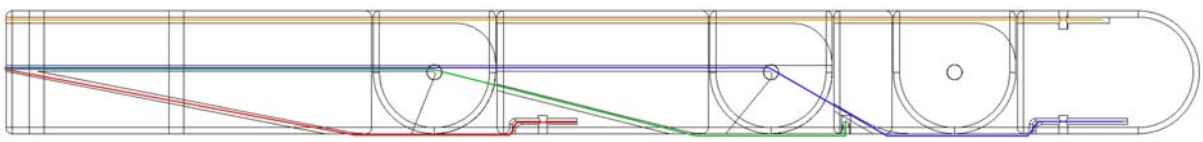


Figure 4.20: *Tendon routing inside the finger. The tendon is routed through the pivot point of the joints, creating minimal torque*

Figure 4.20 shows the routing schematic for the hole finger. The routing channels can also be seen from different angles in figure 4.19. The channel for the current MCP-joint starts at the back end of the MCP, where all three flexor tendons enter the construction. Two tendons (green and blue) are routed straight through the segment, through the pivot point and into the next finger segment. The current tendon (red) is routed diagonally to the bottom of the finger segment and then along to the next segment where it is connected. If a pulling force is exerted to this tendon, the result will be that the connection point of the tendon on the proximal phalanx will be pulled against the MCP causing the joint to flex.

At the top of the routing schematic, a channel for the extensor tendon can be seen. This is just one single tendon that is routed through all the segments of the finger and connected with the last one. To prevent damage to the extensor tendon, too much friction forces and unwanted forces when a joint is flexed, this channel is cut to align concentrically to the pivot point of each joint.

Proximal Phalanx

The proximal phalanx (figure 4.21) is the first finger segment with a joint in each end. The MCP-joint only consists of two plates, fitting exactly into the slots of the MCP. A 5mm hole through the pivot point of the joint allows the screws mentioned in the last section to be inserted. The other end of the proximal phalanx is the PIP-joint and is built up with two slots, exactly like the MCP-joint of the MCP segment. As figure 4.20 shows, the first flexor tendon (red) is connected to the lower side of the proximal phalanx. This is done by making a horizontal channel for the tendon as figure 4.20 and 4.21 show. A vertical hole with a diameter of 2.8mm is then cut through the tendon channel. This will allow a 3mm screw to be mounted into the hole, locking the tendon in place.

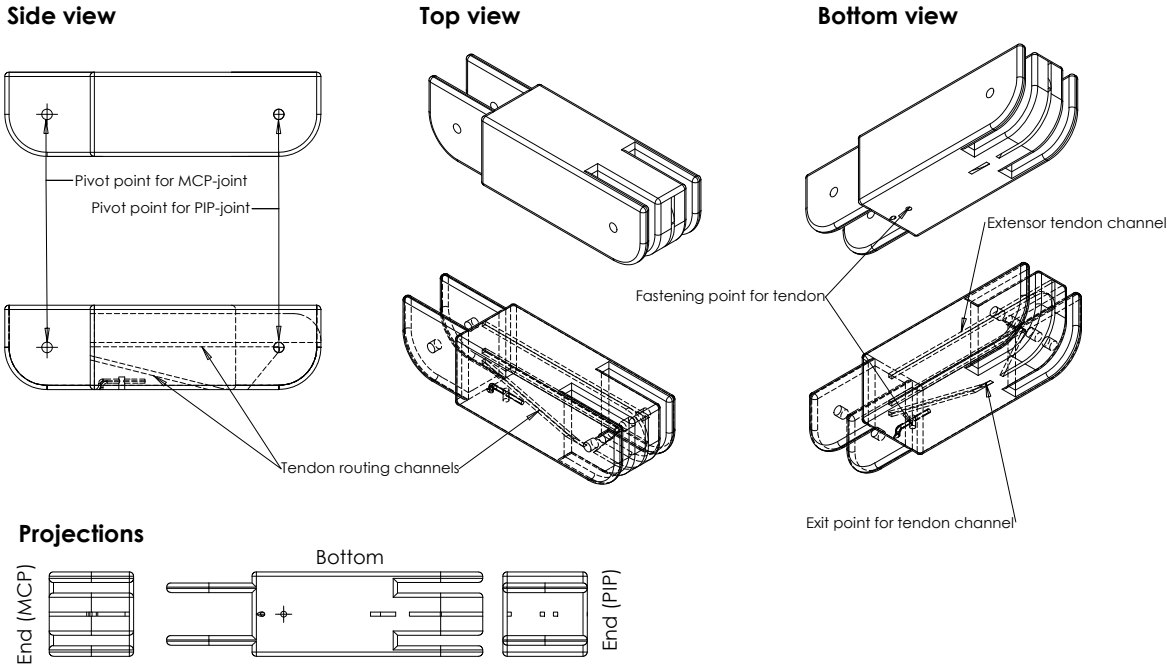


Figure 4.21: *Drawings of the proximal phalanx from different angles. This is the second finger segment*

The proximal phalanx is a bit shorter than the metacarpal phalanx. As figure 4.21 and 4.20 show, this results in a shorter and steeper routing channel for the current flexor tendon (green). To prevent corners in the routing channel, the channel is designed to be orthogonal to the pivot point of the MCP-joint. In practice, this means that the tendon will always go in a straight line from the pivot point to the lower side of the proximal phalanx. The projection view of figure 4.21 clearly shows where the MCP flexor is connected and where the PIP flexor exits. As with the metacarpal phalanx, the proximal phalanx also has a tendon channel for the extensor mechanism in the top of the construction.

Intermediate Phalanx

The intermediate phalanx is constructed very much like the proximal phalanx. The only difference is that the intermediate phalanx is even shorter, making tendon routing a challenge. As figure 4.22 shows, the tendon channel is very steep, leaving almost no room for a tendon fastening mechanism. To be able to fasten the tendon, a channel is constructed as for the proximal phalanx. However, this channel cannot be cut in the longitudinal direction because it would then collide with the other tendon channel. The fastening channel is therefore cut latitudinal, leaving enough room for both the fastening screw and the tendon channel.

As figure 4.20 shows, the tendon (blue) that is routed through the intermediate phalanx is the last flexor tendon, responsible for flexing the DIP-joint. The extensor channel is routed through the top of the phalanx as for the two previous finger segments.

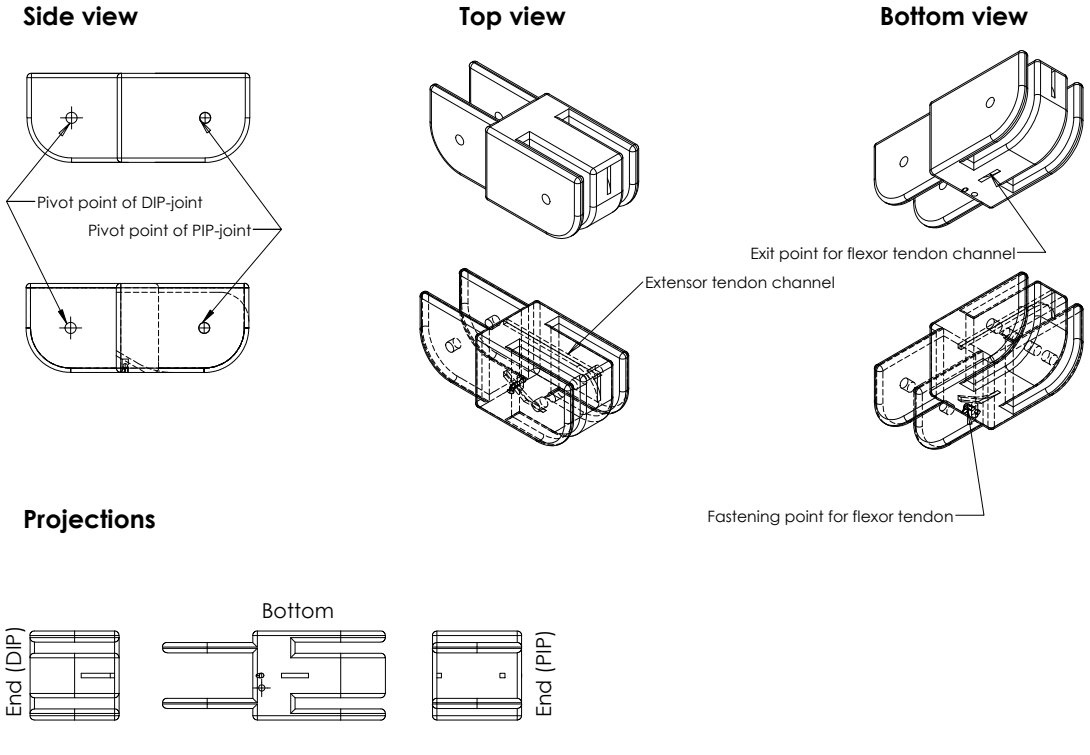


Figure 4.22: *Drawings of the intermediate phalanx from different angles. This is the third finger segment*

Distal Phalanx

The distal phalanx is the last finger segment and therefore differs from the others. Figure 4.23 shows drawings of the distal phalanx. The PIP-joint of the segment is built up exactly as described for the other finger segments with two plates fitting into the other side of the joint. The other side of the phalanx is rounded, making it look like the finger tip of a human finger. Both on the upper and lower side of the finger segment, fastening mechanisms for tendons are situated. The upper one is for the extensor tendon, and has its entering hole inside the PIP-joint. A screw hole is cut to fasten the tendon. Analogous to the proximal phalanx, a tendon channel is cut in the longitudinal direction, assuring a tight fastening of the tendon.

4.5 Humanoid Finger Application

As already mentioned, the above presented design is prototyped with a 3D printer. The resulting finger, produced in ABS plastic, now has to be assembled and put into a finger application. In general, the

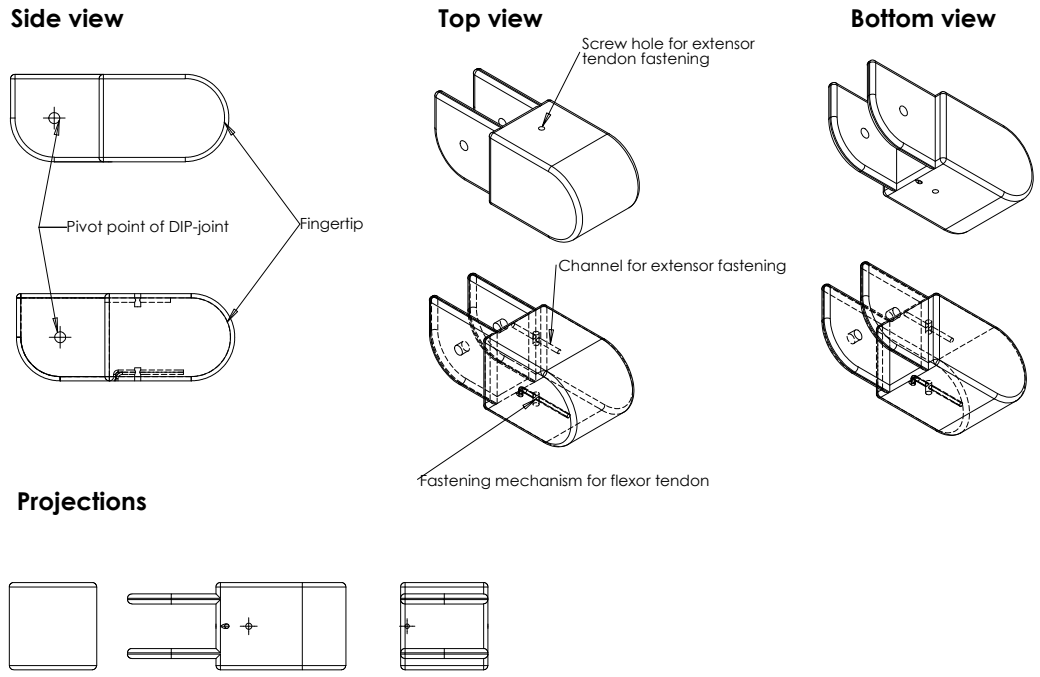


Figure 4.23: *Drawings of the distal phalanx from different angles. This is the tip of the finger*

application consists of three main parts. The main mechanical part is the finger which is supplied with tendons in the form of the fishing line FireLine from Berkley (www.berkley-fishing.com). The finger is fastened to the second part of the application which is a rectangular aluminium tube, acting as a forearm, holding three Flexinol muscle fibers, an extensor return spring and sensors for displacement and force. Finally, an electrical circuit controlled by an Atmel ATMega32 is used to control the muscle fibers, read sensor data and communicate with a computer. Figure 4.24 shows a picture of the finger mounted on the forearm.

4.5.1 Mechanical Design

The base of the mechanical design is a rectangular aluminium tube. The tube is used to wire sensors and power supply, keeping them away from the hot Flexinol wires which potentially could melt the wire isolations, causing short circuits. The aluminium tube could have been used as electrical ground, but this was considered too hazardous. If a wire would break near to the end with the highest potential, and the broken wire would touch the tube, large currents would be expected ($1\Omega \Rightarrow 15A$). This could potentially overheat and damage some parts of the electronics.

Elastic Displacement Transducer

A first attempt to measure the displacement of the muscle fibers is done with an elastic displacement transducer. The principle of this measurement is depicted in figure 4.25. The elastic transducer has a resistance that changes when the transducer is stretched. It is fastened to the joint between the tendon and the Flexinol wire in a stretched state. When the Flexinol wire is contracted, the joint moves to the right, shortening the transducer. By including the transducer in a voltage divider, a voltage proportional to the amount of contraction can be measured. However, as the results in section 5.4.1 shows, the measurements are not stable, making the transducer unsuitable for this application.

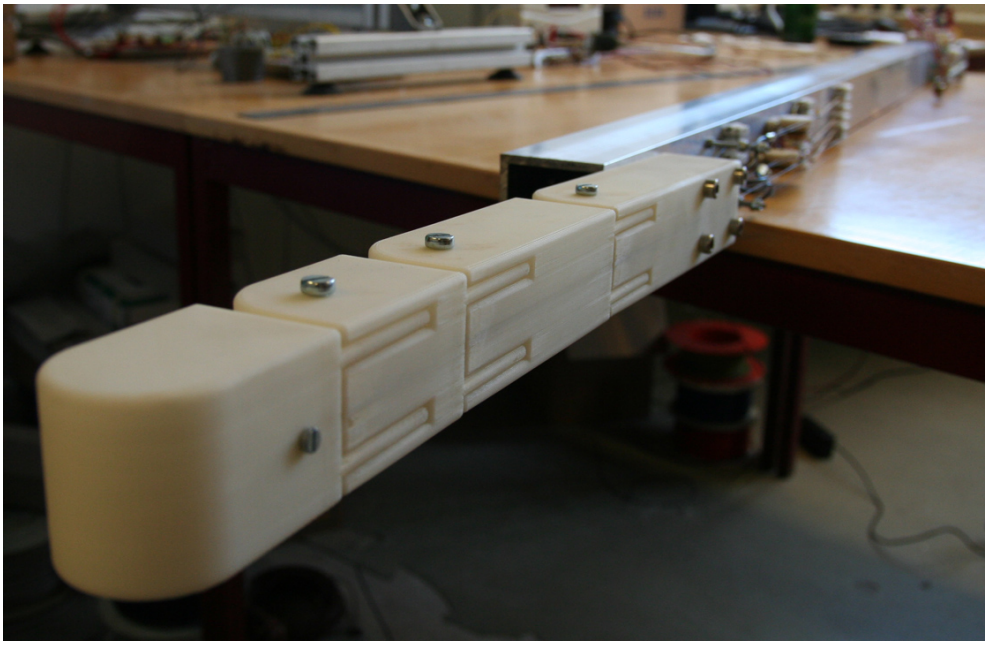


Figure 4.24: *Assembled finger mounted on an aluminium forearm. The finger segments are connected with normal machine screws*

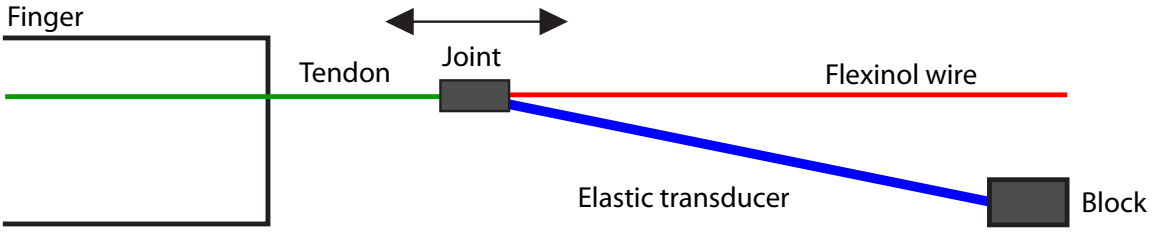


Figure 4.25: *The elastic displacement transducer (blue) changes its resistance when it is stretched*

Radial Displacement Transducer

To overcome the problems of the elastic displacement transducer, a small radial transducer is developed instead (figure 4.27). The transducer is based on a standard one turn trimmer potentiometer which is easily soldered onto a PCB strip board with 2.54mm grid. A bracket is designed to fasten the strip board in a upright position on the aluminium tube. A principle schematic can be seen in figure 4.26.

The expected displacement from the Flexinol wire is about 4.5cm, and to ensure that the transducer can handle this, a wheel with a diameter of 2cm is designed. This makes one turn on the potentiometer equal to a length of $\pi \cdot d = \pi * 2cm = 6.28cm$. A first version of the wheel is depicted in figure 4.28a. The tap in the center of the wheel is mounted inside the potentiometer and then fastened with a screw. The tap appeared to be the weakest structure of the printed part, so a second version of the wheel is developed (figure 4.28b). Instead of the tap, a screw is fastened through the potentiometer, directly into the wheel. A washer is used to separate the wheel from the body of the potentiometer.

The potentiometer is wired as a voltage divider, causing its output to span from 0V to V_{dd}. This leads to a better resolution than the elastic transducer, that would need a resistor in series towards ground. The transducer wheel was developed in SolidWorks and printed on the 3D printer.

Bracket for Force Measurements

The strain gauges used for force measurements in the test frame are too large in size to be fitted into the finger application. Instead, a pressure sensor is used (section 2.5.2). The sensor can be seen in figure 4.29 and is mounted together with a bracket at the end of a Flexinol wire. The bracket is designed in 3D and printed on the 3D printer. It consists of two base parts, each with a hole through the center to

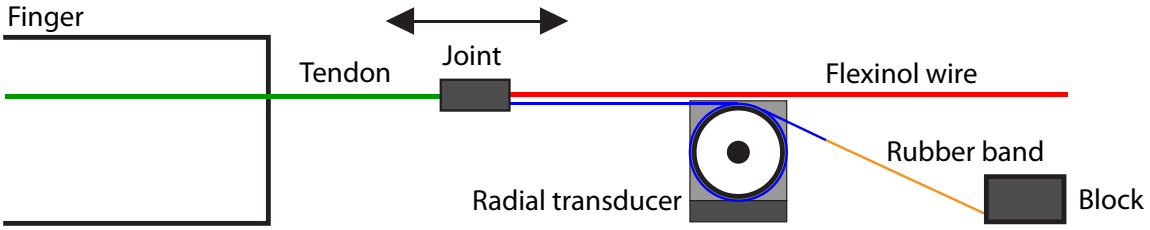


Figure 4.26: *A thread is used to rotate the radial displacement transducer. The thread is returned by a rubber band*

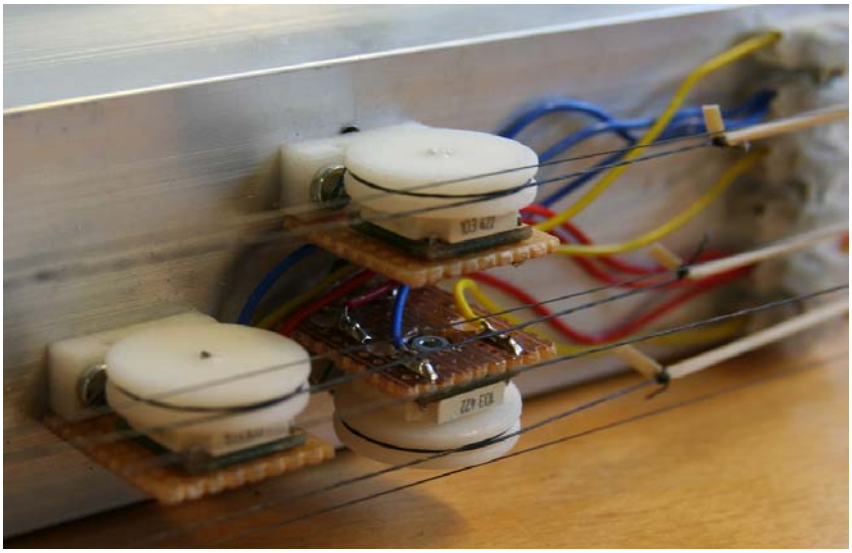
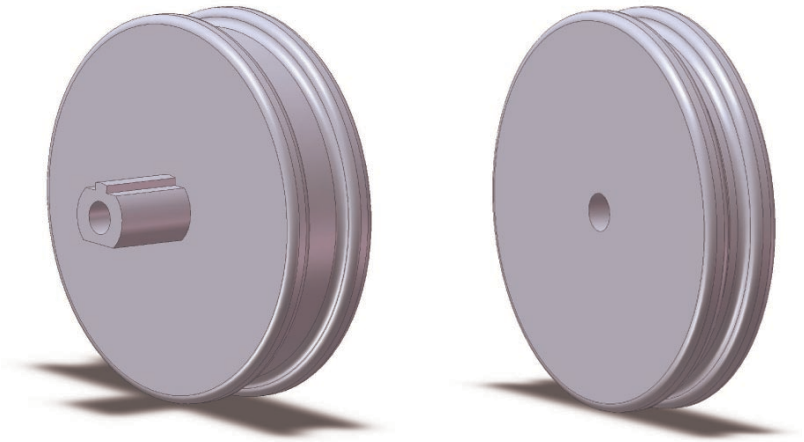


Figure 4.27: *Radial displacement transducer mounted on the forearm. Rubber band and Flexinol wires can also be seen*

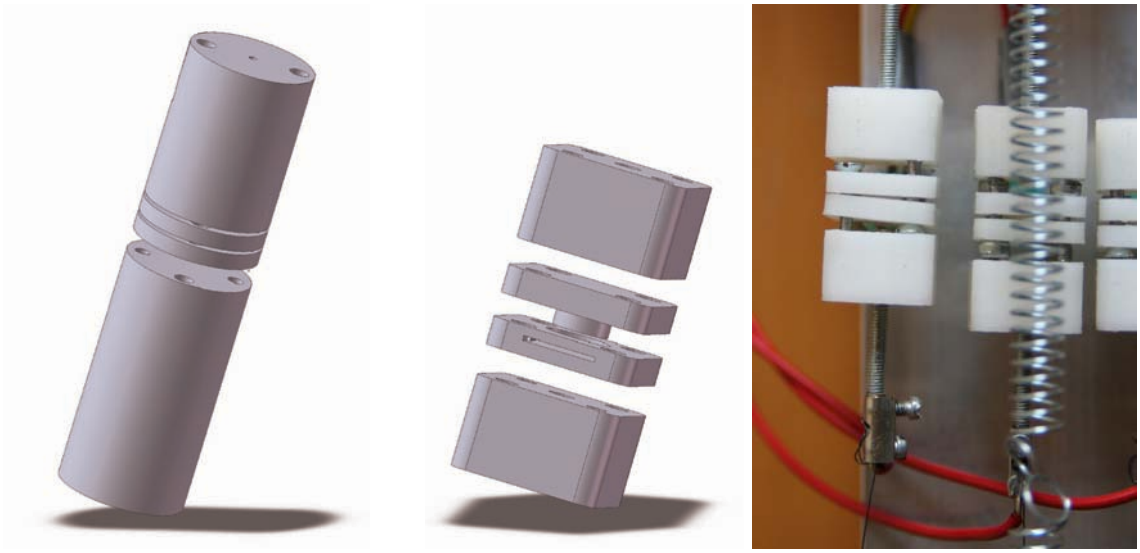


(a) Version 1. The center tap proved to be to weak when printed in ABS plastic

(b) Version 2. The wheel is held in place by a screw instead of the tap

Figure 4.28: *3D model of radial displacement transducer wheel*

fasten screws. The base also has one hole in each corner. Two of them are for fastening screws and two of them can be used with a screwdriver in order to fasten screws in the other base part. The bracket also has two washers between the bases. One of them has a slot where the pressure sensor can be inserted. The pressure sensor is then held in place after the screws have been mounted. The last washer is pressed against the pressure sensor when the base parts are forced away from each other and thus, making a force measurement possible.



(a) Version 1. The circular design left little room for mounting screws.

(b) Version 2. The rounded rectangle shape allows mounting of screws in the corners.

(c) Printed bracket is fixated in one end. When the Flexinol wire is tightened (lower part of image), a force can be measured.

Figure 4.29: *3D model of force bracket*

Force Calibration

The pressure sensor changes its resistance when a weight is pressed against it. To calibrate the pressure sensor, the force bracket is mounted inside the above mentioned test frame. A hook is fastened to the lower end of the bracket, allowing different weights to be hung on it. By applying weights in the span from 100g to 1500g with steps of 100g a good calibration curve is achieved.

4.5.2 Electrical Schematics

Figure 4.30 shows the electrical schematic drawing for the finger application. The control unit of the system is represented by an ATMega32 microcontroller. The microcontroller controls the Flexinol wires that are driven with the same circuit as depicted earlier in figure 4.11. A 5V voltage regulator supplies the microcontroller and is used as excitation for displacement and force measurements. A 6 pin ISP connector is used as programming interface for a computer running the development tool AVR Studio from manufacturer Atmel.

4.5.3 Communication

The finger application is designed to be a remotely controlled stand-alone system. As can be seen from the schematics in figure 4.30, a MAX202 driver chip is connected to the ATMega32. The MAX202 is needed to convert the 0V-5V TTL signal levels from the microcontroller to RS232 levels of $\pm 12V$. The driver (figure 4.31) uses its 5V supply voltage together with a charge pump to build up $+12V$ over an external capacitor. This voltage is then inverted with another charge pump to provide $-12V$. The TX- and RX-pins of the ATMega32 are connected to $T1_{in}$ and $R1_{out}$, correspondingly.

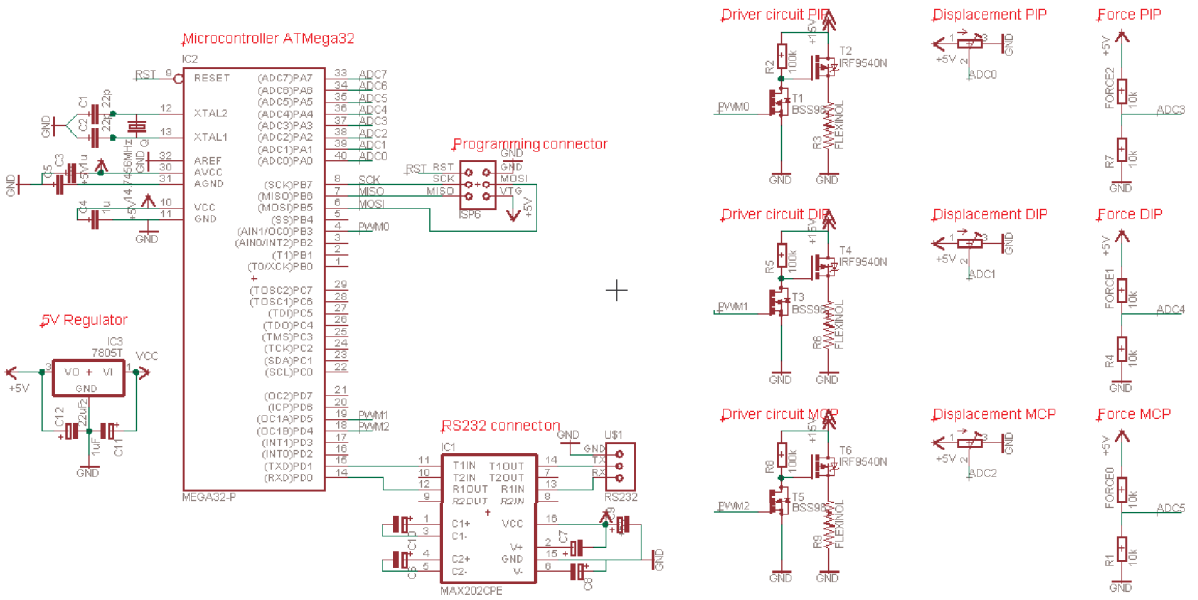


Figure 4.30: *Electrical schematic for the finger application. The ATMega32 microcontroller is used for measurements, computations and output signals*

The MAX202 is connected to a DB9 connector with a minimal connection (ground, rx and tx), which is sufficient for this application. A computer can then be used to connect to the finger application via RS232.

Figure 4.31: *Maxim MAX202 level shifter for serial communication between ATMega32 microcontroller and host computer[78]*

4.5.4 Microcontroller Program

The ATMega32 is controlling the Flexinol wires and reading sensor inputs. For these tasks, a microprogram is needed. Programs for the ATMega32 are normally written in C and compiled with the free WinAVR compiler. Some prefer to write programs directly in AVR assembly, but this solution should only be chosen if the programmer needs control over every clock cycle, as assembly has a much lower readability and scalability than the C language. A block diagram of the program can be seen in figure 4.32. A command interpreter is developed to provide a flexible remote interface. The implemented interpreter is presented in the following section.

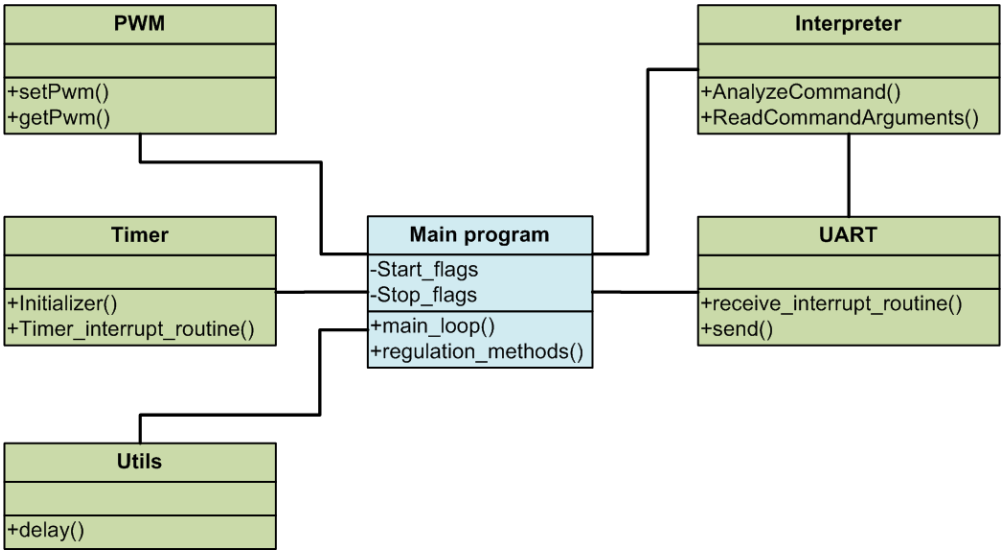


Figure 4.32: *Block diagram of the micro program. Reception of commands is done by the UART module, analyzed by the interpreter module and executed in the main program*

Command Reception

As already mentioned, the program receives commands over the UART interface. The reception of data can occur occasionally and should therefore be handled in a suitable way. There are in general two ways to check whether a data byte has been received. The first method is to check the status register of the UART regularly. This method applies large limitations to the program, as the UART has to be polled continuously to avoid losing data bytes. Instead, the reception of data is done with interrupts. A buffer is implemented as an array, capable of holding 50 bytes of data. The interrupt service routine puts the received data byte in the first free position of the buffer each time an interrupt is triggered. At the same time, the interrupt service routine checks if the received data byte is a newline character. If this is the case, a complete command has been received and has to be analysed.

Command Analysis

The analysis of a command is performed by the interpreter. The interpreter consists of two main functions. The first function is used to determine which command is present. This is done by comparing the first characters of the received buffer to the different commands supported by the interpreter. The comparison function in use is called *strncasecmp(S1,S2,len)* and is available in the *string.h* of the WinAVR distribution. The second function of the interpreter is a method to extract the arguments of the command. This function takes two arguments - a reference to the buffer and the number of arguments to look for. It loops through the buffer and converts every ASCII represented argument to a binary format. The function stops when it finds a newline character, but before it returns, it checks if enough arguments have been found. It then returns a 0 if enough arguments are provided, 1 if not. By allowing commands to be sent in a ASCII format, more buffer memory and more processing power is needed. However, this ensures a command interface that is easier to use and to debug than an interface dealing solely with binary data.

Command Execution

When the interpreter has found a matching command and the number of arguments is found to be correct, a corresponding command flag is set in the main program loop. The main loop holds one flag for each operation it supports and the only task of the loop is to check whether one of these flags has been set. If this is the case, the corresponding function is executed. This behaviour means that during the execution of an operation, the command interface will react to new commands and set command flags, but the arguments of a new command will overwrite the last arguments. To prevent such an unwanted behaviour, a busy flag is introduced. This flag is set to 1 during an operation, and by sending the command `BUSY?` over the interface, the current state is returned.

Command Set

Command	Arguments	Description
IDN?	0	Returns an identifier string for the program
PWM?	1	Returns the current PWM duty cycle
PWM	2	Sets the current PWM duty cycle
CURRENT?	1	Returns the result of a current measurement
DEFLECTION?	1	Returns the result of a deflection measurement
FORCE?	1	Returns the result of a force measurement
CONT_CURRENT_DEFLECTION	7	Starts a continuous measurement of current and deflection
CONT_CURRENT	4	Starts a continuous measurement of current
CAL_CURRENT	1	Calibration function for current control
CONT_DEFLECTION	4	Starts a continuous measurement of deflection
CAL_DEFLECTION	1	Calibration function for deflection measurements
CONT_FORCE	4	Starts a continuous measurement of force
SET_DEFLECTIONS	6	Sets the amount of wanted deflection in all joints
SET_DEFLECTION	4	Sets the amount of wanted deflection in one joint
SET_CURRENT	4	Sets the amount of deflection controlled by the measured current
SET_FORCE	4	Sets the amount of deflection controlled by the measured force
BUSY?	0	Returns the current state of the program
STOP	0	Stops the current operation
RESET	0	Causes a watchdog reset to be performed

Table 4.1: *Command set for the microcontroller program*

Table 4.1 shows all the operations that the interpreter is programmed to receive. The `IDN?` command is just an identification command supported by many instruments. A string containing the name of the micro program and the version number is returned. The control of the three Flexinol wires is done with a PWM-signal. The duty cycle of these signals can be set using the command `PWM ch value`, where `ch` is the channel number (0-3) and `value` is the duty cycle (0-255). The actual PWM signal for the channel `ch` can be queried in the same way with the command `PWM? ch`.

```
1 // Set duty cycle of PWM channel 0 to 50\%
2 PWM 0 128
3
4 // Read the current duty cycle of PWM channel 1
5 PWM? 1
```

The function `DEFLECTION? ch` causes the ADC unit to perform a measurement of the specified Flexinol wire. The result of the measurement is the mean value of five measurements with 1ms delay between each. The mean value is then converted from ADC units to volts and returned with three decimals of precision. The `FORCE? ch` command works in the same way as `DEFLECTION?`, returning the mean value of five measurements of the specified force channel.

```
1 // Read deflection from channel 0
2 DEFLECTION? 0
3
4 // Read force from channel 2
5 FORCE? 2
```

To be able to visualize the data from the sensors, different continuous functions are developed. The command `CONT_CURRENT_DEFLECTION cch0 cch1 cch2 dch0 dch1 dch2 delay` performs continuous measurements of each of the current channels (`cch`) and deflection channels (`dch`) marked with '1'. The number of milliseconds of delay between each measurement is specified in the last argument. For each measurement, the mean value is sent over the RS232 interface to the host computer. The following code demonstrates how to setup the microprogram to perform 5 measurement per second of current and deflection on channel 0.

```
// Read current channel 0 and deflection channel 0 every 200ms (5Hz)
CONT_CURRENT_DEFLECTION 1 0 0 1 0 0 200
```

The results from such a measurement can hopefully be used to make a regulation algorithm that controls the amount of deflection by controlling how much current is flowing through the Flexinol wire (temperature dependent, see section 2.3). Similar to `CONT_CURRENT_DEFLECTION`, the commands `CONT_CURRENT`, `CONT_DEFLECTION` and `CONT_FORCE` perform continuous measurements of current, deflection and force respectively.

```
// Read current channel 0 every 20ms (50Hz)
CONT_CURRENT     1 0 0 20

// Read deflection channel 1 every 50ms (20Hz)
CONT_DEFLECTION 0 1 0 50

// Read force channel 2 as fast as possible
CONT_FORCE       0 0 1 0
```

The commands described so far does not include any form of regulation, they are solely meant to analyse the behavior of Flexinol. There are in general three ways to regulate the contraction rate of Flexinol in this application. The first and most direct way to regulate is to use the deflection measurement as feedback in the regulation loop. To use this method, the microprogram first has to be calibrated with the outer boundaries of each finger joint. This is done with the command `CAL_DEFLECTION`, which performs an automatic calibration. The program presumes that each finger joint is completely stretched when the command is executed. The calibration algorithm first reads the value from the deflection sensor and uses this as the first boundary (0^o). The Flexinol wire corresponding to the same joint is then turned on, making the joint flex. A delay of 1500ms is added to the algorithm in order to be sure that the wire starts to contract. The deflection sensor is then read every 500ms and compared to the last read value. When the difference between the two values is less than $0.01V$, the algorithm saves the last deflection measurement as the second boundary (90^o) and then turns the wire off. The procedure is then repeated for the two other joints.

When the calibration is done, the commands `SET_DEFLECTION` or `SET_DEFLECTIONS` can be used to regulate the Flexinol wires. `SET_DEFLECTION` only regulates one wire and has the channel, the amount of deflection, regulation factor and verbose as parameters. It is only meant for testing purposes as normally all three joints of the finger will be regulated. `SET_DEFLECTION` uses the same regulation algorithm as `SET_DEFLECTIONS`. `SET_DEFLECTIONS` is the main regulation command in the application. It takes one deflection value for each joint, a regulation factor, a verbose flag and how often a verbose should occur as arguments.

```
// Calibrating deflection
CAL_DEFLECTION

// Regulate each joint to 45 degrees, regulation factor 10,
// verbose every 100th regulation cycle
SET_DEFLECTIONS 128 128 128 10 1 100
```

The regulation algorithm first calculates the deflection sensor value corresponding to the set value from the user. This is written as

$$sensorValue = min + \frac{(max - min) \cdot setValue}{255}$$

where min is the deflection sensor value at 0^o and max is the deflection at 90^o. $setValue$ is the deflection value provided through the command arguments. The algorithm then reads the actual deflection D and calculates the absolute error value err

$$err = setValue - D$$

Then the correction value $errP$ is calculated as

$$errP = \frac{err \cdot 255}{max - min} \cdot \frac{reg}{64}$$

The expression takes the relationship between the error err and the total span between min and max and multiplies it with the regulation factor reg. If the regulation factor is chosen to be smaller than 64, a dampening of the regulation is achieved. If it is chosen to be greater than 64, an amplification is achieved. The value of $errP$ is then added to the current PWM output signal. After the new PWM value is calculated, the algorithm checks if the verbose flag is set and checks the verbose frequency value to see if the current values should be sent via RS232. The values of the verbose flag decides what values are to be sent according to table 4.2.

Decimal	Binary	Description
0	`0b0000`	No output
1	`0b0001`	Only the current PWM value
2	`0b0010`	Only the current deflection
4	`0b0100`	Only the current force/current
3	`0b0011`	Deflection and PWM value
5	`0b0101`	Current/force and PWM value
6	`0b0110`	Current/force and deflection
7	`0b0111`	Current/force, deflection and PWM value

Table 4.2: *Values for the verbose flag. By choosing the correct flag, a selection of what data to return from the microcontroller can be done*

The second regulation method uses the amount of current flowing through the wire to control the amount of deflection. As described in section 2.3, the electrical characteristics of Flexinol changes as a function of contraction. This means that by knowing how much current is flowing through the wire at different contraction rates, this can be used to control the wire to the same contraction rate using a current measurement as feedback. This solution has the advantage that it in general does not need a deflection sensor, desirable to minimize systems. However, the electrical characteristics also have a second parameter, namely the amount of stress that is exerted to the Flexinol wire. In the scope of this thesis, regulation is done with only current as feedback, with no stress applied to the wires while regulating. To calibrate the algorithm, the command `CAL_CURRENT` is implemented. `CAL_CURRENT` is dependent on the `CAL_DEFLECTION` command to be executed in advance. It divides the complete deflection range into 16 equal steps and regulates to each of them with deflection as feedback. At each step, a mean value of 150 current measurements is saved. The result of the calibration is an array holding 16 values describing how much current is excepted at different rates of contraction.

When the calibration is done, the command `SET_CURRENT` is used to start the regulation. `SET_CURRENT` takes four arguments - the channel to regulate, the current step to regulate after ($0 - 15 = 0^o - 90^o$), the regulation factor and a verbose flag. The current flowing through the Flexinol wire is determined by the PWM signal from the ATMega32. This means that the current flowing will have the same characteristics as a PWM signal, alternating between no output and maximum output. To control the wires as accurate as possible, the mean value of the current should be measured. This is done by an external RC-filter. Unfortunately, the PWM signal has a rather low frequency, and the RC filter therefore has a time constant $\tau = RC$ that is rather high. This leads to an unwanted delay in the regulation loop. The regulation itself uses the same proportional regulation as `SET_DEFLECTIONS`.

```
1 // Calibrating deflection
2 CAL_DEFLECTION
3
4 // Calibrating current steps for wire 0
```

```
5 CAL_CURRENT 0
6
7 // Regulate wire 0 to current step 7 (45 degrees) with regulation
8 // factor 100 and verbose flag set
9 SET_CURRENT 0 7 100 1
```

During a regulation or calibration, the micro program must be stoppable. This is implemented with a stop flag, similar to the start flags described above. The command `BUSY?` may first be used to check whether the micro program is currently running any functions. If this is the case, the command `STOP` may be sent to abort the current operation. All functions that do regulation and calibration are programmed to set all PWM outputs to 0 when the `STOP` command is received. For unwanted program states during development, the command `RESET` is implemented. `RESET` enables the watchdog timer and sets its time out value to 15ms. It then waits in an infinite loop until the watchdog resets the microcontroller. In this way, a software reset can be done that in normal cases is guaranteed to cause the same effect as power cycling does.

4.5.5 Computer Interface Program

The regulation algorithms and calibration routines described so far are all implemented in the microcontroller. However, the need for sensor data analysis and a command interface makes an interface program necessary. The program supports different profiles, making it backwards compatible with older versions of the microcontroller program. At program startup, the user has to choose which profile folder is to be used. The folder contains one file with all commands and one file with all commands that causes a regulation loop to be executed.

Command Mode

Figure 4.33 shows the first mode of the program, the *command mode*. In this mode, the user chooses one of the commands in the list on the left side of the screen. The available commands, argument names and standard values are read from the command file described above. When the user selects a command, the correct number of text boxes for arguments are shown with corresponding argument names and standard values. When the green send button is pressed, a text string is concatenated from the command and its argument values. The string is then sent to the microcontroller via the serial port. At the same time the string is written to the main text field and a timer is started in the background. The timer is set to check the RS232 receive buffer for new data. If the buffer contains a complete line terminated with a newline character, the line is printed to the main text field on the screen. The red stop button is programmed to quickly send the `STOP` command to the microcontroller in case of unwanted behaviour or emergencies.

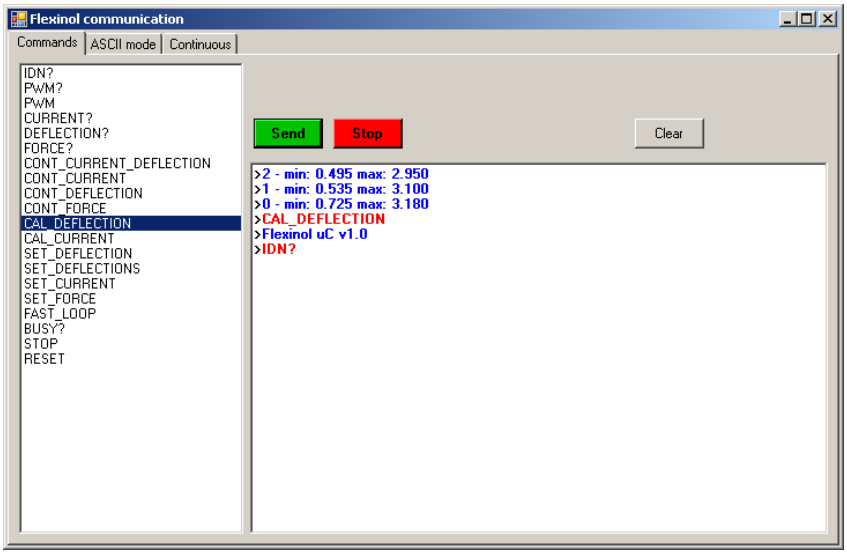


Figure 4.33: *Command mode of the interface program. This mode is used to execute single commands*

ASCII mode

The second mode of the interface program can be seen in figure 4.34 and is called the *ASCII mode*. ASCII mode is the simplest form of communication with the microcontroller and is in reality only a terminal program. When the send button is pressed, the text in the input field is sent to the microcontroller and logged in the large text field. A timer is then started analogous to the command mode. The ASCII mode is meant for command prototyping and easy debugging.

