COMPUTER CONTROL OF A METAL BAR BENDING PRESS

THESIS PREPARED IN PARTIAL FULFILMENT OF THE REQUIREMENTS OF THE DEGREE OF MASTER IN INDUSTRIAL ENGINEERING AT THE UNIVERSITY OF STELLBOSCH

BY

DIRK VAN DER MERWE

STUDY LEADER: DR. D.C. PAGE

DECEMBER 2000

DECLARATION

I, the undersigned, hereby declare that the work contained in this thesis is my own original work and has not previously in its entirety or in part been submitted at any university for a degree.

D.J. van der Merwe

SYNOPSIS

Mechatronics is an exciting research area that stemmed from huge progress in computer technology from the 1970s onwards. In such systems, computer controlled electronic and mechanical devices interact so intimately that it is impossible to tell where the one ends and the other begins.

A mechatronic project aimed at automated manufacturing was undertaken by the Department of Industrial Engineering at the University of Stellenbosch as part of their programme in mechatronics. ROVIC was the industrial partner in the joint venture. ROVIC manufactures a front loader that is used on tractors. The front loader fits on a tractor by means of a kit that has to be manufactured specially for each type of tractor. The design of the kit needed many welding joints, which lead to a difficult and expensive manufacturing process.

A bending press that was capable of bending pre-cut steel into a predefined shape was identified as a viable solution. The mechanical design for the project was done by ROVIC, while the University was responsible for the electronic control.

Bending is carried out by pushing a flat bar over a V-block into the bending position which can be read from a position encoder. The bending blade then bends the metal until the appropriate angle is reached. It is impossible to predict the resulting angle by means of analytical techniques. This is due to non-linear behaviour caused by spring-back.

The position of the bending blade is read with an incremental encoder. An empirical equation is used to convert the distance read by the sensor into the appropriate angle. The parameters are deduced by means of non-linear regression methods. A mean square error of 0.069 was obtained for the equation when operating in the linear area of the plastic region of bending. A software application was written to enable ROVIC to deduce the needed parameters of the equation for all types of steel.

Pulse control was used to control the bending blade. The blade moves continuously until a pulse band is reached. The blade is pulsed from there into a dead band. Consequently the accuracy of the machine is determined by the width of the dead band. The control program is capable of executing bend sequence files to enable the operator to perform a sequence of bends on a metal bar.

Safety was of great importance in the design. The control system was designed to give priority to safety signals over other control signals. Dangerous situations owing to computer failure are thus prevented.

OPSOMMING

Megatronika is 'n opwindende navorsingsarea wat ontstaan het uit groot vooruitgang in rekenaartegnologie vanaf 1970. In megatroniese sisteme werk rekenaar beheerde elektroniese en meganiese toestelle so nou saam dat dit onmoontlik is om agter te kom waar die een begin en die ander eindig.

'n Megatronika projek gerig op vervaardigings outomatisasie is onlangs onderneem deur die Departement van Bedryfsingenieurswese van die Universiteit van Stellenbosch as deel van hul program in megatronika. ROVIC was die industriële vennoot in die projek. ROVIC vervaardig implemente vir gebruik op plase. Een van hul produkte is 'n laaigraaf wat met 'n hegstuk op 'n trekker pas. Die hegstuk moet spesifiek vir elke tipe trekker ontwerp word wat gelei het tot 'n ontwerp met baie sweislaste wat 'n tydrowende vervaardigingsproses vereis.

'n Staal buigmasjien wat vooraf gesnyde staal in 'n spesifieke vorm kan buig sou hierdie probleem kon oorbrug. Die meganiese ontwerp vir die masjien is gedoen deur ROVIC terwyl die Universiteit verantwoordelik was vir die elektroniese beheer.

Staal word gebruik deur 'n staalbalk te skuif oor die buigbed tot in die buigposisie wat afgelees word vanaf 'n posisie enkodeerder. Die buiglem buig dan die staal in die benodigde hoek. Geen analitiese metodes bestaan om die buighoek te voorspel nie weens terugspringing.

Die posisie van die buiglem word ingelees met behulp van 'n inkrementele enkodeerder. 'n Empiriese vergelyking word gebruik om die afstand beweeg om te skakel in 'n buighoek. Die verskillende parameters word afgelei deur middel van regressie metodes. Die gemiddelde foute kwadraat was 0.069 wat getoets is vir slegs die lineêre area van die plastiese gebied. 'n Program is geskryf wat ROVIC instaat stel om die benodigde parameters self af te lei.

Puls beheer is gebruik op die beheer van die buiglem. Die lem beweeg kontinu tot binne die puls band en word daarvandaan gepuls tot by die stopband. Die grootte van die stopband bepaal dus die akkuraatheid van die masjien. Die beheerprogram kan ook buig sekwensie lêers uitvoer wat die operateur instaat stel om 'n reeks van buigings op 'n staal balk uit te voer.

Veiligheid het deurgaans 'n belangrike rol gespeel in die ontwerp van die masjien. Die stelsel is ontwerp om prioriteit te gee aan veiligheidseine bo enige ander beheersein. Gevaarlike situasies wat mag ontstaan as die rekenaar sou faal, word dus voorkom.

ACKNOWLEDGEMENTS

I would like to thank the following persons for their efforts in making this project possible:

Dr. D.C. Page for his help and guidance during the project. He was very helpful when advice were needed.

ROVIC for the opportunity to conduct research for a project in industry. Especially the factory manager and the R&D manager were very helpful.

INDEX

LIST OF FIGURES

LIST OF TABLES

INTRODUCTION

Living in the twentieth century, most people are acquainted with electronic devices. Today, many vital processes depend on such devices. Electronics play an increasingly important role in our everyday lives. Since the sixties the electronic industry has been in a worldwide boom. Countries from especially the new world, all strive to be market leaders in these industries – an area dominated for very long by the old world and Japan.

Open economies and good transportation enable companies to market and sell products in foreign countries with little extra cost. Companies are forced to be more competitive and to be first with new products on the market. A number of design and production topologies exist such as mechatronics that aids companies in achieving this.

A project in mechatronics was launched to develop a flat bar bending machine. This project was third in a series of three such development projects. During the previous efforts, two control systems were developed for a pipe-bending machine (Coetzee, 1997; Jürgens, 1998). None of these projects proved to have stable designs.

This project was initiated by ROVIC, a company specialising in the manufacturing of farm implements.

One of the products manufactured by ROVIC is a front-end loader that is used on tractors. The front-end loader fits on a tractor by means of a kit. Each type of tractor requires its own unique kit owing to the difference in design dimensions of tractors. When new tractor models appear on the market, new kits have to be designed. A current design was usually changed to fit the new dimensions, which often lead to the edition of welding joints on the design – an expensive and time consuming task.

ROVIC decided on a new approach. Flat bars are bent into the appropriate form to minimise welding joints in their designs. Therefore a bending press capable of bending the flat bars was developed. The Department of Industrial Engineering at the University of Stellenbosch was responsible for the design of the control system for the bending press.

RESEARCH OBJECTIVES

The following objectives were defined at the start of the project:

1) A control system for a hydraulic flat bar bending press must be developed that is usable and stable in an industrial environment. The project should consist of the following components, which are to be developed separately:

   - Electronic hardware for the control system.
   - Control program with a graphical user interface.
   - Configuration program to configure machine.
   - Editor for compiling bend sequence programs.
   - Software that process empirical bending data for different types of flat bars to calculate parameters for a bending equation used by the software.

2) This project must adhere to the design principles of mechatronic projects.

STRUCTURE OF THESIS

The thesis is divided into two parts:

1) Literature study on mechatronics.

2) Description of the design project.

Part 1

Chapter 1: The properties of mechatronic systems are discussed. A broad perspective is given on the concept of mechatronics.

Chapter 2: A detailed discussion is given on mechatronics systems. Four types of mechatronic components were identified and reviewed.

Chapter 3: The design process of mechatronic systems is discussed, showing the relationship between concurrent engineering and mechatronics.

Part 2

Chapter 4: A brief background and overview to the project is given.

Chapter 5: The software developed is discussed in this chapter. The discussion is divided into the following parts:

- Control software of the bending machine.
- Program for setting control options.
- Editor for compiling bend sequence programs.
- Regression analyser.

Chapter 6: This chapter focus on the electronic hardware developed. The following hardware was developed and is discussed separately:

- Counter circuitry.
- Connection circuitry.
- Driver circuitry.
- Peripheral devices.

Chapter 7: The results obtained during the project are discussed.

Chapter 8: This chapter contains concluding remarks for this project and recommendations for future projects.

Appendixes

Appendix A: The calculations and necessary equations are given for the different components of the design.

Appendix B: The rationale for the different design decisions is discussed in this appendix.

Appendix C: The correct calibration of the bending press is important. The procedure for calibrating the machine and the derivation of the bending equation parameters is treated.

Appendix D: The safety procedures for operating the bending press are discussed in this appendix.

Appendix E: This appendix contains a detailed description of the control program. The means of realising the different operation functions is discussed.

Appendix F: PC boards were manufactured as part of the project. The layout of the different PC boards is given in their correct dimensions.

Appendix G: This appendix elaborates on the discussion of the bending press construction given in chapter 4.

Appendix H: Different methods are examined for doing regression on the empirical data obtained from measurements.

Appendix I: A number of metal flat bar types were investigated during this project. The regression results for the measured observations are shown.

Appendix J: The algorithm used in the regression analyser is discussed in this appendix. Both the algorithm and the program code are shown.

PART 1

MECHATRONICS

CHAPTER 1

INTRODUCTION TO MECHATRONICS

Most textbooks regard Mechatronics from two viewpoints:

1) The design philosophy of mechatronic systems.

2) The properties of mechatronic systems and mechatronic behaviour.

Both these viewpoints will be discussed in this thesis, but with greater emphasis on the design philosophy of Mechatronics.

A theme that will be emphasised throughout this section is the importance to the mechatronic engineer of having a broad understanding of different fields of engineering. It will enable him to unite a number of engineering disciplines on a development team because of his understanding of the system requirements and needs of all parties involved. The design process is thus accelerated, which leads to a much more efficient solution to design problems.

The purpose of mechatronics can be summarised in the following objectives (Schweitzer, 1996:120):

1) Improvement of manufacturing machinery and processes, making them more effective and cost efficient.

2) Development of novel design solutions for new types of machinery.

3) Development of new consumer products.

4) Deploying of new techniques for working in interdisciplinary design teams.

5) Development of new concepts in the design of intelligent, human-orientated machines.

1.1 HISTORY OF MECHATRONICS

The advent of the electronics age made it possible for mankind to accurately control large machinery for precise manufacturing. Scientists from the Massachusetts Institute of Technology in the USA were the first to develop the NC machine in 1952 (Tan, 1998). NC machines were one of the first major applications that stressed the need to combine electronic and mechanical engineering in mechatronics.

The term "mechatronics" was used during 1969 by a Japanese electrical engineer, Ko Kikuchi, of the Yaskava Electric Company (Comerford, 1994). He, interestingly, registered trademark rights to the term, but relinquished his rights after the popularisation of the term (Kyura, 1996a:10).

The NC machine was one of the first mechatronic devices. Japan played an important role in the development of the NC machines in Asia. Yaskawa produced the first microprocessor-controlled NC machine in Japan in 1974 (Tan, 1998; Kyura, 1996b). The USA was the catalyst for the manufacturing of NC machines, but Japan, with its philosophy of rapid product development, was the first to popularise the machine (Acar, 1996).

Robotics was the second field of study that had a major impact on mechatronics and it was again Japan that played an important role in this regard.

While Asian countries were quick to realise the benefits of mechatronics (the Tianjin University in China had a mechatronics curriculum in 1977 (Tan, 1998)), western countries took a long time to start practising mechatronics. It was not until the late 80s that they fully grasped the advantage of and started to utilise mechatronic systems in their designs.

At the dawn of mechatronics in the early 70s, it was regarded as only the combination of mechanics and electronics in the same engineering project. Information technology was later added during the 80s. It was now possible for machines to give information on their operation, quantities produced and other product information – all stored in a single database. The development of better computers and communication systems via networks made computer-control over networks possible, introducing computer controlled networking mechatronics in the 90s (Harashima, 1996).

1.2 DEFINITION OF MECHATRONICS

The word "mechatronics" was initially described as follows in Yaskawa's trademark application documents (Kyura, 1996a):

> *"The word, mechatronics, is composed of 'mecha' from mechanism and 'tronics' from electronics. In other words, technologies and developed products will be incorporating electronics more and more into mechanisms, intimately and organically, and making it impossible to tell where one ends and the other begins."*

This definition refers specifically to the way components of mechatronic systems function. Kyoro and Oho showed by means of a number of examples in their article on the history of mechatronics that it is only mechatronics when mechanical and electronic engineering not only cannot be told apart, but cannot function without one another (Kyura, 1996a). Examples of such products are robots and microwave ovens.

The meaning of mechatronics has changed a great deal since then. Today researchers regard the entire design process as part of mechatronics. A technical committee, the Industrial Research and Development Advisory Committee (IRDAC), introduced the following definition for mechatronics (Van Brussel, 1996; Van Amerongen, 1996):

> *"The term mechatronics refers to a synergistic combination of precision engineering, electronic control and systems thinking in the design of products and manufacturing processes. It is an interdisciplinary subject that both draws on the constituent disciplines and includes subjects not normally associated with one of the above."*

This definition by IRDAC gives greater insight into the way mechatronic products are designed and built. In simpler terms, the definition states that engineers from different backgrounds, usually electronic engineers (*electronic control*) and mechanical engineers (*precision engineering* or *precision mechanical engineering*), should combine their skills by forming design teams. This forms the execution layer of concurrent engineering as shown in chapter 3.1.

Auslander regarded the definition of IRDAC as too narrow. He defined mechatronics as follows (Auslander, 1996):

> *"Mechatronics is the application of complex decision making to the operation of physical systems."*

"Application" in Auslander's definition refers to the practitioners of mechatronics who must have the required skills to practise mechatronics. *"Physical systems"* refers to any mechanical or "touchable" system, for example a robot actuator or a hydraulic valve. The phrase "*complex decision making*" was specifically chosen by him to describe the control system, because most of the control is done by computer hardware and software today.

Buur introduced a fourth definition (Van Amerongen, 1996):

> *"Mechatronics is a technology which combines mechanics with electronics and information technology to form both functional interaction and spatial integration in components, modules, products and systems."*

This definition is fairly straightforward. Unlike all the previous definitions, it also adds the field of information technology. All these disciplines should function together in the design of both the product and the production thereof.

Van Brussel formulated a fifth significant definition (Van Brussel, 1996):

> *"Mechatronics encompasses the knowledge base and the technologies required for the flexible generation of controlled motion."*

Unlike Auslander, Van Brussel regards the IRDAC definition as too general in certain cases. He states that mechatronics is mainly concerned with using knowledge and technologies together to build better control systems in shorter time (a concurrent engineering concept). His definition is narrower, because it ignores the design of physical properties such as the colour of the product. According to this definition, mechatronics is only concerned with processes that help with the control of motion functions on the product.

The fact that five totally different definitions were given confirms the statement of Tan *et al.* None of the existing definitions define all aspects of mechatronics (especially since it is still an evolving technology (Tan, 1998)).

The following definition for mechatronics is an updated version of the definition given by Yaskava and will be used in this document as the definition of a mechatronic system:

Mechatronic systems have computer controlled electronic and mechanical devices, which interact so intimately that it is impossible to tell where the one ends and the other begins.

CHAPTER 2

MECHATRONIC PRODUCTS

The diagram in Figure 1 gives an introductory view of the relationship between the different

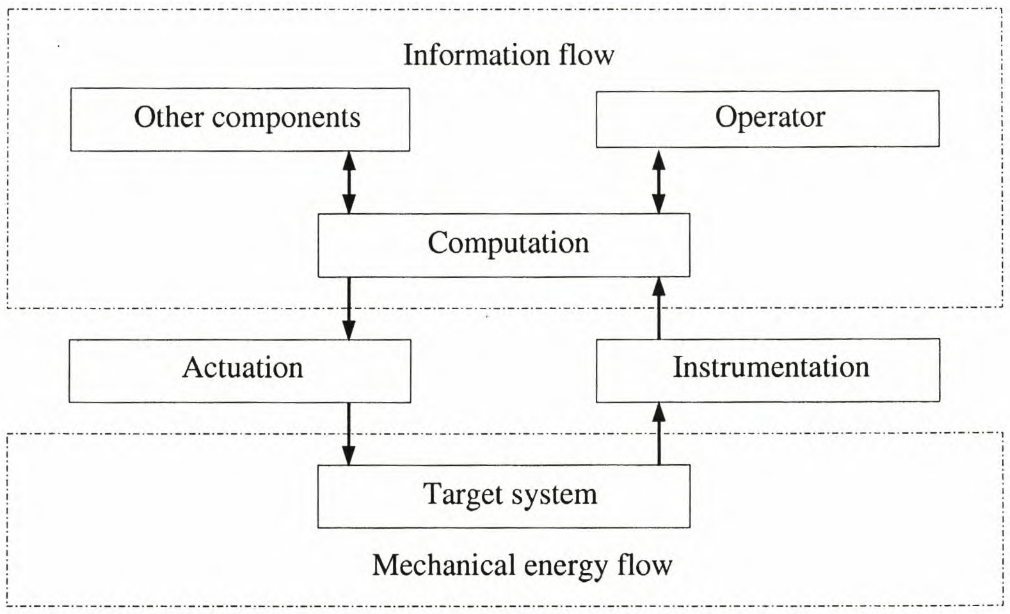


Figure 1 Diagrammatic representation of a mechatronic product (Auslander, 1995; Iserman, 1996)

components of a mechatronic product. It may seem trivial, but it is important to isolate the different parts of a mechatronic system, as will be shown later in this chapter.

"Computation" in Figure 1 usually consists of a computer or microprocessor(s). The control software is part of this building block.

Electronic components such as logic gates and operational amplifiers cannot form the total computational capabilities of the mechatronic system. The computational part has to have decision-making capabilities. The question of when the control part of a system is advanced enough to have decision-making capabilities, remains open to debate.

Table 1 The four mechatronic classes as defined by the JSPMI (Kyura, 1996)

Class	Description
Class 1	Traditionally wholly mechanical products in which the primary function is enhanced by the introduction of electronic technology.
Class 2	Traditionally wholly mechanical products, which have retained their external configuration and primary functions, but have changed their internal configuration with the introduction of electronic technology.
Class 3	Traditionally wholly mechanical products, which have retained their primary function only.
Class 4	Any other products that incorporate mechanical and electronics technology.

For the purpose of this thesis, any electronic device that is programmable or capable of handling states[1] is regarded as having decision-making capabilities. This includes all electronic components such as FPGAs, DSPs, PLCs, microprocessors and PCs.

"Actuation" in Figure 1 refers to the capabilities of the system to cause movement of some sort. This disqualifies a Class 3 mechatronic product as defined by JSPMI[2] as a mechatronic product as listed in Table 1. Products included in these classes are push-button telephones and digital watches.

The *"Target system"* in Figure 1 is the system on which the work is done, for example the metal flat bars that are bent by a mechatronic machine in this thesis.

The system has to have some form of feedback to ensure correct movement of the actuators. In Figure 1 the *"Instrumentation"* block performs this task by obtaining information via sensors from various points on the system. This enables the "C*omputation"*-block to control the whole system efficiently with the correct information.

Iserman used a broader qualification system by regarding the *computation* and other decision-making components as the *information flow* part of the system and the *target* as part of *"mechanical energy flow"* (Iserman, 1996a).

[1] Compare with state machines in (Wakerley, 1990:349)

[2] The Japan Society for the Promotion of Machine Industry

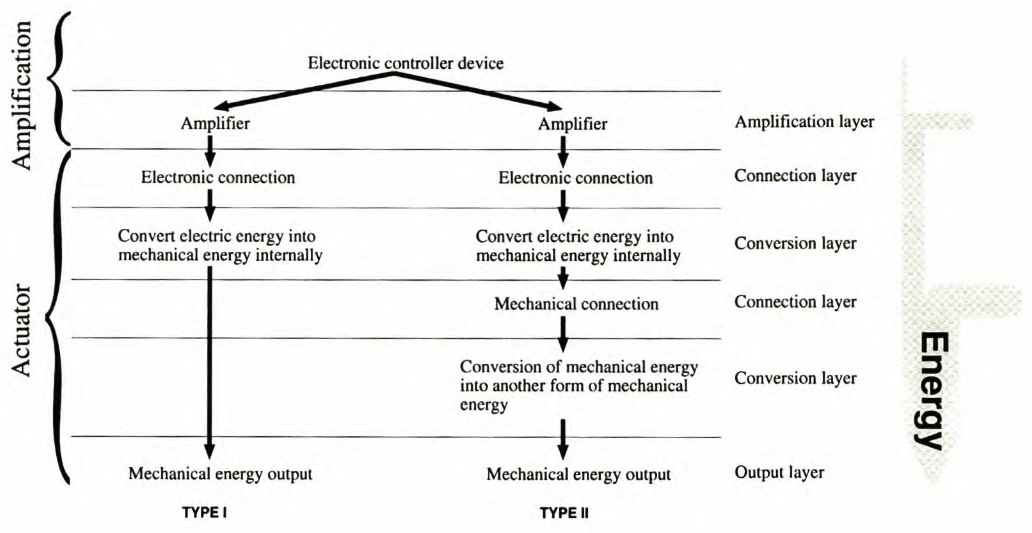


Figure 2 Diagrammatic representation of an actuator

Lastly, it is important to differentiate between mechatronic products and mechatronic components. A mechatronic product is the total system as shown in Figure 1. Mechatronic components are all the parts that make a mechatronic product, for example a computer, hydraulic valves or a position sensor.

2.1 ACTUATORS

Actuator technologies will be discussed only briefly in this section. Kamm defined actuators as devices that use only their control signal as a power source (Kamm, 1996).

Actuators usually convert energy from one form to another and can be divided into different types as shown in Table 2 (Bolton, 1995; Thornley, 1992). These actuating devices are described in detail in the literature.

Piezo-electric actuators are an exciting new development in actuator technology and are used in applications where very high accuracy is needed. Actuators with movement accuracy of as small as 5nm at speeds of up to 150mm/sec are manufactured commercially today (Anorad Corp.,1996).

Table 2 Different types of actuators used in mechatronics

Type	Typical examples
Pneumatic and hydraulic	Control valves Hydraulic and pneumatic cylinders
Mechanical and cams	Belts and chains Gears Ratchets
Electrical	Solenoids Motors Electrical switches
Piezo-electric	Vibromotors Unlimited displacement actuators

An important requirement for actuators to be suitable for mechatronics is that they have to be either electronically controlled or controlled via other electronically controlled actuating devices.

A general diagrammatic layout for mechatronic actuators is shown in Figure 2.

Two types of actuators are shown. The first type converts electrical energy into mechanical energy. Solenoids are typical examples of such actuating devices. The second type converts electrical energy into mechanical energy and then into another form of mechanical energy. A typical example of such a system is a solenoid that controls a hydraulic valve that controls the movement of a hydraulic cylinder. Other configurations are also possible, for example electric-electric-mechanical or electric-electric-mechanical-mechanical set-ups.

Later in this chapter it will be shown that mechatronic systems always have an electronic and amplification layer preceding actuating devices.

2.2 SENSORS

Sensors are very important to mechatronic systems. Sensors are currently not only used to control the system within predefined limits, but they also ensure that the necessary information is available for optimum control of the system. Sensors also improve system reliability (Lou, 1996).

Manufacturing, which is one of the major application areas of mechatronics, is also one of the major application areas of sensor technology. This underlines the importance of sensors to mechatronics (Lou, 1996).

Literature differs much on the definition of a sensor and its role in mechatronic systems (Coetzee, 1997:34; Kamm, 1996). It is often called a transducer (*trans* – Latin for across, *duce* – Latin for lead). The definition of sensors is very philosophical. Only the functioning of sensors in a mechatronic environment is therefore discussed here.

There exist two types of sensors:

1) True-false sensors

2) Continuum sensors

   a) Analogue sensors

   b) Digital sensors

Limit switches and proximity switches are examples of true-false sensors. These sensors do not give an output in proportion to the measured physical quantity. They only give an *on* signal when a specified condition is met. Very little information is therefore embedded in signals from these sensors.

Figure 3(a) shows the diagrammatic representation of these types of switches.

Continuum sensors give much more information to the controller. Their output signal is directly proportional to the size of the physical quantity measured.

In Figure 3(b) it is shown that in analogue sensors, the physical quantity is converted into a small electrical signal. Electrical signals are usually measured in terms of resistance, capacitance or inductance. This is transformed to voltage by means of devices such as the Wheatstone bridge. The resulting voltage is normally very small and must be amplified by an amplifier.

Filters are often used to eliminate noise from the amplified voltage.

The signal usually goes from the amplifier or filter into either a comparator or an analogue-to-digital converter.

Comparators send an *on* signal to the controller whenever the signal is above a specified level. Analogue-to-digital converters convert the amplified voltage into digital values that can be read by the controller. It is necessary to convert signals from the sensor into some form that can be read by microprocessors, because all mechatronic systems have to have some kind of microprocessor. Microprocessors only work with digital values.

The output from digital sensors is usually suitable to connect directly to microprocessors or via minor electronic circuitry.

The most important types of sensors used in mechatronics are listed in Table 3.

Design issues for mechatronics regarding sensors include the following (Iserman, 1996a):

- Integration with the physical process
- Dynamics of the physical process
- Resolution needed
- Mechanical and terminal robustness
- Wear resistance
- Touchless transfer
- Miniturisation
- Easy transfer to digital processing

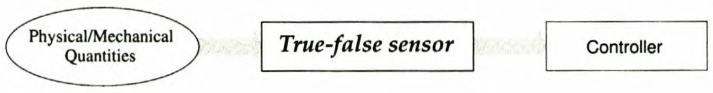
Sensors can be integrated in mechatronic systems in different ways. Rushforth *et al.* defined the following levels of integration (Rushforth, 1992):

The tangible level has sensors that are mounted on the machine to do measurements.

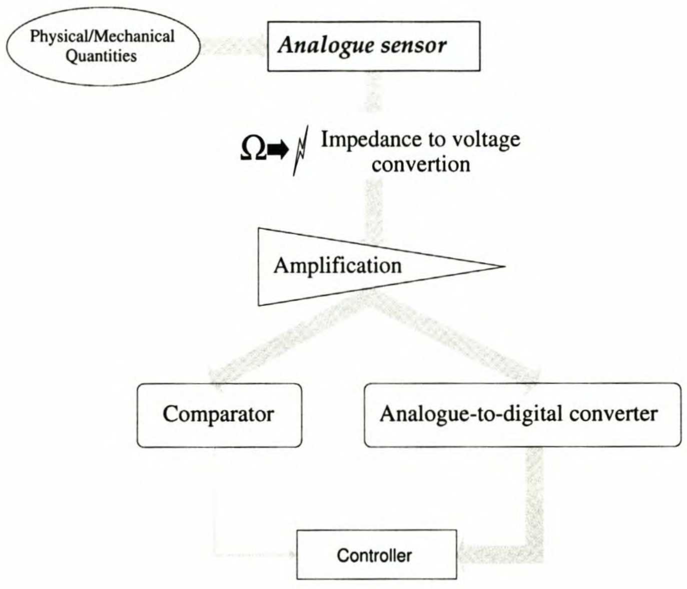
The intangible level has sensors that are combined with a data-processing unit. This unit is usually not a physical part of the sensor but is part of the control unit of the mechatronic system.

Integration between sensors is found where information is needed from more than one sensor to provide meaningful data, for example two cameras that are required for 3D vision.

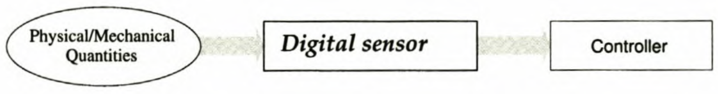
The control level combines control locally with an actuator. Examples of this configuration



(a) Binary sensors


Physical/Mechanical Quantities
Analogue sensor
Ω➡
Impedance to voltage convertion
Amplification
Comparator
Analogue-to-digital converter
Controller

(b) Analogue sensors



(c) Digital sensors

Figure 3 Diagrammatic representations of sensors

include servomotors that are equipped with encoders.

Machine/process level configurations are found where a single controller that is responsible for the control of all the actuators interprets all the signals from all the sensors.

The production planning and monitoring level has sensors that enable the whole mechatronic system to make decisions on the scheduling of production and to store data of the products in a database.

The importance of sensors for mechatronic systems should be clear from this discussion. The improvement in sensor technology should lead to greater utilisation of sensors for process control in modern factories.

2.3 CONTROL SYSTEM

2.3.1 Control system models

The control system is the most important part of a mechatronic system. There are a number

Table 3 Typical sensors used in mechatronics (Bolton, 1995; Monkman, 1992)

Displacement	Potentiometer Strain-gauge element Capacitive element Inductive elements Optical encoders
Velocity	Tachogenerator Incremental encoders
Force	Strain-gauge load cell
Liquid flow	Orifice plate Turbine meter
Liquid level	Floats Differential pressure
Temperature	Bimetallic strips Resistance temperature detectors Thermistors Thermocouples Tactile sensors
Proximity	Magnetic reed switch Inductive proximity switch Micro-switch

of models for control systems. All these models give different perspectives on control systems. The different models should therefore be studied to obtain a better understanding of how control systems operate in mechatronic systems.

During the discussion of control systems, it should be borne in mind that all controller systems in mechatronic systems are microprocessor driven in one way or another.

The explanation by Auslander gives a good overview of the underlying idea of control systems. He divided control systems into two parts, hardware and software (Auslander, 1996).

Hardware has tasks that must be performed simultaneously. Examples of such tasks are robot arms that turn simultaneously at more than one joint. Each task can be divided in states that must be executed sequentially. Hardware control thus consists of *tasks* and *states*.

Software is divided into *processes* and *threads*.

Processes are pieces of program code that are not at the same memory address. Threads are pieces of program code that are located at the same address in memory, but are executed asynchronously. A process contains one or more threads and a thread contains one or more hardware tasks.

This breakdown may seem theoretical, but it is useful when complex work is done.

Figure 4 shows the diagrammatic representation of a controller. This is a combination of the models postulated by Iserman and Kyura *et al.* (Iserman, 1996a; Kyura, 1996). It is important to keep the description by Auslander in mind while viewing the model.

This model divides the controller into different layers. These layers should not be regarded as a definite separation between the different parts of the controller, because most functions of the controller are part of the same module. The layer topology is used only to give a better understanding of all the controller functions and the interrelation between them.

Physical layer

This is the bottom layer. It is responsible for all the physical processes. This layer is not really part of the controller, but is shown to indicate where the actuators and sensors connect to the controller.

Conversion layer

The conversion layer also is not part of the controller. The signal from the controller is converted to the correct medium to activate the actuators. The signals from the sensors are conditioned in this layer in a format that the controller understands. Refer to section 2.1and

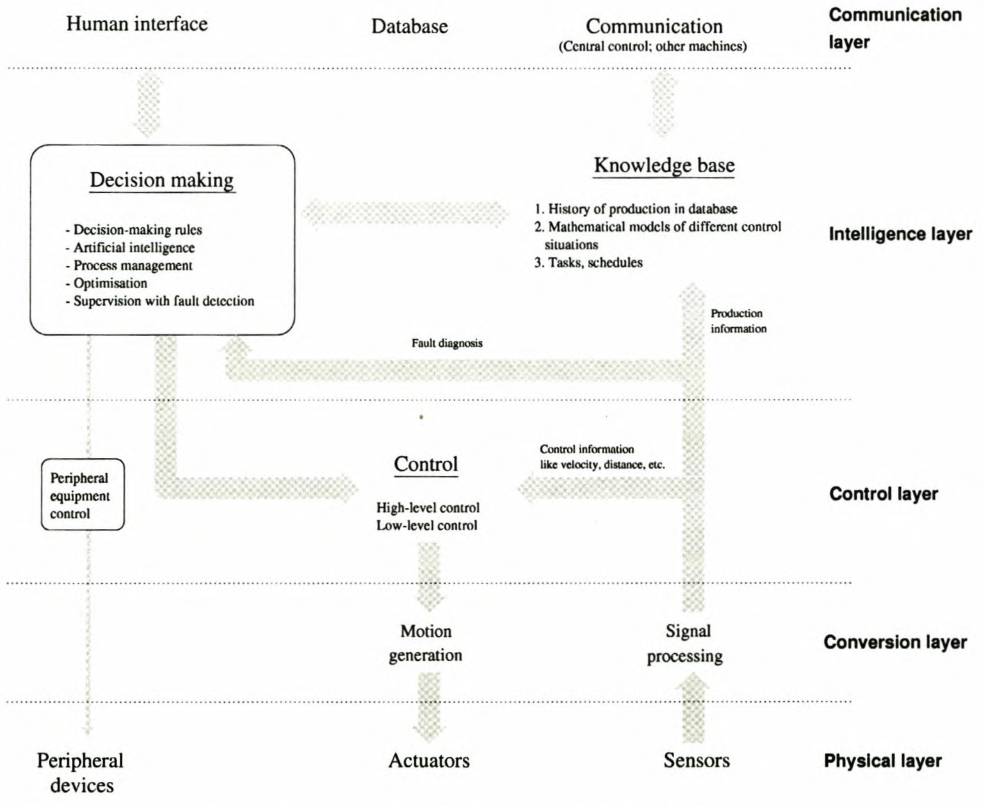


Figure 4 Diagrammatic representation of the controller

section 2.2 for a better explanation.

Control layer

This layer is responsible for the generation of all the control functions. The complexity of the control may differ according to the need of the application.

Low-level control usually takes the form of an analogue controller that enforces damping, stabilises the system or compensates for non-linearities such as friction (Iserman, 1996a)

High-level control makes use of more advanced concepts such as parameter and state estimation. The whole process is controlled with high-level compensation. PD, PI and PID controllers are typical building blocks of high-level control systems. High-level control is usually done in two ways:

1) The processes are controlled by hardware. The control function is generated by hardware components, while a microprocessor or computer generates the reference signal to the controller.

2) The control algorithms are built into software.

The use of control algorithms in software gives much more versatility to the control system than do hardware solutions.

A number of control topologies exist (Kyura, 1996a; Youcef-Tourmi, 1996; Hewit, 1996a):

Linear constant parameter control is used in applications where all the control parameters stay constant. This is the most basic form of control and typical examples are applications that use PI, PD and PID control.

Linear time-varying control is still linear, but the values of the parameters can change over time or as different circumstances arise. State space techniques are often used to analyse these types of control systems (Ogata, 1990:736).

Non-linear control is used in applications where at least one of the components of the control system is non-linear. The motion of underwater vehicles, leakage in hydraulic valves, friction, etc. are all examples of systems that have to be modelled with non-linear control techniques (Slotine 1991:4).

Chaos control (chaos does not refer to random events in this case) is found in systems that are very sensitive to initial conditions. The output therefore cannot be predicted – not even in the long run. This is mostly found in systems that are very non-linear.

Stochastic system control is found in systems with random motion. Both the system model and the input are uncertain. Time variation is therefore only statistically predicted (Slotine 1991:4).

H_∞ control is a design methodology aimed specifically at the modelling of design errors. It is useful in applications with so-called unstructured uncertainty and is normally used to obtain robust control (Grimble, 1994:7)

Robust control takes uncertainty into account in the design of a controller. Such a controller should be insensitive to parameter variations and disturbances. There are a number of robust control techniques such as H_∞ and linear quadratic Gaussian control (LQG). LQG is a very popular design method and is not only relatively simple, but is also one of the most reliable methods to design robust controllers (Grimble, 1994:105).

Adaptive control has the same purpose as robust control techniques, but instead of being insensitive to parameter variation, the controller attempts to predict the behaviour of the controlled object (an actuator for example). It then adapts to this prediction by using new device control parameters (Kanjilal, 1995:1)

Neural networks is similar to adaptive control, but they make use of a "learning function" where the control algorithm "learns" more about the controlled object as time passes. The controller then adapts more efficiently to any changes in the control parameters (Brown, 1994:17)

RMRC (Resolved motion rate control) is used to obtain co-ordination between the links of a robotic arm to move the end effector along any predicted route to a given point in space. Although these types of control were defined initially for robotics, they are currently also being applied in mechatronic systems (Hewit, 1996b).

Recent advances in control topologies emphasise the need for a decision component embedded in the control system. A typical example of such a system is an actuator with active force control (AFC) where adaptive control is used. The control system makes use of an observer to do indirect measurements of both external and internal disturbances. Wind is a typical example of external disturbance, while friction usually is due to internal disturbances. A feed forward system is used to cancel the effect of these disturbances on the actuator (Hewit, 1996b). As will be shown in the next discussion on the intelligent layer, the knowledge base is needed to store the values of the model parameters of the control system from where the controller can obtain correct values for different scenarios.

Intelligence layer

The intelligence layer is a very important part of advanced mechatronic systems. It consists of two parts:

1) Decision making

2) Knowledge base

Decision making usually occurs in accordance with predefined rules. These rules usually take the form of binary decision in elementary systems. More complex rules can be defined as the complexity of the mechatronic system increases. Process management and the optimisation of all the actions and processes executed by the machine are thus the major function of a mechatronic system. Artificial intelligence is used in very complex mechatronic systems where all the conditions cannot be defined beforehand.

Fault detection is also a very important part of decision making. Careful consideration must be given to the design of fault detection circuitry. The detection of faults is usually straightforward, but the correct identification of faults and their locations requires more complex designing. It is thus important to take fault detection into account when the design process is started, especially since unreliability increases as circuit complexity increases (Shorter, 1992:37)

Research is currently being done on the early detection of faults. Early stages of failure are often signified by parameter variation. The transient response of a servomotor will change as it starts to fail. The performance of a machine will therefore be improved if changes in parameters are detected early. The machine's control system then has ample warning time to compensate for erroneous behaviour. The process model and the parameters stored in the knowledge base must be known beforehand to obtain this level of control (Iserman, 1996a).

The knowledge base is used to store all the data that is needed for decision making. The following information is typically stored in the knowledge base:

The production history is typically used by the decision-making component for the following tasks:

a) Decisions on batch sizes for production

b) Controller compensation needed due to machine error factors that arose in previous production runs

External schedulers and human machine operators also use this data for decision making.

Mathematical models of different control situations are stored to provide the controller with extra flexibility. When control parameters for different situations vary much, the decision-making component is able to change parameters automatically or to prompt an operator to change them manually.

Tasks and schedules are also stored in the knowledge base. An external scheduler can look at the current production data, change the schedule and then the decision-making component can proceed to execute the new schedule. This can be handled automatically.

The intelligence layer does not form a controller, but it supports the control layer to form a better control system, thus enhancing the performance of the machine.

Communication layer

The communication layer is the top layer. This layer is also not part of the controller but it is shown to indicate the relationship between the controller and the external environment of the mechatronic system. There are normally three entities that need an interface to the computer, namely:

1) Human operators

2) Databases

3) Other machines

Interfaces will be discussed later in section 2.4.

The most difficult aspect of modelling a mechatronic system is to account for influences between different parts. Although the dynamic and static response for individual components might be known, it is often very difficult to model the system as a whole (Youcef-Tourmi, 1996).

Van Brussel suggests a different topology of so-called "plug-and-play" modules to bypass this problem. These "plug-and-play" modules are called holons. Each of these holons in the machine acts autonomously. Negotiation protocols between the different modules configure the system for better overall performance. Much of this work is still in a research phase (Van Brussel, 1996).

2.3.2 Control system modelling tools

Differential equations were the only design tool available for the control engineer for a very long time. The design of complex control systems such as robots was extremely difficult. Much research was therefore done to develop design tools that were able to give more insight into design problems with greater simplicity.

A very popular tool or technique, bond graphs, was developed by Henry Paynter from MIT. These graphs consist of a set of simple elements showing energy (or power) and information (or signal) flow from different parts in the control system, thus replacing the familiar block diagram (Thoma, 1975:ix). Its ability to store more information than tools such as block diagrams is a significant advantage over other tools.

Computer programs such as ENPORT allow the designer to enter a large variety of bond graphs. The program then either generates state equations for system analysis or predicts the system's response (Karnopp, 1975:27).

Bond graphs are useful in complex mechatronic systems such as robotics, because they give a good overview of the system without much loss of information.

Hardly any control system is designed today without the help of computer programs. Typical programs used by designers are Matlab, with its Simulink toolkit, Matrix-X, Simpack, ACSL and Derive (Iserman, 1996).

All these tools assist the control designer to build better control systems, as control system complexity increases due to improvement in technology.

2.4 INTERFACE SYSTEMS

The interface of the computer can be divided into four parts:

1) Interface to the operator

2) Interface to other computers

3) Interface to databases

4) Interface to peripheral devices

Each of these interfaces will now be discussed separately.

Interface to the operator

Mechatronic devices normally have interaction with an operator. A mechatronic device nearly always needs information or directions from an operator, while the operator usually needs information from the device to make decisions. Typical devices that serve as inputs to the mechatronic device are the following:

1) Control buttons on a control panel or a teach pendant.

2) Touch-sensitive screens, mouse buttons and keyboard keys in PC-controlled projects.

3) Turn-keys and knobs.

There are two types of output devices from the machine:

1) Indicator lights.

2) Display screens.

It is important to have a simple and intuitive layout of the output and input devices in order to decrease the operating learning curve of handling the machine.

Interface to other computers

Computer networks provide an interface between different computers. There are a number of network topologies, each having advantages and disadvantages.

A mechatronic system needs communication with other computers or machines when scheduling is done. A central server or scheduler is normally used to control the processes on all the connected machines for tasks such as production scheduling.

Interface to databases

The ability to store data in databases is important for mechatronic systems. This data does not only include the data used by the mechatronic system itself, but also data used by the total system, including the administration system of an organisation. Some sort of protocol has to exist to allow different applications to access databases simultaneously and to understand the information stored. A layer approach is normally adopted where data is translated from one form into another. Typical protocols used are ODBC, DDE and Active-X. The different protocol layers are often programmed into the applications for faster access to the different types of databases.

Interface to peripheral devices

There are numerous peripheral devices in a mechatronic system. Typical devices are the following:

1) Palmtop computers

2) Printers

3) Measuring equipment

These systems normally connect to the mechatronic system by means of RS424 or RS323 ports.

CHAPTER 3

THE MECHATRONIC DESIGN PROCESSES

This section discusses the design process in accordance with mechatronic principles. Not only the actual process of designing is understood under the concept of *design process*, but also quality control, organisation structure, etc.

Iserman shows that inherently there is a big difference between design in accordance with mechatronical principles and design in accordance with traditional methods. Most of these differences can be attributed to the fact that mechatronics design principles are not used from the start of the project. A working design is often taken and improved by merely adding electronic control. This is not mechatronics and the resulting products are not mechatronic products. The design team must strive to incorporate electronics and mechanics "intimately and organically" as stated in section 1.2.

Simply adding electronics to a mechanical design makes the design bulky, difficult to control and only fit for simple decision-making functions (Iserman, 1996). Hewit and King also warned against the danger of "pseudo-mechatronics" where features are added to a product merely for the sake of adding features. A typical example is the number-remembering facility on a telephone – nobody uses it (Hewit, 1996b).

It can be deduced from these statements that mechatronics is not achieved by the way a product behaves or functions. It is accomplished by the design methodology used. It is therefore meaningful to examine the design process in accordance with mechatronic principles. Mechatronics is not a design topology, but it rather supports design topologies such as concurrent engineering.

3.1 MECHATRONICS AND CONCURRENT ENGINEERING

There is a very close relationship between concurrent engineering and mechatronics as illustrated by an article by Van Brussel (Van Brussel, 1996). Most literature on quality control, production management, project management, etc. states that it is imperative for modern organisations to reduce the time to market for their products due to the short life cycles of especially electronic products. Concurrent engineering aids organisations in compressing design time to take products more quickly from the drawing board to the end-user.

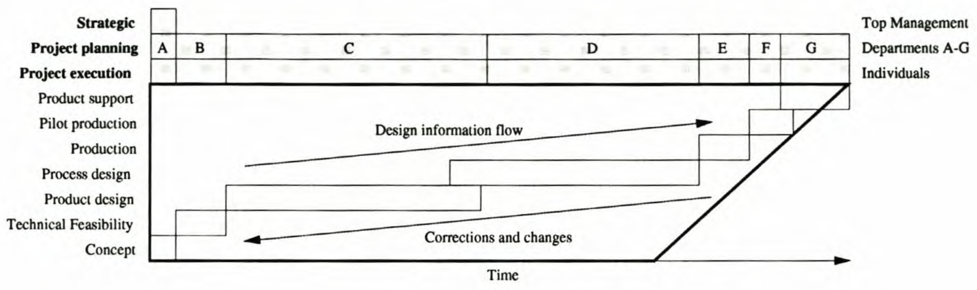
A better understanding of the relationship between concurrent engineering and mechatronic can be obtained by a brief introduction to concurrent engineering. Figure 5 shows the difference between traditional development practices and concurrent engineering. The traditional design approach in Figure 5(a) has two disadvantages that make it undesirable:

1) The different departments involved over the entire life cycle of the product function separately. A department waits for its predecessor before any work is done on the project. This is normally not the major cause of slow development, because most organisations tend to do some processes in parallel.

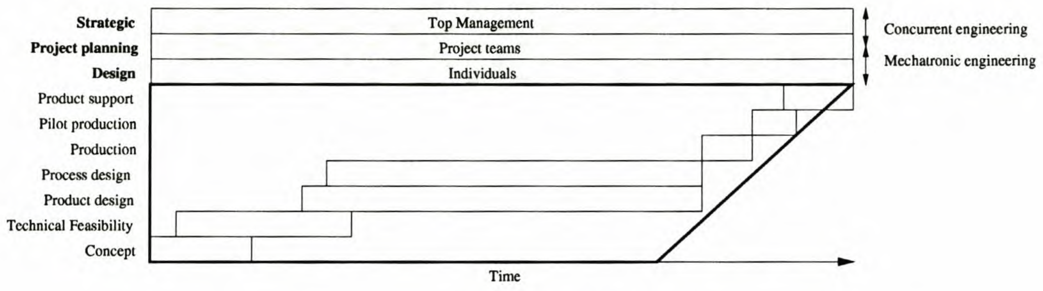
2) A major drawback of the traditional approach is the lack of efficient communication between different departments. Configuration management is often a nightmare. Departments do not understand the needs of other departments, causing much rework. It is quite likely that a product is designed to function well in an R&D department, but mass-production is impossible because of tolerances on manufacturing machinery.

