UNIVERSITY OF BRADFORD

Department of Cybernetics Internet and Virtual Systems

BSc (Hons) Cybernetics and Virtual Worlds

Final Year Project Report

by

Nicholas Hardman

UB Number: 03013084

Project ABLE - *Development of a prototype robotic brick layer for alongside other construction robotics*

Supervised by Dr J Readle

03-05-2006

Declaration:

I declare this work is my own and is original. All references have been documented in the references section and all code is included in the appendix.

Abstract:

Bricklaying is a highly repetitive skilled task ideal for automation. A number of masonry laying projects have attempted to make a brick laying robot, this dissertation will review these projects and take the reader through the process of developing a small scale prototype automatic brick layer. The product developed here has a wooden base, can drive along two pieces of model train track laying foam bricks in a wall and is programmed using Visual Basic.Net. Proposals are made for the future development of this prototype as a full scale model and how this machine could possibly be integrated with a number of other construction robots. Results include a successful interface between hardware and software using USB, a working prototype wall builder and a mechanically functional crane.

List of Contents

List of Figures

Introduction:

This dissertation will demonstrate a process of conceptualisation and implementation of a prototype automatic brick layer. The design process is based upon the building of a small scale product aimed to prove the concept is possible. The dissertation presents a piece of developmental research working with a restriction of 400 hours, and a limited budget of £300. A full scale operational prototype would require a budget somewhere in the millions and therefore far beyond the scope of this project.

Automation has always been of interest, as effective use of robotics can solve many repetitive tasks in a variety of contexts. This idea –"Automatic Brick Laying Engine (ABLE)" has grown from a process of experimentation with a robotic arm, blocks of wood, Lego and foam bricks. The foam bricks have little weight which makes mathematics such as calculation of the moment of inertia and forces acting on the robot arm when at various positions irrelevant. Such problems would need to be considered in the full scale version.

The development of this robot does not encompass some of the problems (discussed later) encountered when working with mortar which still need to be addressed. During ongoing research, an existing solution (See Linklater's Mortar Machine in Related Work section) was found to the mortar issue and as a result, the prototype that will be made for this project will not involve any adhesive.

The design of ABLE is evaluated and modifications are proposed to improve the "commercial viability" feasibility and practicability of the project.

The construction industry shows a steady growth which leads to a rising demand for cost saving technologies. In 2004, the industry showed a growth of 9.7% (DTI, 2004) with a total output value of £102,363 million (Hughes, 2004).

In 2003, some 2,940 million (DTI, 2004) bricks were delivered in the UK and laid by a large number of skilled labourers at a very high cost. Bricklaying is a highly repetitive task that requires high levels of precision for long periods of time. The cost of one square meter of brick wall, including labour costs, can vary from £100 to £500 (Milton Building Services, 2005) dependant on the number of bricks to be laid in the whole contract and the pattern in which the bricks are to be laid. A human can take between one and two hours (Milton Building Services, 2005) to lay one square metre of bricks which gives us some data to compare the effectiveness of the machine to be developed.

The brick laying process can be made more efficient by automation. With a machine to complete the task, construction can run 24 hours a day without breaks and without such a large labour cost. If this was made a commercial application tomorrow, then there would be a number of humans working with the machine – i.e. people would apply cement and the robot would accurately position the bricks. It would be like a hammer – just a tool for making a job easier. Ultimately, this robot would be one of many – a fleet of construction robots (discussed later). The reader is therefore asked to visualise a number of human workers working alongside a brick laying robot, a mortar laying robot and an automatic crane.

As the full commercial version of the robot would be required to build a wall higher than its own reach, the end product must be mountable onto scaffolding or a gantry. Therefore the model developed here runs along a track which could easily be mounted onto a supporting structure.

However the focus of this project is making a robot complete one of the many tasks required to construct a building – laying bricks. My goal is to prove that such a cost saving device is possible by making a robot that can position bricks in the correct locations to build a wall.

Aims and Objectives

The ultimate aim of the robot in development here is to construct a straight line wall multiple bricks long and multiple bricks high.

Objectives include:

- Build a robot to run parallel with a wall which can move forwards and backwards pausing at every interval where a brick should be laid.
- Construct a mechanism that can pick up and place down bricks.
- Mount the brick laying mechanism onto the robot allowing space for a reserve of bricks.
- Program the machine to monitor the number of bricks and request reloading when the machine is empty.
- Program the machine to pick up the next brick in the reserve.
- Program the machine to lay the brick onto the top layer of the wall.

During this research, the possibility of applying bricks from above the wall via a crane will be explored as this could prove to have significant advantages.

Review of related work

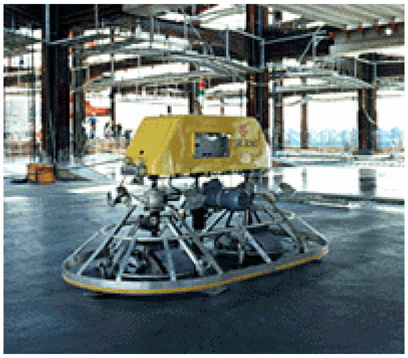
Research into construction robotics is ongoing and many are in development simultaneously across the world. This provides evidence of the significant value of this technology and highlights some of the major challenges faced by the automated construction industry.

A number of completed projects were discovered within the duration of this project; these are summarised below.

Takenaka Corporation

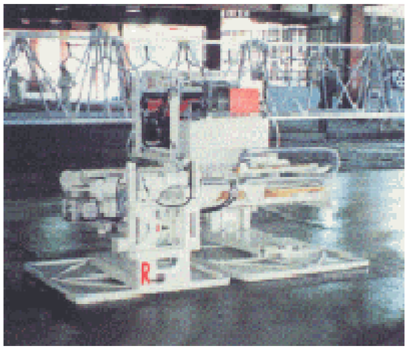
A number of machines have been made by the 'Takenaka Corporation' in Japan. Limited information on some of these machines can be found on the Construction Robotics Page on the Takenaka Website (Takenaka, 2005). Each of these machines have been highlighted because they share one thing with project ABLE. All of these machines have been designed to complete an individual task; collectively these add up to fully automated construction.

Figure 1 - Concrete Flooring Robot



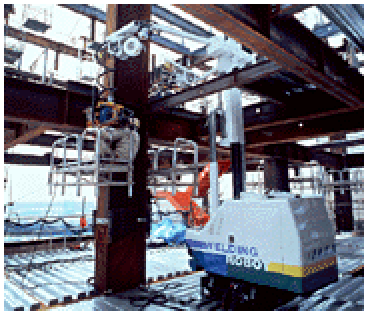
On each floor of a construction, this robot can apply the concrete required for a solid floor. This surface would take a while to dry, and so another machine was invented.

Figure 2 - Water Absorbing Robot



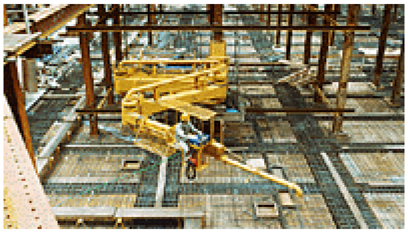
Once a concrete floor is laid, this robot can absorb most of the water making the surface traversable by other machines. A future development of project ABLE could then drive over this freshly laid concrete to lay bricks wherever required.

Figure 3 - Steel Welding Robot



This robot can automatically weld horizontal and vertical steel supports together. This system could be used with an automatic crane to construct a supporting framework for a large structure.

Figure 4 - Horizontal Concrete Distributor

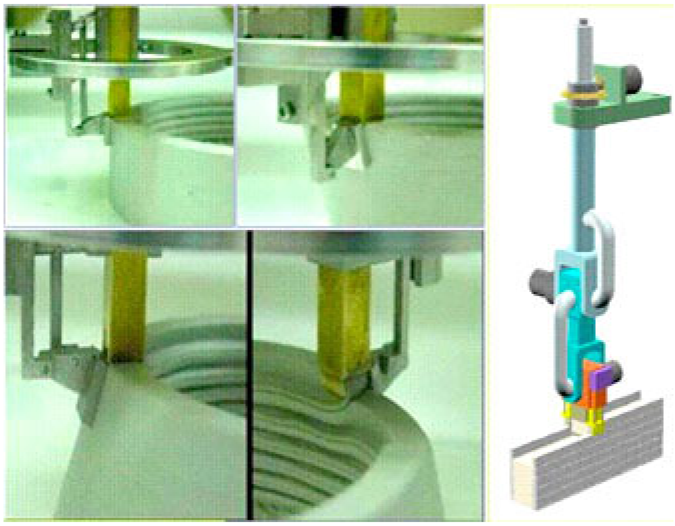


This robot automatically distributes blocks of concrete within steel structures. This is similar to project ABLE in that they both lay bricks. Project ABLE builds walls without supporting steel whereas this robot can only operate within a supporting framework.

Contour Crafting

An emerging technology in the construction industry is Contour Crafting (Khochnevisk, 2004). Contour Crafting involves a large robot dispensing cement-like materials to construct a three dimensional object. It is what I call the 'Printers of the Construction Industry'. More information on how this is achieved is in Appendix L.

Figure 5 - Photo of Contour Crafting



This technology has the potential to replace many existing construction techniques making new structural designs possible. Any developer working in construction robotics should be aware of the advantages Contour Crafting can bring.

Project Rocco 6450

Project Rocco (Cook, 1993) ran from 1992 to 1996 and involved creating two robotic manipulators for industrial and commercial construction. Prefabricated interlocking blocks are used without any conventional mortar, constructing buildings up to 10 metres high. Two machines were developed, one which has a 10m reach and a 500kg payload, and one with a 3.5m reach and 350kg payload. The project was financed as part of EU III 6450 Directive.
Project ABLE could be designed to use interlocking blocks like this machine, but the decision was made to lay normal bricks and rely on the use of some form of adhesive applied by a second machine or human worker.

Project Bronco

It is believed that during the 1980s, Dr Thomas Bock and Dr Micheal Dalacker conducted research into construction robotics
was carried out in Stuttgart, Germany (Gambao & Balaguer, 2002). Project Bronco (Brick Laying Robot for use on Construction Site) was developed along with another machines were developed. Near to the time of completion of this report, I was informed by Dr Thomas Bock "*You can get the research report from BRONCO at IRB, Fraunhofer, Gesellschaft , Nobelstrasse, Stuttgart Germany*". You can obtain a document (in German) on construction robotics including information on project Bronco from http://bibliothek.fzk.de/zb/berichte/FZKA6626.pdf

Star Lifter

A machine which is capable of lifting and orienting a 200kg load (usually another machine).

Figure 6 - Star Lifter

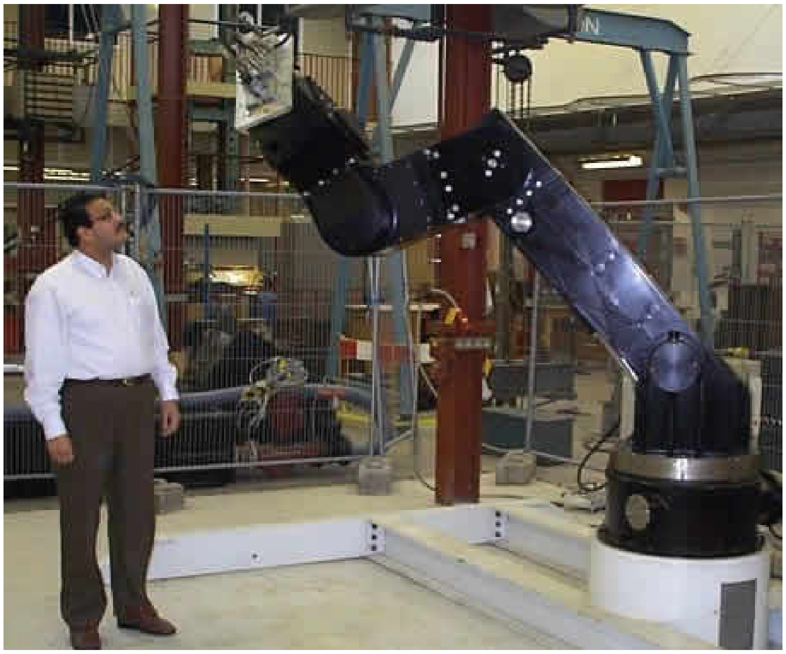
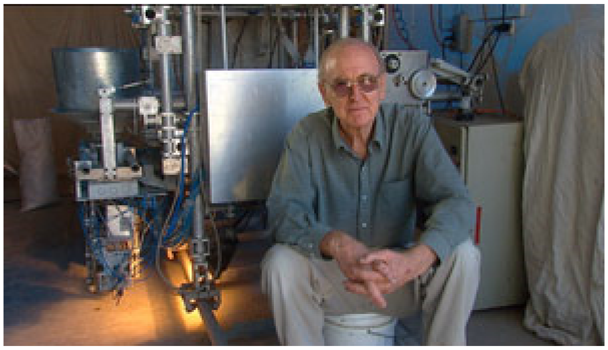


Figure 6 shows Star Lifter designed at the University of Lancaster (Intelligent Control Group, 2006). The machine is designed to carry other machines (potentially, project ABLE) over a work area.