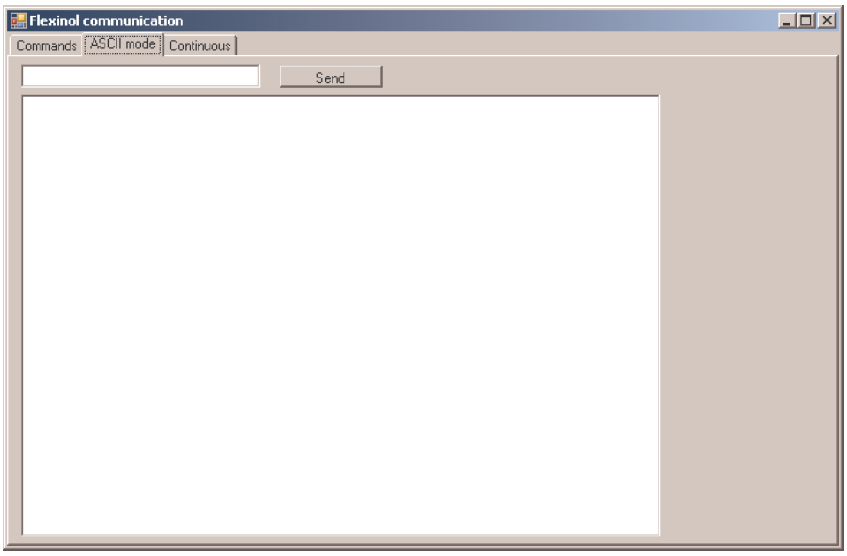


Figure 4.34: *ASCII mode of the interface program. This mode is used to prototype and debug with text input*

Continuous Mode

The third program mode is the *continuous mode*, depicted in figure 4.35. The continuous mode contains a list on the left side of the screen, similar to the command mode. The commands available are read from the file with continuous commands as described above. This mode is used to visualise the different sensor signals over time. As the figure illustrates, instead of a text field, a plotting surface is available. The Y-axis of the graph is auto scaling. The length of the X-axis is adjustable during use to allow different time resolutions. The continuous mode also contains a checkbox that controls whether or not the received data is to be saved. If yes, all data is saved to an auto generated file in the program folder.

4.5.6 Interface for Microsoft Robotics Studio

As described in section 3.3, the Microsoft Robotics Studio is a framework for robotic applications aiming to provide a fast and powerful base for development. A prototype service is therefore developed to connect the finger with Robotics Studio. Figure 4.36 shows a screenshot of an interface that is integrated in the web based control panel of Robotics Studio. The service communicates with the microcontroller via the same module that was used in the interface program in section 4.5.5. The static nature of html pages makes interactive feedback of sensor data difficult, and therefore only a selection of the microcontroller functions are realized. As depicted in the figure, the first table shows if the service is connected to the microcontroller and provides buttons for connecting, resetting or stopping the current function running on the microcontroller. The second table shows the last set of sensor readings. These values are updated every time the page is refreshed. Finally, the last table allows the user to set the duty cycle of each PWM-output or to start the regulation of a given amount of flexion in the joints.

4.6 Summary of Own Methods

In this chapter, a number of self developed methods have been presented. Firstly, different mechanical tests for the Flexinol muscle fibers were proposed. A test frame was built to perform the tests that

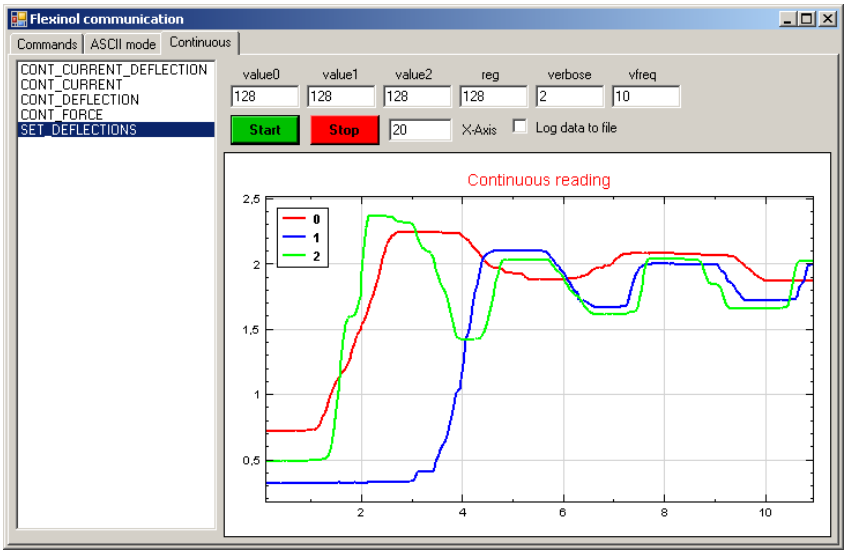


Figure 4.35: *Continuous mode of the interface program. This mode is used to plot data from continuous microcontroller commands*

were controlled and monitored by a self programmed measurement software, running on a laboratory computer. A web application was developed to be able to monitor the tests from a remote connection.

A 3D model of a humanoid finger was then developed using the CAD software SolidWorks. The model was produced in ABS-plastic with a 3D printer and then mounted to an aluminium tube. Tendons were routed and connected to Flexinol wires for actuation. An electronic circuit was built, using a microcontroller as control unit for the Flexinol wires and for sensory input. A microprogram was developed to run on the microcontroller, containing a command set with a number of different control options. An interface program was developed to interact with the microcontroller from the laboratory computer via RS232. An optional interface for Microsoft Robotics Studio was also developed.

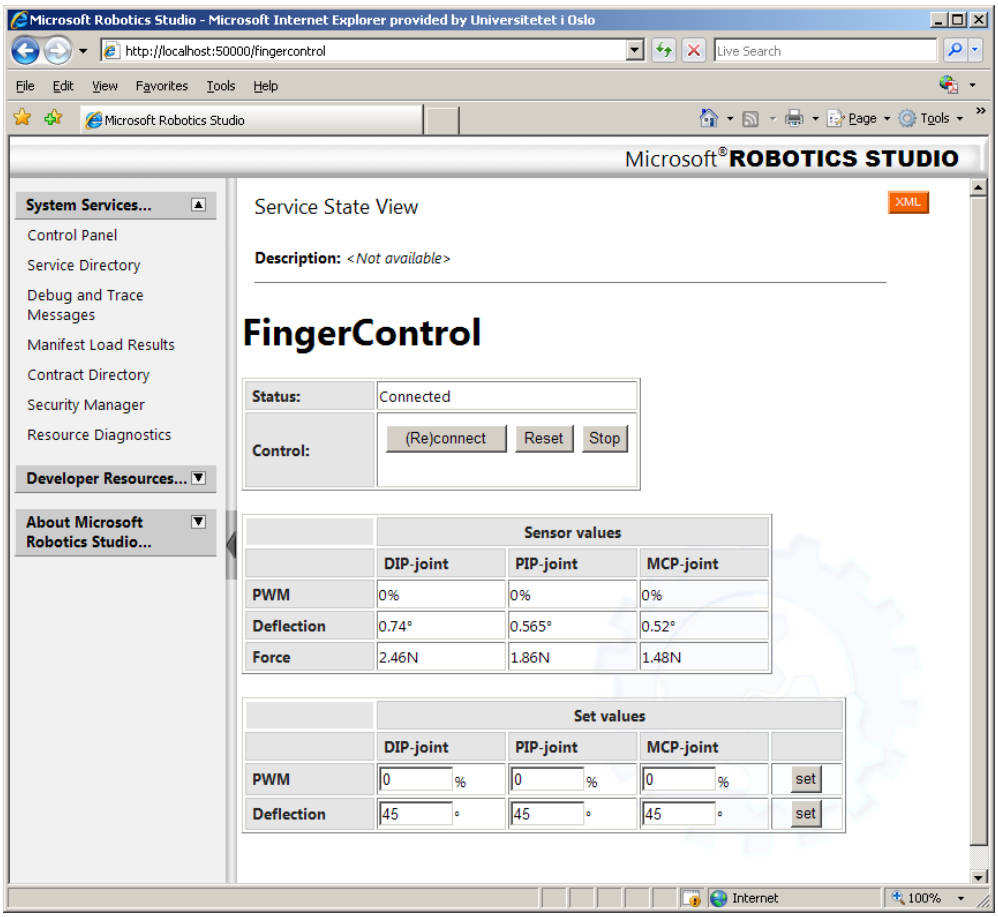


Figure 4.36: *Web interface for the humanoid finger, implemented in Robotic Studio. The user can control the finger through the control panel*

Chapter 5

Experiments

This chapter describes the results from the methods presented in the previous chapter. First, calibration results for force and displacement are given. These results are necessary to perform measurements while testing Flexinol. Secondly, the results from the testing of Flexinol is given and finally, the developed humanoid finger is discussed.

5.1 Calibration Results

Calibration of the test frame was done according to section 4.2.2. Although some non-linearities were discovered, these were considered small enough to be negligible both for displacement and force. Both calibration result tables contain columns named NL (non-linearity). NL is the deviation from a straight line through the calibration points, calculated by the method of least squares. The calculated line is written as $y = Ax + B$, where A is the gain and B the offset found in the calibration tables.

5.1.1 Displacement Calibration

Table 5.1 shows the calibration results for the displacement transducers. The left part of the table contains values for the linear displacement transducers while the right part contains values for the radial transducers. The negative values are caused by the displacement monitor that generates the needed excitation voltage. The first radial transducer was calibrated with 0-90mm as the output value would drop before 100mm was reached.

The results from the calibration are visualized in figure 5.1. The calibration curves show that the output is near to linear. This assumption is also confirmed by the NL-columns of the table. All over, the NL varies around $0.03V$ for the linear displacement transducers, which equals about $0.34mm$ when scaled. This is less than 1% of the $50mm$ measuring range. The worst case is an error of $0.076V$, which equals $0.8mm$ and 1.6%. For the radial transducers, the NL varies around $0.04V$ which equals about $0.43mm$, less than 0.5% of the $100mm$ measuring range. The worst case is an error of $0.073V$, which equals $0.8mm$ and 0.8%.

All mentioned error values are considered negligible.

5.1.2 Force Calibration

Table 5.2 shows the results from the force calibration. By looking at the calibration curves in figure 5.2 and the NL-values in the table it becomes clear that the load cells are also linear. The highest error value is $0.038V$, which equals about $18g$ and $0.18N$. This error is considered negligible.

5.2 Testing of Flexinol

The aluminium test frame described in the previous chapter was used to fasten and test Flexinol wires in different setups. The frame was designed to hold 5 parallel wires of 1m in addition to displacement and force sensors. It proved to be large and stiff enough for all tests.

	Small load		Heavy Load			Flexinol Ant.		Spring Ant.	
cm	**ch 0 [V]**	**NL**	**ch 1 [V]**	**NL**	**cm**	**ch 2 [V]**	**NL**	**ch 3 [V]**	**NL**
0,0	-0,060	0,037	-0,103	0,076	0	0,277	-0,044	-0,105	0,073
0,5	0,340	-0,025	0,239	-0,048	1	1,322	-0,007	0,783	0,028
1,0	0,803	-0,023	0,717	-0,036	2	2,324	-0,013	1,641	-0,048
1,5	1,300	0,012	1,222	0,003	3	3,370	0,025	2,585	-0,038
2,0	1,750	0,000	1,688	0,003	4	4,415	0,062	3,546	-0,010
2,5	2,217	0,006	2,154	0,003	5	5,376	0,015	4,488	-0,002
3,0	2,670	-0,003	2,598	-0,019	6	6,383	0,014	5,361	-0,062
3,5	3,117	-0,017	3,075	-0,009	7	7,373	-0,004	6,332	-0,025
4,0	3,590	-0,006	3,555	0,005	8	8,380	-0,005	7,306	0,016
4,5	4,057	0,000	4,008	-0,008	9	9,350	-0,043	8,253	0,029
5,0	4,538	0,019	4,511	0,029	10			9,196	0,039
Offset	**0,105**		**0,194**			**-0,318**		**0,192**	
Gain	**1,083**		**1,072**			**0,992**		**1,071**	

Table 5.1: *Calibration table for displacement. The results show that the displacement transducers can be considered linear*

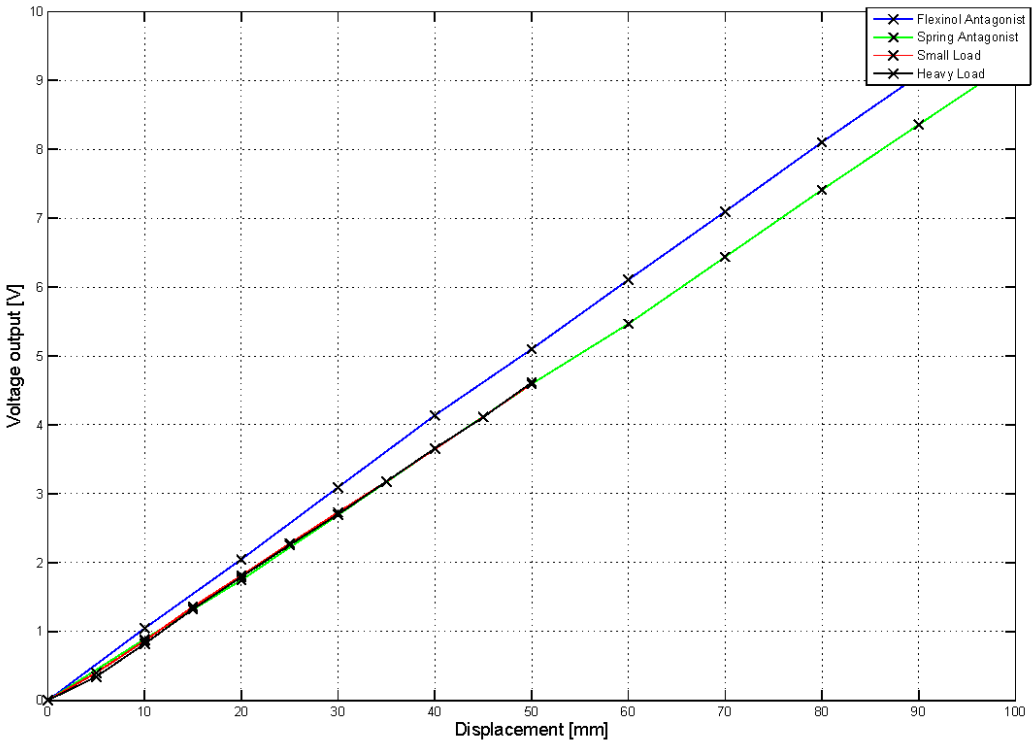


Figure 5.1: *Calibration curve for displacement. Almost no non-linearities are visible. Small deviations are seen around 0mm displacement*

		Fixation		Flexinol Ant.		Spring Ant.	
g	**N**	**ch 0 [V]**	**NL**	**ch 1 [V]**	**NL**	**ch 2 [V]**	**NL**
0	0,000	0,130	0,027	0,301	0,038	0,183	0,014
100	0,981	0,272	0,002	0,478	-0,001	0,455	0,014
200	1,962	0,436	-0,001	0,682	-0,012	0,733	0,019
300	2,943	0,593	-0,011	0,900	-0,009	0,990	0,003
400	3,924	0,758	-0,013	1,116	-0,009	1,248	-0,011
500	4,905	0,927	-0,011	1,334	-0,006	1,499	-0,033
600	5,886	1,101	-0,004	1,549	-0,006	1,786	-0,018
700	6,867	1,265	-0,007	1,768	-0,002	2,061	-0,016
800	7,848	1,437	-0,002	1,985	-0,001	2,343	-0,007
900	8,829	1,606	0,000	2,191	-0,010	2,605	-0,018
1000	9,810	1,784	0,011	2,412	-0,004	2,926	0,030
1100	10,791	1,944	0,004	2,644	0,012	3,187	0,019
1200	11,772	2,110	0,003	2,840	-0,007	3,439	-0,002
1300	12,753	2,277	0,003	3,058	-0,004	3,703	-0,011
1400	13,734	2,442	0,001	3,282	0,005	3,994	0,007
1500	14,715	2,609	0,001	3,509	0,016	4,268	0,009
Offset		**-0,604**		**-1,199**		**-0,465**	
Gain [V⇒N]		**5,873**		**4,556**		**3,545**	

Table 5.2: *Calibration table for force. The results show that the load cells can be considered linear*

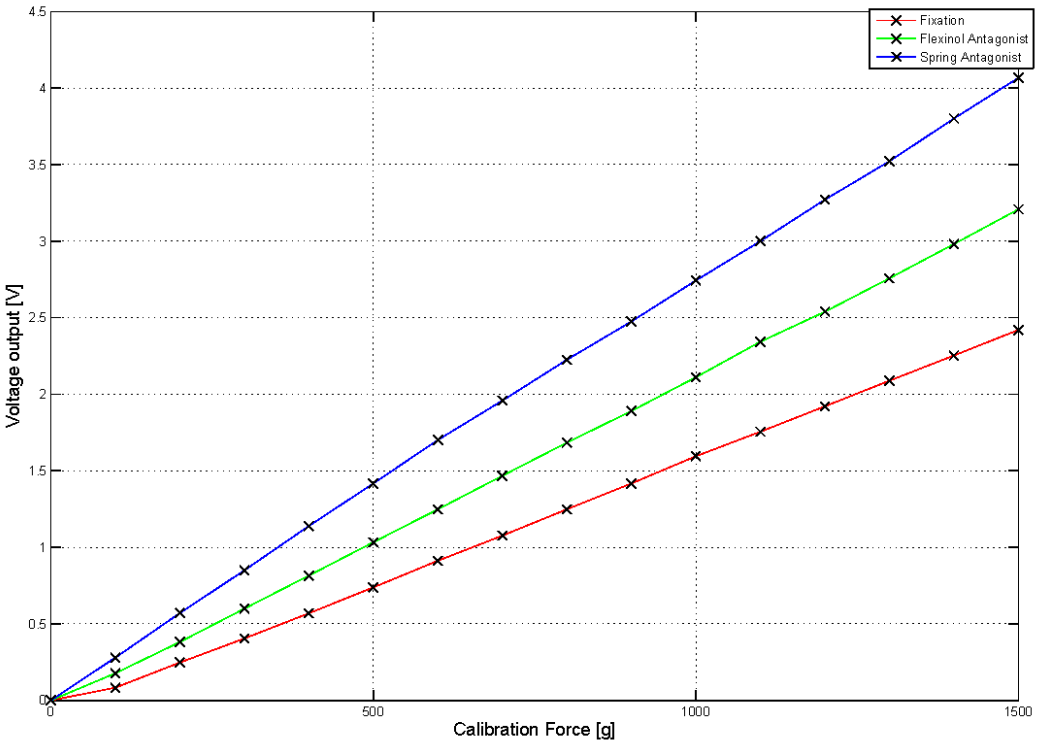


Figure 5.2: *Calibration curve for force. Almost no non-linearities are visible. Small deviations are seen around 0N*

5.2.1 Test Software

The main task of the test software is to perform measurements and to control output signals, as described in section 4.3.1. During the first weeks of testing, the computer used for the measurements tended to run out of memory. A memory leak was suspected, and a first attempt to find the error was done with the Microsoft CLR Profiler, a help tool to locate memory leaks. The profiler showed clearly that the memory was growing constantly, which confirmed the assumption of a memory leak. As memory was growing with little steps, a new assumption was made that the leak had to be in either the function controlling the output or in the function performing the input measurements. The error was finally found to be in the measurement function. Each time a measurement was started, the Keithley KUSB unit was configured with its setup. This function, provided by Keithley, has a minor memory leak that causes about 14KB of memory to be used every time the function is called. Over time this demands a great amount of virtual memory, causing the computer to slow down. The problem was solved by reorganizing the code, making the configuration call only once.

Although the memory leak was fixed, the program seemed to slow down after a couple of days of running. The amount of used memory was stable and the program used about 50% of the processor power. The reduced speed of the program sometimes caused the output to be a little delayed or caused the frequency of the input measurements to slow down. This again caused some of the measurement data to have a lower time resolution than wanted. However, the program speed was varying and not all cycles were affected by the low measurement frequency. Therefore the problem was left unsolved, considered to be a minor problem.

5.2.2 Fixation Test

The first setup described in the previous chapter is the fixation test (figure 4.2a). The wire was fastened as described in section 4.1.1. In the top of the frame, a threaded stem was used to adjust the wire so that it would be tight. The fixation test soon proved itself to be the most work demanding. Each wire mounted, tended to break after approximately two days of actuation. The wire break was always inside the upper or lower fastening mechanism, suggesting that the method for fastening the wire is causing some damage to it.

The first six wires that were tested were unfortunately tested with an erroneous amplifier setup. Two errors were found that certainly have caused wrong force measurements. The first error was the zero adjustment of the strain gauge. The bridge had not been properly adjusted so that with no force applied, the output of the amplifier would be its lower rail. When applying a force, the output would remain at the lower rail until the strain gauge had a large enough output signal to drive the amplifier above the lower rail. This caused an offset error, but also infected the calibration values that were calculated.

The second error that was found also concerned the amplifier. The maximum force measured from the wire with the erroneous setup was 30N. This is three times more than reported in the Flexinol datasheet [37] and therefore it did not appear as obvious that this value actually was the upper rail of the amplifier. However, as the first data analysis was done, it became apparent that the maximum value was the same for every contraction cycle. The gain of the op-amp had been set too high and thus was the output range of the amplifier too small.

To correct the two errors, a zero adjustment was first done as described in section 4.2.1. Then a smaller gain was chosen before a new calibration was done. The new setup showed that the real pull force of the wires was even larger than the first erroneous results. The maximum pull force that was measured exceeded 40N, more than four times the reported maximum pull force. The pull force reported in the datasheet of course refers to the recommended pull force.

Although the first six measurements had to be considered invalid in terms of determining pull force degeneration over time, they were not completely in vain. An important property was uncovered, namely the lifetime of a wire when loaded to its maximum. The lifetime can be seen in table 5.3 and is given as number of contraction cycles before the wire breaks. The column of maximal force contains erroneous results for the first five wires caused by the wrong amplifier setup described above. The delay column shows the time it takes after the wire is turned on until it reaches a force of 5N. The break point tells where the wire snapped.

Wire	Cycles	Max force [N]	Delay [s]	Break Point
1	1969	31.05	1.2	bottom
2	1501	31.06	0.9	top
3	1504	31.04	1.4	top
4	1193	31.05	1.2	bottom
5	1971	31.06	1.1	top
6	512	40.3	0.85	top
7	7876	41.5	0.6	bottom

Table 5.3: *Results from testing a fixated Flexinol wire. The wires always broke either in the upper or lower fastening mechanism*

Measurement Cycle

Figure 5.3 shows the measured force of a wire together with its control signal. A small time delay can be observed between the rising edge of the control signal and the actual contraction start. As described in section 2.3.2, this is caused by the warm-up time of the Flexinol wire. Analogous, a delay is seen as the output is turned off, caused by the cooling time of the wire. As the wire reaches its maximum force of around 40N, the force signal seems to become rather unstable. When the wire is in this state, the temperature is much higher than normally needed to contract the wire. But because of the applied stress (fixed wire), the wire cannot contract. The temperature at this point is probably more than 100^oC, making the difference between the wire temperature and its surrounding temperature very large. An explanation for the unstable behaviour could therefore be that the constantly cooling and heating of the wire at this point causes some kind of oscillation. It should be noted that four times the recommended stress is applied to the Flexinol wire at this point.

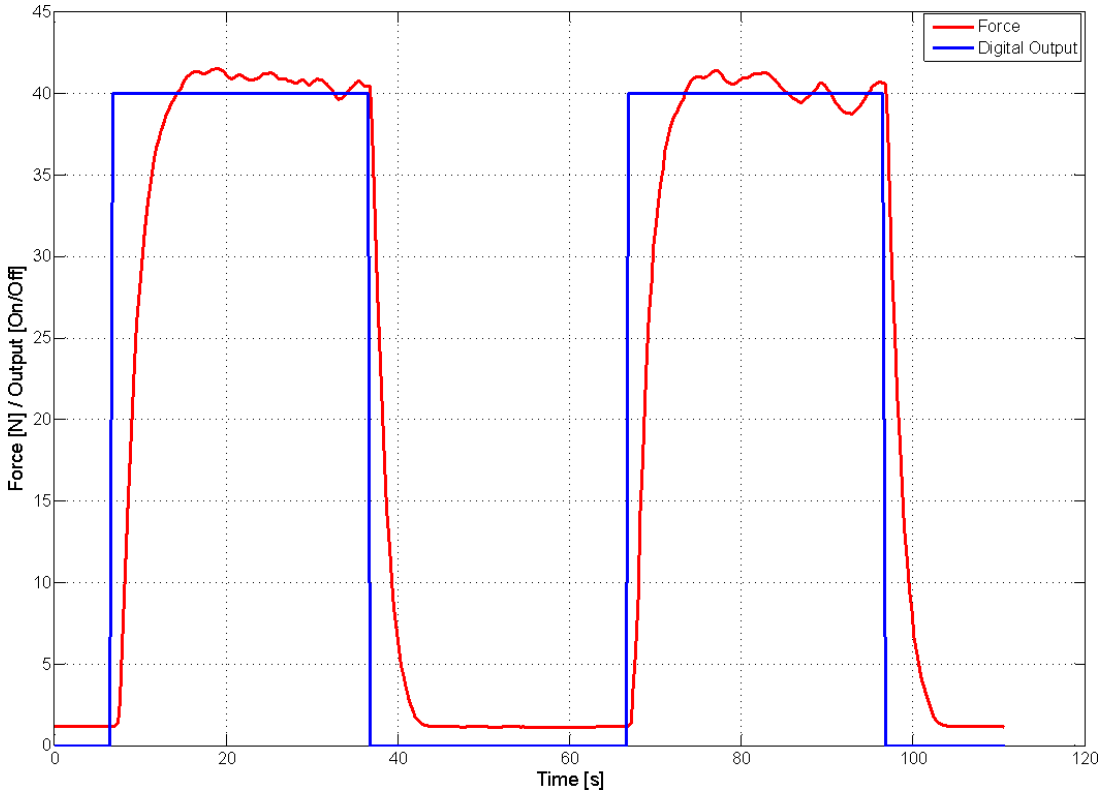


Figure 5.3: *Two contraction cycles of a fixated Flexinol wire. The maximum force of ca 40N shows that Flexinol is able to exert rather large forces*

Degeneration

In figure 5.5, the degeneration of Flexinol over time is illustrated. The figure shows the actual pull force of the wire through about 8000 cycles. Each graph value is the difference in force from the start of a contraction to the top of the contraction curve. As mentioned above, the signal is rather unstable when the wire is at its maximum force, this is also reflected by the rather large deviations in the curve. From cycle 1 to about 3100, a reduction in force is noticed. This was first assumed to be caused by a weakening of the wire but turned out to be caused by a stretching of the wire. At cycle 3100, the wire was tightened and as the graph shows, this causes an increase in force. The figure also shows a constant decrease in force after the tightening which also mainly is caused by a stretching of the wire. Surprisingly, the pull force of the wire after tightening is almost as large as that of a new wire. The stretching of the wire continues for each cycle and it would be interesting to see if the stretching converges. During the performed tests, this was never observed as the wire would break before a trend could be seen. At around cycle 5800, an external disturbance caused a series of erroneous measurements that should not be evaluated.

Both between cycle 1 and 500 and between cycle 6000 and 7000 some additional noise can be seen on the curve. This is most likely caused by temperature variations in the surrounding environment. These variations can be caused by an open window, the central heating of the building or sunshine. However, regardless of the wire being stretched and an unstable test environment, the wire proves that its pull force is very large until the point when the wire snaps. All the wires that were tested snapped at either of its ends. This indicates that the fastening mechanism of the wire is somehow causing damage to the wire, either at time of fastening, at time of contraction, or both. An attempt was made to improve the fastening mechanism. A thin copper plate was cut and placed inside the fastening mechanism between the screws and the Flexinol wire. This did not seem to improve the lifetime of the wire, and the problem was left unsolved as this only was occurring at this particular test setup with a load greatly exceeding the recommended limitations of Flexinol. Overcoming this problem could maybe make it possible to test the fixated wire over a longer period of time. This, however depends on whether or not the rest of the wire is close to snapping when the fastenings have snapped. Maybe a fastening of the wire with a knot similar to the clove hitch, seen in figure 5.4, can be used to prevent the fastening mechanism from being the weakest link. This, however would probably make the fastening mechanism both larger in size and more complex.

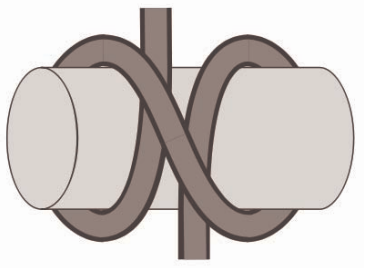


Figure 5.4: *A Clove Hitch knot could maybe be used to create a more gentle fastening mechanism for the Flexinol wires.*

In figure 5.6, a detailed view of two contraction curves can be seen. The red curve shows a contraction cycle after 500 contractions and the blue curve shows contraction cycle after 4500 contractions. One difference that can be observed is a slightly lower pull force after 4500 cycles than after 500. This may be caused by a stretched wire, as described above. Some delay in the cooling of the wire can also be seen, but this is probably due to the time resolution of the measurement software because the slope of the two cooling curves are near to identical.

As already mentioned, a delay occurs between the rising edge of the output signal and the start of the contraction. Figure 5.7 shows the time between the rising edge of the output signal and the point where the force signal is larger than 5N. This can be seen as the reaction time of the wire. The figure shows many high reaction times that are a result of the time resolution of the measurements. However, the interesting part of the data is that the reaction time from the correct measurements seem to be stable around $0.5s - 0.8s$ over the whole period. This observation together with the observations around figure 5.6 can therefore be used to conclude that only a small, negligible change is done to the transformation curve of the wire during the test.

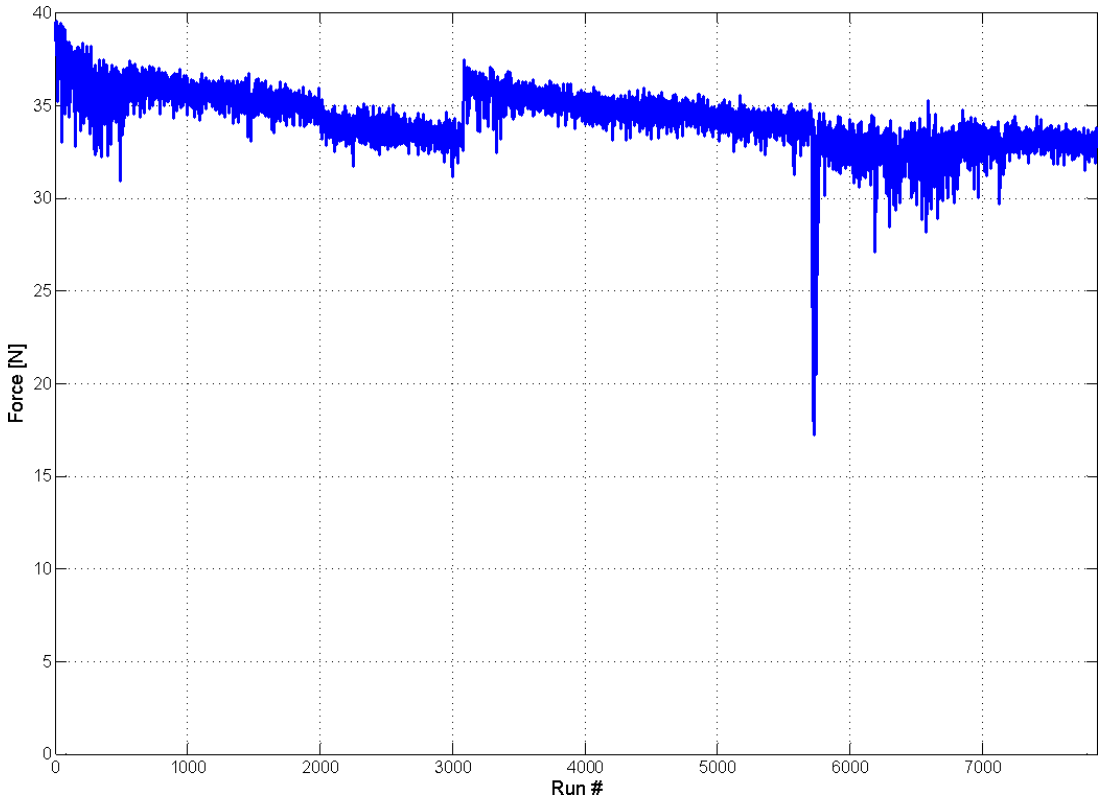


Figure 5.5: *Degeneration of a fixated Flexinol wire over time. The decrease in force is caused by the wire being stretched*

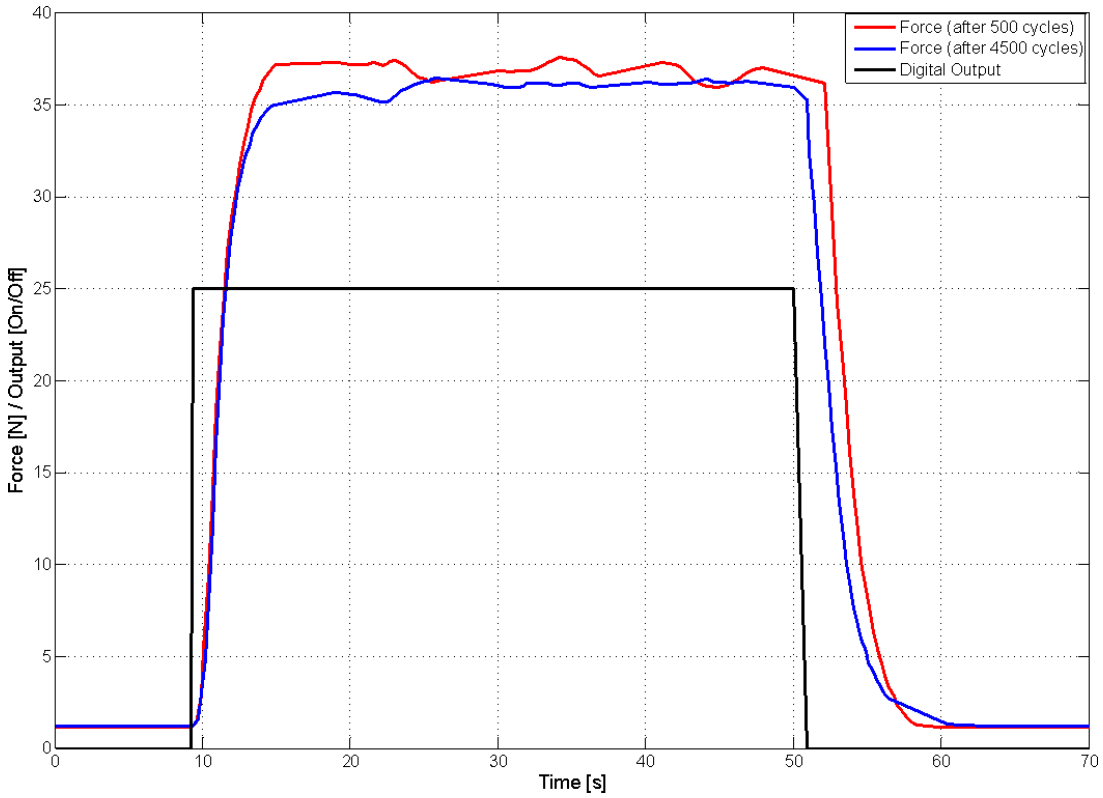


Figure 5.6: *Comparison between contraction curve after 500 and after 4500 cycles for a fixated Flexinol wire. Only very small differences are observed*

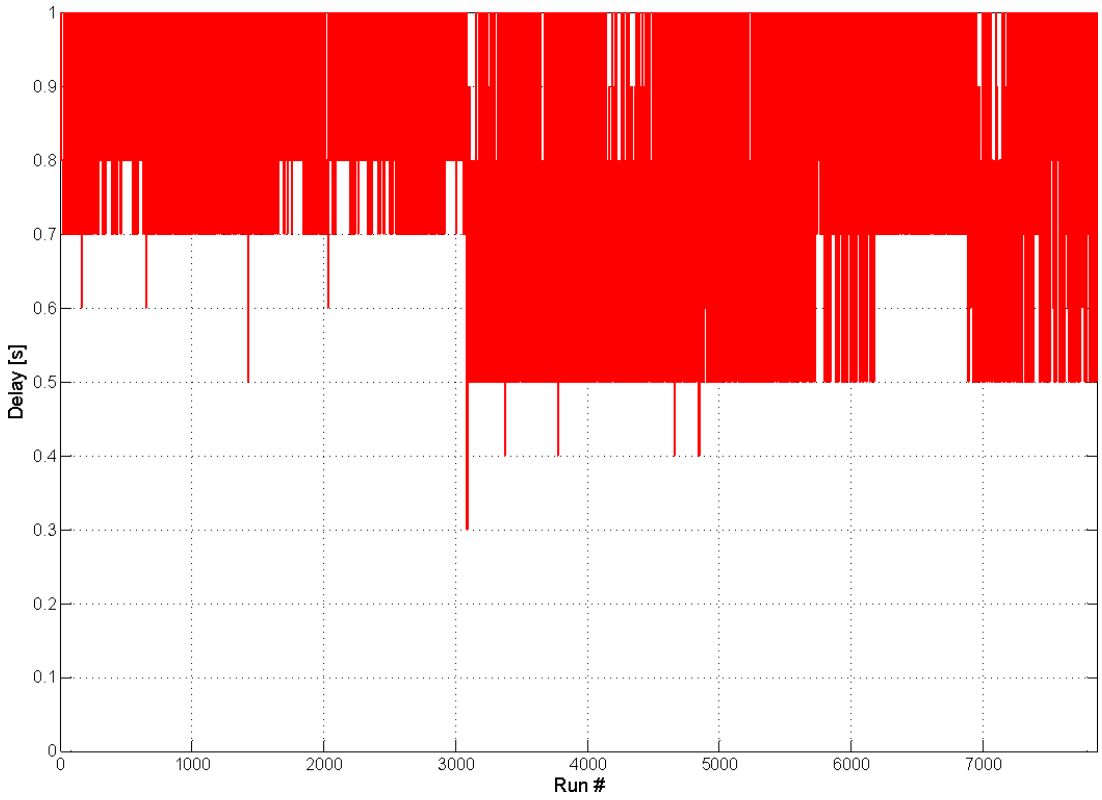


Figure 5.7: *reaction time of a fixated Flexinol wire. The curve shows the time from the wire is turned on until the measured force exceeds 5N. No changes can be observed over time*

5.2.3 Degeneration Test

The two degeneration tests are described in section 4.1.2.

Small Load

In the first test, only a small load was supposed to be used to stretch the wire after a contraction. The control signal for the test is depicted in figure 5.8. First, a weight of $110g$ was molded in lead and fastened to the wire. The result of this test clearly shows that the applied weight is to small to stretch the wire (figure 5.9). During the first hours of operation, the wire stretches less and less after each contraction, and after one day the Flexinol wire only has about $1.5mm$ difference between contracted and stretched state. Table 5.4 shows a survey of the performed tests. The most interesting result is from wire 4, where 48861 cycles were completed. The initial deflection was $41mm$ and the ultimate deflection was $22.5mm$.

Wire	Cycles	Max deflection [cm]	Min deflection [cm]
1	5866	2.9	0
2	7	-	-
3	1662	2.9	1.45
4	48861	4.1	2.25

Table 5.4: *Results from testing a Flexinol wire with a small load. The fourth run is the first run with a large enough weight. The very high number of contraction cycles should be noted*

A number of washers were then fastened on top of the lead weight, making its total weight $200g$. This was not enough, so another $75g$ was added. With $275g$, the wire could be stretched about $20mm$ but was unstable from cycle to cycle. The wire was therefore replaced in case it had been damage through being contracted for several days (figure 5.9). This seemed to be the case, because after changing wires, an improvement in stability could be seen. These results can be seen in figure 5.10, where the long duration of the test also should be noted. The gaps in the data is caused by computer breakdown and

power loss during the measurement period. Applying such a small load to the wire seems to cause very little damage to it. The difference between the contracted state and the stretched state of the wire is illustrated in figure 5.11 and it can be seen that in contraction cycle 1, the deflection is about 37mm. The degeneration of the wire is clearly visible between cycle 1 and cycle 15000 where it seems to converge towards a deflection of about $22.5mm$. This amount of deflection seems to be very stable as it is repeated for the next 30000 cycles.

It is very valuable to know that the wires converge towards a stable deflection rate after a while. However, with a stable contraction rate of about $22.5mm$, only 2.25% deflection is achieved. This has to be considered very low compared to the initial rate of 3.7% and the specified $4.5 - 5.0\%$. Another aspect of this test is the size of the applied weight. The maximum recommended pull force reported by the manufacturer of Flexinol is $930g$. When $275g$ are needed to stretch the wire, this equals about $1/3$ of the maximum, which has to be considered fairly high, limiting the working range.

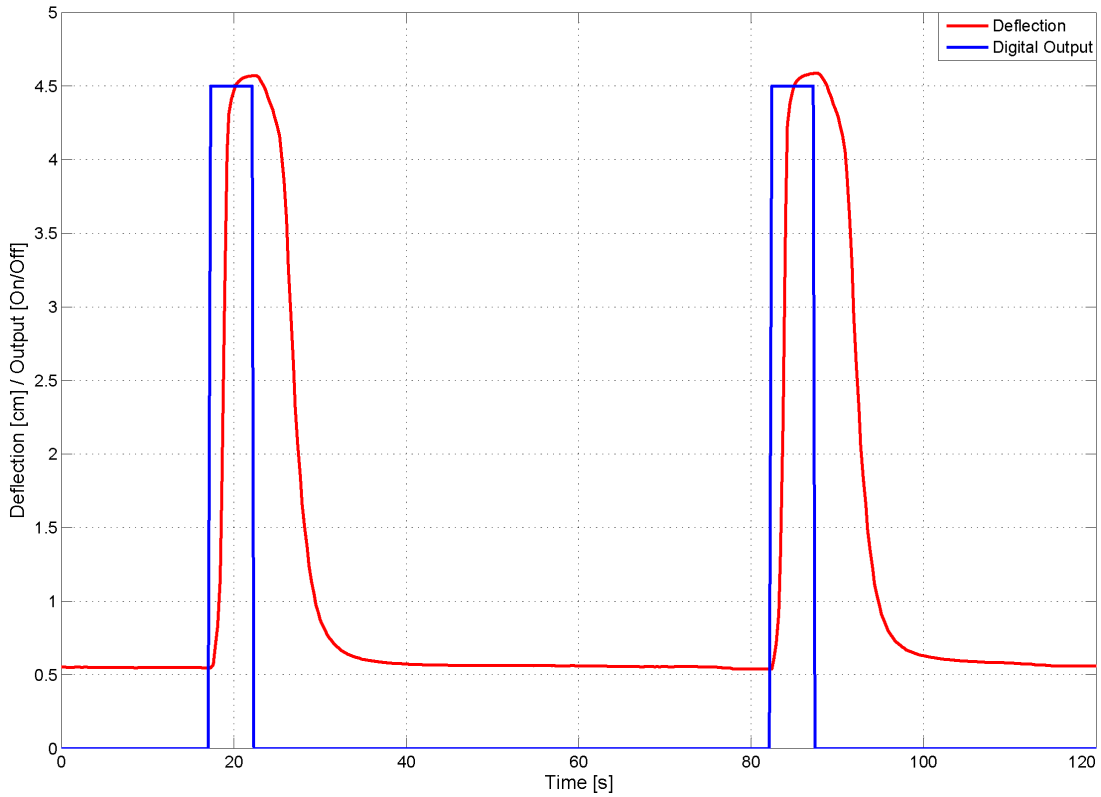


Figure 5.8: *Two contraction cycles for a small load. About 4% contraction is observed for the first two cycles*

Heavy Load

The second of the two degeneration tests is similar to the first one, except for the weight being used to stretch the wire. Again, a weight was molded from lead and trimmed to be ca 930g, the same as the recommended maximum pull force of the Flexinol wire. The weight was fastened to the end of the wire and placed on top of a deflection transducer with a return spring. The return spring causes a small error to the applied weight, but was considered to be negligible. Table 5.5 shows the results from all the tested wires. The life time of each wire varies very little. However, the average life time of ca 12000 cycles is about 1/4 of the life time of the above mentioned wire with a small load.

The control signal for the test can be seen in figure 5.12 together with the deflection. By looking at the time axis, it can be seen that the frequency of this test is the double of the frequency for the light weight. This is due to the fact that the heavy weight stretches the wire much faster than the light weight does.