The traditional approach also has other disadvantages, but most of them are spin-offs of those discussed above. As shown in Figure 5(b), concurrent engineering, when correctly implemented, does not have these drawbacks.

1) The whole organisation is involved. The entire organisation is organised around concurrent engineering and the projects running. This is often not practical. Great resistance is normally experienced when new procedures are introduced in



(a) The traditional design approach



(b) Concurrent Engineering

Figure 5 The difference between the tradisional design process and concurrent engineering (Adapted from Syan, 1994:4; Tomkinson, 1996:44 and Tomkinson, 1996:16)

an organisation. It is therefore much easier to start implementing concurrent engineering in departments or projects and to expand it from there to the whole organisation.

2) Every department that is directly involved in the project at some stage is included in the design team from the outset. This makes the parallel execution of different parts of the project possible. Shorter design times are obtained and, most important, redesigning and changes are decreased due to inputs from different stakeholders over the entire product's life cycle.

Concurrent engineering also has other advantages that are not clear from Figure 5(b) (Syan, 1994:12; Minnaar, 1994:31):

1) Reduction of manufacturing cost.

2) Higher quality.

3) Better configuration control.

4) Reduced production teething problems and cost.

Mechatronics, when implemented, forms the backbone of concurrent engineering. The cross-trained nature of the mechatronic engineer enables him to speak the technical language of all the participants on the design team. This assists concurrent engineering design goals such as the following (Huang, 1996:10):

- Design for manufacture and assembly.
- Design for inspectability.
- Design for maintenance.
- Design for reliability.
- Design for electromagnetic compatibility.
- Design for recycling.
- Design for quality.
- Design for robustness.

As stated earlier, mechatronics is not a design process, but it supports design approaches such as concurrent engineering very effectively to obtain goals as mentioned above. Authors differ on this point, but it is a matter of perspective.

PART 2

CASE STUDY OF A MECHATRONIC DESIGN

CHAPTER 4

INTRODUCTION

A metal flat bar bending machine was designed in this project as a case study of a mechatronic design.

The design of the metal-bending machine will be discussed in this part. It starts with an explanation of the design and all the design functions deployed on the machine. A discussion of the results of the project follows at the end.

Only the design of the control part of the project will be discussed for the purpose of this thesis. ROVIC did the design of the mechanical structure and the hydraulics.

A block diagram of the bending machine is shown in Figure 7.

The design team consisted of three members:

1) D.J. van der Merwe - Electronic Control (University of Stellenbosch)

2) Stèan Kroucamp - Hydraulics and Mechanical Structures (ROVIC)

3) Dr. D.C. Page - Thesis promoter and project advisor (University of Stellenbosch)

4.1 HISTORY OF THE PROJECT

ROVIC is a company specialising in the design and manufacturing of farm implements such as ploughs and front-end loaders. A need for a new manufacturing method has arisen for specifically the front-end loaders.

The front-end loaders fit onto a tractor by means of a kit. Tractor dimensions differ from model to model. Kits are therefore custom-made for each type of tractor. It is manufactured by a welding process. A typical example of a front-end loader is shown in Figure 6.

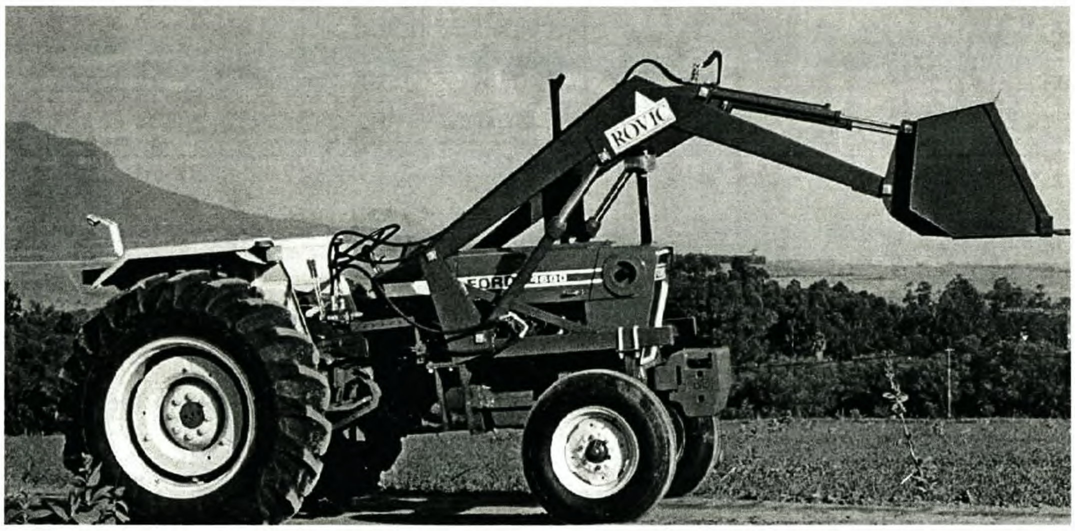


Figure 6 Typical example of a front-end loader fitted on a tractor with a kit

As tractor models are renewed, changes are made in the kit designs. Often these changes are obtained by the mere addition of a few extra welding joints. This has caused designs to be difficult and expensive to manufacture.

ROVIC decided on a new approach. Sheet metal that is pre-cut into the proper shape will be bent into the desired form, thus minimising welding. Design changes will therefore mainly

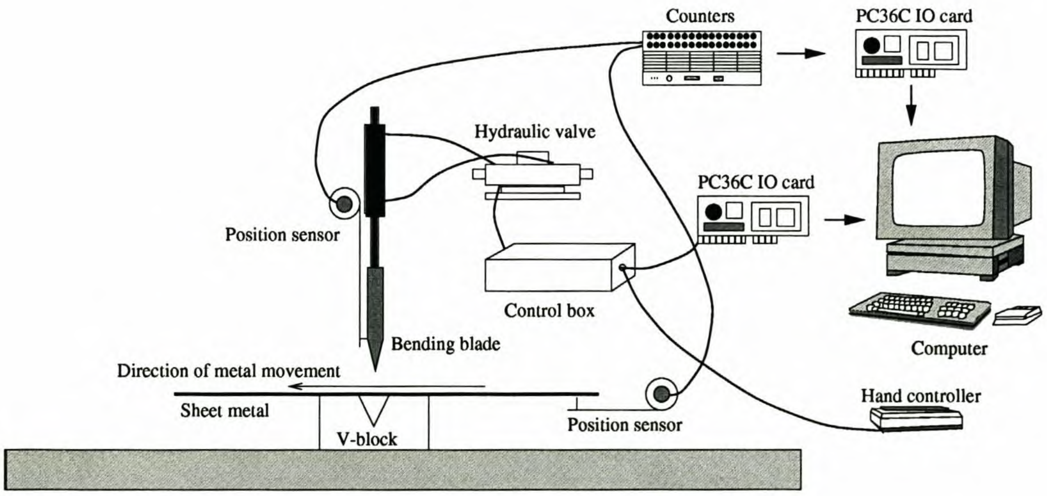


Figure 7 Block diagram of the sheet metal bending machine.

consist of changes in bending parameters, which is much easier to do.

It is important that the bending angle error must be within acceptable margins. An angle of 0.5^0 was defined at the start of the project as the maximum allowable error. Larger errors would cause the kit to be twisted, which could cause additional stresses on the tractor block. This could cause metal fatigue on the tractor block.

4.2 MECHANICAL DESCRIPTION OF THE BENDING MACHINE

The bending machine is shown in Figure 8. The machine is divided into three parts:

1) Bending blade

2) Framework

3) Bending bed

These parts form an integrated design, but each will be discussed separately.

4.2.1 The bending blade

The bending blade presses on the metal bar that lies on the V-block on the bending bed. Two types of bending blades are currently in operation, a 5mm-radius blade for thin metal (<15mm) and a 30mm-radius blade for thicker sheet metal (>15mm). The bending blades were hardened to ensure a longer operational life for the blades.

The blades are connected to the machine frame by a hydraulic cylinder. The cylinder is capable of pressing loads of up to 80 tons. The position sensor of the bending blade is not attached to the hydraulic cylinder, but to the frame. Position is measured directly on the blade. This eliminates most cylinder deflection errors.

4.2.2 Framework

The framework was designed to handle loads of up to 800MN. The control accessories are covered in a metal container on the framework. Three electrical devices are attached to the frame:

Hand controller – The hand controller is placed on the frame so as to be within easy reach of the operator while handling material.

Emergency button – The emergency button is positioned for quick access in emergencies or accidents.

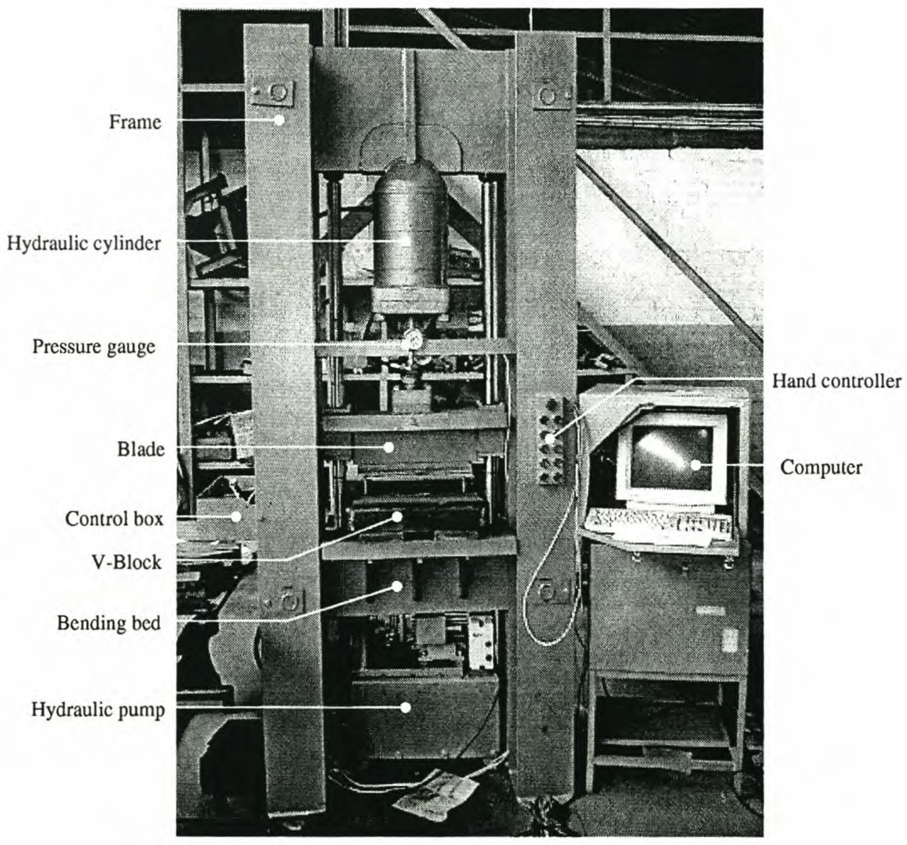


Figure 8 Full-length photo of the bending machine

Micro-switch – A micro-switch is placed at the back of the bending press. The micro-switch is activated when the bending blade reaches the top of its stroke.

4.2.3 Bending bed

The V-block is placed on the bed of the bending press. The V-block is lined up to the bending blade ensure bending in the correct position.

4.3 DESCRIPTION OF THE CONTROLLER

The control design is divided into two parts:

1) Electronic hardware that controls the machine.

2) Control program of the electronic hardware.

The electronic design is discussed in chapter 6. As shown in Figure 7, the electronic hardware consists of the following cards:

PC36C digital IO card – Data IO to the computer is done by means of this card. Only the counter circuit is connected to this card.

Counter circuit – The counter circuit counts the pulses from the incremental encoders.

Interface circuitry – The interface circuitry (situated in the control box in Figure 8) connects the hand controller and the computer to the driver circuit.

Driver circuit – The driver circuit (also in the control box in Figure 8) is used to drive the solenoid.

The control program is also divided into four parts as explained in chapter 5:

1) Regression analyser.

2) Bend sequence file editor.

3) Program option editor.

4) Control program.

4.4 DESIGN PARAMETERS

Several design parameters were defined at the start of the project. These parameters were fundamental decisions taken at the start of the project. The following parameters were defined:

Computer – The hardware responsible for all calculations and decision-making processes of the controller.

Operating system – The operating system provides the environment in which the control program runs.

Programming language – The programming language chosen does not necessarily influence the functionality of the machine, but it can be of great assistance in synthesising the control program.

Position sensor – The position sensors give position feedback of the bending blade and position stop movement to the controller.

IO card – The IO card provides interfacing between the control program and the bending press sensors.

Hydraulic valve and solenoid – These were regarded as part of the mechanical design of the project, but due to the mechatronic nature of this project, design considerations from both mechanical and electronic viewpoints were considered. The hydraulic and solenoid system provides a transducing interface from electronic signals of the controller into mechanical actions.

Electronic controller of hydraulic valve – The electronic controller is responsible for amplification of the signals for opening and closing the hydraulic valves.

Table 4 Results of the design parameter quantification

Item	Dimension
Computer	AMD K5 P133 microprocessor with 24MByte RAM
Operating system	Windows NT 4
Programming language	LabVIEW 4.1
Position sensor	5000 pulse per revolution incremental shaft encoder
IO card	PC36C from Eagle Technology
Hydraulic valve	12V solenoid
Electronic controller of hydraulic valve	Own driver circuitry
Communication	Buffer connection circuitry between hand controller, computer and driver circuitry. Support circuitry for hardware generated functions.

Communication – The communication methods and tools between the operator and the bending machine consist of the computer screen and the hand controller.

The quantification of the different design parameters is discussed in Appendix B. The results of this process are listed in Table 4. Only the most important specifications are listed.

CHAPTER 5

THE BENDING PROGRAM SUITE

The different programs in the bending program suite are discussed in this chapter. The required functionality of the programs and the methods used for realising it are also discussed.

The following functionality was defined at the start of the project:

1) The control program must control the machine according to commands from a GUI (graphical user interface).

2) The program must read a bend sequence file and control the bending machine according to the commands specified by each program line.

3) The program should allow a hand controller to control the machine without intervention from the computer.

4) The program should be able to capture positions and save them in a bend sequence file.

5) A spreadsheet look-alike independent editor must be used to enter and edit bend sequence programs. A syntax checker should be added to ensure that logical values are used.

6) An independent program should be used to set the options for the machine.

7) A program that is capable of determining the bend equations should be added to the suite.

5.1 CONTROL PROGRAM OF THE BENDING PRESS

The control program is responsible for the control of the bending machine. All programming functions related to control are programmed into this program. The different control functions can be grouped as follows:

1) Manual control of the machine through the computer.

2) Automatic control by means of the bend sequence programs.

3) Manual control by means of the hand controller.

4) Position acquisition functions.

5) Change program settings manually.

The control functions should not be regarded as separate pieces of programming code, because they are not. It is more important to realise that the program is capable of executing these functions.

5.1.1 Programming language

The program was written in LabVIEW 4.1. LabVIEW is supplied by National Instruments and was developed specifically for control applications. LabVIEW 4.1 runs in both Windows NT4 and Windows 95. According to a representative of Westplex (local distributor of LabVIEW) application written in LabVIEW for Windows 95 is not guaranteed for correct behaviour. This is due to the fact that Windows NT is a much more stable environment than Windows 95.

LabVIEW differs from traditional programming packages in a number of ways as shown in Table 5.

The use of *data flow* instead of *code flow* to determine the execution sequence of commands is a totally new programming philosophy. With *code flow*, the program has a predefined sequential order by which commands is executed. Most traditional programming language uses this technique. With *data flow* commands are executed when data is available for the command.

Table 5 Difference between LabVIEW and traditional packages

	Labiew	Traditional languages
Programming method	Graphical	Text and/or graphics
Programming executing rules	Data flow	Code flow

It is therefore difficult to predict in which order events occur in a LabVIEW program. In cases where the sequence of actions is important, LabVIEW controls program flow by means of sequence frames. LabVIEW has equivalent programming structures for nearly every programming concept in traditional languages, including for-loops, case statements and while-loops.

The LabVIEW equivalent of programs is virtual instruments. The entire LabVIEW paradigm is built around virtual instruments. The LabVIEW program behaves like and looks like the control of an instrumentation box. The LabVIEW equivalent of procedures is also virtual instruments. Any LabVIEW program (or virtual instrument, more correctly) can make use of other virtual instruments by connecting its inputs and outputs to the appropriate ports in the LabVIEW program.

Typical examples of LabVIEW programs are shown in Appendix E.

5.1.2 Basic program outline

The program is divided into three parts:

1) Initialisation of all the different parameters of the machine according to the values preset in the Options module.

2) Control of the bending machine.

3) Restoration of all the program conditions of the machine to their original states before closing the program.

The functions described in section 5.1.4 through to section 5.1.8 are all embedded in the second part of the control program. These functions are not implemented separately, but are a single programming entity.

5.1.3 Initialisation of program parameters

The different default values for initialising the control program are pre-set according to the values stored in the *Options* module. At the program start-up all values are loaded and stored in

Table 6 The settings loaded from file for the applicable control values of the program.

Control	Function
Pulse period	The period of a single pulse during automated operation.
Pulse length	The width of a pulse during automated operation.
Top	The height (in millimetres) above the zero point to which the bending blade retracts after the *Retract to zero* button has been pressed.
Bottom	The maximum distance in millimeters below the zero point where movement is allowed.
Pulse band	The pulse band in degrees – used for control according to the calculated bending angle.
Stop band	The stop band in degrees – used for control according to the calculated bending angle.
Pulse band (distance)	The pulse band in millimetres – used for control in terms of distance moved by the bending blade.
Stop band (distance)	The pulse band in millimetres – used for control in terms of distance moved by the bending blade.
C-enco	Constant that converts the value read by the position encoder on the bending blade to a distance in millimetres.
C-pos	Constant that converts the value read by the position encoder on the position stop to a distance in millimetres.
Pulse period closed	The period of a single pulse during manual operation using the computer.
Pulse length closed	The width of a pulse during manual operation using the computer.
Metal type	The type of metal bent.
A-param	The value for the A coefficient of the bending equation according to the chosen metal type.
B-param	The value for the B coefficient of the bending equation according to the chosen metal type.
C-param	The value for the C coefficient of the bending equation according to the chosen metal type.
D-param	The value for the D coefficient of the bending equation according to the chosen metal type.

the memory in the same order as listed in Table 6.

After loading these values, the two PC36C cards are initialised into a start-up configuration. The exact realisation of the initialisation process is discussed in Appendix E.2.

1) The counters are zeroed and all counter control pins are set to high.

2) Manual control by the hand control of the bending machine is turned off and control is given to the computer.

A number of controls on the console of the computer program must also be set to predefined conditions. These conditions are hard programmed and cannot be changed. The settings are

Table 7 The predefined settings for the different control values in the program

Control	Scope	Type	Value
Capturing allowed	Local	Boolean	False
Add line	Local	Boolean	False
Insert line	Local	Boolean	False
File index	Global	Integer	0
File info	Local	String	“ ”
File name	Global	String	“ ”
Capture	Local	String array(1)	Enable
		String array(2)	Enable
		String array(3)	“ ”
		String array(4)	“ ”
File loaded	Global	Boolean	False

listed in Table 7 and Table 8.

5.1.4 Manual control using the computer

The manual control of the machine performs the following functions:

1) Move the machine upwards continuously.

2) Move the machine downwards continuously.

3) Pulse the machine upwards.

4) Pulse the machine downwards.

5) Zero the counter of the machine.

6) Retract machine to zero plus x mm.

Table 8 The predefined conditions for the different attribute nodes in the program

Attribute node	Attribute	Value
Reset	Visible	False
Go	Visible	False
Halt	Visible	False
A-param	Visible	False
B-param	Visible	False
C-param	Visible	False
D-param	Visible	False
A-parameter	Visible	False
B-parameter	Visible	False
C-parameter	Visible	False
D-parameter	Visible	False
Param	Visible	True
Unload	Visible	False
Reset	Visible	False
Capturing compensation	Visible	False
Compensation	Visible	True
Metaltype	Visible	False
Angle	Item Names	" "
Information	Item Names	" "
Nr	Item Names	" "
Compensation	Item Names	" "
Metaltype	Item Names	" "

7) Halt the machine if it is busy with a movement. This function operates similar to pause in normal computer programs.

Continuous movement of bending press

The same virtual instrument does both the upward and downward continuous movement of the machine. The virtual instrument responsible for the movement is discussed in Appendix E.3.3.

The movement virtual instrument needs two inputs:

- The direction of movement (*True* = *up* and *False* = *down*)
- Movement (*True* = *There is movement* and *False* = *No movement*)

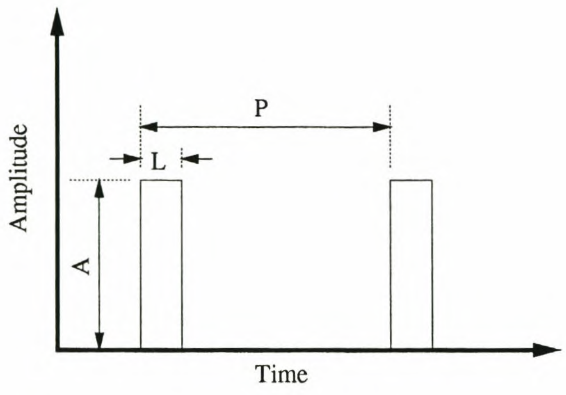


Figure 9 The parameters for a pulse signal

L – Pulse length
P – Pulse period
A – Pulse amplitude

Pulse movement of bending machine

This function uses a *pulse* virtual instrument, discussed in appendix E.3.3. Both the downward and upward pulse functions use this virtual instrument, as was the case with continuous movement.

This virtual instrument has the following inputs:

- The direction of movement (*True* = *up* and *False* = *down*)
- Pulse length – The length of the pulse (output high) in milliseconds
- Period – The period of the pulse, total duration of a pulse (Output high plus output low) in milliseconds

Figure 9 shows the parameters of a pulse.

Zero the bending blade counter or the position stop counter of the machine

These two functions are exactly the same except that they operate on different counters. They do not call a custom virtual instrument, but operate directly on the virtual instruments supplied by the manufacturers of the PC36 boards. The code is discussed in Appendix E.3.3. Zeroing of the counters enables the operator to define a zero point on top of the metal flat bar. The computer needs the zero point to calculate where bending of the metal bars starts.

The zeroing process is as follows:

1) Move the bending blade downward continuously into close proximity of the metal flat bar to be bent.

2) Pulse the blade downwards from there until it reaches the sheet metal and a predefined pressure is shown on the pressure gauge.

The same procedure is used to zero the position stop counter. This counter is zeroed when a suitable reference point is reached.

Retract the machine to zero plus x *millimetres*

With this function the operator retracts the blade to *x* millimetres above the zero point. The value of *x* is loaded during the initialisation stage (see section 5.1.3) of the program.

This function is used when the metal flat bar is moved from its current bending position to the next. The code is discussed in Appendix E.3.3.

Halt the machine if it is busy with a movement

There are a number of functions for moving the bending blade to predefined targets. These target points are sometimes incorrect. The control system of the machine can also fail. This function allows the operator to stop the blade movement if one of these conditions arises.

5.1.5 Automatic control by means of the bend sequence programs

This set of program functions performs the following tasks:

1) Load a bend sequence file.

2) Start a new bend sequence file for capturing positions.

3) Capture bending positions in a bend sequence file.

4) Execute one line of program code.

5) Set the compensation for a given bend.

6) Repeat a bending line.

Load bend sequence file

A bend sequence file has to be loaded into memory before the *Go* button can work. The layout of a bend sequence file is first discussed to provide a clearer understanding of the tasks required in this function.

A typical sample file is shown in Figure 10. The line numbers on the left are added for clarification and are not part of the file.

Line 1 contains information of the file. The information line is obsolete for the purpose of executing bend sequence files, but it is handy during editing to have some description of the file.

Line 2 contains the parameters for the equation used for the calculation of movement vs. bending angle. The equation is as follows:

$$\theta = ax^2 + bx + c - ce^{-dx} \qquad (5.1)$$

Equation (5.1) will be discussed in section 7.3.

The different parameters appear in the same order in line 2 as in equation (5.1). The variable, θ, denotes the desired angle, and x denotes the linear movement of the bending blade.

```
1     This is a bend sequence file for 30mm sheet metal
2     -0.0019; 2.5; 0.8; -0.01
3      1; 30; 1200; 45; 150; 150;  G45;   90; Be careful of this bend;
4      2; 30;  800; 45; 180; 150;  G45;  100; Be careful of this bend;
5      3; 30;  600; 45; 135; 150;  G45;   85; Be careful of this bend;
```

Figure 10 A typical example of a bend sequence file

The information for the bend sequences follows from line 3. Each line performs one bend. The function of each parameter is shown in Table 9. The different parameters also appear in the same order as the values in line 3 and onwards. The obsolete values are marked with asterisks and were added only for compatibility should the need arise for them in future projects.

The tasks performed after calling this function are as follows:

1) If a file is loaded, the operator is prompted to query whether the changes to the current file may be discarded.

2) The operator is then allowed to choose a file from disk.

3) The header line of the file is loaded first into the memory and is shown in the *file information* indicator.

4) The parameter line is loaded second. The parameter buttons are set immediately according to the loaded values. These buttons are never visible to the operator.

5) The bend sequence program is next loaded into memory and the whole program is stored in list boxes as discussed in Appendix E.3.3.

6) Other functions are informed that a file has been loaded into the memory by setting a Boolean flag.

The means of realising this function is discussed in Appendix E.3.3.

Table 9 The purpose of the different parameters in a bending line in a bend sequence file

Parameter		**Purpose**
Number		The number of the bend in the bend sequence file.
Width	*	The width of the sheet metal.
Length		The distance from the position stop where the bending must start.
Thickness	*	Thickness of the metal.
Angle		Desired angle for the bend.
Jig	*	The width of the V-block.
Punch	*	Type of blade that will be used.
Compensation		The percentage compensation for the specific batch of sheet metal.
Info		Any extra information for the specific bending line.

Table 10 Information stored during the capturing process

Item	Description
Line number	The line number of the bend.
Angle	The uncompensated value for the target angle.
Position	The position of the position stop.
Compensation	The amount of compensation required for the bend.

New bend sequence file

This function allows the user to enter the file name for a file in which the operator can save captured positions. The function is described in detail in Appendix E.3.3. The following tasks are executed by the function:

1) The program confirms with the operator whether it is allowed to load a new program.

2) All the program list boxes are emptied and the user is prompted to enter a new file name.

3) There after the user is prompted to enter a new header line (see Figure 10).

4) The other program functions are informed that a file has been loaded by setting a Boolean flag.

Insert single line and add lines

Sometime it is easier to program a bend sequence file by an exemplary process. Programming is done as follows:

1) The bending blade is moved to the desired position.

2) The position is then captured into a *bsq* file.

This facility only serves as a tool for programming and should not be used as the only form of programming.

Lines are added by two methods to the bend sequence file:

1) *Insert line* to insert a line at the current program pointer.

2) *Add line* to add a program line at the end of the *bsq* file.

The functions are divided into the following tasks:

1) Obtain the current values stored in the different parameters.

1) Insert the values at the current file position / add values at the end of the file.

2) Renumber the program lines.

These two functions are discussed in greater detail in Appendix E3.3. Table 10 shows the values that are saved by these two functions.

Reset the computer program

This is a very simple function. The counters of the different program list boxes are set to zero.

Execute one line of program code

This function is one of the most important functions of the control program. It allows the operator to execute one line of program code. It is activated by pressing either the *Go* button on the computer screen or hand controller. The following tasks are performed during this function:

1) The current angle is read from the bending blade encoder counter.

2) The compensated value is compared with the target value of the current program line to determine the direction of movement.

3) The compensated value is compared with the target value of the current program line to determine what type of movement is to be executed:

- Continuous movement.

- Pulsed movement when in the pulse band.
- No movement when in the stop band.

4) The program counter is incremented to execute the next program line.

The program code for the function is discussed in Appendix E.3.3.

Close the file loaded

The loaded bend sequence file is not closed in the same way as on normal programs. The program only resets all the bend sequence list boxes and the program counter. A Boolean flag is set to indicate to other functions that no file has been loaded.

Enable the capturing of positions

This function toggles the program between two states:

1) Capturing of program lines allowed.
2) Capturing of program lines not allowed.

With capturing allowed, the *Add line* and *Insert line* controls are shown. With capturing disallowed, these buttons are invisible.

A different compensation button, *Capturing compensation*, is also used during capturing, to prevent confusion by the LabVIEW program.

Save

When the S*ave* button is pressed, the current file is saved to disk. *The process of saving is exactly the opposite of loading a file.*

Setting the compensation values

This function is performed as part of the *Execute one line of program code* function and was discussed as such. This is also an important function when a specific bend is repeated.

South African steel varies considerably in thickness. The thickness of metal bars has a major influence on the behaviour of the metal during bending. It is therefore difficult to calculate the bending angle if the thickness of the metal varies much. The compensation function was added to compensate for these errors. Angles are corrected by overbending or underbending angles by an adjustable number of percentage points. A compensation value of 100% means no compensation. Percentage values are estimated by the operator and are based solely on experience.

5.1.6 Manual control with the hand controller

The hand controller functions separately from the computer. Its circuitry is discussed in section 6.2.

The following functions are associated with the hand controller:

1) Switching between ownership of control between the hand controller and computer.

2) The buttons pressed on the teach pendant (including the other switches on the bending machine) are read into the computer memory.

Switching between hand controller and computer control

The hand controller is used as the default controller when no computer is present. A control signal from the computer is used to select between computer control or control by the hand controller. This function is divided into the following tasks:

1) Set and reset the control bit. The control is responsible for selecting the control method. It is discussed in section 6.2.

2) Hide all computer controls directly related to computer control. The following buttons are hidden:

Continuous up	Continuous down
Pulse up	Pulse down
Zero	Retract to zero
Reset program	Capture
Go	Halt

3) The counter values are updated concurrently. The operator can therefore determine the blade's displacement while using the hand controller.

Controlling the LabVIEW control program by the hand controller

A number of buttons on the hand controller can initiate program functions while computer control is used:

1) *Go* – Executes current program line.

2) *Halt* – Halts execution of current computer-induced action.

3) *Capture* – Inserts a program line at the current program pointer.

4) *Release* – Releases bending blade to zero+*x* millimetres where *x* refers to the value stored in the global variable, *Top*.

Table 11 List of different states in the control program

Screen state	Function
Auto	Bend sequence file is loaded and program execution is enabled
Parameters	Set flat bar parameters
Hand control	Enable hand controller for manual control

5.1.7 Position acquisition

Displacement of two components is measured:

- Movement of the bending blade.
- Movement of the position stop.

Bending blade displacement

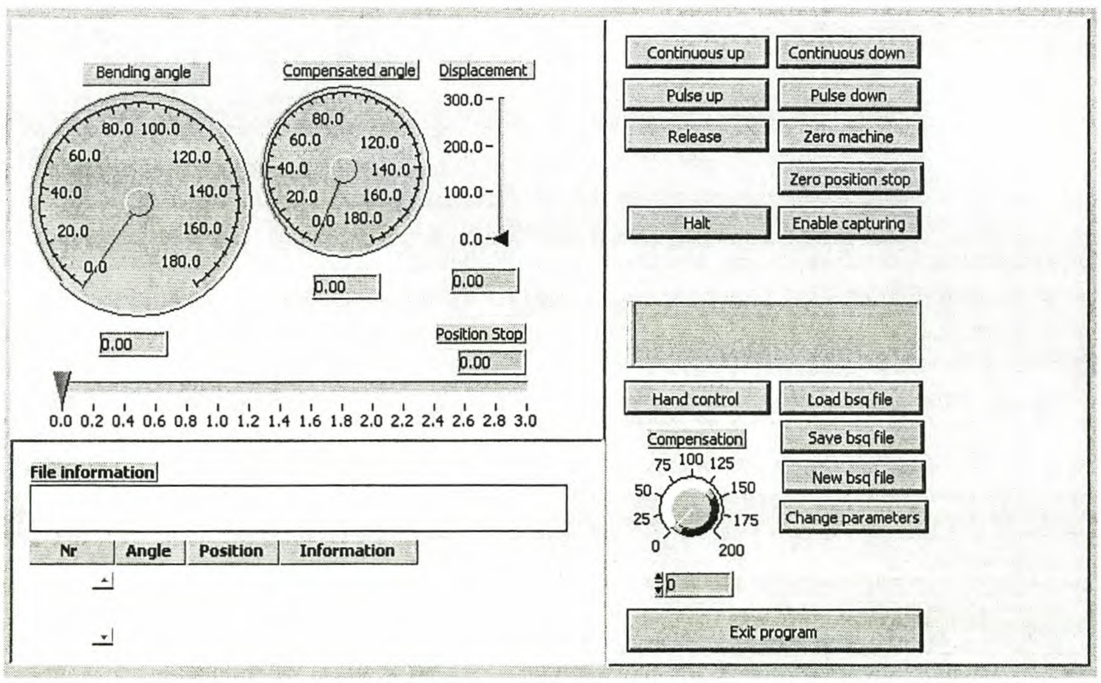


Figure 11 Control screen of computer for the execution of a bend sequence program

Two values are obtained from the displacement measurement of the bending blade:

1) Linear displacement measured in millimetres.

2) Estimated angle of the resulting bend. Spring-back is included in this estimation. This calculation is done according to different parameters for each type of metal as discussed in section 7.2.2.

The realisation of the function is discussed more detail in Appendix 5.1.

Movement of the position stop

The operator needs only the position of the position stop. Only the linear displacement of the position stop is therefore calculated.

5.1.8 Manual change of program settings

The parameters of bending equation 6.1 are set to standard values at the program start-up. These values are changed for different types of flat bars. When a *bsq* file is loaded into the memory, the parameter values are changed to the values specified in the parameter line of the *bsq* file (see section 5.1.5).

If no file is loaded, the parameter values are changed by pressing the *Change parameters* button on the computer screen. A LabVIEW control, containing all the metal types, will be shown. The values for the different metals are loaded during the program start-up. After choosing the appropriate metal type, the *Change parameter* button is released by repressing it.

5.1.9 The layout of the computer screen

A representation of the control screen is shown in Figure 11. The screen has three states as shown in Table 11. Figure 11 shows a screen in auto state.

The screen is divided into input buttons and knobs (known as controls) and outputs (known as indicators). The functionality of the different controls and indicators is listed in Table 1. States are changed by hiding or showing controls and indicators.

Table 12 Functionality of the different input and output controls on the control screen

	Control	Function
Inputs from operator	Continuous up	Move bending blade upwards continuously.
	Continuous down	Move bending blade downwards continuously.
	Pulse up	Pulse bending blade upwards.
	Pulse down	Pulse bending blade downward.
	Go	Execute one bending line.
	Halt	Stop the movement of the machine.
	Zero	Zero the position counters of the machine.
	Retract to zero	Move the blade of the machine to the zero position plus *x* millimetre (higher than zero).
	Reset	Reset the program.
	Load	Load a bend sequence file.
	New	Start a new file for position capturing.
	Change parameters	Change parameters for the angle-linear displacement relationship.
	Hand control	Turn on manual control by hand controller.
	Bend information array	This array of controls shows the control program. The current line for execution can be selected from it.
	Exit	Quit program.
Outputs to operator	Position indicator	The position indicator shows the exact movement of the bend blade.
	Bending angle	Angle estimated from the linear displacement of the bending blade.
	Compensated angle	Angle estimated from the linear displacement of the bending blade with the compensation factor added.

5.2 PROGRAM FOR SETTING CONTROL OPTIONS

This module sets the control parameters. The program was separated from the control program for the following reasons:

1) Normal operators are not allowed to set any parameters. Workers operating the press are often unskilled and do not understand the effects on the bending process when program parameters are changed.

2) Options are saved as text in a text file. LabVIEW 4.1 does not have adequate functionality to handle text. Delphi is better equipped for this function.

The interface of the program is divided by four tabs into the following four sections:

1) Dies.

2) Punches.

3) Bending options.

4) Machine options.

The tabs are chosen by clicking on them with the mouse button.

5.2.1 Layout of the option-file

The following file is loaded at the program start-up:

```
c:\buig\opsies.txt
```

The program will fail if the file is not stored in this directory.

The file is arranged in a specific format, illustrated in Figure 12. Each line represents one unit of information. For instance, all information for a specific blade, H45, is contained in one line, line 5.

The file is divided into different parts by keyword headings as follows:

Dies – Line 1 V-block types are stored in this part. Information is divided into two fields, which are separated by a semicolon:

1) V-block size

2) V-block description

Punch – Line 4 Blade types are stored in the second part. It is also divided into two fields:

1) Blade type

2) Blade description

A semicolon again separates the two fields.

Buig – Line 7 Information of all maximum and minimum values for different numeric fields is stored in this part. Each value is listed on its own line. The following values are listed:

1) Maximum thickness of metal.
2) Minimum thickness of metal.
3) Maximum width of metal.
4) Minimum width of metal.
5) Maximum angle allowed for bending.
6) Minimum angle allowed for bending.
7) Maximum length of metal.
8) Minimum length of metal.
9) Maximum compensation allowed.
10) Minimum compensation allowed.

Buigmasjien -- Line 18 The operational parameter values are stored in the last part. The following values are stored:

1) Pulse length.
2) Pulse period.
3) Top.
4) Bottom.
5) Dead band.
6) Pulse band.

```
1      Dies
2      A24; Big die
3      A12; Small die
4      Punch
5      H45; Red punch
6      K45; Yellow punch
7      Buig
8      100
9      10
10     70
11     34
12     180
13     -10
14     1500
15     0
16     100
17     0
18     Buigmasjien
19     500
20     22
21     300
22     50
23     2
24     0.5
```

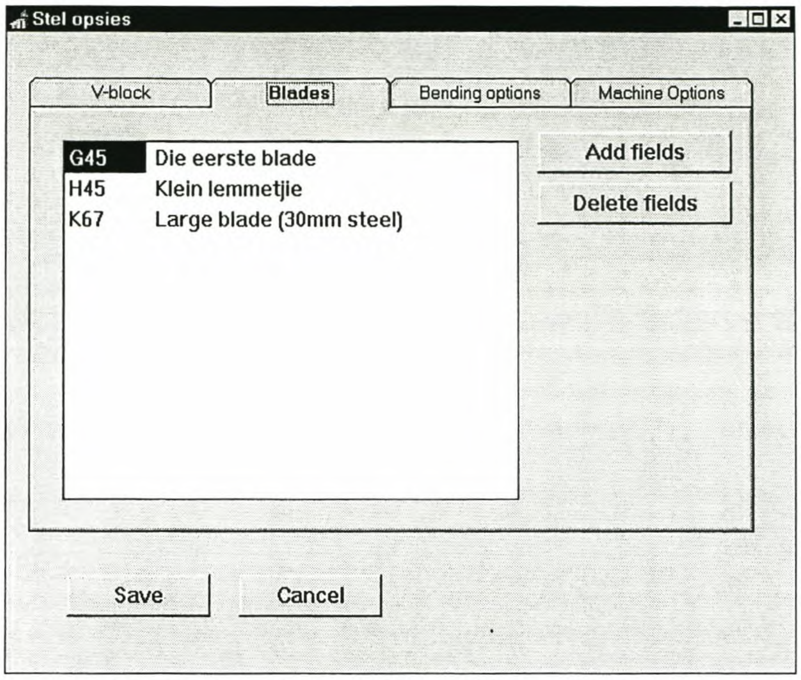
Figure 12 Printout of a typical example of the file, opsies.txt. Line numbers on the left are not part of the file, but are listed to aid understanding.

Record-handling capabilities of raw text in any programming language is usually minimal. The whole file is therefore loaded into the correct record structure at start-up. It is easy to process the information in these structures, using the record-handling capabilities of Delphi.

5.2.2 Blades

The interface is shown in Figure 13. Blades entered in the bend sequence program must correlate with blades listed here. This information enables the editor (see section 5.3) to verify that the correct blades are being used.

The control program does not need this information for correct execution, but the information is added to serve as an extra precaution to ensure that the operator uses the correct blade. It has no influence on bending press behaviour.



G45	Die eerste blade
H45	Klein lemmetjie
K67	Large blade (30mm steel)

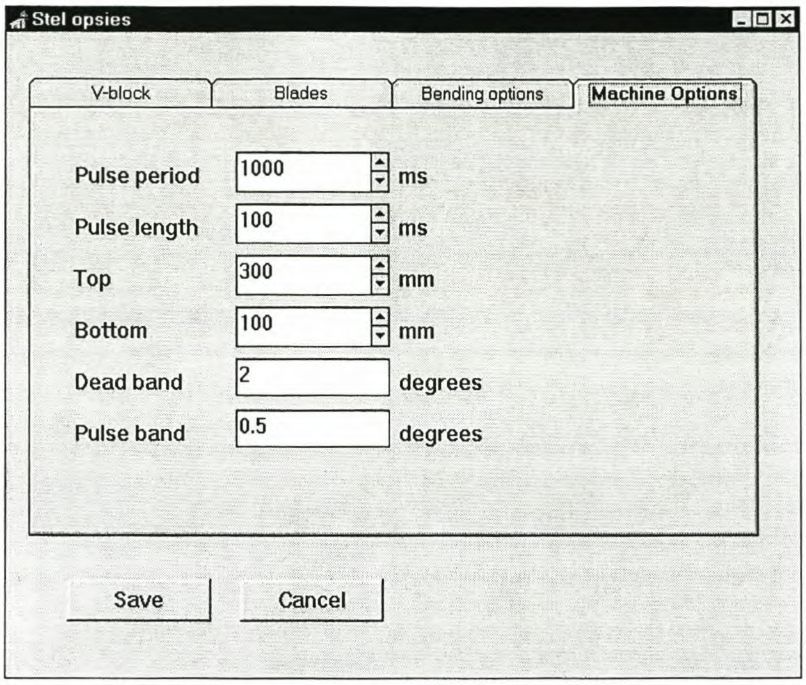
Figure 13 Screen showing the different blades entered in the program

The name of the blade is entered in the first field and the description in the second field. Each line forms a record. A new blade can be added by pressing *Add New Record*. Records are deleted by pressing *Delete Record* after positioning the cursor anywhere on the row earmarked for deletion. A number of rows can be deleted as follows:

1) Select the topmost or bottommost corner of the selection.

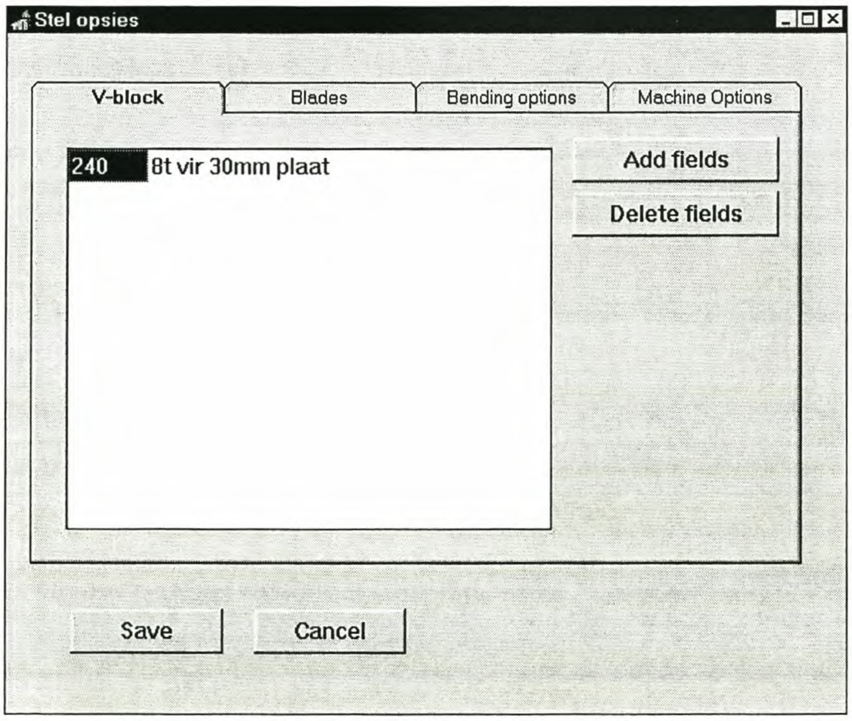
2) Press shift and while holding it down, use the cursor keys to select the other rows.

3) Press *Delete Record* by means of the mouse.



Pulse period 1000 ms
Pulse length 100 ms
Top 300 mm
Bottom 100 mm
Dead band 2 degrees
Pulse band 0.5 degrees

Figure 14 Screen showing the utility for setting bending machine options



240 8t vir 30mm plaat

Figure 15 Screen showing the different v-blocks entered in the program

5.2.3 Machine options

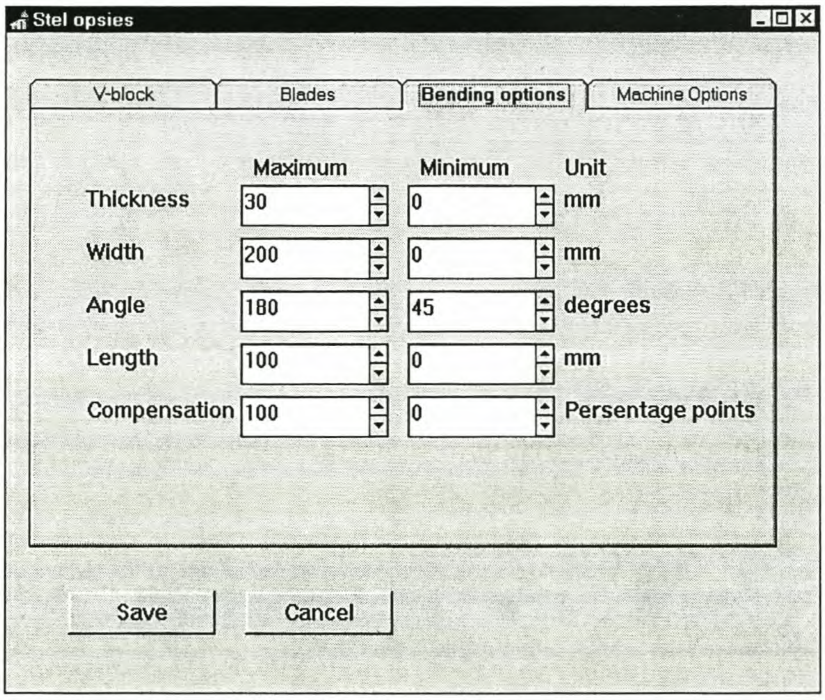
The different parameters that determine bending machine behaviour are entered here as shown in Figure 14. Each parameter's function was dealt with in the discussion of the control program in section 5.1.3.