Mortar Machine

Rod Linklater (2005) developed a machine to handle the application of mortar automatically.

Figure 7 - Mortar Machine



Project ABLE could be the second machine which is required to place the bricks to complete the wall building process.

Building Design Software

There are far too many software packages to conduct a comprehensive review as part of this report. Two examples have been provided here, one designed for steel and concrete structures (applicable to some of the Takenaka Robots) and one designed for brick structures (applicable to project ABLE).

Ram Structural Design

Figure 8 - Ram Structural Design Software

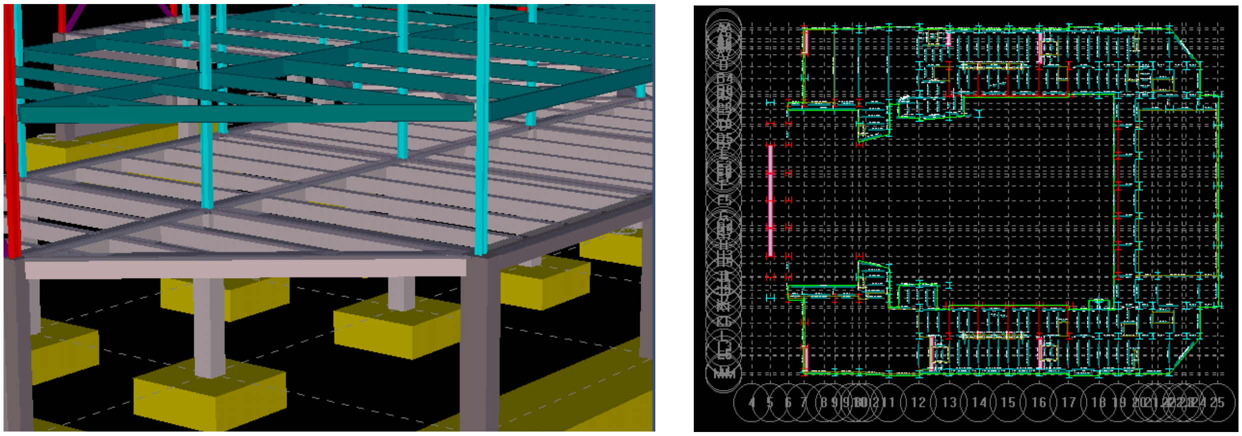
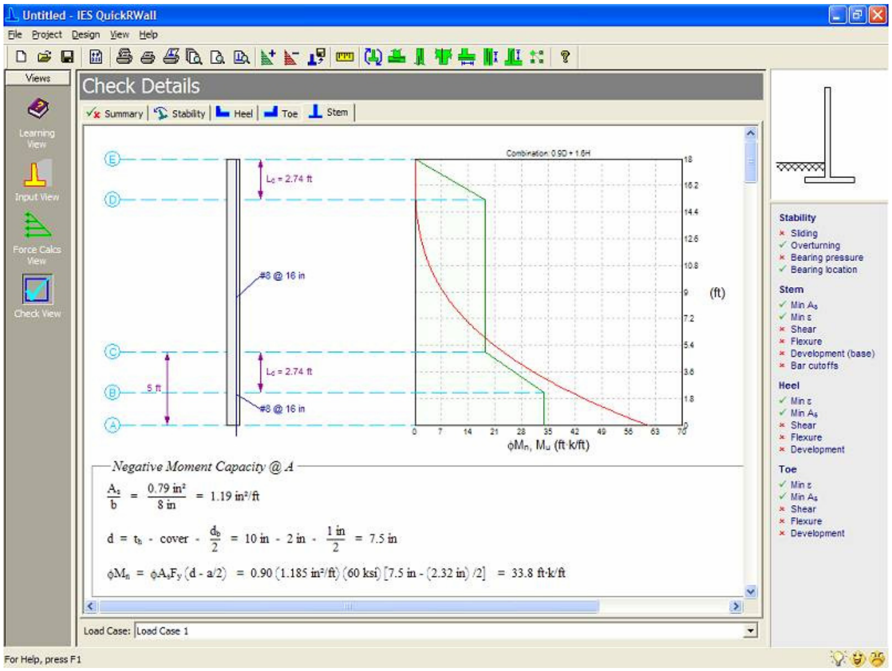


Figure 8 shows a two dimensional and a three dimensional plan of different structures designed in Ram Software. Ram (2006) provide a variety of structural design software, the screenshots here are "Ram Structural System"

Quick R Wall

Integrated Engineering Software (2006) produce another piece of simulation software, QuickRWall which works with bricks instead of steel and concrete. This could potentially be used with Project ABLE to calculate the stability of a wall before it is constructed.

Figure 9 - Quick R Wall Screenshot



Negative Moment Capacity @ A

$$\frac{A_s}{b} = \frac{0.79\ \text{in}^2}{8\ \text{in}} = 1.19\ \text{in}^2/\text{ft}$$

$$d = t_b - \text{cover} - \frac{d_b}{2} = 10\ \text{in} - 2\ \text{in} - \frac{1\ \text{in}}{2} = 7.5\ \text{in}$$

$$\phi M_n = \phi A_s F_y (d - a/2) = 0.90\,(1.185\ \text{in}^2/\text{ft})\,(60\ \text{ksi})\,[7.5\ \text{in} - (2.32\ \text{in})/2] = 33.8\ \text{ft·k/ft}$$

Load Case: Load Case 1

Stability
- ✗ Sliding
- ✓ Overturning
- ✗ Bearing pressure
- ✓ Bearing location

Stem
- ✓ Min A_s
- ✓ Min ε
- ✗ Shear
- ✗ Flexure
- ✗ Development (base)
- ✗ Bar cutoffs

Heel
- ✓ Min ε
- ✓ Min A_s
- ✗ Shear
- ✗ Flexure
- ✗ Development

Toe
- ✓ Min ε
- ✓ Min A_s
- ✗ Shear
- ✗ Flexure
- ✗ Development

Figure 9 shows a screenshot of QuickRWall showing how the wall will handle a load.

Research brick laying

Research has been undertaken into bricklaying by visiting a small construction site and discussing brick laying techniques. During this visit, it became apparent that construction sites are not always accessible and the ground surrounding a wall is not flat and even. For this reason, it was decided that the machine would have to run along a track or be suspended from above. Other issues were raised regarding the ‘perfect mix’ and ‘handling of mortar’ which led to the decision that ABLE would not use any adhesive.

Development Strategy / Project Plan

In this section the development of ABLE (Automatic Brick Laying Engine) will be discussed including preliminary designs, mechanical design, electrical design, interfacing with a computer and the programming which ultimately controls the whole system.

Initial Thoughts and Preliminary Designs

When approaching the problem of deploying a machine on a building site, two approaches were considered; deployment from above (hanging off a crane) or deployment from below (running along scaffolding). Each of the solutions has some health and safety risks, each with pros and cons. Any heavy machinery performing automated construction overhead is potentially dangerous.

A machine deployed from above has advantages of a far higher reach to build very high structures, and easier traversal over difficult terrain, but has the disadvantage of issues with uneven balance and swinging. A machine deployed from a scaffolding structure would not be affected by swinging but is much less self dependant as a working platform must be constructed.
A crane that lowers a platform which clamps to the structure could get the advantages of both approaches but introduces many new challenges.

Design of crane

Figure 10 - Design of Lego Crane System

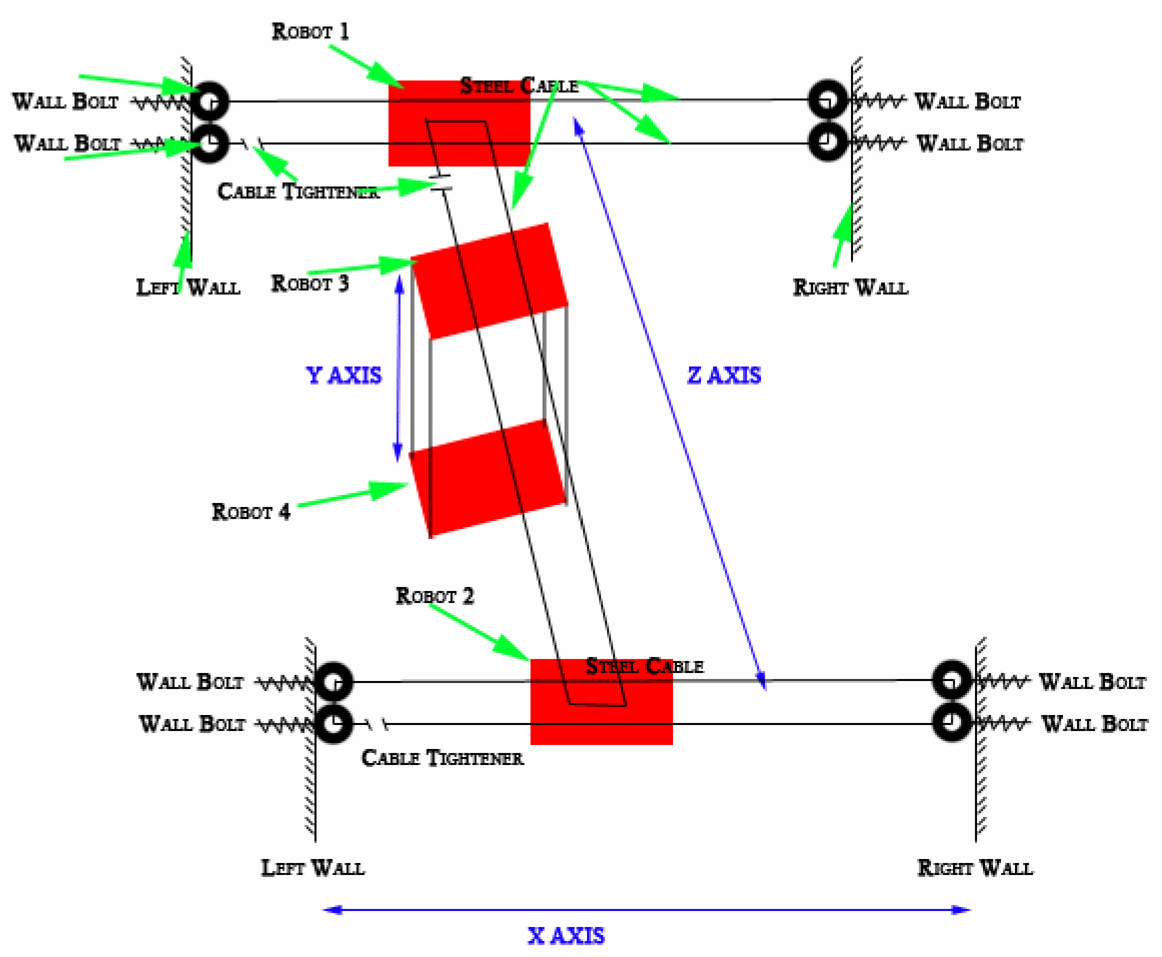


Figure 10 shows the plans for the crane based approach. Robots One and Two drive in synchronisation which provides freedom of movement across the length of the construction site. Robot Three drives along the central support carrying the working platform (Robot Four) across the width of the site and lowers Robot Four to any desired height.

Robots One, Two and Three each move along a single plane. The net effect of this is to carry the working platform (Robot Four) to any desired XYZ position across the construction site.

After the decision was made to use a track based system, a car was designed which would allow the robotic arm to be mounted onto the car allowing space for a stockpile of bricks.