Results from one test can be seen in figure 5.13 and 5.14. As the figures show, a very stable deflection of about $4.25mm$ is observed. However, as figure 5.13 shows, the wire is continuously stretched under

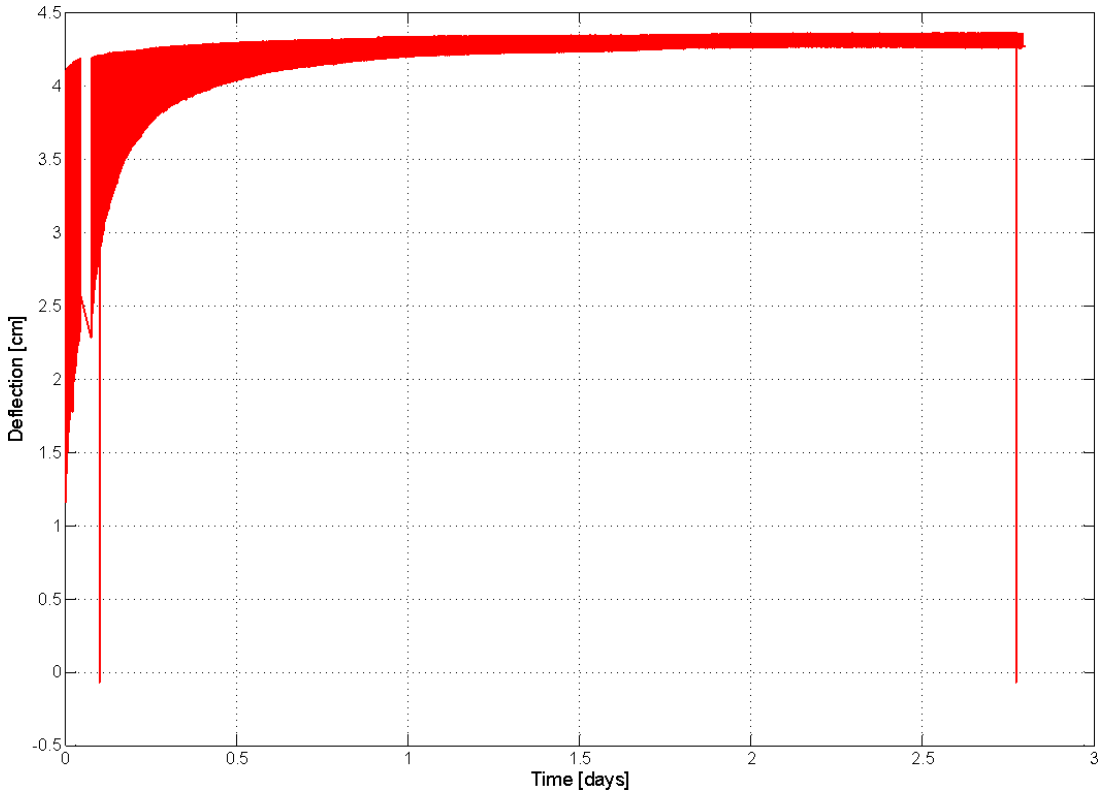


Figure 5.9: *Deflection over time with 110g applied. 110g proved to be insufficient to stretch the Flexinol wire after a contraction*

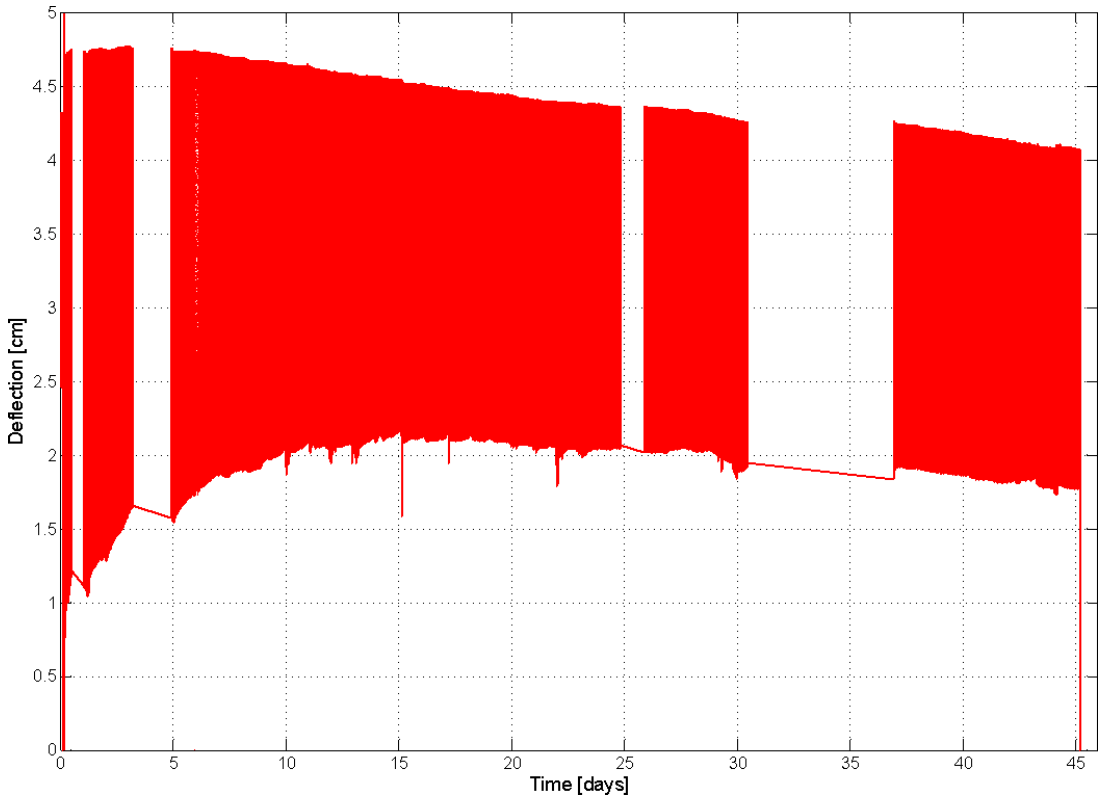


Figure 5.10: *Deflection over time with 275g applied. 275g was enough to stretch the Flexinol wire after a contraction and also proved not to overload it*

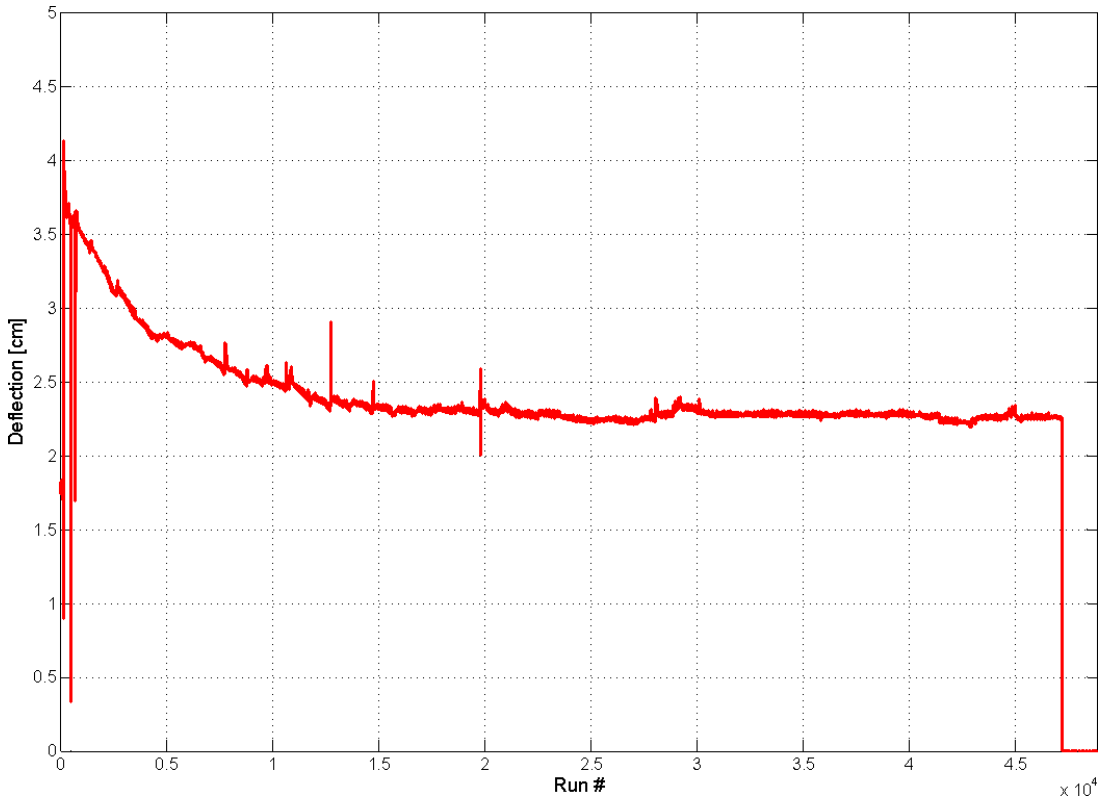


Figure 5.11: *Difference between contracted state and stretched state from figure 5.10. The contraction rate converges towards 2.25% after 15000 contraction cycles*

Wire	Cycles	Max deflection [cm]	Min deflection [cm]
1	11910	4.5	4.1
2	11399	4.4	4.3
3	12994	4.4	4.2
4	11531	4.25	4.0
5	12158	4.3	4.2

Table 5.5: *Results from testing a Flexinol wire with a heavy load. Very similar results for all five runs can be observed. The deflection rates of about 4.5% are equal to the information found in the Flexinol datasheet [37]*

the load of the $950g$. After about 3 days (5500 cycles) the wire was therefore tightened to keep it inside the $50mm$ measuring range of the deflection transducer. Figure 5.14 shows that the deflection transducer already was at its lower limit, causing a slight increase in deflection when tightening the wire. After about 11000 cycles, the wire snapped without any warning signs in terms of radical changes in deflection or speed (figure 5.15).

5.2.4 Flexinol Antagonist

The use of an extra antagonist wire to stretch an agonist wire is described in section 4.1.3. The first attempt with this method can be seen in figure 5.16. As the graph shows, the agonist wire produces a small force when actuating. This comes from stretching the cold antagonist wire. However, when the agonist wire is turned off and the antagonist wire is turned on, a large force is produced. This is because the agonist has not yet been cooled down while the antagonist starts to stretch it. As a result, both wires are stretched. This can also be seen be looking at the figure. The figure shows the 8 first cycles of the test, and during this short time the deflection is reduced from $15mm$ to $6mm$. The maximum deflection could have been about $25mm$ as each wire has a length of $50cm$. During the same period, the force is almost halved, also reflecting that the wires have been deformed. A second attempt was therefore done with a more conservative control signal. Two cycles from this attempt can be seen in figure 5.17. As the

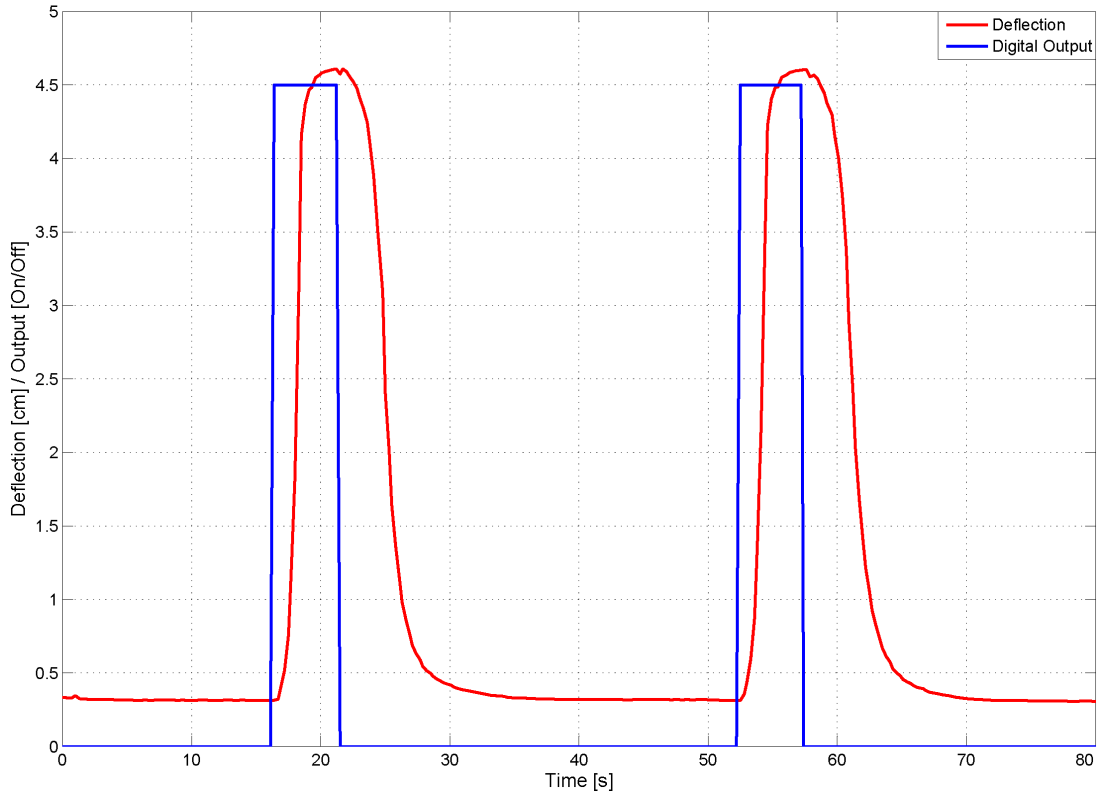


Figure 5.12: *Two contraction cycles for a heavy weight. A contraction of about 4.25% is observed for the first two cycles*

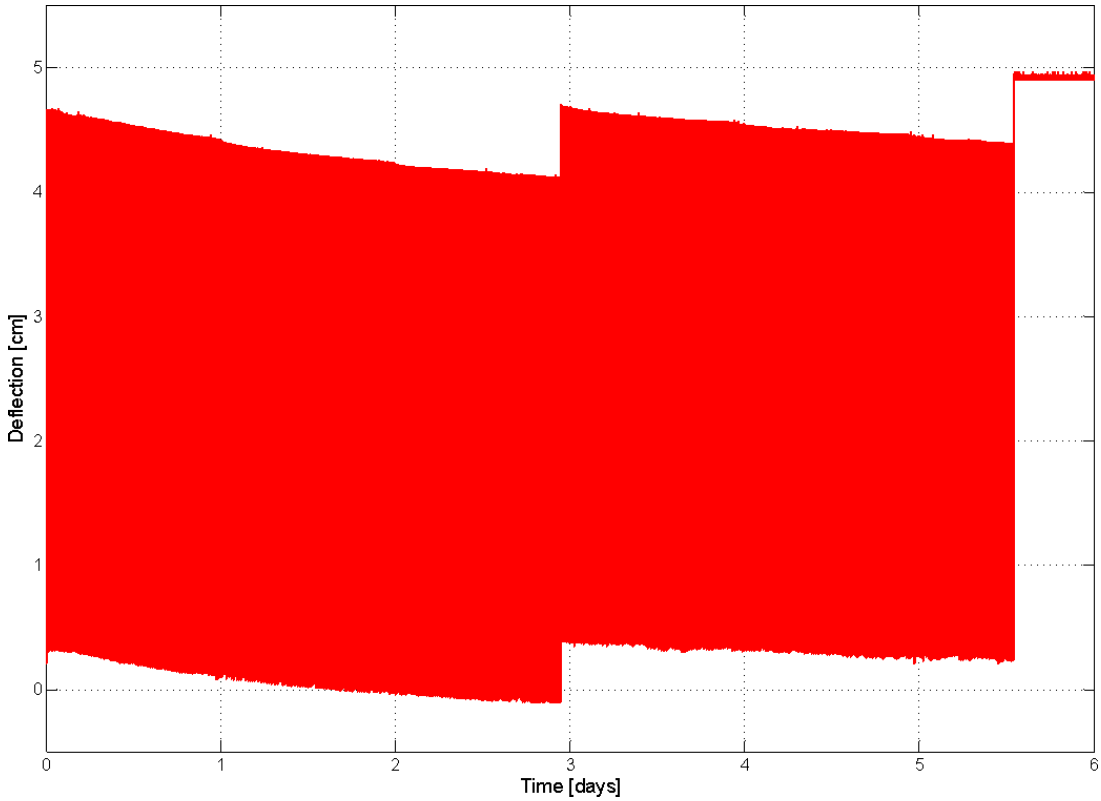


Figure 5.13: *Deflection over time with 950g applied. The Flexinol wire is stretched under this load and had to be tightened after about 3 days*

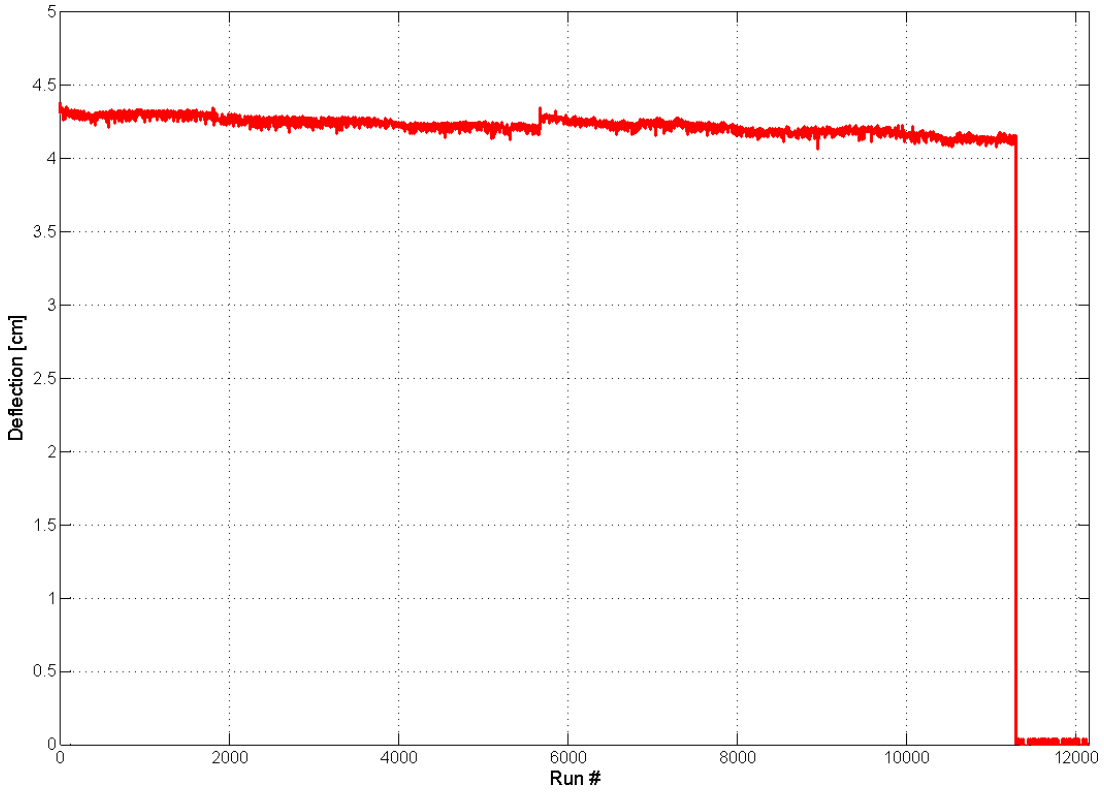


Figure 5.14: *Difference between contracted state and stretched state from figure 5.13. Although the wire is stretched over time, the deflection rate is very constant*

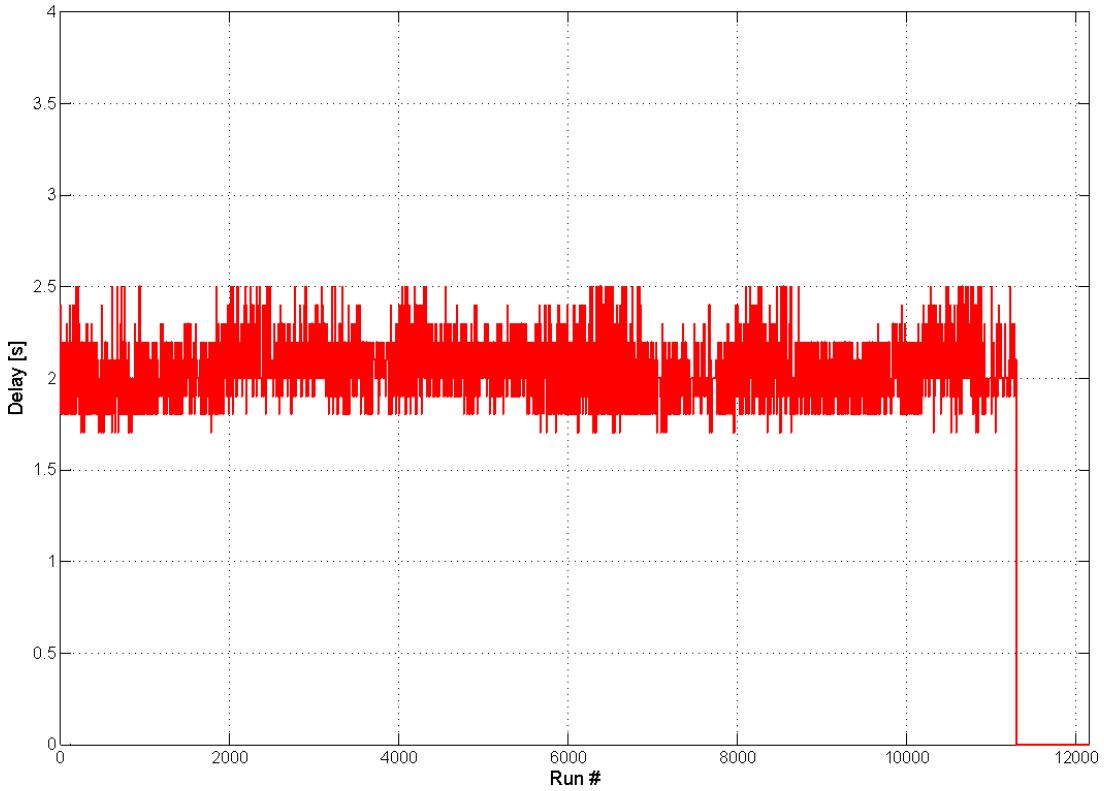


Figure 5.15: *Delay between rising output and 3cm deflection. This curve also shows that no change in reaction time can be observed*

graph shows, much smaller forces are exerted compared to the first attempt, indicating a more gentle use of the wires. A deflection of $22mm$ is equivalent to 4.4%, which is a rather good utilization. However, as figure 5.18 shows, the deflection decreases almost linearly towards $0mm$ during the first 500 contraction cycles. Although performing better than the first attempt, this indicates a rather strong deformation of the wires as a result of the antagonistic behaviour. After about 600 cycles the wires were tightened but they were deformed rather fast, as the graph shows. At about 3900 cycles, a third attempt was done to tighten the wires with the same result. Table 5.6 shows results from the tested wires. The two first wires were unsuccessful setups and therefore, only the third wire shows useful numbers. The maximum deflection is observed in the first contraction cycle, but as the table shows, after some time, the deflection is down to $0mm$. This is also reflected by the minimal and maximal forces.

Wire	**Cycles**	**Max deflection [cm]**	**Min deflection [cm]**	**Max force [N]**	**Min force [N]**
1	5690	0.5	0	30	0
2	345	0.3	0	30	0
3	5182	2.1	0	15	0

Table 5.6: *Results from testing a Flexinol antagonist. The mechanism suffers from unnecessary large forces that stretches the wires and results in a malfunction*

After these attempts the idea of using an active Flexinol antagonist was not further investigated. This was because of the time demanding and inconvenient assembly of the wires in addition to the rather poor performance described above. Maybe an even more conservative way of controlling the wires could prevent the deformation of the wires. However, the contraction cycle described in figure 5.17 is already containing a delay of 10 seconds between agonist and antagonist. A longer delay would be even more inconvenient if the Flexinol wires are used for example in a robot hand.

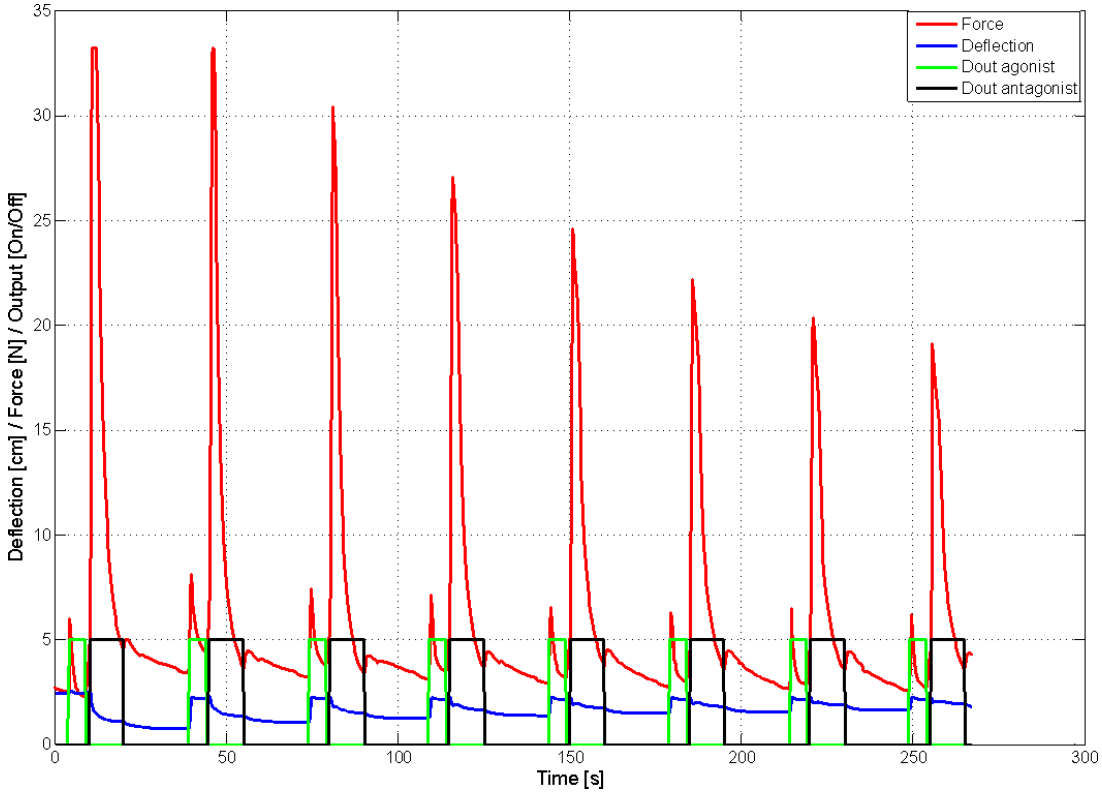


Figure 5.16: *Eight contraction cycles for a Flexinol antagonist. Very large changes can be seen during the cycles*

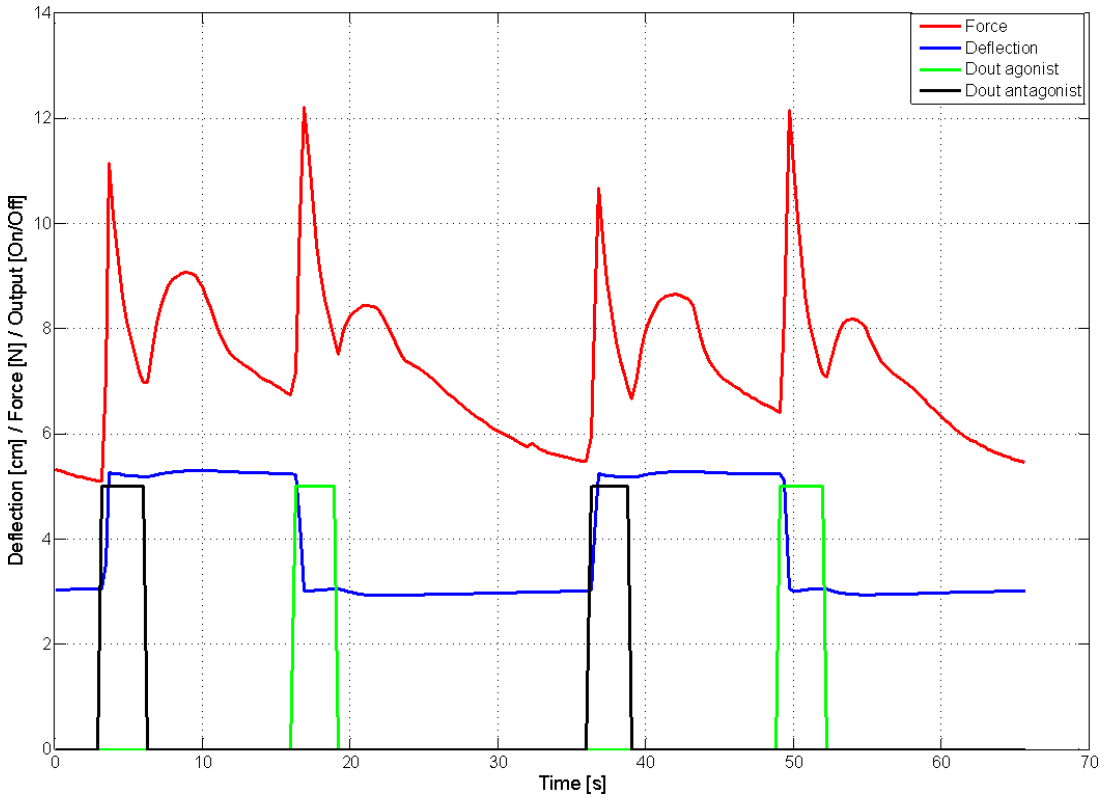


Figure 5.17: *Two contraction cycles for a Flexinol antagonist. A more conservative timing diagram results in more gentle control with smaller forces being exerted*

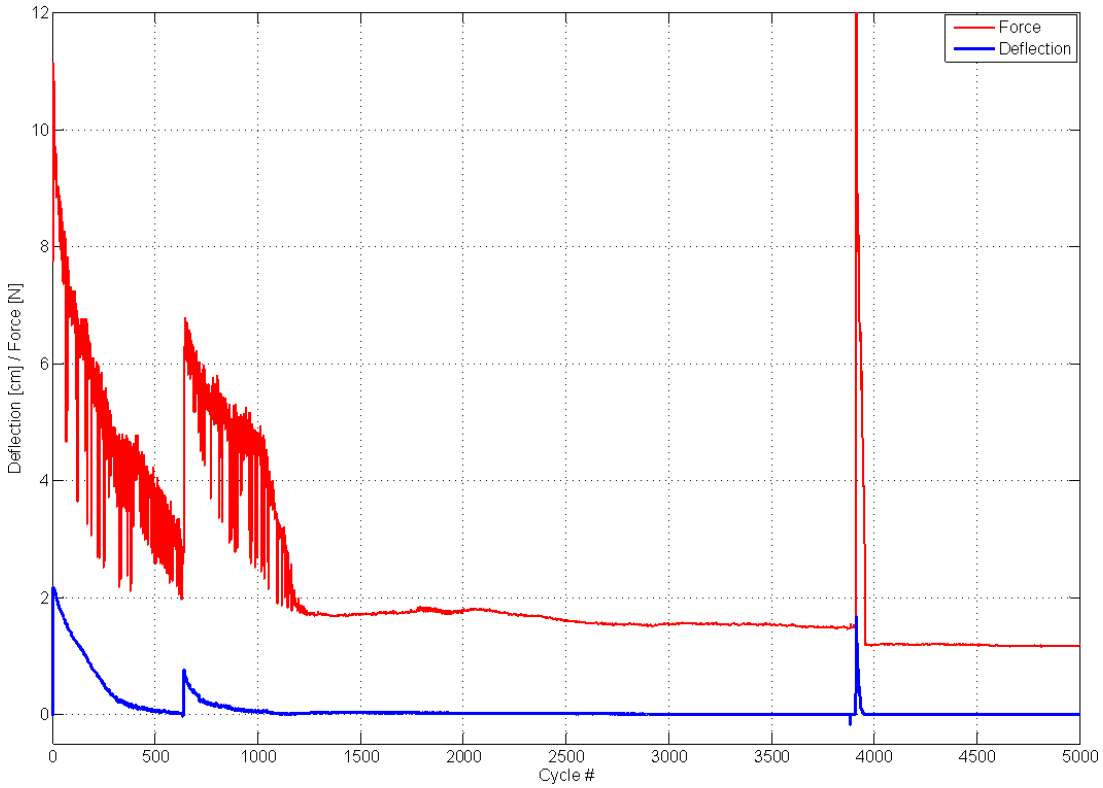


Figure 5.18: *Difference between contracted state and stretched state with Flexinol antagonist. The wires are deformed very quickly, making the deflection converge to 0mm*

5.2.5 Spring Antagonist

The use of a spring as antagonist is described in section 4.1.4. It differs from the use of a Flexinol antagonist in being a passive return force. Two contraction cycles for this test are shown in figure 5.19, where a force measurement can be observed in addition to the digital output signal and the deflection signal. The force measurement may seem redundant as it is proportional to the deflection because of the spring constant. However, it also tells the pretensioning of the spring. The pretensioning is important as it decides the speed of the stretching, but also how much the wire is stressed. This again, also has impact on how much the wire is stretched. A large preload of the wire seems to shorten its lifetime but also causes larger deflections. A smaller preload decreases the deflection slightly, but ensures a slightly longer life of the wire.

Table 5.7 shows results from the tested wires. Wire 1 was tested with almost no pretensioning, reflected by the low deflection rate. Both for wire 2 and 3, the pretensioning was increased, resulting in increased deflection in both cases, but also shorter lives of the wires.

Wire	Cycles	Max deflection [cm]	Min deflection [cm]
1	21497	3.0	2.1
2	20964	3.6	3
3	18836	3.8	3.3

Table 5.7: *Results from testing Flexinol with a spring antagonist. Very stable results can be observed for the life time, but deflection rates are reduced when using a spring as antagonist*

Figure 5.20 shows the development in deflection over time. The graph shows clearly that after a short period of settling, only a very small degeneration of the wire occurs. Although the wire shown in the figure snapped before a convergence was obvious, the trend of the deflection graph seems to be going towards a limit of about 32.5mm.

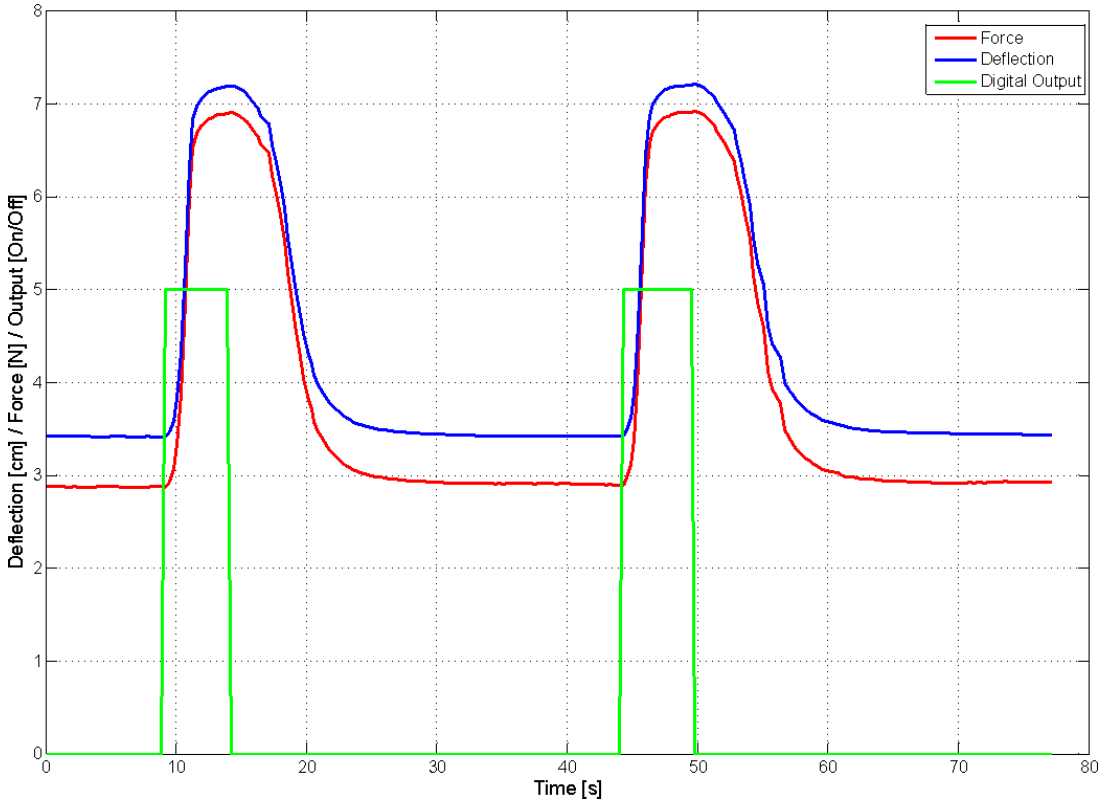


Figure 5.19: *Two contraction cycles for spring antagonist. Very gentle control of the wire is achieved. The force values also reflect this*

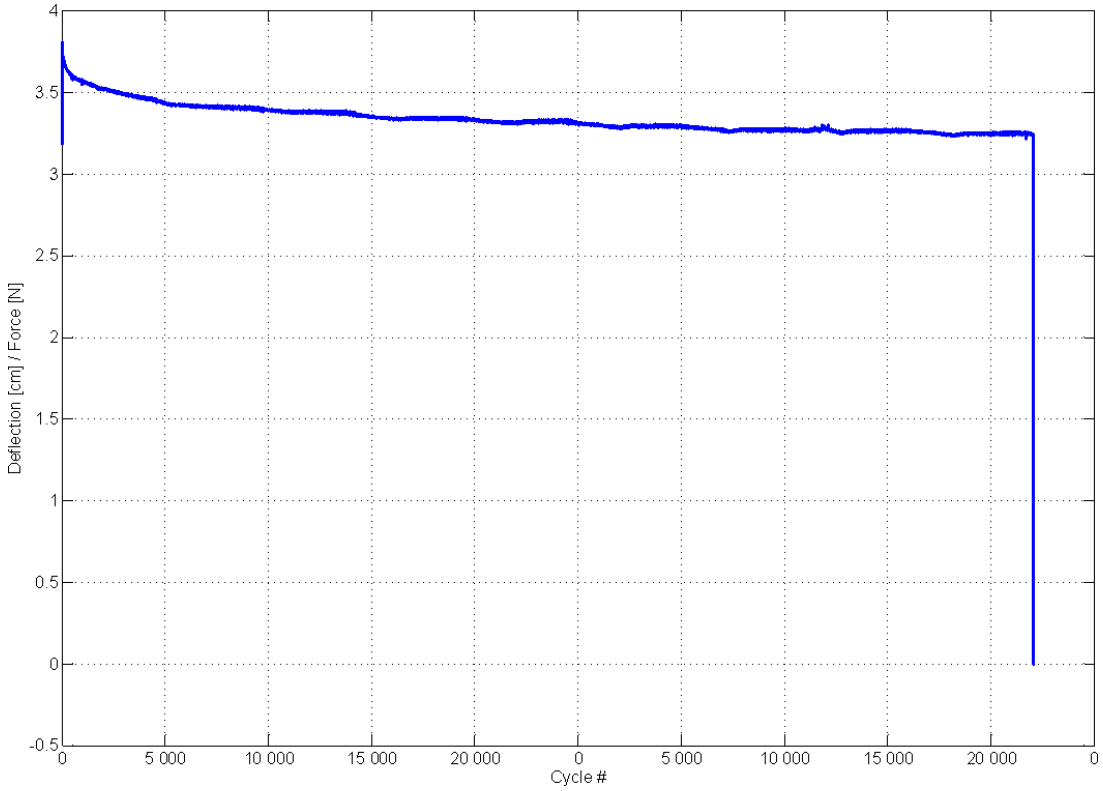


Figure 5.20: *Difference between contracted state and stretched state with spring antagonist. It is apparent that only a small degeneration is done to the wire during 20000 cycles*

5.2.6 PWM-Control

The PWM-control is done with two types of feedback, deflection and current.

Deflection Controlled

The PWM-control of the Flexinol wires was first done with a displacement feedback in the regulation loop. The regulation was performed with the function `SET_DEFLECTIONS` that is a part of the microcontroller command set, described in section 4.5.4. As described in section 4.1.5, the regulation is done on a single Flexinol wire with a spring antagonist as return force.

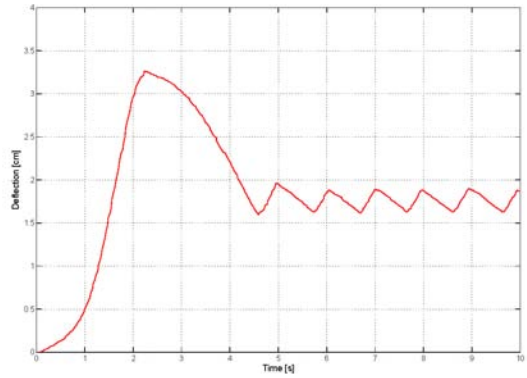
The regulation was done with different regulation factors that are depicted in figure 5.21. The most apparent observation from the results is the oscillation that occurs in all the figures. The figure shows that a small and thus, dampening regulation factor such as 10, causes the largest oscillation amplitude and the lowest oscillation frequency. By increasing the regulation factor to 50, an improvement in the oscillations is visible. By increasing the regulation factor to 100 and 150, further improvement is observed. However, by further increasing the factor to 200, minimal improvement is seen. A regulation factor of 150 was therefore chosen as optimal for this regulation test.

A rather large overshoot can also be seen in all the figures. This is due to hysteresis and delay in the Flexinol wire. This behaviour is further discussed in section 6.2.

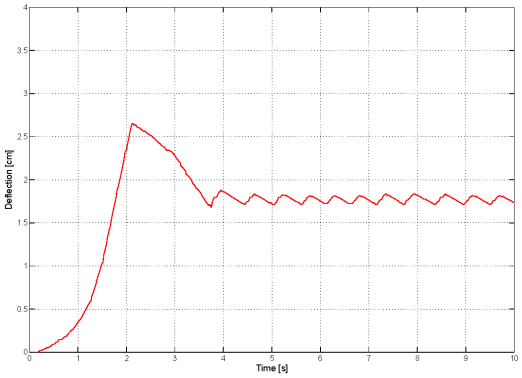
Figure 5.22a shows the result from regulating the Flexinol wire to different degrees of contraction. All plots start with a stretched wire that is contracted to a given degree. Figure 5.22b shows the mean value of all the regulation steps (steady state) plotted against the amount of contraction. It is clear that the regulation is able to control the wire to all degrees of contraction when ignoring the oscillations.

Current Controlled

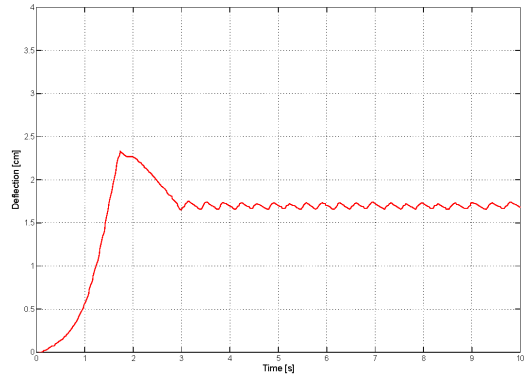
A second regulation was then done with current as feedback in the regulation loop. The regulation was performed with the function `SET_CURRENT` that is a part of the microcontroller command set, described



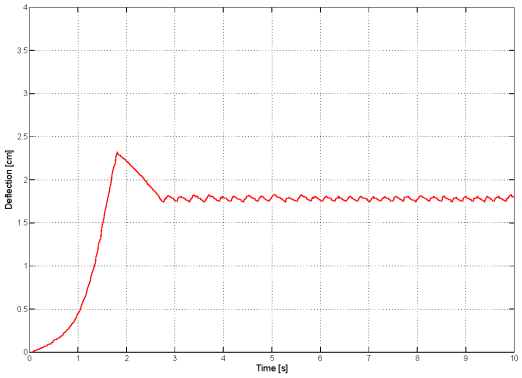
(a) Regulation factor 10



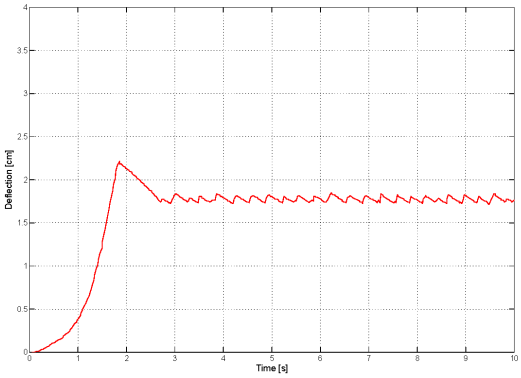
(b) Regulation factor 50



(c) Regulation factor 100

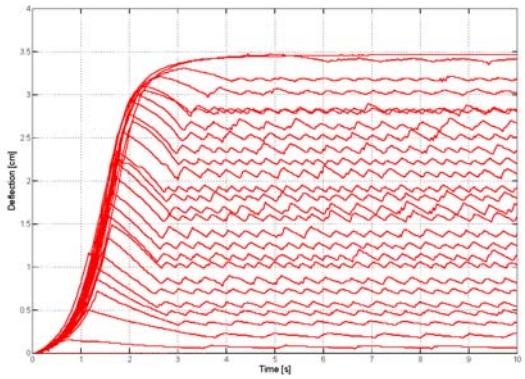


(d) Regulation factor 150. Chosen as the best factor in this test

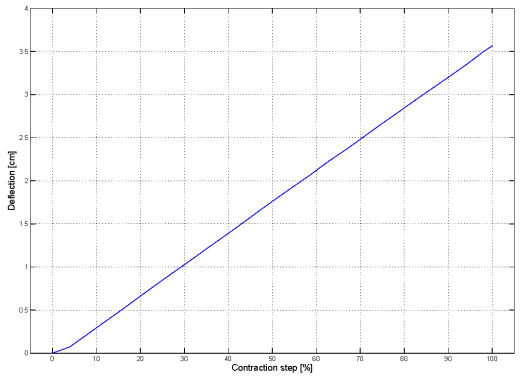


(e) Regulation factor 200

Figure 5.21: *Results of PWM control with different regulation factors. Slow oscillations with large amplitude are seen when using a dampening regulation factor, while for amplifying regulation factors, the oscillations are faster with smaller amplitude*



(a) 27 Regulation steps. The regulation needs about 3s to become stable (with oscillations)

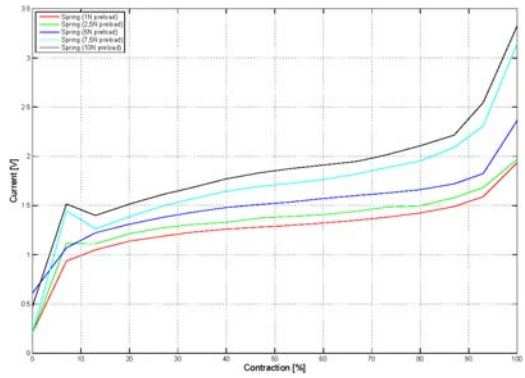


(b) Linearity of regulation. The mean value of each regulation step from figure (a) is plotted against the wanted value. The result shows that all degrees of deflection can be achieved

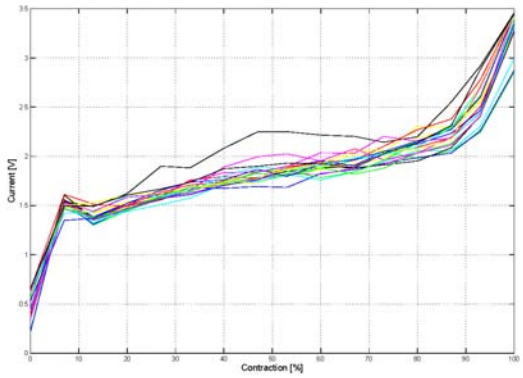
Figure 5.22: *Results from proportional deflection regulation with regulation factor 150*

in section 4.5.4. The regulation is performed on the same setup as described above for the displacement controlled regulation.

Figure 5.23a shows the results from some test measurements to determine the quality of the current measurements. The curves show the current measurements for different pretension forces of the wires. Each point on the curves is an average value from 20 current measurements. When not counting the peak around 7.5%, the curve is monotonic. However, as figure 5.23b shows, rather large deviations are seen within each calibration curve. Many of the curves are not monotonic around 20% to 80%, which may cause trouble when regulating. No explanation was found for the peak that is seen around 7.5% contraction.



(a) Current at different degrees of contraction and with different pretension forces. Each data point is an average of 20 measurements



(b) 20 measurements for a wire with 10N pretensioning. Some of the calibration curves are not monotonic and may therefore cause problems when regulating

Figure 5.23: *Current measurements*

Figure 5.24 shows the calibration values that were measured before the actual regulation was tested. Around 70% to 80%, the curve is non-monotonic, which may cause trouble for the regulation. Only small differences are measured between 25% and 80%, which also may cause problems. However, no peak was seen in this curve.