5.2.4 V-Blocks

The interface is shown in Figure 15. The procedure for entering values is the same as for the blades.

5.2.5 Bend sequence file options

Maximum and minimum values for different numerical parameters of the bending program are entered here. Values are entered by selecting the specific field and entering the new values by means of the keyboard. The program uses these values for debugging the bend sequence program. Prevention of errors is therefore improved if correct values are specified. This function is not required for correct program functioning. A screen of this function is shown in Figure 16.



	Maximum	Minimum	Unit
Thickness	30	0	mm
Width	200	0	mm
Angle	180	45	degrees
Length	100	0	mm
Compensation	100	0	Persentage points

Figure 16 Screen showing the utility for entering bending program options

5.3 BEND MACHINE CODE EDITOR

The editor for the bend sequence programs is easy to operate. It is shown in Figure 17.

The editor is divided into two tabs:

1) Information of the bend sequence file.

2) The bend sequence program.

Entering of password

The user is prompted for a password before access is allowed to the program. The password is set to:

```
rovic
```

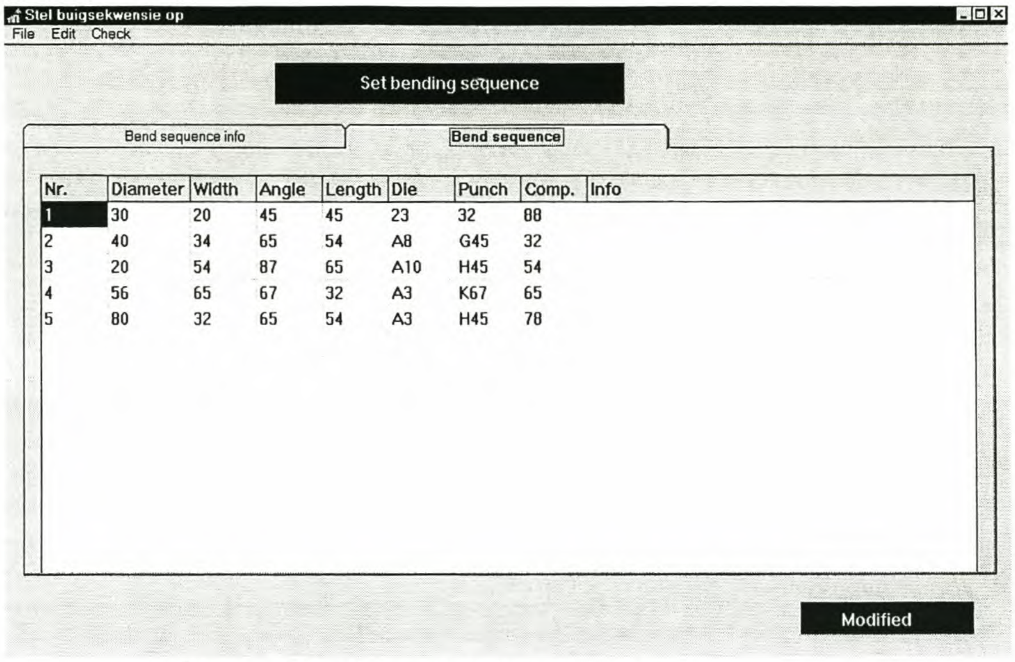
The password is case sensitive. Normal precautions of asterisks that appear instead of password letters were omitted. The password is also not coded, but appears in the following text file:

```
c:\buig\passwd.txt
```

The password was not meant to serve as a major security measure. It prevents unskilled persons from using the program. Only three attempts at entering the correct password are allowed before the program shuts down. The program must be restarted after that.

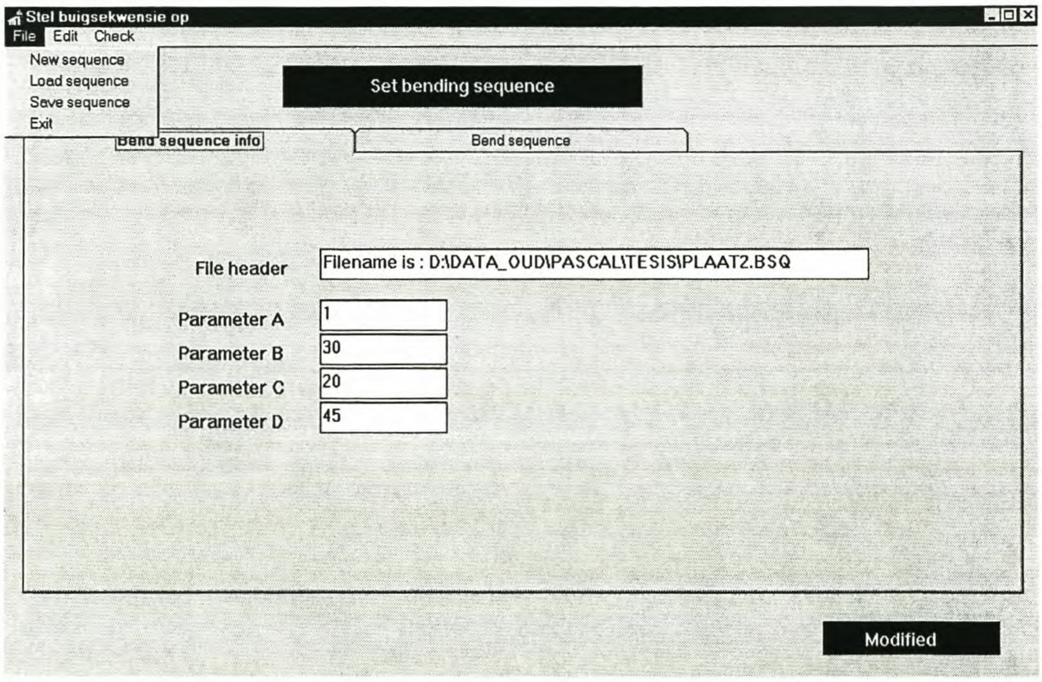
Entering the bend sequence program

The tab for entering the bend sequence program is shown in Figure 17. It consists of a text grid with command buttons at the bottom. The program parameters are entered in the grid. Each horizontal line forms one program line.



Nr.	Diameter	Width	Angle	Length	Die	Punch	Comp.	Info
1	30	20	45	45	23	32	88	
2	40	34	65	54	A8	G45	32	
3	20	54	87	65	A10	H45	54	
4	56	65	67	32	A3	K67	65	
5	80	32	65	54	A3	H45	78	

Figure 17 Screen showing the bend sequence programmer



File header	Filename is : D:\DATA_OUD\PASCAL\TESIS\PLAAT2.BSQ
Parameter A	1
Parameter B	30
Parameter C	20
Parameter D	45

Figure 18 Screen showing the bend sequence programmer for entering bend information

Table 13 Functions of the different menu commands

Submenu	Command	Function
File	New	Load a new bend sequence file. If the current program is not saved, the programmer is prompt to save the program.
	Load	Load a previously saved program from disk. The user is again prompt to save the current program in memory. The current program is deleted from memory before loading the new program.
	Save	Save the program
	Exit	Exit the program
Edit	Insert row	Insert lines
	Delete row	Delete the selected lines
	Copy	Copy the selected lines to the clipboard
	Paste	Paste the lines currently in the clipboard
	Delete	Delete the selected cells
	Cut	Copy the selected lines to the clipboard and delete it from the grid
Check		Check the program for errors. The error checking is done according to the parameters set in the options tool. It is important to notice that the control program will execute bend sequence programs regardless of any errors. The check facility only serves as an aid to prevent program errors.

The user has to enter all the parameters. The program does not enter any parameters, such as line numbers, automatically. Values are entered by selecting the appropriate cell, typing the information and pressing *Enter*.

The *Edit*-submenu on the menu applies only to this tab.

When data is entered the status indicator, at the bottom right, immediately indicates that the file has been changed. The user will be prompted to save the file after any command is given to clear the current bend sequence file from memory.

Entering the information for the bend sequence program

This tab is shown in Figure 18. The header line is only an information line in the file. It helps the operator to know the purpose of a program. The control program does not use the V-block or blade data to indicate what type of V-block or blade is to be used. This information should be added to the header line in future versions.

The boxes for parameters A to D are used to set parameters for the equation describing the angle-linear displacement relationship as discussed in section 5.1.5.

Menu

The program is menu driven. The different menu commands are listed in Table 13. The menu is activated by pressing the *Alt* button or by clicking on it with the mouse button. It works like the normal Windows '95 and NT menu system.

5.4 REGRESSION ANALYSER

The equation to describe the relation between the blade movement and the bending angle was shown in section 5.1.5. It was found, as will be described in section 7.3, that the non-linear equation, $y = ax + b - be^{-cx}$ best fits the data. The metal behaviour was not tested for angles outside the linear region of the plastic region. It was therefore decided to use the equation:

$$y = ax^2 + bx + c - ce^{-dx} \qquad (5.2)$$

The coefficients of equation 5.2 must be found for each type of metal. It is determined by doing regression analysis of the experimental data. The experimental data is shown in Appendix H.1. ROVIC does not have access to programs capable of doing non-linear regression. A program was therefore written in Delphi for this purpose. The source code for the algorithm used is shown in Appendix J.

5.4.1 Program description

The program has the following functions:

Load and save of data file

The data points can be saved to and loaded from disk. A file structure with two fields, similar to bend sequence files, is used. A semicolon is again used to delimit the fields.

Editor for entering data points

Data points are entered in a text grid that serves as an editor. It is very important that all cells are filled with data. Unfilled cells will cause incorrect program behaviour.

Analyser

The analyser is responsible for performing regression analysis. It makes use of the Gause-Newton algorithm by computing the least squares as described by Gallant (1987:30). This algorithm is very sensitive to the signs of the initial values. Initial values are therefore set by default to the most likely signs.

The step size determines the speed at which the equation converges. Large steps normally reach a solution faster then smaller steps, but they are also more likely to diverge. The program checks for diverging behaviour and decreases the step size automatically if necessary. It is important to remember that the step size must be a value between 0 and 1. The analyser is discussed in more detail in Appendix J.

Copy coefficients

This function copies the current coefficients to the initial value boxes. They are then used as initial values in analyses thereafter.

Other menu functions

The menu has the following functions required for editing:

Insert – Add a line at the end of the file.

Delete – Delete the line at the current location of the cursor.

New – Create a new file.

The program can run regression only on equation 5.2. A new program must be developed to run regression on other equations. The user interface is shown in Figure 19.

5.4.2 Method for choosing suitable initial values

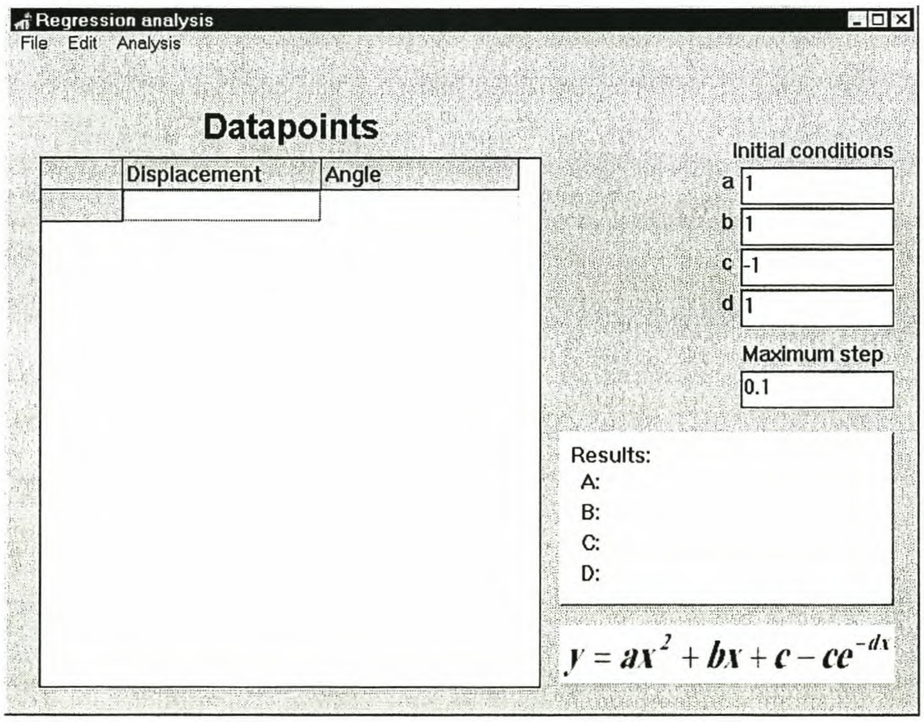
Correctly chosen initial values are very important for regression. Poorly chosen values may prevent the equation from converging. The following principles must be applied to aid the convergence of the bending equation:

1) A large number of accurate readings with no or little outliers improve the accuracy of the equation and the chances of convergence.

2) The signs of the initial parameters must have correct values. This is usually sufficient to ensure convergence.

3) More measurements must be taken, especially in the transitional region from elastic to plastic.

4) Excel may be used to make an educated guess of the parameters. The following



Initial conditions: a 1, b 1, c -1, d 1

Maximum step: 0.1

$y = ax^2 + bx + c - ce^{-dx}$

Figure 19 Screen showing of the regression analyser

procedure is used:

a) Enter the values in Excel.

b) Draw a XY graph with the position values on the X-axis and the measured angles on the Y-axis.

c) Right-click on the graph and choose *Add trend line*.

d) Specify the program to use a second-order trend line. (Make sure that the y offset is not set to zero and that the equation for the trend line is shown on the graph.)

e) Enter the coefficients of the equation as initial values for regression.

If these methods fail, the operator has to derive the equation by means of better statistical programs.

CHAPTER 6

CONTROL CIRCUITRY

The control circuitry is an important part of the project. It consists of the control hardware for the bending press.

An attempt was made to save design costs from the start of the project. Printed circuit boards were developed where no cost-effective commercial cards were available. The circuitry is divided into three groups:

1) Commercial cards.

2) Printed circuit boards developed.

3) Peripheral devices.

Commercial computer cards

Two PC36C cards were purchased from Eagle. The PC36C card is a digital IO card that makes computer connection to external electronic devices possible. The rationale for this choice is discussed in Appendix B. The PC36C card has the following important features:

- An 8255PPI port is simulated. It has three bi-directional ports (Ports A, B and C). It is possible to set up Ports A and B for either output or input. The upper and lower nibbles of Port C are separately set up for input or output.
- 12V, 5V, -12V and -5V voltage sources that are capable of driving small loads.
- Plug and play circuitry that enables Windows NT and '95 to recognise the presence of the board.

The cards were set up as shown in Table 14. The set-up program for the cards was supplied by Eagle technology. The accompanying manual gives clear instructions for the implementation of the cards. Information is also available on the Internet (Eagle 1999).

Table 14 The set up parameters of the two PC36C cards

Number	Base address	Function
1	360h	Position acquisition of the bending blade and the position stop.
2	300h	General control of bending machine.

Printed circuit boards developed

The PC boards were developed specifically for the bending machine, but an attempt was made to keep designs generic. Each PC board can function independently in other applications. The following three boards were developed:

1) Counter for pulses from the incremental shaft encoders.

2) Connection circuitry to connect all electronic components of the bending press.

3) Driver for the solenoids of the hydraulic valve.

These boards are discussed in section 6.1 to section 6.3.

Peripheral devices

The application has a number of peripheral devices:

1) Incremental encoders

2) Hydraulic valve

3) Power supply

4) Hand controller

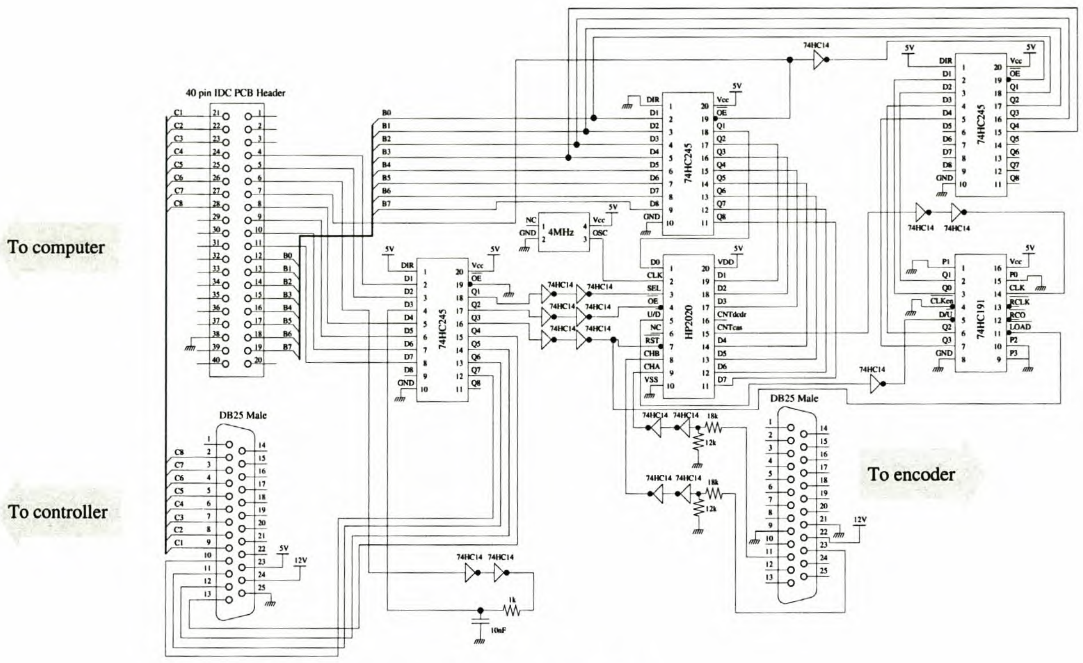


Figure 20 Circuit diagram of the counter circuitry

The peripheral devices are discussed in section 6.4.

6.1 COUNTER CIRCUITRY

The counters are used to count the number of pulses generated by the encoders. The working of the encoder is discussed in section 6.4.4 and Appendix A.1.

This PC board has counter circuitry for each position encoder. One counter is connected to Port B of the PC36C and the other to Port C. Control lines for both counters are connected to Port A.

The circuit diagram for the counter circuitry is shown in Figure 20.

6.1.1 Circuit description

HCTL2020 counter

The HCTL2020 counter is responsible for counting the encoder pulses. The chip reads both phases A and B from the encoder via Schmidt triggered not-gates. It has internal filters that remove most of the noise from the encoder signals. This improves the accuracy of pulse counting.

It has one eight-bit data port with a clock output for a cascaded counter chip. The data port has two internal buffers of eight bits each – called the high byte and the low byte respectively. Data in the two buffers is read sequentially to produce a 16-bit word. The computer program is therefore capable of calculating a maximum count of 65 535. It is possible to enlarge the maximum count number by using a counter cascade.

The internal counting circuitry of the HCTL2020 uses an external clock signal for pulse counting. The calculations for the required oscillator frequency are shown in Appendix A.2. An oscillator of 4Mhz is used.

The board has two HCTL2020s – one for each encoder.

74HC191

This chip forms a cascaded counter and with HCTL2020 forms a counter of twenty bits with a maximum count of 1 048 576. The counter needs a direction and clock signal which are both supplied by the HCTL2020.

Encoder input

The encoder needs a power supply of between 10 and 30V. The PC board supplies 12V to the encoder, and the encoder output is therefore also 12V. This is brought down to TTL levels by means of a voltage divider as shown in Figure 20. The output is sent through Schmidt triggered not-gates before going to the HCTL2020. The direction signal from the encoder is not used, as the HCTL2020 generates its own direction signal.

Buffers

The buffers are used to protect the HCTL2020 from large external signals. They are also used to protect the PC36C from large signals from IC's on the board.

Schmidt triggered not-gates

These gates serve as simple noise filters. The gates change logic states according to a hysteresis function, which removes noise.

4Mhz oscillator

The oscillator provides the clock frequency for the HCTL2020. A frequency of 4Mhz is used for two reasons:

1) 4Mhz is very close to the maximum frequency that the HCTL2020 can handle.

2) Lower frequencies would increase the possibility of missing pulses when the encoders rotate too fast.

Calculations for the required frequency are discussed in Appendix A.2.

Power supply and ground

It was found that the power supply from the computer was not satisfactory, and a power supply was therefore added to the circuit.

A transformer steps the external AC voltage of 220V down to 15V. The alternating current is converted by a diode bridge and capacitor connection to 15V direct current. The components on the PC board can require a current of up to 2.4A. Unacceptably large capacitors are therefore required to keep voltage ripples within acceptable values. Voltage regulators were therefore used to remove ripples from the DC power supply. 12V and 5V voltage regulators were used, one for each counter circuit. The encoders require 12V and the rest of the circuit 5V.

Table 15 Parameters for the zero-pin filter

Parameter	Dimension	Unit
V_{High}	4.8	V
V_{Low}	0.1	V
$V_{Positive\ going\ threshold}$	2.38	V
$V_{Negative\ going\ threshold}$	1.40	V
$t_{Low\ to\ high}$	7.06	ms
$t_{high\ to\ low}$	12.9	ms

An important feature of the transformer is the floating ground provided to the circuit. Ground shorts are thus prevented.

By-pass capacitors were used on all chips to eliminate switching glitches due to internal switching in the ICs.

Capacitor-resistor filter

A capacitor-resistor filter was added to the lines leading to the *zero* pin of the HCTL-2020. This pin was inclined to give switching glitches under specific counting conditions, but it was impossible to discover the cause of the problem. The filter remedied the problem.

Table 16 Calculated specifications for the counter circuit

Item	Dimension	Units
Power supply		
Voltage	12	V
	5	V
Current	3	A
Counters		
Number of counters	2	
Count direction	Bi-directional	
Maximum count number	1 048 576	Each
Maximum pulse count frequency	4	Mhz
Input voltage from encoders	12	V

The filter makes use of the hysteresis properties of Schmidt triggers to filter noise. It has a time constant, τ, of 10μs that forces the computer to negate the *zero* pin of the HCTL-2020 for longer than 13μs when it resets the counter. The calculated operational parameters for both the positive and the negative going voltages are shown in Table 15. Calculations were done using equation 6.1:

$$V_{out} = \left(1 - e^{-\frac{t}{RC}}\right)V_{in} \quad (6.1)$$

6.1.2 Conclusion

It was found that the counter reset owing to unknown causes at the count of 16. The problem was successfully solved by means of a filter circuit.

It was never established whether similar problems existed on other control lines of the HCTL2020. Programming and counting tests were done, but they were inconclusive. No faulty readings were found that were traced back to switching glitches on the HCTL2020 control lines.

The calculated specifications for this PC board are shown in Table 16. These specifications were not practically verified due to a lack of appropriate measuring devices, but there was no evidence during general testing that proved the results wrong.

The PC board has two counters that are connected to incremental encoders. The encoders require a voltage source of 12V, which is supplied by the PC boards. The counter board is connected to the PC36C IO card via a ribbon cable. A list with the cable line names appears in Table 17. The assembly of the PC board is discussed in Appendix F.

6.2 CONNECTION CIRCUITRY

The connection circuitry does not only connect the different boards, but it also has embedded functions to aid in the control of the machine. The circuit diagram is shown in Figure 21 and the line connections in Table 17.

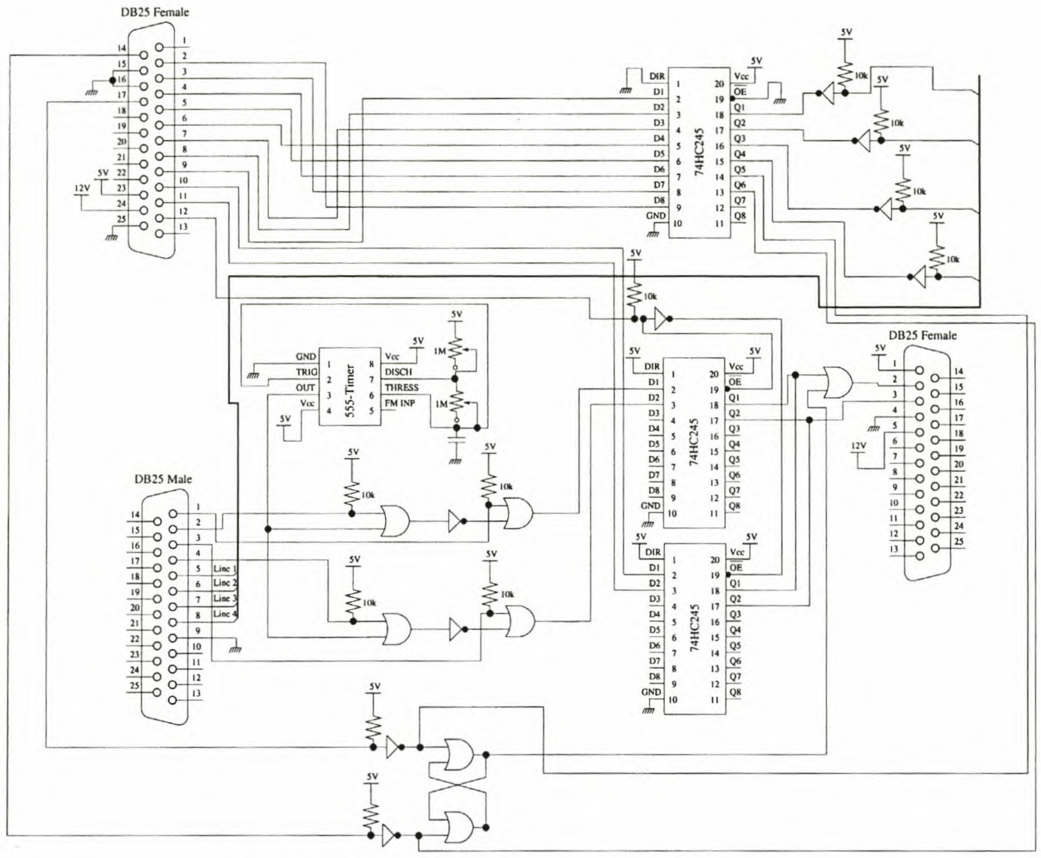


Figure 21 Circuit diagram of the connection circuitry

6.2.1 CIRCUIT DESCRIPTION

555-Timer

The 555-Timer develops the pulse signal that pulses the bending blade in small increments. The length of a pulse is set by potentiometer R1 and the period by the combination of R1 and R2. Both the potentiometers have maximum resistances of 1MΩ. The 100nF capacitor, C1, causes the 555-timer to generate a maximum pulse width of ≈70ms and a maximum pulse period of ≈210ms. Twenty-two turn precision potentiometers were used. Each full turn of R1 therefore causes a change of ≈3.15ms in the pulse width.

The pulse period depends on the values of both R1 and R2 as shown by equations 6.2 and 6.3:

$$Pulse\ length = 0.693 \times (R_1 C) \qquad (6.2)$$

$$Pulse\ period = 0.693 \times C(2R_1 + R_2) \qquad (6.3)$$

Development of control signals for the driver circuit

Four types of signals are needed for the control of the driver circuitry:

1) Continuous up- or downward movement of the bending blade.

2) Pulsed up- or downward movement of the bending blade.

3) Feedback signals to the computer.

4) Emergency signal.

Table 17 Input and output lines from the counter card to the PC36C

	Bit	Pin	Direction	Function
Port A				
Line	0	4	Output	$\overline{OE}$ on HCTL2020 #1
	1	5	Output	$\overline{Select}$ on HCTL2020 #1
	2	6	Output	$\overline{Zero}$ on HCTL2020 #1
	3	7	Output	Choose between input from HCTL2020 #1 or 74HC191 #1
	4	8	Output	$\overline{OE}$ on HCTL2020 #2
	5	9	Output	$\overline{Select}$ on HCTL2020 #2
	6	10	Output	$\overline{Zero}$ on HCTL2020 #2
	7	11	Output	Choose between input from HCTL2020 #2 or 74HC191 #2
Port B				
Line	0--7	12--19	Input	Counter #1
Port C				
Line	0--3	23--20	Input	Lower nibble counter #2
Line	4--7	24--27	Input	Upper nibble counter #2

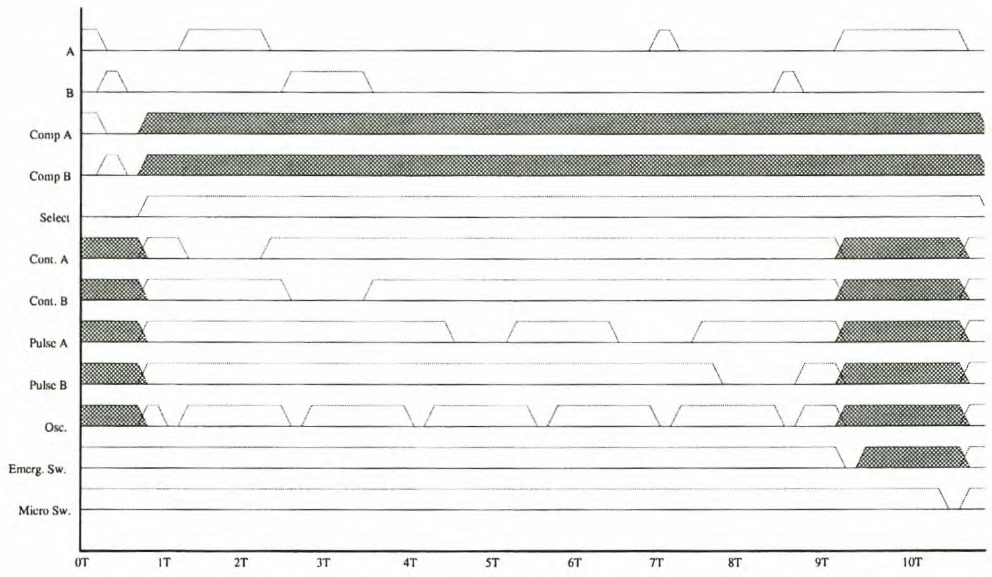


Figure 22 The different signals and their outputs to solenoids A and B

The driver circuit is controlled by two TTL-level signals, one for each solenoid on the hydraulic valve. The output to the solenoid is connected to pins 2 and 3 respectively in the circuit diagram in Figure 21. These signals are denoted by A and B in Figure 22. Signal A causes upward movement and signal B downward.

The solenoid control signals have two sources, the hand controller and the computer. A tri-state buffer is used to switch control between the hand controller and the computer. When the computer is disconnected, a pull-down resistor, R3, is used to force control to the hand controller. This signal is denoted by *Select* in Figure 22. When connected, the computer determines which device generates the control signal.

The generation of the computer control signals (*Comp A* and *Comp B* in Figure 22) is discussed in Appendix E.5.3. No circuitry is therefore required to generate control signals for the driver circuit when computer control is used.

The hand controller (Discussed in section 6.4.2) generates all other signals except the signals from the emergency and micro-switch. The hand controller signals are active low. Inverters are therefore used to change it to active high.

The continuous signals generated by the hand controller are denoted by *Cont. A* and *Cont. B* in Figure 22.

The pulse function is generated by means of an or-gate as shown in Figure 21. The *555-timer* generates an inverted signal (*Osc.*). The signals from the pulse buttons, *Pulse A* and *Pulse B*, are active low as stated above. The or-gate with the 555-timer signal and a button signal as inputs generates an active low control pulse signal when the button is pressed. An inverter is used to change the signal to active high. An or-gate is used to merge the pulsed and continuous signals. This circuitry is used for both pulse buttons.

Emergency switch and micro-switch

The mechanical functionality of the emergency switch (signal *Emerg.~Sw.* in Figure 22) differs from that of normal switches. Once the switch is pressed, the operator has to turn the button to release it. The emergency switch is very important because hydraulics can be very dangerous. The circuit is designed to give precedence to the emergency switch before any other control function.

The micro-switch (signal *Micro-sw.* in Figure 22) is used to signal when the bending blade reaches the top of its allowable stroke. It is activated when the blade presses against the switch when it moves over the switch.

The emergency switch is connected to an SR flip-flop generated by the two NAND-gates. The behaviour of the flip-flop connection is shown in Figure 23. The output is high until the micro-switch is triggered. The emergency signal is connected to the upward signal line (*A* in Figure 22) by an or-gate.

Table 18 Functions of the hand control buttons used by the computer

Button	Name	Function
5	Go	Execute one program line.
6	Halt	Stop the movement of the bending machine blade.
7	Capture	Capture the current position of the bending machine blade.
8	Retract	Retract the bending machine blade to zero + *x* mm.

Button feedback to computer

Output from the unused buttons, micro-switch and emergency switch goes back to the computer. The program is consequently informed of external events. The functions of the unused buttons are listed in Table 18.

6.2.2 CONCLUSIONS

The SR-latch was unstable. Glitches were encountered on the signal lines from the emergency switch that activated emergency mode. The cause is likely to be switching of the solenoid on the driver circuit, which caused a dirty power supply line. The problem was remedied by 100nF capacitors on both the inputs to the SR-latch. The activation time of the emergency switch was consequently slower, but the reaction time was still in the order of microseconds, which is omissible.

Specifications for the connection circuitry are shown in Table 19. The circuitry was also designed for use in other applications where requirements are similar to those for this application.

6.3 DRIVER CIRCUIT

The driver circuit is responsible for switching the solenoid. The solenoids switch the hydraulic valve between its three states (discussed in section 6.4.3).

6.3.1 CIRCUIT DESCRIPTION

The circuit diagram is shown in Figure 24. The driver circuit is divided into two parts:

- Control logic.

Table 19 Specifications for the connection circuitry

Item	Dimension	Unit
Power supply		
Voltage	5	V
	12	V
Current	800	mA
Pulses		
Max. period high	70	ms
Max. period low	140	ms
Max. pulse period	210	ms

- Driver circuitry.

Control logic

Only one solenoid may be switched at any given moment. This prevents the solenoids from working against each one another.

This functionality is obtained with control logic, which ensures that upward signals always take precedence over downward signals. This ensures that the blade always retracts upwards when the emergency button is pressed.

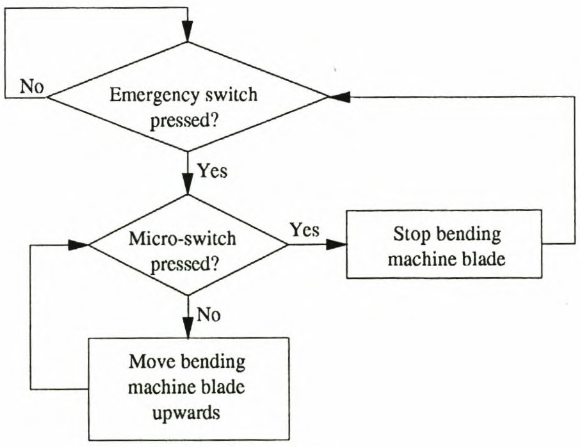

No
Emergency switch pressed?
Yes
Micro-switch pressed?
Yes
Stop bending machine blade
No
Move bending machine blade upwards

Figure 23 Working of the emergency switch.

Table 20 Specifications for the driver circuitry

Item	Dimension	Unit
Supply voltage	5	V
	12	V
Output resistance	0.07	Ω
Input lines	A	Upwards
	B	Downwards

Driver circuitry

The circuitry makes use of pull-up resistors to change the signal's on-voltage level from 5V to 12V. The 12V signal goes to a push-pull transistor pair. The transistors are general-purpose small signal transistors that supply current to switch the mosfets faster into its non-linear regions. This keeps the operation of the mosfets out of the linear region where heat dissipation is high.

The mosfets were specially chosen for their very low internal impedance. The solenoid has an impedance of 4.8Ω. The IRF540 mosfets have an internal impedance of 0.07Ω, which is negligible compared with the solenoid impedance.

The solenoid is connected between the 12V and the mosfets and not between ground and the mosfets.

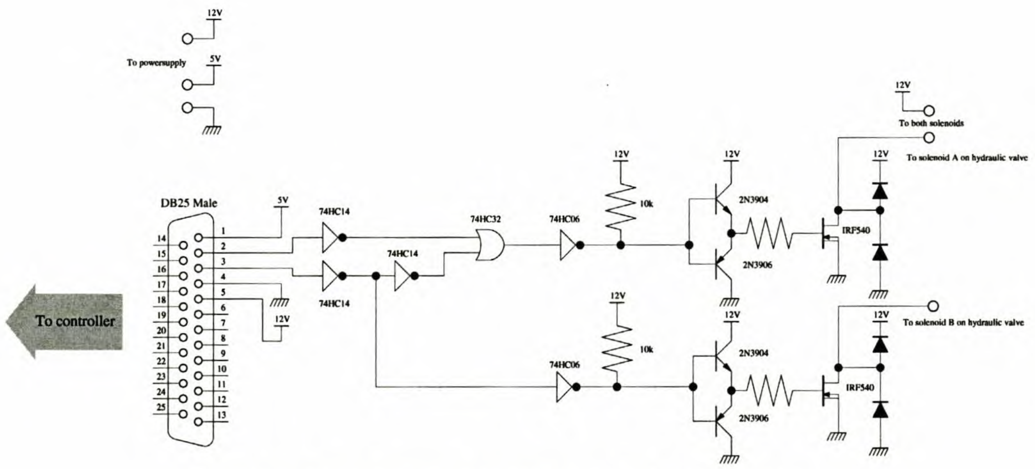


Figure 24 Circuit diagram of the driver circuitry for the solenoids

6.3.2 CONCLUSIONS

The driver circuit causes noise on the power supply line that could influence the behaviour of other PC boards. All the identified problems related to bad power supplies were solved. A new version of the control circuitry for the bending press should have individual power supplies for each PC board.

The specifications for the driver circuit are shown in Table 20.

6.4 PERIPHERAL DEVICES

6.4.1 POWER SUPPLY

The power supply was obtained from an old computer. It is a switch-mode power supply, which means that heat dissipation should be low even with large loads.

Most of the power is used by the driver circuitry. The power supply is consequently connected to the circuitry via the driver circuit so that the thick power supply wires can be shortened and removed from the rest of the circuitry.

The specifications of the power supply are listed in Table 21.

6.4.2 HAND CONTROLLER

The hand controller is attached to the bending machine. It is used for manual control of the bending blade and is equipped with eight control buttons. A schematic layout of the hand

Table 21 Specifications of the power supply used in the project

Item	Dimension	Unit
Type	Switching supply	
Power output	220	W
Max Current at 12V	8	A
Max current at 5V	23	A
Max current at -5V	0.5	A
Max current at -12V	0.5	A
Input voltage	220±15%	V

controller is shown in Figure 25.

The buttons connect their corresponding signal lines to ground when pressed. The pull-up resistors on the connection circuitry pull the signal lines to 5V when the buttons are released.

6.4.3 HYDRAULIC VALVE

The hydraulic valve used was specified by ROVIC. The symbolic representation is shown in Figure 26. The oil flows freely to the sump with the valve in the neutral position, which removes backpressure on the pump.

This configuration reacts slower than when the pump and sump are open connected if the valve is in the neutral position. This did not prevent the press from achieving a high resolution. The specifications for the valve are shown in Appendix B.6.

6.4.4 INCREMENTAL SHAFT ENCODERS

The incremental shaft encoder consists of a rotating disk with a light emitter and photo detector, much like the example shown in Figure 27.

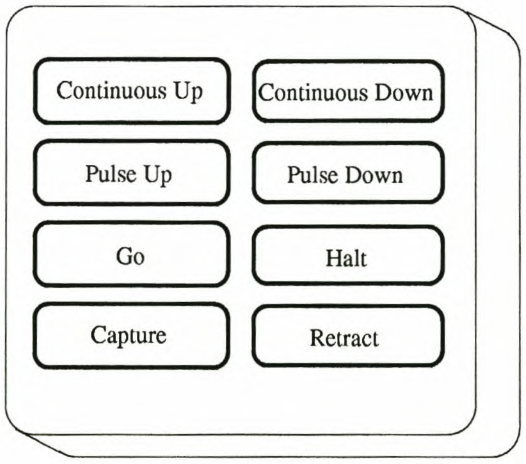


Figure 25 Diagrammatic representation of the hand controller

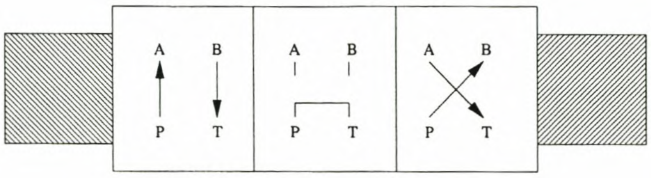


Figure 26 Symbolic representation of the hydraulic valve.

The encoder has two output channels, A and B, which send a pulse when the photo detector detects a signal from the light source. The channels are 90° out of phase to enable external circuitry to determine the direction of movement by detecting which phase is leading. Movement is measured by counting the number of pulses. The HCTL2020 has all these functions embedded (National Instruments, 1999a; National Instruments, 1999b).

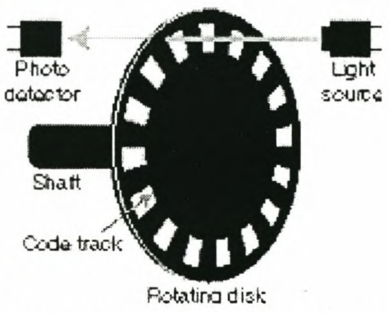


Figure 27 Diagrammatic representation of an optical encoder. (National instruments, 1999a)

CHAPTER 7

RESULTS

7.1 SET-UP PARAMETERS USED FOR EVALUATION OF MACHINE

The set-up parameters for the bending process are entered by means of the option program (see section 5.2.3). Values for the different parameters used during evaluation of the bending

Table 22 Parameter values used during evaluation of the bending press

Parameter	Value	Unit
Pulse period	100	ms
Pulse length	100	ms
Top	10	mm
Bottom	100	mm
Calibration constants		
Bending blade	0.01034	
Position stop	0.0096563088	
Pulse band	2	degrees
	2	mm
Stop band	0.15	degrees
	0.1	mm

press are shown in Table 22.

Best results were obtained with a pulse length of 20ms and a pulse period of 100ms. These values allow sufficient time for the valve to open and close. Compare this with the method used by Jürgens (1998:115) where the valve is pulsed faster than its opening and closing time, i.e. opened for a very brief period.

Movement of the bending blade is slow, leaving adequate time for valve switching. Faster movement would prevent the press from achieving an accuracy of ±0.5°.

Optimum values for the pulse band and stop band were not researched, but the values shown in Table 22 gave adequate results.

It is important that the calibration values should not change if it is not absolutely necessary. The constants have an influence on the parameter values of all the equations deduced for the different metals. These values were determined as accurately as possible to minimise errors. The calibration process is discussed in Appendix C.1.

7.2 THE BENDING PROCESS

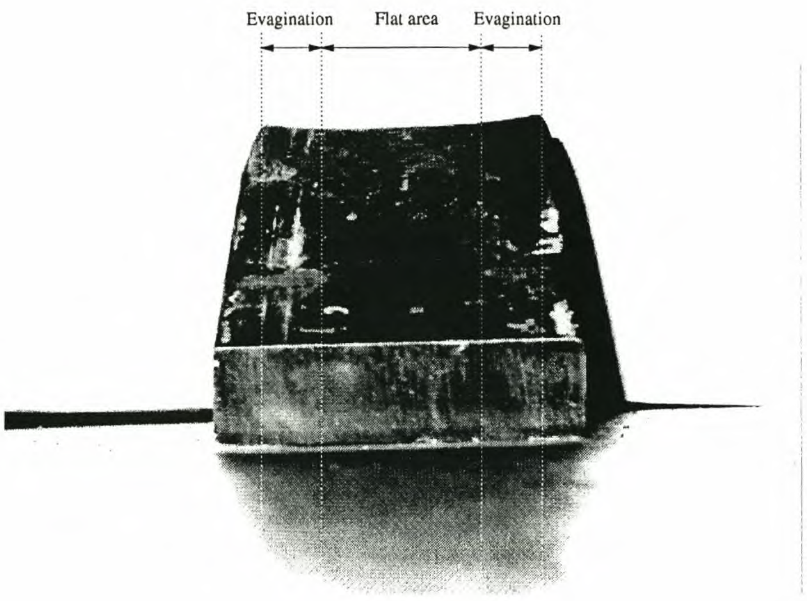
7.2.1 Bending of metal flat bars

Figure 28 shows photos of practical bends done on metal flat bars. Photo (A) shows a front view of the bent material. Two regions in the bend can be distinguished:

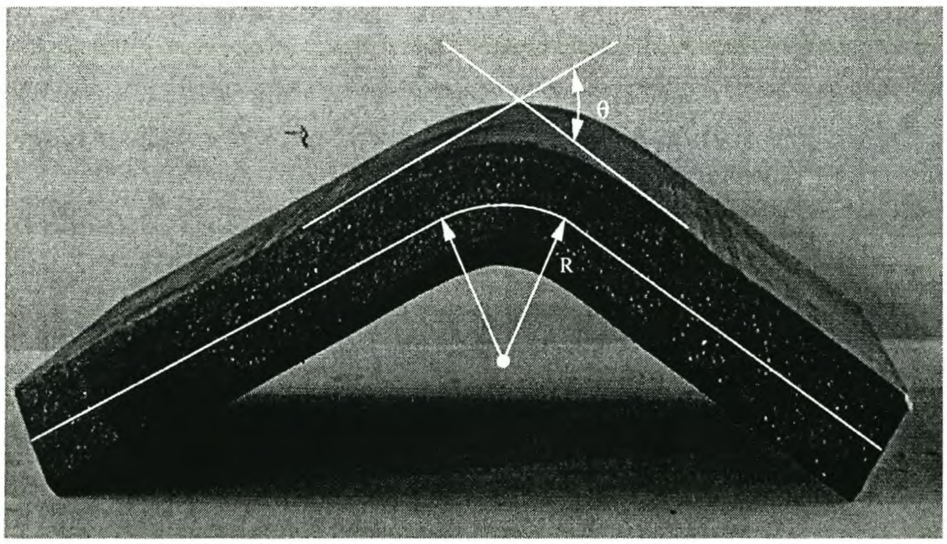
Evagination: This is caused by the lateral forces in the material that start to play a major role at the boundaries of the material.

Flat area: In this area, the metal stretches on the top surface, creating a "thinning" appearance on top, while the bottom surface contracts, creating a "thickening" appearance on the bottom.

Photo (B) in Figure 28 shows the bent metal from the side. The symbol R refers to the radius of bending, while the symbol θ indicates the measured angle.



(A)



(B)

Figure 28 Different views of bent flat bars. See text for description.

Table 23 The different parameters in the equation describing spring-back.