Design of Car

Figure 11 - Design of Wooden Car

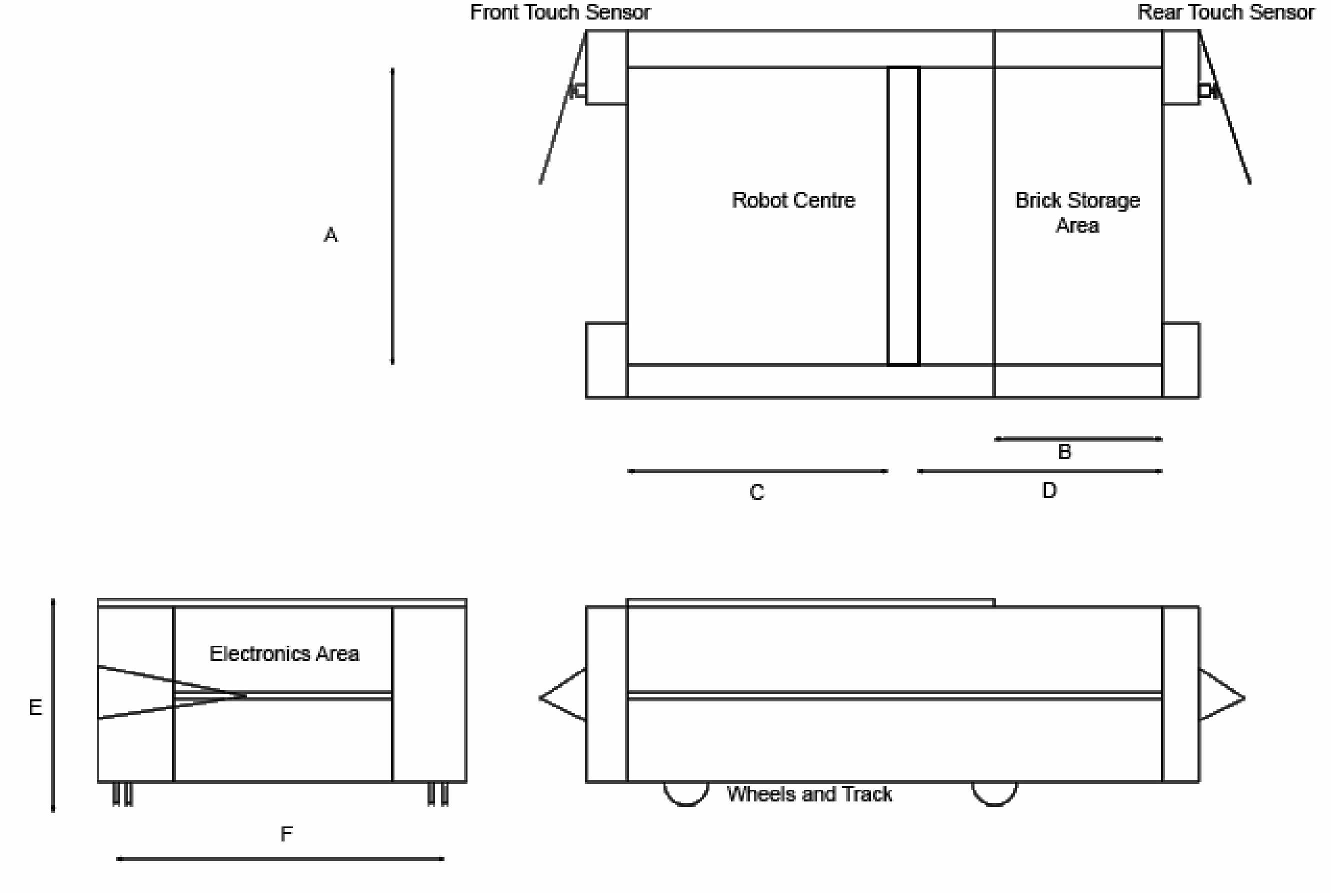


Figure 11 shows the plans for the chosen, track based approach. The measurements on the drawing are as follows

A=Internal Width = 186mm
B=Brick Storage Area Length = 111mm
C=Electronics Storage Area Length = 175mm
D=Brick Storage Area Length Inc. Under Robot = 145mm
E=Height of Base = 118mm
F=Length of Axel = 222mm

The decision was made to have the robot run along a track rather than be fully mobile and drive in any direction. This significantly reduced the number of challenges to prove the concept of an automated construction device without putting the robotic arm at risk of falling off a crane. Forcing the robot to move along a single straight line keeps it perfectly aligned with the wall. It also makes navigating the robot much more simple.

The next stage in the development of ABLE is to work out how each of the different system components are to be connected.

Systems model of ABLE

Figure 12 - Representation of Systems Model

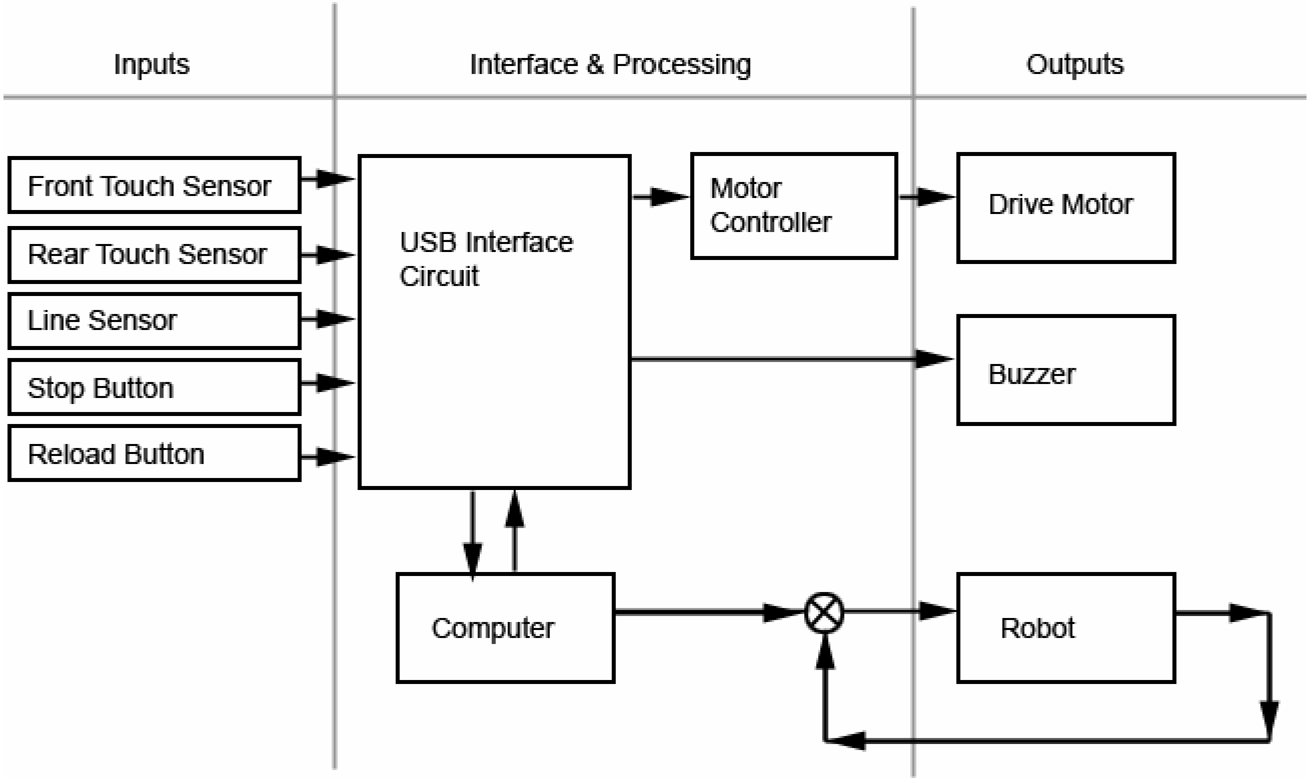
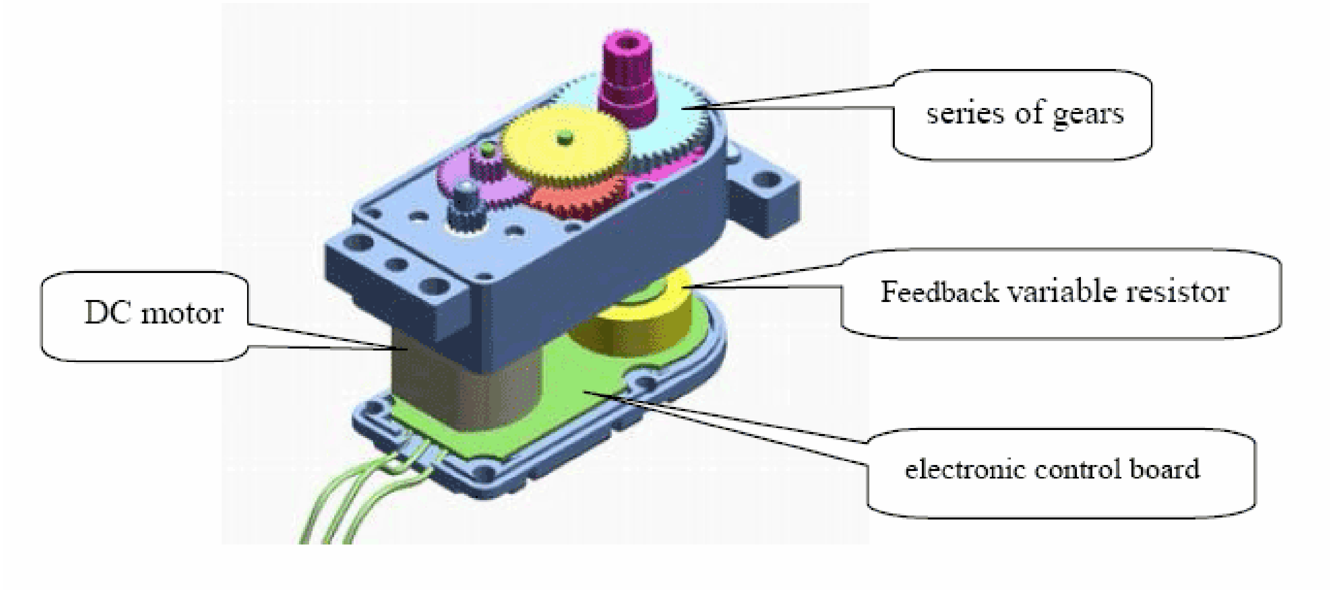


Figure 12 shows the system model of the prototype. The systems model of a real sized functioning machine would have feedback and much more position control. ABLE is an open loop system with some feedback on the control circuits within the arm.

Below is a diagram of the structure of each motor within the arm which highlights the control circuits which contain feedback via a potentiometer.

Figure 13 - Model of Inside Servo Motor



As the motor turns, the resistance is changed in the variable resistor which provides feedback allowing position control. This change in resistance is measured by the control board highlighted above and required voltage is output to the motor.

Potentiometers performance decays over time due to friction within the component. The solution to this issue would be to use an incremental shaft encoder in replacement of the potentiometer. As the shaft encoder has no contact between internal moving parts, the sensor has a much longer life span.

Each motor on the arm will move to within 10 degrees of the desired position. As this error accumulates across the arm, the overall accuracy of the end tool could exceed the width of the wall.

Project Outline

Here is a list of tasks completed in the whole project. More information on each of these is available below in the development section.

Choice of Robotic Arm

A robotic arm was purchased which could be interfaced with a laptop.

Research into Power Sources

An experiment was carried out and resulting decision was made about how to power the main components of the system.

Design of the Car

The car was designed to carry the brick alignment mechanism across the build area.

Building the Car

The car was constructed according to the designs presented above in Figure 11.

Programming of the Car Drive to a Specific Position

The car is able to drive along the track whilst retaining knowledge of where it is in relation to the origin.

Choice of Bricks

An ideal prototype brick was found, and the company who made it was able to send fifty bricks free of charge.

Programming the Arm to Stack Bricks

The robot was programmed to carry bricks from the source pile through a fixed path and place them onto the wall.

Testing the Machine

The tests decided whether the prototype successfully met all the goals of the project.

Development of ABLE

An operational model functioning on a building site would require properties such as resistance to water and robustness and would also need to withstand frequent hosing down with a high pressure jet. As the prototype will never be run out of laboratory conditions such considerations can be disregarded at this stage.

When considering which materials to use, the following was considered:

- Price
- Strength
- Availability
- Difficulty to work with

The design was created to be possible within the constraints of the allocated resources.

Mechanics

Materials

The steps required are to choose a robotic arm, mount it onto a base, and then motorise the base to allow movement. Calibration is needed to ensure the device knows its position in relative to the wall and so wooden stops were mounted at both ends of the track and sensors were added to detect these stops.

Robotic arm

Rather than design a new robotic arm, a pre-built arm was used which has enough degrees of freedom to reach the wall and the pile of bricks from either the crane or the car.

A number of robotic arms are available which are capable of completing the task, each with a different price. An appropriate arm was chosen due to the relatively low cost, as the budget for the project was restraining. A more expensive arm would have been more precise and would have aligned the bricks much better.

A Joinmax Digital robotic arm (Joinmax, 2005) was used in this prototype which is available as a self constructing kit. It is a seven axis servo-motor controlled arm with enough degrees of freedom of movement to complete the task, and internal feedback to control the position of the arm.

The robotic arm, like many other devices, interfaces with a computer through the com port. The laptop used to control this robot does not have a com port, so a Serial to USB converting cable and software were purchased. As the robotic arm chosen was a kit made in China, liaising with the manufacturers was difficult and increased development time.

Base

The first plans for the base of the robot involved constructing a Lego car, but after experimentation with the crane (and much stress!) it was decided that a Lego was not strong enough and so the base was made out of wood.