Figure 5.25a shows the results from the regulation. It is clear that the three first regulation steps are performed correctly. This is reflected by large differences between the steps in the calibration curve from figure 5.24. However, the next 10 steps can only hardly be distinguished from each other. Slow regulation and oscillations are observed, but the main problem is that the regulation steps are not correct. This is also reflected by figure 5.25b, that clearly shows that not all values are regulated correctly. However, the curve does not tell the entire truth, as it contains the mean values of the rather large oscillations from

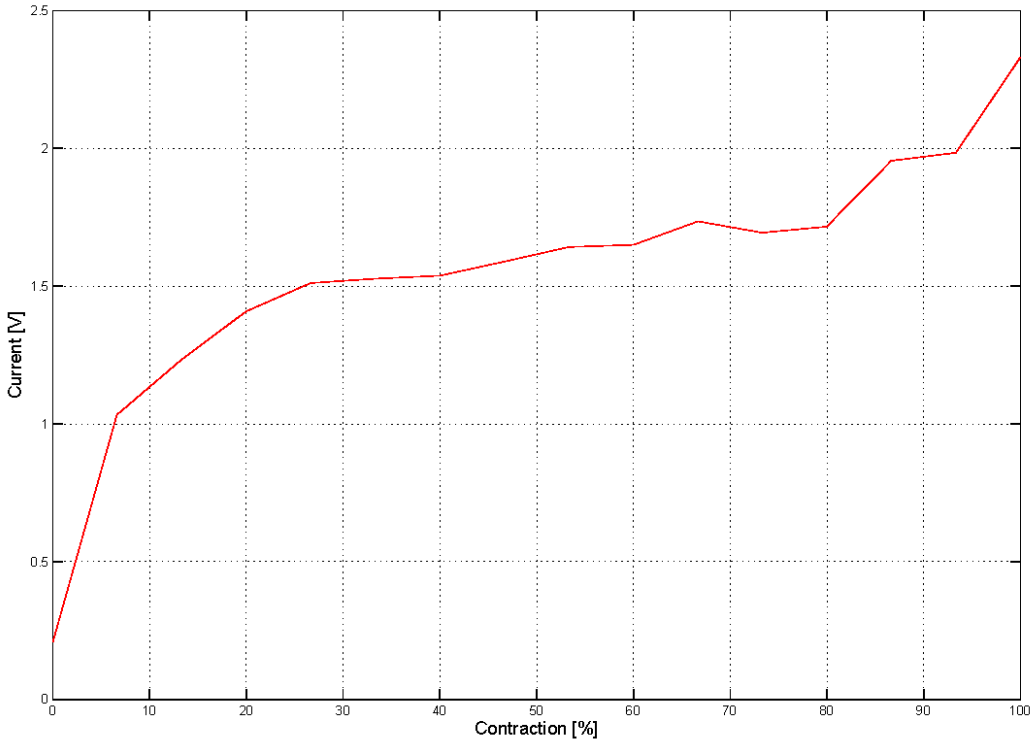


Figure 5.24: *The calibration curve used for the regulation test*

figure 5.25a.

5.2.7 Summary

The testing of Flexinol wires in different setups reveals some very important aspects that are left unmentioned in the documentation provided by the manufacturer Dynalloy, Inc.

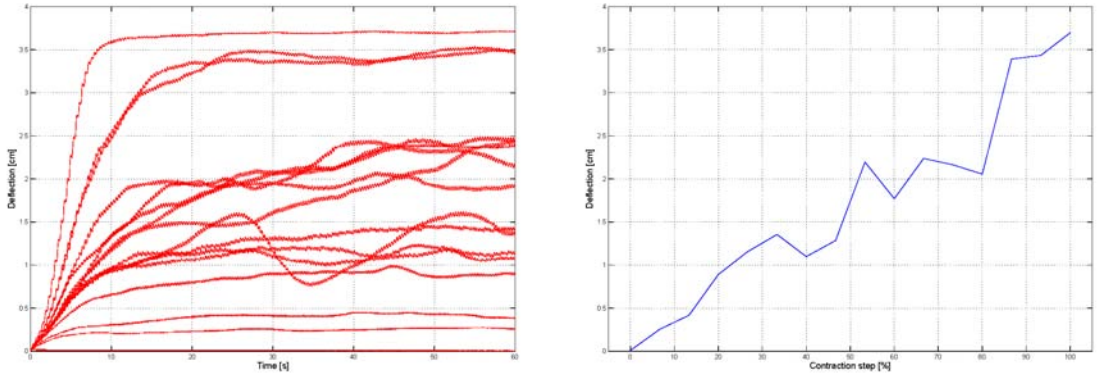
Force

Flexinol wires have proved to be able to exert very impressive forces. However, when loaded to a maximum, the wires have a very short life. This is also informed in the documentation provided by the manufacturer, where it is stated that the lifetime of Flexinol is shortened when loaded to more than the recommended maximum pull force. The documentation does not give any hints of how many contraction cycles to expect from an average wire before it breaks. With a maximum pull force of about $40N$ the wires are able to lift about 3119 times their own weight:

$$
\begin{aligned}
\text{Maximal force of Flexinol } \mathbf{F} &= 40\text{N} \\
\text{Density of Flexinol } \mathbf{D} &= 6.45 g/cm^3 \\
\text{Radius of Flexinol wire } \mathbf{R} &= 0.0254cm/2 = 0.0127cm \\
\text{Length of Flexinol wire } \mathbf{L} &= 100cm \\
\text{Volume of Flexinol wire } \mathbf{V} &= \pi \cdot R^2 \cdot L = \pi \cdot 0.0127cm^2 \cdot 100cm = 0.203cm^3 \\
\text{Weight of Flexinol } \mathbf{M} &= V \cdot D = 0.203cm^3 \cdot 6.45g/cm^3 \approx 1.31g = 0.0128N \\
\\
\text{Pull ratio} &= F/M = 40N/0.0128N = 3119
\end{aligned}
$$

Wire Deformation

In use, the wires are deformed and stretched. This behaviour was not expected when driving the wires gently, but as figure 5.10 shows, this is also the case with a load only slightly heavy enough to stretch



(a) 16 Regulation steps. Some of the steps do not settle

(b) Linearity of regulation. The mean value of each regulation step from figure (a) is plotted against the wanted value. The result shows that only the degrees of contraction close to no contraction or full contraction are correctly regulated

Figure 5.25: *Results from proportional current regulation of a single Flexinol wire*

the wire after a contraction. The deformation of the wire seems to develop faster with a heavier load, as depicted in figure 5.13. However, the deformation of the wire does not seem to have too much influence on the deflection rate. The deflection rate seems to be rather stable after some time of settling.

The effect of the wires being stretched is very dependent on the application in which Flexinol is integrated. In a setting where an object has to be actuated over a given distance, a displacement offset caused by a stretched wire is highly unwanted. In many applications, this would introduce a demand for some kind of tightening mechanism. Such a mechanism does not necessarily have to be very advanced, but compared to only fastening a wire of Flexinol, it is far more complicated. This is for example a huge drawback in applications concerning multiple Flexinol wires.

In some designs where physical space is not an issue and the number of wires is low, a stretched wire may not be a problem. However, in small applications like the Electrostem^{TM} valve presented in section 2.3.2, a replacement would cause a complete disassembly and reassembly of the valve.

Deflection Rate

As mentioned above, the degree of deflection seems to be stable as long as the wire is stretched properly after each contraction. When stretched with a dead weight of ca 1kg, a deflection rate of about 4.25% can be expected, when stretched with a load of 300g, 2.25% and when stretched with a spring, about 3.25% can be expected. This again shows that Flexinol does not fit into all applications. Where a properly sized reverse force is not available, the wires can not be expected to deliver the specifications given in the Flexinol datasheet.

5.3 Humanoid Finger Design

The humanoid finger presented in section 4.4 is evaluated in this section. The finger was produced with a 3D printer in ABS plastic. Each joint is assembled with a 5mm machine screw from each side. Three flexor tendons and one extensor tendon is routed through their respective tendon channels and fastened with screws.

5.3.1 Joints

As described in section 4.4, the joints are designed in such a way that the tendons are routed straight through the pivot point of each joint. This was done to prevent the tendons from distal joints to cause movement in the joints closer to the hand base. To test this function, a soft spring was connected to the extensor tendon to extend the finger. Then, combinations of all the flexor tendons were tested. The design proved to work very well, as no joints were affected by the actuation of other joints.

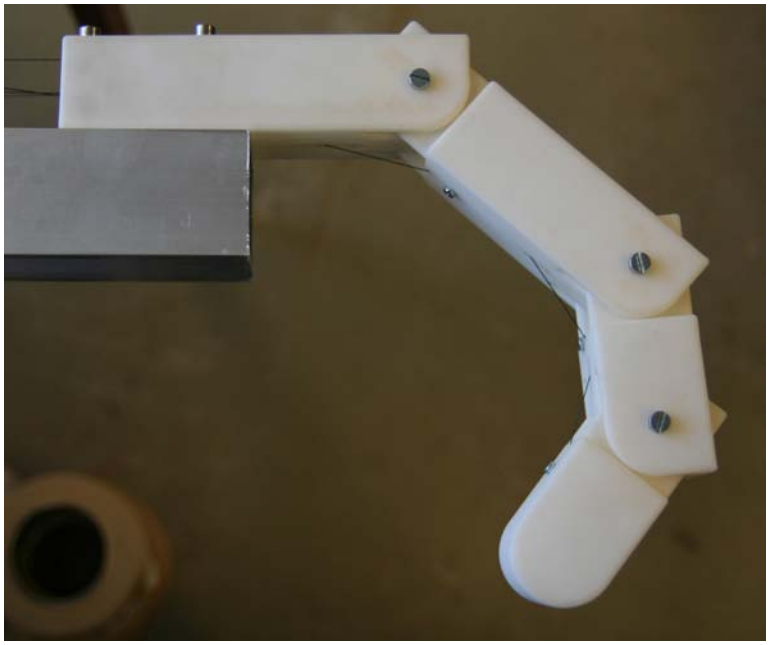


Figure 5.26: *Assembled finger mounted on an aluminium tube. The flexor tendons can be seen on the lower side of each joint. Each joint is actuated about* 45°

5.3.2 Tendons

The tendon channels seems to function good. Very little friction is noticed when actuating the different joints with hand force. As the three flexor tendons share one tendon channel in the first finger segment, there was a chance that this could cause problems due to friction and backlash. This neither seems to be a problem. The way of squeezing the wires in place with a screw also seems to work well. No tendons have come loose when actuated with hand force or when actuated with Flexinol wires. However, as the ABS plastic is a rather soft material, the fastening mechanisms can not be expected to withstand many cycles of assembly and disassembly.

5.3.3 Friction

The surface of the 3D print has a slightly uneven structure that is reflecting the layers and the resolution of the printer. The finger segments where printed in such a way that these structures should be as small as possible on the surfaces of the hinge joints. Nevertheless, these surfaces are not totally even and therefore friction forces are observed. The friction seems to hold back the joint when actuated gently, causing at first no actuation, then a sudden large movement. This behaviour can partly be omitted by applying grease to the joint surfaces. However, it may be a better solution to produce the joints with slightly larger margins between the surfaces or producing the finger in another material.

5.4 Humanoid Finger Application

In this section, the assembled finger is evaluated together with its control electronics and an aluminium tube representing a forearm, holding muscles and tendons.

5.4.1 Mechanics

As described in section 4.5.1, the finger is mounted on a rectangular aluminium tube. The tube is also used to fasten Flexinol wires, displacement transducers, force transducers and the control electronics.

The fastening of the finger was done using the four available screw holes in the metacarpal finger segment (figure 4.19). The finger is positioned in the center of the tube, which ensures a centering of the exit of the tendon channels. The three flexor tendons are separated like figure 5.27 shows and each tendon is then connected to one of the three Flexinol wires. From each of the connection points between

tendon and wire, a thread is routed around the corresponding radial displacement transducer and then fastened to a rubber band.

Displacement Transducer

The depth of the radial displacement transducer was almost causing problems. As figure 5.27 shows, no extra room is left between the displacement transducer and the next wire. Although the physical size of the transducer is a little larger than wanted, it functions well.

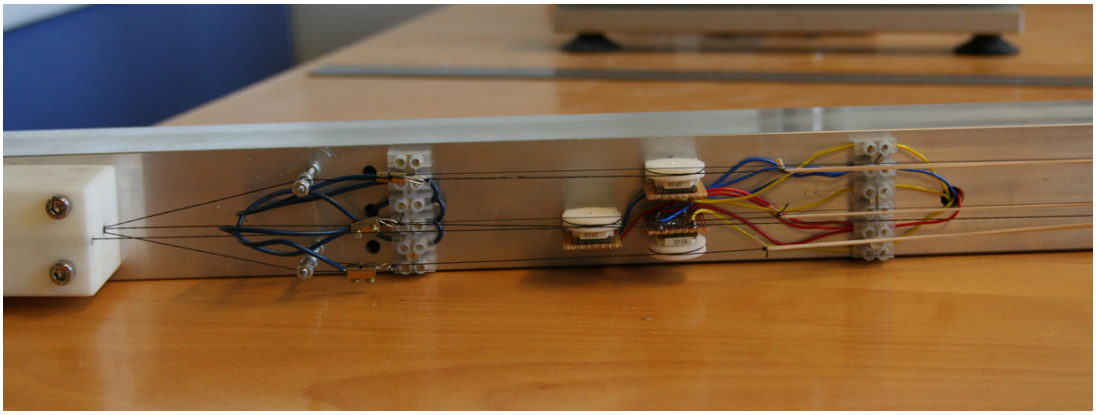


Figure 5.27: *Separation of the tendons. The center tendon is routed straight out of the finger while the outer tendons are separated from the center with two padded machine screws*

Force Sensor Bracket

The Flexinol wire that is fastened to the finger tendon in one end is fastened to the force sensor bracket in the other. The force sensor bracket is printed in ABS plastic, just like the displacement transducer and the finger. Just as experienced with the finger joints, the uneven structures on the surfaces of the force bracket caused it to slide in steps rather than smoothly. This introduced some problems when calibrating the force sensors because the light calibration weights were not heavy enough to overcome the friction forces. A calibration curve for the force sensors can be seen in figure 5.28.

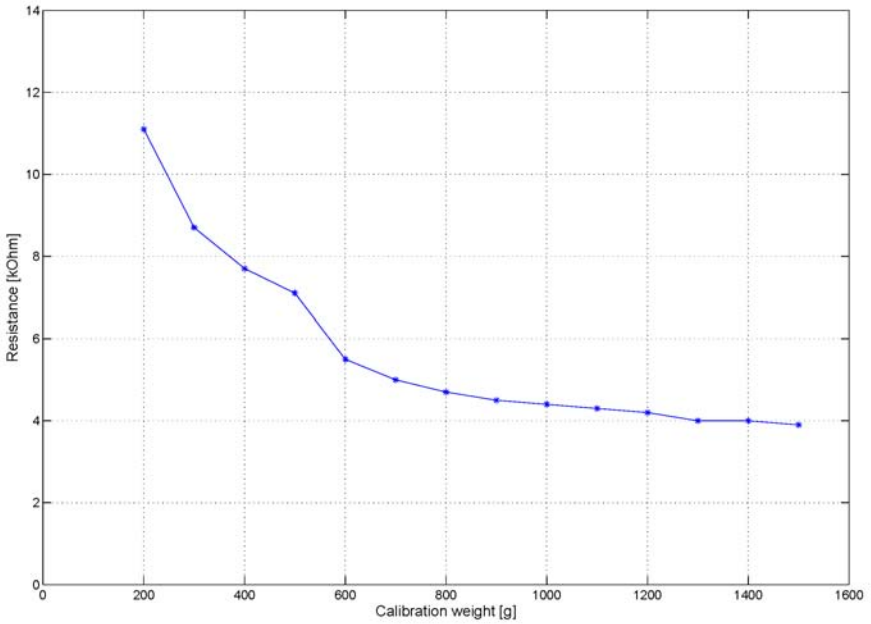


Figure 5.28: *Calibration curve for the Interlink Force Sensor. A non-linear but monotonic behaviour is seen.*

Elastic Displacement Transducer

As described in section 4.5.1, an elastic displacement transducer was considered as an option. However, as figure 5.29 shows, the use of such a transducer is non-trivial. In the figure, the green vertical lines mark each time the elastic transducer was stretched and the black line shows the actual physical stretching. A negative peak is observed by the first stretching. The peak leasts for about one second before the curve moves in the correct direction. However, as the graph shows, the transducer does not immediately reflect the change in displacement. After 10 seconds, the output starts to stabilize, but is slightly increased for another 15 seconds.

The same behaviour is seen again when the transducer is further stretched at around 32 seconds. Also when decreasing the amount of stretch at 40 seconds, a negative peak is seen. The following peak is caused by a mechanical distortion. The fact that such a small distortion causes a great peak in the measurement indicates that regulation with such feedback can be difficult. In addition to the slow response of the transducer, the overall performance is therefore considered too inaccurate for the humanoid finger application.

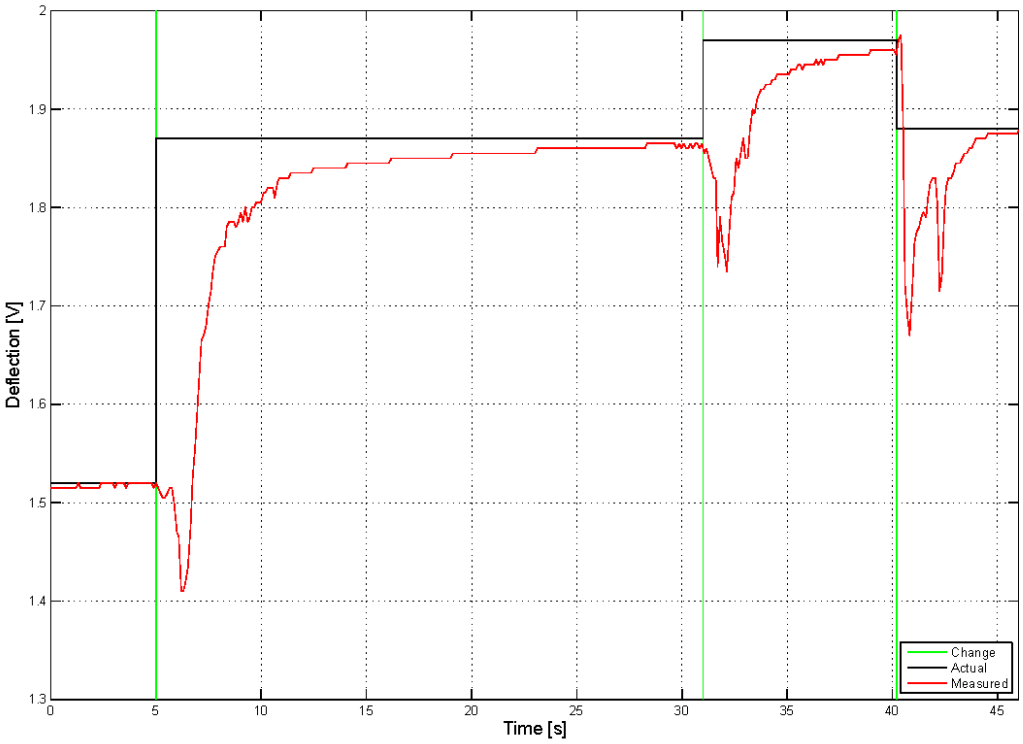


Figure 5.29: *Results from testing of an elastic displacement transducer. The results show that the transducer suffers from large delays. In addition, negative peaks are generated when the transducer is stretched.*

5.4.2 Electronics

The electronics schematic of the finger application is presented in section 4.5.2 and its breadboard realisation can be seen in figure 5.30. The prototype was built on a breadboard to be easily expandable in case of adjustments. The biggest IC is the ATMega32 microcontroller which has a 6-pin programming connector situated on its right side. The smaller IC is the MAX202 level shifter, used to convert between TTL and ±12V signals for RS232. On the right side of the picture, a 5V regulator can be seen with two capacitors for noise filtering of the power supply. On the left side of the picture, three power hexfet transistors can be seen. These are used to drive the Flexinol wires. The large, white ceramic resistors are used for current measurements.

The electronic design is not very complex, but functioned well for its purpose as prototype base for Flexinol wire regulation. The PWM-frequency of the outputs is desirable to keep as high as possible. The power hexfet transistors that are controlled by the PWM-signal will therefore have to keep up with this

frequency. Because of the large gate capacitance of the transistors, a small resistor had to be used to pull up the gate fast enough. With a too small pull up resistor, this resulted in too large power dissipation in the resistor which was fried. This experience caused a limitation in the PWM-frequency, resulting in a lower frequency than wanted.

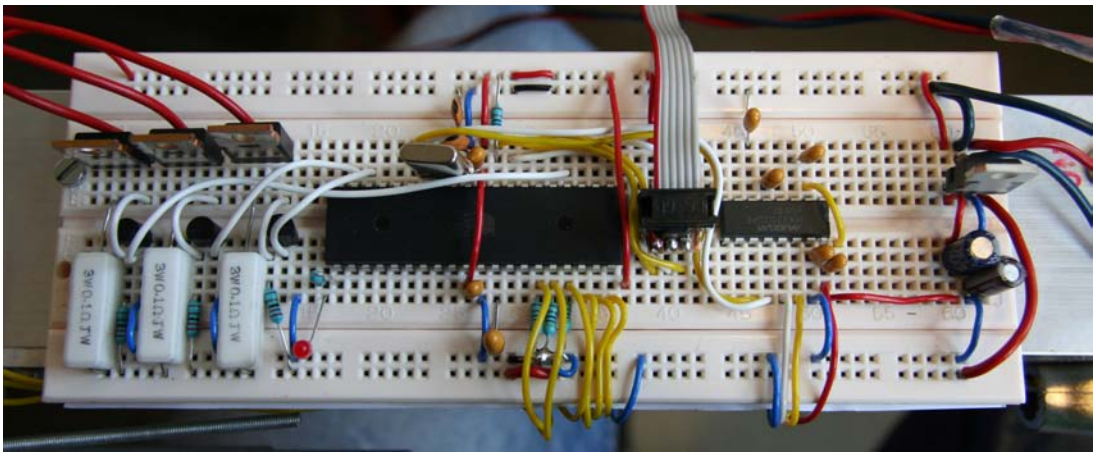


Figure 5.30: *Circuit realised on breadboard. The microcontroller reads sensor data, controls PWM-signals to the Flexinol wires and communicates with a computer via RS232*

5.4.3 Software

In this section, the three different software tools, developed to control the finger, are evaluated. The computer interface program and the interface for Robotics Studio are redundant. As described below, this is mainly because Robotics Studio proved to be a framework rather unsuitable for prototyping.

Microcontroller Software

The program running on the microcontroller is described in section 4.5.4. The purpose of the program is to receive commands via the serial interface of the microcontroller and perform actions accordingly.

The reception of data via the serial interface is done based on interrupts, as described in section 4.5.4. The simplest use of the microprogram is to pass it a command, let it perform an action while waiting for a return value. This highly sequential way of operation has not caused any problems while testing the program. However, when the microprogram was set to perform a continual operation such as regulating one or more wires, a problem occurred. The problem was that the serial communication with the interface program on the laboratory computer stopped. This happened after sending a number of different set values to the micro program while at the same time receiving sensor data from it. This combination of sending and receiving caused the communication to stop although both a send and a receive buffer are implemented in the microcontroller software. However, when not multiple set values are sent to the microcontroller, no problems have been encountered.

The most critical part when performing the requested regulation commands is the regulation loop speed. With the microcontroller running at 14.318MHz, this was never an issue and a prescaler had to be implemented in the regulation loop to prevent it from returning data over the serial interface to often.

Computer Interface Software

The computer interface for the finger application handles the communication with the microcontroller. It is an abstraction from the textual input of commands by providing a list of the available commands and standard values for the command arguments. As described in section 4.5.5, the program lets the user choose between different command sets, defined in input files. This function proved as very usable when testing different versions of the microcontroller program.

Except for the above described problem with a halt in the serial communication, no other problems were encountered. The graph view for continuous sensor data view was very helpful when testing different regulation settings.

Robotics Studio Interface

The interface developed for Microsoft Robotics Studio was an attempt to fit the finger application into an existing framework for robotic applications as described in section 3.3. However, the framework appeared as cumbersome in use. When programming only a small service such as in this case, very much extra work had to be put into the strict definition of the communication and data structure of it. These strict definitions are of course necessary when dealing with larger systems with many concurrent subsystems, but in a prototype setting, the overhead caused by the framework was greatly slowing the productivity down.

However, after defining the interface of the service and implementing its functions, a highly reusable module is at hand. This is a great benefit for example when different programs use the same hardware modules. In practice however, it is often desirable to improve such an interface service. With Robotics Studio, this involves expanding code that already appears as messy when produced by the code generator. Robotics Studio may be a good choice when developing a large modularized concurrent system, but for this prototype purpose it simply was too heavy.

Chapter 6

Regulation

This chapter focuses closer on the regulation results that were given in the last chapter.

6.1 Finger Regulation

In figure 6.1, the results from regulating a finger can be seen. The curves show the angle of each finger joint, and it is clear that oscillations are present. Oscillations are already present when regulating one single wire, as described in the previous chapter. However, the oscillations that occur when controlling the finger are slower than for one single wire. The shape of the finger oscillation looks more pulse shaped than the sine shape of the single wire control. This behaviour is caused by the mechanical construction of the finger as the exact same regulation function is used as for the single wire control.

The surface of the finger joints introduces static friction forces. This can be seen when comparing the deflection curves in figure 6.1 with the force curves in figure 6.2. When looking at the force and deflection curve for the PIP-joint at 6s, it is apparent that the force increases at this point, but no deflection is measured before one second later. The same seems to happen at around 8s when the force decreases. The PIP-joint starts to stretch when no force is applied to the joint.

6.2 PWM-Controlled

The results from regulation of a single Flexinol wire was given in section 5.2.6. The results revealed that a control of a wire is possible, but far from trivial. Three factors are causing the non-trivial behaviour.

6.2.1 Transformation Curve

The most obvious factor can be seen by looking at the transformation curve of Flexinol in figure 2.13. When heating the wire from a cold state, very little change occurs before the wire reaches a temperature of ca $70°C$. Between $70°C$ and $85°C$, the transformation curve is nonlinear, reaching a contraction rate of about 0.5%. However, the range between $85°C$ and $95°C$ results in contraction from 0.5% to 3.5%. This part of the contraction curve spans over $10°C$ and controls 3% of the available 4% of contraction. This means that very small changes in temperature causes very large changes in contraction. At $90°C$, such changes can come from air turbulence caused by someone passing by, or a window being open. This behaviour is also reflected by the different curves in figure 5.22. When looking at the four lower steps and the four upper steps, lower oscillation amplitudes and frequencies can be observed.

To compensate for this behaviour, accurate sensor feedback and current control is needed. By developing a model for the wire contraction, a better regulation could possibly be done.

6.2.2 Hysteresis

The hysteresis of the Flexinol wires is already mentioned in section 2.3.1 and can clearly be seen in figure 2.13. The hysteresis can easily be modelled for a complete cycle of heating and cooling, but it is unclear from the contraction curve what happens if the heating is stopped at around 2% and then the wire is cooled. Does this lead to a smaller hysteresis or is the behaviour unchanged? This question

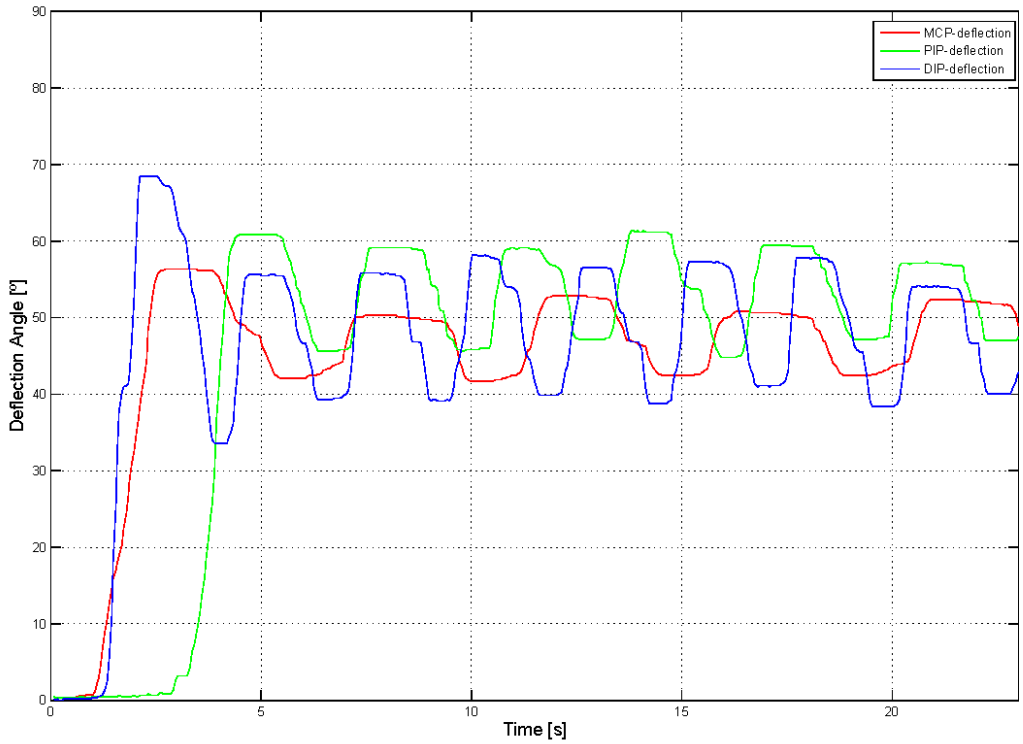


Figure 6.1: *Regulation of a finger where each joint has a set value of* $45°$*. The curves show the angle of each joint. Oscillations of about* $\pm 10°$ *are observed*

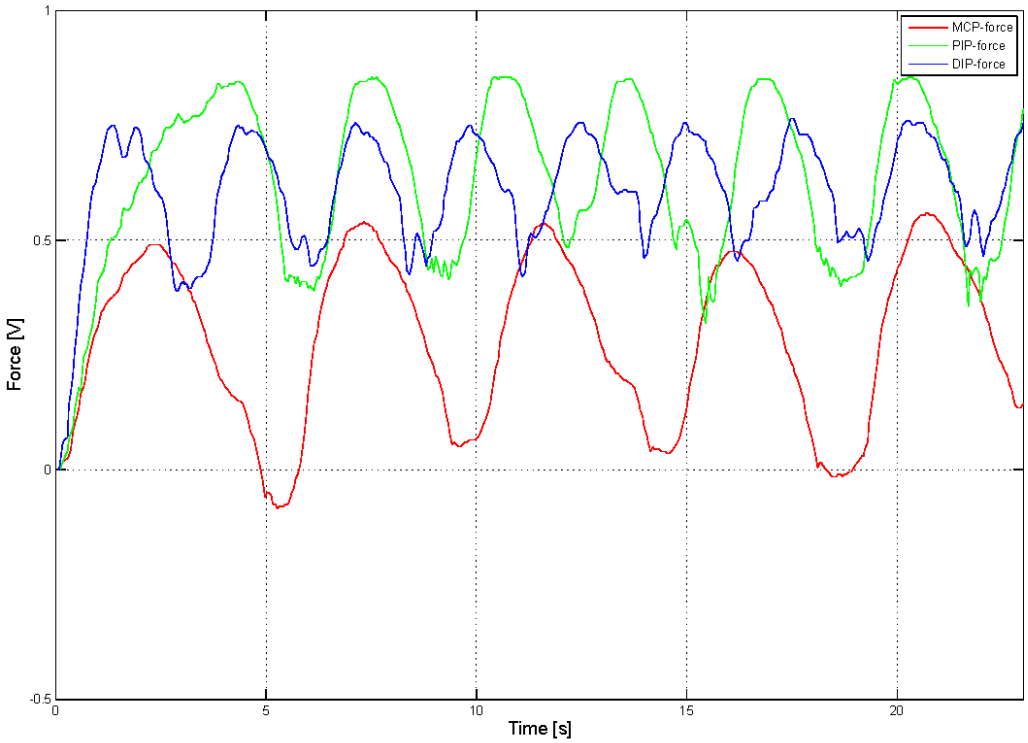


Figure 6.2: *Force measurements from the same regulation as figure 6.1. The oscillations are also seen as changes in force. The MCP-joint exerts the largest forces because of its tendon fastening*

is difficult to answer as no temperature measurements have been performed during this thesis due to technical difficulties and the small size of the wire. The hysteresis behaviour makes it necessary to keep record of the current state of the wire while regulating. A regulation model will in other words need to know whether or not it must expect hysteresis behaviour when loaded with a new set point.

6.2.3 Delay

The third factor that makes regulation non-trivial is the delay that occurs after a change in current and before a change in contraction. The delay is caused by the time needed to heat the wire. When increasing or decreasing the current flowing through the Flexinol wire, the heating or cooling of the wire begins immediately, but it takes some time before the process is completed. As the curves in figure 5.21 show, a rather large overshoot can be observed when controlling the wire from 0% contraction to a given contraction rate. This is caused by the overall delay in the system, including both regulation and wire delay. The proportional regulation used in section 5.2.6 is probably not the best solution to regulate the wire, because it does not predict any behaviour like this. It increases the output until it reaches its set value and then decreases the output to work against the overshoot. However, as this system contains a rather long delay, more overshoot than wanted is observed. figure 6.3 shows on contraction curve of a Flexinol wire. A delay of 2.5s is seen between the start and the end of the contraction. This is 2.5 times longer than specified in the Flexinol datasheet.

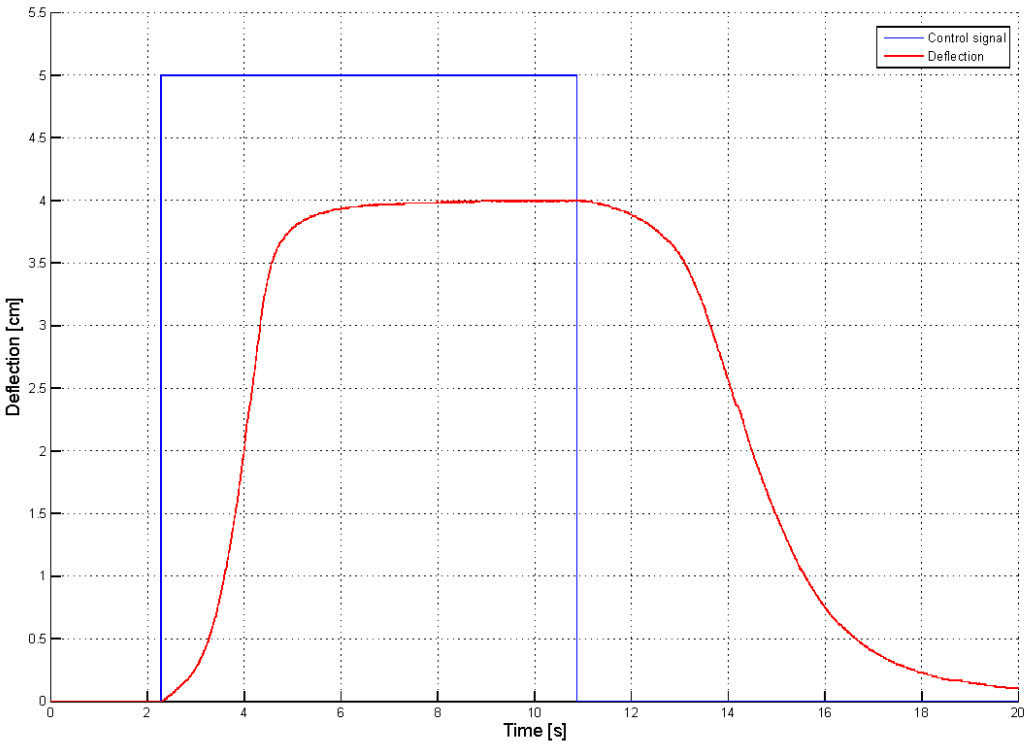


Figure 6.3: *One contraction cycle of Flexinol with high resolution. A delay of ca* 2.5s *can be seen when contracting the wire. This is 2.5 times more than specified in the datasheet*

6.3 Regulation Models by Other Authors

As already mentioned, some approaches have already been performed by other authors to investigate the possibility to regulate the contraction rate of Flexinol.

Grant and Hayward present a working two-stage regulation mechanism in [71]. An antagonistic setup with two high strain SMA actuators [22] is controlled to exert a specified force. The two-stage controller first uses a high regulation factor to kick-start the regulation. When a threshold value is reached, a

smaller regulation factor is used to fine tune the regulation. The authors report good performance from step regulation.

Ma and Song present another regulation model in [73] where a PWM-controlling PD-controller (proportional, derivative) is used. No tracking- or step response curves are shown, but the authors report that they achieve working regulation.

The most recently written paper found is written by Yee Harn Teh and Roy Featherstone [74]. The authors use a PID-controller (proportional, integral, derivative) in combination with anti-overload and ant-slack mechanisms to control the force of two antagonistic SMA wires. Good tracking response and step response is reported.

Song, Chaudhry and Batur present a working setup in [72] that uses a feedforward neural network to regulate a SMA wire and to reduce hysteresis behaviour.

6.4 Own Regulation Experiments

The oscillations (figure 5.21) that occurred when regulating Flexinol wires were assumed to be caused either by the PWM-control signal or by the regulation algorithm. A quick experiment was therefore done to regulate a Flexinol wire manually. The setup is depicted in figure 6.4 and consists of a microcontroller that reads a voltage signal from a potentiometer. The voltage value (0-5V) is mapped linearly to a pwm output (0-100% duty cycle) that controls the RN-VN2 dual motor driver [79]. The RN-VN2 is built upon the motor driver IC VNH2SP30 [80] which delivers a 20kHz output signal.

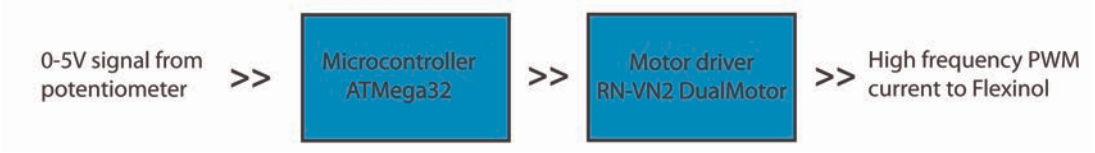


Figure 6.4: *Block diagram of manual regulation. The voltage signal from the potentiometer is mapped directly to a pwm signal from the motor driver*

The manual regulation of the wires proved to work very well. By controlling the current per hand, all degrees of contraction could be regulated. When touching the test frame during the regulation, vibrations with high frequency could be sensed, probably reflecting the 20kHz signal of the motor driver. Figure 6.5 shows that different degrees of contraction could be regulated manually.

6.5 Summary

When combining own experiments with examples found in the literature, it is apparent that several methods can be used to regulate Flexinol. Examples of two-stage P-regulation, PD-regulation, PID-regulation and neural networks have been seen. However, because SMA wires of different dimensions have been used, the results are not always easily comparable. Thinner wires will react faster both upon heating and cooling, but will also exert smaller forces.

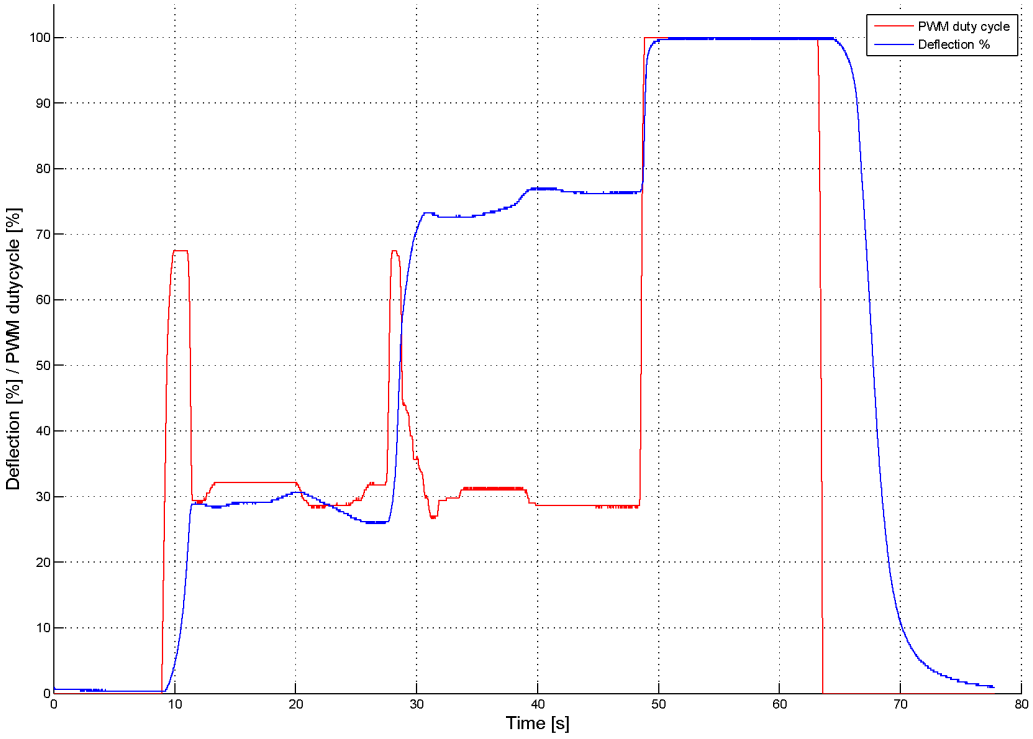


Figure 6.5: *Manual regulation of a Flexinol wire. The curve shows clearly that regulation is feasible*

Chapter 7

Future Work

Flexinol wires have been on the market for several years, but very few real life applications have been found during this thesis. It seems like little basis research has been done in the field, maybe one of the reasons that make the inclusion of Flexinol in larger systems rare. This thesis aims to design a humanoid finger which is to be actuated with Flexinol wires and some tests of Flexinol had to be performed before the finger could be designed. During the scope of the thesis, the goal was to develop a working finger. To achieve this goal, some milestones had to be reached faster than with another time perspective. This chapter therefore focuses on the aspects of the thesis that deserves more attention in addition to interesting problems that were discovered during development.

7.1 Flexinol Testing

The results from testing of Flexinol wires in a test frame show that this is very time consuming work. Wires have to be changed and data sets need to be analyzed. As an example for this is the testing with a small load where one wire lasted for 40 days. To get enough empirical data to conclude exactly how Flexinol wires can be used in the best way, many more tests should be performed. This would include building a bigger test frame so that many equal test setups can be run in parallel. Deflection transducers with a working range of more than $50mm$ should also be used so that the wires don't have to be tightened during runtime. A solution could be to use radial displacement transducers for all wires.

A fastening mechanism should also be developed to fasten the fixated Flexinol wire in a more gentle way when testing. In this way, more accurate measures of how long the wire lasts can be collected. Although the use of a Flexinol wire as antagonist was abandoned during this thesis, the concept should also be closer investigated.

Attempts were made to measure how much a wire had been stretched after it had broken. This proved to be very difficult as the wires broke in different contraction states. An attempt was made to stretch the wire and then measure, but this resulted in the wire also breaking other places. This challenge could maybe be overcome by always measuring the wires in a contracted state, thus not having to stretch the wire after it is broken. This would reaveal information about how much a wire can be stretched before it breaks. In this way a better prediction can hopefully be done.

7.2 Regulation

During this thesis, only one regulation approach is done in addition to an successful attempt to manually regulate Flexinol. A regulation algorithm for a single Flexinol wire should be developed in the future. If such an algorithm is able to control a single wire without oscillations, chances are good that a finger also can be regulated.

7.3 Developed Finger

The prototyped finger was printed in ABS-plastic and the results from this process are described in section 5.3. The finger was designed with a size ratio of 3:1. A second finger should be produced in a

1:1 ratio and preferably in metal to get even surfaces to keep friction at a minimum. To achieve the true movements of the human finger, the MCP-joint should also support adduction and abduction as described in section 4.4.1. In this way, the kinetic sketch of the finger would be the same as the human finger except from tendon routing.

Touch sensors or force sensors could be mounted on the inside of the fingers, making grasping operations possible. By also producing the finger with a rubber like material on the inside, grasping would be even more feasible. If good control of one finger is achieved, it would be very interesting to see if a whole hand could be assembled from the fingers. In this case, a thumb should also be developed, a non-trivial task both mechanically and in terms of regulation.

7.4 Electronics

The electronic circuits built during the thesis were realized on bread board. For a second prototype, it would be preferable with a smaller realization on a printed circuit board (PCB). It would also be preferable with a much higher PWM-frequency for the output, alternatively could a motor driver like the L6205 [81] from STMicroelectronics be used to control the current flowing through the Flexinol wires. In this way, the current measurements would probably be much more accurate be cause of the elimination of the PWM-shaped current flow.

Chapter 8

Conclusion

This thesis describes the development of a humanoid finger that is actuated with three Flexinol artificial muscle fibers. The goal is to investigate whether or not Flexinol wires are suitable as actuators in such an application an how well they perform. This conclusion summarizes the answers to the problems posed at the end of the introduction chapter.

- Does Flexinol stand the long time use as a robot hand actuator?

The thesis starts with an investigation of the physical long term properties of Flexinol. Different tests are performed that reveal interesting behaviours not published by the manufacturer. First of all, Flexinol wires are deformed after some time of actuation. An accurate amount could not be measured, but for a wire loaded to its maximum, a deformation of about 1% could be observed after a week of actuation. A positive observation while testing is that the wires exert even higher forces than given the datasheet. All in all, the testing of Flexinol revealed that the wires can be used as actuators in robotics, but that care should be taken not to overload or underload them. The stretching force is 1/3 of the maximum allowed force, making the dynamic work range of the wires somewhat limited.

- Are the properties of Flexinol changing over time?

Except for the deformation of the Flexinol wire, only small changes are seen when the wire is properly stretched after each contraction. This includes very little decrease in strain rate and contraction speed. This observation is positive when looking solely on whether or not Flexinol is suitable as a robotic actuator. However, this observation also makes it difficult to predict when a wire is about to break, as no properties are dramatically changing before the break occurs.

- Is it possible to regulate the contraction of Flexinol when used as actuator for a humanoid finger?

The first attempt to regulate a Flexinol wire is done on a single wire with a PWM signal from an ATMega32 microcontroller. The results show clearly that regulation of the wire is possible. However, the proportional regulation in use causes some oscillations that leave room for improvement.

A humanoid finger is then designed and prototyped with a 3D printer. The finger has a tendon routing scheme that makes each joint independent of each other. A control mechanism is implemented in an ATMega32 microcontroller and a displacement transducer is developed for each joint. The regulation described above is then used to regulate the three joints of the finger individually. The results show that the oscillations described above are amplified through friction in the finger.