Parameter	Description	Value	
ϕ	The angle for bending as shown in Figure 29		
θ	The bend angle also shown in Figure 29		
L	The distance between the V-block's contact points	0.150	m
μ	Poisson's ratio	0.285	
S	Maximum yield stress	370	$MN.m^{-2}$
t	Thickness	0.025	m
E	Young's modulus	208	$GN.m^{-2}$

7.2.2 Equation for the bending process

Spring-back for simple bending is given by the following equation (see Appendix A.3):

$$\phi = \frac{180^0}{\pi} \cdot \frac{3L\left(1-\mu^2\right)S}{tE} \qquad (7.1)$$

The different parameters are described in Table 23 and illustrated in Figure 29.

Equation 7.2 describes the relationship between movement and the resulting angle, ignoring spring-back:

$$\phi = 360^0 - \frac{180}{\pi} \cdot 4 \cdot \arctan\left(\frac{L}{\Delta\rho}\right) \qquad (7.2)$$

This equation is obtained through simple trigonometric relationships.

After combining equations 7.1 and 7.2 the following equation results:

$$\phi = 360^0 - \frac{180}{\pi}\left(4 \cdot \arctan\left(\frac{L}{\Delta\rho}\right) - \frac{3L\left(1-\mu^2\right)S}{tE}\right) \qquad (7.3)$$

It describes the relationship between displacement and the bending angle.

The bending angle θ can now be derived:

$$\phi = \frac{180}{\pi}\left(4 \cdot \arctan\left(\frac{L}{\Delta\rho}\right) - \frac{3L\left(1-\mu^2\right)S}{tE}\right) - 180^0 \qquad (7.4)$$

Equation (8.4) was derived with the following assumptions in mind:

1) The radius of bending is much larger than the thickness of the material under bending. This is an important assumption, because most equations during the derivation process in Appendix A.3 were simplified based on this assumption. This is a major shortcoming in the equation, since the radius of bending is of the same order as the thickness of the bent metal as shown in Figure 28 (B). It is not possible to obtain an analytical solution without this assumption.

2) The thickness of the material stays constant during bending. This is also false, as shown in Figure 28 (A). It is clear that the thickness did not stay constant during

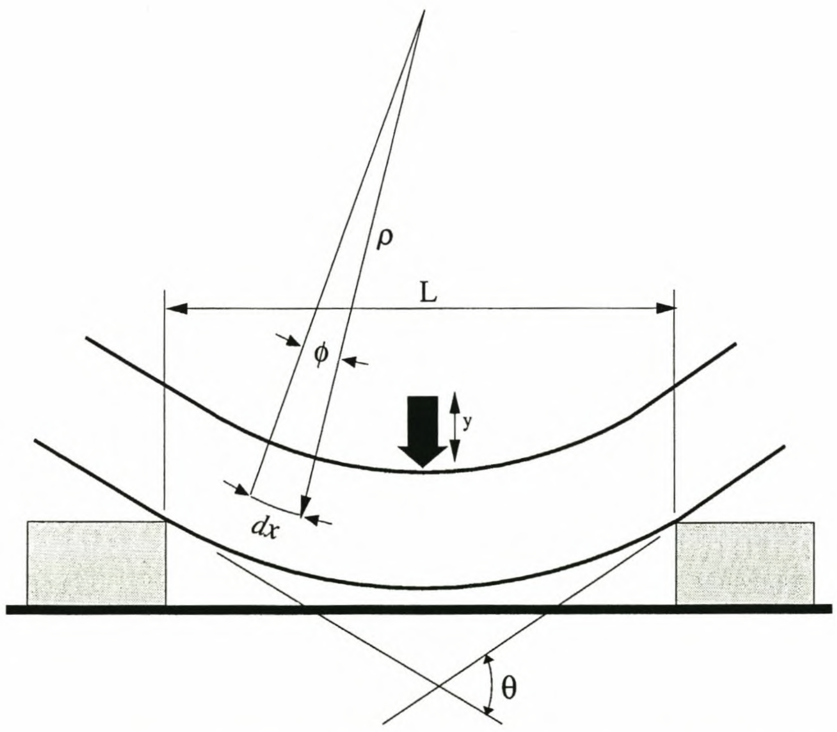


Figure 29 Diagrammatic explanation of the parameters used in the bending equations

bending.

3) The width stays constant. The bent metal in Figure 28 (A) again negates this assumption, as the lateral forces play a major role in the evagination area, causing the width to change.

4) Metal in the area where bending occurs is nearly immediately in the plastic region. The equation does not account for areas that are partly in the elastic region.

5) A point load is used for bending. The actual blade has a radius of 25mm.

These assumptions also imply that the bend has the form of a perfect circle. This is not true if the blade's displacement from the zero position is large in comparison with the distance between the two contact points of the V-block.

Figure 30 shows the difference between the derived equation and the actual data points obtained during testing.

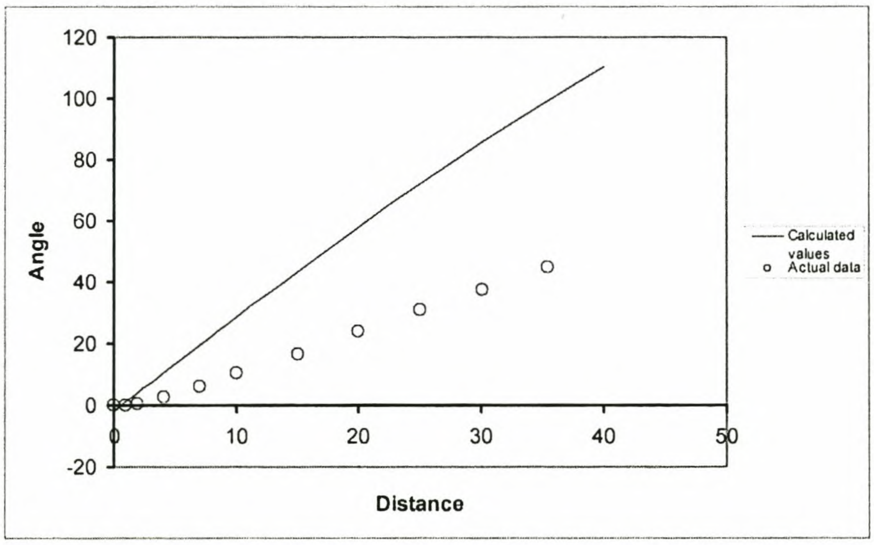


Figure 30 Comparison of the derived equation with the actual data for a 25mm thick piece of metal

Table 24 Properties of different equation describing the angle-displacement relationship.

Equation	MS	Trends
$y = ax$	2.765	Definite trends.
$y = ax^2 + bc$	1.062	Definite trends.
$y = ax^3 + bx^2 + cx$	0.3635	Definite trend in the 0-10mm region.
$y = ax^4 + bx^3 + cx^2 + dx$	0.1769	Definite trend in the 0-10mm region.
$y = ax + b - be^{-cx}$	0.0696	No trends.

MS – Mean square of the residuals
Trends – Trends observed in the signs of the residuals (see Appendix H)

It is obvious from this graph that equation 7.4 is inadequate for use in this application. The equation describing the relationship between the resulting angle and the displacement of the bending angle was therefore empirically devised as shown in the next section.

7.3 EMPIRICAL SOLUTION FOR THE ANGLE-DISPLACEMENT RELATIONSHIP

The empirical solution was devised by measuring the resulting angles for a number of displacements. Results of these measurements are discussed in Appendix H. Only one replication was done. Further experimentation is required to reach a definite conclusion regarding these results.

Experiments were conducted with a number of equations to fit the data. Table 24 shows the properties of the different equations tested.

The linear equations did not succeed in estimating angles within acceptable values over the entire measured range. The non-linear equation,

$$y = ax + b - be^{-cx} \qquad (7.3)$$

fits the data extremely well. The steel was tested only in the linear area of the plastic region of deformation. A quadratic term was added to model non-linear behaviour better when large angles were bent:

$$y = ax^2 + bx + c - ce^{-dx} \qquad (7.4)$$

The term was added only for compatibility to future programs. It did not have much influence on the results as shown in Appendix I.

No software tool commonly available is capable of doing non-linear regression. A program capable of computing the parameters of equation (7.4) for each type of steel, was therefore developed.

Figure 31 shows a graph of the non-linear equation and the practical data. Two regions can be identified:

Elastic region: Spring-back plays a major role in this region, because most fibres in the steel are still elastically deformed.

Plastic region: Most fibres in the steel are plastically deformed in this region.

The behaviour of the angle-displacement ratios can now be explained as follows:

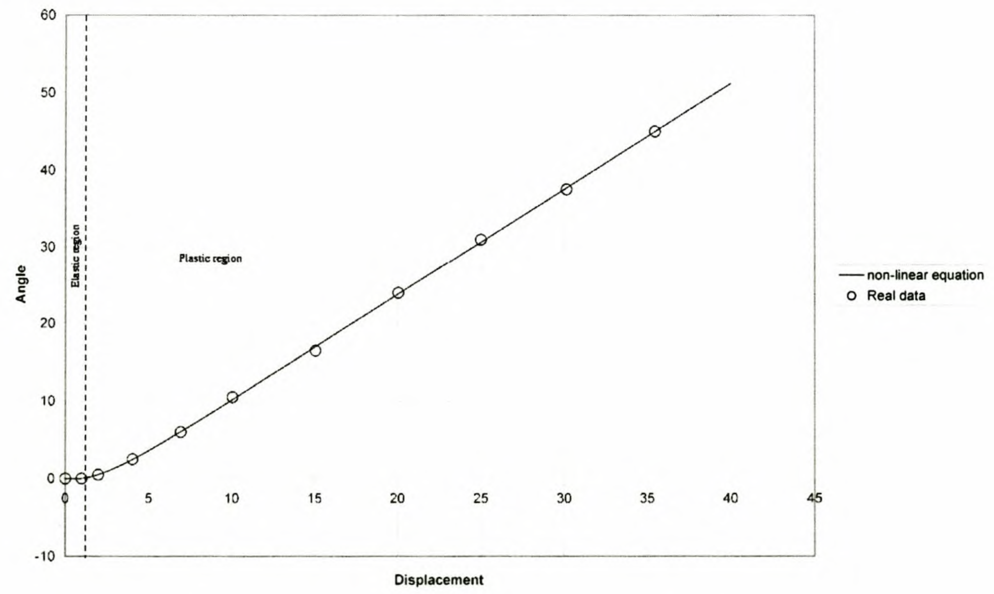


Figure 31 Graph of $y=ax+b-be^{-cx}$ and the practical data

- The function forms a linear equation ($ax+b$) in the plastic region. It can be assumed that spring-back is virtually constant in this region, which also explains why the constant b in the equation has a negative value (causing the negative y-axis interception when the angle is plotted on the y-axis).

- The contribution of plastic forces decreases in the elastic region, explaining the curvature, which has the form of an exponential equation in this region.

After the derivation of equation (7.4), it was tested practically. The results are listed in Appendix I. A summary of the results appears in Table 25.

Nearly all measurements were within 0.5^0 of the target. A number of factors caused inaccuracies in the bending process:

1) The process for zeroing the machine does not work properly. Currently the operator has to guess when the machine is correctly set up for zeroing. This had a significant effect on the results.

2) Some of the results will improve if the correct compensation factor is chosen. A compensation of 100 was used in all tests to evaluate the results as is.

3) While tests were conducted, experiments were done to find the optimum time constant for the press. These tests will also have an influence on the results.

Table 25 Summary of results obtained with measurements done on different types of steel

	Predicted measurement	Actual measurement	Difference
25x80mm:			
15^0			
$\bar{x}$	15.42	15.27	0.233
σ	0.252	0.116	0.046
max	16.65	16	0.65
30^0			
$\bar{x}$	30.24	30.05	0.173
σ	0.001	0.04	0.0216
max	30.37	30.33	0.57
30x100mm			
15^0			
$\bar{x}$	15.03	15.18	0.4
σ	0.044	0.141	0.58
max	15.47	15.5	0.58
12.5x100mm			
30^0			
$\bar{x}$	29.95	29.66	0.28
σ	0.023	0.08	0.018
max	30.13	30	0.37
45^0			
$\bar{x}$	44.87	42	2.87
σ	0.021	0	0.021
max	45.02	42	3.02
15x80mm			
15^0			
$\bar{x}$	14.85	14.83	0.023
σ	0.0026	0.083	0.092
max	14.9	15	0.37
30^0			
$\bar{x}$	29.87	30	-0.126
σ	0.008	0	0.008
max	29.93	30	-0.07

CHAPTER 8

CONCLUSIONS AND RECOMMENDATIONS

8.1 CONCLUSIONS

Mechatronics is an exciting new field that utilise the benefits of engineers educated in more than one discipline. Improved insight is achieved in design problems leading to more effective solutions of complex problems. Multi-discipline training is also a pitfall for mechatronic engineers. It can easily leads to establishing only an overview of the different disciplines. In a development environment where it is important to achieve stable designs, it is also important to understand the underlying theory.

It was found that it is unrealistic to expect a working device from a laboratory, to work in a robust environment with huge EMC problems. For example, a week was spent to correct an earth leak found on the bending machine, which was caused by faulty wiring in the factory. Many problems were encountered which were related to EMC.

8.1.1 Control program

The control program was written in LabVIEW 4.1. LabVIEW does not have very good database handling capabilities, which made programming very difficult. Therefore only the control program was written in LabVIEW.

LabVIEW is very easy to use in control applications once it is mastered. It requires a new way of programming shifting from *code flow* to *data flow* programming. LabVIEW runs under Windows 95, Windows NT, Mac OS and Unix. Windows NT was used because of its stability and availability. Programs running under any Windows environment cannot control real time control systems with sample periods of faster than 1ms. External circuitry such as PC boards is required to achieve faster sampling periods. Fortunately this application did not need sample periods of such high accuracy.

LabVIEW has a very nice graphical user interface, which forces programmers to develop user-friendly programs. Active-X implemented in LabVIEW from LabVIEW 5.0 onwards, opens many doors such as Internet control to programmers.

8.1.2 Regression Analyser

The regression analyser was written in Delphi. It serves as a tool for devising the parameters used in the bending equation. The results obtained with the analyser were compared with results obtained in *Statistica*. *Statistica* produced better results. The analyser should only be regarded as a rough tool for devising the parameters of the bending equations. Parameters should be verified in applications like Excel before implementation to ensure that logical parameters are used.

It remains to be debated if ROVIC will use the regression analyser. It was found that it is faster for the operator to use an old equation of a different flat bar instead of devising the parameters for each type of metal. The operator only changes the compensation value until the desired angle is obtained.

8.1.3 Electronic hardware

Commercially available hardware was preferably used except when solutions were too expensive or functions unavailable. Three PC boards were developed in this project:

1) Counter circuitry.

2) Connection circuitry.

3) Driver circuitry.

Counter circuitry

This PC board was developed due to the unavailability of an inexpensive solution. Much trouble was experienced with instabilities on the counter circuitry. A number of redesigning efforts were conducted. The problems were remedied after adding an own power supply for the control circuitry and the use of appropriate filters in the control program.

A position read accuracy of 0.03mm was achieved with the encoders while angles were bent at an accuracy of roughly 0.5^o.

Connection circuitry

The connection circuitry was designed to enable the operator to disconnect the computer, and still be able to operate the bending press by means of the hand controller. This circuitry is also responsible for handling the *emergency stop* signal. When the *Emergency* button is pressed, the bending blade retracts to the top of its stroke. This is a huge improvement over the emergency buttons used in the previous projects conducted by the mechatronics group, where the signal was relayed to the computer, which handled the emergency procedures. The current emergency operation is totally independent of the computer and will not fail should the computer crash.

Driver circuitry

This was the third PC board developed for project. It caused noise on the power supply lines, which caused incorrect behaviour on the other PC boards. The error was corrected by adding by-pass capacitors to the PC boards and giving the counter circuitry its own power supply. This circuitry was designed to give precedence to the upward direction ensuring that the bending blade will always retract when the safety button is pressed regardless of any other control signal.

PC36C IO boards

These IO boards were purchased from Eagle, a local electronic manufacturing company. They were shipped with LabVIEW VIs and appropriate device drivers. They are easy to use and controlled by LabVIEW.

8.2 RECOMMENDATIONS

The control program

The interface with dials is not needed. It would be much more meaningful to write an interface in Delphi or a similar application and use *Active-X* to connect it to LabVIEW. This solution gets rid of the dials, making space for more functions like a bend sequence editor, which can be evoked by a menu system. Better control can be achieved over execution and editing of bend sequence programs.

The file handling capabilities of applications like Delphi is better than those of LabVIEW. These capabilities could also be implemented in LabVIEW via *Active-X*, which will improve the database functionalities of the LabVIEW programs. LabVIEW should thus serve as a platform for the execution of a Delphi module, which will provide excellent control over the hardware to Delphi.

Control system

Much can be done to improve the control system. LabVIEW is only capable of generating signals larger than 1ms accurately. Taking the overhead programming into account, it is evident that program execution is slower which leads to incorrect results. This has an influence on the behaviour of the hydraulic valve.

It would be better to embed the control system on the electronic hardware, either by means of a CPU or commercially available boards. This will ensure that the system behaviour is more predictable.

Bending equations

No data was obtained of the behaviour of the angle-displacement relationship in the plastic region when different blades, steel or V-blocks were used. The equation will have different forms for each of these variables.

Further investigation should be done on the behaviour of the metal during bending to develop a generic model, which will also take the non-linear upper area of the plastic region into account.

REFERENCES

Acar, 1996	Acar, M. and Parkin R.M. Industrial electronics. *IEEE Transactions on Industrial Electronics*, 43(1):108, February 1996.
Anorad, 1996	Anorad Corp. Stretching piezo technology. *Machine Design*, 68(8):146, April 1996.
Auslander, 1996	Auslander D.M. What is mechatronics? *IEEE/ASME Transactions on Mechatronics*, 1(1):5 – 9, March 1996.
Avner, 1974	Avner, S.H. *Introduction to Physical Metallurgy*, page 37. McGraw-Hill Book Company, New York, USA, second edition, 1974.
Bell, 1987	Bell, Doug. *Software engineering: a programming approach*, page 7. Prentice Hall International (UK) Ltd, London, 1987.
Bolton, 1995	Bolton, W. *Mechatronics: electronic control systems in mechanical engineering*, page 1. Addison Wesley Longman Ltd, Essex, UK, 1995.
Brown, 1994	Brown, M. & Harris, C. J. *Neural networks for modelling and control. Advances in intelligent control*, pages 17 – 56, 1994.
Coetzee, 1997	Coetzee, J.H. *Applications in computerised automation using linear displacement transducers*, page 34. Master's thesis, University of Stellenbosch, 1997.
Comerford, 1994	Comerford, R. Mecha ... what? *IEEE Spectrum*, 1(1):46, August 1994.
Crawford, 1987	Crawford R.J. & Benham, P.P. *Mechanics of Engineering Materials*, page 124. John Wiley and Sons, New York, USA, 1987.
Dekey, 1998	Dekey, S. *Making Windows NT a real-time solution – a technical overview*. http://www.natinst.com, 1998.

Eagle, 1999	Eagle Technology. *Support, drivers and download page.* http://www.eagle.co.za, 1999.
Gallant, 1987a	Gallant, A.R. *Non-linear statistical models*, pages 26 – 30. John Wiley and Sons, New York, 1987.
Gallant, 1987b	Gallant, A.R. *Non-linear statistical models*, pages 8 – 17. John Wiley and Sons, New York, 1987.
Grimble, 1994	Grimble, M.J.; *Robust industrial control*, pages 7 – 12, 105. Prentice Hall International, Englewood Cliffs, New Jersey, USA, 1994.
Harashima, 1996	Harashima, Fumio Mechatronics – What is it Why and How? *IEEE Transactions on Mechatronics*, 1(1):1 – 4, March 1996.
Hewit, 1996a	Hewit, J.R. & Bouazza-Marouf, K. Practical control enhancement via mechatronic design. *IEEE Transactions on Industrial Electronics*, 43(1):16 – 22, February 1996.
Hewit, 1996b	Hewit, J.R. & King, T.G. Mechatronic design for product enhancement. *IEEE/ASME Transactions on Mechatronics*, 1(2):111 – 119, June 1996.
Hibbeler, 1997	Hibbeler, R.C. *Mechanics of Materials*, page 71. Prentice Hall, Upper Saddle River, New Jersey, USA, third edition, 1997.
Huang, 1996	Huang, G.Q. *Design for X: Concurrent engineering imperatives,* page 10, Chapman and Hall, London, 1996.
Iserman, 1996a	Iserman R. Modelling and design methodologies for mechatronic systems. *IEEE/ASME Transactions on Mechatronics*, 1(1):16 – 28, March 1996.
Iserman, 1996b	Isermann, I. On the design and control of mechatronic systems – A survey. *IEEE transactions on industrial electronics*, 43(1):4 – 15, February 1996.

Jürgens, 1998	Jürgens, M.L.J. *An application of mechatronics in manufacturing with object-orientated programming in a windows environment, page 115.* Master's thesis, University of Stellenbosch, 1998.
Kamm, 1996	Kamm L.J. *Understanding electro-mechanical engineering: an introduction to mechatronics*, page 111, IEEE, New York, USA, 1996.
Kanjilal, 1995	Kanjilal, P.P. *Adaptive prediction and predictive control*, pages 1 – 5. Peter Peregrinus Ltd, Stevenage, UK, 1995.
Karnopp, 1975	Karnopp, D. & Rosenberg, R. *System dynamics: A unified approach*, page 27. John Wiley & Sons, New York, USA, 1975.
Kyura, 1996a	Kyura, N. and Ohio H. Mechatronics – an industrial perspective. *IEEE/ASME Transactions on Mechatronics*, 1(1):10 – 15, March 1996.
Kyura, 1996b	Kyura, N. The development of a controller for mechatronic equipment. *IEEE Transactions on Industrial Electronics*, 43(1):31, February 1996.
Linux, 1999	Linux Resource Exchange. *Operating system comparison chart.* http://www.linuxrx.com, 1999.
Luo, 1996	Luo, R.C. Sensor technologies and microprocessor issues for mechatronic systems. *IEEE transactions on Mechatronics*, 1(1):39 – 49, March 1996.
Marciniak, 1992	Marciniak, Z. *Mechanics of Sheet Metal Forming*, page 27. Edward Arnold, London, United Kingdom, 1992.
Minnaar, 1994	Minnaar, S. *Concurrent engineering in South Africa: A practical strategy and support system.* Master's thesis, page 31, University of Stellenbosch, 1994.
Monkman, 1992	Monkman, G.J. A rapid response thermal tactile sensor. *Mechatronics – The integration of engineering design*, page 81, Mechanical Engineering Publications Limited, 1992.

National Instruments, 1999a National Instruments. *Optical encoder fundamentals.* http://www.ni.com/sensors/optical.htm, 1999a.

National Instruments, 1999b National Instruments. *Quadratic encoder fundamentals.* http://www.ni.com/sensors/quad.htm, 1999b.

Ogata, 1990 Ogata, K. *Modern control engineering*, page 736. Prentice Hall International, Englewood Cliffs, New Jersey, USA, second edition, 1990.

Prosise, 1995 Prosise, Jeff. *What is an operating system?* http://www.zdnet.com/pcmag, 1995.

Ruocco, 1987 Ruocco, S.R. *Robot sensors and transducers*, page 14. Halsted Press, New York, 1987.

Rushforth, 1992 Rushforth, E.J., West, T. C. & King P. D.; Sensor integration in high speed packaging. *Mechatronics – The integration of engineering design*, pages 53 – 59, Mechanical Engineering Publications Limited 1992.

Schweitzer, 1996 Schweitzer, G. Mechatronics for the design of human-oriented machines. *IEEE/ASME Transactions on Mechatronics*, 1(2): 120 – 126, June 1996.

Shorter, 1992 Shorter, R.J. Development of a new diagnostic system by Ford Motor Company and GenRad. *Mechatronics – The integration of engineering design*, pages 37 – 40, Mechanical Engineering Publications Limited, 1992.

Slotine, 1991 Slotine, J.-J. & Weiping, L.; *Applied nonlinear control*, pages 4 – 12. Prentice Hall International, Englewood Cliffs, New Jersey, USA, 1991.

Syan, 1994a Syan, C.S. & Menon, U. *Concurrent engineering: Concepts, implementation and practice*, page 12. Chapman and Hall, London, 1994.

Syan, 1994b	Syan, C.S. & Menon, U.; *Concurrent engineering: Concepts, implementation and practice*, pages 4 – 8. Chapman and Hall, London, 1994.
Tan, 1998	Tan, K.K. Various developments in mechatronics in Asia. *Mechatronics*, 8(7):777 – 791, October 1998.
Thoma, 1975	Thoma, J.U. *Introduction to bond graphs and their applications*, page x. Pergamon Press, New York, USA, 1975.
Thornley, 1992	Thornley, J.K., King, T. G. & Preston, M. E. The design of mechanical amplifiers using piezo-electric multilayer devices for use as fast actuators. *Mechatronics – the integration of engineering design*, pages 69 – 74, 1992.
Tomkinson 1996a	Tomkinson, D. & Horne, J. *Mechatronics engineering*, page 16. McGraw-Hill, New York, 1996.
Tomkinson 1996b	Tomkinson, D. & Horne, J. *Mechatronics engineering*, page 44. McGraw-Hill, New York, 1996.
Van Amerongen, 1996	Van Amerongen, Job et al. Mechatronics in the Netherlands. *IEEE/ASME Transactions on Mechatronics*, 1(2):106 – 110, June 1996.
Van Brussel, 1996	Van Brussel, Hendrik M.J. Mechatronics – a powerful concurrent engineering framework. *IEEE/ASME Transactions on Mechatronics*, 1(2):127 – 136, June 1996.
Wakerly, 1990	Wakerly, J.F. *Digital design principles and practices*, pages 349 – 351. Prentice Hall, Englewood Cliffs, New Jersey, USA, 1990.
Youcef-Tourmi, 1996	Youcef-Tourmi, K. Modeling, design and control integration: a necessary step in mechatronics. *IEEE/ASME Transactions on Mechatronics*, 1(1):29 – 38, March 1996.

BIBLIOGRAPHY

Coetzee, 1997	Coetzee, J.H. *Applications in computerised automation using linear displacement transducers*. Master's thesis, University of Stellenbosch, 1997.
Jürgens, 1998	Jürgens, M.L.J. *An application of mechatronics in manufacturing with object-orientated programming in a windows environment.* Master's thesis, University of Stellenbosch, 1998.

APPENDIX A

DESIGN CALCULATIONS

A.1 CALCULATIONS FOR THE ENCODER

The calculations for the geared shaft encoder will be discussed in this section. The configuration is shown in Figure A 1. The encoder operation is discussed in section 6.4.4. A picture of the encoder and cable connection is shown and discussed in appendix G.

In (A) the smaller gear has a circumference of C_{Small}, the larger gear has a circumference of C_{Large} and the maximum distance to be measured is l. The configuration in (B) has the same parameters except that the encoder is directly connected to the larger gear and does not have a small gear. θ in (C) is the angle between the phases of the pulses of channels A and B. It usually has a value of 90^0. l_s is the distance that the cable moves between pulses. ϕ is the

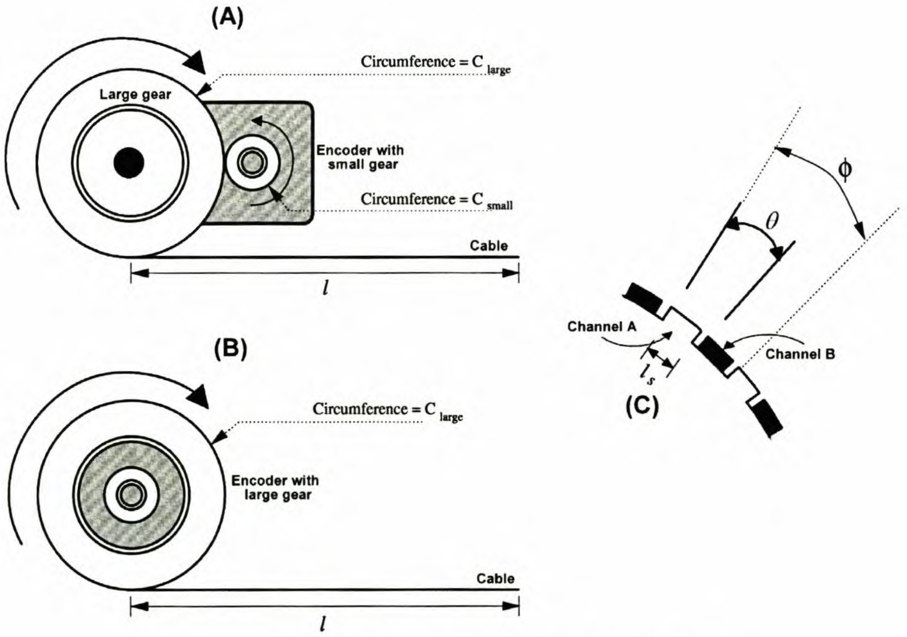


Figure A 1 Block diagrams of then encoder gear mechanism.

angle between two consecutive pulses of a specific channel.

The resolution and the linearity for the configuration in (B) are calculated as follows:

$$Number\ of\ pulses\ per\ revolution \equiv N \tag{A.1}$$

$$Resolution = l_s = \frac{C_{large}}{N} \tag{A.2}$$

$$Linearity = \frac{Resolution}{Total\ distance\ moved} = \frac{l_s}{l} \tag{A.3}$$

If $C_{Large} = l$, equation (A.3) is reduced to:

$$Linearity = \frac{l_s}{C_{large}} = \frac{C_{large}}{N \cdot C_{large}} = \frac{1}{N} \tag{A.4}$$

The configuration with gears in (A) can only be used with incremental encoders, because potentiometers and absolute encoders make a discrete switch from a maximum value to a minimum value each time the encoder completes a revolution. This complicates position calculations in the control program forcing the program to keep track of each revolution of the potentiometer. The incremental encoder does not pose this problem. The counter keeps counting after the completion of a revolution. The maximum count value is determined by the counter modulus. Linearity calculations for the geared incremental encoder are as follows:

$$Gear\ ratio = r = \frac{C_{large}}{C_{small}} \tag{A.5}$$

$$Number\ of\ pulses\ per\ revolution \equiv N \tag{A.6}$$

$$\mathrm{Re}\,solution = l_s = \frac{C_{small}}{N} = \frac{C_{large}}{N \cdot r} \tag{A.7}$$

$$Linearity = \frac{Resolution}{Total\ distance\ moved} = \frac{l_s}{l} \tag{A.8}$$

If $C_{Large} = l$, equation (A.8) is reduced to:

$$Linearity = \frac{l_s}{C_{large}} = \frac{C_{large}}{r \cdot N \cdot C_{large}} = \frac{1}{N \cdot r} \tag{A.9}$$

Equation (A.9) gives a much better result than the set-up without the gear. Inaccuracies due to slip in the gears were not included in these calculations.

A.2 THE CLOCK FREQUENCY REQUIRED FOR THE HP2020 COUNTER

There are two design parameters according to the HP2020 data sheets:

1) $t_e \geq 3t_{CLK}$

2) $t_{es} \geq t_{CLK}$

The parameters have the following definitions:

t_e	— Encoder pulse width
t_{es}	— Encoder state period
t_{CLK}	— Clock pulse period

t_e is calculated as follows (see Figure A 2 for the timing diagram):

Distance moved in one clock pulse

$$= \frac{Number\ of\ pulses\ per\ revolution\ of\ encoder\ shaft}{2 \times Distance\ moved\ per\ revolution\ of\ encoder\ shaft} \tag{A.10}$$

or for a worst-case scenario:

$$= \frac{Maximum\ count}{2 \times Maximum\ distance\ moved}$$
$$= \frac{n}{2l} \tag{A.11}$$

Speed of movement:

$$= \frac{Distance\ moved}{Time\ required\ for\ movement}$$
$$= \frac{l}{t} \tag{A.12}$$

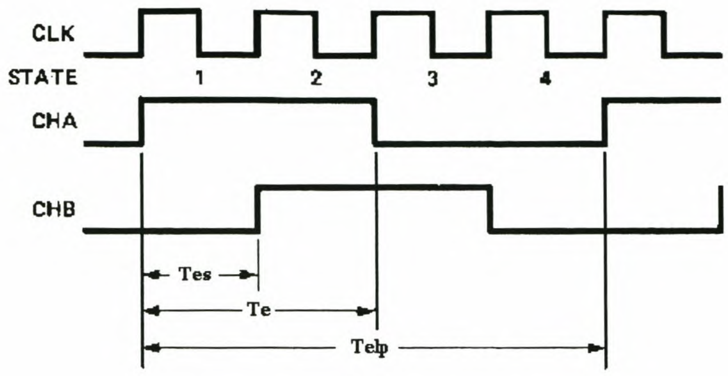


Figure A 2 Signal diagram of the clock signal for the HP2020

t_e can now be calculated from equation (A.11) and (A.12):

$$\begin{aligned} t_e &= \frac{\textit{Distance moved in one clock pulse}}{\textit{Speed of movement}} \\ &= \frac{n}{2t} \end{aligned} \qquad \text{(A.13)}$$

The value of t_{es} is determined by the angle θ between channels A and B. The angle is normally set at 90^0, but it is possible that the angle has a smaller value when the encoder changes direction. A conservative value of 15^0 was consequently used.

t_{es} is calculated as follows from equation (A.13):

$$\begin{aligned} t_{es} &= \frac{\theta}{360^0} \times t_e \\ &= \frac{\theta n}{360^0} \cdot 2t \end{aligned} \qquad \text{(A.14)}$$

The results are listed in Table A 1. A clock frequency of 4MHz has a period of 250ns, which is within the range specified at the start of this section.

Table A 1 The calculated values for the clock frequency of the HP2020

N	l	T	θ	$\frac{t_e}{3}$	t_{es}
2^{20}	1500mm	5s	15^0	264.91ns	298.02ns

A.3 SPRING-BACK COMPENSATION

A model for spring-back is derived from basic principles to develop a better insight into the mechanism of bending. A model for only simple bending is developed. Non-linear terms are introduced to the equation as the number of contributing factors increases.

The following assumptions were made for simple bending:

1) In simple bending the bending radius is much larger than the thickness of the sheet of metal being bent.
2) The thickness of the material stays constant during bending.
3) The width of the material stays constant in all the surfaces during bending.
4) Only forces and moments in the bending regions will be considered and not in the straight parts.
5) The shear forces in the radial plane and the gradient surfaces are considered as zero.
6) The neutral surface where the length of the fibres stay constant lies on the middle surface.

The theory will be discussed analogous to the derivations made by Marciniak (1992:20).

A.3.1 Strain

Definitions

The term *stress* in this section refers to the internal resistance or forces that oppose the external forces that cause bending. *Total strain* refers to the total change in dimension of the material in a specific direction. *Unit strain* is the strain per unit length in that specific direction (Avner, 1974:37). Stress is usually denoted by σ and strain by ε.

In most literature the word *strain* refers to unit strain. The same protocol will be adopted here.

General model for strain

Suppose a piece of sheet metal is bent as shown in Figure A 3, with fibres *AB* above the neutral surface, *EF* on the neutral surface and *CD* below the neutral surface. When the sheet is bent as shown, *AB* lengthens to a length of *A'B'* and *CD* shortens to *C'D'*, while *EF* can either lengthen or shorten to *E'F'*. *EF* has a length of l_0 before bending and a length of l_S after bending.

The sheet has a width of *b*.

It is now possible to define strain mathematically as follows:

$$\varepsilon = \frac{\Delta l}{l} \tag{A.15}$$

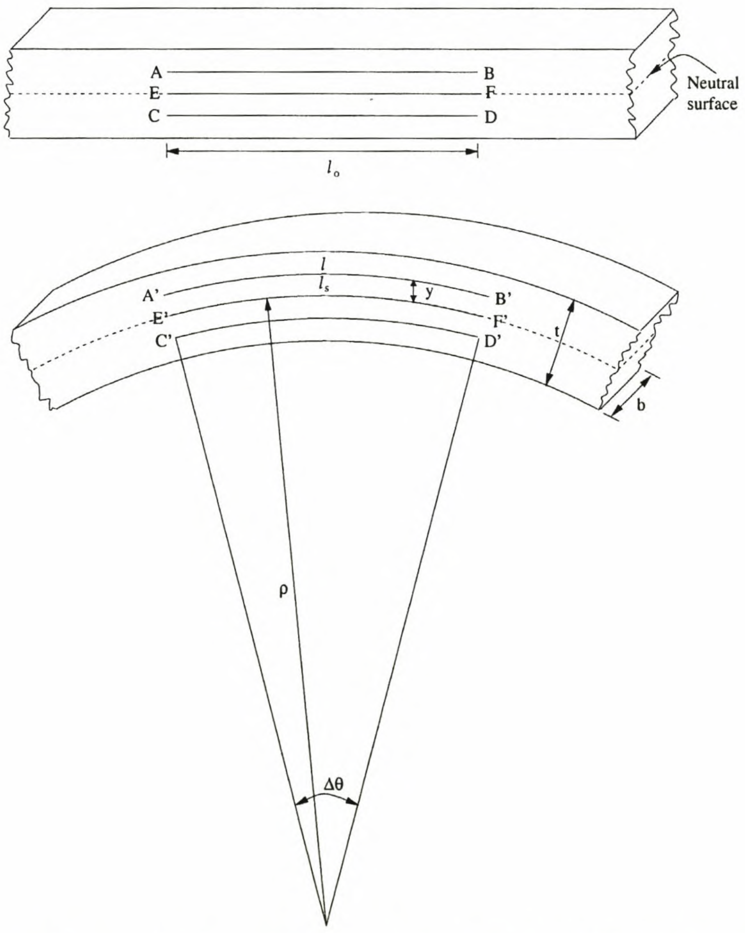


Figure A 3 The parameters used during the derivation of the bending equation

Δl is the difference in length after a fibre is stretched or compressed. If we let *Δl→0,* then equation A.15 can be denoted as follows according to (Hibbeler ,1997:71):

$$d\varepsilon = \frac{dl}{l} \qquad \text{(A.16)}$$

The same method as that followed by Marciniak (1992:27) is now used to integrate over the length to find true or natural strain along *l*. The following result is then obtained on the neutral surface:

$$\begin{aligned} \varepsilon &= \int_0^{l_s+\Delta l_s} \frac{1}{l} dl \\ &= \ln\left(\frac{l_s + \Delta l_s}{l_0}\right) \\ &= \ln\left(\frac{l}{l_0}\right) \end{aligned} \qquad \text{(A.17)}$$

with *l* chosen as $l=l_s+\Delta l_s$. It is also possible to write l_s as follows:

$$\frac{l_s}{\rho} = \sin \Delta\theta \approx \Delta\theta \qquad \text{(A.18)}$$

and

$$\frac{l}{\rho + y} = \sin \Delta\theta \approx \Delta\theta \qquad \text{(A.19)}$$

with $\Delta\theta \rightarrow 0$. Using A.18 and A.19 the following result is obtained:

$$\begin{aligned} l &= \frac{l_s}{\rho}(\rho + y) \\ &= l_s\left(1 + \frac{y}{\rho}\right) \end{aligned} \qquad \text{(A.20)}$$

Substituting A.18 in A.19, strain is now rewritten as follows:

$$\begin{aligned} \varepsilon &= \ln\left(\frac{l_s\left(1+\frac{y}{\rho}\right)}{l_0}\right) \\ &= \ln\left(\frac{l_s}{l_0}\right) + \ln\left(1+\frac{y}{\rho}\right) \end{aligned} \qquad \text{(A.21)}$$

This result is a general solution for the calculation of strain in fibres. l_s is the length of the fibres on the middle surface ≠ neutral surfaces in the deformed state. l_0 is the length of a fibre before deformation and l is the length of a fibre after deformation.

A.3.2 Forces developed during bending

The sum of the forces developed during bending must be nought, because there can be no resultant force:

$$\begin{aligned}\sum F_i &= \int_A dF \\ &= \int_A \sigma dA \\ &= b\int_{-\frac{t}{2}}^{\frac{t}{2}} \sigma dy = 0 \qquad \text{(A.22)}\end{aligned}$$

The sum of the moments is M, because there is a resulting moment during bending:

$$\begin{aligned}\sum M &= \int_A y dF \\ &= \int_A \sigma y dA \\ &= b\int_{-\frac{t}{2}}^{\frac{t}{2}} \sigma y dy = M \qquad \text{(A.23)}\end{aligned}$$

These results will be used to develop a model to describe the behaviour of sheet metal during simple elastic bending.

A.3.3 Simple elastic bending

According to Marciniak the stress-strain curve for material usually takes the following form (Marciniak 1992:27):

$$\sigma = K'\varepsilon^n \qquad \text{(A.24)}$$

Equation A.21 can be simplified during simple bending to:

$$\varepsilon = \frac{y}{\rho} \qquad \text{(A.25)}$$

$l_S=l_0$ and $ln\ (1 + x) \approx x$, when x is very small. Combining A.24 and A.25 the following is obtained:

$$\sigma = K'\left(\frac{y}{\rho}\right)^n \tag{A.26}$$

Substituting A.25 in A.23:

$$\begin{aligned} M &= b\int_{-\frac{t}{2}}^{\frac{t}{2}} K'\left(\frac{y}{\rho}\right)^n y\,dy \\ &= \frac{bK'}{\rho^n}\int_{-\frac{t}{2}}^{\frac{t}{2}} y^{n+1}\,dy \\ &= \frac{bK'}{\rho^n}\frac{1}{n+2}\Bigg|_{-\frac{t}{2}}^{\frac{t}{2}} \\ &= \frac{K'}{\rho^n}\frac{bt^{n+2}}{(n+2)(2)^{n+1}} \\ &= \frac{K'I_n}{\rho^n} \end{aligned} \tag{A.27}$$

with $I_n = \dfrac{bt^{n+1}}{(n+2)(2)^{n+1}}$. In simple bending, *K'*=*E'* with $E' = \dfrac{E}{1-\mu^2}$. Young's modulus is denoted by *E* and Poisson's ratio by μ.

By taking n=1 for simple elastic bending, and using A.26, A.27 is simplified to:

$$\frac{M}{\dfrac{bt^{n+2}}{(n+2)2^{n+1}}} = \frac{E}{\rho^n\left(1-\mu^2\right)} = \frac{\sigma}{y^n} \tag{A.28}$$

$$\frac{M}{\frac{1}{12}bt^3} = \frac{E}{\rho\left(1-\mu^2\right)} = \frac{\sigma}{y} \tag{A.29}$$

This is the same result obtained by Benham (1987:124) where the problem is presented in a much simplified form with $\mu = 0$.

S is the maximum stress in the metal before plastic deformation starts. The stress of the fibres on the surface at $y = \frac{t}{2}$ will be *S* during simple bending (refer to Figure A 3). Only the fibres on the surface has a value of *S* thus setting σ at the surface to *S*. Substituting this result in equation (A.27) and taking *n=0* for simple plastic bending, we obtain the following:

$$\frac{M_p}{\dfrac{bt^{n+2}}{(n+2)2^{n+1}}} = \frac{\sigma}{y^n}$$

$$\Rightarrow \quad M_p = \frac{bt^2 S}{4} \qquad \text{(A.30)}$$

A.3.4 Spring-back after simple bending

For elastic spring-back, we only need to calculate the elastic behaviour on the return path AB as illustrated by the Figure A 4.

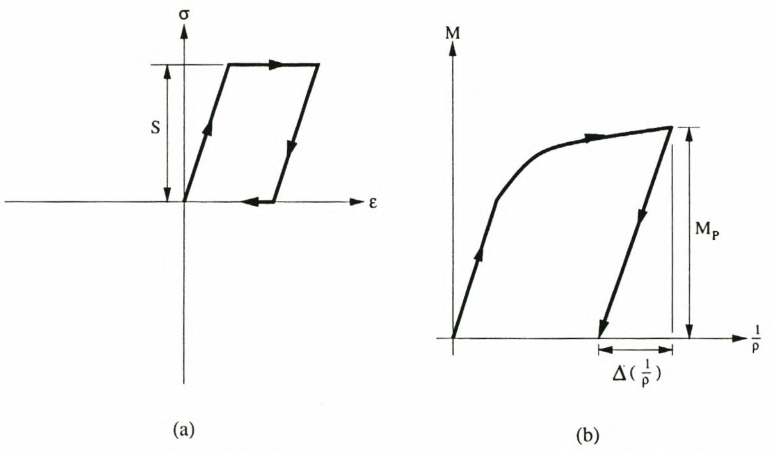


Figure A 4 Strain-stress diagram (Adapted from Marciniak 1992:27)

Equation (A.29) can be rewritten as follows:

$$\frac{\Delta M}{\dfrac{1}{12}bt^3} = \Delta\left(\frac{1}{\rho}\right)\cdot E' = \frac{\Delta\sigma}{y} \qquad \text{(A.31)}$$

Substitute the maximum moment from A.30 in A.38 (See Figure A 4):

$$\Delta M = \frac{Sbt^2}{4}$$

$$\Rightarrow \quad \Delta\left(\frac{1}{\rho}\right) = \frac{\dfrac{Sbt^2}{4}}{\dfrac{1}{12}bt^3 E'}$$

$$= \frac{3S}{E't} \qquad \text{(A.32)}$$

and:

$$\Delta\sigma = \frac{\frac{Sbt^2}{4}\frac{t}{2}}{\frac{1}{12}bt^3}$$

$$= \frac{3}{2}S \tag{A.33}$$

In Figure A 5, the following relationship exists:

$$\frac{L}{\phi} \approx \frac{dx}{d\phi}$$

$$\Rightarrow \phi = L\frac{d\phi}{dx} \tag{A.34}$$

It should be clear that this assumption is true only for very large values of ρ.