Figure 14 - Constructing with weak materials

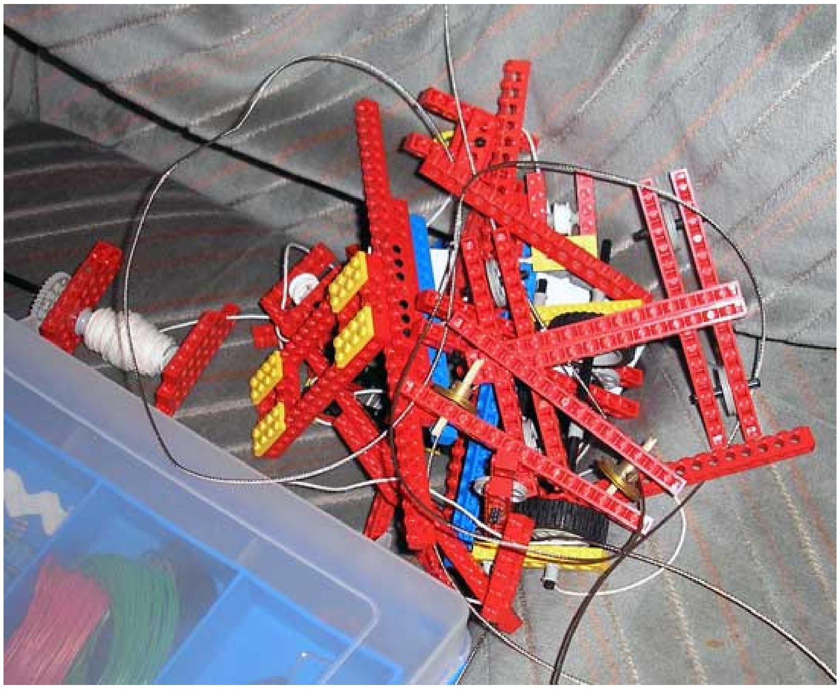


Figure 14 shows the disadvantage of working with low strength materials. This caused much frustration.

A wooden base was chosen as wood is cheap, freely available and can easily be cut and screwed into any desired size or shape which is ideal for this prototype, but not adequate for a full scale commercial application.

Mechanical Power Transmission

The transmission of mechanical power from the motor to the wheels could be done using an arrangement of cogs or a drive belt. A drive belt was used here as this arrangement is far less fragile.

The first drive belt was a standard elastic band which barely worked with issues due to uncontrolled elasticity. When the motor starts turning tension builds up in the front edge of the band until the force is equal to the inertia of the car and the car begins to move. When you stop the motor, the car continues to roll due to the tension in the front edge of the band.

The solution to this problem is an un-stretchable drive belt so the robot drive wheels were measured with a micrometer and the required band specification was calculated.

When I was asking shop staff for an un-stretchable elastic band, I was told to go look by the “buckets of steam” and the “chocolate tea pots”. This taught me you need to phrase your questions carefully when asking for help.

After some persistence, the search was directed towards a company called Benson Bett, Cleckheaton who can make a band of a specific length, width and stretch but at £25 for a one of a kind rubber band, I was advised to try a local drive belt specialist. The prototype was taken to Drive Design, Baldwin Lane, Halifax who gave me two bands for free – Result!

Track End Detection

Touch sensors were mounted onto front and back of the machine. This allows the robot to calibrate itself by detecting that it has touched the stop at end of the track. This means it can build everything from a known fixed position.

Levers

The force required to activate the push to break switch exceeded the frictional force between the wheels and the track. The result of this was the robot being unable to detect a collision. A simple solution was found, where levers were mounted over the buttons to increase the pressure applied to the button when the robot reached the end of the track.
The levers are standard cupboard hinges, made of metal with their swing is restricted by elastic bands.

Prototype Bricks

Initial plans for prototype bricks involved cutting wood into small pieces. After discovering the ideal brick (one made out of foam, with little weight and being the ideal size), research was conducted to locate the manufacturer.

V Brick Systems (2005) use a perfect brick as a marketing gimmick; however, they do not sell the items as retail, but did donate 50 free foam bricks.

Mathematics

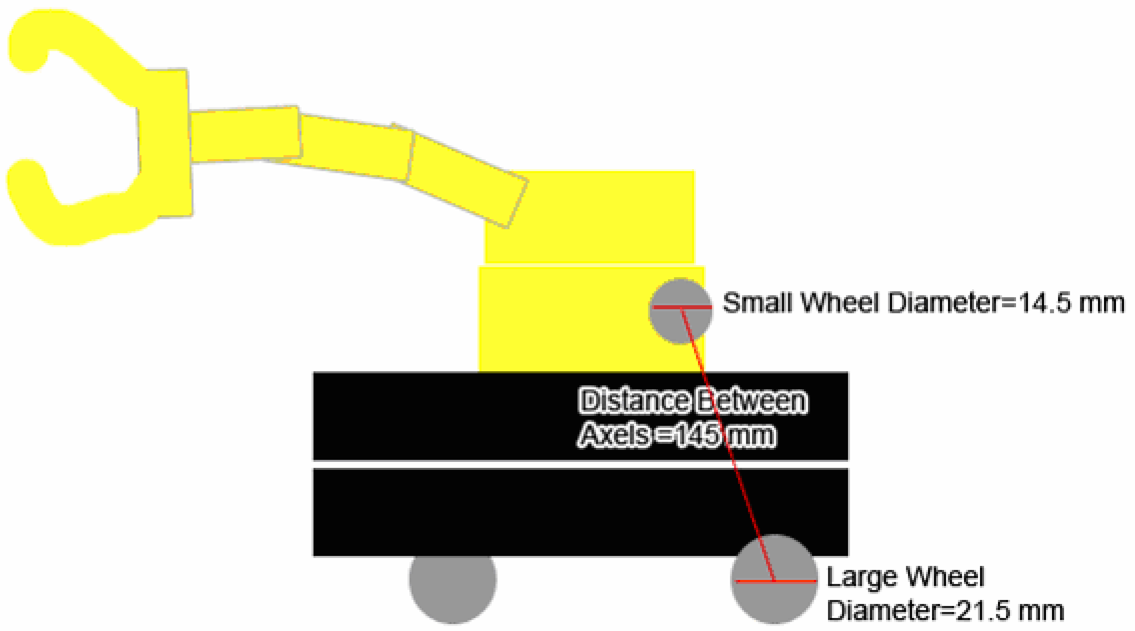
There were three areas where mathematics were used in this project to give some desired behaviours.

Robot Dimensions

The specification of the robotic arm (Appendix B) was used with the dimensions of the base (Figure 11) to calculate the drive belt length and the output angles (Inverse Kinematics) on the robot required to lay the bricks.

Drive Belt Length

Figure 15 - Calculation of Drive Belt Length


Small Wheel Diameter=14.5 mm
Distance Between Axels=145 mm
Large Wheel Diameter=21.5 mm

Circumfrance= Pi * D
The band goes round half of each wheel and the distance between them twice
Band Length = (Pi * 14.5 / 2) + (Pi * 21.5 / 2) + (145 * 2)
=22.77 + 33.76 + 290
= 346.5 mm

Figure 15 shows the dimensions of the drive wheels and the distance between them. This allowed calculations to be made for the ideal band length.

Fuzzy Logic

Figure 16 – Fuzzy Logic Membership Functions

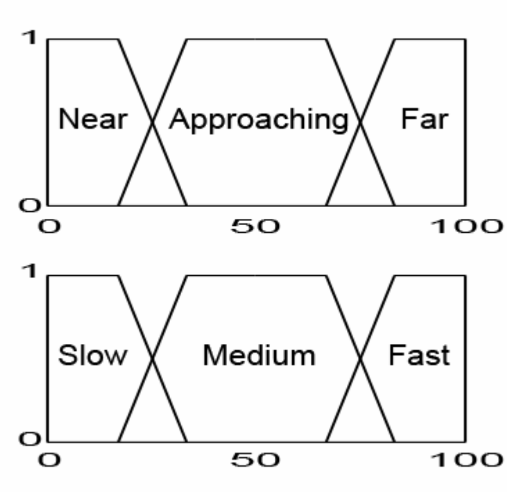


Figure 16 shows the input and output membership functions for the fuzzy logic controlled motor. There were three rules applied to these sets, as specified below.

Rule 1 : If target is near, Speed is Slow

$$\frac{\sum AX}{\sum X} = \frac{40+20}{2} * [SlowInput] = 60 * [SlowInput]$$

Rule 2 : If target is approaching, Speed is Medium

$$\frac{\sum AX}{\sum X} = \frac{60+20}{2} * [MedInput] = 40 * [MedInput]$$

Rule 3 : If target is far, Speed is Fast

$$\frac{\sum AX}{\sum X} = \frac{40+80}{2} * [FastInput] = 60 * [FastInput]$$

Output Speed =

$$\frac{(60*[SlowInput]*0)+(40*[MedInput]*50)+(60*[FastInput]*100)}{(60*[SlowInput])+(40*[MedInput])+(60*[FastInput])}$$

(See Appendix O pg 76 `Private Sub CalculateFuzzyVoltage(ByVal DistanceRemaining As Integer, ByRef OutputVoltage As Integer)`

Due to the nature of the interface between the USB interface board and the H-Bridge Motor Controller (discussed below), it is not possible to supply a voltage to the motor other than Positive max, Negative max or Zero. For this reason, a more crude method of control is used, a fixed timer.

Kinematics

The robot is required to lay a brick at a specified height, and the inverse kinematics must be calculated to move the arm to the specified location in 3d space.

The kinematics calculations are usually supplied with any normal industrial robot but as the robot is a 'Do It Yourself Kit', an alternative solution was to be found.

A search of the web for similar robot kinematic models found an Excel spreadsheet (Hoon, 2006) which contained kinematics for a similar model. In correspondence with Dr Hoon, it was identified that the kinematics equation used is not 100% accurate as link 1 and link 2 the robot are different lengths but in the spreadsheet they are the same.

This solution was attempted in code, but was not used in the end product due to errors.

This would not be a problem in the commercial application as the robot designers would calculate the inverse kinematics and supply them as standard with the purchased arm.

Mounting Robot Arm

Initially, the robot arm was mounted onto the robot the wrong way, as the 180 degree envelope of freedom was positioned over the wall. This did not allow for the robot to move to the 'get brick' position but was easily solved by re-mounting the arm with the range of movement over the brick pile and wall.

Bricks were positioned at angles which enabled the robot to function well within its limits rather than close to its constraints.

Mechanical Design Problems and Solutions

Misalignment of wheels

When the robot was first mounted onto the track, the wheels were not perfectly aligned. This was solved by re-mounting them. Another issue encountered was second hand wonky wheels. This was solved by replacing them.

Threaded Axels

Because the axels are threaded, the robot can move from left to right by about a cm as it drives back and forth. Possible solutions are to attach a bolt to either end of the axel to restrict movement or replace the axel with a non-threaded alternative.

Using Screws

The robot was held together by a combination of nails and screws. The screws worked well when a hole was pre-drilled, but the nails caused the wood to split. As a commercial version would not be made out of wood, this wouldn't be a problem.

Too Many Wires

One thing to be considered in development of other robots is the number of wires connecting the machine to a fixed point in the real world (such as a computer or plug socket).
Because the wires were mounted at ground level, the robot can drive into cables and is restrained by them. The robot derailed itself whilst trying to pull against the wires; this issue was solved by attaching the wires to a support above with an elastic band.
Wireless technology would allow remote interfacing, and batteries could be used but a decision was made against this possibility.

Misaligned Light Sensor

To allow quick adjustment of the light sensor, an adjustable height mechanism was constructed using Lego. As this wobbled and did not provide any significant advantages, it was replaced with a cable tie at a fixed height.

Grabbing Mechanism Experimentation

As an experiment, the grabbing mechanism was covered in rubber. The difference was worthwhile when the robot was handling wooden bricks, but as the foam bricks squash slightly, the rubber was not necessary and was removed.

Electronics

Choosing a power supply

There were three options for powering the machine, mains, USB and battery. The devices needing power are USB interface board, motor, robot and light sensor.

When batteries become drained, performance decreases. This was learnt this when using a Lego Mindstorms Robotics kit, and discovering the robots drive a different distance in the same time period when the battery is low.

For this reason, it was decided that all devices should be mains powered to prevent the possibility of a battery failure spoiling performance.

The focus of this project was to build a functioning prototype with limited resources. In a commercial version, the robot would be powered by a 110 Volt AC supply as this is a health and safety requirement HSE (2005). As there was no need for this level of control in this prototype, power regulating circuitry was not developed.

Robot Power Supply

The first thought was to power the robotic arm with batteries as described in the instructions manual. The robotic arm is powered by five 1.5v AA batteries which is isolated from the rest of the project. This would prevent any accidental electronic damage to the robot.

However, because of past experience with battery powered devices, it was decided to purchase a regulated mains power supply for the robot.

USB Interface Board Power Supply

The USB interface board draws power from the USB port in the laptop.

Motor Power Supply

As the USB interface drew power from the laptop, an experiment to power the motor from the USB was carried out.
A USB cable was cut and a transistor was soldered to the end. When the control signal was measured with the transistor attached to the digital-out port of the USB interface board, a reading of 4.4 volts showed there was a high enough voltage to power the motor. When connected, there was not enough power to spin the motor.
As a test, an attempt was made to connect the motor directly to the USB interface board power supply which resulted in the circuit powering down and losing the connection to the computer.

An old phone charger proved to provide an adequate supply to run the motor.