Although some difficulties are uncovered regarding regulation of the finger, the results show that a regulation of Flexinol is possible with a more sophisticated regulation loop. This is also seen when looking at the work of other authors that have applied PID-regulation and Neural Networks successfully.

- Is it possible to improve the strain rate of Flexinol?

This question is not directly answered through practical experience. However, a practical example was given in the background chapter from the author of the paper *Design of shape memory alloy actuator with high strain and variable structure control* [22]. This gear mechanism is reported to improve the strain rate of Flexinol to 18% instead of the original 4.5%. This is a great improvement, that makes the use of Flexinol in different applications possible. However, the gear makes the size of the actuator very large compared to a single wire.

- Is it possible to use multiple Flexinol wires in parallel?

Whether or not a parallel use of Flexinol wires is possible is not answered in this thesis through experiments. However, the experiences gained through experiments with single Flexinol wires have shown that this may be complicated. In many setups, variances are seen between different wires regarding how fast and how much they are stretched. These differences are mainly caused by the use of different loads for the wires. This implicates that in a parallel setup, all wires should be equally loaded. To achieve this, a very accurate and homogenous fastening of each wire has to be performed. In [21], a parallel bundle actuator is presented. However, the article does not report whether or not the parallel wires behave homogenous.

Bibliography

[1] K.B. Fite, T.J. Withrow, Xiangrong Shen, K.W. Wait, J.E. Mitchell, and M. Goldfarb. A gas-actuated anthropomorphic prosthesis for transhumeral amputees. *Robotics, IEEE Transactions on*, 24(1):159–169, Feb. 2008.

[2] Kathryn J. De Laurentis and Constantinos Mavroidis. Mechanical design of a shape memory alloy actuated prosthetic hand. *IOS Press, Technology and Health Care*, 10(2):91–106, 2002.

[3] B. Massa, S. Roccella, M.C. Carrozza, and P. Dario. Design and development of an underactuated prosthetic hand. In *Robotics and Automation, 2002. Proceedings. ICRA '02. IEEE International Conference on*, volume 4, pages 3374–3379 vol.4, 2002.

[4] Kenji Kaneko, Kensuke Harada, and Fumio Kanehiro. Development of multi-fingered hand for life-size humanoid robots. In *ICRA*, pages 913–920. IEEE, 2007.

[5] Kiyoshi Hoshino and Ichiro Kawabuchi. Mechanism of humanoid robot arm with 7 dofs having pneumatic actuators. *IEICE Trans Fundamentals*, E89-A(11):3290–3297, 2006.

[6] Naoki Fukaya, Shigeki Toyama, Tamim Asfour, and Rdiger Dillmann. Design of the tuat/karlsruhe humanoid hand, May 17 2000.

[7] Shadow Robot Company. Design of a dextrous hand for advanced clawar applications www.shadowrobot.com/downloads/dextrous_hand_final.pdf.

[8] A. Kargov, T. Asfour, C. Pylatiuk, R. Oberle, H. Klosek, S. Schulz, K. Regenstein, G. Bretthauer, and R. Dillmann. Development of an anthropomorphic hand for a mobile assistive robot. *Rehabilitation Robotics, 2005. ICORR 2005. 9th International Conference on*, pages 182–186, June-1 July 2005.

[9] H. Kawasaki, T. Komatsu, and K. Uchiyama. Dexterous anthropomorphic robot hand with distributed tactile sensor: Gifu hand ii. *Mechatronics, IEEE/ASME Transactions on*, 7(3):296–303, Sep 2002.

[10] Bertrand Tondu, Serge Ippolito, Jérémie Guiochet, and A. Daidie. A seven-degrees-of-freedom robot-arm driven by pneumatic artificial muscles for humanoid robots. *I. J. Robotic Res*, 24(4):257–274, 2005.

[11] Bielefeld University. www.techfak.uni-bielefeld.de/ags/ni/projects/m6setup/hand.html.

[12] B. Almasri and F. B. Ouezdou. New design of one motor driven under actuated humanoid hand. *Intelligent Robots and Systems, 2007. IROS 2007. IEEE/RSJ International Conference on*, pages 1491–1496, Oct. 29 2007-Nov. 2 2007.

[13] Karsten Berns, Tamim Asfour, and Rüdiger Dillmann. Armar an anthropomorphic arm for humanoid service robot. In *ICRA*, pages 702–707, 1999.

[14] Tamim Asfour, Karsten Berns, and Rüdiger Dillmann. The humanoid robot armar: Design and control. pages 7–8, 2000.

[15] T. Dutta and T. Chau. A feasibility study of flexinol as the primary actuator in a prosthetic hand. In *IEEE CCECE 2003, Canadian Conference on Electrical and Computer Engineering*, volume 3, pages 1449–1452 vol.3, 2003.

[16] F. Gori, D. Carnevale, A. Doro Altan, S. Nicosia, and Pennestrí. A new hysteretic behavior in the electrical resistivity of flexinol shape memory alloys versus temperature. *International Journal of Thermophysics*, 27(3):866–879, May 2006.

[17] H.H. Selden, B.; Kyu-Jin Cho; Asada. Segmented binary control of shape memory alloy actuator systems using the peltier effect. *Robotics and Automation, 2004. Proceedings. ICRA '04. 2004 IEEE International Conference on*, 5:4931–4936 Vol.5, 26 April-1 May 2004.

[18] M G Faulkner T B Wolfe (née Bunton) and J Wolfaardt. Development of a shape memory alloy actuator for a robotic eye prosthesis. *Smart Materials and Structures*, 14(4):759–768, 2005.

[19] W M Ostachowicz A J Zak, M P Cartmell and M Wiercigroch. One-dimensional shape memory alloy models for use with reinforced composite structures. *Smart Materials and Structures*, 12(3): 338–346, 2003.

[20] Pavel L. Potapov and Edson P. Da Silva. Time response of shape memory alloy actuators. *Journal of Intelligent Material Systems and Structures*, 11(2):125–134, 2000.

[21] Michael J. Mosley and Constantinos Mavroidis. Design and dynamics of a shape memory alloy wire bundle actuator, 1999.

[22] V. Grant, D.; Hayward. Design of shape memory alloy actuator with high strain and variable structure control. *Robotics and Automation, 1995. Proceedings., 1995 IEEE International Conference on*, 3:2305–2312 vol.3, 1995.

[23] Wikipedia. Hand en.wikipedia.org/wiki/Hand, Aug 2008.

[24] Wikipedia. Muscle en.wikipedia.org/wiki/Muscle, Oct 2008.

[25] Johannes Sobotta. *Atlas of Human Anatomy, Volume 1 Head, Neck, Upper Limb*. Urban & Fischer, 2001.

[26] John Lin, Ying Wu, and T.S. Huang. Modeling the constraints of human hand motion. *Human Motion, 2000. Proceedings. Workshop on*, pages 121–126, 2000.

[27] Gates Corporation. Introduction to hydraulics, www.gates.com/common/downloads-/files/Gates/autoEducation/428-7153.pdf.

[28] S. Schulz, C. Pylatiuk, and G. Bretthauer. A new ultralight anthropomorphic hand. In *Robotics and Automation, 2001. Proceedings 2001 ICRA. IEEE International Conference on*, volume 3, pages 2437–2441 vol.3, 2001.

[29] Jozef Barycki, Miroslaw Ganczarek, Waclaw Kollek, Tadeusz Mikulczynski, and Zdzislaw Samsonowicz. Performances of high-speed pneumatic drive with self-acting impulse valve. *Mechanism and Machine Theory*, 39(6):657–663, June 2004.

[30] F. Oaerden and D. Lefeber. Pneumatic artificial muscles: actuators for robotics and automation. *European Journal of Mechanical and Environmental Engineering*, 47:11–21, 2002.

[31] Hitec RCD. Servo list, www.hitecrcd.com/product_file/file/124/Servo_Chart.pdf.

[32] Hitec RCD. Datasheet hitec hsr-8498hb, www.hitecrcd.com/product_file/file/109/HSR-8498HB_GENERAL_SPECIFICATION_050623.pdf.

[33] Johnson Electric. Datasheet ledex tubular solenoids, www.ledex.com/ltr2/access.php?file=pdf/Tubular_Section_M.pdf.

[34] Dialight BLP. Datasheet black night series 124, www.blpcomp.com/products/datasheet/bkseries124.pdf.

[35] C. Mavroidis. Development of advanced actuators using shape memory alloys and electrorheological fluids. *Research in Nondestructive Evaluation*, 14(1):1–32, January 2002.

[36] Dynnaloy, Inc. Flexinol actuator wire heat curves, www.dynalloy.com/docs/25ksi90C1.pdf.

[37] Dynnaloy, Inc. Technical characteristics of flexinol actuator wire, www.dynalloy.com/docs/TCF1140RevD.pdf.

[38] R.A. Russell and R.B. Gorbet. Improving the response of sma actuators. *Robotics and Automation, 1995. Proceedings., 1995 IEEE International Conference on*, 3:2299–2304 vol.3, May 1995.

[39] N. Troisfontaine, P. Bidaud, and M. Larnicol. Optimal design of micro-actuators based on sma wires. *Smart Material Structures*, 8:197–203, April 1999.

[40] N. Lakhkar, M. Hossain, and D. Agonafer. Cfd modeling of a thermoelectric device for electronics cooling applications. *Thermal and Thermomechanical Phenomena in Electronic Systems, 2008. ITHERM 2008. 11th Intersociety Conference on*, pages 889–895, May 2008.

[41] Custom Thermoelectric. Thermoelectric cooler 00711-5l31-03ca www.customthermoelectric.com/tecs/pdf/00711-5L31-03CA_spec_sht.pdf.

[42] KRYOTHERM. Thermoelectric cooler snowball-71 www.eureca.de/pdf/cooling/peltier-elements/SnowBall-71.pdf.

[43] Dynnaloy, Inc. Information on the electrostem® valve, www.dynalloy.com/docs/electrostem.pdf.

[44] Jonathan W. Mills. Stiquito: A small, simple, inexpensive hexapod robot part 1. locomotion and hard-wired control. Technical report, Computer Science Department; Indiana University; Bloomington, Indiana 47405, April 21 1999.

[45] J.M. McClain, A.; Conrad. Software design of the stiquito micro robot. *Proceedings IEEE Southeast Con, 8-10 April 2005.*, pages 143–147, 2005.

[46] Alberto Paiva and Marcelo Amorim Savi. An overview of constitutive models for shape memory alloys. *Mathematical Problems in Engineering*, 2006:30 pages, 2006.

[47] Qin Chang-jun, Ma Pei-sun, and Yao Qin. A prototype micro-wheeled-robot using sma actuator. *Sensors and Actuators A: Physical*, 113(1):94–99, June 2004.

[48] Technicsche Universität Darmstadt. Elektroaktive polymere, www.emk.tu-darmstadt.de/institut/fachgebiete/m_ems/forschung/dielektrische_polymeraktoren.

[49] Yoseph Bar-Cohen. Bionic humans using eap as artificial muscles reality and challenges. *International Journal of Advanced Robotic Systems*, 1(3):217–222, November 08 2004.

[50] Kentaro Takagi, Masanori Yamamura, Zhi-Wei Luo, Masaki Onishi, Shinya Hirano, Kinji Asaka, and Yoshikazu Hayakawa. Development of a rajiform swimming robot using ionic polymer artificial muscles. *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*, pages 1861–1866, Oct. 2006.

[51] J.D.W. Madden, N.A. Vandesteeg, P.A. Anquetil, P.G.A. Madden, A. Takshi, R.Z. Pytel, S.R. Lafontaine, P.A. Wieringa, and I.W. Hunter. Artificial muscle technology: physical principles and naval prospects. *Oceanic Engineering, IEEE Journal of*, 29(3):706–728, July 2004.

[52] Jaewook Ryu, Younkoo Jeong, Younghun Tak, Byungmok Kim, Byungkyu Kim, and Jong-Oh Park. A ciliary motion based 8-legged walking micro robot using cast ipmc actuators. *Micromechatronics and Human Science, 2002. MHS 2002. Proceedings of 2002 International Symposium on*, pages 85–91, 2002.

[53] N.N. Pak, S. Scapellato, G. La Spina, G. Pernorio, A. Menciassi, and P. Dario. Biomimetic design of a polychaete robot using ipmc actuator. *Biomedical Robotics and Biomechatronics, 2006. BioRob 2006. The First IEEE/RAS-EMBS International Conference on*, pages 666–671, 0-0 0.

[54] M. Shahinpoor, T. Xue, and J. O. Simpson. Ionic polymer-metal composites (ipmc) as biomimetic sensors and actuators, April 19 2001.

[55] Shuxiang Guo, T. Fukuda, and K. Asaka. A new type of fish-like underwater microrobot. *Mechatronics, IEEE/ASME Transactions on*, 8(1):136–141, March 2003.

[56] Dominiek Reynaerts and Hendrik Van Brussel. Design aspects of shape memory actuators. *Mechatronics*, 8(6):635–656, August 1998.

[57] Anaheim Automation. Stepper sealed high torque motor - 42n65 series www.anaheimautomation.com/stepper-sealed-high-torque-motor-42N65.aspx.

[58] Wikipedia. Power-to-weight ratio en.wikipedia.org/wiki/Power-to-weight_ratio, Oct 2008.

[59] K. Ikuta. Micro/miniature shape memory alloy actuator. *Robotics and Automation, 1990. Proceedings., 1990 IEEE International Conference on*, pages 2156–2161 vol.3, May 1990.

[60] Y. Bar-Cohen, T. Xue, M. Shahinpoor, J. O. Simpson, and J. Smith. Low-mass muscle actuators using electroactive polymers (eap). *Smart materials and technologies; Proceedings of the Meeting, San Diego, CA; UNITED STATES; 4-5 Mar.*, 3324:218–223, July 1998.

[61] John D. Madden, Ryan A. Cush, Tanya S. Kanigan, and Ian W. Hunter. Fast contracting polypyrrole actuators. *Synthetic Metals*, 113(1-2):185–192, June 2000.

[62] John D. Madden, Peter G. Madden, and Ian W. Hunter. Polypyrrole actuators: modeling and performance. *Smart Structures and Materials 2001: Electroactive Polymer Actuators and Devices*, 4329(1):72–83, 2001.

[63] Ledex. Datasheet ledex low profile 8ecm www.ledex.com/ltr2/access.php?file=pdf/LowProfile_8ECM.pdf.

[64] Ledex. Datasheet ledex pull tubular 13x14mm www.ledex.com/ltr2/access.php?file=pdf/Tubular_13x14_Pull.pdf.

[65] RDP Group. D5w submersible lvdt displacement transducer www.rdpe.com/ex/d5w.htm.

[66] RDP Group. How it works - lvdt www.rdpe.com/displacement/lvdt/lvdt-principles.htm.

[67] Interlink Electronics. Force sensing resistor www.interlinkelectronics.com/force_sensors/technologies/fsr.html.

[68] Atmel. Datasheet atmel avr atmega32, www.atmel.com/dyn/resources/prod_documents/doc2503.pdf.

[69] Keithley Instruments Inc. Model kusb-3100 user's manual www.keithley.com/data?asset=50306.

[70] Microsoft. Microsoft robotics studio user guide, msdn.microsoft.com/en-us/library/bb483024.aspx.

[71] D. Grant and V. Hayward. Constrained force control of shape memory alloy actuators. *Robotics and Automation, 2000. Proceedings. ICRA '00. IEEE International Conference on*, 2:1314–1320 vol.2, 2000.

[72] G. Song, V. Chaudhry, and C. Batur. Precision tracking control of shape memory alloy actuators using neural networks and a sliding-mode based robust controller. *Smart Material Structures*, 12: 223–231, April 2003.

[73] N. Ma and G. Song. Control of shape memory alloy actuator using pulse width modulation. *Smart Material Structures*, 12:712–719, October 2003.

[74] Yee Harn Teh and Roy Featherstone. An architecture for fast and accurate control of shape memory alloy actuators. *The International Journal of Robotics Research*, 27(5):595–611, 2008.

[75] Infineon. Datasheet bss98 n-channel sipmos small-signal transistor www.infineon.convergy.de/upload/documents/techdoc/GF_52/bss98.pdf.

[76] International Rectifier. Datasheet irf95420n single p-channel hexfet power mosfet www.irf.com/product-info/datasheets/data/irf9540n.pdf.

[77] Keithley Instruments Inc. Dataacq sdk user's manual www.keithley.com/data?asset=50242.

[78] Maxim IC. Datasheet for max202 datasheets.maxim-ic.com/en/ds/MAX200-MAX213.pdf.

[79] robotikhardware.de. Datasheet www.shop.robotikhardware.de/shop/catalog/product_info.php?cPath=65&products_id=112.

[80] STMicroelectronics. Datasheet fully integrated h-bridge motordriver www.st.com/stonline/products/literature/ds/10832/vnh2sp30-e.pdf.

[81] STMicroelectronics. Datasheet for l6205 motor driver www.st.com/stonline/products/literature/ds/7616/l6205.htm.

Appendix A

Code attachment

A.1 Software for Test Frame Control and Measurement

The below code is the implementation of the program described in section 4.3.1. An evaluation of the program was given in section 5.2.1.

Listing A.1: *Program Loader (Program.cs)*

```
ï≫¿using System;
using System.Collections.Generic;
using System.Linq;
using System.Windows.Forms;

namespace flexinolRiggSansRS {
    static class Program {
        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        [STAThread]
        static void Main() {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new FlexinolRigg());
        }
    }
}
```

Listing A.1: *Program Loader (Program.cs)*

Listing A.2: *Main program (FlexinolRigg.cs)*

```
ï≫¿using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.Diagnostics;

namespace flexinolRiggSansRS {
    public partial class FlexinolRigg : Form {
        public string version = "FlexinolRigg v1.0";

        private double[] lastAin = { 0, 0, 0, 0, 0, 0, 0, 0 };
        private bool[] lastDout = { false, false, false, false, false, false, false, false };
        private double[] nullAin = { 0, 0, 0, 0, 0, 0, 0, 0 };

        private ngraph[] graphs = new ngraph[5];
        private Color[] linecolors = new Color[]{
            Color.Red,
            Color.Blue,
            Color.Green,
            Color.Black,
            Color.Purple,
            Color.Yellow,
            Color.Brown,
            Color.Magenta,
            Color.Cyan,
            Color.DarkGray,
            Color.Orange,
            Color.LightGray};

        private setup _setup;
        private DataGridView[] dg;
        private GroupBox[] gb;
        private Button[] btnZero;
        private dataWriter writer;

        private Kusb3100 kusb;
        private bool doMeasurementLoop = false;
```

```
41          private bool doReloadDoutProg = false;
42          private bool[] reloadIndex = new bool[6];
43
44          public FlexinolRigg() {
45              InitializeComponent();
46
47              _setup = new setup("c:\\ServiceSetup\\flexinolFrame.txt");
48          }
49
50          private void FlexinolRigg_Load(object sender, EventArgs e) {
51              this.Text = version;
52              dg = new DataGridView[]{dg1, dg2, dg3, dg4, dg5};
53              gb = new GroupBox[]{grpWire1, grpWire2, grpWire3, grpWire4, grpWire5};
54              btnZero = new Button[] { btnZero1, btnZero2, btnZero3, btnZero4, btnZero5 };
55
56              this.FormClosing += new FormClosingEventHandler(FlexinolRigg_FormClosing);
57              try {
58                  kusb = new Kusb3100();
59              } catch (Exception ex) {
60                  System.Windows.Forms.MessageBox.Show("Connection to Keithley KUSB-3100 failed: " + ex.Message ,
                        "Error", System.Windows.Forms.MessageBoxButtons.OK);
61                  this.Close();
62              }
63
64              int color_cnt = 0;
65
66              int w = panGraph.Width / 3;
67              int h = panGraph.Height / 2;
68
69              string[] prefix = new string[5];
70
71              for (int i = 0; i < 5; i++) {
72                  dg[i].RowCount = _setup.getInteger("wire" + i + "cnt");
73                  gb[i].Text = _setup.getString("wire" + i + "name");
74                  prefix[i] = _setup.getString("wire" + i + ".prefix") + "_";
75                  string[] names = new string[dg[i].RowCount];
76                  double[] values = new double[dg[i].RowCount];
77
78                  for(int j = 0; j < dg[i].RowCount; j++){
79                      dg[i][0, j].Value = _setup.getString("wire" + i + "." + j);
80                      dg[i][1, j].Value = "";
81                      names[j] = _setup.getString("wire" + i + "." + j);
82                      color_cnt++;
83                      values[j] = 0.0;
84                  }
85                  btnZero[i].Click += btnZero_Click;
86
87                  graphs[i] = new ngraph(
88                                  names,
89                                  _setup.getInteger("xmax"),
90                                  _setup.getString("wire" + i + "name") + " ([cm] / [N] / [V])",
91                                  _setup.getString("dump_folder") + i + ".png");
92
93
94                  if (i < 3) {
95                      panGraph.Controls.Add(graphs[i]);
96
97                      graphs[i].Location = new Point(w * i, 0);
98                  }
99                  else {
100                     panGraph.Controls.Add(graphs[i]);
101                     graphs[i].Location = new Point(w * (i-3), h);
102                 }
103
104                 graphs[i].Size = new Size(w, h);
105                 //graphs[i].addValues(values, DateTime.Now);
106             }
107
108             writer = new dataWriter(prefix, _setup.getInteger("filintervall"), _setup.getString("datafolder"));
109
110             lstStatus.Location = new Point(panGraph.Left + w * 2 + 35, panGraph.Top + h + 37);
111             lstStatus.Height = graphs[0].Height - 59;
112             lstStatus.Width = graphs[0].Width - 49;
113             lstStatus.BorderStyle = BorderStyle.FixedSingle;
114
115
116             setStatus("FlexinolRiggSansRS loaded");
117
118         }
119
120         void FlexinolRigg_FormClosing(object sender, FormClosingEventArgs e) {
121             if (doMeasurementLoop) e.Cancel = true;
122             Debug.WriteLine("FormClosing (" + e.Cancel.ToString() + ")");
123             if (e.Cancel) setStatus("Stop measurement before closing the form");
124         }
125
126         //delegate void SetStatusCallback(string status);
127         private void setStatus(string status) {
128             /*if (lstStatus.InvokeRequired) {
129                 SetStatusCallback d = new SetStatusCallback(setStatus);
130                 lstStatus.Invoke(d, new object[] { status });
131             } else {
132             */
133                 status = DateTime.Now + " - " + status;
134                 lstStatus.Items.Insert(0, status);
135
136                 while (lstStatus.Items.Count > 50) {
137                     lstStatus.Items.RemoveAt(lstStatus.Items.Count - 1);
138                 }
139             //}
140         }
141
142         private void btnZero_Click(object sender, EventArgs e) {
143             int i = Array.IndexOf(btnZero, (Button)sender);
144
145             int chnum;
146             string ch;
147
148             for (int j = 0; j < _setup.getInteger("wire" + i + "cnt"); j++) {
149                 ch = _setup.getString(_setup.getString("wire" + i + "." + j + "_ch") + "_ch");
150                 if (ch.StartsWith("Ain")) {
151                     chnum = int.Parse(ch.Substring(3, 1));
152                     nullAin[chnum] = lastAin[chnum];
```

```
153                     }
154                 }
155             }
156
157             private void setValue(string[] v, int i) {
158                 for (int j = 0; j < dg[i].RowCount; j++) dg[i][1, j].Value = v[j];
159             }
160
161             private void startToolStripMenuItem_Click(object sender, EventArgs e) {
162                 doMeasurement();
163             }
164
165             private void stopToolStripMenuItem_Click(object sender, EventArgs e) {
166                 stopToolStripMenuItem.Enabled = false;
167                 doMeasurementLoop = false;
168             }
169
170             private void exitProgramToolStripMenuItem_Click(object sender, EventArgs e) {
171                 this.Close();
172             }
173
174
175             private void doMeasurement() {
176                 for (int i = 0; i < 6; i++) reloadIndex[i] = false;
177
178                 setStatus("Measurement started");
179
180                 startToolStripMenuItem.Enabled = false;
181                 stopToolStripMenuItem.Enabled = true;
182                 exitProgramToolStripMenuItem.Enabled = false;
183                 reloadDoutValuesToolStripMenuItem.Enabled = true;
184
185                 reloadDoutValuesForWire1ToolStripMenuItem.Enabled = true;
186                 reloadDoutValuesForWire2ToolStripMenuItem.Enabled = true;
187                 reloadDoutValuesForWire3ToolStripMenuItem.Enabled = true;
188                 reloadDoutValuesForWire4ToolStripMenuItem.Enabled = true;
189                 reloadDoutValuesForWire5ToolStripMenuItem.Enabled = true;
190
191                 doMeasurementLoop = true;
192                 int mean_ms = _setup.getInteger("mean_ms");
193                 int periode_ms = _setup.getInteger("periode_ms");
194
195
196                 setup doutsetup = new setup("c:\\ServiceSetup\\flexinolFrameDoutProg.txt");
197                 string[] progs = new string[6];
198
199                 for (int i = 0; i < 6; i++) progs[i] = doutsetup.getString("wire_" + i);
200                 kusb.setDoutProgs(progs);
201
202                 double flexinol_vdd = _setup.getDouble("flexinol_vdd");
203                 double[] analogIn = {0,0,0,0,0,0,0,0};
204
205                 int wirecnt = _setup.getInteger("wirecnt");
206                 int[] lengths = new int[wirecnt];
207                 int max = 0;
208                 for (int i = 0; i < lengths.Length; i++) {
209                     lengths[i] = _setup.getInteger("wire" + i + "cnt");
210                     max = Math.Max(lengths[i], max);
211                 }
212
213                 double[][] values = new double[wirecnt][];
214                 string[][] v = new string[wirecnt][];
215                 double[][] a = new double[wirecnt][];
216                 double[][] b = new double[wirecnt][];
217                 string[][] ch = new string[wirecnt][];
218                 int[][] chnum = new int[wirecnt][];
219                 bool[][] isAin = new bool[wirecnt][];
220
221                 for (int i = 0; i < v.Length; i++) {
222                     v[i] = new string[max];
223                     values[i] = new double[max];
224                     a[i] = new double[max];
225                     b[i] = new double[max];
226                     ch[i] = new string[max];
227                     chnum[i] = new int[max];
228                     isAin[i] = new bool[max];
229                 }
230
231                 for (int i = 0; i < a.Length; i++) {
232                     for (int j = 0; j < lengths[i]; j++) {
233                         ch[i][j] = _setup.getString(_setup.getString("wire" + i + "." + j + "_ch") + "_ch");
234
235                         if (ch[i][j].StartsWith("Ain")) {
236                             chnum[i][j] = int.Parse(ch[i][j].Substring(3, 1));
237                             isAin[i][j] = true;
238                             a[i][j] = _setup.getDouble(_setup.getString("wire" + i + "." + j + "_ch") + "_A");
239                             b[i][j] = _setup.getDouble(_setup.getString("wire" + i + "." + j + "_ch") + "_B");
240                         } else if (ch[i][j].StartsWith("Dout")) {
241                             chnum[i][j] = int.Parse(ch[i][j].Substring(4, 1));
242                             isAin[i][j] = false;
243                         } else {
244                             throw new Exception("Feil i oppsettfil");
245                         }
246                     }
247                 }
248
249                 kusb.setMeanMS(mean_ms);
250
251                 kusb.startDout(doutsetup.getInteger("dout_tick"));
252                 DateTime loopTimer = DateTime.Now;
253
254                 double value;
255                 while (doMeasurementLoop) {
256
257                     if (doReloadDoutProg) {
258                         doutsetup.readFile();
259                         for (int i = 0; i < 6; i++) {
260                             if (reloadIndex[i]) {
261                                 progs[i] = doutsetup.getString("wire_" + i);
262                                 kusb.setDoutProg(progs[i], i);
263                             }
264                         }
265
```

```
266                     setStatus("Digital output values updated");
267                     doReloadDoutProg = false;
268                     for (int i = 0; i < 6; i++) reloadIndex[i] = false;
269                 }
270 
271                 analogIn = kusb.readAnalog();
272                 for (int i = 0; i < analogIn.Length; i++) lastAin[i] = analogIn[i];
273                 lastDout = kusb.lastDout;
274 
275 
276 
277                 for (int i = 0; i <= dg.GetUpperBound(0); i++) {
278                     for (int j = 0; j < lengths[i]; j++) {
279 
280                         if (isAin[i][j]) {
281                             value = analogIn[chnum[i][j]] * a[i][j] + b[i][j];
282                             v[i][j] = value.ToString(" 0.000;-0.000; 0.000"); ;
283                             values[i][j] = value;
284 
285                         }else{
286                             if (lastDout[chnum[i][j]]) {
287                                 v[i][j] = flexinol_vdd.ToString();
288                                 value = flexinol_vdd;
289                             } else {
290                                 v[i][j] = "0";
291                                 value = 0;
292                             }
293                             values[i][j] = value;
294                         }
295                     }
296                     //setValue((string[])v.Clone(), i);
297                     setValue(v[i], i);
298 
299                     try {
300                         graphs[i].addValues(values[i], DateTime.Now);
301                     } catch (Exception ex) {
302                         setStatus("Problems plotting: " + ex.Message);
303                     }
304 
305                     try {
306                         writer.addValues(values[i], DateTime.Now, i,lengths[i]);
307                     } catch (Exception ex) {
308                         setStatus("Problem saving: " + ex.Message);
309                     }
310 
311                 }
312 
313                 //Loop timer
314 
315                 while (DateTime.Now.Subtract(loopTimer).TotalMilliseconds < periode_ms) {
316                     Application.DoEvents();
317                 }
318                 Debug.WriteLine("Loop time: " + DateTime.Now.Subtract(loopTimer).TotalMilliseconds.ToString() +
                        "ms");
319                 loopTimer = DateTime.Now;
320             }
321             kusb.stopDout();
322 
323 
324 
325             startToolStripMenuItem.Enabled = true;
326             exitProgramToolStripMenuItem.Enabled = true;
327             reloadDoutValuesToolStripMenuItem.Enabled = false;
328 
329             reloadDoutValuesForWire1ToolStripMenuItem.Enabled = false;
330             reloadDoutValuesForWire2ToolStripMenuItem.Enabled = false;
331             reloadDoutValuesForWire3ToolStripMenuItem.Enabled = false;
332             reloadDoutValuesForWire4ToolStripMenuItem.Enabled = false;
333             reloadDoutValuesForWire5ToolStripMenuItem.Enabled = false;
334 
335             setStatus("Measurement ended");
336         }
337 
338         private void reloadDoutValuesToolStripMenuItem_Click(object sender, EventArgs e) {
339             doReloadDoutProg = true;
340             for (int i = 0; i < 5; i++) reloadIndex[i] = true;
341         }
342 
343         private void reloadDoutValuesForWire1ToolStripMenuItem_Click(object sender, EventArgs e) {
344             doReloadDoutProg = true;
345             reloadIndex[0] = true;
346         }
347 
348         private void reloadDoutValuesForWire2ToolStripMenuItem_Click(object sender, EventArgs e) {
349             doReloadDoutProg = true;
350             reloadIndex[1] = true;
351 
352         }
353 
354         private void reloadDoutValuesForWire3ToolStripMenuItem_Click(object sender, EventArgs e) {
355             doReloadDoutProg = true;
356             reloadIndex[2] = true;
357 
358         }
359 
360         private void reloadDoutValuesForWire4ToolStripMenuItem_Click(object sender, EventArgs e) {
361             doReloadDoutProg = true;
362             reloadIndex[3] = true;
363             reloadIndex[4] = true;
364 
365         }
366 
367         private void reloadDoutValuesForWire5ToolStripMenuItem_Click(object sender, EventArgs e) {
368             doReloadDoutProg = true;
369             reloadIndex[5] = true;
370 
371         }
372 
373         private void timGC_Tick(object sender, EventArgs e) {
374             System.GC.Collect();
375         }
376 
377 
```

```
378
379
380         }
381     }
```

Listing A.2: *Main program (FlexinolRigg.cs)*

Listing A.3: *Class for storing data (dataWriter.cs)*

```
1   ï»¿using System;
2   using System.Collections.Generic;
3   using System.Text;
4   using System.IO;
5
6   namespace flexinolRiggSansRS {
7       class dataWriter {
8           int interval;
9
10          string[] filePrefix;
11          string[] filenames;
12          string savetofolder;
13          DateTime filenameDate;
14
15
16          public dataWriter(string[] filePrefix, int newfileIntervalDays, string savetofolder) {
17              interval = newfileIntervalDays;
18              this.filePrefix = filePrefix;
19              this.savetofolder = savetofolder;
20
21              filenames = getFilenames(filePrefix);
22          }
23
24
25          public void addValues(double[] values, DateTime d, int filenr, int len){
26              if (DateTime.Now.Subtract(filenameDate).TotalDays >= interval) filenames = getFilenames(filePrefix)
                    ;
27
28              FileStream fs = null;
29              StreamWriter sw;
30
31              if (File.Exists(filenames[filenr])){
32                  sw = File.AppendText(filenames[filenr]);
33              }else{
34                  fs = File.Create(filenames[filenr]);
35                  sw = new StreamWriter(fs);
36              }
37
38
39              sw.Write(d.ToString("dd.MM.yyyy HH:mm:ss.f") + "\t");
40              for (int i = 0; i < len; i++) {
41                  sw.Write(string.Format("{0:0.000}", values[i]) + "\t");
42              }
43              sw.Write("\n");
44
45              sw.Close();
46              if(fs != null) fs.Close();
47          }
48
49
50          private string[] getFilenames(string[] prefix) {
51              string[] ret = new string[prefix.Length];
52              for (int i = 0; i < prefix.Length; i++) {
53                  ret[i] = getFilename(prefix[i]);
54              }
55
56              return ret;
57          }
58
59          private string getFilename(string prefix) {
60              string filename = savetofolder + "\\" + prefix + DateTime.Now.ToString("yyyy.MM.dd");
61              if (!File.Exists(filename + ".txt")) return filename + ".txt";
62              int cnt = 0;
63              filenameDate = DateTime.Now;
64
65              while (true) {
66                  string filename2 = filename + "_" + cnt + ".txt";
67                  if(!File.Exists(filename2)) return filename2;
68                  cnt++;
69              }
70
71
72
73          }
74      }
75  }
```

Listing A.3: *Class for storing data (dataWriter.cs)*

Listing A.4: *Class for control of digital output signals (doutProgram.cs)*

```
1   ï»¿using System;
2   using System.Collections.Generic;
3   using System.Collections;
4   using System.Linq;
5   using System.Text;
6
7   namespace flexinolRiggSansRS {
8       class doutProgram {
9
10          bool[] prog;
```

```
        int prog_i = 0;

        public doutProgram(string prog) {
            string[] prog_el = prog.Split('%')[0].Trim().Split(';');

            ArrayList tmp = new ArrayList();

            for (int i = 0; i < prog_el.Length; i++) {
                System.Diagnostics.Debug.WriteLine(prog_el[i]);
                string[] prog_el2 = prog_el[i].Split('*');
                bool v = bool.Parse(prog_el2[1]);
                for (int j = 0; j < int.Parse(prog_el2[0]); j++) {
                    tmp.Add(v);
                }
            }

            this.prog = (bool[])tmp.ToArray(typeof(bool));
        }

        public bool nextValue() {
            bool ret = prog[prog_i++];
            if (prog_i > prog.GetUpperBound(0)) prog_i = 0;

            return ret;
        }
    }
}
```

Listing A.4: *Class for control of digital output signals (doutProgram.cs)*

Listing A.5: *Class for communication with DAQ-module KUSB3100 (Kusb3100.cs)*

```
ï≫¿using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

using OpenLayers.Base;

namespace flexinolRiggSansRS {
    class Kusb3100 {

        private DeviceMgr deviceMgr;
        private Device device;

        private AnalogInputSubsystem ainSS;
        private AnalogOutputSubsystem aoutSS;

        private DigitalInputSubsystem dinSS;
        private DigitalOutputSubsystem doutSS;

        private int buf_len;
        private OlBuffer daqBuffer;

        private double[] ain = {0,0,0,0,0,0,0,0};
        private bool ain_done;

        public bool[] lastDout = { false, false, false, false, false, false, false, false };
        public bool[] dout = { false, false, false, false, false, false, false, false };
        double[] buf;

        private int mean_ms = 100;
        private int ainClk = 10000;

        private System.Windows.Forms.Timer timer = new System.Windows.Forms.Timer();
        private doutProgram[] doutProgs;

        public Kusb3100() {
            deviceMgr = DeviceMgr.Get();

            // Connects to first device. Multi device support should be added later
            device = deviceMgr.GetDevice(deviceMgr.GetDeviceNames()[0]);
            ainSS = device.AnalogInputSubsystem(0);
            // Create event handlers
            //ainSS.DriverRunTimeErrorEvent += new DriverRunTimeErrorEventHandler(HandleDriverRunTimeErrorEvent
                );
            ainSS.BufferDoneEvent += new BufferDoneHandler(HandleBufferDone);
            //ainSS.QueueDoneEvent += new QueueDoneHandler(HandleQueueDone);
            //ainSS.QueueStoppedEvent += new QueueStoppedHandler(HandleQueueDone);

            // Configuring analog output system
            aoutSS = device.AnalogOutputSubsystem(0);
            aoutSS.DataFlow = DataFlow.SingleValue;
            aoutSS.Config();

            // Configuring digital input system
            dinSS = device.DigitalInputSubsystem(0);
            dinSS.DataFlow = DataFlow.SingleValue;
            dinSS.Config();

            // Configuring digital output system
            doutSS = device.DigitalOutputSubsystem(0);
            doutSS.DataFlow = DataFlow.SingleValue;
            doutSS.Config();



            timer.Tick += new EventHandler(timer_Tick);
        }

        public void setDoutProgs(string[] progs) {
            doutProgs = new doutProgram[progs.Length];
            for (int i = 0; i < progs.Length; i++) {
                doutProgs[i] = new doutProgram(progs[i]);
            }
        }

```

```
75          public void setDoutProg(string prog, int i) {
76              doutProgs[i] = new doutProgram(prog);
77          }
78
79          public void startDout(int tick_ms) {
80              timer.Interval = tick_ms;
81              timer.Enabled = true;
82          }
83
84          public void stopDout() {
85              timer.Enabled = false;
86
87              for (int i = 0; i < dout.Length; i++) dout[i] = false;
88
89              writeDigital(dout);
90              lastDout = dout;
91          }
92
93          void timer_Tick(object sender, EventArgs e) {
94              for (int i = 0; i < dout.Length; i++) dout[i] = false;
95
96              for (int i = 0; i < doutProgs.Length; i++) {
97                  dout[i] = doutProgs[i].nextValue();
98              }
99
100             writeDigital(dout);
101
102             lastDout = dout;
103         }
104
105         public void setMeanMS(int mean_ms){
106             this.mean_ms = mean_ms;
107             buf_len = (int)(mean_ms * ainClk) / (1000 * 8);
108             if (daqBuffer != null) {
109                 daqBuffer.Dispose();
110                 daqBuffer = null;
111             }
112             daqBuffer = new OlBuffer(buf_len * 8, ainSS);
113             ainSS.ChannelList.Clear();
114             for (int i = 0; i < 8; i++) ainSS.ChannelList.Insert(i, i);
115             ainSS.DataFlow = DataFlow.Continuous;
116             ainSS.Clock.Frequency = ainClk;
117             ainSS.Config();
118
119         }
120
121         public double[] readAnalog() {
122
123
124
125             ainSS.BufferQueue.FreeAllQueuedBuffers();
126
127             //daqBuffer.Reallocate(buf_len * 8);
128             //daqBuffer = new OlBuffer(buf_len * 8, ainSS);
129
130             ainSS.BufferQueue.QueueBuffer(daqBuffer);
131
132
133             ain_done = false;
134             //ainSS.Config();
135             ainSS.Start();
136
137             while (!ain_done) {
138                 System.Windows.Forms.Application.DoEvents();
139             }
140
141             return ain;
142         }
143
144         /*
145         public void writeAnalog(double[] outputValues) {
146             aoutSS.SetSingleValueAsVolts(0, outputValues[0]);
147             aoutSS.SetSingleValueAsVolts(1, outputValues[1]);
148
149         }
150         */
151         /*
152         public bool[] readDigital() {
153
154             int din = dinSS.GetSingleValue();
155             bool[] tmp = new bool[8];
156             int t = 1;
157
158             for (int i = 0; i < 8; i++) {
159                 tmp[i] = (((t << i) & din) != 0);
160             }
161
162             return tmp;
163         }
164         */
165         public void writeDigital(bool[] outputValues) {
166             int tmp = 0x00;
167
168             for (int i = 0; i < 8; i++) {
169                 if (outputValues[i]) {
170                     tmp += (int)Math.Pow(2, i);
171                 }
172             }
173
174             doutSS.SetSingleValue(tmp);
175
176         }
177
178
179         public void HandleBufferDone(object sender, BufferDoneEventArgs bufferDoneData) {
180             OlBuffer olBuffer = bufferDoneData.OlBuffer;
181
182             // Get the data as voltages
183             buf = olBuffer.GetDataAsVolts();
184             //olBuffer.Dispose();
185             //Analog input system does not support simultaneous sample and hold
186             for (int i = 0; i < ain.Length; i++) ain[i] = 0;
187
```

```
188                 for (int n = 0; n < buf_len * 8; n++) {
189                     ain[n % 8] += buf[n];
190                 }
191 
192                 for (int i = 0; i < 8; i++) {
193                     ain[i] = ain[i] / buf_len;
194                 }
195 
196                 while (ainSS.State == SubsystemBase.States.Running) ;
197                 //ainSS.Dispose();
198                 ain_done = true;
199             }
200         }
201 }
```

Listing A.5: *Class for communication with DAQ-module KUSB3100 (Kusb3100.cs)*

Listing A.6: *Wrapper class for NPlot plotting library (ngraph.cs)*

```
1  using System;
2  using System.Collections.Generic;
3  using System.Text;
4  using System.Drawing;
5  using System.Windows.Forms;
6  using System.Diagnostics;
7  using NPlot;
8  
9  namespace flexinolRiggSansRS {
10     class ngraph : Panel{
11         public NPlot.Windows.PlotSurface2D xygraph = new NPlot.Windows.PlotSurface2D();
12         private LinePlot[] plots;
13         List<DateTime> xvalues;
14         List<double>[] yvalues;
15 
16         private int xlength;
17         private string title;
18 
19         private string dump;
20         private DateTime lastDump = DateTime.Now;
21 
22         public int XLength
23         {
24           get { return xlength; }
25           set { xlength = value; }
26         }
27 
28         public Color[] linecolors = {
29             Color.Red,
30             Color.Blue,
31             Color.Lime,
32             Color.Black,
33             Color.Purple,
34             Color.Yellow,
35             Color.Cyan,
36             Color.Gray,
37             Color.Brown
38         };
39 
40         public ngraph(string[] names, int xlength, string title, string dump) {
41             Controls.Add(xygraph);
42             this.title = title;
43             xygraph.Location = new Point(0, 0);
44             xygraph.Size = this.Size;
45             XLength = xlength;
46             this.Resize += new EventHandler(ngraph_Resize);
47             this.dump = dump;
48 
49             initGraph(names);
50         }
51 
52         void ngraph_Resize(object sender, EventArgs e) {
53             xygraph.Size = this.Size;
54         }
55 
56         private void initGraph(string[] names){
57             int plotcnt = names.Length;
58             xygraph.Clear();
59 
60             Font myFont = new Font("Arial", 8, FontStyle.Bold);
61 
62             //Add a background grid for better chart readability.
63             Grid grid = new Grid();
64             grid.VerticalGridType = Grid.GridType.Coarse;
65             grid.HorizontalGridType = Grid.GridType.Coarse;
66             grid.MajorGridPen = new Pen(Color.LightGray, 1.0f);
67             xygraph.Add(grid);
68 
69             xygraph.Title = title;
70             xygraph.TitleColor = Color.Red;
71             xygraph.Capture = false;
72             xygraph.CausesValidation = false;
73 
74             xvalues = new List<DateTime>();
75             yvalues = new List<double>[plotcnt];
76             plots = new LinePlot[plotcnt];
77 
78             for (int i = 0; i < plotcnt; i++) {
79                 yvalues[i] = new List<double>();
80                 plots[i] = new LinePlot();
81                 plots[i].Color = linecolors[i % linecolors.GetUpperBound(0)];
82                 plots[i].AbscissaData = xvalues;
83                 plots[i].DataSource = yvalues[i];
84                 plots[i].Label = names[i];
85                 xygraph.Add(plots[i]);
86             }
87 
88             //Balance plot general settings.
```

```
89              xygraph.ShowCoordinates = true;
90              xygraph.YAxis1.Label = "Deformasjon [cm] / Kraft [N]";
91              xygraph.YAxis1.LabelOffsetAbsolute = true;
92              xygraph.YAxis1.LabelOffset = 30;
93              //xygraph.YAxis1.WorldMin = 0;
94              //xygraph.YAxis1.WorldMax = 50;
95
96              xygraph.XAxis1.Label = "Tid [sek]";
97              xygraph.Padding = 15;
98
99              //Refresh surfaces.
100             xygraph.Refresh();
101
102             Legend legend = new Legend();
103             legend.AttachTo(PlotSurface2D.XAxisPosition.Top, PlotSurface2D.YAxisPosition.Left);
104             legend.VerticalEdgePlacement = Legend.Placement.Inside;
105             legend.HorizontalEdgePlacement = Legend.Placement.Inside;
106             legend.BorderStyle = LegendBase.BorderType.Line;
107             legend.XOffset = 10;
108             legend.YOffset = 10;
109             legend.Font = myFont;
110             xygraph.Legend = legend;
111         }
112
113         /*
114         private void dumpData() {
115             try {
116                 Bitmap b = new Bitmap(xygraph.Width, xygraph.Height);
117                 xygraph.DrawToBitmap(b, new Rectangle(new Point(0, 0), xygraph.Size));
118
119                 b.Save(dump);
120             } catch (Exception ex) {
121                 Debug.WriteLine("Datadump error: " + ex.Message);
122             }
123
124             lastDump = DateTime.Now;
125         }
126         */
127
128         public void addValues(double[] values, DateTime d) {
129
130             while(xvalues.Count > 0){
131                 if (d.Subtract(xvalues[0]).TotalSeconds > XLength) {
132                     xvalues.RemoveAt(0);
133                     for (int j = 0; j < yvalues.Length; j++) {
134                         yvalues[j].RemoveAt(0);
135                     }
136                 } else break;
137             }
138
139             for (int i = 0; i < yvalues.Length; i++) {
140                 xygraph.Remove(plots[i], true);
141             }
142
143             xvalues.Add(d);
144             for (int i = 0; i < yvalues.Length; i++) {
145                 yvalues[i].Add(values[i]);
146                 xygraph.Add(plots[i]);
147             }
148             //xygraph.Refresh();
149             doRefresh();
150
151             //if(DateTime.Now.Subtract(lastDump).TotalMinutes > 1) dumpData();
152         }
153
154
155         delegate void doRefreshCallback();
156         private void doRefresh() {
157             if (xygraph.InvokeRequired) {
158                 doRefreshCallback d = new doRefreshCallback(doRefresh);
159                 xygraph.Invoke(d);
160             }
161             else {
162                 xygraph.Refresh();
163             }
164
165         }
166
167     }
168 }
```

Listing A.6: *Wrapper class for NPlot plotting library (ngraph.cs)*

Listing A.7: *Class for reading setup files (setup.cs)*

```
1   ï≫¿using System;
2   using System.Collections.Generic;
3   using System.Text;
4   using System.IO;
5   using System.Diagnostics;
6
7   namespace flexinolRiggSansRS {
8       public class setup {
9
10          private System.Collections.Hashtable values;
11          private string filename;
12
13          public setup(string filename) {
14              this.filename = filename;
15              readFile();
16          }
17
18          public void readFile() {
19              FileStream fs = File.OpenRead(this.filename);
20              StreamReader sr = new StreamReader(fs);
21
22              values = new System.Collections.Hashtable();
```

```
23
24              string tmp, _name, _value;
25              int _ind;
26              while (!sr.EndOfStream) {
27                  tmp = sr.ReadLine();
28                  try {
29                      _ind = tmp.IndexOf("]=");
30                      _name = tmp.Substring(1, _ind - 1);
31                      _value = tmp.Substring(_ind + 2);
32                      values.Add(_name, _value);
33                  } catch (Exception e) {
34                      Debug.WriteLine("Error in setup file. Exception details: " + e.Message);
35                  }
36              }
37
38              sr.Close();
39              fs.Close();
40          }
41
42          public string getString(string _name){
43              return (string)values[_name];
44          }
45
46          public int getInteger(string _name) {
47              return int.Parse(getString(_name));
48          }
49
50          public double getDouble(string _name) {
51              return double.Parse(getString(_name));
52          }
53      }
54  }
```

Listing A.7: *Class for reading setup files (setup.cs)*

A.2 Software for Web Surveillance of Test Frame

The below code is the implementation of the program described in section 4.3.2.