The following relationship can also be derived from Figure A 5:

$$\frac{dx}{\Delta\rho} = \sin d\phi \approx d\phi$$

$$\Rightarrow \Delta\left(\frac{1}{\rho}\right) = \frac{d\phi}{dx} \tag{A.35}$$

Combining these two results, the following can be obtained:

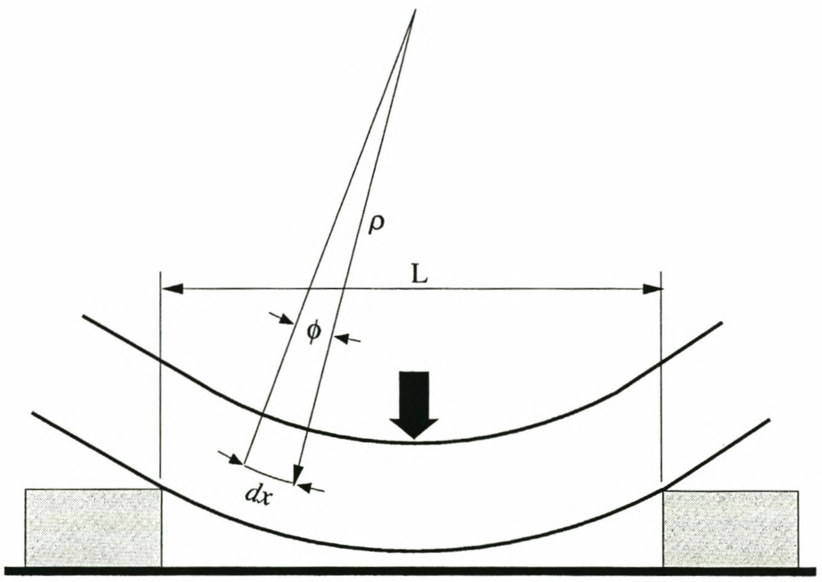


Figure A 5 Diagrammatic representation of bending process

$$\phi = \Delta\left(\frac{1}{\rho}\right)L \tag{A.36}$$

Substitute A.32 in A.36:

$$\begin{aligned} \Rightarrow \phi &= L \cdot \frac{3S}{tE'} \\ &= \frac{3L\left(1-\mu^2\right)S}{tE} \end{aligned} \tag{A.37}$$

APPENDIX B

DESIGN PARAMETER QUANTIFICATION

B.1 COMPUTER

B.1.1 Speed

The speed requirements of the computer were determined by the following:

Signal frequency of external hardware

The maximum square wave frequency required by the hydraulic system and solenoids on the bending machine should not be faster than 10kHz. This is within reach of any 8086 microprocessor-driven or faster computer.

Program execution speed

It is primarily the type of CPU, motherboard and memory available that determines the speed at which computers execute programs. Operating systems slow a computer down if inadequate hardware is used, and they therefore play a major role in the choice of computer. According to a comparison done by the *Linux Resource Exchange*, the different operating systems have the following minimum CPU requirements (Linux Resource Exchange:1999):

DOS does not require very fast computers. Any 386SX or more powerful CPU would suffice if the control program runs under DOS.

Windows 95 and NT put a much greater burden on the CPU. At least a Pentium or more powerful computer is required if Windows is used. The manufacturers claim that a computer with an 80486 or compatible CPU should be fast enough.

Unix systems usually require large Unix servers. Linux has to run on at least an 80386 or faster CPU. X-Windows in Linux requires a computer similar to a Windows NT system.

OS/2 Warp has similar requirements to Windows although the manufacturers claim that it can run on a computer with an 80386 CPU.

Mac OS requires an Apple iMac or Power Mac.

B.1.2 Memory

The memory available on a computer plays a very important role in the performance of an operating system. Large operating systems are not only slow in memory-scarce environments, but are also unstable.

DOS programs usually do not require computers with more than 4MByte memory.

Windows 95 and NT programs require at least 16 to 24MByte of memory regardless of the programming environment (32MByte is preferable).

Unix systems usually require more memory than Windows. Linux can run with 8MByte RAM and X-Windows for Linux 32MByte.

OS/2 requires 32MByte memory.

Max OS machines use standard hardware.

B.1.3 Robustness to EMC problems

The machine is used in an environment with very high EMC emissions due to the use of electric equipment such as welding machines, etc. The bending machine could be dangerous should the control mechanisms fail. The control system should thus be immune to electric noise to prevent machine failure.

B.1.4 Strength of terminals

The computer is placed in a labour-intensive environment where terminals are easily damaged. The computer terminal, keyboard and other peripheral terminal devices are protected from rough handling.

B.1.5 Future need of computer

It is currently a trend in technologically advanced companies to integrate all manufacturing control functions into one central control system. These programs usually require powerful computers.

B.1.6 Conclusion

Windows NT was used in the project. A Pentium computer with 24MByte of RAM installed proved to be powerful enough to control the bending machine. It was found during the development phase that the computer slowed down when many virtual instruments were opened for editing. This should not cause a problem when the machine is operational. The operator will only have access to a compiled version of the program.

B.2 OPERATING SYSTEM

B.2.1 Predictability of real-time tasks in different operating systems

DOS does not allow multi-tasking. The CPU is dedicated to a process until the process is terminated. Control programs can accurately determine execution time of commands. Real-time control is therefore more accurate and easy to predict in DOS (Prosise:1995)

Multi-tasking operating systems such as Unix, Windows 95, Windows NT, Mac OS and OS/2, can execute a number of processes simultaneously. A scheduler is used to give each process a share of the CPU time according to a predefined scheduling algorithm. Processes are usually classified according to importance. A more important process has more CPU time than a less important process. It is therefore possible that the operating system can give CPU time to a new process during millisecond critical control. This could cause malfunction when peripheral devices are signalled too slowly (Dekey:1998).

B.2.2 Complexity of programming techniques required for the operating system

It is much more difficult to do real-time control programming in a multi-tasking environment than in DOS, but manufacturers avoid this problem by supplying device drivers and objects for Windows 95 and NT for their hardware. Windows have a number of RAD (Rapid Application Development) tools that simplify Windows programming in general.

Most hardware companies do not support device drivers to operating systems such as Linux and OS/2 Warp as well as for Windows.

B.2.3 Future information technology needs

DOS does not have the network capability of Windows. Most systems will have to have networked database capabilities in the future to allow sharing of information with other applications. An example of this is the scheduler of a manufacturing plant.

Linux and Mac OS also have very good networking capabilities.

B.2.4 Future of the operating system

Most programming houses have ceased supporting DOS. Windows 95 and NT are currently the most popular operating systems in South Africa. OS/2 Warp and Mac OS are not commonly used in South Africa. Linux is still immature technology that is rapidly gaining ground, especially in the academic community.

B.2.5 Conclusion

It is shortsighted to ignore the fact that DOS is currently seldom used. Windows 95 or NT is running on most desktop computers in South Africa. National Instruments does not guarantee the correct operation of LabVIEW programs in Windows 95, but does give a guarantee for Windows NT. Windows NT was therefore the preferred operating system.

B.3 PROGRAMMING LANGUAGE

Four factors determined the choice of programming languages for the project:

1) The availability of the programming language and the availability of reference material for the language.

2) The complexity of the programming language and the time required to learn the language.

3) The ability of the programming language to control external applications.

4) The ease with which complex control algorithms can be implemented.

Four programming environments were considered:

1) Delphi v1, a Pascal programming tool for Windows developed by Borland, now called Inprise.

2) Turbo Pascal v7.0, also a Pascal programming tool developed by Borland, but operating under DOS.

3) LabVIEW v4.1, a graphical programming tool developed by National Instruments.

4) Borland C++, a C++ programming tool for Windows developed by Borland.

B.3.1 Properties of the different programming languages

The properties of the different languages considered are listed in Table B 1.

B.3.2 Operating system used

The version of Turbo Pascal available allows only programming for DOS. Borland C++, where as LabVIEW and Delphi allow for programming for Windows 3.x, Windows 95 and Windows NT.

B.3.3 Database requirements

Bend sequences must be stored in a text file from where they can be obtained by the application to perform the selected sequence of movements. This text file can be converted into a format understood by other applications by using macros written in a macro language such as Visual Basic.

LabVIEW 4.1 has limited database functionality, but it is able to perform simple file-handling tasks.

Delphi allows the programmer to store data in nearly any format. It also capable of accessing the Microsoft Jet Database Engine and other more powerful database engines such as Oracle.

Turbo Pascal has better database functionality than LabVIEW, but it is difficult to use for control applications.

Borland C++ has the same capabilities as Turbo Pascal.

B.3.4 SIMPLICITY OF PROGRAMMING LANGUAGE

Turbo Pascal requires a very good knowledge of object-orientated programming (OOP). The programmer has to have a good knowledge of all the procedures and functions available in Pascal.

Delphi puts fewer burdens on a programmer's knowledge of OOP, but it is more complicated if special functions are required.

LabVIEW requires a whole new mindset in programming. A programmer has to make a paradigm shift from program execution determined by code flow to program execution by data. It is a graphical environment. Once the programmer is used to LabVIEW, it is much easier to program in LabVIEW provided all required functionalities are available in LabVIEW.

Borland C++ is very difficult to use for an inexperienced programmer. The programmer has to have a thorough knowledge of OOP and all objects available in Windows.

B.3.5 Availability of people with knowledge of programming language

The availability of people with knowledge of the programming language used, can play an important role in the choice of a programming language:

1) Much programming time is saved if the programmer has access to program examples or help from other programmers who have encountered problems.

2) After the completion of the project, it is not possible to give any maintenance support. ROVIC will have to have access to programmers with knowledge of the chosen programming language.

Table B 1 Design parameters for programming languages (Bell, 1987:7)

	Delphi	**Turbo Pascal**	**LabVIEW**	**Borland C++**
Programming environment	Windows	DOS	Windows	Windows
Programming techniques	Graphics and code	Text	Graphical Data flow	Text
Database capabilities	Strong	Moderate	Poor	Moderate
Complexity	Understand OOP[1] and Windows programming	Understand OOP	Data flow programming	Understand OOP and Windows programming
Support	Common programming language	Archaic	Became popular only recently	Common programming language
Maintenance	Easy if correct programming techniques are used	Difficult	Easy if correct programming procedures are used	Moderate even with correct programming procedures
Time required for development of a control system application	Moderate	Long	Fast	Long

[1] OOP – Object-orientated programming

All languages considered can cause maintenance problems if programming is not done structurally. Structured programming simplifies the understanding of the program afterwards. LabVIEW is particularly susceptible to poor structural programming.

B.3.6 Conclusion

LabVIEW was chosen as the preferred programming environment. National Instruments developed LabVIEW as a program development tool for control applications. It has many control functionalities embedded that other development tools lack. The fact that this is the first time that LabVIEW has been used at the Industrial Engineering department of the University of Stellenbosch also played a mayor role in the choice of operating systems, as it enabled the establishment of new technology at the department.

B.4 POSITION SENSOR

B.4.1 Properties of different position sensors required

Position sensors can be divided into two main types (Ruocco, 1987:14):

Incremental position sensors: This type of sensor always gives new positions relative to previous positions by counting the number of increments that the object has moved. The size of the increments is determined by the physical properties of the sensor.

Absolute position sensors: These types of sensors always give the absolute position relative to a specific reference point. Position readings are linear (e.g. potentiometers) or small increments (e.g. absolute shaft encoders).

Position sensors can also be categorised according to the mechanical operation of the sensor. The following four types of sensors are most commonly used:

- Potentiometers.
- Position encoders.
- Capacitive displacement transducers.
- Inductive displacement transducers.

Capacitive and inductive displacement transducers are more expensive than potentiometers and position encoders. They were therefore not considered in this project. Potentiometers and position sensors are also more often used in automation projects than other types of position sensors.

The most important parameter of potentiometers and encoders is linearity over a specified distance. Typical values for both encoders and potentiometers are shown in Table B 1. The calculation of the different parameters is discussed in Appendix A.1 .

B.4.2 Conclusion

The incremental shaft encoder was the most suitable position sensor for this application. The technology required was well known to the design team. An absolute encoder would probably be a better choice because simpler communication electronics would have been involved in the design. The incremental encoder requires extra counter circuitry to count the number of pulses for position acquisition.

Potentiometers were decided against because of electric noise in the factory. Wear and tear may alter the characteristics of this device after prolonged used.

B.5 IO CARDS USED

B.5.1 Ports needed on the IO card

The ports given below were defined at the start of the project. The actual usage of lines is

Table B 2 Typical specification for position sensors

Type	Distance	Resolution	Linearity	Robustness
Linear potentiometers	300mm	0.225mm	0.075	Wear & tear
Shaft potentiometer with cable	1000mm	1mm	0.1%	Wear & tear
Linear encoder	480mm	0.003mm	<0.001%	
Incremental shaft encoder with cable	1000mm	1mm	0.1%	
Incremental shaft encoder with cable &gear	1500mm	0.015mm	0.01%	
Absolute shaft encoder with cable	1000mm	1mm	0.1%	

discussed in table17.

1) Two sets of four control lines to control two counter ICs of the incremental encoders.
2) Four control lines for the control of the bending press.
3) Input buffer for data from each counter IC.
4) Input buffer for information from external buttons and switches pressed.

B.5.2 Driving capabilities of output buffers

Each solenoid of the hydraulic control valve uses 30Watt electrical energy when switching. Special IO driver lines are required on IO cards to drive loads of such high energy requirements.

B.5.3 Availability of device drivers for different programming packages

It is very difficult to do programming for Windows NT if IO cards are addressed directly. The programmer will have to write his own device drivers. Consequently it is important that the manufacturer of the IO card supplies device drivers (or virtual instruments in the case of LabVIEW) to make programming easier.

B.5.4 Conclusions

A PC36C IO card, supplied by a local manufacturer, was chosen. The card is much cheaper and maintenance is easier than in the case of foreign cards. The card has sufficient output lines to comply with the project's requirements. The card cannot drive loads like solenoids, but driver circuitry was developed to rectify the problem. LabVIEW virtual instruments were also available for this card.

B.6 HYDRAULIC VALVE AND SOLENOID

The hydraulic valve formed the link between mechanical and electrical engineering. Properties for the valve are only specified for the electrical design.

B.6.1 ELECTRICAL SPECIFICATIONS

Power available for driving the solenoids is a very important factor in choosing a solenoid. The method used for switching is also important.

Two types of solenoids are available:

- Direct current with operating voltages of 12V, 24V and 48V.
- Alternating current with operating voltages of 24V, 48V, 110V, 220V at 50Hz and 115V and 230V at 60Hz.

The second important factor is the switch-on and switch-off time of the solenoid. Typical values are listed in Table B 3.

B.6.2 CONCLUSIONS

An NG6 valve operating from 12V was chosen.

Power supplies of only 12V DC or 220V @ 50Hz AC were available. A 12V power supply was readily available for both the control valve and driver circuitry. The choice of a direct current supply had very little influence on the complexity of the driver circuitry.

Table B 3 Typical response times for the solenoids.

	D.C.	A.C.
Switch-on	20 - 60 ms	10 - 25 ms
Switch-off	50 - 70 ms	25 - 50 ms

Technology for the switching circuit of the 12V power supply was also available.

It was not possible to determine the influence of response time before testing the response of the entire system, but it proved to be adequate for this specific application.

B.7 ELECTRONIC CONTROLLER OF THE HYDRAULIC VALVE

B.7.1 CONTROL LINES OF DRIVER CIRCUITRY

The driver needs two directional lines to control direction of movement. Circuitry is added to give upward movement precedence over downward movement, ensuring that the valve does not try to send the bending machine in both directions, which will cause unpredictable behaviour.

B.8 COMMUNICATION

B.8.1 CONTROL LINES

Eight control lines were defined for the teach pendant:

1) Continuous upward movement of the bending press.
2) Continuous downward movement of the bending press.
3) Pulsed upward movement of the bending press.
4) Pulsed downward movement of the bending press.
5) Execute one bending sequence line.
6) Halt the machine's movement.
7) Capture current position into the computer's memory.
8) Turn the teach pendant on.

Three control lines from the computer were defined:

1) Move the bending press upwards.
2) Move the bending press downwards.
3) Turn the teach pendant on and off.

Six lines provide feedback from the hardware to the computer:

1) Execute one bending sequence line.

2) Halt the machine's movement.

3) Capture current position into the computer's memory.

4) Turn the teach pendant on.

5) Micro-switch pressed.

6) Emergency switch pressed.

B.8.2 SUPPORT CIRCUITRY

Support circuitry must be added for the following functions:

1) Pulse generator for pulsing the bending blade in both directions.

2) Correct retraction of the bending blade when the emergency stop is pressed.

APPENDIX C

CALIBRATION OF THE MACHINE AND DERIVATION OF THE BENDING EQUATION

C.1 CALIBRATION

It is very important that the calibration should be as accurate as possible. The parameters of the bending equation are determined by the calibration constant. When the calibration constant changes and new position output values differ from corresponding previous outputs, results will be incorrect.

The calibration is done as follows:

1) The calibration constant is initially set to a value of 0.001.

2) The bending blade is moved to the top where the machine is zeroed.

3) The position of the blade is measured with a calliper.

4) The blade is moved downwards as far as possible before reaching the end of its stroke.

5) The position is measured again and the distance is calculated.

6) The new calibration constant is calculated with equation (C.1):

$$constant_{new} = \frac{distance\ measured\ by\ computer}{distance\ measured\ with\ calliper} \cdot constant_{old} \tag{C.1}$$

7) This process is repeated until the measured error is less than 0.1 mm over the total stroke distance of the bending machine.

8) The same procedure is used for the position stop.

The following are typical values used for the calibration constants:

Position stop measurement	0.0096563088
Bending blade movement	0.01034

C.2 DERIVATION OF THE PARAMETERS FOR THE BENDING EQUATIONS

The theory describing the statistical mechanics of the bending equation is discussed briefly in Appendix H.

The following procedure is used to calculate the parameters of the bending equation:

1) A number of target positions are identified beforehand. More measurements have to be done in the elastic area than in the plastic region (see section 7.3).

   Ten values are typically sufficient to deduce the parameters. Typical target values are shown in Table C 1:

The blade is moved down manually by means of either the hand controller or the computer to the desired position. Position is read in millimetres.

2) Release the bending blade.

3) Measure the resulting angle on the bent object. The angle to be measured is

Table C 1 Typical target values used in the derivation of parameters

0mm
1mm
2mm
4mm
7mm
10mm
15mm
20mm
25mm
30mm
35mm

shown in Figure 29.

4) Repeat the process with a new piece of metal for the next measurement.

5) After the measurements have been done, enter the values into the regression analyser as discussed in section 5.4.

6) After analysing the data, the values are added to the list of metals by means of the metal editor or are added manually to the appropriate programs.

APPENDIX D

SAFETY PROCEDURES

Correct safety procedures are important to prevent accidents while operating the machine. Procedure is divided into preventive or maintenance actions and emergency actions.

Maintenance

A number of tasks must be performed before start-up:

1) Check that all hydraulic valves are in a good condition.

2) Check that all hydraulic connections are well connected and firmly attached.

3) Check the hydraulic fluid level in the hydraulic pump tank.

After these checkpoints have been approved, the following actions must be performed:

1) Start the hydraulic pump and check the pressure gauge.

2) Turn on the electronics on the machine. The bending blade will retract to the top of its stroke. If it does not retract, press the emergency switch to enforce retraction. Check the pressure gauge during this operation.

3) Turn on the computer if it is not on.

4) Load the control program and test the machine by moving it up and down continuously. Remember to check the pressure gauge while performing this operation.

If no errors are found, the machine is ready for operation.

Emergency errors

Depending on the circumstances, two procedures are possible, namely retracting of the bending blade or switching off the power supply.

If a body part is caught in the machine, press the emergency button. The machine will retract to the top of its stroke. The flat bar being bent will tend to fall to the ground. The operator or helper will have to keep the metal from falling on human beings or machinery.

If the machine behaves incorrectly, press the emergency button to enforce retraction of the blade. After retraction, turn off the power supply and restart the machine.

APPENDIX E

DETAILED DESCRIPTION OF THE CONTROL PROGRAM

The program discussion in this section is aimed at program maintenance and not to enhance functional understanding.

E.1 OVERVIEW OVER THE PROGRAM

The program is divided into three parts as shown in Figure E 1. Each part is discussed separately.

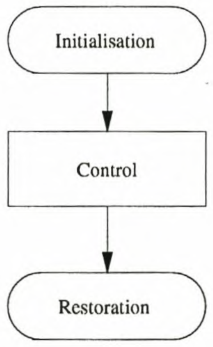


Figure E 1 Flow diagram of the program overview

E.2 INITIALISATION STAGE

The program is initialised stepwise by means of a sequence structure. Each option line stored on file is thus individually loaded into memory. Typical code flow for loading a line from file is shown in Figure E 2.

The *Load Line* block reads the data. This block loads an entire data line from file on each reading. A Carriage Return or EOL character denotes the end of a line. The *Load Line* block requires four data inputs:

1) File name – set to c\buig\opsies.txt.

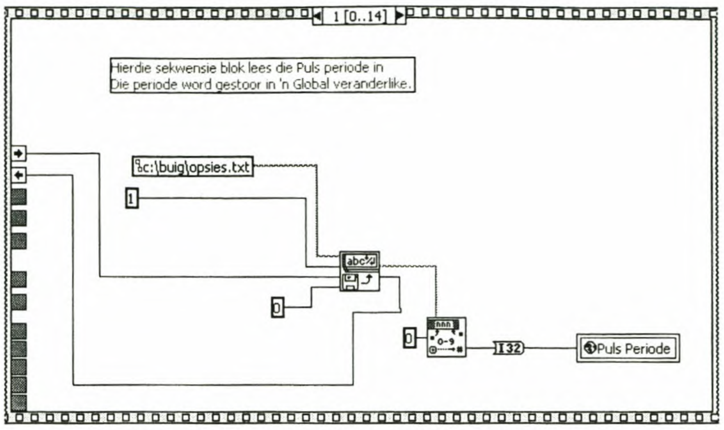

1 [0..14]
Hierdie sekwensie blok lees die Puls periode in
Die periode word gestoor in 'n Global veranderlike.
c:\buig\opsies.txt
1
0
0
I32
Puls Periode

Figure E 2 Code diagram for loading a line from file.

2) Number of lines to be read – set to one line.

3) Current file position to start reading of line.

4) Error in.

The output value is a string value that is immediately converted to an integer or float value.

A list of the initialisation values loaded appears in section 5.1.3. After all the different

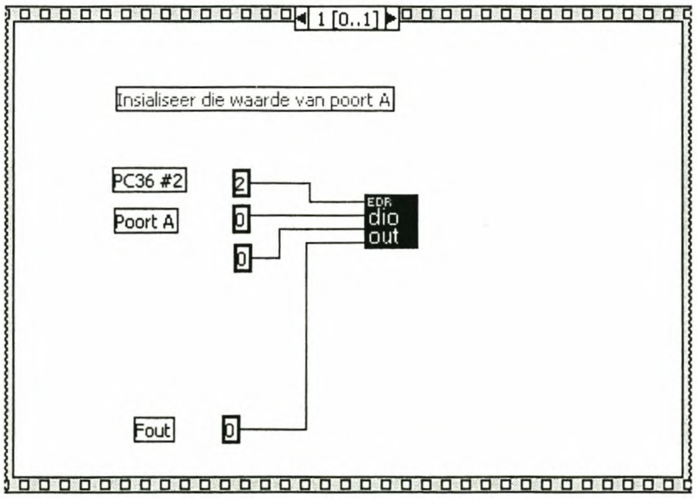


Figure E 3 Initialisation of the program configuration option values have been loaded, the hardware is initialised:

1) The ports of PC36C boards are configured for output and input as shown in Table 17.

2) After the correct set-up of each port, the control boards are initialised as discussed in section 5.1.3.

Figure E 3 shows the initialisation of the ports. The *dioout* block requires the following inputs:

1) The device handle number as specified in the set-up program of the PC36C supplied by Eagle Technology.

2) The port number that is to be used (Port A = 0, Port B = 1, etc.).

3) Data sent to the specific board. A value of 7 is used in this case. It turns all machinery off and sets the counters in the correct state to start position reading.

4) Error signals are ignored in this case by setting them to zero.

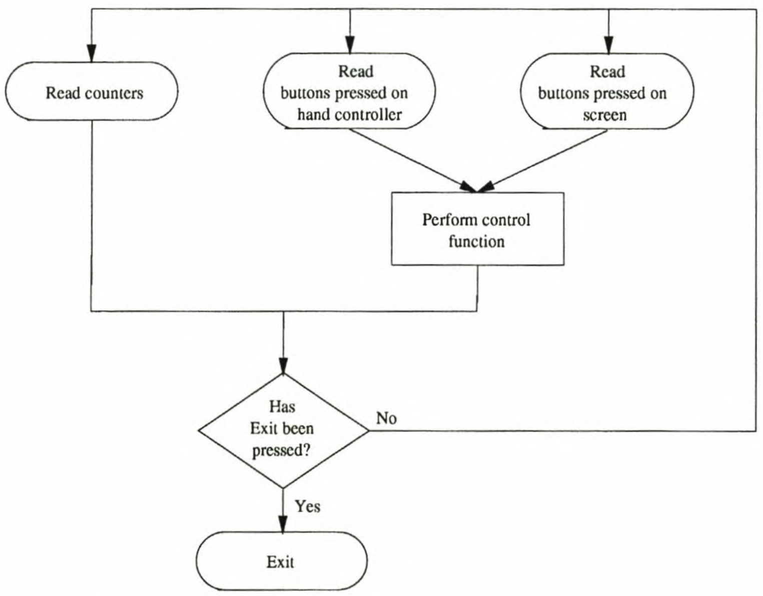

Read counters
Read buttons pressed on hand controller
Read buttons pressed on screen
Perform control function
Has Exit been pressed?
No
Yes
Exit

Figure E 4 Diagrammatic overview of the control program

E.3 CONTROL STAGE

The control stage is remarked for three concurrent processes as shown in Figure E 4.

The order of execution of these commands does not matter. Concurrency is obtained due to the dataflow nature of LabVIEW. It is repeated until the encompassing *while*-condition is satisfied by pressing the *Exit*-button on the computer screen.

The different parts of the control stage will now be discussed.

E.3.1 Reading of counters

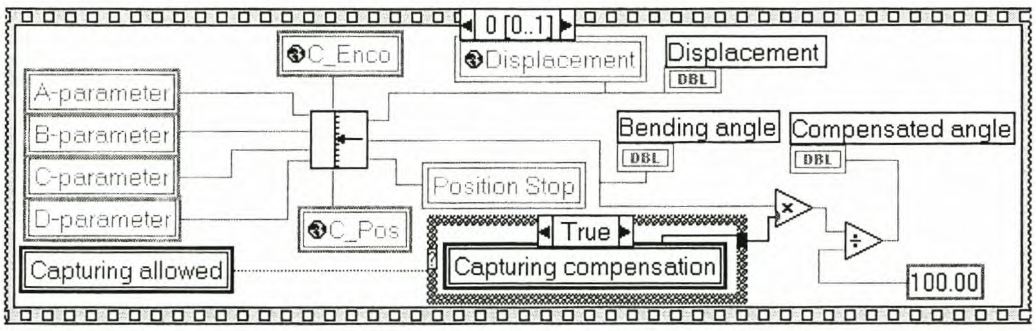


Figure E 5 Reading of the counter values

A typical example of how the position acquisition is done is shown in Figure E 5. Digital data acquisition is used to read values stored in the counters.

The two counters are read simultaneously by means of a *position acquisition* VI. The VI requires four parameters, namely the four parameters for the bend angle-linear movement relation given by equation 5.1 in section 5.1.5. The values for these parameters are stored in variables by either the bend sequence load function or the metal list. The *position acquisition* VI loads the equation parameters from these variables.

The two PC36C boards were initially probed concurrently by separate VIs, but the readings tended to be faulty. The problem was solved by probing the boards sequentially with a single VI. Reading of the counters is discussed in Appendix 5.1.

E.3.2 Reading of buttons pressed

The program reads two types of buttons:

1) Hand control buttons and the different switches.

2) Control buttons on the computer screen.

These sets of buttons are read concurrently as shown in Figure E 4. Some buttons in the control program have the same purpose as corresponding buttons on the hand controller. Inputs of buttons with similar functions are combined by or-gates as shown in Figure E 6.

Inputs from buttons are grouped in a Boolean array, which is converted to a number. This number determines which control function is executed. Only part of the buttons is shown to illustrate the programming of the button functionality.

The micro-switch status is also read in and saved in a Boolean variable.

E.3.3 Execution of control functions

The bending machine has a number of control functions as discussed in section 5.1. The

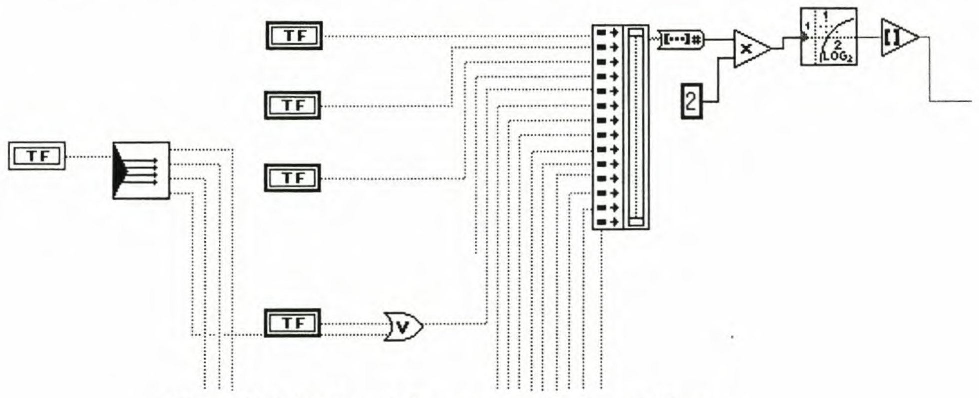


Figure E 6 Reading of the screen and hand control buttons

realisation of each of these functions will be discussed separately.

Continuous upward movement of the bending machine blade

Figure E 7 shows the program code for upward movement of the bending blade. The Boolean value for the switch is set as shown in Figure E 6 in Appendix E.3.2. When the micro-switch is pressed, no movement is possible other than downwards. The *movement* sub-VI discussed in Appendix E.5.2 is used to move the blade.

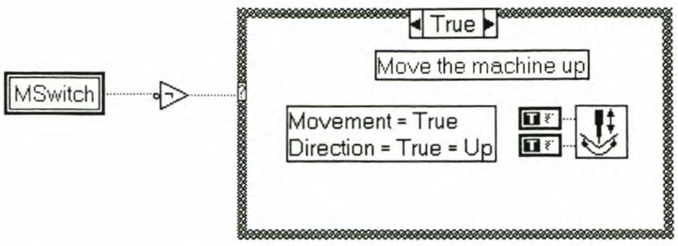


Figure E 7 Continuous upward movement of the blade

Continuous downward movement of the bending machine blade

The program code for continuous downward movement is similar to the code for upward movement except that the bottom limit is software specified. This is done by reading the current position as discussed in Appendix E.3.1 and comparing the value with the bottom limit stored in the global value *Bottom* (See section 5.1.3).

The code is shown in Figure E 8.

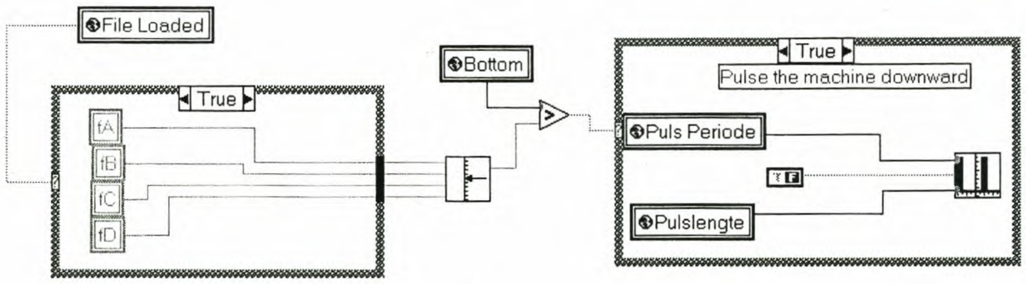


Figure E 8 Continuous downward movement of the blade

Pulsed upward movement of the bending machine blade

This program code is based on the code for continuous upward movement of the bending blade. However, the *Pulse* sub-VI has different parameters. The pulse length and pulse period (see section 5.1.4) have to be specified with the direction of movement as shown in Figure E 9.

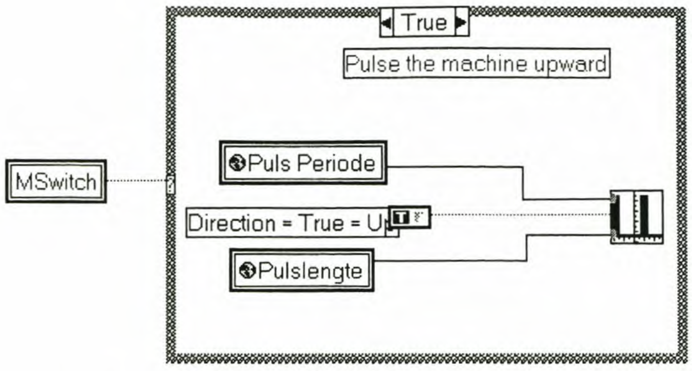


Figure E 9 Pulsed upward movement of the blade

The state of the micro-switch again determines whether the machine blade is allowed to pulse upwards.

Pulsed downward movement of the bending machine blade

Pulsed downward movement operates exactly like pulsed upward movement except for direction and the method for limiting movement. The limit is determined in the same way as the continuous downward movement discussed earlier. The code diagram is shown in Figure E 10.

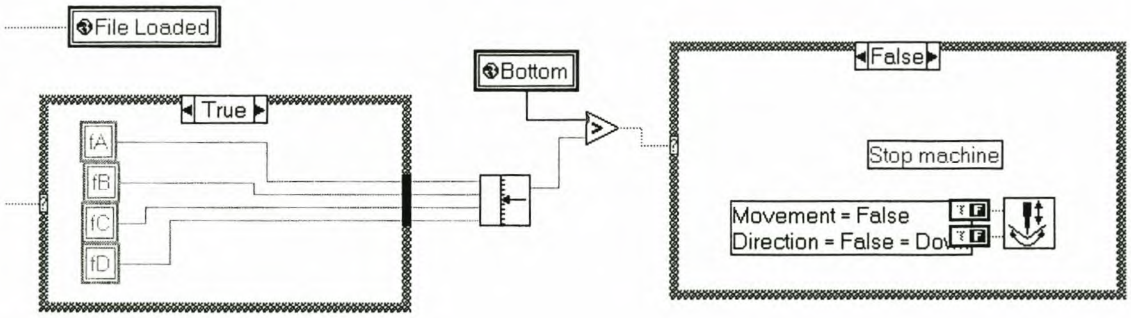


Figure E 10 Pulsed downward movement of the bending machine blade

Release the bending machine blade

Part of the program code is shown in Figure E 11. Movement is stopped when the bending blade reaches its target or the *Halt* button on either the screen or hand controller is pressed. Values for the target and dead band are loaded during the initialisation phase.

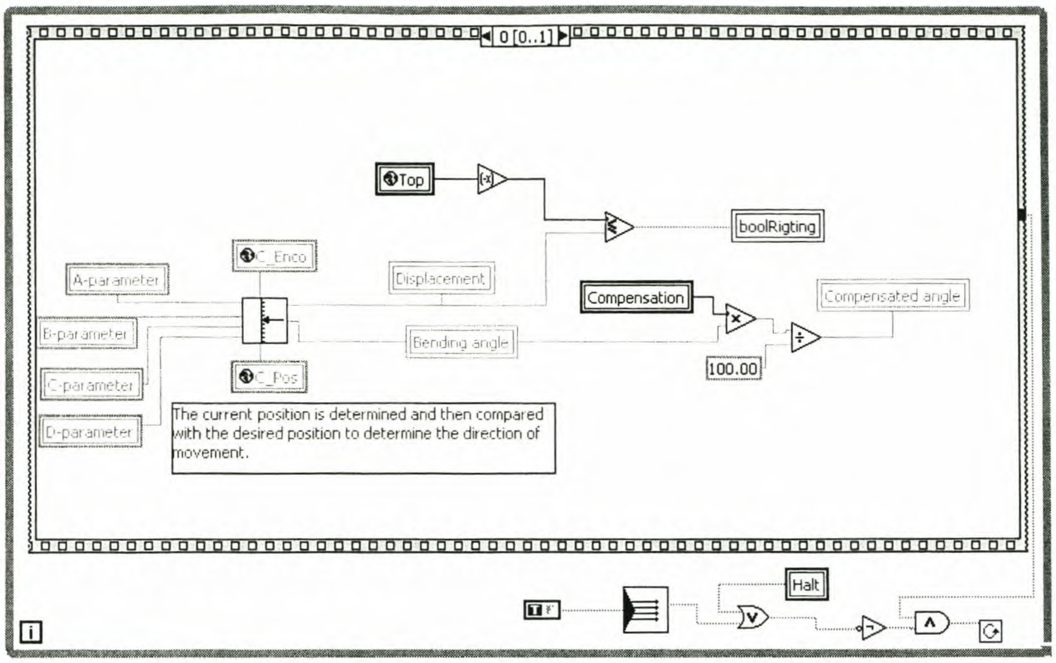


Figure E 11 Program code for the retraction of the bending machine blade to Top

The method used for position acquisition in this function is similar to the method described in Appendix E.3.1.

This function is executed by means of the following steps:

1) Read the current position by means of the *position acquisition* VI as discussed previously. Compare it with the value stored in the variable *Top* to determine the direction of movement.

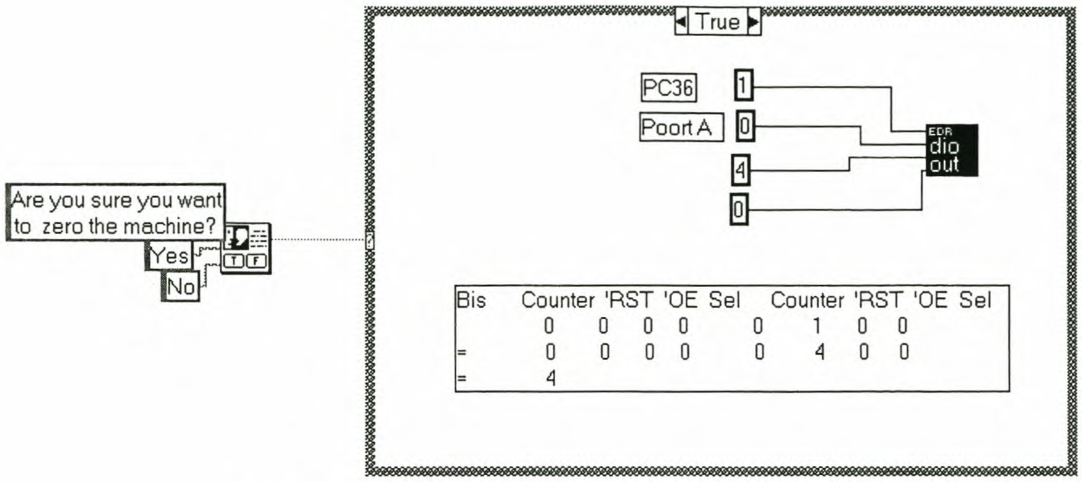
2) A *Case* structure is used to move the bending blade continuously until the pulse band is reached from where the blade is pulsed until the stop band is reached.

The *Halt* button on both the computer screen and the hand controller is probed concurrently to determine whether movement should stop.

Zero the counter for the bending blade

The program code for zeroing the bending blade counter is shown in Figure E 12.

The counter is zeroed by sending a high signal to the *Reset* pin. All other pins are zeroed to ensure that the counter is in the correct state before data acquisition is restarted.



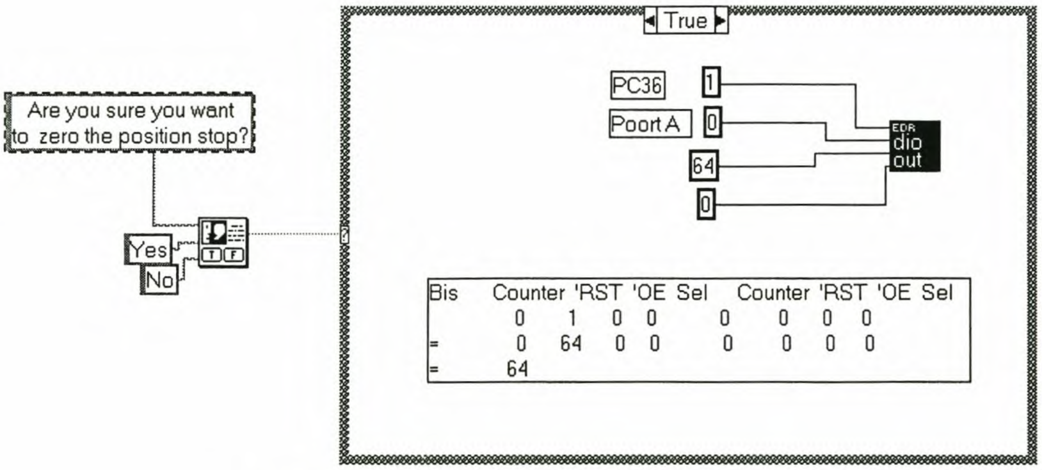
```
Bis   Counter 'RST 'OE Sel   Counter 'RST 'OE Sel
         0     0    0   0       0     1    0   0
=        0     0    0   0       0     4    0   0
=        4
```

Figure E 12 Zeroing the bending blade counter

The user is prompted to confirm the zeroing operation by means of a *Case* structure.

Zero the counter for the position stop

The program code for zeroing the position stop counter is shown in Figure E 13. It is virtually the same as for the bending blade counter.



```
Bis   Counter 'RST 'OE Sel   Counter 'RST 'OE Sel
         0     1    0   0       0     0    0   0
=        0    64    0   0       0     0    0   0
=       64
```

Figure E 13 Zeroing the position stop counter

The counter is zeroed in the same way as the bending blade counter. A *Case* structure is again used to prompt the user to confirm the zeroing of the counter.

Reset the bend sequence program

This function is responsible for resetting the bend sequence program to its first line of execution. It is executed in the following sequence:

1) Check to see whether a file has been loaded into the memory.

2) Confirm with the operator whether the function should be executed.

3) Reset the program to the top row by zeroing the index value of the *Number* list box of the bend sequence program list boxes.

Insert single line

This function inserts a bend sequence command when capturing is enabled. The function

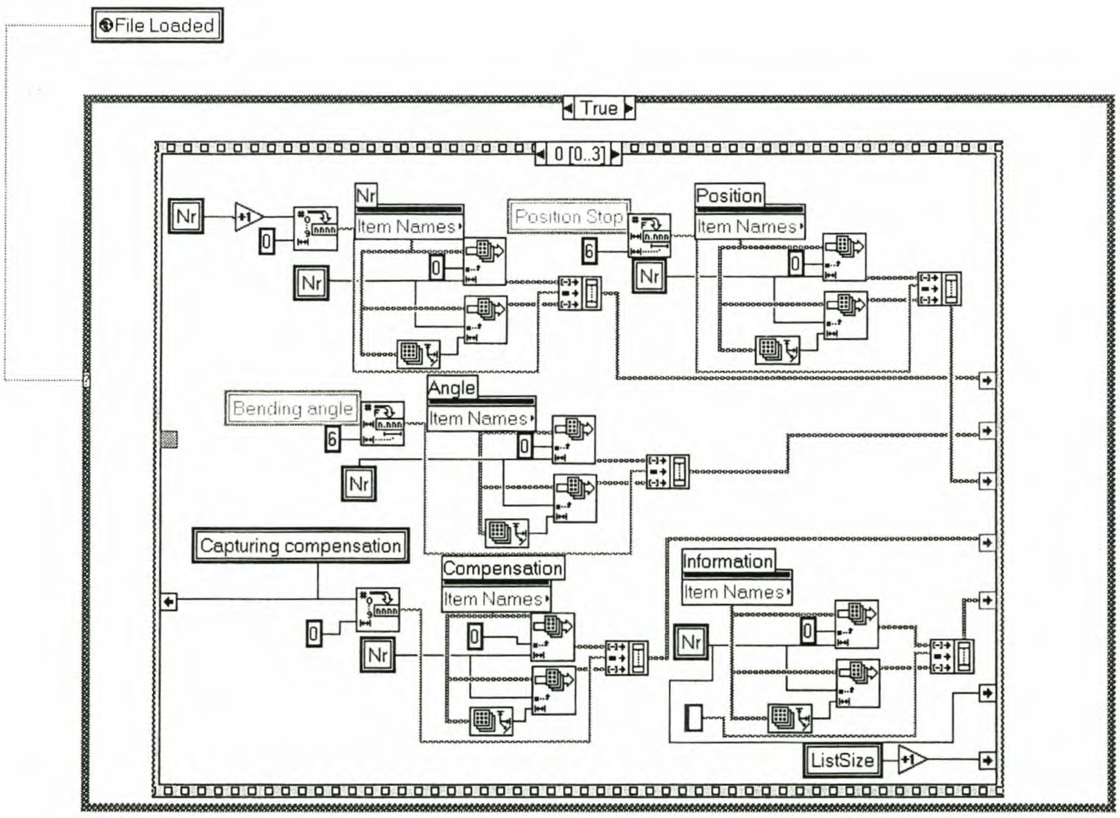


Figure E 14 Part of the code for insert bend sequence lines in the bend sequence program is executed as follows:

1) The current number is used as an indexing value to break the bend sequence list box arrays into two arrays, inserting the new line between them when rejoining them. Values for the new line are obtained from the current encoder reading of the *position acquisition* VI. The code for performing this step is shown in Figure E 14. List sizes are increased simultaneously with line insertion.

2) The new arrays are stored in their appropriate bend sequence list boxes. This step is repeated until the operator releases the *Insert line* button.

3) The list box containing the line numbers is zeroed.

4) The list box is then repopulated with numbers in the correct order.