Line Sensor Power Supply

The sensor also draws its power from an old phone charger.

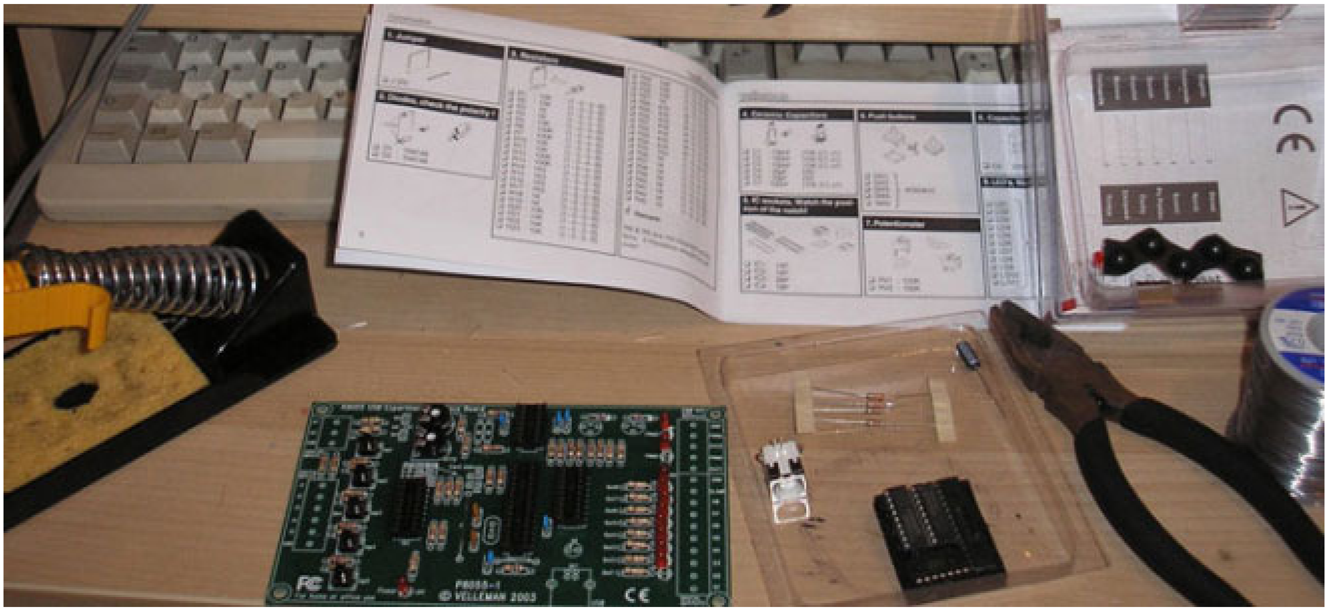
Buzzer Power Supply

The buzzer draws its power out of the digital out port on the USB interface board.

Building and Using a USB interface circuit

A USB Interface Experiment kit is available at Maplins for little money, which allows interfacing of electronics with a computer.
The kit was purchased and the board was made by following the instructions manual.

Figure 17 - Photograph of Interface Board under Construction



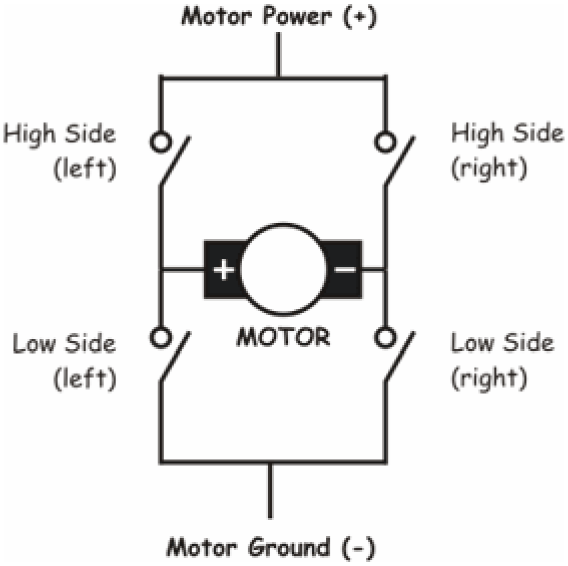
The interface board does not have a reprogrammable microprocessor onboard so all processing is done on the pc. This will allow a higher level of control for the end user.

Motor Control Interface with USB circuit

My Experimental Motor Control

The output signal from the USB interface board is always a positive voltage. To make the motor spin in two directions, a negative voltage is needed, so some control circuitry is required. McManis (2003) provides an idealised diagram along with an in depth description into how the H Bridge works.

Figure 18 - The H Bridge Circuit



When the top left and bottom right switches are pressed, current flows through the motor in one direction. When the opposite pair of switches is pressed, current flows in the other direction.

Figure 19 - First Attempt at Home Made H Bridge

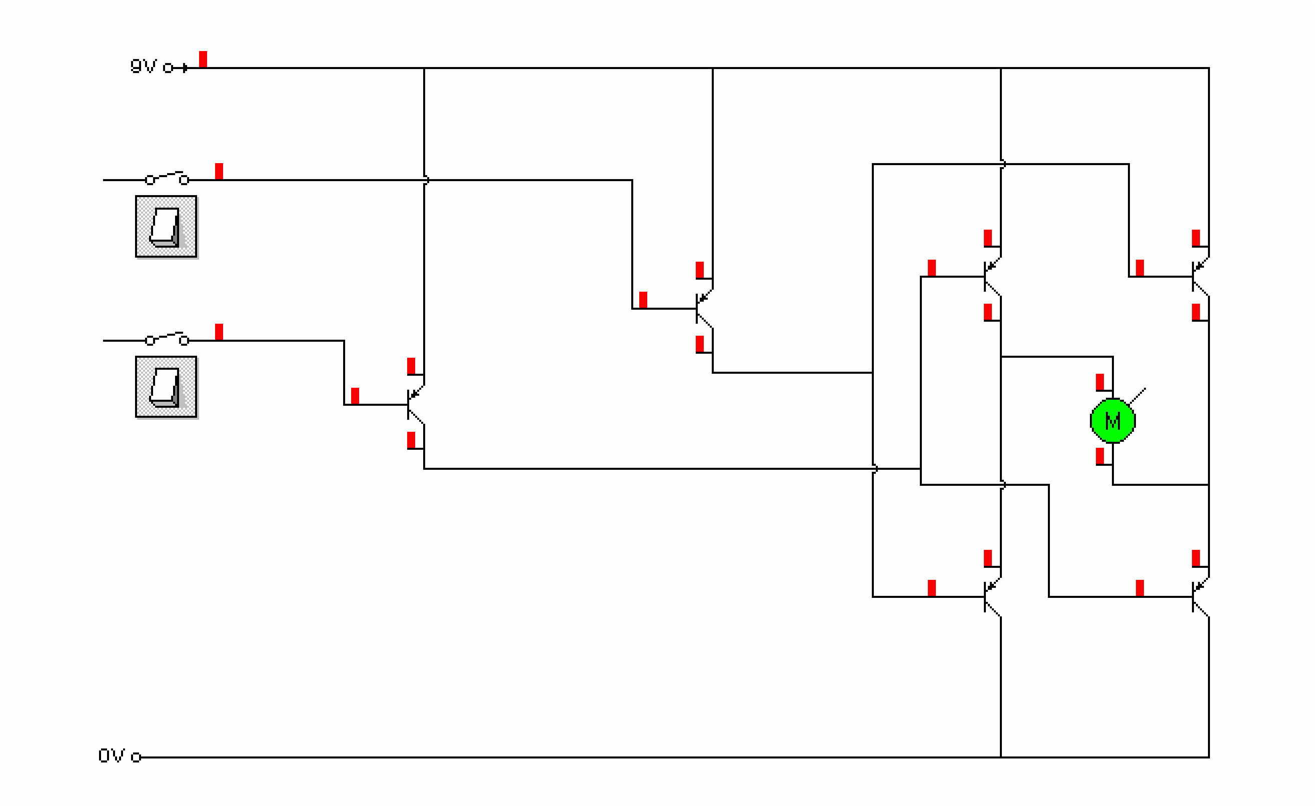


Figure 19 shows the circuit diagram that was attempted to apply a negative voltage over the motor. When testing this circuit, the output voltage ranged from -0.35 volts to 0.35 volts. The switches represent the signal from the USB Interface board. Each of the signals is fed into two diagonally opposite transistors as described in Figure 18.

Figure 20 - Second Attempt at Home Made H Bridge

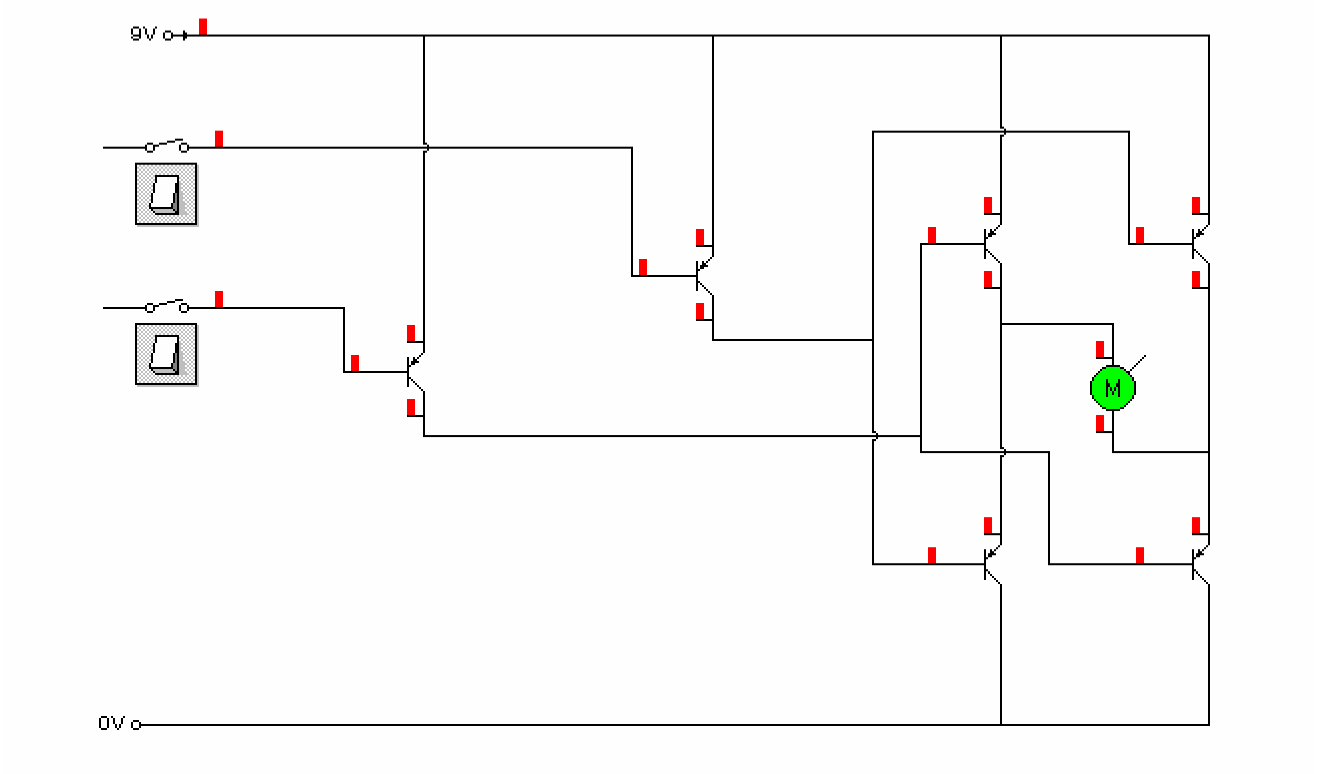


Figure 20 shows the circuit diagram modified with a 'Darlington Pair' to step up the input voltage to the H Bridge. When testing this circuit, the output voltage ranged from -0.63 volt to 0.63 volts.

Figure 21 - Second H Bridge built on a bread board

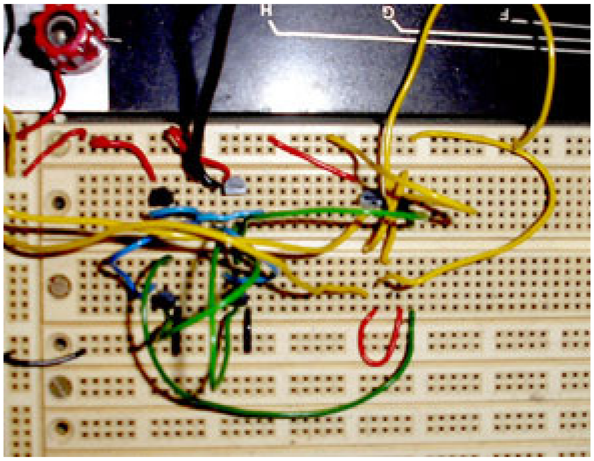


Figure 21 shows the breadboard layout of circuit diagram in Figure 20.