Listing A.8: *Main webpage (Default.aspx)*

```
1   ï»¿<%@ Page Language="VB" AutoEventWireup="false" CodeFile="Default.aspx.vb" Inherits="_Default"  %>
2
3   <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
        transitional.dtd">
4
5
6   <html>
7   <head>
8           <link rel="stylesheet" type="text/css" href="styles.css">
9           <title>FlexinolRigg datadump</title>
10  </head>
11  <body>
12
13  <%
14      Dim folders(4) As String
15      Dim names(4) As String
16      Dim plotnames(4) As String
17
18      folders(0) = "C:\FlexinolRiggData\fixation\"
19      folders(1) = "C:\FlexinolRiggData\small_load\"
20      folders(2) = "C:\FlexinolRiggData\heavy_load\"
21      folders(3) = "C:\FlexinolRiggData\flexinol_antagonist\"
22      folders(4) = "C:\FlexinolRiggData\spring_antagonist\"
23
24      names(0) = "Fixation test"
25      names(1) = "Test with small load"
26      names(2) = "Test with heavy load"
27      names(3) = "Flexinol as antagonist"
28      names(4) = "Spring as antagonist"
29
30      plotnames(0) = "Force;Output"
31      plotnames(1) = "Deflection;Output"
32      plotnames(2) = "Deflection;Output"
33      plotnames(3) = "Force;Deflection;Output;Output Antagonist"
34      plotnames(4) = "Force;Deflection;Output"
35
36      Dim action As String = Request("action")
37
38      If action = "viewfolder" Then
39          %>
40          <a href="Default.aspx">Back to main menu</a>
41          <%
42              Dim folder As String = Int(Request("folder"))
43              Response.Write("<h2>" & names(folder) & "</h2>")
44
45              Dim fileinfos As System.Collections.ArrayList = doSort(folders(folder))
46
47              Response.Write("<table border=0><tr><td valign='top'>")
48              Response.Write("<table border=0><tr><th width=200 align=left>Name</th><th width=100 align=left>Size
                    </th><th width=150 align=left>Last modified</th></tr>")
49              For Each fileinfo As IO.FileInfo In fileinfos
50                  Dim url As String = "default.aspx?action=viewfile&folder=" & folder & "&file=" & fileinfo.Name
51                  Response.Write("<tr><td><a href='" & url & "'>" _
52                   & fileinfo.Name & "</a></td><td>" _
53                   & Int(fileinfo.Length / 1024) & " KB</td><td>" _
54                  & fileinfo.LastWriteTime & "</td>" _
55                  & "</tr>")
56              Next
57
58              Response.Write("</table>")
```

```
59              Response.Write("</td></tr></table>")
60
61          ElseIf action = "viewfile" Then
62              Dim folder As String = Int(Request("folder"))
63              Dim file As String = folders(folder) & Request("file")
64              Dim start As Integer = 0 : Integer.TryParse(Request("start"), start)
65              Dim length As Integer = 200 : Integer.TryParse(Request("length"), length)
66              Dim skip As Integer = 1 : Integer.TryParse(Request("skip"), skip)
67              Dim showdata As String = Request("showdata")
68              Dim method As String = Request("method")
69
70              If showdata = "on" Then showdata = "checked"
71
72
73              If length = 0 Then length = 200
74              If skip < 1 Then skip = 1
75
76              Dim objFSO As New Scripting.FileSystemObject
77              Dim objText As Scripting.TextStream = objFSO.OpenTextFile(file)
78
79      %>
80      <a href="Default.aspx?action=viewfolder&folder=<%=folder%>">Back to <%=names(Integer.Parse(folder))%>
            folder</a><br /><br />
81      <form action="default.aspx?action=viewfile&folder=<%=folder%>&file=<%=request("file")%>&method=span" method
            ="post">
82      Start at line <input type="text" size="10" name="start" value="<%=start%>"/> and
83      show <input type="text" size="10" name="length" value="<%=length%>" /> lines. Show every <input type="text"
             size="10" name="skip" value="<%=skip%>" /> line.
84      Show data <input name="showdata" type="checkbox" <%=showdata%> />
85      <input type="submit" value="Go!" />
86      </form>
87      <br />
88      <form action="default.aspx?action=viewfile&folder=<%=folder%>&file=<%=request("file")%>&method=last" method
            ="post">
89      Show <input type="text" size="10" name="length" value="<%=length%>" /> last lines.
90      Show data <input name="showdata" type="checkbox" <%=showdata%> />
91      <input type="submit" value="Go!" />
92      </form>
93
94      <%
95          If showdata = "checked" Then
96              Dim cnt As Integer = 0
97              For i As Integer = 1 To start
98                  If objText.AtEndOfStream Then Exit For
99                  For j As Integer = 1 To skip
100                     If objText.AtEndOfStream Then Exit For
101                     objText.SkipLine()
102                 Next
103                 cnt = cnt + 1
104             Next
105
106             Dim text As String = ""
107             For i As Integer = start To start + length
108                 If objText.AtEndOfStream Then Exit For
109                 text = text & objText.ReadLine & vbCrLf
110                 For j As Integer = 1 To skip - 1
111                     objText.SkipLine()
112                 Next
113                 cnt = cnt + 1
114             Next
115
116             While Not objText.AtEndOfStream
117                 For j As Integer = 1 To skip
118                     If objText.AtEndOfStream Then Exit For
119                     objText.SkipLine()
120                 Next
121                 cnt = cnt + 1
122             End While
123
124
125             objText.Close()
126             objText = Nothing
127             objFSO = Nothing
128
129             Response.Write("<h2>" & file & " (" & cnt & " lines)</h2>")
130             Response.Write("<table border='0' width='1200'><tr>")
131
132
133             text = Replace(text, vbTab, "</td><td width=75>")
134             text = Replace(text, vbLf, "</td></tr><tr><td width=75>")
135
136             text = "<table border=0><tr><td width=150>" & text & "</td></tr></table>"
137             Response.Write("<td valign='top'>" & text & "</td>")
138         Else
139             Response.Write("<h2>" & file & "</h2>")
140             Response.Write("<table border='0' width='1200'><tr>")
141         End If
142
143
144         Response.Write("<td valign='top'>")
145
146         Response.Write("<img src='plot.aspx?")
147         If method = "span" Then Response.Write("start=" & start)
148         Response.Write("&length=" & length)
149         Response.Write("&names=" & plotnames(folder))
150         Response.Write("&file=" & file)
151         Response.Write("&title=" & names(folder))
152         Response.Write("&method=" & method)
153         Response.Write("' alt='' >")
154
155     %>
156     </td></tr>
157     </table>
158
159
160     <%
161     ElseIf action = "viewlast" Then
162         Dim length As Integer = 200 : Integer.TryParse(Request("length"), length)
163         If length = 0 Then length = 1000
164         %>
165         <a href="default.aspx">Back to main menu</a><br /><br />
166         <form action="default.aspx?action=viewlast" method="post">
167             Show <input type="text" size="10" name="length" value="<%=length%>" /> last lines.
```

```
            <input type="submit" value="Go!" />
        </form>
        <%
            Dim fileinfos As System.Collections.ArrayList
            For i As Integer = 0 To folders.GetUpperBound(0)
                fileinfos = doSort(folders(i))

                Dim file As String = folders(i) & DirectCast(fileinfos(0), IO.FileInfo).Name

                Response.Write("<h2>" & names(i) & "</h2>")
                Response.Write("<img src='plot.aspx?")
                Response.Write("&length=" & length)
                Response.Write("&names=" & plotnames(i))
                Response.Write("&file=" & file)
                Response.Write("&title=" & names(i))
                Response.Write("&method=last")
                Response.Write("' alt='' ><br><br>")


            Next

    Else
        Response.Write("<h2>FlexinolRigg</h2>")
        Response.Write("<table width='400' border='0'><tr><td>")
        Response.Write("<ul>")
        For i As Integer = 0 To UBound(folders)
            Response.Write("<li><a href='default.aspx?action=viewfolder&folder=" & i & "'>" & names(i) & "</a
                ></li>")
        Next
        Response.Write("</ul>")
        Response.Write("</td><td valign='top'><ul><li><a href='default.aspx?action=viewlast'>View last data</a
            ></li></ul></td></tr></table>")


        'Siste bilde fra webcam
        Response.Write("<img src='http://oyvindsroboticstudio.dyndns.org:8080/' alt=''>")
    End If



    %>

</body>
</html>
```

Listing A.8: *Main webpage (Default.aspx)*

Listing A.9: *Code for main webpage (Default.aspx.vb)*

```
ï»¿Imports System.Collections
Imports System.Collections.Generic
Imports System.IO

Partial Class _Default
    Inherits System.Web.UI.Page


    Private Class FileInfoCreationComparer
        Implements IComparer

        Public Function Compare(ByVal x As Object, ByVal y As Object) As Integer Implements System.Collections.
            IComparer.Compare
            Dim xFile As FileInfo = CType(x, FileInfo)
            Dim yFile As FileInfo = CType(y, FileInfo)
            Dim time As TimeSpan = yFile.LastWriteTime.Subtract(xFile.LastWriteTime)
            Return time.TotalMinutes
        End Function
    End Class

    Public Function doSort(ByVal path As String) As ArrayList
        Dim dirInfo As New DirectoryInfo(path)
        Dim FileNo As Integer = dirInfo.GetFiles("*.*").Length
        If FileNo > 0 Then
            ''Changed Code'''
            Dim files() As FileInfo = dirInfo.GetFiles() ''No need for filter "*.*" if you are selecting
                everything
            Dim list As New ArrayList(files) ''Add the files array to an Array(list)
            list.Sort(New FileInfoCreationComparer())
            Return list
        Else
            Return Nothing
        End If
    End Function

End Class
```

Listing A.9: *Code for main webpage (Default.aspx.vb)*

Listing A.10: *Plot generator (plot.aspx)*

```
ï»¿<%@ Page Language="VB" AutoEventWireup="false" CodeFile="plot.aspx.vb" Inherits="plot" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
    <HEAD>
        <title>Demo1</title>
        <meta name="GENERATOR" content="Microsoft Visual Studio .NET 7.1">
        <meta name="CODE_LANGUAGE" content="Visual Basic .NET 7.1">
        <meta name="vs_defaultClientScript" content="JavaScript">
        <meta name="vs_targetSchema" content="http://schemas.microsoft.com/intellisense/ie5">
```

```
        </HEAD>
        <body>

                <form id="Form2" method="post" runat="server">

                </form>

        </body>
</HTML>
```

Listing A.10: *Plot generator (plot.aspx)*

Listing A.11: *Code for plot generator (plot.aspx.vb)*

```
ï»¿Imports System.Collections.Generic
Imports System.Drawing
Imports NPlot
Imports System.IO


Partial Class plot
    Inherits System.Web.UI.Page

    Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs) Handles Me.Load
        Dim memStream As New System.IO.MemoryStream
        Dim xygraph As New NPlot.Bitmap.PlotSurface2D(800, 600)

        Dim plots As LinePlot()
        Dim xvalues As List(Of Date)
        Dim yvalues As List(Of Double)()

        Dim linecolors As Color() = { _
            Color.Red, _
            Color.Blue, _
            Color.Lime, _
            Color.Black, _
            Color.Purple, _
            Color.Yellow, _
            Color.Cyan, _
            Color.Gray, _
            Color.Brown}

        Dim method As String = Request("method")
        Dim filename As String = Request("file")
        Dim names() As String = Request("names").Split(";")
        Dim plottitle As String = Request("title")
        Dim length As Integer = Request("length")
        Dim start As Integer

        If method Is Nothing Or method = "" Then method = "span"
        If method = "span" Then
            start = Request("start")
        End If

        If length = 0 Then length = 200

        Dim fso As New Scripting.FileSystemObject
        Dim ts As Scripting.TextStream = fso.OpenTextFile(filename, Scripting.IOMode.ForReading)


        Dim cnt As Integer = 0
        Dim text As String = ""
        Dim lines() As String = {}

        If method = "span" Then
            For i As Integer = 1 To start
                If ts.AtEndOfStream Then Exit For
                ts.SkipLine()
            Next


            For i As Integer = start To start + length
                If ts.AtEndOfStream Then Exit For
                text = text & ts.ReadLine & vbCrLf
                cnt = cnt + 1
            Next

            lines = text.Trim().Split(vbCrLf)
        ElseIf method = "last" Then

            text = ts.ReadAll()
            lines = text.Split(vbLf)
            Dim a As New ArrayList(lines)

            While a.Count > length
                a.RemoveAt(0)
            End While

            lines = CType(a.ToArray(GetType(String)), String())
        End If

        ts.Close()



        Dim plotcnt As Integer = names.GetUpperBound(0)

        xvalues = New List(Of Date)
        ReDim yvalues(plotcnt)
        For i As Integer = 0 To plotcnt
            yvalues(i) = New List(Of Double)
        Next

        ReDim plots(plotcnt)

        For i As Integer = 0 To lines.GetUpperBound(0)
            Dim line() As String = lines(i).Trim().Split(vbTab)
```

```
            If line.Length < 2 Then Continue For
            Dim d As Date = Date.Parse(line(0))
            xvalues.Add(d)
            For j As Integer = 1 To line.GetUpperBound(0)
                yvalues(j - 1).Add(Double.Parse(line(j)))
            Next
        Next

        xygraph.Clear()

        Dim myFont As New Font("Arial", 8, FontStyle.Bold)

        'Add a background grid for better chart readability.
        Dim grid As New Grid()
        grid.VerticalGridType = grid.GridType.Coarse
        grid.HorizontalGridType = grid.GridType.Coarse
        grid.MajorGridPen = New Pen(Color.LightGray, 1.0F)
        xygraph.Add(grid)

        xygraph.BackColor = Color.White

        xygraph.Title = plottitle
        xygraph.TitleColor = Color.Red

        For i As Integer = 0 To plotcnt
            plots(i) = New LinePlot()
            plots(i).Color = linecolors(i Mod linecolors.GetUpperBound(0))
            plots(i).AbscissaData = xvalues
            plots(i).DataSource = yvalues(i)
            plots(i).Label = names(i)
            xygraph.Add(plots(i))
        Next

        'Balance plot general settings.

        xygraph.YAxis1.Label = "Deformasjon [cm] / Kraft [N]"
        xygraph.YAxis1.LabelOffsetAbsolute = True
        xygraph.YAxis1.LabelOffset = 30
        'xygraph.YAxis1.WorldMin = 0
        'xygraph.YAxis1.WorldMax = 50

        xygraph.XAxis1.Label = "Tid [sek]"
        xygraph.Padding = 15

        Dim legend As New Legend()
        legend.AttachTo(PlotSurface2D.XAxisPosition.Top, PlotSurface2D.YAxisPosition.Left)
        legend.VerticalEdgePlacement = legend.Placement.Inside
        legend.HorizontalEdgePlacement = legend.Placement.Inside
        legend.BorderStyle = LegendBase.BorderType.Line
        legend.XOffset = 10
        legend.YOffset = 10
        legend.Font = myFont
        xygraph.Legend = legend

        'Refresh surfaces.
        xygraph.Refresh()

        Response.Buffer = True
        Response.ContentType = "image/Gif"
        xygraph.Bitmap.Save(memStream, System.Drawing.Imaging.ImageFormat.Gif)
        memStream.WriteTo(Response.OutputStream)
        Response.End()
    End Sub


End Class
```

Listing A.11: *Code for plot generator (plot.aspx.vb)*

Listing A.12: *Web page style sheet (styles.css)*

```
body
{
        font-family: Tahoma;
        font-size:10px;
}

a
{
        font-family: Tahoma;
}
table
{
        font-size:10px;
}
```

Listing A.12: *Web page style sheet (styles.css)*

A.3 Microcontroller Program for PWM-Control

Listing A.13: *Main program (main.c)*

```
#include "main.h"

```

```
3   uint8_t busy = 0;
4   uint8_t start_cal_deflection = 0;
5   uint8_t start_set_deflection = 0;
6   uint8_t start_cont_deflection = 0;
7
8   uint8_t start_cal_current = 0;
9   uint8_t start_set_current = 0;
10
11  uint8_t stop = 0;
12
13  uint8_t largs[6];
14  uint32_t defCal[3][2] = {{0,0},{0,0},{0,0}};
15  uint32_t curCal[3][16] = {{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0}, {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
        {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0}};
16
17  int main(){
18
19
20          DDRB = 0x0F;
21
22          initADC();
23          initUART();
24          initPwm();
25          initTimer();
26
27          sei();
28
29          while(1){
30
31                  if(start_cal_deflection != 0) calDeflection();
32                  if(start_set_deflection != 0) setDeflection();
33                  if(start_cont_deflection != 0) contDeflection();
34
35                  if(start_cal_current != 0) calCurrent();
36                  if(start_set_current != 0) setCurrent();
37          }
38
39          return 0;
40  }
41
42  void copyArgs(){
43          for(uint8_t i = 0; i < 6; i++) largs[i] = args[i];
44  }
45
46
47  /*
48  void setDeflection(){
49          //copyArgs();
50          start_set_deflection = 0;
51
52          uint32_t I;     //current
53          uint32_t D; //deflection
54
55          busy = 1;
56          stop = 0;
57
58          uint8_t pwm = 0;
59          //uint32_t setValue = ((defCal[args[0]][1] - defCal[args[0]][0]) * (uint8_t)args[1]) /255;
60          uint32_t setValue;
61
62          uint8_t cnt = 0;
63
64          while(stop == 0){
65                  I = readVolt(ADC_current[args[0]]);;
66                  D = readVolt(ADC_deflection[args[0]]);;
67
68                  setValue = defCal[args[0]][0] + (((defCal[args[0]][1] - defCal[args[0]][0]) * (uint8_t)args[1])
                        /255);
69
70                  if(D < setValue){
71                          if(pwm <= 255-args[2]) pwm+=args[2];
72                  }else if(D > setValue){
73                          if(pwm >= args[2]) pwm-=args[2];
74                  }
75
76                  setPwm(PWM_ch[args[0]], pwm);
77
78                  // Verbose
79                  if(args[3] != 0){
80                          if((cnt % 3) == 0){
81                                  sendUARTdec(pwm,0);
82                                  sendUART(' ');
83                                  sendUARTdec(I,3);
84                                  sendUART(' ');
85                                  sendUARTdecn(D,3);
86                          }
87                  }
88                  cnt++;
89
90          }
91
92          setPwm(PWM_ch[args[0]], 0);
93
94          busy = 0;
95          sendSuccess();
96          start_set_deflection = 0;
97  }
98  */
99
100 void setDeflection(){
101
102         uint32_t D; //deflection
103         uint32_t I; //current
104
105         busy = 1;
106         stop = 0;
107
108
109         uint8_t pwm = 0;
110         uint32_t setValue;
111         uint8_t reg;
112         uint8_t verbose;
113         uint8_t vf;
```

```
114 |
115 |         uint32_t min;
116 |         uint32_t max;
117 |         int32_t span;
118 |
119 |         uint8_t cnt = 0;
120 |         int32_t err = 0;
121 |         int32_t errP = 0;
122 |
123 |         min = defCal[0][0];
124 |         max = defCal[0][1];
125 |         span = max - min;
126 |
127 |         while(1){
128 |                 reg = args[1];
129 |                 verbose = args[2];
130 |                 vf = args[3];
131 |
132 |                 setValue = min + (((max - min) * (uint8_t)args[0]) /255);
133 |
134 |                 D = readVolt(ADC_deflection[0]);
135 |
136 |                 if(setValue > D){
137 |                         err =  setValue - D;
138 |                 }else{
139 |                         err = D - setValue;
140 |                         err *= -1;
141 |                 }
142 |
143 |                 errP = (err * 255) / span;
144 |                 errP = (errP * (int32_t)reg) / 64;
145 |
146 |                 if(errP > 0){
147 |                         if(pwm > 255-(uint32_t)errP){
148 |                                 pwm = 255;
149 |                         }else{
150 |                                 pwm += (uint32_t)errP;
151 |                         }
152 |                 }else{
153 |                         errP *= -1;
154 |                         if(pwm < (uint32_t)errP){
155 |                                 pwm = 0;
156 |                         }else{
157 |                                 pwm -= (uint32_t)errP;
158 |                         }
159 |                 }
160 |
161 |                 if(verbose && (cnt % vf) == 0) I = readVolt(ADC_current[0]);
162 |
163 |                 setPwm(PWM_ch[0], pwm);
164 |
165 |                 //sendUARTnewline();
166 |
167 |                 if(verbose && (cnt % vf) == 0){
168 |                         if(verbose & 0x01){
169 |                                 sendUARTdec(pwm,0);
170 |                                 sendUART(' ');
171 |                         }
172 |
173 |                         if(verbose & 0x02){
174 |                                 sendUARTdec(D,3);
175 |                                 sendUART(' ');
176 |                         }
177 |
178 |                         if(verbose & 0x04){
179 |                                 sendUARTdec(I,3);
180 |                                 sendUART(' ');
181 |                         }
182 |
183 |                         sendUARTnewline();
184 |
185 |                         cnt = 0;
186 |                 } else cnt++;
187 |
188 |                 if(stop==1) break;
189 |                 delay(10);
190 |         }
191 |
192 |         setPwm(PWM_ch[0], 0);
193 |
194 |
195 |         busy = 0;
196 |         sendSuccess();
197 |         start_set_deflection = 0;
198 | }
199 |
200 | void contDeflection(){
201 |         start_cont_deflection = 0;
202 |         copyArgs();
203 |
204 |         busy = 1;
205 |         uint8_t t = largs[3];
206 |
207 |         stop = 0;
208 |         while(stop == 0){
209 |                 for(uint8_t i = 0; i < 3; i++){
210 |                         if(args[i] == 1){
211 |                                 sendUARTdec(readVolt(ADC_deflection[i]),3);
212 |                                 sendUART(' ');
213 |                         }
214 |                 }
215 |                 sendUARTnewline();
216 |                 delay(t);
217 |         }
218 |         busy = 0;
219 |         start_cont_deflection = 0;
220 |         sendSuccess();
221 |
222 | }
223 |
224 | void calDeflection(){
225 |         copyArgs();
226 |         start_cal_deflection = 0;
```

```
227
228         defCal[0][0] = readVolt(ADC_deflection[0]);
229         setPwm(PWM_ch[0],255);
230         uint32_t tmp = defCal[0][0];
231
232         busy = 1;
233         stop = 0;
234         while(stop == 0){
235                 delay(500);
236                 defCal[0][1] = readVolt(ADC_deflection[0]);
237                 if(defCal[0][1]-tmp < 10) break;                    // Delta deflection small - finished
238                 tmp = defCal[0][1];
239         }
240
241         setPwm(PWM_ch[0],0);
242
243         sendUARTdec(defCal[0][0],3);
244         sendUART(' ');
245         sendUARTdecn(defCal[0][1],3);
246         busy = 0;
247 }
248
249 void setCurrent(){
250         start_set_current = 0;
251         copyArgs();
252         uint32_t I;      //current
253         uint32_t D; //deflection
254
255         busy = 1;
256         stop = 0;
257
258         uint8_t pwm = 0;
259         uint32_t setValue;
260
261         uint8_t cnt = 0;
262
263         while(stop == 0){
264                 I = readVolt(ADC_current[args[0]]);;
265                 D = readVolt(ADC_deflection[args[0]]);;
266
267                 setValue = curCal[largs[0]][largs[1]];
268
269                 if(I < setValue){
270                         if(pwm <= 255-args[2]) pwm+=args[2];
271                 }else if(I > setValue){
272                         if(pwm >= args[2]) pwm-=args[2];
273                 }
274
275                 setPwm(PWM_ch[args[0]], pwm);
276
277                 // Verbose
278                 if(args[3] != 0){
279                         if((cnt % 3) == 0){
280                                 //sendUARTdec(pwm,0);
281                                 //sendUART(' ');
282                                 //sendUARTdec(I,3);
283                                 //sendUART(' ');
284                                 sendUARTdecn(D,3);
285                         }
286                 }
287                 cnt++;
288
289         }
290
291         setPwm(PWM_ch[args[0]], 0);
292
293         busy = 0;
294         sendSuccess();
295         start_set_deflection = 0;
296 }
297
298
299 void calCurrent(){
300         start_cal_current = 0;
301
302         int32_t err = 0;
303         int32_t errP = 0;
304
305         uint32_t min = defCal[0][0];
306         uint32_t max = defCal[0][1];
307         int32_t span = max - min;
308
309         //uint32_t currents[16];
310         uint32_t D, I, setValue;
311         uint8_t pwm = 0;
312         stop = 0;
313         uint8_t reg = 150;
314
315
316         for(uint8_t i = 0; i < 16; i++){
317
318                 //setValue = defCal[0][0] + (((defCal[0][1] - defCal[0][0]) * i * 16) /255);
319
320                 setValue = min + (((max - min) * i * 16) /255);
321
322                 curCal[largs[0]][i] = 0;
323                 uint16_t ncal = 1000;
324                 for(uint16_t j = 0; j < ncal; j++){
325                         D = readVolt(ADC_deflection[0]);
326                         I = readVolt(ADC_current[0]);
327                         if(j > 49) curCal[largs[0]][i] += I;
328
329
330
331                         if(setValue > D){
332                                 err =  setValue - D;
333                         }else{
334                                 err = D - setValue;
335                                 err *= -1;
336                         }
337
338                         errP = (err * 255) / span;
339                         errP = (errP * (int32_t)reg) / 64;
```

```
340
341                             if(errP > 0){
342                                     if(pwm > 255-(uint32_t)errP){
343                                             pwm = 255;
344                                     }else{
345                                             pwm += (uint32_t)errP;
346                                     }
347                             }else{
348                                     errP *= -1;
349                                     if(pwm < (uint32_t)errP){
350                                             pwm = 0;
351                                     }else{
352                                             pwm -= (uint32_t)errP;
353                                     }
354                             }
355
356
357                             setPwm(PWM_ch[0], pwm);
358
359
360                             if(stop != 0){
361                                     setPwm(PWM_ch[0],0);
362                                     sendSuccess();
363                                     start_cal_current = 0;
364                                     return;
365                             }
366
367
368                             /*
369                             if(D < setValue){
370                                     if(pwm <= 255-100) pwm+=150;
371                             }else if(D > setValue){
372                                     if(pwm >= 100) pwm-=150;
373                             }
374                             setPwm(PWM_ch[0], pwm);
375
376                             if(stop != 0){
377                                     setPwm(PWM_ch[0],0);
378                                     sendSuccess();
379                                     start_cal_current = 0;
380                                     return;
381                             }
382                             */
383
384                     }
385
386                     curCal[largs[0]][i] /= (ncal-50);
387
388                     sendUARTdec(setValue,3);
389                     sendUART(' ');
390                     sendUARTdecn(curCal[largs[0]][i],3);
391             }
392
393             setPwm(PWM_ch[0],0);
394
395             start_cal_current = 0;
396
397     }
```

Listing A.13: *Main program (main.c)*

Listing A.14: *Header for main program (main.h)*

```
1   #ifndef _MAIN_H
2   #define _MAIN_H 1
3
4   #include <stdint.h>
5
6   #include "uart.h"
7   #include "pwm.h"
8   #include "adc.h"
9   #include "timer.h"
10  #include "utils.h"
11  #include "timer.h"
12
13  int main();
14  void setDeflection();
15  void calDeflection();
16  void setCurrent();
17  void contDeflection();
18
19  void setCurrent();
20  void calCurrent();
21
22  void copyArgs();
23
24
25  extern uint8_t busy;
26
27  extern uint8_t start_cal_current;
28  extern uint8_t start_set_current;
29
30  extern uint8_t start_cal_deflection;
31  extern uint8_t start_set_deflection;
32  extern uint8_t start_cont_deflection;
33
34  extern uint8_t stop;
35
36  #endif
```

Listing A.14: *Header for main program (main.h)*

Listing A.15: *Module for ADC operations (adc.c)*

```
#include "adc.h"

void initADC(){
        //DDRADC &= ~((1<<PINA0));
        DDRADC = 0x00;

        //Ref: AVCC with cap to AREF
        ADMUX = (0<<REFS1)|(1<<REFS0)|(0<<ADLAR);
        ADCSRA = (1<<ADEN)|(1<<ADPS2)|(1<<ADPS1)|(1<<ADPS0);



        readADC((1<<MUX0));
}


uint16_t readADC(uint8_t mux){
        ADMUX = (ADMUX & 0b11100000) | mux;
        ADCSRA |= (1<<ADSC);
        while ( ADCSRA & (1<<ADSC) );
        return ADC;
}

uint32_t readADCMiddel(uint8_t mux, uint8_t middel, uint16_t periode){

        ADMUX = (ADMUX & 0b11100000) | mux;

        uint32_t m = 0;
        uint8_t i;
        for(i = 0; i < middel; i++){
                ADCSRA |= (1<<ADSC);
                while ( ADCSRA & (1<<ADSC) );
                m += ADC;
                delay(periode);
        }

        return (uint32_t)(m/middel);
}

uint32_t readVolt(uint8_t mux){
        uint32_t adc = readADCMiddel(mux, 5, 1);
        uint32_t u = ((adc*1000)/1024) * 5;      // Covert ADC => Volt
        return u;
}
```

Listing A.15: *Module for ADC operations (adc.c)*

Listing A.16: *Header for ADC operations (adc.h)*

```
#ifndef _ADC_H
#define _ADC_H  1

#import <avr/io.h>
#import <util/delay.h>
#import "utils.h"

#define DDRADC  DDRA

#define ADC_0   0
#define ADC_1   1
#define ADC_2   2
#define ADC_3   3
#define ADC_4   4
#define ADC_5   5


uint16_t readADC(uint8_t mux);
uint32_t readADCMiddel(uint8_t mux, uint8_t middel, uint16_t periode);

void initADC();
uint32_t readVolt(uint8_t mux);


#endif
```

Listing A.16: *Header for ADC operations (adc.h)*

Listing A.17: *Module for command interpretation (interpreter.c)*

```
#include "interpreter.h"

uint32_t args[] = {0,0,0,0,0,0};
uint8_t ADC_current[]           = {ADC_1, ADC_4, ADC_5};
uint8_t ADC_deflection[]        = {ADC_0, ADC_1, ADC_2};
uint8_t PWM_ch[]                        = {PWM_0, PWM_1, PWM_2};

void analyzeCmd(char *cmd){
        if(busy != 0 && strncasecmp("BUSY?",cmd, 5) == 0){
                        sendUARTprintln((uint8_t *)"yes");
                        return;
        }

        // IDN? - Returns identification
        if(strncasecmp("IDN?",cmd, 4) == 0){
                sendUARTprintln((uint8_t *)"Flexinol uC regulator test");

        //PWM? - Returns PWM setting for specified channel
```

```
19          }else if(strncasecmp("PWM?",cmd, 4) == 0){
20                  if(readArgs((uint8_t *)cmd, 1) != 0){
21                          sendFailure((uint8_t *)"Wrong number of arguments! PWM? takes 1 argument");
22                          return;
23                  }
24                  uint8_t pwm = getPwm(PWM_ch[args[0]]);
25                  sendUARTdecn(pwm,0);
26
27
28
29          //PWM - Sets PWM value for specified channel
30          }else if(strncasecmp("PWM",cmd, 3) == 0){
31                  if(readArgs((uint8_t *)cmd, 2) != 0){
32                          sendFailure((uint8_t *)"Wrong number of arguments! PWM takes 2 arguments");
33                          return;
34                  }
35                  setPwm(PWM_ch[args[0]], args[1]);
36                  return;
37
38          //CURRENT? - Returns current for specified channel
39          }else if(strncasecmp("CURRENT?",cmd, 7) == 0){
40                  if(readArgs((uint8_t *)cmd, 1) != 0){
41                          sendFailure((uint8_t *)"Wrong number of arguments! CURRENT? takes 1 argument");
42                          return;
43                  }
44
45                  uint32_t u = readVolt(ADC_current[args[0]]);
46
47                  sendUARTdecn(u,3);
48
49          //DEFLECTION? - Returns deflection for specified channel
50          }else if(strncasecmp("DEFLECTION?",cmd, 11) == 0){
51                  if(readArgs((uint8_t *)cmd, 1) != 0){
52                          sendFailure((uint8_t *)"Wrong number of arguments! DEFLECTION? takes 1 argument");
53                          return;
54                  }
55                  uint32_t u = readVolt(ADC_deflection[args[0]]);
56
57                  sendUARTdecn(u,3);
58
59
60          //CAL_DEFLECTION
61          }else if(strncasecmp("CAL_DEFLECTION",cmd, 14) == 0){
62                  start_cal_deflection = 1;
63
64          //SET_DEFLECTION
65          }else if(strncasecmp("SET_DEFLECTION",cmd, 14) == 0){
66                  if(readArgs((uint8_t *)cmd, 4) != 0){
67                          sendFailure((uint8_t *)"Wrong number of arguments! SET_DEFLECTION takes 4 arguments");
68                          return;
69                  }
70                  start_set_deflection = 1;
71
72
73          //CAL_CURRENT
74          }else if(strncasecmp("CAL_CURRENT",cmd, 11) == 0){
75                  if(readArgs((uint8_t *)cmd, 1) != 0){
76                          sendFailure((uint8_t *)"Wrong number of arguments! CAL_CURRENT takes 1 argument");
77                          return;
78                  }
79                  start_cal_current = 1;
80
81          //SET_CURRENT
82          }else if(strncasecmp("SET_CURRENT",cmd, 11) == 0){
83                  if(readArgs((uint8_t *)cmd, 4) != 0){
84                          sendFailure((uint8_t *)"Wrong number of arguments! SET_CURRENT takes 4 arguments");
85                          return;
86                  }
87                  start_set_current = 1;
88
89          //CONT_DEFLECTION - Starts continuing deflection measurement
90          }else if(strncasecmp("CONT_DEFLECTION",cmd, 15) == 0){
91                  if(readArgs((uint8_t *)cmd, 4) != 0){
92                          sendFailure((uint8_t *)"Wrong number of arguments! CONT_DEFLECTION takes 4 arguments");
93                          return;
94                  }
95                  start_cont_deflection = 1;
96                  sendSuccess();
97
98          //BUSY? - Returns busy state of u-program
99          }else if(strncasecmp("BUSY?",cmd, 5) == 0){
100                 sendUARTprintln((uint8_t *)"no");
101
102         //STOP - Stops the current operation
103         }else if(strncasecmp("STOP",cmd, 4) == 0){
104                 stop = 1;
105
106         }else{
107                 sendFailure((uint8_t *)"Command not recognized!");
108         }
109
110 }
111
112 uint8_t readArgs(uint8_t *cmd, uint8_t n){
113         uint8_t arg_n = 0;
114         uint8_t start = 1;
115         while(*cmd && (*cmd != '\n')){
116                 if(*cmd == ' '){
117                         if(start == 1) start = 0; else arg_n++;
118                         cmd++;
119                 }else{
120                         if(start == 0){
121                                 args[arg_n] = readArg(cmd);
122                                 //sendUARTdec(args[arg_n],1);
123                                 while(*cmd && (*cmd != ' ') && (*cmd != '\n')) cmd++;
124                         }else{
125                                 cmd++;
126                         }
127                 }
128
129         }
130
131         if(arg_n+1 >= n && start==0) return 0;
```

```
132         return 1;
133 }
134
135 uint32_t readArg(uint8_t *cmd){
136         uint32_t v = 0;
137         uint8_t zero = '0';
138         while(*cmd && (*cmd != '\n')){
139                 if(*cmd == ' ') return v;
140                 v *= 10;
141                 v += (*cmd - zero);
142                 cmd++;
143         }
144
145         return v;
146 }
147
148 void sendSuccess(){
149         sendUARTprintln((uint8_t *)"OK!");
150 }
151
152 void sendFailure(uint8_t *message){
153         sendUARTprint((uint8_t *)"ERR! ");
154         sendUARTprintln(message);
155 }
```

Listing A.17: *Module for command interpretation (interpreter.c)*

Listing A.18: *Header for command interpretation (interpreter.h)*

```
1  #ifndef _INTERPRETER_H
2  #define _INTERPRETER_H  1
3
4  #include <string.h>
5  #include "uart.h"
6  #include "main.h"
7  #include "adc.h"
8
9  void analyzeCmd(char *cmd);
10 uint8_t readArgs(uint8_t *cmd, uint8_t n);
11 uint32_t readArg(uint8_t *cmd);
12
13 void sendSuccess();
14 void sendFailure(uint8_t *message);
15
16 extern uint32_t args[];
17 extern uint8_t ADC_current[];
18 extern uint8_t ADC_deflection[];
19 extern uint8_t PWM_ch[];
20 #endif
```

Listing A.18: *Header for command interpretation (interpreter.h)*

Listing A.19: *Module for PWM operations (pwm.c)*

```
1  #include "pwm.h"
2
3  void initPwm(){
4          DDRPWM1 |= (1<<OC0);
5          DDRPWM2 |= (1<<OC1A)|(1<<OC1B);
6
7
8          //Timer/counter 0 (8 bit)
9          // Fast PWM, inverting mode (for å få 'ren' 0), prescaling 256 (225Hz)
10         TCCR0 = (1<<WGM01)|(1<<WGM00)|(1<<COM01)|(1<<COM00)|(0<<CS02)|(0<<CS01)|(1<<CS00);
11         OCR0 = 255;
12
13         //Timer/counter 1 (16 bit)
14         // Fast PWM 8bit, inverting mode (for å få 'ren' 0), prescaling 256 (225Hz)
15         TCCR1A = (1<<COM1A1)|(0<<COM1A0)|(1<<COM1B1)|(1<<COM1B0)|(0<<FOC1A)|(0<<FOC1B)|(0<<WGM11)|(1<<WGM10);
16         TCCR1B = (0<<ICNC1)|(0<<ICES1)|(0<<WGM13)|(1<<WGM12)|(0<<CS12)|(0<<CS11)|(1<<CS10);
17
18         OCR1A = 255;
19         OCR1B = 255;
20
21 }
22
23 void setPwm(uint8_t channel, uint8_t value){
24         switch(channel){
25                 case PWM_0: OCR0 = 255-value; break;
26                 case PWM_1: OCR1A = 255-value; break;
27                 case PWM_2: OCR1B = 255-value; break;
28         }
29 }
30
31 uint8_t getPwm(uint8_t channel){
32         switch(channel){
33                 case PWM_0: return 255-(uint8_t)OCR0;
34                 case PWM_1: return 255-(uint8_t)OCR1A;
35                 case PWM_2: return 255-(uint8_t)OCR1B;
36         }
37         return 0;
38 }
```

Listing A.19: *Module for PWM operations (pwm.c)*

Listing A.20: *Header for PWM operations (pwm.h)*

```
#ifndef _PWM_H
#define _PWM_H  1

#import <avr/io.h>

#define DDRPWM1 DDRB
#define DDRPWM2 DDRD

#define OC0             PINB3
#define OC1A    PIND5
#define OC1B    PIND4

#define PWM_0   0
#define PWM_1   1
#define PWM_2   2

void initPwm();
void setPwm(uint8_t channel, uint8_t value);
uint8_t getPwm(uint8_t channel);

#endif
```

Listing A.20: *Header for PWM operations (pwm.h)*

Listing A.21: *Module for Timer/Counter operations (timer.c)*

```
#include "timer.h"

uint8_t timCnt = 0;

void initTimer(){
        //Timer 2, CTC-mode (TOP=OCR2), prescaler 1024
        TCCR2 = (1<<WGM21)|(0<<WGM20)|(1<<CS22)|(1<<CS21)|(1<<CS20);

        OCR2 = 239;

        TIMSK |= (1<<OCIE2);
}


ISR(SIG_OUTPUT_COMPARE2)
{
        timCnt = (timCnt+1) % 6;
        // timCnt == 0 => 10ms
        if(timCnt == 0) PORTB ^= 0x01;
        return;
}
```

Listing A.21: *Module for Timer/Counter operations (timer.c)*

Listing A.22: *Header for Timer/Counter operations (timer.h)*

```
#ifndef _TIMER_H
#define _TIMER_H        1

#import <avr/io.h>
#import <avr/interrupt.h>

void initTimer();

#endif
```

Listing A.22: *Header for Timer/Counter operations (timer.h)*

Listing A.23: *Module for UART operations (uart.c)*

```
#include "uart.h"

char rx_buf[rx_bufsize];                // Rx-buffer
uint8_t rx_tail = 0;                    // Rx-buffer index


void initUART(){

        UART_DDR |=  (1<<UART_TX);
        UART_DDR &= ~(1<<UART_RX);

        UBRRH = (uint8_t)(MY_UBRR >> 8);
        UBRRL = (uint8_t)MY_UBRR;
        UCSRB = (1<<RXEN)|(1<<TXEN)|(1<<RXCIE);
        UCSRC = (1<<URSEL)|(3<<UCSZ0);
}

void sendUART(uint8_t c){
        while ( !( UCSRA & (1<<UDRE)) );
        UDR = c;
}

void sendUARTprintln(uint8_t *s){
        sendUARTprint(s);
        sendUARTnewline();
}

```

```

void sendUARTprint(uint8_t *s){
        while(*s){
                sendUART(*s);
                s++;
        }
}

/*
void sendUARTbyte(uint8_t b){
        uint8_t i = 1;
        for(i = 0; i < 8; i++){
                if((b & i) == 0){
                        sendUART('0');
                }else{
                        sendUART('1');
                }
        }
}
*/

void sendUARTint(uint32_t n){
        char s[10];
        uint8_t i = 0;
        while(n > 0){
                s[i++] = '0' + (n % 10);
                n /= 10;
        }

        if(i==0){
                sendUART('0');
        }else{
                for(uint8_t j = i; j > 0; j--){
                        sendUART(s[j-1]);
                }
        }
}

void sendUARTdec(uint32_t n, uint8_t dec){
        char s[10];
        uint8_t i = 0;
        while(n > 0){
                s[i++] = '0' + (n % 10);
                n /= 10;
        }

        if(i==0){
                sendUART('0');
        }else if(i > 0 && dec >= i){
                sendUARTprint((uint8_t *)"0.");
                for(uint8_t j = dec; j > 0; j--){
                        if(j > i){
                                sendUART('0');
                        }else{
                                sendUART(s[j-1]);
                        }
                }
        }else{
                for(uint8_t j = i; j > 0; j--){
                        if(j == dec) sendUART('.');
                        sendUART(s[j-1]);
                }
        }
}

void sendUARTdecn(uint32_t n, uint8_t dec){
        sendUARTdec(n,dec);
        sendUARTnewline();
}

void sendUARTnewline(){
        sendUARTprint((uint8_t *)newline);
}

char readUART(){
        while(!(UCSRA & (1<<RXC)));

        return UDR;
}


ISR(SIG_UART_RECV)
{
        char data = UDR;

        if(rx_tail == rx_bufsize-1){
                //Buffer full
                rx_tail = 0;
                sendUARTprintln((uint8_t *)"Buffer full! Wait for 'OK' before sending a new command");
        }else{
                rx_buf[rx_tail++] = data;
                if(data=='\n'){
                        analyzeCmd(rx_buf);
                        rx_tail = 0;
                }
        }

        return;
}
```

Listing A.23: *Module for UART operations (uart.c)*

Listing A.24: *Header for UART operations (uart.h)*

```

#ifndef _UART_H
#define _UART_H 1

```

```
#import <avr/io.h>
#import <util/delay.h>
#import <avr/interrupt.h>
#include <avr/eeprom.h>
#include "interpreter.h"

#define CLOCK                   F_CPU
#define BAUD                    57600
#define MY_UBRR                 ((CLOCK)/((BAUD)*16L)-1)

#define UART_DDR                DDRD
#define UART_TX                 PD1
#define UART_RX                 PD0

#define newline                 "\n"

void initUART();
void sendUART(uint8_t c);
void sendUARTprintln(uint8_t *s);
void sendUARTprint(uint8_t *s);
void sendUARTbyte(uint8_t b);
void sendUARTint(uint32_t n);
void sendUARTdec(uint32_t n, uint8_t dec);
void sendUARTdecn(uint32_t n, uint8_t dec);
void sendUARTnewline();
char readUART();


#define rx_bufsize      50

#endif
```

Listing A.24: *Header for UART operations (uart.h)*

Listing A.25: *Utilities module (utils.c)*

```
#include "utils.h"

void delay(uint16_t ms){
        while(ms > 16){
                _delay_ms(16);
                ms -= 16;
        }

        _delay_ms(ms);
}
```

Listing A.25: *Utilities module (utils.c)*

Listing A.26: *Header for utilities module (utils.h)*

```
#ifndef _UTILS_H
#define _UTILS_H        1

#import <util/delay.h>

void delay(uint16_t ms);


#endif
```

Listing A.26: *Header for utilities module (utils.h)*

A.4 Microcontroller Program for Finger Control

The below code is the implementation of the program described in section 4.5.4.