The execution of a single bend sequence command

This function is responsible for the execution of a single bend sequence command. Each

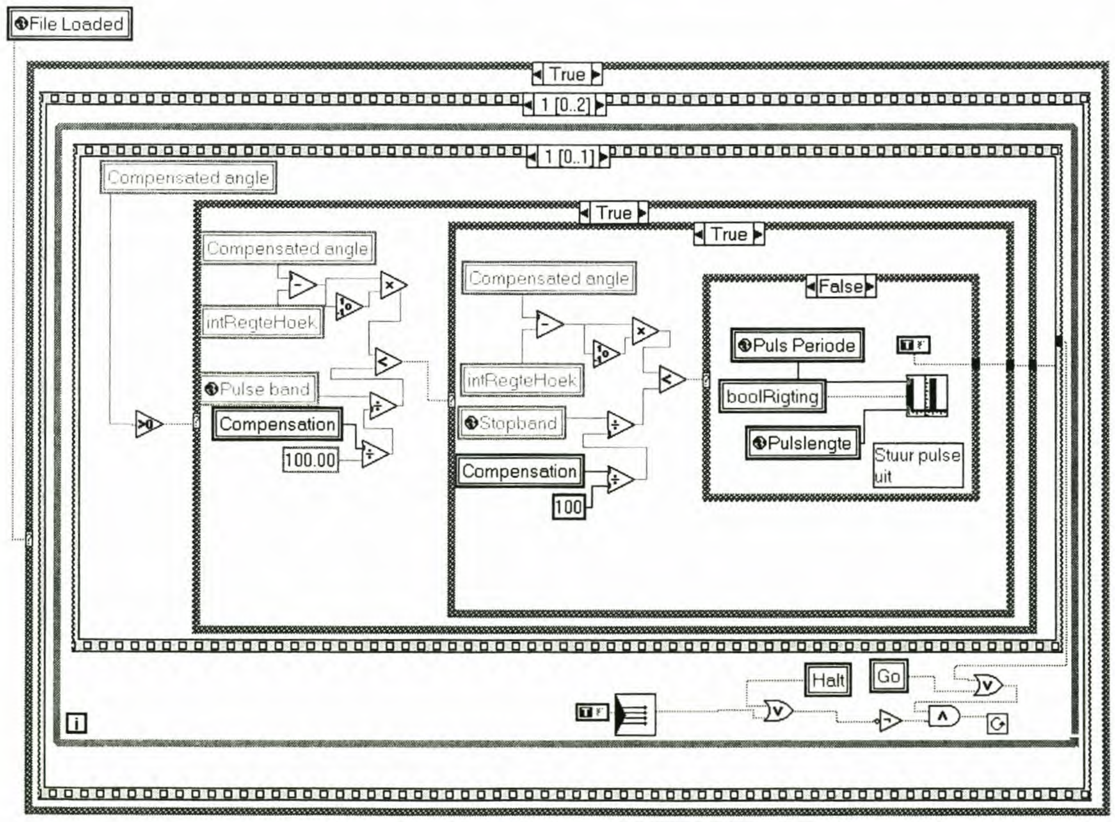


Figure E 15 Program code for movement of the bending blade line in the different bend sequence list boxes represents a bend sequence command. It is divided into the following steps:

1) Verify that a file has been loaded.

2) Obtain the target angle from the bend sequence list boxes.

3) Read the current position from the encoder and compare it with the target angles to determine direction of movement.

4) Determine what type of movement is required as shown in Figure E 15.

5) After reaching the stop band, increment the pointer from the current bend sequence command to the next command.

Halt the machine

This command halts the machine when necessary by means of the following steps:

1) Read the current values of Port A of the control PC36 board (Board 2).

2) Change only bits 2 and 3 to false to halt the machine. An or/and combination is used to retain the values of the other bits.

Load a bend sequence file

This function identifies the file to be loaded from disk for execution. It is divided into the following steps:

1) Verify that a file has been loaded into the memory. If a file has been loaded, confirm with the operator to proceed with loading a new file.

2) Prompt the user to choose a file from disk and set the following parameter attributes to the following values:

   a) *Reset* button – Visible

   b) *Parameter* button – Invisible

   c) *Go*-button – Visible

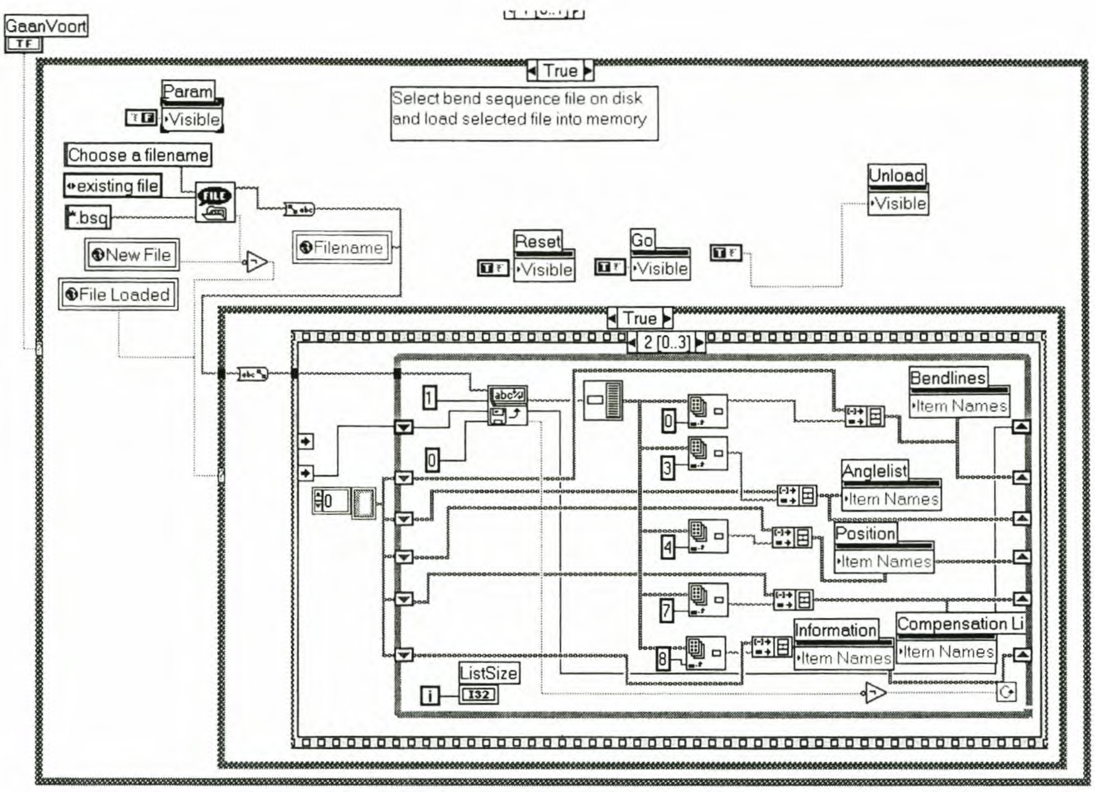


Figure E 16 Part of the code flow to load a bend sequence file

d) *Unload* button – Visible

3) Load the chosen file's information line and store the value into the *File information* indicator.

4) Load the parameter line and store the values in appropriate boxes.

5) Load the different bend sequence commands from file, translate them into correct data types and store them in appropriate list boxes. Repeat this step until the *end of file* mark is reached as shown in Figure E 16.

Turn hand controller on

This function gives control to the hand controller. When this function is active, the bending machine blade can only move by means of the hand controller. It is divided into the following steps:

1) Read the current values of Port A of the control PC36 board (board 2).

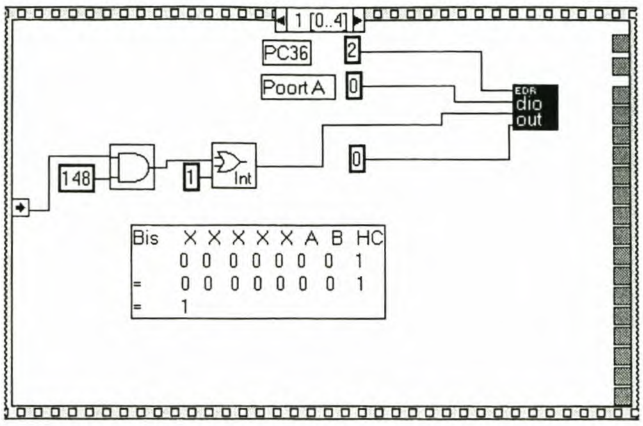


Figure E 17 Part of the program code to turn hand control on

2) Turn on the hand controller without changing the "does-not-matter" bits as shown in Figure E 17.

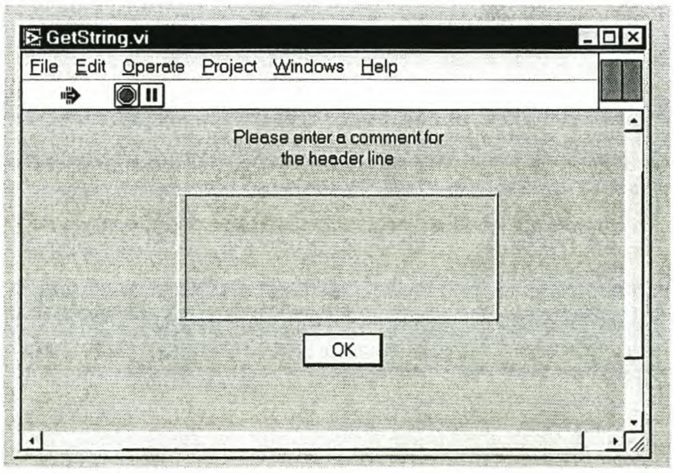
3) Save the current status of the different buttons.

4) Make unnecessary buttons invisible and show the current encoder readings. This step is repeated until the hand control button is depressed.

5) Restore the different buttons to their original status.

Create new file

This function allows the user to create a new file. The following steps are performed



Please enter a comment for the header line

Figure E 18 The dialogue screen for entering the header line of the new bend sequence program during the execution of the function:

1) A file dialogue VI is used to prompt the user to enter the name for a new document. All the bend sequence list boxes are reset during this step.

2) The user is prompted by a custom VI to enter a header line. The dialogue box is shown in Figure E 18. The following parameters are reset to the values shown:

    a) List Size – -1

    b) Ou Nommer – -1

    c) Nr – -1

    d) Unload button – Visible

    e) Param button – Invisible

Choose parameters

Parameters for different types of metal are set in this function.

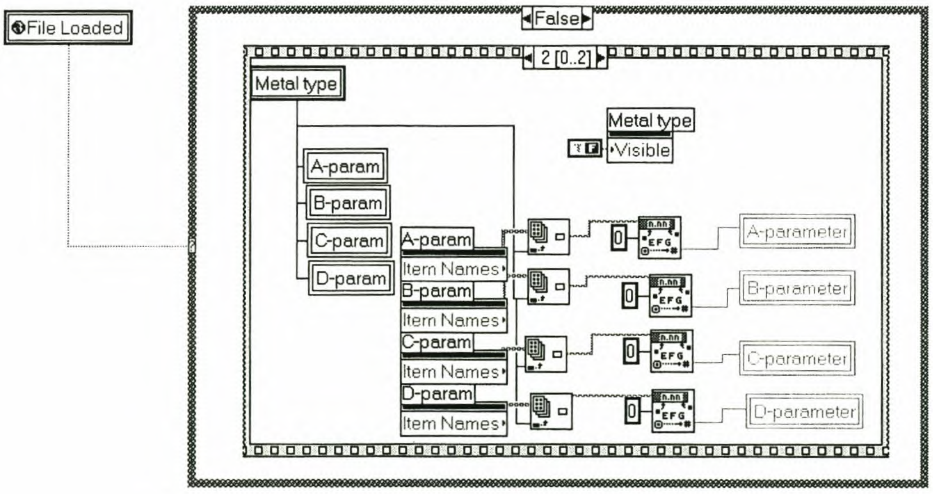


Figure E 19 Part of the program code to set new parameters

The program first verifies that no file has been loaded before the following steps are executed:

1) The metal list box with the different values is shown.

2) A while-loop ensures that the list box is visible until the parameter button is depressed.

3) Set all the parameter list boxes to the same item index as the *Name* box in the group and store the corresponding values in the applicable parameter variables as shown in Figure E 19.

4) Make the metal list box invisible again.

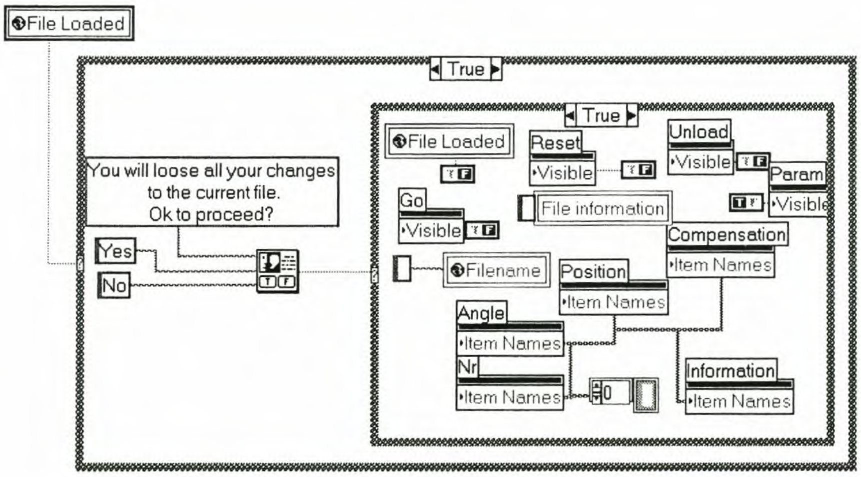


Figure E 20 Part of the program code to unload the current file

Close file

The program removes the current *bsq* file from the memory and indicates to the rest of the program that no file has been loaded.

The program first verifies that a file has been loaded in the memory before the following actions are executed:

1) The program verifies that the operator wants to remove the file from the memory.

2) All file-related variables are reset as shown in Figure E 20.

Add line to bend sequence program

This function enables the operator to add a new line to the end of the loaded bend sequence file.

The program verifies that a file has been loaded in the memory before the following steps are performed (see Figure E 21):

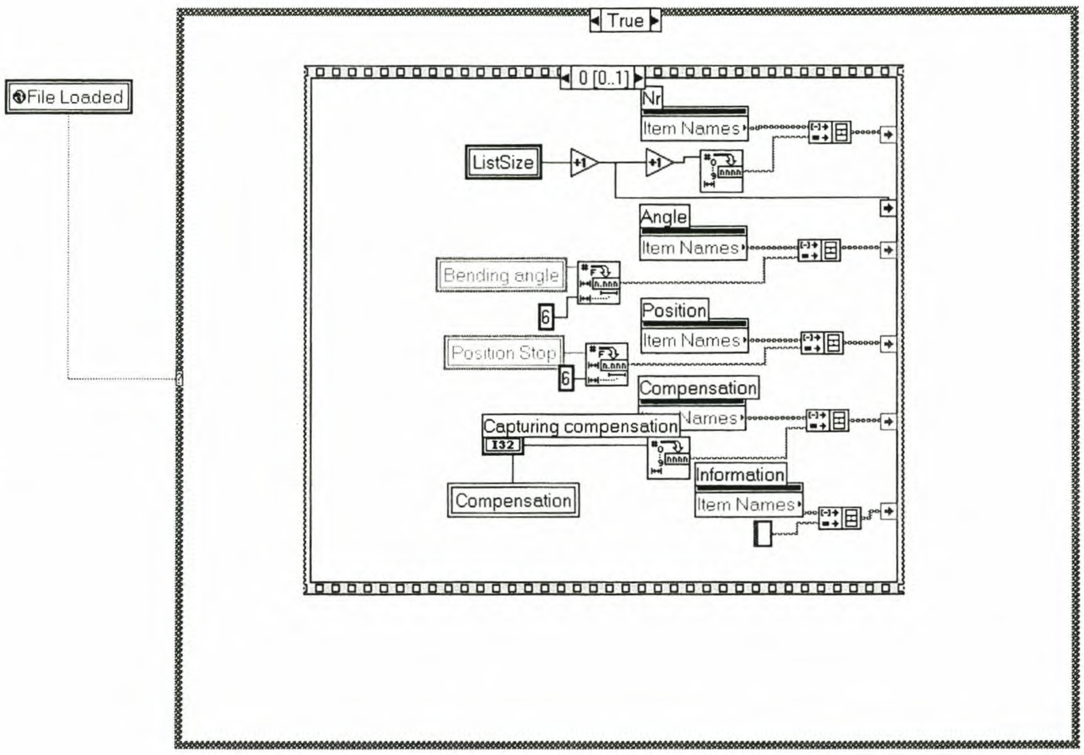


Figure E 21 Part of the program code for inserting a new line to the loaded bend sequence program

1) A new number is added to the last placeholder of the number list box.

2) The current value of the *Capturing Compensation* knob is entered in the last placeholder of the Compensation list box.

3) The values of the bending angle and the position stop indicators are entered in the last placeholder of the bending angle and position stop list boxes.

4) The counter storing the number of items in list boxes is incremented.

5) A while-structure prevents further execution of the program until the *Add line* button is released.

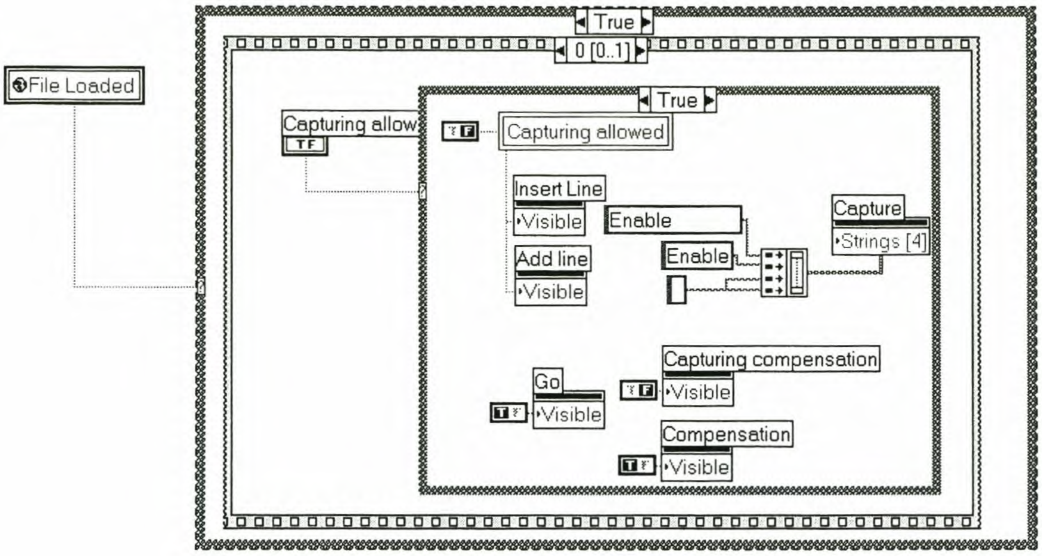


Figure E 22 Part of the program code for adding a new line to the loaded bend sequence program

Enable capturing of bend sequence program lines

This function allows the operator to capture program lines into the current bend sequence program. The program first verifies that a file has been loaded in the memory before the following steps are executed:

1) Verify that capturing is allowed. If capturing is allowed, the following tasks are performed (see Figure E 22):

   a) Turn the *Capturing allowed* function off by resetting the *Capturing allowed* Boolean flag.

   b) Make the buttons *Add line* and *Insert line* invisible, as well as the *Capturing compensation* control.

   c) Show the *Go* button and the *Compensation* control.

d) Set the label of the *Capture* button to *Enable capturing*.

If capturing is not allowed, the program does exactly the opposite.

a) Turn the *Capturing allowed* function on by setting the *Capturing allowed* Boolean flag.

b) Show the buttons *Add line* and *Insert line,* as well as the *Capturing compensation* control.

c) Make the *Go* button and the *Compensation* control invisible.

d) Set the label of the *Capture* button to *Disable capturing*.

2) Do a while-loop until the *Capture*-button is released.

Save the bend sequence program to file

This function enables the operator to save the loaded file to disk. The program first verifies that a file has been loaded before executing the following steps:

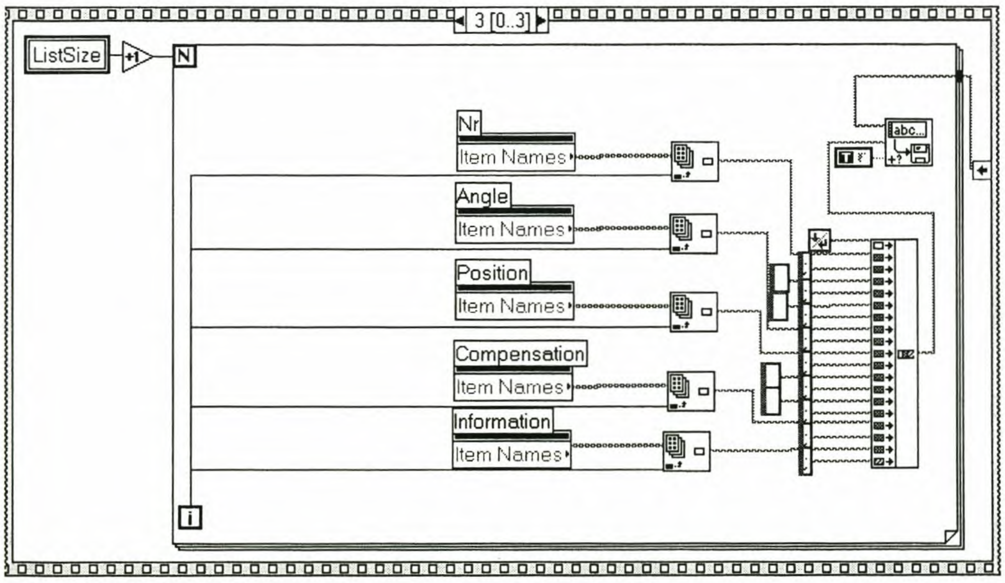


Figure E 23 Part of the program code for saving a file

1) Delete current file from disk.

2) A new file is created when saving the header line to disk.

3) The bend equation parameters stored in memory are saved in the second line of the created file.

4) A for-loop is used (see Figure E 23) to save the program lines one by one in the newly created file. Program lines are compiled from the different bend sequence list boxes.

E.3.4 List of variables used and their functions

The function of the different variables is listed in Table E 2. Items marked with an asterisk are global values that are saved into a file; all the others are local values.

Table E 1 List of variables

Variable		Type	Function
Puls Periode	*	Float	Pulse period
Puls Lengte	*	Float	Pulse length
Top	*	Integer	Top limit for the allowed bending blade movement
Bottom	*	Integer	Bottom limit for the allowed bending blade movement
MSwitch		Boolean	State of micro-switch
Exit		Boolean	State of *Exit* button
Continuous Up		Boolean	State of *Continuous Up* button
Continuous Down		Boolean	State of *Continuous Down* button
Pulse Up		Boolean	State of *Pulse Up* button
Pulse Down		Boolean	State of *Pulse Down* button
Release		Boolean	State of *Release to zero* button
Zero		Boolean	State of the *Zero* button
Capture		Boolean	State of the *Capture* button
Load		Boolean	State of the *Load* button
Reset		Boolean	State of the *Reset* button
Go		Boolean	State of the *Go* button
Halt		Boolean	State of the *Halt* button
strLine		String	Total string loaded from file or to be saved to file
intNommer		Integer	Line number of file loaded from disk or to be saved to disk
intDikte		Integer	Thickness of the metal to be bent
intWydte		Integer	Width of the metal to be bent
intHoek		Float	Target angle for the bend
intLengte		Integer	Position of the bend
intKomp		Integer	Compensation for the bend
strDie		String	Name of the type of blade to be used
intPunch		Integer	Width of the V-block
strInfo		String	Extra information of a bend command
intRegteHoek		Float	Angle measured by the encoder
Dummy #1		Integer	Dummy variable
btnHandControl		Boolean	State of the *Hand control* button
A-parameter		Float	Value of *Parameter A* knob of bend-angle relation
B-parameter		Float	Value of *Parameter B* knob of bend-angle relation
C-parameter		Float	Value of *Parameter C* knob of bend-angle relation
D-parameter		Float	Value of *Parameter D* knob of bend-angle relation
fA		Float	Value of parameter A of bend-angle relation loaded from file
fB		Float	Value of parameter B of bend-angle relation loaded from file

Table E 2List of variables (continued).

Variable	**Type**	**Function**
fC	Float	Value of parameter C of bend-angle relation loaded from file
fD	Float	Value of parameter D of bend-angle relation loaded from file
Compensation	Integer	Compensation value of the angle
Line #1	String	Header line of *bsq* file
Line #2	String	Parameter line of *bsq* file
New	Boolean	State of the *New* button
Param	Boolean	State of the *Change parameters* button
Position Stop	Float	Value read in from the position stop
Repeat	Boolean	State of the *Repeat* button
Unload	Boolean	State of the *Unload* button
Compensation value for previous bend	Integer	Compensation value for the previous bend
MayRepeat	Boolean	Indicator whether a specific bending line may be repeated
GaanVoort	Boolean	
Bending Angle	Float	Current angle of the bend
Displacement	Float	Displacement of bending blade
Information for next bend	String	Information for next bend
Information for previous bend	String	Information for previous bend
File Info	String	
boolRigting	Boolean	Direction of the movement
OEbsqF	Boolean	Indicator if end of the bend sequence file is reached
Compensated Angle	Float	Value of the compensated angle
strLine Old	String	String value of the previous bend
Pasgesave	Boolean	Boolean flag used for the saving of program lines

E.4 RESTORATION PHASE

This step is not needed in the compiled version of the program, but is used in the uncompiled version to ensure that all the controls are shown for editing.

When the program is finished, the controls of the program are reset. The visible attribute of all controls and indicators is set to *true*.

E.5 OTHER IMPORTANT VIs

E.5.1 Reading of the counters

This VI is responsible for reading values from the encoders. It is divided into the following steps:

1) Port A of the PC36 board 1 is initialised for reading.

2) By setting the appropriate pins the counter PC board is instructed to make the middle byte of the counters available.

3) Data from counter one is loaded as shown in Figure E 24.

4) Data from counter two is loaded.

5) The counter PC board is instructed to make the lower byte of the counters available.

6) Data from counter one is loaded.

7) Data from counter two is loaded.

8) The counter PC board is instructed to make the higher byte of the counters

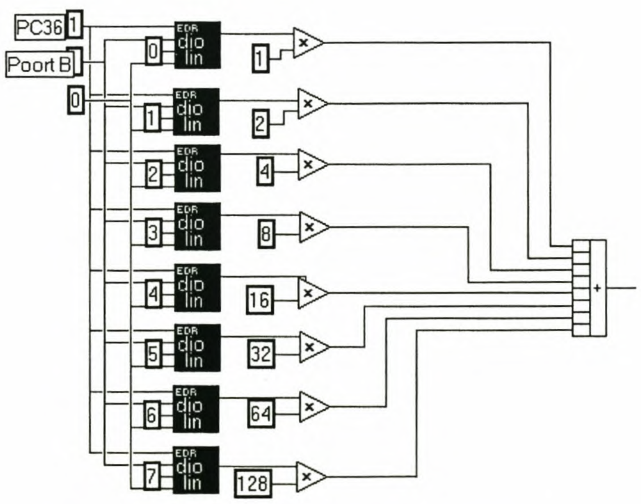


Figure E 24 Program code for generating movement available.

9) Data from counter one is loaded.

10) Data from counter two is loaded.

11) Positions are calculated by means of formula nodes. The parameters for the formula node are supplied by the parent VI.

E.5.2 Continuous movement

This VI is responsible for the up and down movement of the bending blade. It receives two inputs, which have the following effects:

Input	True	False
Up/Down	Up	Down
On/Off	On	Off

The function is divided in the following steps:

1) The current output is read in from the PC36 board 2.

2) The output for the new action is determined by sending the values of the input buttons through a series of case structures. An *and/or* combination is used to change only the relevant bits of Port A on the PC36 board.

E.5.3 Pulsed movement

This function is used to develop a square wave. It uses the *continuous movement* VI to create a signal on the PC36C board.

The inputs to this VI are shown in Table E 2:

Table E 2 Inputs to pulse movement *VI*

Input	Values	Description
Pulse Period	Integer	Total period of the pulse
Pulse length	Integer	The on-time of the pulse
Direction	Boolean – True	Up
	Boolean – False	Down

The following steps execute the function:

1) Input values are read to determine the function behaviour.

2) The pulse period is converted to frequency and the duty cycle is calculated by dividing the pulse length by the pulse period. Both values are sent to the function-generating block, which generates a pulse train stored in an array.

3) The values from the pulse train are read by the combination of an array index and a for-loop once every millisecond to generate the movement signal. The movement signal is sent to the *continuous movement* VI.

The function is shown in Figure E 25.

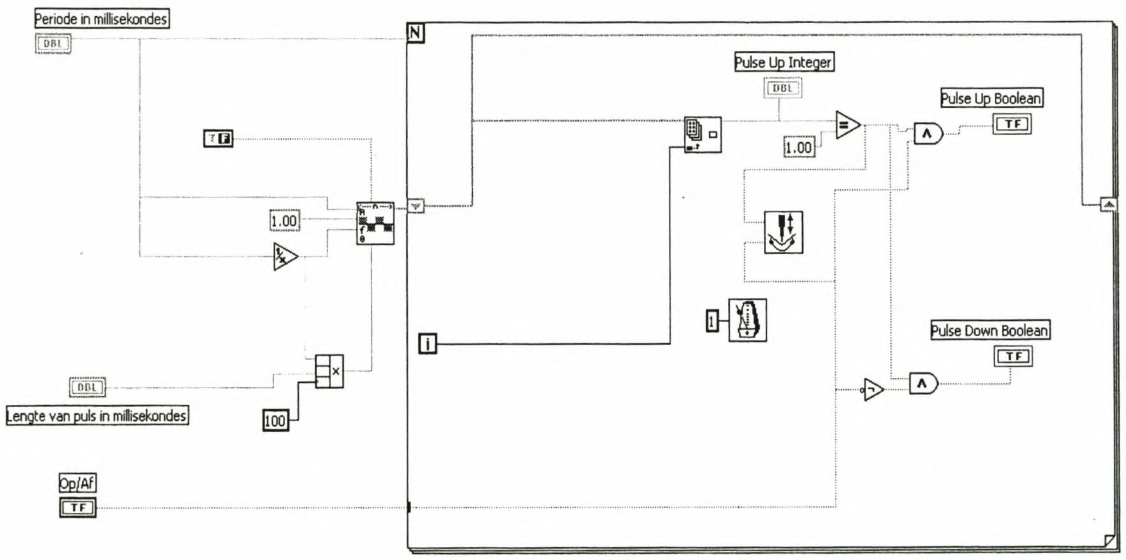


Figure E 25 Program code for generating pulses

APPENDIX F

LAYOUT OF THE DIFFERENT PC BOARDS

Layout of all the different PC boards is discussed in this section. There is more than one version of some of the PC boards, and all versions are shown and discussed.

Four printouts are supplied with each board:

Top layer: This layer can be recognised by the inscription, *component side*. All the components are placed on this side of the board.

Bottom layer: Components are soldered to this layer on the board, which can therefore be recognised by the inscription, *solder side*.

Bottom layer mirrored: The bottom layer is a mirror image that fits under the top layer. The top layer and mirrored bottom layer are therefore used for the manufacturing of the boards.

Silk layer: This layer gives a silk screen of the component layout. The silk screen is printed on the top of the board to indicate the component placement. An inexpensive manufacturing process was used for the PC boards in this project, and printing the silk screen was therefore not possible.

The via's and pads for the components were not coppered through the holes in the manufacturing process. Wire-wrap and pins of components were used to connect the top and bottom layers of the PC board.

All PC board layouts shown in this section have the correct dimensions and can be copied directly for manufacturing.

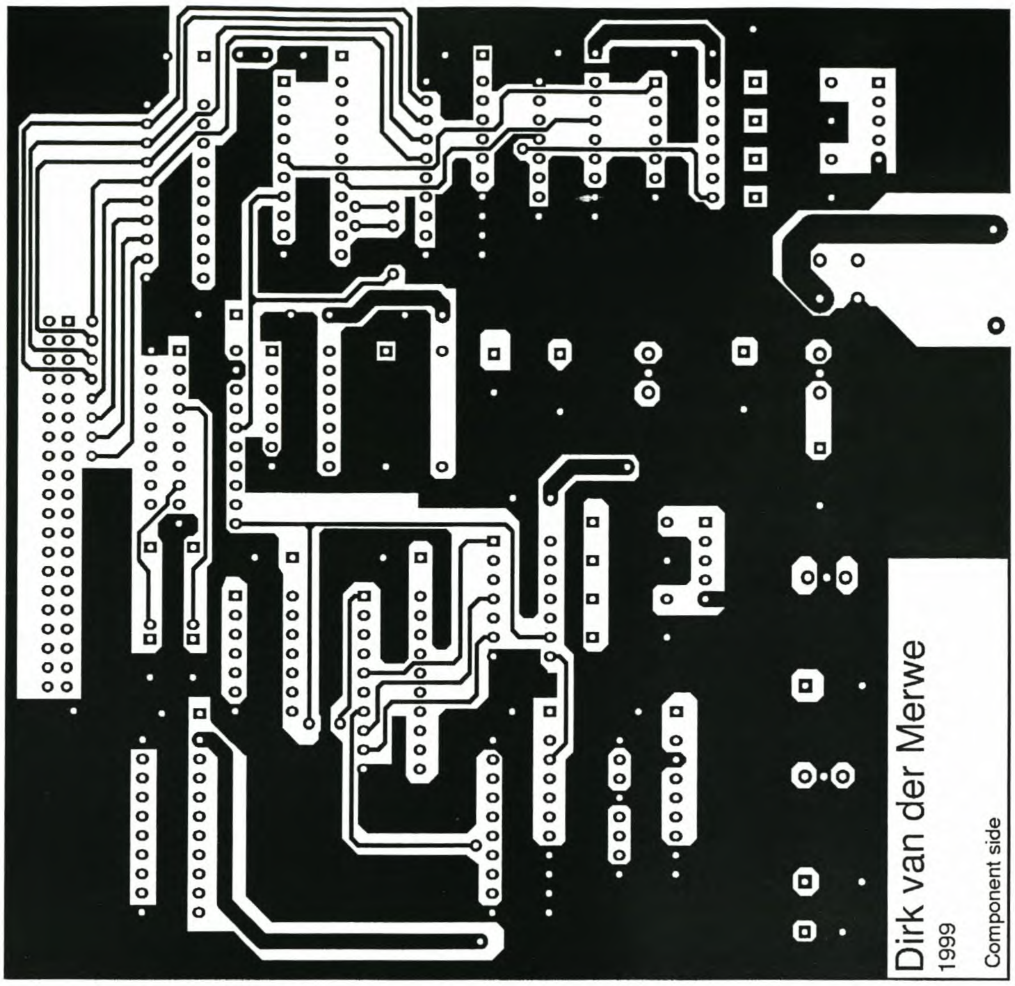


Figure F 1 PC board layout of the top of counter board version 1.01

F.1 THE COUNTER CIRCUIT

The counter circuit is used to count the pulses of the encoder. The circuit is described in section 6.1.1.

Diagrams for the PC board layout for version 1.01 of this board are shown in figures F 1 to F 2. There were two major errors on version 1.00:

1) The ground pin of one of the buffers was omitted. (Component U109 on the silk screen in Figure F 3.)

2) The connections for the diode bridge were connected incorrectly.

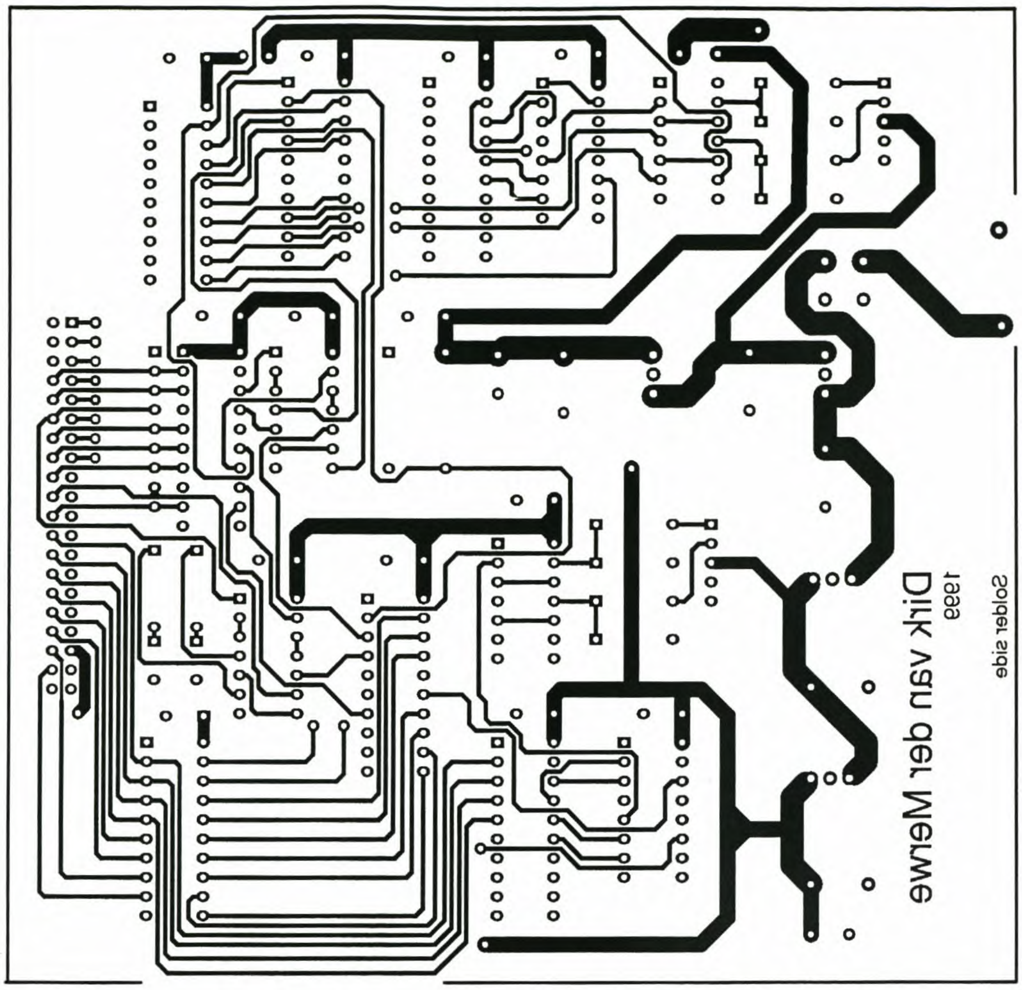


Figure F 2 PC board layout of the bottom of counter board version 1.01

These errors were rectified on version 1.01 of the board. Version 1.00 was, however, still used in the project. The unconnected lines were corrected by soldering wire-wrap to the connection points.

Components soldered on the board are listed in Table F 2 and the pin connections for the wire-wrapping are shown in Table F 1. The component layout for version 1.01 is the same as for version 1.00.

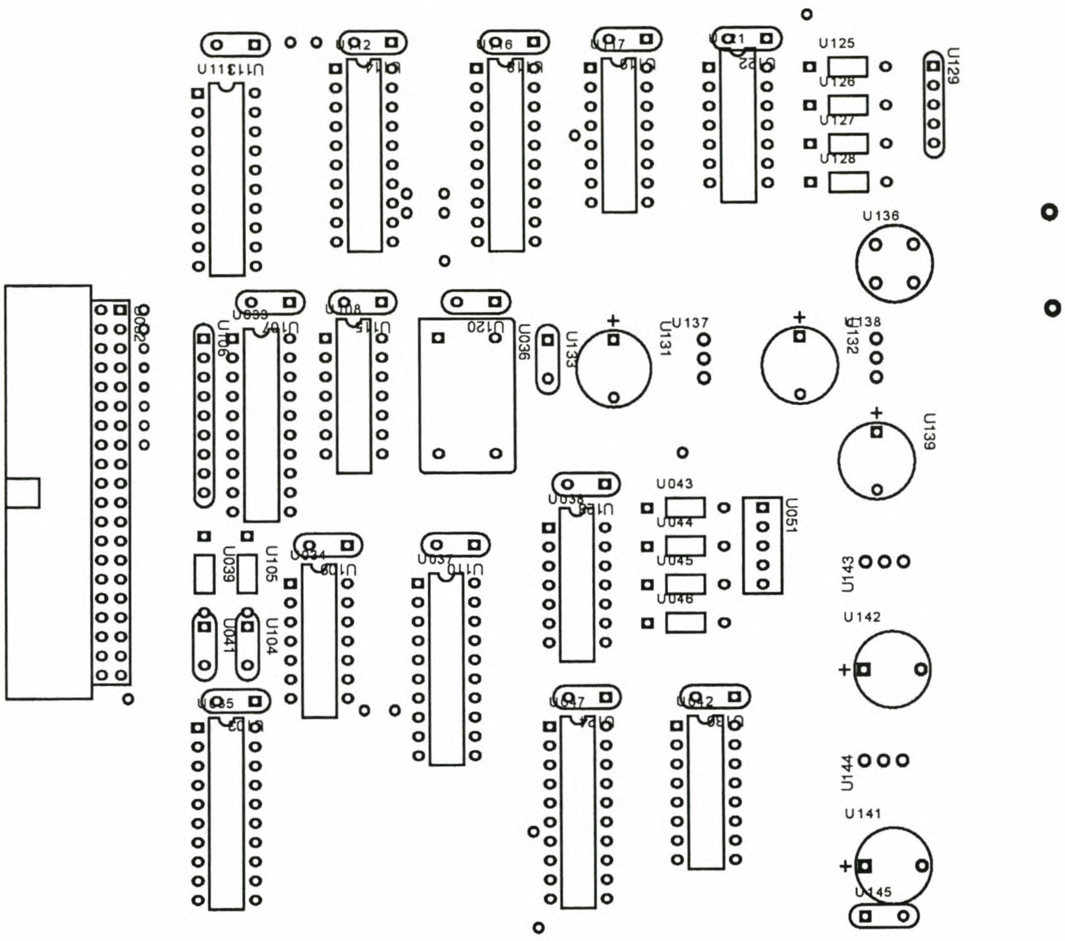


Figure F 3 PC board layout of the silk screen of counter board version 1.01

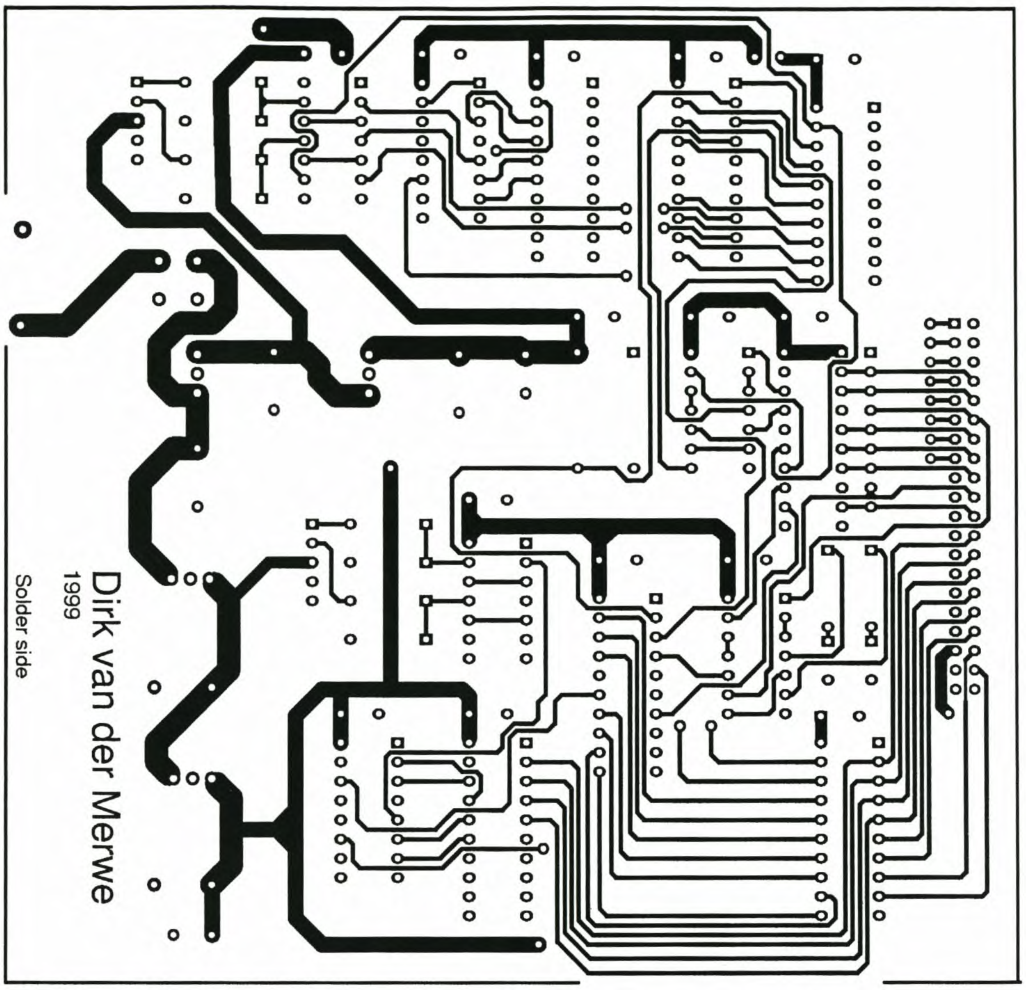


Figure F 4 PC board layout of the mirrored bottom of counter board version 1.01

Table F 1 Pin connections of wire-wrap connections on the counter version 1.01 PC board

	Pin on board	DB25 plug
U13	1	Channel A
	2	Channel B
	3	12V
	4	Ground
	5	Ground
U42	1	Channel A
	2	Channel B
	3	12V
	4	Ground
	5	Ground

Table F 2 Components for the counter board

No.	Component
U01	74HC245
U03	HCTL2020
U05	74HC245
U07	74HC191
U09	74HC14
U11	18kΩ resistor
U12	10kΩ resistor
U13	Pads for connections to encoder
U14	18kΩ resistor
U15	10kΩ resistor
U16	IDC PCB Header socket
U20	5A diode bridge rectifier
U21	DIL 10kΩ resistors
U22	74HC245
U23	74HC14
U24	5V 4Mhz Crystal Oscillator
U26	220μF 25V Electrolytic Capacitor
U27	7805 Voltage regulator
U28	220μF 25V Electrolytic Capacitor
U29	7812 Voltage regulator
U31	18kΩ resistor
U32	220μF 25V Electrolytic Capacitor
U33	1kΩ resistor
U34	1kΩ resistor
U35	74HC14
U37	HCTL2020
U39	74HC14
U40	10kΩ resistor
U41	18kΩ resistor
U42	Pads for connections to encoder
U43	7812 Voltage regulator
U44	100nF capacitor
U46	100nF capacitor
U48	10kΩ resistor
U50	220μF 25V Electrolytic Capacitor
U51	74HC245
U52	74HC245
U53	74HC191
U54	7805 Voltage regulator
U55	220μF 25V Electrolitic Capacitor
100nF capacitors were used as by-pass capacitors	

F.2 CONNECTION CIRCUITRY

This board connects the different boards to one another as discussed in section 6.2.

The PC board layout is given in figures F5 to F8.

The connection circuitry also had some errors that caused incorrect behaviour. The 74HC02 was found to be very susceptible to noise on its input lines. This error was first encountered after the design and construction of the PC board. The problem was remedied by soldering a capacitor from the input lines, pins 12 and 9, to ground. This circuitry also had no by-pass capacitors between its chip's power supply pins and ground. They should be inserted in future designs.

A list of components used is given in Table F 3.