On searching the web for other motor control circuit diagrams, a ready made solution (H Bridge Controller below) was discovered so work on this circuit became unnecessary.

H Bridge Motor Controller

The H Bridge Controller is a ready made solution to the motor drive issue.
The circuit was purchased and easily interfaced with the USB interface board.
There is an enable pin on the H bridge board which does not function the way it should. When power is supplied, the circuit is disabled and when the power removed, it works. This contradicts the instructions book, but is easily corrected in software.

Choice of sensors

The inputs to the system are two collision sensors, two input buttons and a line sensor.
A line sensor was purchased instead of using an incremental shaft encoder which turned out to be a bad decision as the desired effect was not obtained due to programming restrictions. This should have been done using shaft encoders to count the distance the car has travelled.

White paper was mounted under the track to allow the black lines (the small black sleepers on the train track) to stand out more. The line sensor worked well between about 1 cm and 0.5 cm.

The collision sensors are standard ‘push to break’ switches, the emergency stop button is an illuminated (non-powered) latching ‘push to make’ switch and the reload button is an illuminated (non-powered) ‘push to break’ switch.

Issues encountered and resolutions

Bad Design

Two buttons were mounted onto the robot, an emergency stop button (red) and a reload button (blue). However, the robot builds a wall in front of these buttons which does not make them very accessible. As a result, they were ignored and a button was introduced to the control software

Loose Connections & Faulty wires

The robot started only driving one way. The problem was two of the wires coming out of the H Bridge board had broken. This was easily solved with some solder.

Disassembly

As ideas developed, it was necessary to disassemble the robot to conduct tests and experiments. This caused problems when the circuits are wired together and need to be removed. A solution was needed to allow complete disassembly without the need for cutting wires.

Small connectors are introduced in the middle of each wire to allow quick disassembly. Connectors are colour coded to ensure quick and accurate reconstruction of the robot and labelled wires make it easier when a wire falls out.

Damaged robot

The robot was purchased second hand, and through testing it was discovered that two of the motors were not performing correctly.

A test was carried out involving a software loop which continually moved the faulty motors in a fixed pattern so their performance could be observed. This highlighted damaged motors which were replaced. Experimentation with an oscilloscope revealed the control signal was present and hence it was the motor which was damaged.

Another motor was damaged when a programming error tried to move the arm into a position it was not capable of moving to.

The robotic arm is unaware of its physical location which means it cannot make small adjustments to compensate for error. Also, it cannot detect a damaged motor therefore some testing system would be required.

Programming

Choice of development environment

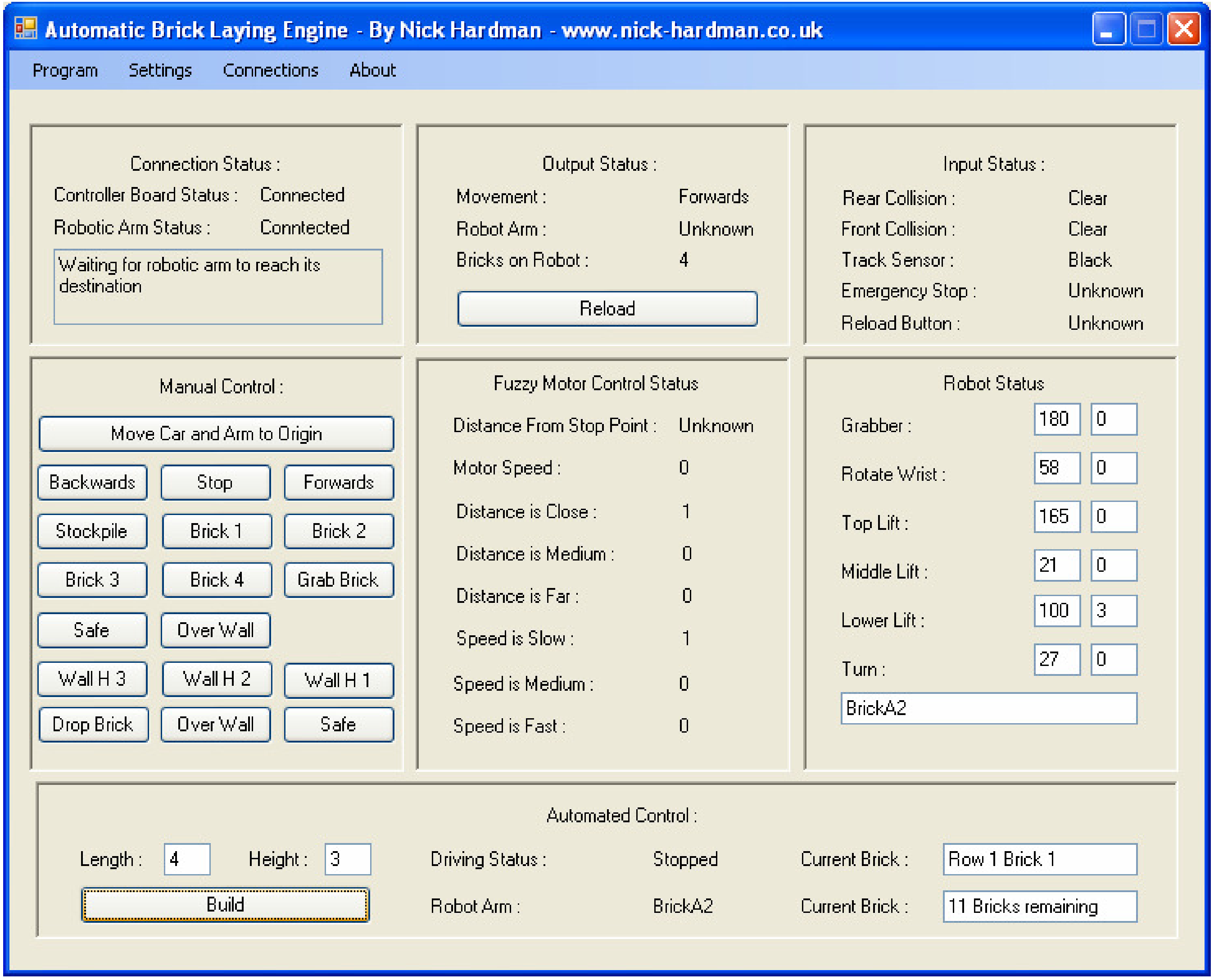
The manufacturers of the USB interface board provided code in Visual Basic 6, Visual C++ 6 and a couple of other languages.
The manufacturers of the robotic arm both provided source code in Visual Basic 5. (Please see Appendix M & Appendix N.
To ensure cross compatibility of both sources, source code was upgraded to the most up to date version of Visual Basic available (Visual Basic.Net).

Timing and Task Scheduling

The timing of positioning a brick is critical as a brick can only be laid when the car is at the correct point along the track. A timer has been used called which causes the robot to wait a number of seconds in between actions to allow the robot to arrive at its given position.

ABLE Control Panel

Figure 22 - Screenshot of Control Software



Connection Status :
Controller Board Status : Connected
Robotic Arm Status : Conntected
Waiting for robotic arm to reach its destination

Output Status :
Movement : Forwards
Robot Arm : Unknown
Bricks on Robot : 4

Input Status :
Rear Collision : Clear
Front Collision : Clear
Track Sensor : Black
Emergency Stop : Unknown
Reload Button : Unknown

Fuzzy Motor Control Status
Distance From Stop Point : Unknown
Motor Speed : 0
Distance is Close : 1
Distance is Medium : 0
Distance is Far : 0
Speed is Slow : 1
Speed is Medium : 0
Speed is Fast : 0

Robot Status
Grabber : 180 0
Rotate Wrist : 58 0
Top Lift : 165 0
Middle Lift : 21 0
Lower Lift : 100 3
Turn : 27 0
BrickA2

Automated Control :
Length : 4 Height : 3
Driving Status : Stopped
Robot Arm : BrickA2
Current Brick : Row 1 Brick 1
Current Brick : 11 Bricks remaining

Figure 22 shows a screenshot of the operation software displaying seven panels of options and information.

Top left panel shows the connection status and an advice box where messages are displayed which inform the user what the robot is currently doing.
Top middle panel shows the output status. The brick count is displayed here along with the reload button.
Top right panel shows the status of each of the inputs when last read.

Left middle panel shows the Manual Control Panel. Here, instructions can be entered manually.
Centre panel shows the output from the fuzzy logic maths.
Centre right panel shows the robotic arm status. In this panel, there are two boxes associated with each motor, the left of the two shows the current position, and the right

shows how much that motor will move each time the `Private Sub RefreshRobotPosition()` is called.

The bottom panel is automated control. By entering a length and height, then clicking build, the robot will automatically build a wall.

Program Scheduling

The program is controlled with a number of multi threaded self-dependent functions running simultaneously passing data through function calls and global variables. Each function is named according to its job, and a brief description is supplied in the comments. A pause function was written to force the car to wait for the robot. (See Appendix O pg 67 `Sub Pause(ByVal WaitFor As Integer)`

Figure 23 - Flowchart representation of Program

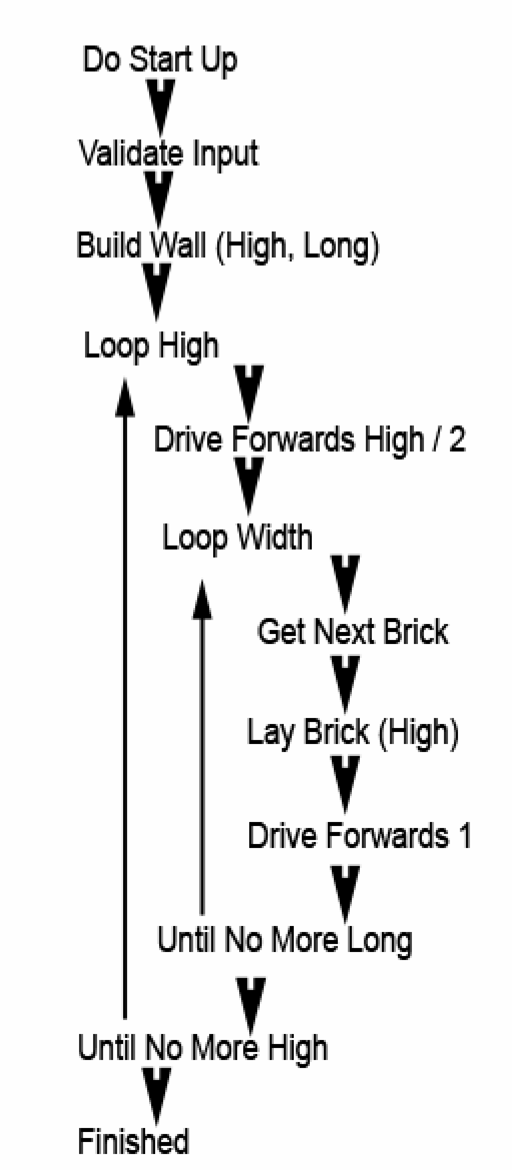


Figure 23 shows how the software calls the functions required to construct the wall. This is a simplification. (See Appendix O pg 57 `Public Sub BuildWall(ByVal WallLength, ByVal WallHeight)`

Fuzzy Logic

Fuzzy logic was used to calculate the output voltage required for the motor to allow a less sudden stop. Due to the nature of the H Bridge Controller, this calculation of voltage (section of code) could not be applied. There was an error concerning the output voltage when the robot was close to the target destination. This problem was not resolved as the code was no longer required.

Mechanical Considerations

The robot arm has two motors which are mechanically connected and moving them in different directions would cause the robot to break. A function was coded to prevent this from ever happening.

Communication

The code for communication is included in the Appendix O.

Communicate with robot

The robotic arm was interfaced with a computer using a 'USB Virtual Com Port'. This was a shiny wire purchased from China which has a USB end and a Serial end.
The 'Virtual Com Port' allowed the robot to be controlled using the 'Com Dll' within Visual Studio without a physical Com Port. This allowed ABLE's control software to write to the registers on the robot control circuit in order to move the robot.