Listing A.27: *Main program (main.c)*

```
#include "main.h"

uint8_t busy = 0;
uint8_t start_cont_current = 0;
uint8_t start_cal_current = 0;
uint8_t start_set_current = 0;
uint8_t start_cont_deflection = 0;
uint8_t start_cont_current_and_deflection = 0;
uint8_t start_cal_deflection = 0;
uint8_t start_set_deflection = 0;
uint8_t start_set_deflections = 0;
uint8_t start_cont_force = 0;
uint8_t start_set_force = 0;

uint8_t start_fast = 0;
uint8_t stop = 0;

uint8_t largs[6];
uint16_t defCal[3][2] = {{0,0},{0,0},{0,0}};
```

```
20  uint32_t curCal[3][16] = {{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0}, {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
        {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0}};
21
22  uint16_t e_dc0_min EEMEM = 0x0000;
23  uint16_t e_dc0_max EEMEM = 0x0000;
24  uint16_t e_dc1_min EEMEM = 0x0000;
25  uint16_t e_dc1_max EEMEM = 0x0000;
26  uint16_t e_dc2_min EEMEM = 0x0000;
27  uint16_t e_dc2_max EEMEM = 0x0000;
28
29  int main(){
30
31          wdt_disable();
32
33          DDRB = 0x0F;
34          PORTB = 0x00;
35          DDRD = (1<<PIND7);
36          PORTD = 0x00;
37
38          initPwm();
39          initADC();
40          initUART();
41          initTimer();
42
43          readEEMEM();
44
45          sei();
46
47          PORTD |= (1<<PIND7);
48          delay(200);
49          PORTD &= ~(1<<PIND7);
50
51
52          while(1){
53
54
55                  if(start_cont_current != 0) contCurrent();
56                  if(start_cal_current != 0) calCurrent();
57                  if(start_set_current != 0) setCurrent();
58
59                  if(start_cont_current_and_deflection != 0) contCurrentAndDeflection();
60
61                  if(start_cont_deflection != 0) contDeflection();
62                  if(start_cal_deflection != 0) calDeflection();
63                  if(start_set_deflection != 0) setDeflection();
64
65                  if(start_set_deflections != 0) setDeflections();
66
67                  if(start_cont_force != 0) contForce();
68                  if(start_set_force != 0) setForce();
69
70                  if(start_fast != 0) fastLoop();
71          }
72
73          return 0;
74  }
75
76  void copyArgs(){
77          for(uint8_t i = 0; i < 6; i++) largs[i] = args[i];
78  }
79
80  void contCurrent(){
81          copyArgs();
82
83          busy = 1;
84          uint8_t t = largs[3];
85          //for(uint8_t i = 0; i < 3; i++) if(args[i] == 1) t -= 3;
86          stop = 0;
87          while(!stop){
88                  for(uint8_t i = 0; i < 3; i++){
89                          if(args[i] == 1){
90                                  sendUARTdec(readVolt(ADC_current[i]),3);
91                                  sendUART(' ');
92                          }
93                  }
94                  sendUARTnewline();
95                  delay(t);
96          }
97          busy = 0;
98          sendSuccess();
99
100         start_cont_current = 0;
101 }
102
103 void contDeflection(){
104         start_cont_deflection = 0;
105         copyArgs();
106
107         busy = 1;
108         uint8_t t = largs[3];
109
110         stop = 0;
111         while(!stop){
112                 for(uint8_t i = 0; i < 3; i++){
113                         if(args[i] == 1){
114                                 sendUARTdec(readVolt(ADC_deflection[i]),3);
115                                 sendUART(' ');
116                         }
117                 }
118                 sendUARTnewline();
119                 delay(t);
120         }
121         busy = 0;
122         start_cont_deflection = 0;
123         sendSuccess();
124
125 }
126
127 void contForce(){
128         copyArgs();
129
130         busy = 1;
131         uint8_t t = largs[3];
```

```
132         //for(uint8_t i = 0; i < 3; i++) if(args[i] == 1) t -= 3;
133         stop = 0;
134         while(!stop){
135                 for(uint8_t i = 0; i < 3; i++){
136                         if(args[i] == 1){
137                                 sendUARTdec(readVolt(ADC_force[i]),3);
138                                 sendUART(' ');
139                         }
140                 }
141                 sendUARTnewline();
142                 delay(t);
143         }
144         busy = 0;
145         sendSuccess();
146         start_cont_force = 0;
147 }
148
149 void contCurrentAndDeflection(){
150         copyArgs();
151
152         start_cont_current_and_deflection = 0;
153         busy = 1;
154         uint8_t t = largs[6];
155         //for(uint8_t i = 0; i < 6; i++) if(largs[i] == 1) t -= 10;
156         stop = 0;
157         while(stop == 0){
158                 for(uint8_t i = 0; i < 3; i++){
159                         if(largs[i] == 1){
160                                 sendUARTdec(readVolt(ADC_current[i]),3);
161                                 sendUART(' ');
162                         }
163                 }
164
165                 for(uint8_t i = 0; i < 3; i++){
166                         if(largs[i+3] == 1){
167                                 sendUARTdec(readVolt(ADC_deflection[i]),3);
168                                 sendUART(' ');
169                         }
170                 }
171                 sendUARTnewline();
172                 delay(t);
173         }
174         busy = 0;
175         sendSuccess();
176 }
177
178 void setDeflection(){
179         copyArgs();
180         start_set_deflection = 0;
181
182         uint32_t D; //deflection
183
184         busy = 1;
185         stop = 0;
186
187         uint32_t min = defCal[largs[0]][0];
188         uint32_t max = defCal[largs[0]][1];
189
190         uint8_t pwm = 0;
191         uint32_t setValue = min + (((max - min) * (uint8_t)largs[1]) /255);
192
193
194
195         sendUARTdecn(setValue,3);
196
197         uint8_t cnt = 0;
198
199         while(1){
200                 D = readVolt(ADC_deflection[largs[0]]);;
201
202                 if(D < setValue){
203                         if(pwm <= 255-args[2]) pwm+=largs[2];
204                 }else if(D > setValue){
205                         if(pwm >= args[2]) pwm-=largs[2];
206                 }
207
208                 setPwm(PWM_ch[largs[0]], pwm);
209
210                 // Verbose
211                 if(largs[3] != 0){
212                         if((cnt % 30) == 0){
213                                 sendUARTdec(pwm,0);
214                                 sendUART(' ');
215                                 //sendUARTdec(I,3);
216                                 //sendUART(' ');
217                                 sendUARTdecn(D,3);
218                         }
219                 }
220                 cnt++;
221                 if(stop==1) break;
222                 delay(10);
223
224         }
225
226         setPwm(PWM_ch[largs[0]], 0);
227
228         busy = 0;
229         sendSuccess();
230         start_set_deflection = 0;
231 }
232
233
234 void setDeflections(){
235
236         uint32_t D[3]; //deflection
237         uint32_t F[3]; //force
238
239         busy = 1;
240         stop = 0;
241
242
243         uint8_t pwm[3] = {0, 0, 0};
244         uint32_t setValues[3];
```

```
245         uint8_t reg;
246         uint8_t verbose;
247         uint8_t vf;
248 
249         uint32_t min[3];
250         uint32_t max[3];
251         int32_t span[3];
252 
253         uint8_t cnt = 0;
254         int32_t err = 0;
255         int32_t errP = 0;
256 
257         for(uint8_t i = 0; i < 3; i++){
258                 min[i] = defCal[i][0];
259                 max[i] = defCal[i][1];
260                 span[i] = max[i] - min[i];
261         }
262 
263         while(1){
264                 reg = args[3];
265                 verbose = args[4];
266                 vf = args[5];
267 
268                 for(uint8_t i = 0; i < 3; i++){
269                         setValues[i] = min[i] + (((max[i] - min[i]) * (uint8_t)args[i]) /255);
270                         //sendUARTdec(setValues[i], 3);
271 
272                         D[i] = readVolt(ADC_deflection[i]);
273 
274                         if(setValues[i] > D[i]){
275                                 err =  setValues[i] - D[i];
276                         }else{
277                                 err = D[i] - setValues[i];
278                                 err *= -1;
279                         }
280 
281                         errP = (err * 255) / span[i];
282                         errP = (errP * (int32_t)reg) / 64;
283 
284                         if(errP > 0){
285                                 if(pwm[i] > 255-(uint32_t)errP){
286                                         pwm[i] = 255;
287                                 }else{
288                                         pwm[i] += (uint32_t)errP;
289                                 }
290                         }else{
291                                 errP *= -1;
292                                 if(pwm[i] < (uint32_t)errP){
293                                         pwm[i] = 0;
294                                 }else{
295                                         pwm[i] -= (uint32_t)errP;
296                                 }
297                         }
298 
299                         /*
300                         if(D[i] < setValues[i]){
301                                 if(pwm[i] <= 255-reg) pwm[i]+=reg;
302                         }else if(D[i] > setValues[i]){
303                                 if(pwm[i] >= reg) pwm[i]-=reg;
304                         }
305                         */
306 
307                         if(verbose && (cnt % vf) == 0) F[i] = readVolt(ADC_force[i]);
308 
309                         setPwm(PWM_ch[i], pwm[i]);
310                 }
311                 //sendUARTnewline();
312 
313                 if(verbose && (cnt % vf) == 0){
314                         if(verbose & 0x01){
315                                 for(uint8_t i = 0; i < 3; i++){
316                                         sendUARTdec(pwm[i],0);
317                                         sendUART(' ');
318                                 }
319                         }
320 
321                         if(verbose & 0x02){
322                                 for(uint8_t i = 0; i < 3; i++){
323                                         sendUARTdec(D[i],3);
324                                         sendUART(' ');
325                                 }
326                         }
327 
328                         if(verbose & 0x04){
329                                 for(uint8_t i = 0; i < 3; i++){
330                                         sendUARTdec(F[i],3);
331                                         sendUART(' ');
332                                 }
333                         }
334 
335                         sendUARTnewline();
336 
337                         cnt = 0;
338                 } else cnt++;
339 
340                 if(stop==1) break;
341         }
342 
343         for(uint8_t i = 0; i < 3; i++){
344                 setPwm(PWM_ch[i], 0);
345         }
346 
347         busy = 0;
348         sendSuccess();
349         start_set_deflections = 0;
350 }
351 
352 void setForce(){
353         copyArgs();
354 
355         uint32_t F; //force
356 
357         busy = 1;
```

```
358         stop = 0;
359
360         uint8_t pwm = 0;
361         uint32_t setValue = largs[1];
362
363         uint8_t cnt = 0;
364
365         while(1){
366                 F = readVolt(ADC_force[largs[0]]);;
367
368
369                 if(F < setValue){
370                         if(pwm <= 255-args[2]) pwm+=largs[2];
371                 }else if(F > setValue){
372                         if(pwm >= args[2]) pwm-=largs[2];
373                 }
374
375                 setPwm(PWM_ch[largs[0]], pwm);
376
377                 // Verbose
378                 if(largs[3] != 0){
379                         if((cnt % 30) == 0){
380                                 //sendUARTdec(pwm,0);
381                                 //sendUART(' ');
382                                 //sendUARTdec(I,3);
383                                 //sendUART(' ');
384                                 sendUARTdecn(F,3);
385                         }
386                 }
387                 cnt++;
388                 if(stop==1) break;
389                 delay(10);
390
391         }
392
393         setPwm(PWM_ch[largs[0]], 0);
394
395         busy = 0;
396         sendSuccess();
397         start_set_force = 0;
398 }
399
400 void calDeflection(){
401         copyArgs();
402         start_cal_deflection = 0;
403
404         for(uint8_t i = 0; i < 3; i++){
405                 defCal[i][0] = readVolt(ADC_deflection[i]);
406                 setPwm(PWM_ch[i],255);
407                 uint32_t tmp = defCal[i][0];
408
409                 busy = 1;
410                 stop = 0;
411                 delay(1500);
412                 while(stop == 0){
413
414                         delay(500);
415                         defCal[i][1] = readVolt(ADC_deflection[i]);
416                         if(defCal[i][1]-tmp < 10) break;                 // Delta deflection small - finished
417                         tmp = defCal[i][1];
418                 }
419
420                 setPwm(PWM_ch[i],0);
421                 sendUARTint(i);
422                 sendUARTprint((uint8_t *)" - min: ");
423                 sendUARTdec(defCal[i][0],3);
424                 sendUARTprint((uint8_t *)" max: ");
425                 sendUARTdecn(defCal[i][1],3);
426
427                 delay(2000);
428         }
429         busy = 0;
430         start_cal_deflection = 0;
431         writeEEMEM();
432 }
433
434
435 void setCurrent(){
436         start_set_current = 0;
437         copyArgs();
438         uint32_t I;      //current
439         uint32_t D; //deflection
440
441         busy = 1;
442         stop = 0;
443
444         uint8_t pwm = 0;
445         uint32_t setValue;
446
447         uint8_t cnt = 0;
448
449         while(stop == 0){
450                 I = readVolt(ADC_current[args[0]]);;
451                 D = readVolt(ADC_deflection[args[0]]);;
452
453                 setValue = curCal[largs[0]][largs[1]];
454
455                 if(I < setValue){
456                         if(pwm <= 255-args[2]) pwm+=args[2];
457                 }else if(I > setValue){
458                         if(pwm >= args[2]) pwm-=args[2];
459                 }
460
461                 setPwm(PWM_ch[args[0]], pwm);
462
463                 // Verbose
464                 if(args[3] != 0){
465                         if((cnt % 3) == 0){
466                                 sendUARTdec(pwm,0);
467                                 sendUART(' ');
468                                 sendUARTdec(I,3);
469                                 sendUART(' ');
470                                 sendUARTdecn(D,3);
```

```
471                         }
472                 }
473                 cnt++;
474
475         }
476
477         setPwm(PWM_ch[args[0]], 0);
478
479         busy = 0;
480         sendSuccess();
481         start_set_deflection = 0;
482 }
483
484
485 void calCurrent(){
486         start_cal_current = 0;
487
488         //uint32_t currents[16];
489         uint32_t D, I, setValue;
490         uint8_t pwm = 0;
491         stop = 0;
492         for(uint8_t n = 0; n < 20; n++){
493
494                 for(uint8_t i = 0; i < 16; i++){
495
496                         setValue = defCal[0][0] + (((defCal[0][1] - defCal[0][0]) * i * 16) /255);
497                         curCal[largs[0]][i] = 0;
498                         for(uint8_t j = 0; j < 150; j++){
499                                 D = readVolt(ADC_deflection[0]);
500                                 I = readVolt(ADC_current[0]);
501                                 if(j > 49) curCal[largs[0]][i] += I;
502                                 if(D < setValue){
503                                         if(pwm <= 255-100) pwm+=100;
504                                 }else if(D > setValue){
505                                         if(pwm >= 100) pwm-=100;
506                                 }
507                                 setPwm(PWM_ch[0], pwm);
508
509                                 if(stop != 0){
510                                         setPwm(PWM_ch[0],0);
511                                         sendSuccess();
512                                         start_cal_current = 0;
513                                         return;
514                                 }
515                         }
516
517                         curCal[largs[0]][i] /= 100;
518
519                         sendUARTdec(setValue,3);
520                         sendUART(' ');
521                         sendUARTdecn(curCal[largs[0]][i],3);
522                 }
523
524                 setPwm(PWM_ch[0],0);
525
526                 start_cal_current = 0;
527                 sendUARTint(n+1);
528                 //sendUARTprintln((uint8_t *)"OK!");
529                 //sendSuccess();
530                 delay(10000);
531         }
532 }
533
534 void fastLoop(){
535
536         uint32_t D;
537
538
539         uint8_t pwm = 0;
540         uint8_t ch = args[0];
541         uint32_t setValue = defCal[ch][0] + (((defCal[ch][1] - defCal[ch][0]) * args[1]) /255);
542         uint8_t p = (uint8_t)args[2];
543
544         stop = 0;
545         while(stop == 0){
546                 D = readVolt(ADC_deflection[ch]);;
547
548                 if(D < setValue){
549                         if(pwm <= 255-p) pwm+=p;
550                 }else if(D > setValue){
551                         if(pwm >= p) pwm-=p;
552                 }
553
554                 setPwm(PWM_ch[ch], pwm);
555
556         }
557         start_fast = 0;
558         setPwm(PWM_ch[ch], 0);
559
560
561 }
562
563 ISR(BADISR_vect)
564 {
565     PORTD ^= (1 << PIND7);
566 }
567
568
569 void readEEMEM(){
570         defCal[0][0] = eeprom_read_word(&e_dc0_min);
571         defCal[0][1] = eeprom_read_word(&e_dc0_max);
572         defCal[1][0] = eeprom_read_word(&e_dc1_min);
573         defCal[1][1] = eeprom_read_word(&e_dc1_max);
574         defCal[2][0] = eeprom_read_word(&e_dc2_min);
575         defCal[2][1] = eeprom_read_word(&e_dc2_max);
576
577 }
578
579 void writeEEMEM(){
580         eeprom_write_word(&e_dc0_min, defCal[0][0]);
581         eeprom_write_word(&e_dc0_max, defCal[0][1]);
582         eeprom_write_word(&e_dc1_min, defCal[1][0]);
583         eeprom_write_word(&e_dc1_max, defCal[1][1]);
```

```
584         eeprom_write_word(&e_dc2_min, defCal[2][0]);
585         eeprom_write_word(&e_dc2_max, defCal[2][1]);
586 }
```

Listing A.27: *Main program (main.c)*

Listing A.28: *Header for main program (main.h)*

```
1  #ifndef _MAIN_H
2  #define _MAIN_H 1
3
4  #include <stdint.h>
5  #include <avr/wdt.h>
6  #include <avr/eeprom.h>
7
8  #include "uart.h"
9  #include "pwm.h"
10 #include "adc.h"
11 #include "timer.h"
12 #include "utils.h"
13 #include "timer.h"
14
15 int main();
16 void contCurrent();
17 void contDeflection();
18 void contCurrentAndDeflection();
19 void contForce();
20 void setForce();
21 void setDeflection();
22 void setDeflections();
23 void calDeflection();
24 void setCurrent();
25 void calCurrent();
26 void copyArgs();
27
28 void fastLoop();
29
30 void readEEMEM();
31 void writeEEMEM();
32
33 extern uint8_t busy;
34 extern uint8_t start_cont_current;
35 extern uint8_t start_cal_current;
36 extern uint8_t start_set_current;
37 extern uint8_t start_cont_deflection;
38 extern uint8_t start_cont_current_and_deflection;
39 extern uint8_t start_cal_deflection;
40 extern uint8_t start_set_deflection;
41 extern uint8_t start_set_deflections;
42 extern uint8_t start_cont_force;
43 extern uint8_t start_set_force;
44
45 extern uint8_t stop;
46
47 extern uint8_t start_fast;
48
49 #endif
```

Listing A.28: *Header for main program (main.h)*

Listing A.29: *Module for ADC operations (adc.c)*

```
1  #include "adc.h"
2
3  void initADC(){
4          //DDRADC &= ~((1<<PINA0));
5          DDRADC = 0x00;
6
7          //Ref: AVCC with cap to AREF
8          ADMUX = (0<<REFS1)|(1<<REFS0)|(0<<ADLAR);
9          ADCSRA = (1<<ADEN)|(1<<ADPS2)|(1<<ADPS1)|(1<<ADPS0);
10
11
12
13         readADC((1<<MUX0));
14 }
15
16
17 uint16_t readADC(uint8_t mux){
18         ADMUX = (ADMUX & 0b11100000) | mux;
19         ADCSRA |= (1<<ADSC);
20         while ( ADCSRA & (1<<ADSC) );
21         return ADC;
22 }
23
24 uint32_t readADCMiddel(uint8_t mux, uint8_t middel, uint16_t periode){
25
26         ADMUX = (ADMUX & 0b11100000) | mux;
27
28         uint32_t m = 0;
29         uint8_t i;
30         for(i = 0; i < middel; i++){
31                 ADCSRA |= (1<<ADSC);
32                 while ( ADCSRA & (1<<ADSC) );
33                 m += ADC;
34                 delay(periode);
35         }
36
37         return (uint32_t)(m/middel);
38 }
```

```
39
40 uint32_t readVolt(uint8_t mux){
41         uint32_t adc = readADCMiddel(mux, 5, 1);
42         uint32_t u = ((adc*1000)/1024) * 5;      // Covert ADC => Volt
43         return u;
44 }
```

Listing A.29: *Module for ADC operations (adc.c)*

Listing A.30: *Header for ADC operations (adc.h)*

```
1  #ifndef _ADC_H
2  #define _ADC_H  1
3
4  #import <avr/io.h>
5  #import <util/delay.h>
6  #import "utils.h"
7
8  #define DDRADC  DDRA
9
10 #define ADC_0   0
11 #define ADC_1   1
12 #define ADC_2   2
13 #define ADC_3   3
14 #define ADC_4   4
15 #define ADC_5   5
16
17
18 uint16_t readADC(uint8_t mux);
19 uint32_t readADCMiddel(uint8_t mux, uint8_t middel, uint16_t periode);
20
21 void initADC();
22 uint32_t readVolt(uint8_t mux);
23
24
25 #endif
```

Listing A.30: *Header for ADC operations (adc.h)*

Listing A.31: *Module for command interpretation (interpreter.c)*

```
1  #include "interpreter.h"
2
3  uint32_t args[] = {0,0,0,0,0,0,0};
4  uint8_t ADC_current[]          = {ADC_3, ADC_4, ADC_5};
5  uint8_t ADC_deflection[]       = {ADC_0, ADC_1, ADC_2};
6  uint8_t ADC_force[]            = {ADC_3, ADC_4, ADC_5};
7  uint8_t PWM_ch[]                        = {PWM_0, PWM_1, PWM_2};
8
9  void analyzeCmd(char *cmd){
10         if(busy != 0 && strncasecmp("BUSY?",cmd, 5) == 0){
11                 sendUARTprintln((uint8_t *)"yes");
12                 return;
13         }
14
15         // IDN? - Returns identification
16         if(strncasecmp("IDN?",cmd, 4) == 0){
17                 sendUARTprintln((uint8_t *)"Flexinol uC demo v2.0");
18
19         //PWM? - Returns PWM setting for specified channel
20         }else if(strncasecmp("PWM?",cmd, 4) == 0){
21                 if(readArgs((uint8_t *)cmd, 1) != 0){
22                         sendWrongNoOfArgs((uint8_t *)"PWM?", 1);
23                         return;
24                 }
25                 uint8_t pwm = getPwm(PWM_ch[args[0]]);
26                 sendUARTdecn(pwm,0);
27
28
29
30         //PWM - Sets PWM value for specified channel
31         }else if(strncasecmp("PWM",cmd, 3) == 0){
32                 if(readArgs((uint8_t *)cmd, 2) != 0){
33                         sendWrongNoOfArgs((uint8_t *)"PWM", 2);
34                         //sendFailure((uint8_t *)"Wrong number of arguments! PWM takes 2 arguments");
35                         return;
36                 }
37                 setPwm(PWM_ch[args[0]], args[1]);
38                 return;
39
40         //CURRENT? - Returns current for specified channel
41         }else if(strncasecmp("CURRENT?",cmd, 7) == 0){
42                 if(readArgs((uint8_t *)cmd, 1) != 0){
43                         sendWrongNoOfArgs((uint8_t *)"CURRENT?", 1);
44                         //sendFailure((uint8_t *)"Wrong number of arguments! CURRENT? takes 1 argument");
45                         return;
46                 }
47
48                 uint32_t u = readVolt(ADC_current[args[0]]);
49
50                 sendUARTdecn(u,3);
51
52         //DEFLECTION? - Returns deflection for specified channel
53         }else if(strncasecmp("DEFLECTION?",cmd, 11) == 0){
54                 if(readArgs((uint8_t *)cmd, 1) != 0){
55                         sendWrongNoOfArgs((uint8_t *)"DEFLECTION?", 1);
56                         //sendFailure((uint8_t *)"Wrong number of arguments! DEFLECTION? takes 1 argument");
57                         return;
58                 }
59                 uint32_t u = readVolt(ADC_deflection[args[0]]);
```

```
60                  sendUARTdecn(u,3);
61
62          //FORCE? - Returns force from specified channel
63          }else if(strncasecmp("FORCE?",cmd, 6) == 0){
64                  if(readArgs((uint8_t *)cmd, 1) != 0){
65                          sendWrongNoOfArgs((uint8_t *)"FORCE?", 1);
66                          //sendFailure((uint8_t *)"Wrong number of arguments! FORCE? takes 1 argument");
67                          return;
68                  }
69                  uint32_t u = readVolt(ADC_force[args[0]]);
70
71                  sendUARTdecn(u,3);
72
73          //CONT_CURRENT_DEFLECTION - Starts continuing current and deflection measurement
74          }else if(strncasecmp("CONT_CURRENT_DEFLECTION",cmd, 23) == 0){
75                  if(readArgs((uint8_t *)cmd, 7) != 0){
76                          sendWrongNoOfArgs((uint8_t *)"CONT_CURRENT_DEFLECTION?", 7);
77                          //sendFailure((uint8_t *)"Wrong number of arguments! CONT_CURRENT_DEFLECTION takes 7
                                arguments");
78                          return;
79                  }
80                  sendSuccess();
81                  start_cont_current_and_deflection = 1;
82                  return;
83
84
85          //CONT_CURRENT - Starts continuing current measurement
86          }else if(strncasecmp("CONT_CURRENT",cmd, 12) == 0){
87                  if(readArgs((uint8_t *)cmd, 4) != 0){
88                          sendWrongNoOfArgs((uint8_t *)"CONT_CURRENT", 4);
89                          //sendFailure((uint8_t *)"Wrong number of arguments! CONT_CURRENT takes 4 arguments");
90                          return;
91                  }
92                  start_cont_current = 1;
93                  sendSuccess();
94
95          //CAL_CURRENT
96          }else if(strncasecmp("CAL_CURRENT",cmd, 11) == 0){
97                  if(readArgs((uint8_t *)cmd, 1) != 0){
98                          sendWrongNoOfArgs((uint8_t *)"CAL_CURRENT", 1);
99                          //sendFailure((uint8_t *)"Wrong number of arguments! CAL_CURRENT takes 1 argument");
100                         return;
101                 }
102                 start_cal_current = 1;
103
104         //CONT_DEFLECTION - Starts continuing deflection measurement
105         }else if(strncasecmp("CONT_DEFLECTION",cmd, 15) == 0){
106                 if(readArgs((uint8_t *)cmd, 4) != 0){
107                         sendWrongNoOfArgs((uint8_t *)"CONT_DEFLECTION", 4);
108                         //sendFailure((uint8_t *)"Wrong number of arguments! CONT_DEFLECTION takes 4 arguments
                                ");
109                         return;
110                 }
111                 start_cont_deflection = 1;
112                 sendSuccess();
113
114         //CAL_DEFLECTION
115         }else if(strncasecmp("CAL_DEFLECTION",cmd, 14) == 0){
116                 //if(readArgs((uint8_t *)cmd, 1) != 0){
117                         //sendWrongNoOfArgs((uint8_t *)"CAL_DEFLECTION", 1);
118                         //sendFailure((uint8_t *)"Wrong number of arguments! CAL_DEFLECTION takes 1 argument");
119                 //      return;
120                 //}
121                 start_cal_deflection = 1;
122
123         //CONT_FORCE - Starts continuing deflection measurement
124         }else if(strncasecmp("CONT_FORCE",cmd, 10) == 0){
125                 if(readArgs((uint8_t *)cmd, 4) != 0){
126                         sendWrongNoOfArgs((uint8_t *)"CONT_FORCE", 4);
127                         //sendFailure((uint8_t *)"Wrong number of arguments! CONT_FORCE takes 4 arguments");
128                         return;
129                 }
130                 start_cont_force = 1;
131                 sendSuccess();
132
133         //SET_DEFLECTIONS
134         }else if(strncasecmp("SET_DEFLECTIONS",cmd, 15) == 0){
135                 if(readArgs((uint8_t *)cmd, 6) != 0){
136                         sendWrongNoOfArgs((uint8_t *)"SET_DEFLECTIONS", 6);
137                         //sendFailure((uint8_t *)"Wrong number of arguments! SET_DEFLECTIONS takes 6 arguments
                                ");
138                         return;
139                 }
140                 start_set_deflections = 1;
141
142
143         //SET_DEFLECTION
144         }else if(strncasecmp("SET_DEFLECTION",cmd, 14) == 0){
145                 if(readArgs((uint8_t *)cmd, 4) != 0){
146                         sendWrongNoOfArgs((uint8_t *)"SET_DEFLECTION", 4);
147                         //sendFailure((uint8_t *)"Wrong number of arguments! SET_DEFLECTION takes 4 arguments")
                                ;
148                         return;
149                 }
150                 start_set_deflection = 1;
151
152         //SET_CURRENT
153         }else if(strncasecmp("SET_CURRENT",cmd, 11) == 0){
154                 if(readArgs((uint8_t *)cmd, 4) != 0){
155                         sendWrongNoOfArgs((uint8_t *)"SET_CURRENT", 4);
156                         //sendFailure((uint8_t *)"Wrong number of arguments! SET_CURRENT takes 4 arguments");
157                         return;
158                 }
159                 start_set_current = 1;
160
161         //SET_FORCE
162         }else if(strncasecmp("SET_FORCE",cmd, 9) == 0){
163                 if(readArgs((uint8_t *)cmd, 4) != 0){
164                         sendWrongNoOfArgs((uint8_t *)"SET_FORCE", 4);
165                         //sendFailure((uint8_t *)"Wrong number of arguments! SET_FORCE takes 4 arguments");
166                         return;
167                 }
168
```

```
169                     start_set_force = 1;
170
171
172             //FAST LOOP
173             }else if(strncasecmp("FAST_LOOP",cmd, 9) == 0){
174                     if(readArgs((uint8_t *)cmd, 3) != 0){
175                             sendWrongNoOfArgs((uint8_t *)"FAST_LOOP", 3);
176                             //sendFailure((uint8_t *)"Wrong number of arguments! FAST_LOOP takes 3 arguments");
177                             return;
178                     }
179                     sendSuccess();
180                     start_fast = 1;
181
182             //BUSY? - Returns busy state of u-program
183             }else if(strncasecmp("BUSY?",cmd, 5) == 0){
184                     sendUARTprintln((uint8_t *)"no");
185
186             //STOP - Stops the current operation
187             }else if(strncasecmp("STOP",cmd, 4) == 0){
188                     stop = 1;
189                     //sendSuccess();
190
191             //RESET - Resets microcontroller
192             }else if(strncasecmp("RESET",cmd, 5) == 0){
193                     wdt_enable(WDTO_15MS);
194                     while(1);
195
196             }else{
197                     sendFailure((uint8_t *)"Command not recognized!");
198             }
199
200
201     }
202
203     uint8_t readArgs(uint8_t *cmd, uint8_t n){
204             uint8_t arg_n = 0;
205             uint8_t start = 1;
206             while(*cmd && (*cmd != '\n')){
207                     if(*cmd == ' '){
208                             if(start == 1) start = 0; else arg_n++;
209                             cmd++;
210                     }else{
211                             if(start == 0){
212                                     args[arg_n] = readArg(cmd);
213                                     //sendUARTdec(args[arg_n],1);
214                                     while(*cmd && (*cmd != ' ') && (*cmd != '\n')) cmd++;
215                             }else{
216                                     cmd++;
217                             }
218                     }
219
220             }
221
222             if(arg_n+1 >= n && start==0) return 0;
223             return 1;
224     }
225
226     uint32_t readArg(uint8_t *cmd){
227             uint32_t v = 0;
228             uint8_t zero = '0';
229             while(*cmd && (*cmd != '\n')){
230                     if(*cmd == ' ') return v;
231                     v *= 10;
232                     v += (*cmd - zero);
233                     cmd++;
234             }
235
236             return v;
237     }
238
239     void sendSuccess(){
240             sendUARTprintln((uint8_t *)"OK!");
241     }
242
243     void sendFailure(uint8_t *message){
244             sendUARTprint((uint8_t *)"ERR! ");
245             sendUARTprintln(message);
246     }
247
248     void sendWrongNoOfArgs(uint8_t *cmd, uint8_t noOfArgs){
249             sendUARTprint((uint8_t *)"Error: ");
250             sendUARTprint((uint8_t *)"Wrong number of arguments! ");
251             sendUARTprint(cmd);
252             sendUARTprint((uint8_t *)" takes exactly ");
253             sendUARTint(noOfArgs);
254             if(noOfArgs == 1){
255                     sendUARTprintln((uint8_t *)" argument");
256             }else{
257                     sendUARTprintln((uint8_t *)" arguments");
258             }
259     }
```

Listing A.31: *Module for command interpretation (interpreter.c)*

Listing A.32: *Header for command interpretation (interpreter.h)*

```
1   #ifndef _INTERPRETER_H
2   #define _INTERPRETER_H  1
3
4   #include <string.h>
5   #include <avr/wdt.h>
6   #include "uart.h"
7   #include "main.h"
8   #include "adc.h"
9
10  void analyzeCmd(char *cmd);
11  uint8_t readArgs(uint8_t *cmd, uint8_t n);
```

```
12 uint32_t readArg(uint8_t *cmd);
13
14 void sendSuccess();
15 void sendFailure(uint8_t *message);
16 void sendWrongNoOfArgs(uint8_t *cmd, uint8_t noOfArgs);
17
18 extern uint32_t args[];
19 extern uint8_t ADC_current[];
20 extern uint8_t ADC_deflection[];
21 extern uint8_t ADC_force[];
22 extern uint8_t PWM_ch[];
23 #endif
```

Listing A.32: *Header for command interpretation (interpreter.h)*

Listing A.33: *Module for PWM operations (pwm.c)*

```
1  #include "pwm.h"
2
3  void initPwm(){
4          DDRPWM1 |= (1<<OC0);
5          DDRPWM2 |= (1<<OC1A)|(1<<OC1B);
6
7
8          //Timer/counter 0 (8 bit)
9          // Fast PWM, inverting mode (for å få 'ren' 0), prescaling 256 (225Hz)
10         TCCR0 = (1<<WGM01)|(1<<WGM00)|(1<<COM01)|(1<<COM00)|(0<<CS02)|(1<<CS01)|(1<<CS00);
11         OCR0 = 255;
12
13         //Timer/counter 1 (16 bit)
14         // Fast PWM 8bit, inverting mode (for å få 'ren' 0), prescaling 256 (225Hz)
15         TCCR1A = (1<<COM1A1)|(1<<COM1A0)|(1<<COM1B1)|(1<<COM1B0)|(0<<FOC1A)|(0<<FOC1B)|(0<<WGM11)|(1<<WGM10);
16         TCCR1B = (0<<ICNC1)|(0<<ICES1)|(0<<WGM13)|(1<<WGM12)|(0<<CS12)|(1<<CS11)|(1<<CS10);
17
18         OCR1A = 255;
19         OCR1B = 255;
20
21 }
22
23 void setPwm(uint8_t channel, uint8_t value){
24         switch(channel){
25                 case PWM_0: OCR0 = 255-value; break;
26                 case PWM_1: OCR1A = 255-value; break;
27                 case PWM_2: OCR1B = 255-value; break;
28         }
29 }
30
31 uint8_t getPwm(uint8_t channel){
32         switch(channel){
33                 case PWM_0: return 255-(uint8_t)OCR0;
34                 case PWM_1: return 255-(uint8_t)OCR1A;
35                 case PWM_2: return 255-(uint8_t)OCR1B;
36         }
37         return 0;
38 }
```

Listing A.33: *Module for PWM operations (pwm.c)*

Listing A.34: *Header for PWM operations (pwm.h)*

```
1  #ifndef _PWM_H
2  #define _PWM_H  1
3
4  #import <avr/io.h>
5
6  #define DDRPWM1 DDRB
7  #define DDRPWM2 DDRD
8
9  #define OC0             PINB3
10 #define OC1A    PIND5
11 #define OC1B    PIND4
12
13 #define PWM_0   0
14 #define PWM_1   1
15 #define PWM_2   2
16
17 void initPwm();
18 void setPwm(uint8_t channel, uint8_t value);
19 uint8_t getPwm(uint8_t channel);
20
21 #endif
```

Listing A.34: *Header for PWM operations (pwm.h)*

Listing A.35: *Module for Timer/Counter operations (timer.c)*

```
1  #include "timer.h"
2
3  uint8_t timCnt = 0;
4
5  void initTimer(){
6          //Timer 2, CTC-mode (TOP=OCR2), prescaler 1024
7          TCCR2 = (1<<WGM21)|(0<<WGM20)|(1<<CS22)|(1<<CS21)|(1<<CS20);
8
```

```
        OCR2 = 239;

        TIMSK |= (1<<OCIE2);
}


ISR(SIG_OUTPUT_COMPARE2)
{
        timCnt = (timCnt+1) % 6;
        // timCnt == 0 => 10ms
        if(timCnt == 0) PORTB ^= 0x01;
        return;
}
```

Listing A.35: *Module for Timer/Counter operations (timer.c)*

Listing A.36: *Header for Timer/Counter operations (timer.h)*

```
#ifndef _TIMER_H
#define _TIMER_H        1

#import <avr/io.h>
#import <avr/interrupt.h>

void initTimer();

#endif
```

Listing A.36: *Header for Timer/Counter operations (timer.h)*

Listing A.37: *Module for UART operations (uart.c)*

```
#include "uart.h"

char rx_buf[rx_bufsize];                // Rx-buffer
uint8_t rx_tail = 0;                    // Rx-buffer index


void initUART(){

        UART_DDR |=  (1<<UART_TX);
        UART_DDR &= ~(1<<UART_RX);

        UBRRH = (uint8_t)(MY_UBRR >> 8);
        UBRRL = (uint8_t)MY_UBRR;
        UCSRB = (1<<RXEN)|(1<<TXEN)|(1<<RXCIE);
        UCSRC = (1<<URSEL)|(3<<UCSZ0);
}

void sendUART(uint8_t c){
        while ( !( UCSRA & (1<<UDRE)) );
        UDR = c;
}

void sendUARTprintln(uint8_t *s){
        sendUARTprint(s);
        sendUARTnewline();
}


void sendUARTprint(uint8_t *s){
        while(*s){
                sendUART(*s);
                s++;
        }
}

/*
void sendUARTbyte(uint8_t b){
        uint8_t i = 1;
        for(i = 0; i < 8; i++){
                if((b & i) == 0){
                        sendUART('0');
                }else{
                        sendUART('1');
                }
        }
}
*/

void sendUARTint(uint32_t n){
        char s[10];
        uint8_t i = 0;
        while(n > 0){
                s[i++] = '0' + (n % 10);
                n /= 10;
        }

        if(i==0){
                sendUART('0');
        }else{
                for(uint8_t j = i; j > 0; j--){
                        sendUART(s[j-1]);
                }
        }
}

void sendUARTdec(uint32_t n, uint8_t dec){
        char s[10];
        uint8_t i = 0;
```

```
69          while(n > 0){
70                  s[i++] = '0' + (n % 10);
71                  n /= 10;
72          }
73
74          if(i==0){
75                  sendUART('0');
76          }else if(i > 0 && dec >= i){
77                  sendUARTprint((uint8_t *)"0.");
78                  for(uint8_t j = dec; j > 0; j--){
79                          if(j > i){
80                                  sendUART('0');
81                          }else{
82                                  sendUART(s[j-1]);
83                          }
84                  }
85          }else{
86                  for(uint8_t j = i; j > 0; j--){
87                          if(j == dec) sendUART('.');
88                          sendUART(s[j-1]);
89                  }
90          }
91  }
92
93  void sendUARTdecn(uint32_t n, uint8_t dec){
94          sendUARTdec(n,dec);
95          sendUARTnewline();
96  }
97
98  void sendUARTnewline(){
99          sendUARTprint((uint8_t *)newline);
100 }
101
102 char readUART(){
103         while(!(UCSRA & (1<<RXC)));
104
105         return UDR;
106 }
107
108
109 ISR(SIG_UART_RECV)
110 {
111         char data = UDR;
112
113         if(rx_tail == rx_bufsize-1){
114                 //Buffer full
115                 rx_tail = 0;
116                 sendUARTprintln((uint8_t *)"Buffer full! Wait for 'OK' before sending a new command");
117         }else{
118                 rx_buf[rx_tail++] = data;
119                 if(data=='\n'){
120                         analyzeCmd(rx_buf);
121                         rx_tail = 0;
122                 }
123         }
124
125         return;
126 }
```

Listing A.37: *Module for UART operations (uart.c)*

Listing A.38: *Header for UART operations (uart.h)*

```
1   #ifndef _UART_H
2   #define _UART_H 1
3
4   #import <avr/io.h>
5   #import <util/delay.h>
6   #import <avr/interrupt.h>
7   #include <avr/eeprom.h>
8   #include "interpreter.h"
9
10  #define CLOCK                   F_CPU
11  #define BAUD                    57600
12  #define MY_UBRR                 ((CLOCK)/((BAUD)*16L)-1)
13
14  #define UART_DDR                DDRD
15  #define UART_TX                 PD1
16  #define UART_RX                 PD0
17
18  #define newline                 "\n"
19
20  void initUART();
21  void sendUART(uint8_t c);
22  void sendUARTprintln(uint8_t *s);
23  void sendUARTprint(uint8_t *s);
24  void sendUARTbyte(uint8_t b);
25  void sendUARTint(uint32_t n);
26  void sendUARTdec(uint32_t n, uint8_t dec);
27  void sendUARTdecn(uint32_t n, uint8_t dec);
28  void sendUARTnewline();
29  char readUART();
30
31
32  #define rx_bufsize      50
33
34  #endif
```

Listing A.38: *Header for UART operations (uart.h)*

Listing A.39: *Utilities module (utils.c)*

```
#include "utils.h"

void delay(uint16_t ms){
        if(ms == 0) return;
        while(ms > 16){
                _delay_ms(16);
                ms -= 16;
        }

        _delay_ms(ms);
}
```

Listing A.39: *Utilities module (utils.c)*

Listing A.40: *Header for utilities module (utils.h)*

```
#ifndef _UTILS_H
#define _UTILS_H        1

#import <util/delay.h>

void delay(uint16_t ms);


#endif
```

Listing A.40: *Header for utilities module (utils.h)*

A.5 Computer Interface Program

The below code is the implementation of the program described in section 4.5.5.