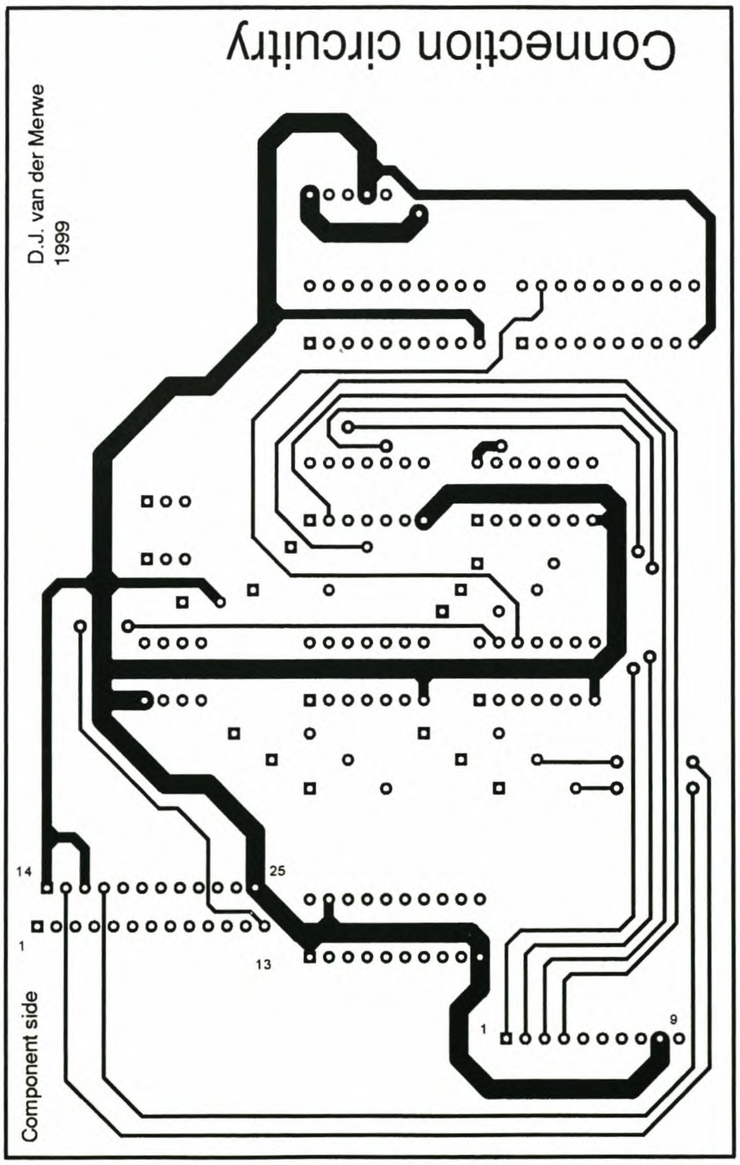


Figure F 5 PC board layout of the top of the connection circuitry version 1.00

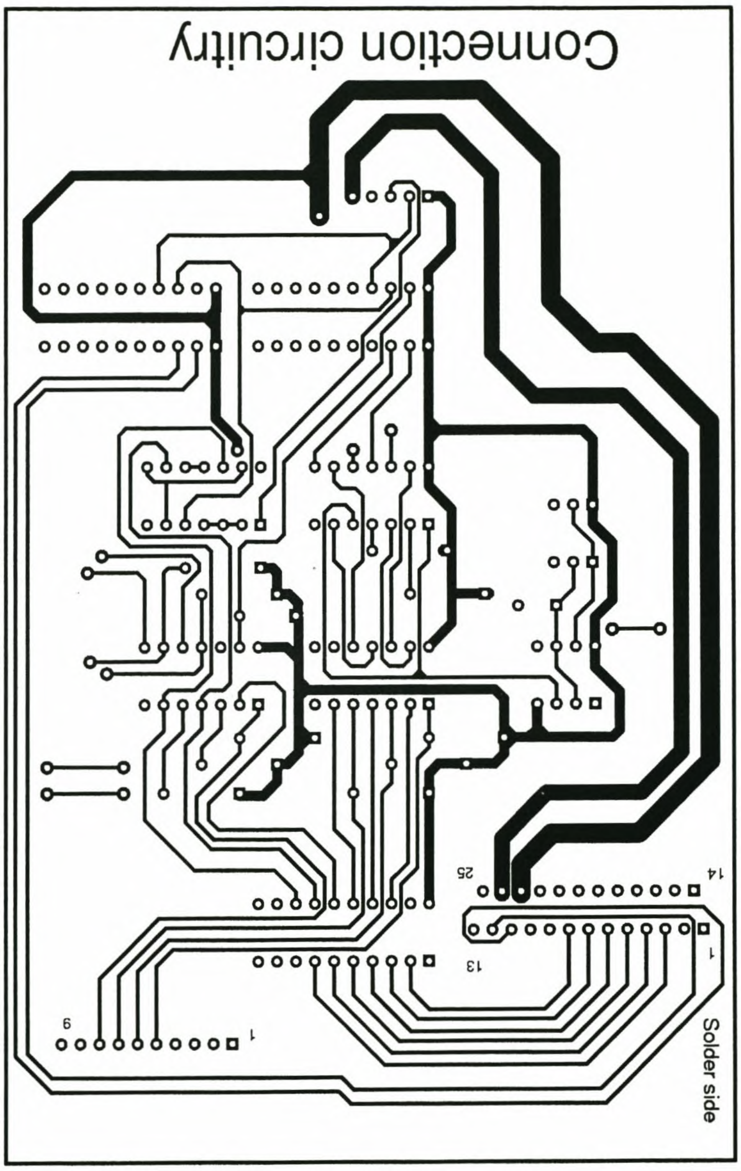


Figure F 6 PC board layout of the mirrored bottom of the connection circuitry version 1.00

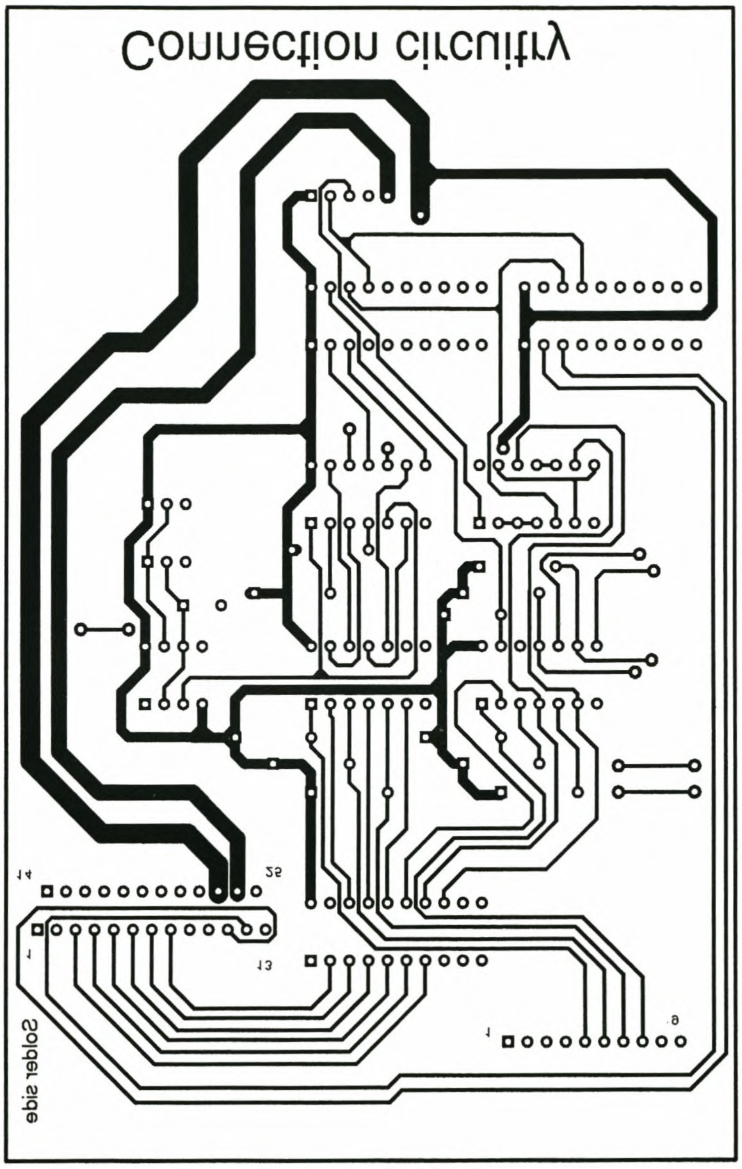


Figure F 7 PC board layout of the bottom of the connection circuitry version 1.00

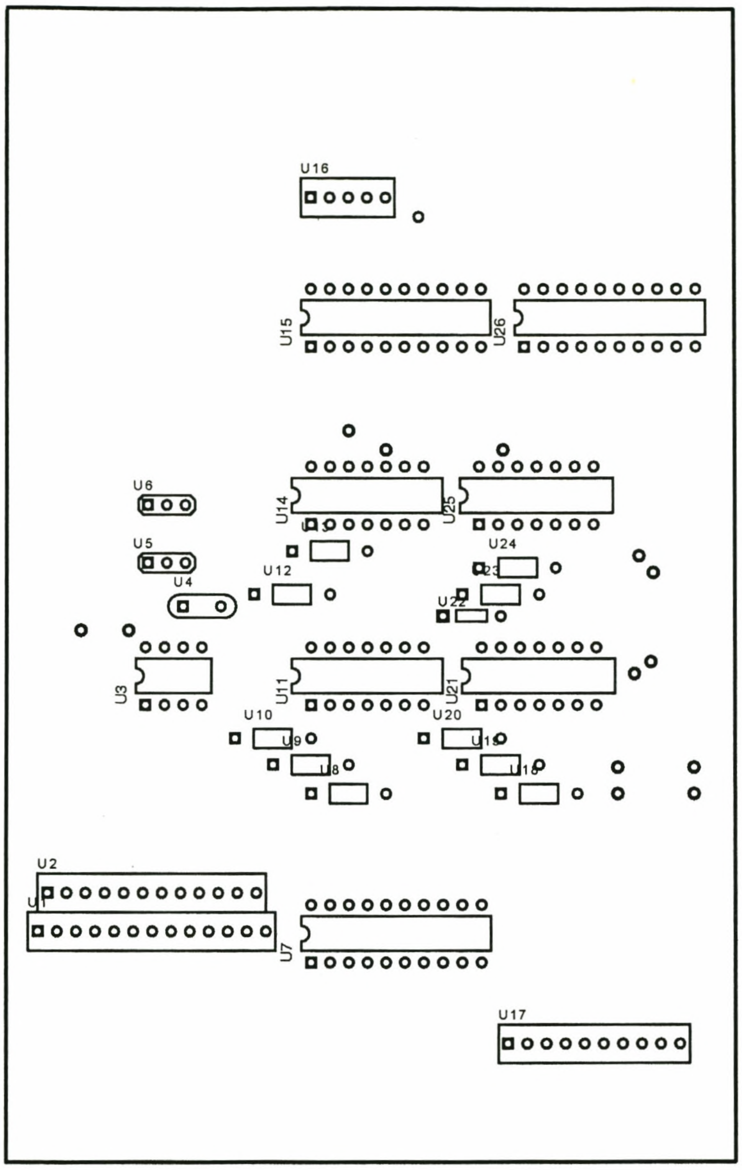


Figure F 8 PC board layout of the silk screen of the connection circuitry version 1.00

Table F 3 List of components for the connection circuitry board.

No.	Component
U1 & U2	Wire-wrap connection
U3	555-timer
U4	10nF capacitor
U5	1MΩ 22 turn precision potentiometer
U6	1MΩ 22 turn precision potentiometer
U7	74HC245
U8-10	10kΩ resistor
U11	74HC14
U12-13	10kΩ resistor
U14	74HC14
U15	74HC245
U16	Wire-wrap connection
U17	Wire-wrap connection
U18-20	10kΩ resistor
U21	74HC14
U22-24	10kΩ resistor
U25	74HC02
U26	74HC245

F.3 DRIVER CIRCUIT

The driver circuit drives the solenoid valve. It is described in section 6.3. The PC board diagrams for version 1.01 are shown in figures F9 to F12.

Version 1.00 had two major faults, which were rectified in version 1.01:

1) The diode, U12, in Figure F 12 was connected with both terminals to ground. This was rectified by cleaning away the surrounding copper on the ground pane at the positive pin.

2) The resistors, U9 and U17 in Figure F 12, were omitted in version 100. The copper connection was broken and two holes bored through the lines. The resistors were connected through these holes to restore connection.

A component list for version 1.01 appears in Table F 4.

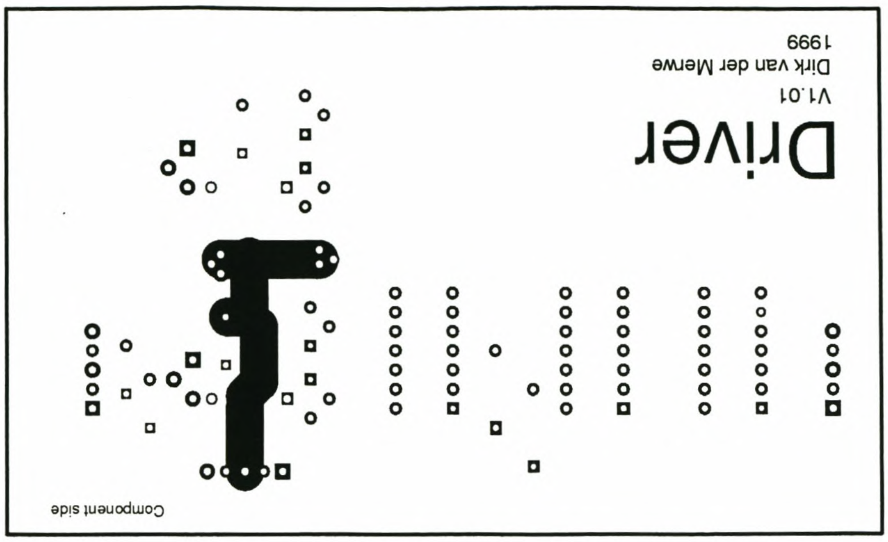


Figure F 9 PC board layout of the top of driver board version 1.01

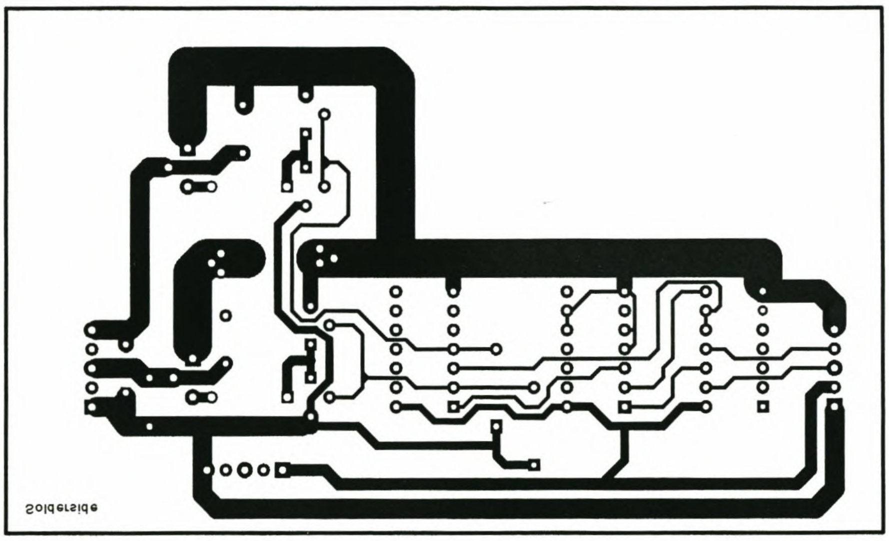


Figure F 10 PC board layout of the bottom of driver board version 1.01

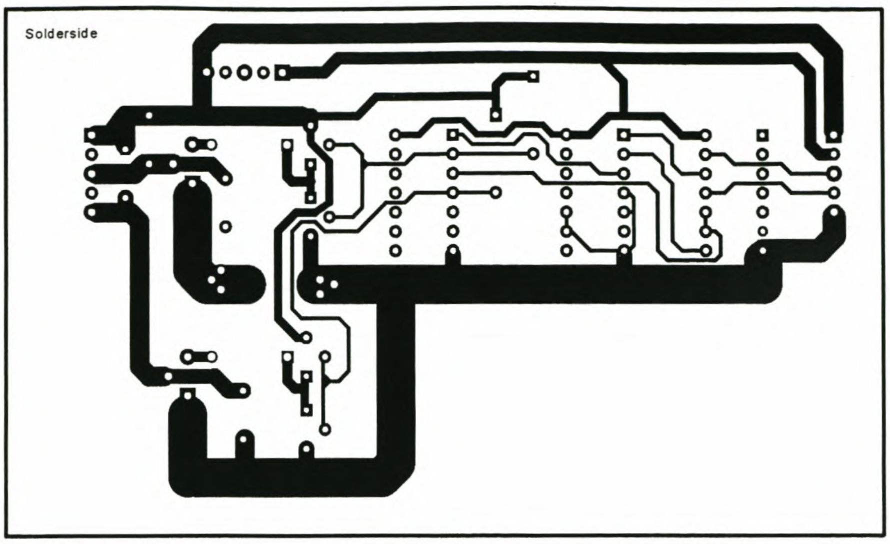


Figure F 11 PC board layout of the mirrored bottom of driver board version 1.01

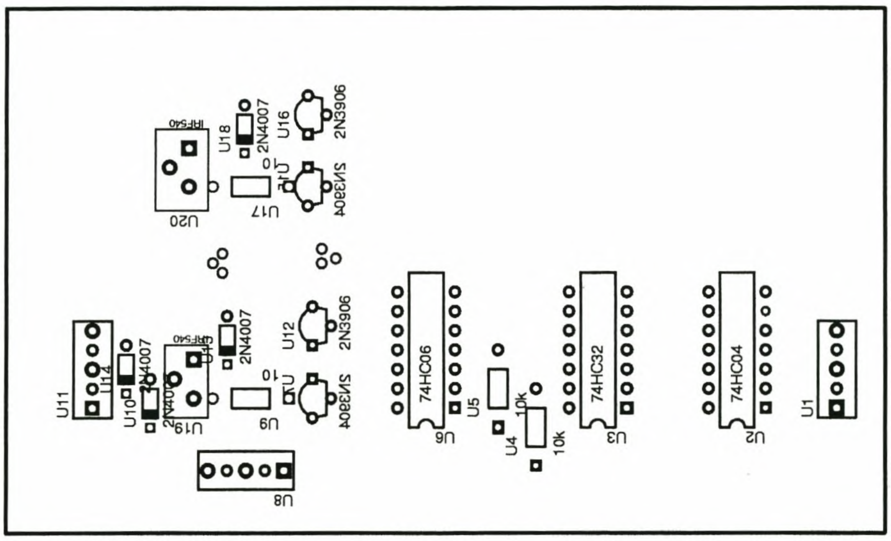


Figure F 12 PC board layout of the silk screen of driver board version 1.00

Table F 4 List of components for the driver board

No.	Component
U1	Wire-wrap connection
U2	74HC04
U3	74HC32
U4	10kΩ resistor
U5	10kΩ resistor
U6	74HC06
U7	2N3904 transistor
U8	Flex wire connection
U9	10Ω resistor
U10	1N4007 diode
U11	Flex wire connection
U12	2N3906 transistor
U13	1N4007 diode
U14	1N4007 diode
U15	2N3904 transistor
U16	2N3906 transistor
U17	10Ω resistor
U18	1N4007 diode
U19	IRF540 mosfet
U20	IRF540 mosfet

APPENDIX G

DESCRIPTION OF THE BENDING PRESS STRUCTURE

An overview of the bending press structure was given in section 4.2. More photos of the bending machine are discussed and shown here.

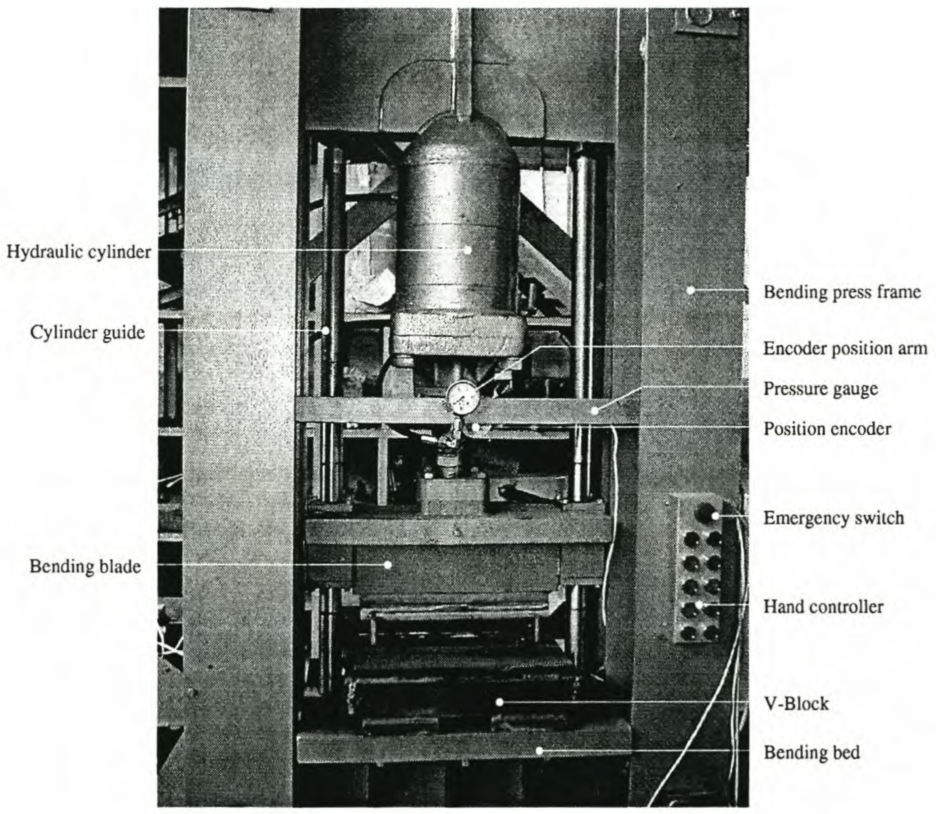


Figure G 1 Front view of the bending machine

Figure G 1 shows a close-up view of the front of the bending machine. A number of features can be identified:

Hydraulic cylinder: The hydraulic cylinder forces the bending blade down on the metal flat bar lying on the bending bed.

Cylinder guides: The hydraulic cylinder is not capable of handling the lateral forces on the bending blade during bending. Cylinder guides are therefore used to keep the bending blade in its horizontal position.

Encoder: The encoder is placed as close as possible to the horizontal centre of the bending blade. This minimises incorrect readings due to deflection on the bending machine.

Encoder position arm: The encoder is attached to the bending machine via the encoder position arm. Measurements are carried out between the encoder and the bending blade. If the encoder was fixed to the hydraulic valve, readings would be less accurate due to deflections on the hydraulic valve.

Blade: The blade of the press used for bending.

Emergency switch: The emergency switch sets the machine control in emergency mode.

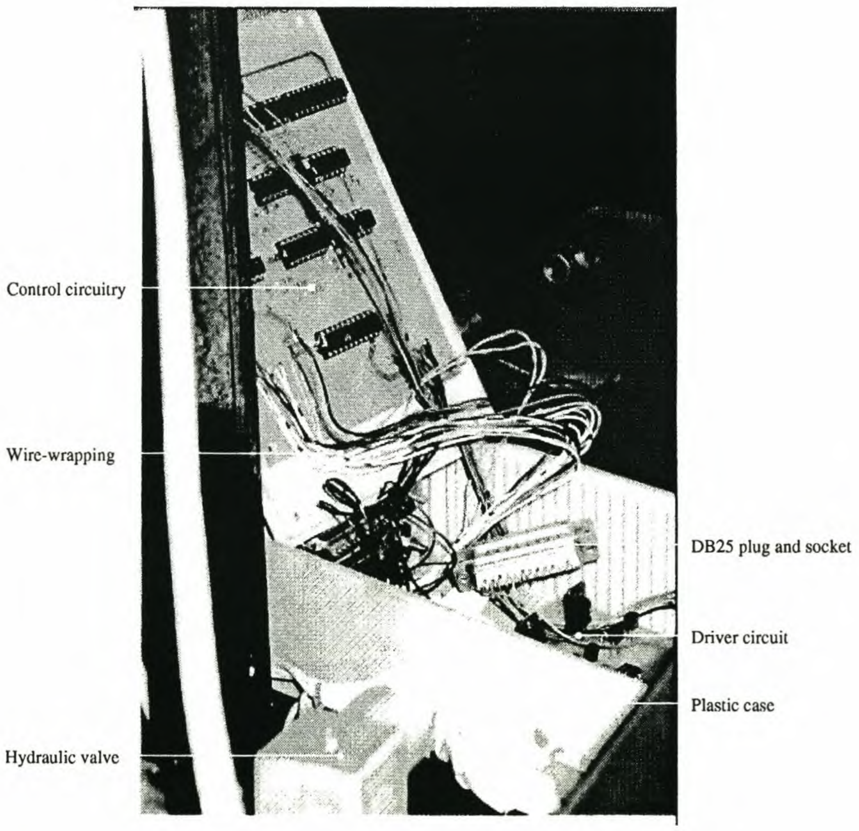


Figure G 2 Cconstruction of the control box

Bending press bed: The V-block is placed on the bending bed.

V-Block: The metal flat bar is placed on the V-Block when bending. Different V-Blocks may be used.

Metal flat bar: The metal flat bar to be bent.

Bending press frame: Framework of the bending press.

The plastic control box shown in Figure G 2 is placed in a metal housing on the press.

The control circuitry is attached to the lid of the control box, while the driver circuitry is placed in the bottom of the box. Wire-wrapping and DB25 plugs are used to connect the two circuits.

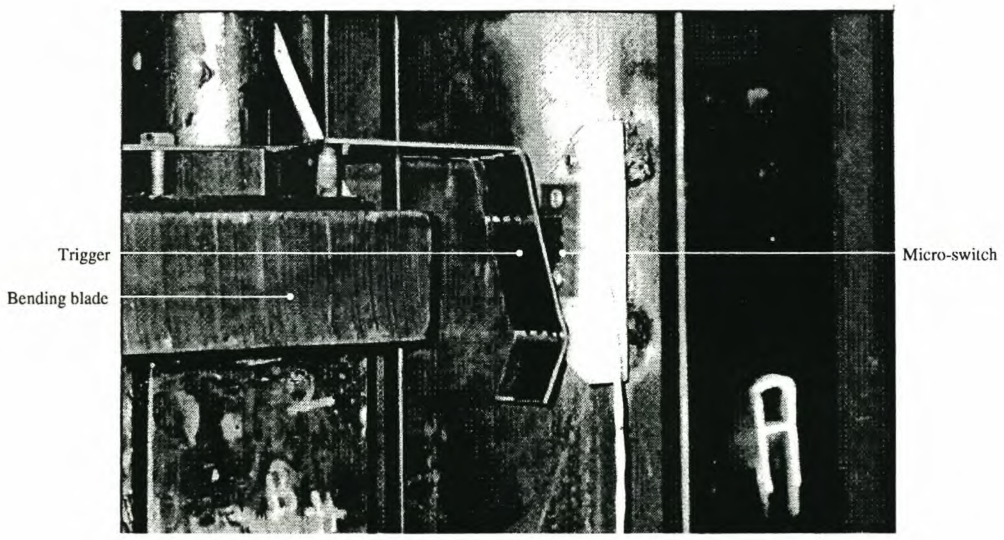


Figure G 3 Implementation of the micro-switch

Figure G 3 shows the implementation of the micro-switch. The trigger moves up and down with the bending blade. When the trigger, which is spring-loaded, moves over the micro-switch, it activates the switch to indicate the top of the stroke.

The construction of the encoder set-up is shown in Figure G 4. The encoder is attached to the frame of the bending machine by means of the encoder position arm. A spring-loaded pulley connects the encoder by a stainless steel cable to the bending blade. The circumference

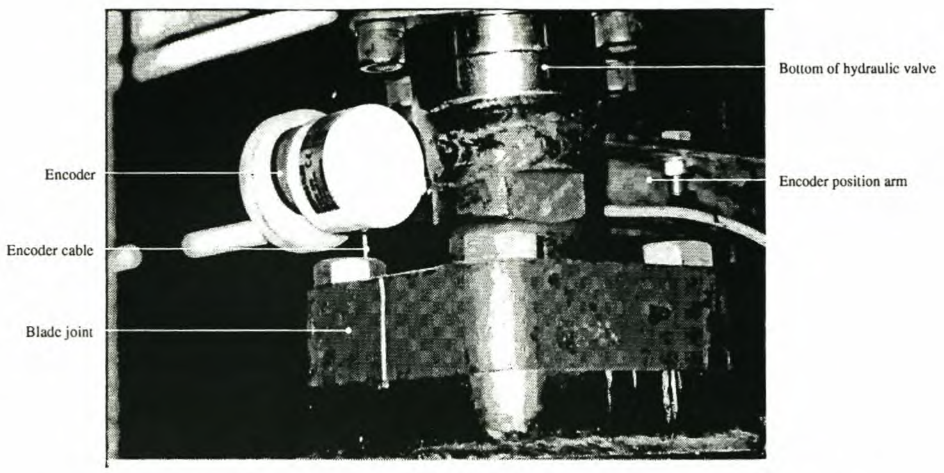


Figure G 4 Construction of the encoder set-up on the bending machine

of the pulley is chosen so that the pulley completes one rotation over the full stroke of the bending blade.

The encoder cable is made of stainless steel to prevent any inaccuracies due to corrosion of the cable.

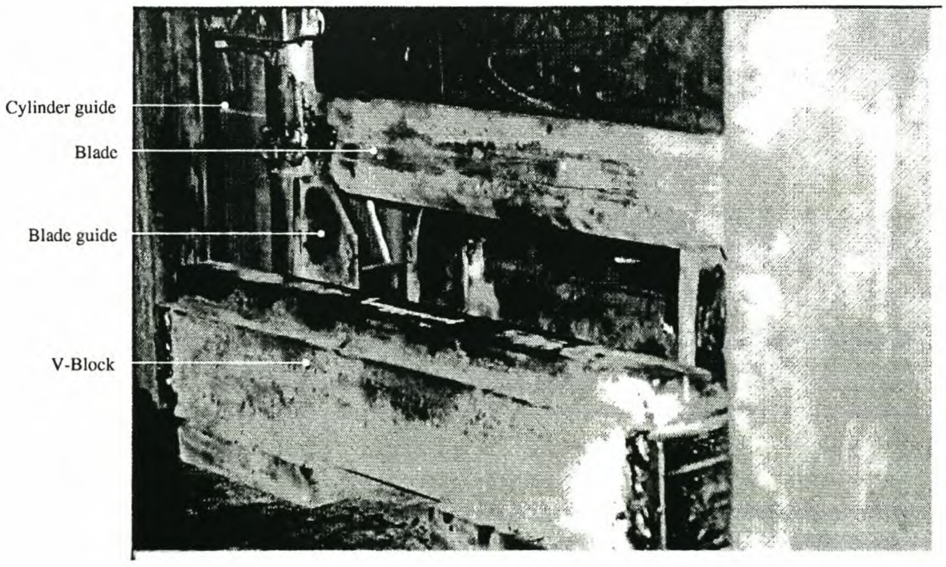


Figure G 5 A close-up view of the bending machine bed

Figures G5 and G6 show photos of the bending machine bed. The V-block has guides to ensure that it stays aligned with the bending blades.

The V-block aligns with the bending blade when the blade moves downwards through the V-block guides.

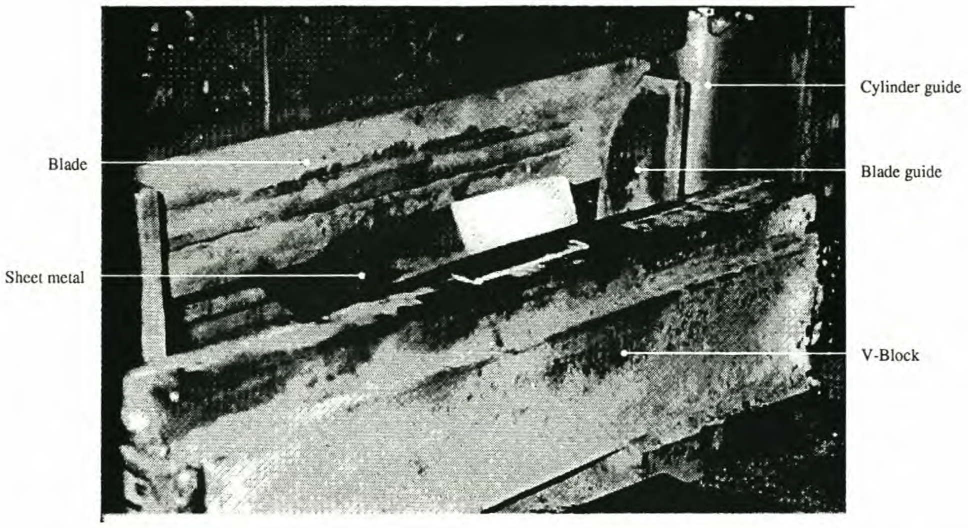


Figure G 6 A close-up view of the bending machine bed while bending a piece of sheet metal

Metal flat bars for bending are placed on the bed as shown in Figure G 6. The blade move downwards to bend the metal on the V-block.

Figure G 7 shows the position stop. It is used to indicate the position on the flat bar

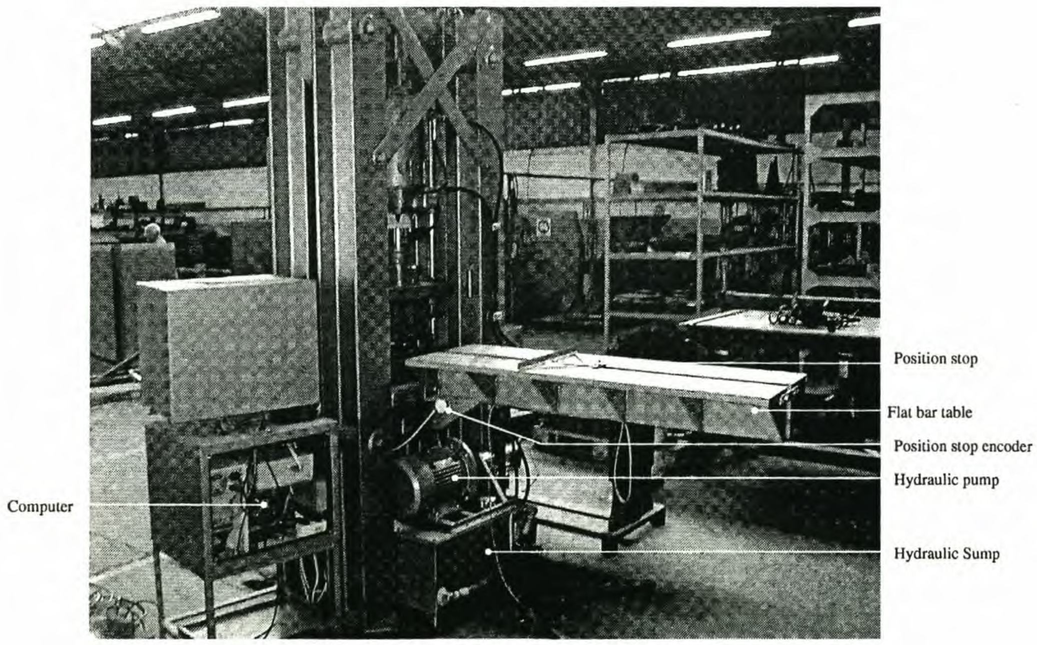


Figure G 7 View of the position stop and flat bar table where the bending process should be executed. The position stop is moved by a weight when its solenoid is released. The position is measured by a cable-and-pulley system on an encoder, similar to the bending blade encoder. The computer is normally enclosed in a metal housing like the screen in the top housing. It was opened for illustration purposes.

APPENDIX H

RESULTS OF REGRESSION DONE ON BEND EQUATION

H.1 DATA USED

Data used for the regression analysis is listed in Table H 1. Only one measurement was done for each observation. It is therefore fairly possible that there are some reading errors in the data. The data was obtained from measurements on 25mm x 80mm steel flat bars.

The correct procedure would be to do more replications of the data point measurements, but results obtained were satisfactory despite the small number of observations.

The angle is denoted as y and the displacement of the bending blade as x in the regression equations. The equations are forced to have a value of zero at the y-axis intercept.

H.2 REGRESSION FOR $Y = AX$

The equation does not fit the data well as shown in Table H 2 and Figure H 1.

- The mean square value for the residuals is quite large in comparison with

Table H 1 Data used during the various regression analysis methods

Angle (y)	Displacement (x)
0.	0
0.	0.98
0.5	1.98
2.5	4.04
6.0	6.94
10.5	10.05
16.6	15.06
24.0	20.05
31.0	25.0
37.0	30.14
45.0	35.47

results obtained for other equations.

- The deviations of the real values from the equation are not only due to noise in the readings. Trends in deviations can be identified by the large number of consecutive residuals with the same sign.
- The residuals for measurements between 0-10mm are large and will cause reading errors of larger that 0.5^0.

Table H 2 Regression done for the equation $y = ax$

SUMMARY OUTPUT

Regression Statistics

Multiple R	0.994782982
R Square	0.989593181
Adjusted R Square	0.889593181
Standard Error	1.662756992
Observations	11

ANOVA

	df	SS	MS	F	Significance F
Regression	1	2629.03421	2629.03421	950.9083737	1.94316E-10
Residual	10	27.64760815	2.764760815		
Total	11	2656.681818			

	Coefficients	**Standard Error**	**t Stat**	**P-value**
Intercept	0	#N/A	#N/A	#N/A
X Variable 1	1.222415279	0.027748423	44.0535043	8.73103E-13

	Lower 95%	**Upper 95 %**
Intercept		
X Variable 1	1.160587928	1.284242629

RESIDUAL OUTPUT

Observation	**Predicted Y**	**Residuals**	**Standard Residuals**
1	0	0	0
2	1.197966973	-1.197966973	-0.75563559
3	2.420382252	-1.920382252	-1.211309835
4	4.938557726	-2.438557726	-1.538156768
5	8.483562035	-2.483562035	-1.566543908
6	12.28527355	-1.785273552	-1.126088001
7	18.4095741	-1.909574098	-1.204492431
8	24.50942634	-0.50942634	-0.321328285
9	30.56038197	0.43961803	0.27729565
10	36.8435965	0.656403497	0.414036326
11	43.35906994	1.640930062	1.035041185

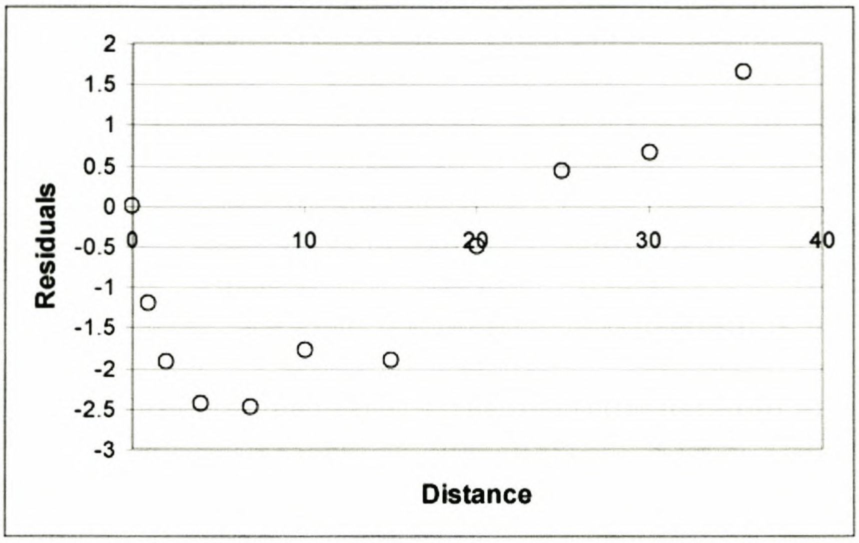


Figure H 1 Graph of the residuals for y = ax

H.3 REGRESSION FOR $Y = AX^2+BX$

This second-order equation also fits the data poorly. The results for the regression are shown in Table H 3 and Figure H 2.

- The mean square error for the residuals is still high (>1) in comparison to other equations tested.
- Trends can be identified in the signs of the residuals, indicating that the

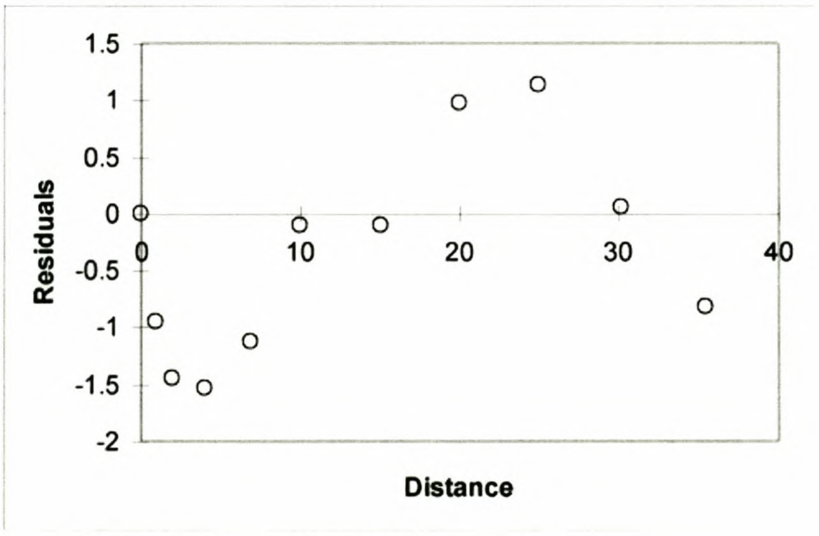


Figure H 2 Graph of the residuals for $y = ax^2 + bx$

equation does not account for all observations.

- Many residual values are also large, causing reading errors of more than 0.

Table H 3 Regression done for the equation $y = ax^2 + bx$

SUMMARY OUTPUT					
Regression Statistics					
		Multiple R	0.998200082		
		R Square	0.996403404		
		Adjusted R Square	0.884892671		
		Standard Error	1.03037274		
		Observations	11		
ANOVA					
	df	**SS**	**MS**	**F**	**Significance F**
Regression	2	2647.126806	1323.563403	1246.682978	1.04629E-10
Residual	9	9.555011848	1.061667983		
Total	11	2656.681818			
	Coefficients	**Standard Error**	**t Stat**	**P-value**	
Intercept	0	#N/A	#N/A	#N/A	
X Variable 1	0.962547949	0.06525619	14.75029336	1.30506E-07	
X Variable 2	0.009281892	0.002248435	4.128155863	0.002566534	
	Lower 95%	**Upper 95%**			
Intercept	#N/A	#N/A			
X Variable 1	0.814928078	1.110167819			
X Variable 2	0.004195574	0.01436821			

Observation	Predicted Y	Residuals	Standard Residuals
1	0	0	0
2	0.952211319	-0.952211319	-1.02167806
3	1.942233668	-1.442233668	-1.547449045
4	4.040189043	-1.540189043	-1.652550566
5	7.127132102	-1.127132102	-1.209359852
6	10.61110119	-0.11110119	-0.119206363
7	16.60113865	-0.101138647	-0.108517022
8	23.03043019	0.969569807	1.040302903
9	29.86488127	1.135118732	1.217929131
10	37.44304786	0.056952136	0.061106969
11	45.81931816	-0.819318157	-0.879089933

H.4 REGRESSION FOR $Y = AX^3+BX^2+CX$

This equation fits the data much better than previous equations. However, the equation still cannot explain data in the region between 0-10mm as shown in Table H 4 and Figure H 3.

- The mean square error for the residuals is much smaller than found in previous equations, indicating that the equation fits the data better.

- The residuals again have trends indicating that some data is not accounted for by the equation. A definite trend can be identified in the region of 0-10mm

Table H 4 Regression done for the equation $y = ax^3 + bx^2 + cx$

SUMMARY OUTPUT					
Regression Statistics					
	Multiple R	0.999452581			
	R Square	0.998905462			
	Adjusted R Square	0.873631828			
	Standard Error	0.602892919			
	Observations	11			
ANOVA					
	df	**SS**	**MS**	**F**	**Significance F**
Regression	3	2653.773979	884.5913264	2433.673485	6.33001E-11
Residual	8	2.907838975	0.363479872		
Total	11	2656.681818			
	Coefficients	**Standard Error**	**t Stat**	**P-value**	
Intercept	0	#N/A	#N/A	#N/A	
X Variable 1	0.628625073	0.086920648	7.232171986	8.95944E-05	
X Variable 2	0.04043721	0.007403242	5.462094794	0.00060004	
X Variable 3	-0.000639108	0.00014945	-4.276399831	0.002700648	
	Lower 95 %	**Upper 95 %**			
Intercept	#N/A	#N/A			
X Variable 1	0.42818557	0.829064576			
X Variable 2	0.023365292	0.057509127			
X Variable 3	-0.00098374	-0.000294475			

RESIDUAL OUTPUT			
Observation	**Predicted Y**	**Residuals**	**Standard Residuals**
1	0	0	0
2	0.654286945	-0.654286945	-1.272563377
3	1.398246677	-0.898246677	-1.747055836
4	3.157502968	-0.657502968	-1.278818422
5	6.096634426	-0.096634426	-0.187950306
6	9.753198976	0.746801024	1.452499766
7	16.45542247	0.044577529	0.08670161
8	23.70848848	0.291511524	0.566978897
9	31.00282581	-0.002825807	-0.005496087
10	38.18209509	-0.682095093	-1.326649174
11	44.65170054	0.34829946	0.677429285

where it is suspected that the function should produce lower output values.

- Some of the residuals, again especially in the 0-10mm region, have large values causing reading errors of larger that 0.5^0.