Communicate with USB interface board

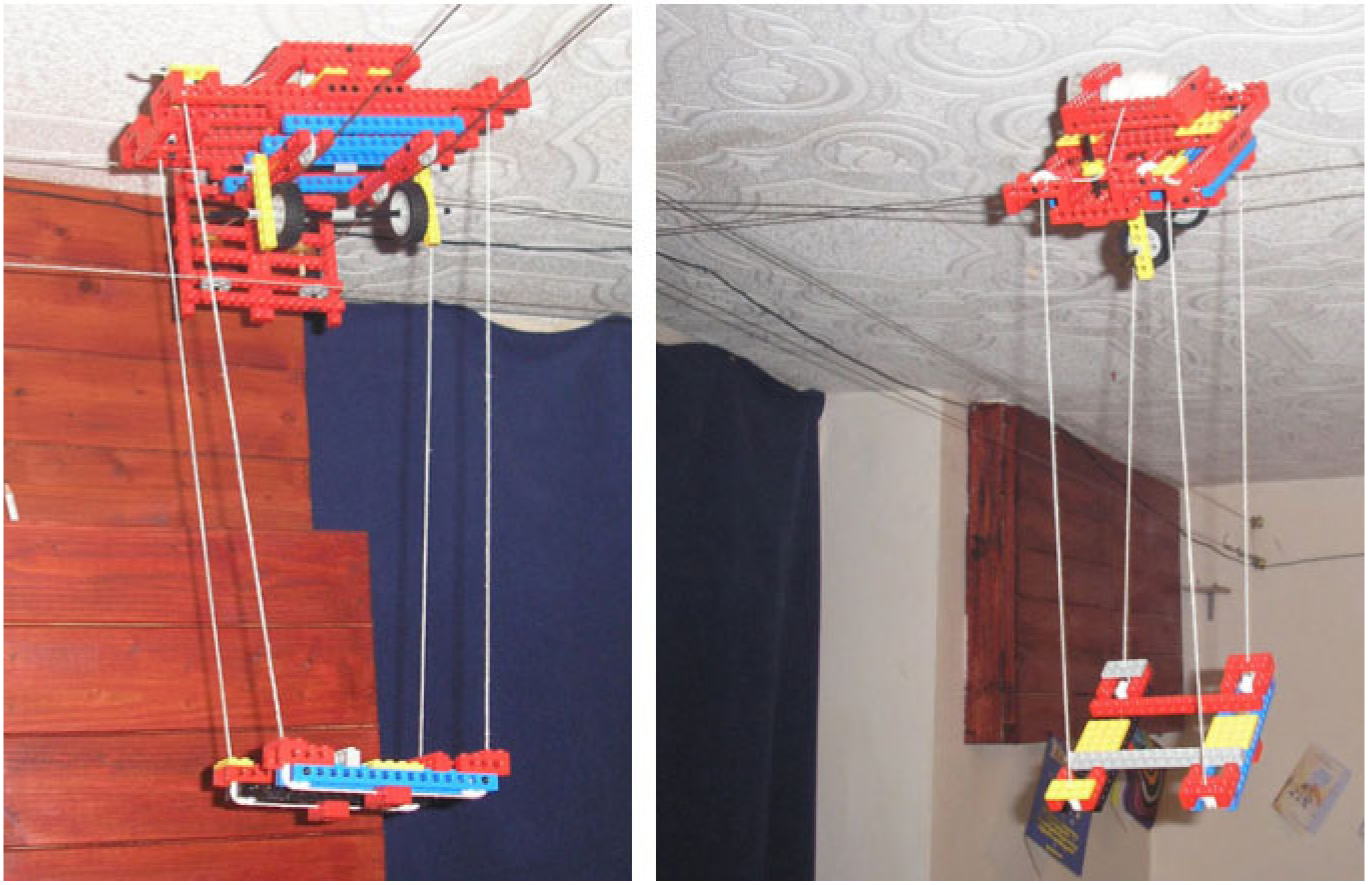
The USB interface came with open source software for connecting to the board which worked well once upgraded to Visual Basic.Net.

Experimentation with the crane

Whilst ABLE proves the concept that automated construction of a wall is possible, it does not have much use on the common building site. The limitations are that the robot can lay bricks along a single plane (the xy plane) as the robot cannot drive off the track.

A potential prototype crane was constructed according to designs specified earlier. The crane was un-motorised and made from steel cables and Lego as an experiment.

Figure 24 - Photograph of Crane System



The metal cables used to make the crane sometimes oscillate at a low frequency. This movement causes problems for a robot hanging on the bottom. This could be resolved if the robot clamped onto the structure being built.

Balancing the crane also became an issue which is not a problem in the track based system. This problem could be resolved by using a solid steel frame to mount the robot on instead of steel cables.

Health and Safety

During the development of this project, health and safety considerations were made resulting in a number of features designed to protect the people working around the prototype.

When the robot arm is carrying a brick from the source to the destination, in the real world it is likely to hit someone on the head with a brick. Staff on building sites wear helmets, but not face guards. For this reason, a beeper was mounted to act as a warning when the robot arm is turning. A switch was mounted on this to turn off this potentially irritating feature, but this would not be done on a full scale model.

A red emergency stop button has been mounted onto the front which latches in the on state which ensures the button press is detected regardless of when the input is sampled.

An emergency stop button on a commercial version would be required to stop all movement instantly. This could be done by cutting the power, but consideration must be made to electrically controlled motors as they loose all torque when the power is cut.

Results

During this project, bits of wood, educational toys and random donations of materials have been used to automate a task which people pay hundreds of thousands of pounds for. With this in mind, it is understandable that a brick will occasionally be dropped or misaligned by a couple of centimetres or a few degrees.

Interfacing PC with USB Interface Board

A USB Experiment Kit was purchased and constructed without any problems.

Figure 25 - Comparison of Desired out against Measured Out

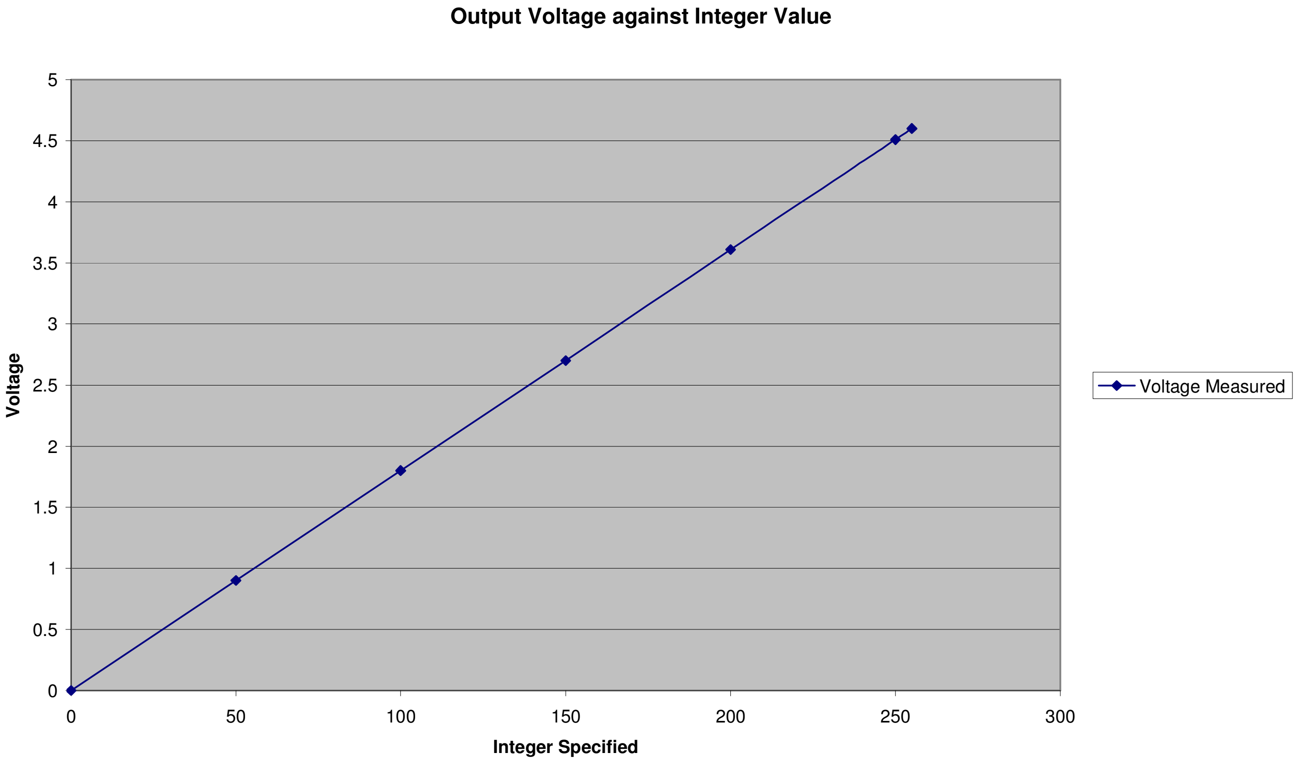


Figure 25 shows the measured voltage (on the y axis) plotted against the integer value specified (on the x axis).

Figure 25 also shows that the output voltage on the USB Interface board analogue ports behaves linearly.

Line Sensor Test Results

Once the robot was driving forwards and backwards, the line sensor was tested to determine which speed settings would give optimal detection of the number of sections of track the car has passed over.

Figure 26 - Adjusting Motor Speed to Test Sensor

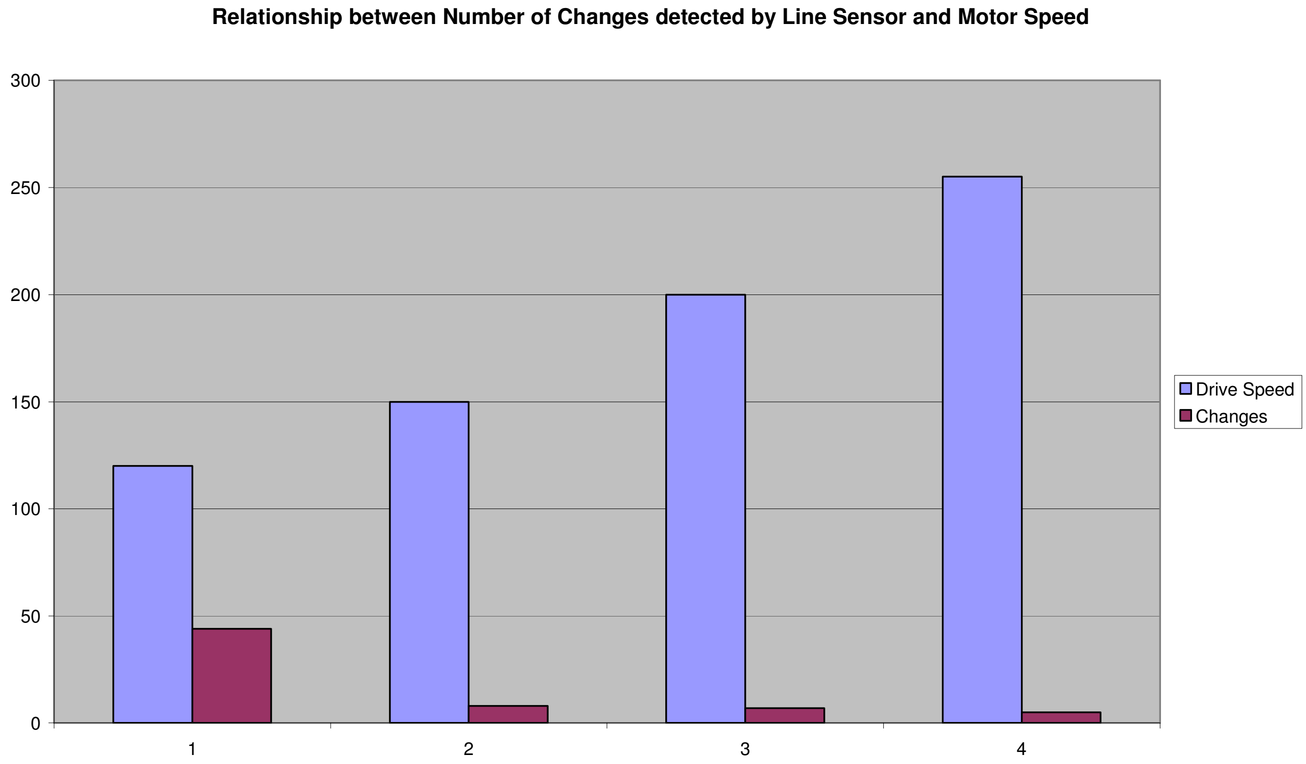


Figure 26 shows how, as the drive speed decreases, the number of pieces of track detected increases. When the speed was set below 120, the motor failed to move. As the number of pieces of track detected is much less than the number of pieces of track driven over, these settings are too inaccurate to work with.