Listing A.41: *Program Loader (Program.cs)*

```
ï≫¿using System;
using System.Collections.Generic;
using System.Linq;
using System.Windows.Forms;

namespace uC_Flexinol {
    static class Program {
        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        [STAThread]
        static void Main() {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new main());
        }
    }
}
```

Listing A.41: *Program Loader (Program.cs)*

Listing A.42: *Main program (main.cs)*

```
ï≫¿using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Diagnostics;
using System.IO.Ports;
using System.Globalization;
using System.IO;

namespace uC_Flexinol {
    public partial class main : Form {

        private struct flexinolCmd {
            public string cmd;
            public string[] args;
            public string[] std;
            public string[] values;

            public override string ToString() {
                string ret = cmd;
                for (int i = 0; i < values.Length; i++) {
                    ret += " " + values[i];
                }
                return ret;
            }

        }
```

```
32
33          private List<flexinolCmd> cmdList;
34          private List<flexinolCmd> cmdCont;
35
36          private TextBox[] txtArgs;
37          private Label[] lblArgs;
38          private TextBox[] txtCArgs;
39          private Label[] lblCArgs;
40
41          private uC_comm uC;
42          private rtf_builder rtf_cmd;
43          private rtf_builder rtf_asc;
44
45          private OyvindLib.ngraph graph;
46
47          private flexinolCmd cmdStop;
48          private DateTime contStart;
49
50          private StreamWriter contLog = null;
51
52          public main() {
53              InitializeComponent();
54
55              txtArgs = new TextBox[] { txtArg1, txtArg2, txtArg3, txtArg4, txtArg5, txtArg6 ,txtArg7};
56              lblArgs = new Label[] { lblArg1, lblArg2, lblArg3, lblArg4, lblArg5, lblArg6, lblArg7};
57
58              txtCArgs = new TextBox[] { txtCArg0, txtCArg1, txtCArg2, txtCArg3, txtCArg4, txtCArg5, txtCArg6};
59              lblCArgs = new Label[] { lblCArg0, lblCArg1, lblCArg2, lblCArg3, lblCArg4, lblCArg5, lblCArg6};
60
61              addHandlers();
62
63              this.Visible = true;
64              this.WindowState = FormWindowState.Normal;
65
66
67              FolderBrowserDialog fb = new FolderBrowserDialog();
68              //fb.RootFolder = Environment.SpecialFolder.MyComputer;
69              fb.SelectedPath = Application.StartupPath + "\\cmd\\finger_demo";
70              fb.Description = "Choose setup folder";
71              fb.ShowNewFolderButton = false;
72
73              fb.ShowDialog();
74              string listPath = fb.SelectedPath;
75
76              //populateCmdList("regulator");
77              //populateCmdList("finger_demo");
78
79              try {
80                  populateCmdList(listPath);
81              } catch (Exception) {
82                  System.Windows.Forms.MessageBox.Show("Invalid path for command files!", "Error",
                        MessageBoxButtons.OK, MessageBoxIcon.Error);
83                  Environment.Exit(0);
84              }
85
86              try {
87                  uC = new uC_comm(new SerialPort());
88              } catch (Exception e) {
89                  System.Windows.Forms.MessageBox.Show(e.Message, "Error", MessageBoxButtons.OK, MessageBoxIcon.
                        Error);
90                  Environment.Exit(0);
91              }
92
93              //txtCmdOutput.Rtf += rtfColors + "tester";
94              //txtASCOutput.Rtf += rtfColors;
95              string header = "\\rtf1\\ansi\\deff0{\\colortbl;\\red0\\green0\\blue255;\\red255\\green0\\blue0;}";
96              rtf_cmd = new rtf_builder(header);
97              txtCmdOutput.Rtf = rtf_cmd.ToString();
98
99              rtf_asc = new rtf_builder(header);
100             txtASCOutput.Rtf = rtf_asc.ToString();
101
102             cmdStop = createCmd("STOP", new string[0], new string[0]);
103
104             graph = new OyvindLib.ngraph(new String[] { "0", "1", "2", "3", "4", "5" }, "Continuous reading",
                    10);
105             graph.Dock = DockStyle.Fill;
106             panPlot.Controls.Add(graph);
107
108             txtX_LostFocus(null, null);
109         }
110
111         void cmd_KeyDown(object sender, KeyEventArgs e) {
112
113             if (e.KeyCode == Keys.Return) doSend();
114         }
115
116         void populateCmdList(string folder) {
117             cmdList = new List<flexinolCmd>();
118
119             setup cmdSetup = new setup(folder + "\\setup.txt");
120
121             int i = 0;
122             try {
123                 while (true) {
124                     string[] cmd = cmdSetup.getString(i.ToString()).Split('\t');
125                     string name = cmd[0];
126                     string[] args = new string[int.Parse(cmd[1])];
127                     string[] std = new string[int.Parse(cmd[1])];
128
129                     for (int j = 0; j < args.Length; j++) {
130                         args[j] = cmd[j + 2];
131                         std[j] = cmd[j + 2 + args.Length];
132                     }
133
134                     cmdList.Add(createCmd(name, args, std));
135                     i++;
136                 }
137             } catch (Exception) { }
138
139             lstCommand.DataSource = cmdList;
140
141             setup cmdSetupCont = new setup(folder + "\\cont.txt");
```

```
142             string[] nr = cmdSetupCont.getString("nr").Split(',');
143             cmdCont = new List<flexinolCmd>();
144             for (i = 0; i < nr.Length; i++) {
145                 cmdCont.Add(cmdList[int.Parse(nr[i])]);
146             }
147 
148             lstCmdCont.DataSource = cmdCont;
149         }
150 
151         flexinolCmd createCmd(string cmd, string[] args, string[] std) {
152             flexinolCmd ret = new flexinolCmd();
153             ret.cmd = cmd;
154             ret.args = args;
155             ret.std = std;
156             ret.values = new string[args.Length];
157 
158             return ret;
159         }
160 
161         void addHandlers() {
162             lstCommand.SelectedIndexChanged += new EventHandler(cmbCommand_SelectedIndexChanged);
163             for (int i = 0; i < txtArgs.Length; i++) {
164                 txtArgs[i].KeyDown += new KeyEventHandler(cmd_KeyDown);
165             }
166             lstCommand.KeyDown += new KeyEventHandler(cmd_KeyDown);
167 
168             txtX.LostFocus += new EventHandler(txtX_LostFocus);
169         }
170 
171         void txtX_LostFocus(object sender, EventArgs e) {
172             double x = 0;
173             double.TryParse(txtX.Text, out x);
174             txtX.Text = x.ToString();
175             graph.setXLength(x);
176         }
177 
178         void cmbCommand_SelectedIndexChanged(object sender, EventArgs e) {
179             flexinolCmd cmd = (flexinolCmd)lstCommand.SelectedItem;
180             for (int i = 0; i < txtArgs.Length; i++) {
181                 if (i < cmd.args.Length) {
182                     txtArgs[i].Visible = true;
183                     txtArgs[i].Text = cmd.std[i];
184                     lblArgs[i].Text = cmd.args[i];
185                 } else {
186                     txtArgs[i].Visible = false;
187                     txtArgs[i].Text = "";
188                     lblArgs[i].Text = "";
189                 }
190             }
191         }
192 
193         private void btnSend_Click(object sender, EventArgs e) {
194             doSend();
195 
196         }
197 
198         private void doSend() {
199             timASCRead.Stop();
200             timCont.Stop();
201             flexinolCmd cmd = (flexinolCmd)lstCommand.SelectedItem;
202             sendCmd(cmd, timCmdRead, txtArgs, rtf_cmd);
203         }
204 
205         private void sendCmd(flexinolCmd cmd, Timer tim, TextBox[] txt, rtf_builder rtf) {
206             for (int i = 0; i < cmd.args.Length; i++) {
207                 cmd.values[i] = txt[i].Text;
208             }
209 
210             string data = cmd.ToString();
211             uC.send(data);
212             if (rtf != null) {
213                 rtf.appendFirst("\\b>\\cf2 " + data + "\\cf0\\b0\\line");
214                 txtCmdOutput.Rtf = rtf.ToString();
215             }
216             if (!tim.Enabled) tim.Start();
217         }
218 
219 
220         private void btnASCSend_Click(object sender, EventArgs e) {
221             timCmdRead.Stop();
222             timCont.Stop();
223             uC.send(txtASCInput.Text);
224             rtf_asc.appendFirst("\\b>\\cf2 " + txtASCInput.Text + "\\cf0\\b0\\line");
225             txtASCOutput.Rtf = rtf_asc.ToString();
226             if (!timASCRead.Enabled) timASCRead.Start();
227         }
228 
229         private void timASCRead_Tick(object sender, EventArgs e) {
230             string data = "";
231             try {
232 
233                 data = uC.read(1);
234                 rtf_asc.appendFirst("\\b>\\cf1 " + data + "\\cf0\\b0\\line");
235                 txtASCOutput.Rtf = rtf_asc.ToString();
236             } catch (Exception) {}
237 
238         }
239 
240         private void timCmdRead_Tick(object sender, EventArgs e) {
241             string data = "";
242             try {
243                 data = uC.read(1);
244                 rtf_cmd.appendFirst("\\b>\\cf1 " + data + "\\cf0\\b0\\line");
245                 txtCmdOutput.Rtf = rtf_cmd.ToString();
246             } catch (Exception) { }
247 
248         }
249 
250 
251         private void btnClear_Click(object sender, EventArgs e) {
252             rtf_cmd.clear();
253             txtCmdOutput.Text = "";
254         }
```

```
255
256         private void btnStop_Click(object sender, EventArgs e) {
257             sendCmd(cmdStop, timCmdRead, txtArgs, rtf_cmd);
258         }
259
260         private void lstCmdCont_SelectedIndexChanged(object sender, EventArgs e) {
261             flexinolCmd cmd = (flexinolCmd)lstCmdCont.SelectedItem;
262             for (int i = 0; i < txtArgs.Length; i++) {
263                 if (i < cmd.args.Length) {
264                     txtCArgs[i].Visible = true;
265                     txtCArgs[i].Text = cmd.std[i];
266                     lblCArgs[i].Text = cmd.args[i];
267                 } else {
268                     txtCArgs[i].Visible = false;
269                     txtCArgs[i].Text = "";
270                     lblCArgs[i].Text = "";
271                 }
272             }
273         }
274
275         private void btnStartCont_Click(object sender, EventArgs e) {
276             graph.removeValues();
277             contStart = DateTime.Now;
278
279             timASCRead.Stop();
280             timCmdRead.Stop();
281
282             if (chkSave.Checked) {
283                 if (contLog != null) contLog.Close();
284                 contLog = new StreamWriter(Application.StartupPath + "\\data\\" + DateTime.Now.ToString("yyyy.
                        MM.dd") + ".txt", true);
285             } else {
286                 contLog = null;
287             }
288             flexinolCmd cmd = (flexinolCmd)lstCmdCont.SelectedItem;
289             sendCmd(cmd, timCont, txtCArgs, null);
290
291         }
292
293         private void timCont_Tick(object sender, EventArgs e) {
294             string[] data = null;
295             double[] dvalues = null;
296             CultureInfo culture = CultureInfo.CreateSpecificCulture("en-US");
297             try {
298                 string tmp = uC.read(1);
299
300                 data = tmp.Split(' ');
301                 if (data.Length < 2) return;
302                 dvalues = new double[data.Length-1];
303                 for (int i = 0; i < dvalues.Length; i++) {
304                     double.TryParse(data[i],NumberStyles.Any  , culture.NumberFormat, out dvalues[i]);
305                 }
306             } catch (Exception) { }
307
308             if (dvalues != null) {
309                 graph.addValues(dvalues, DateTime.Now.Subtract(contStart).TotalSeconds);
310
311                 if (contLog != null) {
312                     contLog.Write(DateTime.Now.Subtract(contStart).TotalSeconds.ToString("0.00") + "\t");
313                     for (int i = 0; i < dvalues.Length; i++) {
314                         contLog.Write(dvalues[i].ToString() + "\t");
315                     }
316                     contLog.Write("\n");
317                 }
318             }
319
320
321         }
322
323         private void btnStopCnt_Click(object sender, EventArgs e) {
324             sendCmd(cmdStop, timCont, txtCArgs, null);
325         }
326
327
328     }
329 }
```

Listing A.42: *Main program (main.cs)*

Listing A.43: *Wrapper class for NPlot plotting library (ngraph.cs)*

```
1  using System;
2  using System.Collections.Generic;
3  using System.Text;
4  using System.Drawing;
5  using System.Windows.Forms;
6  using System.Diagnostics;
7  using NPlot;
8
9  namespace OyvindLib {
10     class ngraph : Panel{
11         public NPlot.Windows.PlotSurface2D xygraph = new NPlot.Windows.PlotSurface2D();
12         private LinePlot[] plots;
13         List<double> xvalues;
14         List<double>[] yvalues;
15
16         private string title;
17         private double xlength = 20;
18
19         public Color[] linecolors = {
20             Color.Red,
21             Color.Blue,
22             Color.Lime,
23             Color.Black,
24             Color.Purple,
25             Color.Yellow,
26             Color.Cyan,
```

```
27              Color.Gray,
28              Color.Brown
29          };
30
31          public void setXLength(double length) {
32              xlength = length;
33          }
34
35          public ngraph(string[] names, string title, double xlength) {
36              Controls.Add(xygraph);
37              this.title = title;
38              this.xlength = xlength;
39              xygraph.Location = new Point(0, 0);
40              xygraph.Size = this.Size;
41              xygraph.Dock = DockStyle.Fill;
42              //xygraph.Resize += new EventHandler(xygraph_Resize);
43              initGraph(names);
44          }
45
46          void xygraph_Resize(object sender, EventArgs e) {
47              //xygraph.Size = this.Size;
48          }
49
50
51          private void initGraph(string[] names){
52              int plotcnt = names.Length;
53              xygraph.Clear();
54
55              Font myFont = new Font("Arial", 8, FontStyle.Bold);
56
57              //Add a background grid for better chart readability.
58              Grid grid = new Grid();
59              grid.VerticalGridType = Grid.GridType.Coarse;
60              grid.HorizontalGridType = Grid.GridType.Coarse;
61              grid.MajorGridPen = new Pen(Color.LightGray, 1.0f);
62              xygraph.Add(grid);
63
64              xygraph.Title = title;
65              xygraph.TitleColor = Color.Red;
66              xygraph.Capture = false;
67              xygraph.CausesValidation = false;
68
69              xvalues = new List<double>();
70              yvalues = new List<double>[plotcnt];
71              plots = new LinePlot[plotcnt];
72
73              for (int i = 0; i < plotcnt; i++) {
74                  yvalues[i] = new List<double>();
75                  plots[i] = new LinePlot();
76                  plots[i].Color = linecolors[i % linecolors.GetUpperBound(0)];
77                  plots[i].AbscissaData = xvalues;
78                  plots[i].DataSource = yvalues[i];
79                  plots[i].Label = names[i];
80                  plots[i].Pen = new Pen(plots[i].Color, 2);
81                  xygraph.Add(plots[i]);
82              }
83
84              //Balance plot general settings.
85              xygraph.ShowCoordinates = true;
86              xygraph.YAxis1.Label = "Voltage [V]";
87              xygraph.YAxis1.LabelOffsetAbsolute = true;
88              xygraph.YAxis1.LabelOffset = 30;
89              xygraph.YAxis1.WorldMin = -0.1;
90              xygraph.YAxis1.WorldMax = 5.1;
91
92              xygraph.XAxis1.Label = "Time [s]";
93              xygraph.Padding = 15;
94
95              //Refresh surfaces.
96              xygraph.Refresh();
97
98              Legend legend = new Legend();
99              legend.AttachTo(PlotSurface2D.XAxisPosition.Top, PlotSurface2D.YAxisPosition.Left);
100             legend.VerticalEdgePlacement = Legend.Placement.Inside;
101             legend.HorizontalEdgePlacement = Legend.Placement.Inside;
102             legend.BorderStyle = LegendBase.BorderType.Line;
103             legend.XOffset = 10;
104             legend.YOffset = 10;
105             legend.Font = myFont;
106             xygraph.LegendZOrder = 1;
107             xygraph.Legend = legend;
108
109         }
110
111         public void addValues(double[] values, double d) {
112
113             //double min = values[0];
114             //double max = values[0];
115
116             while (xvalues.Count > 0) {
117                 //Debug.WriteLine(d - xvalues[0]);
118                 if (d-xvalues[0] > xlength) {
119                     xvalues.RemoveAt(0);
120                     for (int j = 0; j < yvalues.Length; j++) {
121                         yvalues[j].RemoveAt(0);
122                     }
123                 } else break;
124             }
125
126             for (int i = 0; i < yvalues.Length; i++) {
127                 xygraph.Remove(plots[i], true);
128             }
129
130             xvalues.Add(d);
131             for (int i = 0; i < yvalues.Length; i++) {
132                 if (i < values.Length) {
133                     yvalues[i].Add(values[i]);
134                     xygraph.Add(plots[i]);
135                 } else {
136                     yvalues[i].Add(0.0);
137                 }
138
139                 //min = Math.Min(min, findMin(yvalues[i]));
```

```
                //max = Math.Max(max, findMax(yvalues[i]));
            }

            //xygraph.YAxis1.WorldMin = min;
            //xygraph.YAxis1.WorldMax = max;

            doRefresh();
        }

        public void removeValues() {
            xvalues = new List<double>();

            for (int i = 0; i < yvalues.Length; i++) {
                yvalues[i] = new List<double>();
                plots[i].AbscissaData = xvalues;
                plots[i].DataSource = yvalues[i];
            }
        }

        private double findMin(List<double> v){
            double min = v[0];
            for (int i = 1; i < v.Count; i++) {
                min = Math.Min(min, v[i]);
            }

            return min;
        }
        private double findMax(List<double> v) {
            double max = v[0];
            for (int i = 1; i < v.Count; i++) {
                max = Math.Max(max, v[i]);
            }

            return max;
        }

        delegate void doRefreshCallback();
        private void doRefresh() {
            if (xygraph.InvokeRequired) {
                doRefreshCallback d = new doRefreshCallback(doRefresh);
                xygraph.Invoke(d);
            }
            else {
                xygraph.Refresh();
            }

        }

    }
}
```

Listing A.43: *Wrapper class for NPlot plotting library (ngraph.cs)*

Listing A.44: *Rich Text Format (rtf) builder class (rtf_builder.cs)*

```
ï»¿using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace uC_Flexinol {
    class rtf_builder {

        List<string> rtf_text;
        string header;
        public rtf_builder(string header) {
            rtf_text = new List<string>();
            this.header = header;
        }

        public void append(string rtf) {
            rtf_text.Add(rtf);
        }

        public void clear() {
            rtf_text.Clear();
        }

        public void appendFirst(string rtf) {
            rtf_text.Insert(0, rtf);
        }

        public override string ToString() {
            string ret = "{" + header;

            for (int i = 0; i < rtf_text.Count; i++) {
                ret += rtf_text[i];
            }

            ret += "}";
            return ret;
        }

    }
}
```

Listing A.44: *Rich Text Format (rtf) builder class (rtf_builder.cs)*

Listing A.45: *Class for reading setup files (setup.cs)*

```
ï≫¿using System;
using System.Collections.Generic;
using System.Text;
using System.IO;
using System.Diagnostics;

namespace uC_Flexinol {
    public class setup {

        private System.Collections.Hashtable values;
        private string filename;

        public setup(string filename) {
            this.filename = filename;
            readFile();
        }

        public void readFile() {
            FileStream fs = File.OpenRead(this.filename);
            StreamReader sr = new StreamReader(fs);

            values = new System.Collections.Hashtable();

            string tmp, _name, _value;
            int _ind;
            while (!sr.EndOfStream) {
                tmp = sr.ReadLine();
                try {
                    _ind = tmp.IndexOf("]=");
                    _name = tmp.Substring(1, _ind - 1);
                    _value = tmp.Substring(_ind + 2);
                    values.Add(_name, _value);
                } catch (Exception e) {
                    Debug.WriteLine("Error in setup file. Exception details: " + e.Message);
                }
            }

            sr.Close();
            fs.Close();
        }

        public string getString(string _name){
            return (string)values[_name];
        }

        public int getInteger(string _name) {
            return int.Parse(getString(_name));
        }

        public double getDouble(string _name) {
            return double.Parse(getString(_name));
        }
    }
}
```

Listing A.45: *Class for reading setup files (setup.cs)*

Listing A.46: *Class for communication with microcontroller (uC_comm.cs)*

```
ï≫¿using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.IO;
using System.IO.Ports;
using System.Diagnostics;

namespace uC_Flexinol {
    class uC_comm {

        private SerialPort rs232;

        public uC_comm(SerialPort rs232) {
            this.rs232 = rs232;

            rs232.BaudRate = 57600;
            rs232.StopBits = System.IO.Ports.StopBits.One;
            rs232.Parity = System.IO.Ports.Parity.None;
            rs232.DataBits = 8;
            rs232.NewLine = "\n";

            string[] portnames = SerialPort.GetPortNames();
            bool hit = false;
            for (int i = 0; i < portnames.Length; i++) {

                rs232.PortName = portnames[i];
                try {
                    rs232.Open();
                    string id = query("IDN?");
                    if (id.StartsWith("Flexinol uC")) {
                        Debug.WriteLine(id);
                        hit = true;
                        break;
                    }
                } catch (Exception) {}
                if (rs232.IsOpen) rs232.Close();
            }

            if (!hit) throw new Exception("Microcontroller not found! Check connection");
        }

        public void send(string data) {
            rs232.WriteLine(data);
        }

        public string read() {
            return read(1000);
```

```
        }

        public string read(int timeout) {
            rs232.ReadTimeout = timeout;
            return rs232.ReadLine();
        }

        public string query(string data) {
            send(data);
            return read();
        }

        public string query(string data, int timeout) {
            send(data);
            return read(timeout);
        }
    }
}
```

Listing A.46: *Class for communication with microcontroller (uC_comm.cs)*

A.6 Interface for Microsoft Robotics Studio

The below code is the implementation of the program described in section 4.5.6

Listing A.47: *Main Program (FingerControl.cs)*

```
//------------------------------------------------------------------------------
// <auto-generated>
//     This code was generated by a tool.
//     Runtime Version:2.0.50727.1433
//
//     Changes to this file may cause incorrect behavior and will be lost if
//     the code is regenerated.
// </auto-generated>
//------------------------------------------------------------------------------

using Microsoft.Ccr.Core;
using Microsoft.Dss.Core;
using Microsoft.Dss.Core.DsspHttp;
using Microsoft.Dss.Core.Attributes;
using Microsoft.Dss.ServiceModel.Dssp;
using Microsoft.Dss.ServiceModel.DsspServiceBase;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Xml;
using W3C.Soap;
using fingercontrol = Robotics.FingerControl;


namespace Robotics.FingerControl
{


    /// <summary>
    /// Implementation class for FingerControl
    /// </summary>
    [DisplayName("FingerControl")]
    [Description("The FingerControl Service")]
    [Contract(Contract.Identifier)]
    public class FingerControlService : DsspServiceBase
    {
        //HTML-schema
        [EmbeddedResource("Robotics.FingerControl.FingerControl.xslt")]
        string _transform = null;

        /// <summary>
        /// _state
        /// </summary>
        private FingerControlState _state = new FingerControlState();

        /// <summary>
        /// _main Port
        /// </summary>
        [ServicePort("/fingercontrol", AllowMultipleInstances=false)]
        private FingerControlOperations _mainPort = new FingerControlOperations();

        /// <summary>
        /// Default Service Constructor
        /// </summary>
        public FingerControlService(DsspServiceCreationPort creationPort) :
                base(creationPort)
        {
        }

        /// <summary>
        /// Service Start
        /// </summary>
        protected override void Start()
        {
                        base.Start();
                        // Add service specific initialization here.
        }

        /// <summary>
        /// Get Handler
        /// </summary>
        /// <param name="get"></param>
        /// <returns></returns>
        [ServiceHandler(ServiceHandlerBehavior.Concurrent)]
```

```
        public virtual IEnumerator<ITask> GetHandler(Get get)
        {
            get.ResponsePort.Post(_state);
            yield break;
        }

        /// <summary>
        /// Http Get Handler
        /// </summary>
        [ServiceHandler(ServiceHandlerBehavior.Concurrent)]
        public IEnumerator<ITask> HttpGetHandler(HttpGet httpGet) {
            //httpGet.ResponsePort.Post(new HttpResponseType(_state));
            httpGet.ResponsePort.Post(new HttpResponseType(System.Net.HttpStatusCode.OK, _state, _transform));
            yield break;
        }
    }
}
```

Listing A.47: *Main Program (FingerControl.cs)*

Listing A.48: *Type definitions (FingerControlTypes.cs)*

```
//------------------------------------------------------------------------------
// <auto-generated>
//      This code was generated by a tool.
//      Runtime Version:2.0.50727.1433
//
//      Changes to this file may cause incorrect behavior and will be lost if
//      the code is regenerated.
// </auto-generated>
//------------------------------------------------------------------------------

using Microsoft.Ccr.Core;
using Microsoft.Dss.Core.Attributes;
using Microsoft.Dss.ServiceModel.Dssp;
using Microsoft.Dss.Core.DsspHttp;
using System;
using System.Collections.Generic;
using W3C.Soap;
using fingercontrol = Robotics.FingerControl;


namespace Robotics.FingerControl
{


    /// <summary>
    /// FingerControl Contract class
    /// </summary>
    public sealed class Contract
    {

        /// <summary>
        /// The Dss Service contract
        /// </summary>
        public const String Identifier = "http://schemas.tempuri.org/2008/05/fingercontrol.html";
    }

    /// <summary>
    /// The FingerControl State
    /// </summary>
    [DataContract()]
    public class FingerControlState
    {
        // **** Datamembers ****
        private string _uCStatus = "Not connected";
        [DataMember]
        public string uCStatus { get { return _uCStatus; } set { _uCStatus = value; } }


        // **** End Datamembers ****

    }

    /// <summary>
    /// FingerControl Main Operations Port
    /// </summary>
    [ServicePort()]
    public class FingerControlOperations : PortSet<DsspDefaultLookup, DsspDefaultDrop, Get, HttpGet>
    {
    }

    /// <summary>
    /// FingerControl Get Operation
    /// </summary>
    public class Get : Get<GetRequestType, PortSet<FingerControlState, Fault>>
    {

        /// <summary>
        /// FingerControl Get Operation
        /// </summary>
        public Get()
        {
        }

        /// <summary>
        /// FingerControl Get Operation
        /// </summary>
        public Get(Microsoft.Dss.ServiceModel.Dssp.GetRequestType body) :
                base(body)
        {
        }

        /// <summary>
        /// FingerControl Get Operation
        /// </summary>
```

```
85          public Get(Microsoft.Dss.ServiceModel.Dssp.GetRequestType body, Microsoft.Ccr.Core.PortSet<
                FingerControlState,W3C.Soap.Fault> responsePort) :
86              base(body, responsePort)
87          {
88          }
89      }
90  }
```

Listing A.48: *Type definitions (FingerControlTypes.cs)*

Listing A.49: *Web interface structure (FingerControl.xslt)*

```
1   ï»¿<?xml version="1.0" encoding="UTF-8" ?>
2   <xsl:stylesheet version="1.0"
3                   xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
4                   xmlns:fc="http://schemas.tempuri.org/2008/05/fingercontrol.html">
5
6     <xsl:import href="/resources/dss/Microsoft.Dss.Runtime.Home.MasterPage.xslt" />
7     <xsl:output method="html"/>
8
9     <xsl:template match="/fc:FingerControlState">
10      <html>
11        <head>
12          <title>FingerControl</title>
13          <link rel="stylesheet" type="text/css" href="/resources/dss/Microsoft.Dss.Runtime.Home.Styles.Common.
                css" />
14        </head>
15        <body>
16          <h1>FingerControl</h1>
17          <table border="1">
18            <tr>
19                <th width="100">Status:</th>
20                <td width="200"><xsl:value-of select="fc:uCStatus"/></td>
21            </tr>
22          </table>
23          <br/>
24
25          <table border="1">
26            <tr>
27              <th>Property</th>
28              <th colspan="3"><center>Values</center></th>
29            </tr>
30            <tr>
31              <th width="100">Â </th>
32              <th width="100">Channel 0</th>
33              <th width="100">Channel 1</th>
34              <th width="100">Channel 2</th>
35            </tr>
36            <tr>
37              <th>PWM</th>
38              <td>0%</td>
39              <td>0%</td>
40              <td>0%</td>
41            </tr>
42            <tr>
43              <th>Deflection</th>
44              <td>0Â°</td>
45              <td>0Â°</td>
46              <td>0Â°</td>
47            </tr>
48            <tr>
49              <th>Force</th>
50              <td>0N</td>
51              <td>0N</td>
52              <td>0N</td>
53            </tr>
54            <tr>
55              <th>Current</th>
56              <td>0.0A</td>
57              <td>0.0A</td>
58              <td>0.0A</td>
59            </tr>
60          </table>
61          <br/>
62
63          <table border="1">
64            <tr>
65              <th>Property</th>
66              <th colspan="4">
67                <center>Values</center>
68              </th>
69            </tr>
70            <tr>
71              <th width="100">Â </th>
72              <th width="100">Channel 0</th>
73              <th width="100">Channel 1</th>
74              <th width="100">Channel 2</th>
75              <th width="50">Â </th>
76            </tr>
77            <tr>
78              <th>PWM</th>
79              <td>
80                <input type="text" name="pwm0" size="5">
81                </input>%
82              </td>
83              <td>
84                <input type="text" name="pwm1" size="5">
85                </input>%
86              </td>
87              <td>
88                <input type="text" name="pwm2" size="5">
89                </input>%
90              </td>
91              <td>
92                <center>
93                  <input type="submit" value="set"/>
```

```
                </center>
              </td>
            </tr>
            <tr>
              <th>Deflection</th>
              <td>
                <input type="text" name="def0" size="5">
                </input>Â°
              </td>
              <td>
                <input type="text" name="def1" size="5">
                </input>Â°
              </td>
              <td>
                <input type="text" name="def2" size="5">
                </input>Â°
              </td>
              <td>
                <center>
                  <input type="submit" value="set"/>
                </center>
              </td>
            </tr>
            <tr>
              <th>Force</th>
              <td>
                <input type="text" name="for0" size="5">
                </input>N
              </td>
              <td>
                <input type="text" name="for1" size="5">
                </input>N
              </td>
              <td>
                <input type="text" name="for2" size="5">
                </input>N
              </td>
              <td>
                <center>
                  <input type="submit" value="set"/>
                </center>
              </td>
            </tr>

          </table>
        </body>
      </html>
    </xsl:template>
</xsl:stylesheet>
```

Listing A.49: *Web interface structure (FingerControl.xslt)*

A.7 Matlab Scripts for Data Analysis

The below code is used to analyze and plot the data results from testing of Flexinol.

Listing A.50: *Analysis of fixated Flexinol wire (FlexinolAnalyzeFixation.m)*

```
%Script for analyzing data from Fixated wire

%data = fix_run1;
data = fix_old_5;

%Defining data columns
colDout = 3;
colAin = 2;
colPlot = [2 3];
th = 30;

%Analyzing peak count and locations
[n, locsRise, locsFall] = FlexinolPeaks(data,colDout);
cycles = n

%Finding minimum and maximum force
max_force = max(data(:,colAin))
min_force = min(data(:,colAin))

%Finding delay values
delayCurve = FlexinolDelayCurve(data(:,colAin), data(:,1), locsRise);

%Finding force difference in each contraction (max-min)
deltaCurve = FlexinolDeltaCurve(data(:,colAin), locsRise, locsFall);

close all

%Plotting delay values
axes('FontSize', 12)
plot(delayCurve, 'r', 'LineWidth', 1)
w = size(delayCurve);
axis([0 w(1) 0 3])
grid on
xlabel('Run #', 'FontSize', 14)
ylabel('Delay [s]', 'FontSize', 14)
%legend('Delay')

%Plotting force differences
figure
axes('FontSize', 12)
plot(deltaCurve, 'LineWidth', 2)
axis([0 n 0 40])
grid on
```

```
xlabel('Run #', 'FontSize', 14)
ylabel('Force [N]', 'FontSize', 14)
%legend('Force')

%Plotting 3 first cycles
figure
test = data(1:400,:);
axes('FontSize', 12)
plot(test(:,1)*0.1, test(:,2),'r','LineWidth', 2)
hold on
plot(test(:,1)*0.1, test(:,3)*8,'b','LineWidth', 2)
grid on
xlabel('Time [s]', 'FontSize', 14)
ylabel('Force [N] / Output [On/Off]', 'FontSize', 14)
legend('Force', 'Digital Output')

%return

%Plotting difference between first contraction and late contraction
%Index er tilpasset fix_run2
figure
plen = 200;
start0 = 100000-100+356;
y0a = data(start0:start0+plen,2);
y0b = data(start0:start0+plen,3)*5;
x0 = data(start0:start0+plen,1)-data(start0,1);
%start1 = 1532692;
start1 = 900000-25+175;
y1a = data(start1:start1+plen,2);
y1b = data(start1:start1+plen,3)*5;
x1 = data(start1:start1+plen,1)-data(start1,1);
axes('FontSize', 12)
plot(x0*0.1, y0a,'r','LineWidth', 2)
hold on
plot(x1*0.1, y1a,'b','LineWidth', 2)
%plot(x0*0.1, y0b,'g','LineWidth', 2)
plot(x1*0.1, y1b,'k','LineWidth', 2)
axis([0 70 0 40]);
grid on
xlabel('Time [s]', 'FontSize', 14)
ylabel('Force [N] / Output [On/Off]', 'FontSize', 14)
legend('Force (after 500 cycles)', 'Force (after 4500 cycles)', 'Digital Output')
```

Listing A.50: *Analysis of fixated Flexinol wire (FlexinolAnalyzeFixation.m)*

Listing A.51: *Analysis of a Flexinol antagonist (FlexinolAnalyzeFlexinolAntagonist.m)*

```
%Script for analyzing data from Flexinol antagonist

data = flex_run3;
%data(:,3) = data(:,3) - ((data(:,3) >10)*10);

%Selecting data columns
colDout = 5;
colAin = 3;
colPlot = [2 3 4 5];

% Plotting all values
close all
FlexinolPlot(data, colPlot,1)
legend('Force','Deflection','Dout agonist','Dout antagonist')


%Plotting selected data
figure
%n0 = 1; %run1
%l = 1000; %run1
n0 = 120; %run3
l = 230; %run3
test = data(n0:n0+l,:);
axes('FontSize', 12)
x = (test(:,1)-data(n0,1))*0.1;
plot(x, test(:,2),'r','LineWidth', 2)
hold on
plot(x, test(:,3),'b','LineWidth', 2)
plot(x, test(:,4),'g','LineWidth', 2)
plot(x, test(:,5),'k','LineWidth', 2)
grid on
%axis([0 80 0 8]);
xlabel('Time [s]', 'FontSize', 14)
ylabel('Deflection [cm] / Force [N] / Output [On/Off]', 'FontSize', 14)
legend('Force', 'Deflection', 'Dout agonist', 'Dout antagonist')


%Analyzing peak count and locations
[n, locsRise, locsFall] = FlexinolPeaks(data,colDout);
cycles = n

%Finding differences between min and max
deltaCurve = FlexinolDeltaCurve(data(:,colAin), locsRise, locsFall);

%Finding differences for antagonist wire
deltaCurve2 = FlexinolMaxCurve(data(:,2), locsRise, locsFall);

% Finding max values
max_delta = max(deltaCurve .* (deltaCurve<3.5))
min_delta = min(deltaCurve)

figure
axes('FontSize', 12)

%Plotting values
plot(deltaCurve2, 'r', 'LineWidth', 1)
hold on
plot(deltaCurve, 'b', 'LineWidth', 2)
axis([0 5000 -0.5 12]) %run3
```

```
grid on
xlabel('Cycle #')
ylabel('Deflection [cm] / Force [N]')
legend('Force', 'Deflection')
```

Listing A.51: *Analysis of a Flexinol antagonist (FlexinolAnalyzeFlexinolAntagonist.m)*

Listing A.52: *Analysis of Flexinol with heavy load (FlexinolAnalyzeHeavyLoad.m)*

```
%Script for analyzing data from heavy loaded wire

data = heavy_run5;
%data(:,3) = data(:,3) - ((data(:,3) >10)*10);

%Selecting data columns
colDout = 3;
colAin = 2;
colPlot = 2;
th = 30;

%Analyzing peak count and locations
[n, locsRise, locsFall] = FlexinolPeaks(data,colDout);
cycles = n

%Analyzing differences between min and max displacement
deltaCurve = FlexinolDeltaCurve(data(:,colAin), locsRise, locsFall);

%Analyzing max and min displacement
max_delta = max(deltaCurve .* (deltaCurve<3.5))
min_delta = min(deltaCurve)

%Analyzing delay curve
delayCurve = FlexinolDelayCurve2(data(:,colAin), data(:,1), locsRise, 50, 2);

close all
% Color collection
colors = ['r' 'g' 'b' 'k'];
axes('FontSize', 12)

%Plotting all data
for i=1:length(colPlot)
    plot(data(:,1)*0.1*(1/86400),data(:,colPlot(i)),colors(i), 'LineWidth', 1)
    hold on
end
axis([0 data(length(data(:,1)),1)*0.1*(1/86400) -0.5 5.5])
grid on
xlabel('Time [days]', 'FontSize', 14)
ylabel('Deflection [cm]', 'FontSize', 14)


%Plotting difference between max and min
figure
axes('FontSize', 12)
plot(deltaCurve, 'r', 'LineWidth', 2)
axis([0 n 0 5])
grid on
xlabel('Run #', 'FontSize', 14)
ylabel('Deflection [cm]', 'FontSize', 14)


%Plotting 2 cycles from heavy_run5
figure
test = data(42000:42300,:);
axes('FontSize', 12)
x = (test(:,1)-data(42000,1))*0.1;
plot(x, test(:,2),'r','LineWidth', 2)
hold on
plot(x, test(:,3)*0.9,'b','LineWidth', 2)
grid on
axis([0 80 0 5]);
xlabel('Time [s]', 'FontSize', 14)
ylabel('Deflection [cm] / Output [On/Off]', 'FontSize', 14)
legend('Deflection', 'Digital Output')

%Plotting delay curves
figure
axes('FontSize', 12)
plot(delayCurve, 'r', 'LineWidth', 1)
w = size(delayCurve);
axis([0 w(1) 0 4])
grid on
xlabel('Run #', 'FontSize', 14)
ylabel('Delay [s]', 'FontSize', 14)
```

Listing A.52: *Analysis of Flexinol with heavy load (FlexinolAnalyzeHeavyLoad.m)*

Listing A.53: *Analysis of single PWM control (FlexinolAnalyzePwm.m)*

```
%Script for plotting single PWM measurement

%Variable selection
%data = reg_010;
%data = reg_050;
%data = reg_100;
%data = reg_150;
data = reg_200;

%Plotting data
axes('FontSize', 12)
plot(data(:,1),(data(:,2)-data(1,2))*2.1, 'r', 'LineWidth', 1)
```

```
axis([0 10 0 4])
grid on
xlabel('Time [s]', 'FontSize', 14)
ylabel('Deflection [cm]', 'FontSize', 14)
```

Listing A.53: *Analysis of single PWM control (FlexinolAnalyzePwm.m)*

Listing A.54: *Analysis of linearity in PWM control (FlexinolAnalyzePwm_150.m)*

```
%Script for plotting series of step regulation results

%Initializing data variable
data = zeros(400,27*2);

%PWM values series 1-10
data(:,1:2) = reg_150_000(1:400,:);
data(:,3:4) = reg_150_010(1:400,:);
data(:,5:6) = reg_150_020(1:400,:);
data(:,7:8) = reg_150_030(1:400,:);
data(:,9:10) = reg_150_040(1:400,:);
data(:,11:12) = reg_150_050(1:400,:);
data(:,13:14) = reg_150_060(1:400,:);
data(:,15:16) = reg_150_070(1:400,:);
data(:,17:18) = reg_150_080(1:400,:);
data(:,19:20) = reg_150_090(1:400,:);

%PWM values series 11-20
data(:,21:22) = reg_150_100(1:400,:);
data(:,23:24) = reg_150_110(1:400,:);
data(:,25:26) = reg_150_120(1:400,:);
data(:,27:28) = reg_150_130(1:400,:);
data(:,29:30) = reg_150_140(1:400,:);
data(:,31:32) = reg_150_150(1:400,:);
data(:,33:34) = reg_150_160(1:400,:);
data(:,35:36) = reg_150_170(1:400,:);
data(:,37:38) = reg_150_180(1:400,:);
data(:,39:40) = reg_150_190(1:400,:);

%PWM values series 20-27
data(:,41:42) = reg_150_200(1:400,:);
data(:,43:44) = reg_150_210(1:400,:);
data(:,45:46) = reg_150_220(1:400,:);
data(:,47:48) = reg_150_230(1:400,:);
data(:,49:50) = reg_150_240(1:400,:);
data(:,51:52) = reg_150_250(1:400,:);
data(:,53:54) = reg_150_255(1:400,:);


%Plotting all series
close all
axes('FontSize', 12)
m = zeros(27,2);
for i=1:27
    plot(data(:,i*2-1),(data(:,i*2)-data(1,i*2))*2.1, 'r', 'LineWidth', 1)
    hold on

    m(i,2) = mean(data(150:400,i*2));
    m(i,1) = i*10 -10;
end
m(27,1) = 255;

axis([0 10 0 4])
grid on
xlabel('Time [s]', 'FontSize', 14)
ylabel('Deflection [cm]', 'FontSize', 14)

%Plotting linearity
figure
plot(m(:,1)* (100/255),(m(:,2)-m(1,2))*2.1)
axis([-5 105 0 4])
grid on
xlabel('Contraction step [%]', 'FontSize', 14)
ylabel('Deflection [cm]', 'FontSize', 14)
```

Listing A.54: *Analysis of linearity in PWM control (FlexinolAnalyzePwm_150.m)*

Listing A.55: *Analysis of Flexinol with small load (FlexinolAnalyzeSmallLoad.m)*

```
%Script for analyzing small load
data = small_run4;
%data(:,3) = data(:,3) - ((data(:,3) >10)*10);

%Selecting data columns
colDout = 3;
colAin = 2;
colPlot = 2;
th = 30;

%Analyzing peak count and locations
[n, locsRise, locsFall] = FlexinolPeaks(data,colDout);
cycles = n

%Analyzing difference between max and min displacement
deltaCurve = FlexinolDeltaCurve(data(:,colAin), locsRise, locsFall);

%Finding max and min displacement
max_delta = max(deltaCurve .* (deltaCurve<3.5))
min_delta = min(deltaCurve)

close all
```

```
colors = ['r' 'g' 'b' 'k'];
axes('FontSize', 12)

%Plotting all data
for i=1:length(colPlot)
    plot((data(:,1)-data(1,1))*0.1*(1/86400),data(:,colPlot(i)),colors(i), 'LineWidth', 1)
    hold on
end
%axis([0 data(length(data(:,1)),1)*0.1*(1/86400) 0 5])
grid on
xlabel('Time [days]', 'FontSize', 14)
ylabel('Deflection [cm]', 'FontSize', 14)

%Plotting differences
figure
axes('FontSize', 12)
plot(deltaCurve, 'r', 'LineWidth', 2)
axis([0 n 0 5])
grid on
xlabel('Run #', 'FontSize', 14)
ylabel('Deflection [cm]', 'FontSize', 14)


%Plotting 2 cycles from small_run4
figure
test = data(42050:42500,:);
axes('FontSize', 12)
x = (test(:,1)-data(42050,1))*0.1;
plot(x, test(:,2),'r','LineWidth', 2)
hold on
plot(x, test(:,3)*0.9,'b','LineWidth', 2)
grid on
axis([0 120 0 5]);
xlabel('Time [s]', 'FontSize', 14)
ylabel('Deflection [cm] / Output [On/Off]', 'FontSize', 14)
legend('Deflection', 'Digital Output')
```

Listing A.55: *Analysis of Flexinol with small load (FlexinolAnalyzeSmallLoad.m)*

Listing A.56: *Analysis of a spring antagonist (FlexinolAnalyzeSpringAntagonist.m)*

```
%Script for analyzing data from spring antagonist
data = spring_run3;
%data(:,3) = data(:,3) - ((data(:,3) >10)*10);

%Selecting data columns
colDout = 4;
colAin = 3;
colPlot = [4 3 2];

%Analyzing peak count and locations
[n, locsRise, locsFall] = FlexinolPeaks(data,colDout);
cycles = n

%Analyzing difference between max and min displacement
deltaCurve = FlexinolDeltaCurve(data(:,colAin), locsRise, locsFall);

%Analyzing difference between max and min force
deltaCurve2 = FlexinolDeltaCurve(data(:,2), locsRise, locsFall);

%Plotting all data
close all
FlexinolPlot(data, colPlot)

%Plotting differences
figure
axes('FontSize', 12)
plot(deltaCurve, 'b', 'LineWidth', 2)
grid on
xlabel('Cycle #')
ylabel('Deflection [cm] / Force [N]')
%legend('Deflection', 'Force')

%Plotting 2 contraction cycles
figure
n0 = 500;
l = 250;
test = data(n0:n0+l,:);
axes('FontSize', 12)
x = (test(:,1)-data(n0,1))*0.1;
plot(x, test(:,2),'r','LineWidth', 2)
hold on
plot(x, test(:,3),'b','LineWidth', 2)
plot(x, test(:,4),'g','LineWidth', 2)
grid on
axis([0 80 0 8]);
xlabel('Time [s]', 'FontSize', 14)
ylabel('Deflection [cm] / Force [N] / Output [On/Off]', 'FontSize', 14)
legend('Force', 'Deflection', 'Digital Output')
```

Listing A.56: *Analysis of a spring antagonist (FlexinolAnalyzeSpringAntagonist.m)*

A.7.1 Help Scripts

Listing A.57: *Script for finding all contractions (FlexinolPeaks.m)*

```
%Script for finding number of Flexinol contractions
function [ contractions, locsRise, locsFall ] = FlexinolPeaks( data, contractColumn)

    data1 = data(:,contractColumn);

    x = data1(:);
    upordown = sign(diff(x));

    RiseFlags = upordown>0;
    locsRise = find(RiseFlags);

    FallFlags = upordown<0;
    locsFall = find(FallFlags);

    contractions = sum(RiseFlags);
```

Listing A.57: *Script for finding all contractions (FlexinolPeaks.m)*

Listing A.58: *Calculation of delay for displacement (FlexinolDelayCurve.m)*

```
%Script for calculating time before a given displacment is achieved (2cm)
function delayCurve = FlexinolDelayCurve(Ain, T, locsRise)

delayCurve = zeros(length(locsRise)-1,1);

for i=1:length(locsRise)-1
    maxLoc = -1;
    for j=locsRise(i):locsRise(i)+15
        if j >= length(T)
            maxLoc = -1;
            break
        end

        if Ain(j) >= 2
            maxLoc = j-locsRise(i);
            break
        end
    end
    if maxLoc == -1
        delayCurve(i) = -1;
    else
        delayCurve(i) = (T(locsRise(i)+maxLoc) - T(locsRise(i))) * 0.1;
        if delayCurve(i) > 2, delayCurve(i) = 2; end
        %if delayCurve(i) > 2, delayCurve(i) = delayCurve(i-1); end
        %coord = [locsRise(i); locsRise(i)+maxLoc; T(locsRise(i)+maxLoc) - T(locsRise(i))]
    end
end
```

Listing A.58: *Calculation of delay for displacement (FlexinolDelayCurve.m)*

Listing A.59: *Calculation of delay for force (FlexinolDelayCurve2.m)*

```
%Script for calculating the delay before a force F is reached
function delayCurve = FlexinolDelayCurve2(Ain, T, locsRise, steps, F)

delayCurve = zeros(length(locsRise)-1,1);

for i=1:length(locsRise)-1
    maxLoc = -1;
    for j=locsRise(i):locsRise(i)+steps
        if j >= length(T)
            maxLoc = -1;
            break
        end

        if Ain(j) >= F
            maxLoc = j-locsRise(i);
            break
        end
    end
    if maxLoc == -1
        delayCurve(i) = -1;
    else
        delayCurve(i) = (T(locsRise(i)+maxLoc) - T(locsRise(i))) * 0.1;
        %if delayCurve(i) > 4, delayCurve(i) = 4; end
        if delayCurve(i) > 2.5, delayCurve(i) = delayCurve(i-1); end
        %coord = [locsRise(i); locsRise(i)+maxLoc; T(locsRise(i)+maxLoc) - T(locsRise(i))]
    end
end
```

Listing A.59: *Calculation of delay for force (FlexinolDelayCurve2.m)*

Listing A.60: *Calculation of absolute displacement or force (FlexinolDeltaCurve.m)*

```
% Script for calculating difference between maximum and minimum values in
% data
function deltaCurve = FlexinolDeltaCurve(data, locsRise, locsFall)

deltaCurve = zeros(length(locsRise)-1,1);
for i=1:length(locsRise)-1
    %deltaCurve(i) = data(locs(i)+2) - data(locs(i+1)-10);
    deltaCurve(i) = data(locsFall(i)) - data(locsRise(i));
end
```

Listing A.60: *Calculation of absolute displacement or force (FlexinolDeltaCurve.m)*

Listing A.61: *Calculation of absolute displacement or force (FlexinolMaxCurve.m)*

```
% Calculates difference between maximum and minimum values
function deltaCurve = FlexinolMaxCurve(data, locsRise, locsFall)

deltaCurve = zeros(length(locsRise)-1,1);
for i=1:length(locsRise)-1
    maxvalue = data(locsRise(i));
    for j=locsRise(i):locsFall(i)
        maxvalue = max(maxvalue,data(j));
    end
    deltaCurve(i) = maxvalue;
end
```

Listing A.61: *Calculation of absolute displacement or force (FlexinolMaxCurve.m)*

Listing A.62: *Standardscript for plotting Flexinol data (FlexinolPlot.m)*

```
%Plotting all columns from data defined in colPlot
function FlexinolPlot(data, colPlot, varargin)
if length(varargin) > 0 , w = cell2mat(varargin); else w = 2; end

colors = ['r' 'g' 'b' 'k'];
    axes('FontSize', 12)

    for i=1:length(colPlot)
        plot(data(:,1)*0.1,data(:,colPlot(i)),colors(i), 'LineWidth', w(1))
        hold on
    end
```

Listing A.62: *Standardscript for plotting Flexinol data (FlexinolPlot.m)*