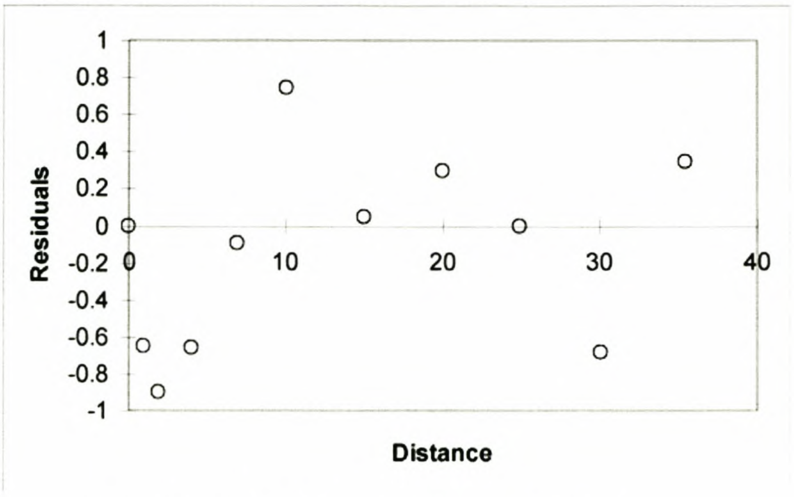


Figure H 3 Graph of the residuals for $y = ax^3+bx^2+cx$

H.5 REGRESSION FOR $Y = AX^4+BX^4+CX^2+DX$

Results for the regression are shown in Table H 5 and Figure H 4. A graph of the equation is shown in Figure H 4 where the values can be compared visually with real data.

- The mean square error for the residuals is low in comparison with results

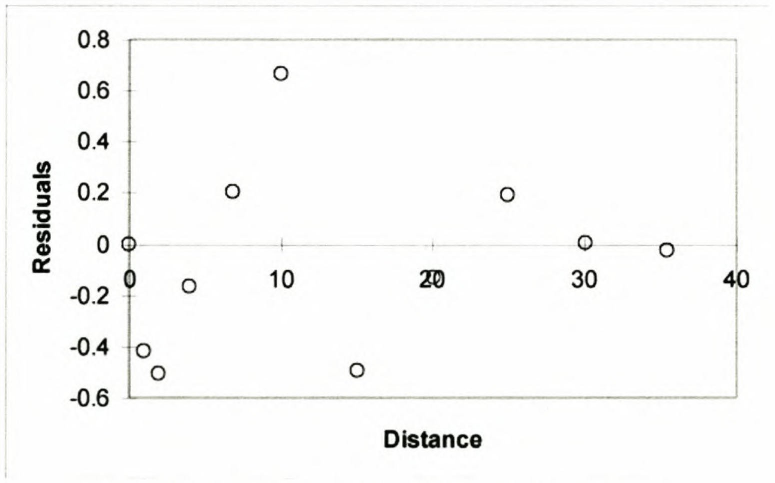


Figure H 4 Graph of the residuals for $y = ax^4+bx^3+cx^2+dx$

obtained for previous equations. The equation therefore fits the data much better.

- Trends can still be identified in the residuals, and again it is the 0-10mm region that is not well explained by the data. Trends can also be identified in other regions indicated by consecutive points having the same signs, but there are not enough observations to give a decisive answer. By looking at Figure H 5 it is suspected that the equation has some errors in describing the observations.

Table H 5 Regression done for the equation $y = ax^4 + bx^3 + cx^2 + dx$

SUMMARY OUTPUT				
Regression Statistics				
	Multiple R	0.999766981		
	R Square	0.999534016		
	Adjusted R Square	0.856477166		
	Standard Error	0.420538795		
	Observations	11		
ANOVA				
	df	**SS**	**MS**	**F**	**Significance F**
Regression	4	2655.443848	663.860962	3753.747002	2.54852E-10
Residual	7	1.237970149	0.176852878		
Total	11	2656.681818			
	Coefficients	**Standard Error**	**t Stat**	**P-value**	
Intercept	0	#N/A	#N/A	#N/A	
X Variable 1	0.336689926	0.112703897	2.987384962	0.020301377	
X Variable 2	0.092282651	0.017644918	5.229984608	0.001212469	
X Variable 3	-0.003208556	0.000842663	-3.80763969	0.006648303	
X Variable 4	3.80091E-05	1.23695E-05	3.072805788	0.017996386	
	Lower 95 %	**Upper 95 %**			
Intercept	#N/A	#N/A			
X Variable 1	0.070187749	0.603192103			
X Variable 2	0.050559079	0.134006224			
X Variable 3	-0.005201135	-0.001215977			
X Variable 4	8.75987E-06	6.72583E-05			

RESIDUAL OUTPUT			
Observation	**Predicted Y**	**Residuals**	**Standard Residuals**
1	0	0	0
2	0.415599577	-0.415599577	-1.238843234
3	1.004109072	-0.504109072	-1.502677451
4	2.66498341	-0.16498341	-0.491792083
5	5.796986523	0.203013477	0.605154302
6	9.835336519	0.664663481	1.981267312
7	16.99642586	-0.496425858	-1.479774883
8	24.12954023	-0.129540233	-0.386141011
9	30.80750998	0.192490016	0.573785363
10	37.49576692	0.004233077	0.012618199
11	45.02490006	-0.024900063	-0.074223546

- Most residuals are small ($<0.5^0$), which suggests that the accuracy of the equation is high enough for use in the bending equation.

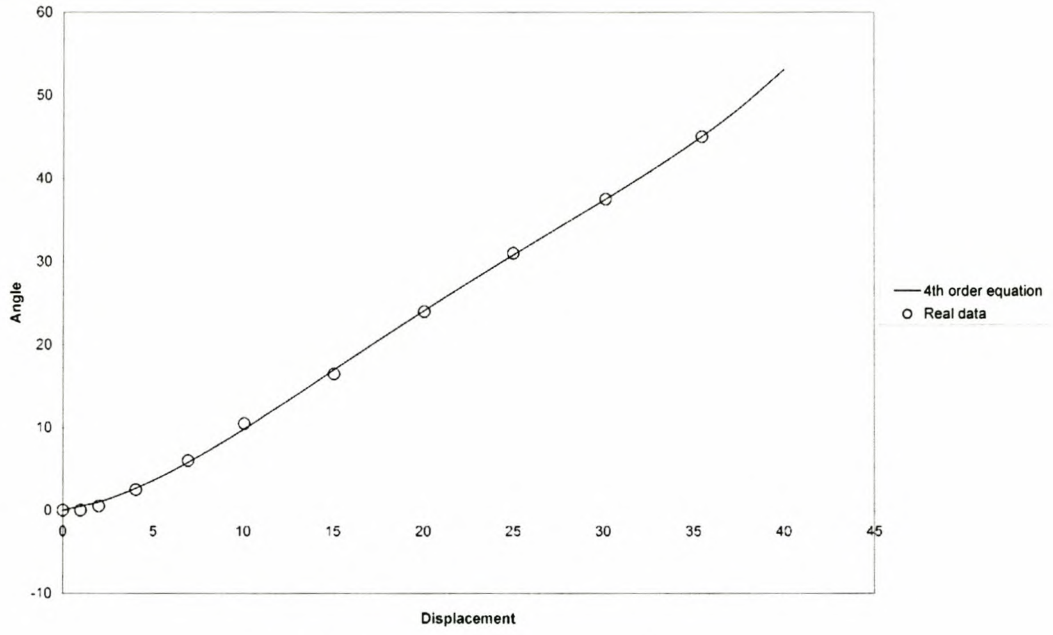


Figure H 5 Graph of $y = ax^4+bx^3+cx^2+dx$

H.6 REGRESSION FOR $Y = AX^{10} + BX^9 + CX^8 + DX^7 + EX^6 + FX^5 + GX^4+HX^3+IX^2+JX$

This equation describes the data extremely well. It should, however, be noted from Figure H 7 that the equation is inadequate. From experience one should guess that the equation should be more or less linear with some deviations in the 0-10mm region. This equation was included only to illustrate the problem of using an equation of too high an order. The following results for the regression analysis are obtained from Table H 1 and Figure H 6:

- The mean square value for the residuals is very low, indicating that the data points are fitted very well. This is known not to be true (as argued above) – only the observations fit the equation well.

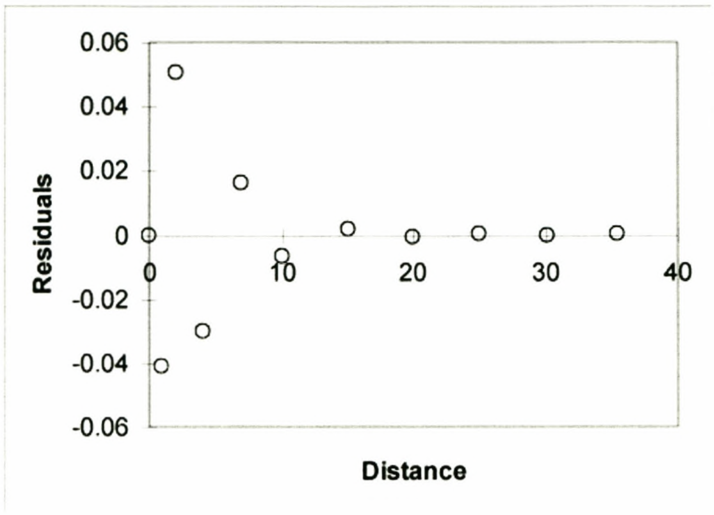


Figure H 6 Graph of the residuals for $y = ax^{10} + bx^9 + cx^8 + dx^7 + ex^6 + fx^5 + gx^4 + hx^3 + ix^2 + jx$

- No trends can be identified.
- Residual values are small, falsely indicating a good fit.

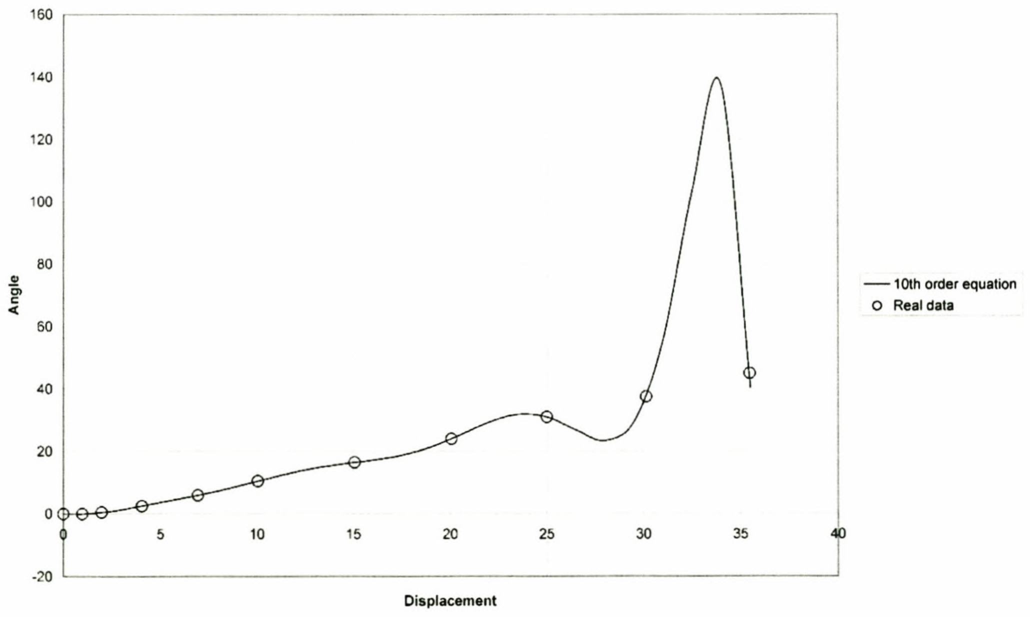


Figure H 7 Graph of for $y = ax^{10} + bx^9 + cx^8 + dx^7 + ex^6 + fx^5 + gx^4 + hx^3 + ix^2 + jx$

Table H 6 Regression done for the equation for $y = ax^{10} + bx^9 + cx^8 + dx^7 + ex^6 + fx^5 + gx^4 + hx^3 + ix^2 + jx$

SUMMARY OUTPUT

Regression Statistics

Multiple R	0.999998966
R Square	0.999997933
Adjusted R Square	-2.06707E-05
Standard Error	0.074105056
Observations	11

ANOVA

	df	SS	MS	F	Significance F
Regression	10	2656.676327	265.6676327	48377.44932	#NUM!
Residual	1	0.005491559	0.005491559		
Total	11	2656.681818			

	Coefficients	Standard Error	t Stat	P-value
Intercept	0	#N/A	#N/A	#N/A
X Variable 1	0.112784816	0.508287631	0.221891718	0.860991544
X Variable 2	-0.326091425	0.730997359	-0.446091113	0.732875272
X Variable 3	0.347149462	0.373827805	0.928634674	0.523546118
X Variable 4	-0.104322901	0.09384358	-1.111667966	0.466365981
X Variable 5	0.016147367	0.013186748	1.224514718	0.435964397
X Variable 6	-0.00144189	0.001101622	-1.308878827	0.415336508
X Variable 7	7.6632E-05	5.57065E-05	1.375636704	0.400163966
X Variable 8	-2.38323E-06	1.66749E-06	-1.429232118	0.388661947
X Variable 9	3.99195E-08	2.71114E-08	1.472424203	0.379805213
X Variable 10	-2.77441E-10	1.84064E-10	-1.507305558	0.37290786
	Lower 95 %	**Upper 95%**		
Intercept	#N/A	#N/A		
X Variable 1	-6.345594215	6.571163846		
X Variable 2	-9.614253741	8.96207089		
X Variable 3	-4.402762818	5.097061742		
X Variable 4	-1.296713531	1.088067728		
X Variable 5	-0.151405435	0.183700169		
X Variable 6	-0.015439269	0.012555489		
X Variable 7	-0.000631184	0.000784448		
X Variable 8	-2.35707E-05	1.88042E-05		
X Variable 9	-3.04562E-07	3.84402E-07		
X Variable 10	-2.61619E-09	2.06131E-09		

RESIDUAL OUTPUT

Observation	Predicted Y	Residuals	Standard Residuals
1	0	0	0
2	0.041244273	-0.041244273	-1.845916962
3	0.449329511	0.050670489	2.267793995
4	2.530169187	-0.030169187	-1.350243527
5	5.983774854	0.016225146	0.726168029
6	10.50680756	-0.006807555	-0.304677007
7	16.49829721	0.001702785	0.076209373
8	24.00052564	-0.000525637	-0.023525244
9	30.99987119	0.000128813	0.005765113
10	37.50001939	-1.93912E-05	-0.000867865
11	44.99999864	1.35566E-06	6.06734E-05

H.7 REGRESSION FOR $Y=AX+B-Be^{-CX}$

The best results were obtained with this equation. The analysis was done in *Statistica,* a statistical software tool. Results for the regression are shown in Figure H 8 and Table H 7.

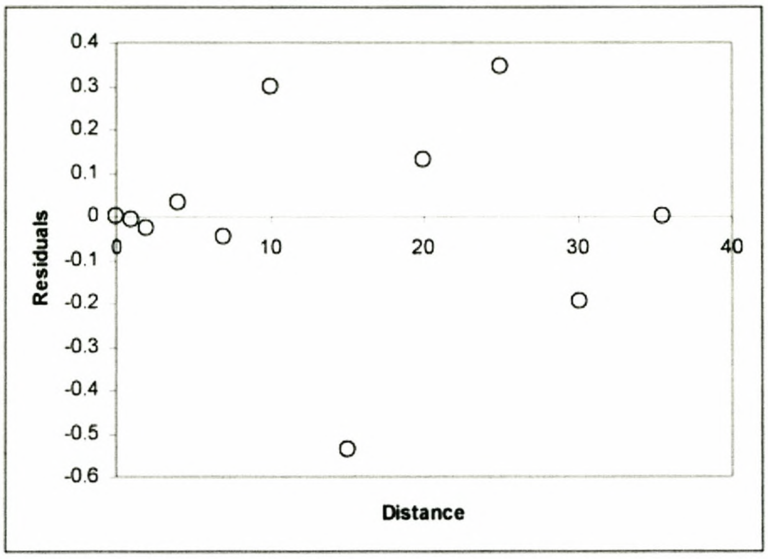


Figure H 8 Graph of the residuals for Angle = $ax+b-be^{-cx}$.

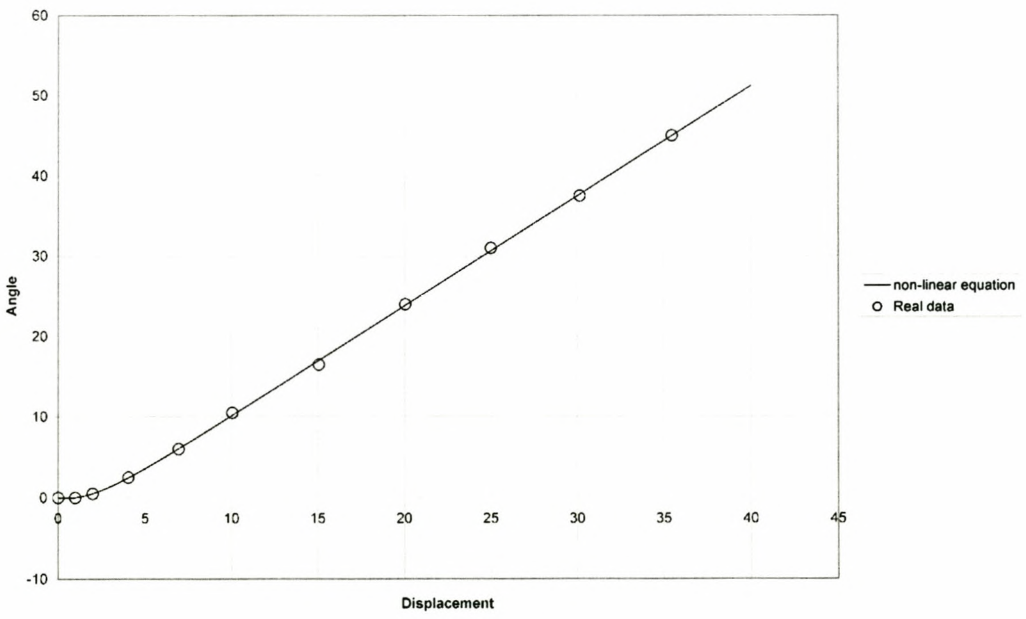


Figure H 9 Graph of Angle = $ax+b-be^{-cx}$.

- The mean square error of the residuals is small in comparison with previous values.
- No trends can be identified in the residual signs. This also suggests a good fit. Any deviations can consequently be attributed to inaccurate readings. The distribution for the residuals is shown in Figure H 10.
- Residual values are small, suggesting quite accurate prediction of the real data.

A graph of the equation is shown in Figure H 8.

Table H 7 Regression done for the equation $y=ax+b-be^{-cx}$.

Model: $y=ax+b-be^{-cx}$
Dep. var: ANGLE Loss: $(OBS\text{-}PRED)^2$
Final loss: 0.556559663 R=.99990 Variance explained: 99.979%

	A	B	C
Estimate	1.370087945	-3.598443559	0.472564996

Means and Standard Deviations

	Mean	**St. dev**	**Minimum**	**Maximum**
Displacement	13.60999966	12.46259212	0	35.47000122
Angle	15.77272701	16.29933167	0	45

ANOVA

	df	**SS**	**MS**
Regression	3	2656.125259	664.0313146
Residual	8	0.556559663	0.069569958
Total	11	2656.681818	

RESIDUAL OUTPUT

Observation	**Residuals**	**Predicted Y**
1	0	0
2	-0.008819004	0.008819004
3	-0.026070189	0.526070178
4	0.029982937	2.470016956
5	-0.045423962	6.045423985
6	0.297904611	10.20209503
7	-0.538000405	17.03800011
8	0.127904072	23.87209511
9	0.346218318	30.65378189
10	-0.196009442	37.69601059
11	0.001423969	44.99857712

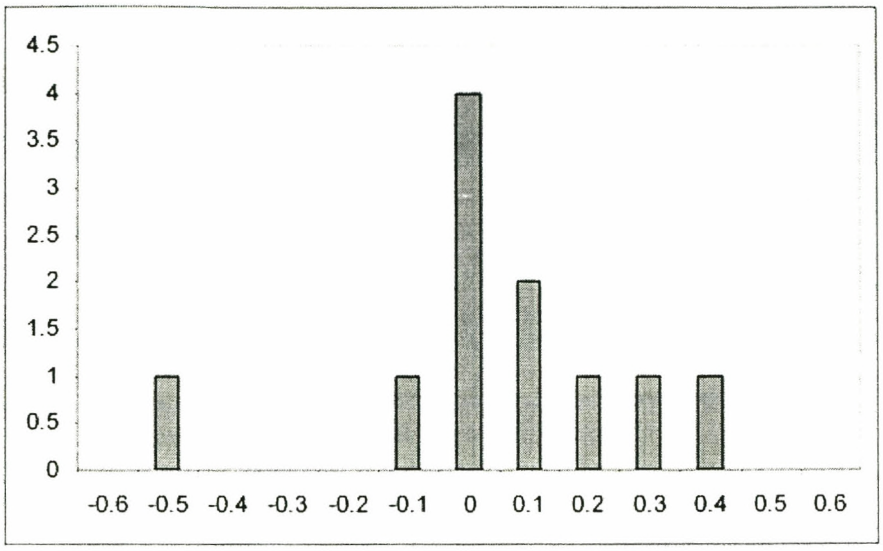


Figure H 10 Distribution of the residuals for Angle = $ax+b-be^{-cx}$.

APPENDIX I

RESULTS OBTAINED FROM MEASUREMENTS DONE ON DIFFERENT TYPES OF METAL FLAT BARS

Measurements were done to obtain the parameters for the equations for the different types of metal flat bars. The results are listed separately for each type of metal.

I.1 30MM X 100MM SHEET METAL

Measurements:

Target distance in millimetres	Actual distance in millimetres	Measured angle in degrees
0	0	0
1	0.99	0
2	2.41	1
4	4.01	2.5
7	7.02	6
10	9.95	9
15	15.09	16.5
20	20.45	23.5
25	24.99	29.5
30	29.95	36.5
35	34.98	42.5
40	40.2	49.75
45	45.72	57.5
50	50.31	62

Bend equation parameters according to Regression Analyser:

Parameter	Value
A	-0.005409
B	1.73287
C	-11.222
D	0.10373

Bend equation parameters according to Statistica:

Parameter	Value
A	-0.00201
B	1.447565
C	-5.24269
D	0.25626

Results:

Target angle in degrees	Measured angle In degrees
14.92	15.5
14.93	15.3
14.93	15.5
14.93	15
15.15	15.5
14.9	14.5
15.47	15

Calculated residuals for Regression Analyser parameters:

Observation	Predicted angle	Residual
0	0	0
1	0.61502	0.61502
2	1.662597	0.662597
3	3.04308	0.54308
4	6.093999	0.093999
5	9.482426	0.482426
6	16.04105	0.458949
7	23.29841	0.201592
8	29.54451	0.044514
9	36.32773	0.17227
10	43.07336	0.573357
11	49.87163	0.121628
12	56.7961	0.703899
13	62.32875	0.328754

Statistical properties of equation according to Statistica:

Model: $y = ax^2 + bx + c - ce^{-dx}$

Dep. var: ANGLE Loss: (OBS-PRED)**2

Final loss: 1.464345138 R=.99989 Variance explained: 99.977%

	A	B	C	D
Estimate	-0.00201	1.447565	-5.24269	0.25626

Means and Standard Deviations

	Mean	**St. dev.**	**Minimum**	**Maximum**
Displacement	20.43357	17.44042	0	50.31
Angle	24.01786	22.37239	0	62

RESIDUAL OUTPUT

Observation	**Residual**	**Predicted Angle**
0	0	0
1	-0.25637	0.256366
2	-0.06134	1.061336
3	0.094159	2.405841
4	0.312465	5.687535
5	-0.37073	9.37073
6	0.247628	16.25237
7	-0.04594	23.54594
8	-0.18351	29.68351
9	0.191357	36.30864
10	-0.43069	42.93069
11	0.053499	49.6965
12	0.767799	56.7322
13	-0.48921	62.48921

I.2 25MM X 80MM SHEET METAL

Measurements:

Target distance in millimetres	Actual distance in millimetres	Measured angle in degrees
0	0	0
1	1.09	0
2	2.01	0
4	4.01	2.66
7	7.04	6.33
10	10.5	11
15	15.14	17
20	20.36	24
25	24.91	30
30	29.98	37.5
35	35.06	43.5
40	39.9	50.5
45	44.98	55
50	50	61.66

Bend equation parameters according to Regression Analyser:

Parameter	Value
A	-0.0026286
B	1.44476
C	-3.9868
D	0.49461

Bend equation parameters according to Statistica:

Parameter	Value
A	-0.00289
B	1.462125
C	-4.19475
D	0.455232

Results:

Target angle in degrees	Measured angle in degrees
14.89	15.25
15.5	15.5
15.35	15.33
15.47	15.33
15.37	15
15.06	15
16.65	16
15.15	15
15.35	15
30.27	30.33
30.25	30
30.17	30.33
30.32	29.75
30.08	29.85
30.05	30
30.21	30
30.37	30.25
30.27	30
30.25	30
45.23	45
45.19	45.5
45.15	46
45.26	45.5

Calculated residuals for Regression Analyser parameters:

Observation	Predicted angle	Residual
0	0	0
1	-0.0898	0.0898
2	0.381791	0.381791
3	2.313013	0.346987
4	6.176604	0.153396
5	10.91552	0.084484
6	17.28657	0.286571
7	24.33905	0.33905
8	30.37112	0.371122
9	36.96452	0.53548
10	43.4354	0.064599
11	49.47437	1.025633
12	55.68032	0.68032
13	61.6797	0.0197

Statistical properties of equation according to Statistica:

Model: $y = ax^2 + bx + c - ce^{-dx}$

Dep. var: ANGLE Loss: (OBS-PRED)**2

Final loss: 2.422414715 R=.99981 Variance explained: 99.962%

	A	B	C	D
Estimate	-0.00289	1.462125	-4.19475	0.455232

RESIDUAL OUTPUT

Observation	Residual	Predicted Angle
0	0	0
1	0.050551	-0.05055
2	-0.41246	0.412464
3	0.362238	2.297762
4	0.204705	6.125295
5	0.126381	10.87362
6	-0.28251	17.28251
7	-0.37449	24.37449
8	-0.43052	30.43052
9	0.462184	37.03782
10	-0.00892	43.50892
11	0.964691	49.53531
12	-0.71464	55.71465
13	-0.01422	61.67422

I.3 15MM X 80MM SHEET METAL

Measurements:

Target distance in millimetres	Actual distance In millimetres	Measured angle in degrees
0	0	0
1	1.13	0
2	2.37	1.5
4	3.98	3
7	7.04	6.25
10	10.04	10
15	15.17	17
20	20.1	23
25	24.96	30
30	30.06	37
35	35.48	44
40	40.4	50

Bend equation parameters according to Regression Analyser:

Parameter	Value
A	0.0006249
B	1.29504
C	-2.8229957
D	0.454847

Bend equation parameters according to Statistica:

Parameter	Value
A	-0.00219
B	1.44966
C	-4.82911
D	0.230089

Results:

Target angle in degrees	Measured angle in degrees
14.87	14.5
14.80	15.0
14.90	15.0
29.93	30.0
29.77	30.0
29.92	30.0
59.94	60.0

Calculated residuals for Regression Analyser parameters:

Observation	Predicted angle	Residual
0	0	0
1	0.329665	0.329665
2	1.210367	0.289633
3	2.803023	0.196977
4	6.439884	0.189884
5	10.27154	0.271535
6	16.96941	0.030586
7	23.46008	0.460076
8	29.89055	0.10945
9	36.67057	0.329428
10	43.91167	0.088333
11	50.51656	0.516557

Statistical properties of equation according to Statistica:

Model: $y = ax^2 + bx + c - ce^{-dx}$

Dep. var: ANGLE Loss: (OBS-PRED)**2

Final loss: .679778367 R=.99990 Variance explained: 99.981%

	A	B	C	D
Estimate	-0.00219	1.44966	-4.82911	0.230089

RESIDUAL OUTPUT

Observation	**Residual**	**Predicted Angle**
0	0	0
1	-0.529694319	0.529694319
2	0.106487058	1.393512964
3	0.161521941	2.838478088
4	0.026346231	6.223653793
5	0.016266009	9.983734131
6	0.195170224	16.80483055
7	-0.470495582	23.47049522
8	-0.003764193	30.00376511
9	0.22896336	36.77103806
10	0.154158175	43.84584045
11	-0.158600554	50.15859985

I.4 12.5MM X 100MM SHEET METAL

Measurements:

Target distance In millimetres	**Actual distance in millimetres**	**Measured angle in degrees**
0	0.00	0
1	0.96	0
2	2.27	0.5
4	4.38	2.5
4	4.14	2.5
7	6.99	5.0
7	7.04	6.0
10	10.04	9.0
15	15.15	15.0
15	15.08	15.0
20	20.38	22.5
25	25.00	28.5

Bend equation parameters according to Regression Analyser:

Parameter	Value
A	0.008906
B	1.004105
C	-2.03508
D	0.774779

Bend equation parameters according to Statistica:

Parameter	Value
A	0.008906
B	1.004105
C	-2.03508
D	0.774779

Results:

Target angle in degrees	Measured angle in degrees
30.13	30
29.87	29.5
29.86	29.5
44.73	42
45.02	42
44.87	42

Calculated residuals for Regression Analyser parameters:

Observation	Predicted angle	Residual
0	0	0
1	-0.095637346	0.095637346
2	0.640693654	0.140693654
3	2.602114152	0.102114152
4	2.35688749	0.14311251
5	5.427810376	0.427810376
6	5.483919363	0.516080637
7	8.944724975	0.055275025
8	15.22125439	0.221254386
9	15.13212195	0.132121955
10	22.12763741	0.372362591
11	28.63379501	0.133795008

Statistical properties of equation according to Statistica:

Model: $y = ax^2 + bx + c - ce^{-dx}$

Dep. var: ANGLE Loss: (OBS-PRED)**2

Final loss: .735230542 R=.99962 Variance explained: 99.925%

	A	B	C	D
Estimate	0.008906	1.004105	-2.03508	0.774779

RESIDUAL OUTPUT

Observation	Residual	Predicted Angle
0	0	0
1	0.095638052	-0.095638052
2	-0.140693739	0.640693724
3	-0.102119654	2.602119684
4	0.143107876	2.356892109
5	-0.427828997	5.427828789
6	0.516061723	5.483938217
7	0.055232596	8.944766998
8	-0.221356824	15.22135639
9	-0.132223397	15.13222313
10	0.372172713	22.12782669
11	-0.134084016	28.6340847

APPENDIX J

DETAILED DESCRIPTION OF THE REGRESSION ANALYSER

The regression analyser is discussed in this section. Only the algorithm and the accompanying programming techniques are discussed.

The non-linear equation to be solved from section 5.4 is

$$y = ax^2 + bx + c - ce^{-dx} \tag{J.1}$$

The regression method used is one of the simplest algorithms for computing the least-square estimates for non-linear regression and it is called the Gauss-Newton algorithm. The algorithm will be discussed based upon the description by (Gallant, 1987:8).

The sum of the square surfaces can be approximated as follows:

$$SSE_T = \left\| \mathbf{y} - \mathbf{f}(\boldsymbol{\theta}_\mathrm{T}) - \mathbf{F}(\boldsymbol{\theta}_\mathrm{T})(\boldsymbol{\theta} - \boldsymbol{\theta}_\mathrm{T}) \right\|^2 \tag{J.2}$$

This equation was derived from Taylor's theorem and is described in (Gallant, 1987:8). The different parameters are denoted as follows:

- $\boldsymbol{\theta} = \begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix}$ with a, b, c and d the different coefficients in the regression equation 5.2.
- $\boldsymbol{\theta}_\mathrm{T}$ is the current measurements of the parameters.
- $\mathbf{f}(\boldsymbol{\theta}_\mathrm{T})$ is the result of the regression equation for the input value x and the parameters $\boldsymbol{\theta}_\mathrm{T}$.

- $\mathbf{F}(\boldsymbol{\theta}_T)$ is defined as $\mathbf{F}(\boldsymbol{\theta}_T) = \frac{d}{d\boldsymbol{\theta}_T'}\mathbf{f}(\boldsymbol{\theta}_T)$ as shown in (Gallant, 1987:8).

New parameter values, $\boldsymbol{\theta}$, for a given iteration are written as follows:

$$\boldsymbol{\theta} = \boldsymbol{\theta}_T + \lambda(\boldsymbol{\theta}_M - \boldsymbol{\theta}_T)\, with\; 0 < \lambda < \lambda_{max} < 1 \qquad (J.3)$$

$\boldsymbol{\theta}_M$ is the value for $\boldsymbol{\theta}_M$ at the minimum value of $SSE_T(\boldsymbol{\theta})$.

λ is the step taken from one iteration to the next.

Equation (J.2) must satisfy the criteria $SSE(\boldsymbol{\theta})<\mathrm{SSE}(\boldsymbol{\theta}_T)$ to ensure convergence.

The algorithm can now be defined as follows:

1) Find:

$$\mathbf{D}_i = [\mathbf{F}'(\boldsymbol{\theta}_i)\mathbf{F}(\boldsymbol{\theta}_i)]^{-1}\mathbf{F}'(\boldsymbol{\theta}_i)[\mathbf{y} - \mathbf{f}(\boldsymbol{\theta}_i)] \qquad (J.4)$$

with the parameter values defined as follows:

$\mathbf{D}_i$ – an estimate of $\boldsymbol{\theta}_M - \boldsymbol{\theta}_T$ after the *i*'th iteration.

2) Calculate $SSE(\boldsymbol{\theta}_i + \lambda_i\mathbf{D}_i) < SSE(\boldsymbol{\theta}_i)$ with $0 < \lambda < 1$.

3) Set $\boldsymbol{\theta}_{i+1} = \boldsymbol{\theta}_i + \lambda_i\mathbf{D}_i$ for the next iteration and repeat from 1.

The algorithm stops when both the following conditions are met:

1) $\|\boldsymbol{\theta}_i - \boldsymbol{\theta}_{i+1}\| < \varepsilon(\|\boldsymbol{\theta}_i\| + \tau)$

2) $|SSE(\boldsymbol{\theta}_i) - SSE(\boldsymbol{\theta}_{i+1})| < \varepsilon(SSE(\boldsymbol{\theta}_i) + \tau)$

The values of ε and τ were set at 0.00001 and 0.001 respectively.

The program uses linked lists to ensure that any number of data points can be used to calculate the parameters. The matrix multiplication and inversion are done by algorithms written for this problem and are not general solutions.

J.1 ALGORITHM USED

The algorithm was written in Delphi. The source code is listed below. Only the source code for regression analysis is shown.

```
procedure TfrmRegression.Runregression1Click(Sender: TObject);

type
    VMatrix=Array[1..4,1..4] of extended;
    VMatrixs=Array[1..4] of extended;

var a,b,c,d,L:extended;
    F:TList;
    Ft:TList;
    t:integer;
    Fdata:PDatapoints;
    x:extended;
    M,Minv:VMatrix;
    Msec,Mtrd:VMatrixs;
    Fx:TList;
    y:PAngle;
    OldMacDonaldHaveAFarm:Boolean;
    count:integer;

procedure MultiplyMatrixInv(F,Ft:TList;Var M:VMatrix);

var l : integer;
    t,u : integer;
    element : extended;
    FData : PDatapoints;
    Madj:VMatrix;
    detA:extended;
    Minv:VMatrix;
{       tlist}
begin
  l:=F.Count;
  for t:=1 to 4 do
  for u:=1 to 4 do
  begin
   M[t,u]:=0;
  end;
  Madj:=M;
  Minv:=M;
  for t := 0 to l-1 do
  begin
    FData:=F.Items[t];
    element:=FData^.df01*FData^.df01;
    M[1,1]:=M[1,1]+element;
    element:=FData^.df02*FData^.df02;
    M[2,2]:=M[2,2]+element;
    element:=FData^.df03*FData^.df03;
    M[3,3]:=M[3,3]+element;
    element:=FData^.df04*FData^.df04;
    M[4,4]:=M[4,4]+element;
    element:=FData^.df02*FData^.df01;
```

```
      M[2,1]:=M[2,1]+element;
      element:=FData^.df03*FData^.df01;
      M[3,1]:=M[3,1]+element;
      element:=FData^.df04*FData^.df01;
      M[4,1]:=M[4,1]+element;
      element:=FData^.df03*FData^.df02;
      M[3,2]:=M[3,2]+element;
      element:=FData^.df04*FData^.df02;
      M[4,2]:=M[4,2]+element;
      element:=FData^.df04*FData^.df03;
      M[4,3]:=M[4,3]+element;
    end;
    M[1,2]:=M[2,1];
    M[1,3]:=M[3,1];
    M[1,4]:=M[4,1];
    M[2,3]:=M[3,2];
    M[2,4]:=M[4,2];
    M[3,4]:=M[4,3];

  Madj[1,1]:=M[2,2]*M[3,3]*M[4,4]+M[3,2]*M[4,3]*M[4,2]+M[2,4]*M[3,2]*M[
  4,3];
    Madj[1,1]:=Madj[1,1]-M[4,2]*M[3,3]*M[4,2]-M[2,2]*M[4,3]*M[4,3]-
  M[3,2]*M[3,2]*M[4,4];

  Madj[2,1]:=M[2,1]*M[3,3]*M[4,4]+M[3,1]*M[4,3]*M[4,2]+M[1,4]*M[3,2]*M[
  4,3];
    Madj[2,1]:=Madj[2,1]-M[1,4]*M[3,3]*M[4,2]-M[1,2]*M[3,4]*M[4,3]-
  M[1,3]*M[3,2]*M[4,4];
    Madj[2,1]:=-Madj[2,1];

  Madj[3,1]:=M[1,2]*M[3,2]*M[4,4]+M[1,3]*M[2,4]*M[4,2]+M[1,4]*M[2,2]*M[
  4,3];
    Madj[3,1]:=Madj[3,1]-M[1,4]*M[2,3]*M[4,2]-M[1,2]*M[2,4]*M[4,3]-
  M[1,3]*M[2,2]*M[4,4];

  Madj[4,1]:=M[1,2]*M[2,3]*M[3,4]+M[1,3]*M[2,4]*M[3,2]+M[1,4]*M[2,2]*M[
  3,3];
    Madj[4,1]:=Madj[4,1]-M[1,4]*M[2,3]*M[3,2]-M[1,2]*M[2,4]*M[3,3]-
  M[1,3]*M[2,2]*M[3,4];
    Madj[4,1]:=-Madj[4,1];

  Madj[2,2]:=M[1,1]*M[3,3]*M[4,4]+M[1,3]*M[3,4]*M[4,1]+M[1,4]*M[3,1]*M[
  4,3];
    Madj[2,2]:=Madj[2,2]-M[1,4]*M[3,3]*M[4,1]-M[1,1]*M[3,4]*M[4,3]-
  M[1,3]*M[3,1]*M[4,4];

  Madj[3,2]:=M[1,1]*M[2,3]*M[4,4]+M[1,3]*M[2,4]*M[4,1]+M[1,4]*M[2,1]*M[
  4,3];
    Madj[3,2]:=Madj[3,2]-M[1,4]*M[2,3]*M[4,1]-M[1,1]*M[2,4]*M[4,3]-
  M[1,3]*M[2,1]*M[4,4];
    Madj[3,2]:=-Madj[3,2];

  Madj[4,2]:=M[1,1]*M[2,3]*M[3,4]+M[1,3]*M[2,4]*M[3,1]+M[1,4]*M[2,1]*M[
  3,3];
    Madj[4,2]:=Madj[4,2]-M[1,4]*M[2,3]*M[3,1]-M[1,1]*M[2,4]*M[3,3]-
  M[1,3]*M[2,1]*M[3,4];

  Madj[3,3]:=M[1,1]*M[2,2]*M[4,4]+M[1,2]*M[4,2]*M[4,1]+M[1,4]*M[2,1]*M[
  4,2];
  Madj[3,3]:=Madj[3,3]-M[1,4]*M[2,2]*M[4,1]-M[1,1]*M[2,4]*M[4,2]-
M[1,2]*M[2,1]*M[4,4];

Madj[4,3]:=M[1,1]*M[2,2]*M[3,4]+M[1,2]*M[2,4]*M[3,1]+M[1,4]*M[2,1]*M[
3,2];
  Madj[4,3]:=Madj[4,3]-M[1,4]*M[2,2]*M[3,1]-M[1,1]*M[2,4]*M[3,2]-
M[1,2]*M[2,1]*M[3,4];
  Madj[4,3]:=-Madj[4,3];

Madj[4,4]:=M[1,1]*M[2,2]*M[3,3]+M[1,2]*M[2,3]*M[3,1]+M[1,3]*M[2,1]*M[
3,2];
 Madj[4,4]:=Madj[4,4]-M[1,3]*M[2,2]*M[3,1]-M[1,1]*M[2,3]*M[3,2]-
M[1,2]*M[2,1]*M[,3];
  Madj[1,2]:=Madj[2,1];
  Madj[1,3]:=Madj[3,1];
  Madj[1,4]:=Madj[4,1];
  Madj[2,3]:=Madj[3,2];
  Madj[2,4]:=Madj[4,2];
  Madj[3,4]:=Madj[4,3];
  detA:=0;
  For t:=1 to 4 do
    Begin
      detA:=detA+M[t,1]*Madj[t,1];
    end;
  For t:=1 to 4 do
  for u:= 1 to 4 do
    Begin
      Minv[t,u]:=Madj[t,u]/detA;
    end;
  M:=Minv;
end;

procedure MultiplyMatrix(Fx,Ft:TList;Var M:VMatrixs);

var y:PAngle;
    yg,dy:extended;
    l:integer;
    t,u:integer;
    FData : PDatapoints;
    element:Extended;

begin
  l:=F.Count;
  for t:=1 to 4 do
  begin
   M[t]:=0;
  end;
  for t := 0 to l-1 do
  begin
    y:=Fx.Items[t];
    yg:=StrToFloat(sgrDataPoints.Cells[2,t+1]);
    dy:=yg-y^;
    FData:=Ft.Items[t];
    element:=dy*FData^.dfo1;
    M[1]:=M[1]+element;
    element:=dy*FData^.df02;
    M[2]:=M[2]+element;
    element:=dy*FData^.df03;
    M[3]:=M[3]+element;
```

```
      element:=dy*FData^.df04;
      M[4]:=M[4]+element;
    end;
  end;

procedure MultiFour(Minv:VMatrix;Var Msec,Mtrd:VMAtrixs);

var t,u:integer;

begin
   For t:=1 to 4 do
      Mtrd[t]:=0;
   For t:=1 to 4 do
   for u:=1 to 4 do
     begin
        Mtrd[t]:=Mtrd[t]+Minv[t,u]*Msec[u];
     end;
end;

procedure SSE(var a,b,c,d,L:extended;Mtrd:Vmatrixs;va
OldMacDonaldHaveAFarm:Boolean);

const

     eps=0.00001;
     tau=0.001;

var
     y,youd:extended;
     t:integer;
     x,angle:extended;
     an,bn,cn,dn:extended;
     som,somoud:extended;
     ta,tb,tc,td,tsom:extended;

begin
  an:=a+L*Mtrd[1];
  bn:=b+L*Mtrd[2];
  cn:=c+L*Mtrd[3];
  dn:=d+L*Mtrd[4];
  som:=0;
  somoud:=0;
  For t:=1 to sgrDataPoints.Rowcount-1 do
  begin
      x:=StrToFloat(sgrDataPoints.Cells[1,t]);
      angle:=StrToFloat(sgrDataPoints.Cells[2,t]);
      y:=angle-(an*x*x+bn*x+cn-cn*exp(-dn*x));
      youd:=angle-(a*x*x+b*x+c-c*exp(-d*x));
      y:=y*y;
      youd:=youd*youd;
      som:=som+y;
      somoud:=somoud+youd;
  end;
  ta:=abs(a-an)-eps*(abs(a)+tau);
  tb:=abs(b-bn)-eps*(abs(b)+tau);
  tc:=abs(c-cn)-eps*(abs(c)+tau);
  td:=abs(d-dn)-eps*(abs(d)+tau);
  tsom:=abs(som-somoud)-abs(som+tau);
  If (ta<=0) and (tb<=0) and (tc<=0) and (td<=0) and (tsom<=0) then
  begin
    OldMacDonaldHaveAFarm:=true
  end;
  if som<somoud then
    begin
      a:=an;
      b:=bn;
      c:=cn;
      d:=dn;
    end
  else
    begin
      L:=L/5;
    end;
  lblA.caption:='A:  '+FloatToStr(a);
  lblB.caption:='B:  '+FloatToStr(b);
  lblC.caption:='C:  '+FloatToStr(c);
  lblD.caption:='D:  '+FloatToStr(d);
end;

begin
  a:=strtofloat(edtA.Text);
  b:=strtofloat(edtB.Text);
  c:=strtofloat(edtC.Text);
  d:=strtofloat(edtD.Text);
  L:=strtofloat(edtStep.Text);
  OldMacDonaldHaveAFarm:=False;
  count:=0;
  Repeat
  F:=TList.create;
  Ft:=TList.create;
  Fx:=TList.create;
  For t:=1 to sgrDataPoints.Rowcount-1 do
  begin
     new(Fdata);
     new(y);
     x:=StrToFloat(sgrDataPoints.Cells[1,t]);
     FData^.df01:=x*x;
     FData^.df02:=x;
     FData^.df03:=1-exp(-d*x);
     FData^.df04:=c*x*exp(-d*x);
     y^:=a*x*x+b*x+c-c*exp(-d*x);
     F.add(Fdata);
     Ft.add(Fdata);
     Fx.add(y);
  end;
  MultiplyMatrixInv(F,Ft,Minv);
  MultiplyMatrix(Fx,Ft,Msec);
  MultiFour(Minv,Msec,Mtrd);
  SSE(a,b,c,d,L,Mtrd,OldMacDonaldHaveAFarm);
  FOR t:= 0 to F.Count-1 do
    begin
     Dispose(F.Items[t]);
    end;
  F.Free;
  FOR t:= 0 to Ft.Count-1 do
    begin
     Dispose(Ft.Items[t]);
    end;
  Ft.Free;
  FOR t:= 0 to Fx.Count-1 do
    begin
     Dispose(Fx.Items[t]);
    end;
  Fx.Free;
  until OldMacDonaldHaveAFarm;

end;


procedure TfrmRegression.Statisticalresults1Click(Sender: TObject);
var
  TextValue:string;

begin
    TextValue:=lblA.Caption;
    delete(TextValue,1,2);
    edtA.text:=TextValue;
    TextValue:=lblB.Caption;
    delete(TextValue,1,2);
    edtB.text:=TextValue;
    TextValue:=lblC.Caption;
    delete(TextValue,1,2);
    edtC.text:=TextValue;
    TextValue:=lblD.Caption;
    delete(TextValue,1,2);
    edtD.text:=TextValue;
end;

end.
```