Figure 27 – Varying Drive Time

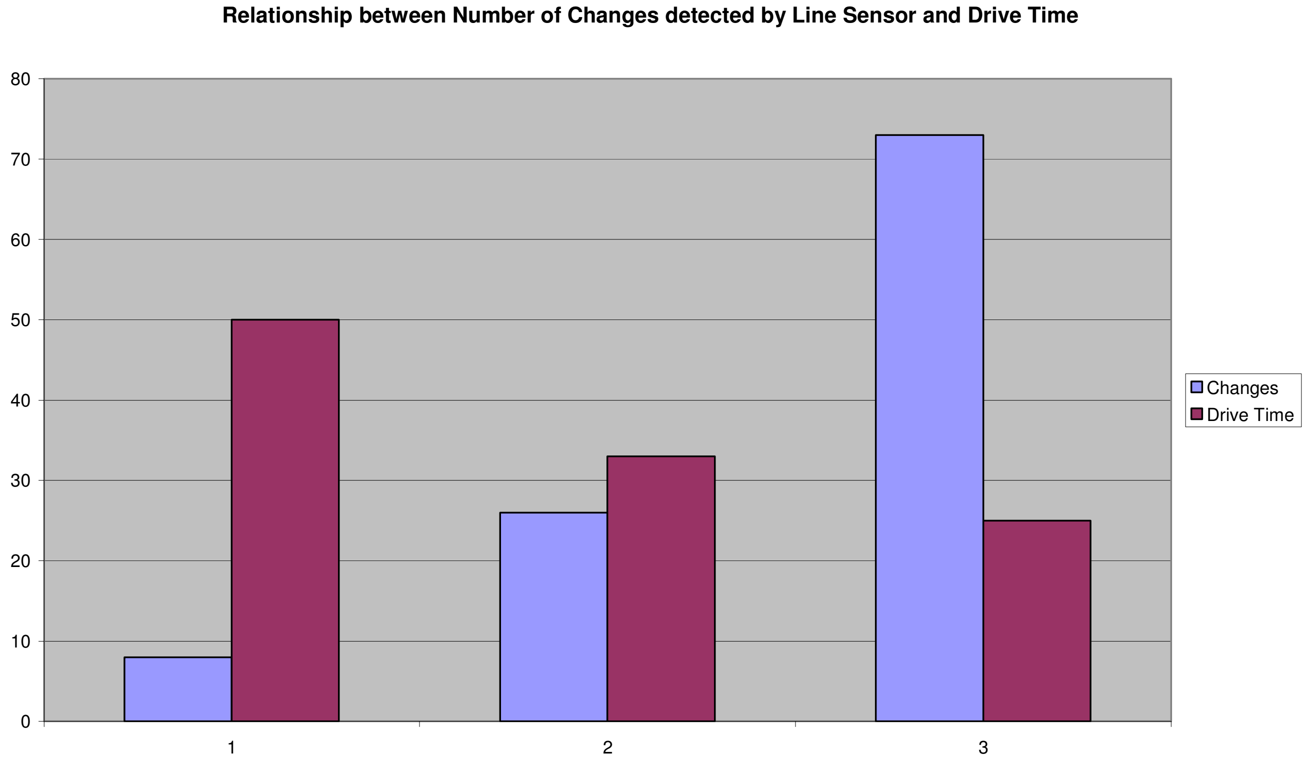


Figure 27 shows the effect of placing waiting intervals in the drive sequence. In Test 1, the effect of waiting 50% of each second and driving for 50% does not increase the number of changes detected. Test 3 showed using a 25% drive time and 75% wait time, far more pieces of track were detected. Test 3 counted more than the total number of pieces of track crossed; the reason for this is believed to be the robot oscillating over track edges each time it stops.

As the project drew to a close, the decision was made to disregard the line sensor and control the position of the car by driving the robot with small counted pulses. This was done by using a Timer in Visual Basic which applied power across the motor during the ‘drive’ time and cut the power during the ‘wait’ time.

Robotic Arm Test Results

Interfacing PC with Robotic Arm

For a number of weeks during the project, Visual Studio.Net would not interface with the robot. When searching the web for the error codes, few results were found, none of which appeared relevant to Visual Basic.Net.
It is believed the problem was due to a bad installation of Visual Studio.
The solution was to reformat the laptop a number of times solving the problem using trial and error by installing software in different orders.

Operating Speed of Prototype

As the prototype is not laying real bricks, nor placing cement, it would not be fair to compare its working speed with the speed of a human brick layer.

Move Car to a Specified Position

The robot is able to drive along the track in increments of about 2 cm. As the length of a brick is 8 cm, this provides enough accuracy to stop the car in brick intervals.

In 100ms, (minimum sampling rate in Visual Basic) at the minimum driving speed the robot can drive over more than 1 black line on the train track. This means the sampled input signal changes faster than the minimum sampling rate and a form of aliasing occurs. This renders the line sensor redundant so it was decided to program the robot to drive forwards in notches of the minimum increment. (See Appendix O pg70 `Sub MoveCar(ByVal Distance As Integer, ByVal Direction As String)`)

Construct and Mount Brick Laying Mechanism

Figure 28 - ABLE Holding Brick

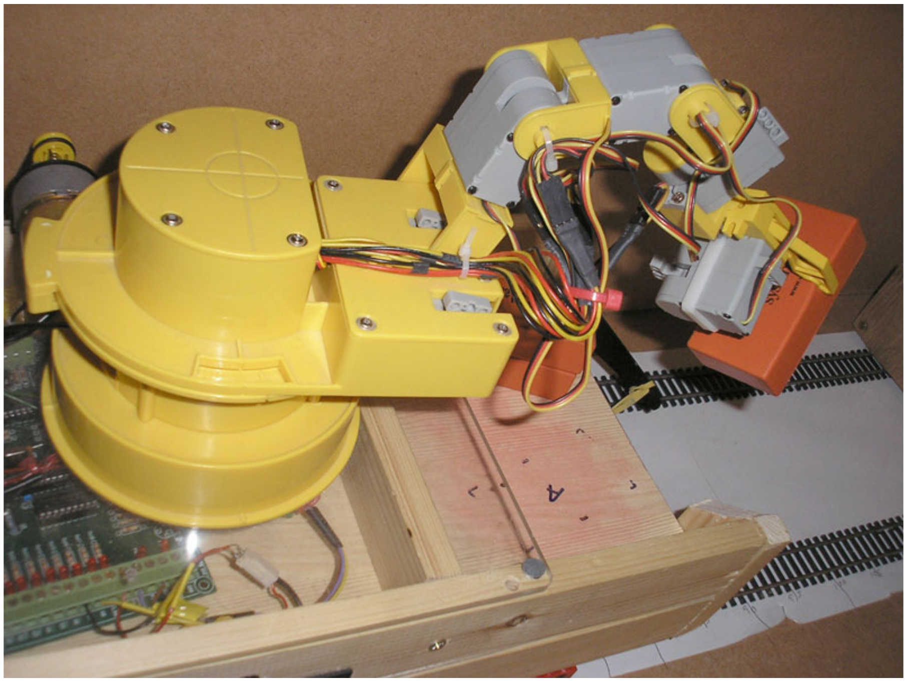


Figure 28 shows the Joinmax Digital robotic arm mounted on top of the car holding a brick.

Monitor Brick Count and Request Reload

The robotic arm keeps a count of the number of bricks in the holding area (where the bricks are stacked on the car) using a global variable called BricksOnRobot.

When the robot is given a new instruction, the variable holding the number of bricks is checked. (See Appendix O pg 62 `If BricksOnRobot = 0 Then`)Should this variable be equal to zero, the robot will stop and wait for confirmation that reloading has been completed via a button on the control software.

Program Arm to Get Next Brick

The robotic arm did not have a high repeatability as each time it moved to the brick pile it would stop at a different position. As a result of this, occasionally, the robotic arm would not grab the brick adequately and would drop the brick when transporting it to the wall.

Program Arm to Lay Brick

The robotic arm was capable of laying a brick at three specified heights. Video evidence is provided on the CD accompanying this document. (See Appendix O pg 63 `Public Sub LayBrick(ByVal Height As Integer)`
)

Potential Development of ABLE

The small scale prototype developed in this research project has proved the concept that automated construction is possible. ABLE could be developed to work along side an ever growing fleet of construction robotics. On developing a commercial application, other considerations would be made. These are:

Brick Orientation

The lay brick function could be modified to include an orientation which would allow walls to be built in different shapes.

Accurate Positioning

Accurate positioning can be achieved by using a more accurate robotic arm and incremental shaft encoders on the driving mechanism to measure distance travelled.

Connecting a camera on the end of the robotic arm could provide video feedback allowing image recognition software to notice small misalignments such as a brick being slightly misaligned on the brick pile.

Electronic level sensors could provide accurate readings allowing the robot to make minor adjustments.

CAD Software Interface

It is possible to interface a construction robot with a CAD program as demonstrated within this project. Excellent construction design software is available and development kits for such programs could be used to output the designs as a series of instructions for a robot.

Brick Size & Shape

ABLE could be modified to carry and position different bricks of different shapes and sizes.

Curb Laying

Another potential application for this technology is a Curb Laying Robot. Street curbs are heavy and require careful alignment. A future modification of ABLE could assist road construction engineers by laying curbs.

Health & Safety

In the prototype here, the emergency stop button was inaccessible. On a full size prototype, multiple emergency stop buttons would be required which are easily accessible. Other health and safety considerations would also need to be made.

Excess Cement

The technique of aligning bricks results in mortar being pushed out of the spaces between the bricks which require smoothing off. If the mortar was applied via a system similar to the contour crafting technology discussed earlier in this report, there would be no excess cement which later required removal.

Cement Damage

When working with cement, one of the most significant issues is that cement will dry and stick to everything it comes into contact with. This could clog up machines, preventing them from functioning.

Building Speed Restriction

There is a general rule of thumb in bricklaying which specifies no more than eleven bricks high should be laid in a single day. Should this not be followed, the weight of the wall will begin to squash the mortar out of the joints leaving an unstable wall.
A potential solution to this problem would be to apply less water to the mix, or apply heat to the newly constructed wall to speed up the drying (and strengthening) process.

Conclusion

Any error in a small scale prototype may not be measurable. When scaled up to a full scale model, this error could accumulate over the height of a construction and hence this issue needs to be addressed.

Construction robotics is in development across the world. These companies need to work closely together to produce a fleet of construction robotics running in synchronisation capable of automatically constructing any design.

Project ABLE has proved the concept that automated construction is possible. Through researching for this project, a number of construction robots have been discovered and highlighted in this report. Through the correct application of these emerging technologies, it is possible to automate the construction industry.

Many aspects of manufacturing are currently done using a combination of robotics and computer integrated systems. In my opinion, the construction industry resembles the manufacturing industry closely and so it would appear that both industries are set to follow the same path; machines can work faster and more efficiently than humans with a much higher precision.

Modern constructions today are designed and tested using computer aided software a long time before the first brick is laid. This software outputs a specification which is carried out by a network of people with tools. I propose that the output of this CAD software could be interfaced with a network of robots to automate the construction process.

Instead of providing a blueprint to a brick layer, a list of instructions could be passed to a commercial application of project ABLE. Instead of showing a painter or a plasterer a 3d model of a room which has been made as part of the building design process, this instruction could be exported into a set of rules e.g. “navigate to this coordinate”, “run the paint roller along this vector in 3d space” to apply the material (paint or plaster) according to the architect’s original plans.

Whilst it remains theoretically possible for a completely automated building site, as automated machines begin to integrate their way into the construction industry, many new health and safety hazards are introduced to the people around them.

A common problem for the development of any robotics is the fear of mass unemployment. Without embracing the latest technologies, countries will fall behind the rest of the world’s development with a devastating impact in the long term. Owen (1984) warns that “the countries and companies which do not adopt the mega industry of microelectronics in all of its guises will fail to be competitive and so fall into stagnation, with disastrous results for the population (Pg 108)”. Basically, if we do not develop the machines which are feasible to make, someone else will and we will lose our market share in the construction industry. Why pay British workers £400,000 for a construction if a Japanese company is offering to a higher quality job for £135,000? (These figures are made up to demonstrate the point).

Simple processes are easy to automate and hence financially viable, but for many complex processes it may not worth the cost of developing an automated solution. Whilst automated construction machines will reduce the work load, humans are required to develop, use, maintain and control the machines. The same is true for the planning of construction – a person must design a structure before a machine can be used to assist in the construction of it. As Owen asserts, “machines never replace humans within the manufacturing industry, they simply remove the need for humans to perform certain tasks (Pg 108)”.

References

Cook, N. (1993) http://research.cs.ncl.ac.uk/cabernet/www.laas.research.ec.org/esp-syn/text/6450.html (accessed 02/05/06)

DTI (2004) Construction Statistics Annual. London: DTI

Gambao, E.& Balaguer, C. (2002) Robotics and Automation in Construction. In Robotics & Automation Magazine, IEEE. Volume 9, Issue 1, Mar 2002 Pp:4 – 6

Hoon, H. Inverse Kinematics cited in http://www.lynxmotion.com/images/html/proj058.htm (accessed 03/05/06)

HSE, Health & Safety Executive (2005) http://www.hse.gov.uk (accessed 03/05/06)

Hughes, S. (2004) GB Construction Review cited in http://www.construction-statistics.co.uk/ (accessed 26/10/05)

Integrated Engineering Software (2006) http://www.iesweb.com/products/quickrwall/ (accessed 13/12/05)

Intelligent Control Group, University of Lancaster (2006) http://www.engineering.lancs.ac.uk/REGROUPS/INTELLIGENT/research/CRarticleDetail.asp?ID=25 (accessed 03/05/06)

Joinmax (2005) http://www.pololu.com/products/joinmax/0097(accessed 7/11/05 – 14/12/05)

Khochnevisk, B. (2004) Houses of the Future: Construction by Contour Crafting, Building Houses for Everyone. University of Southern California

Linklater, R. (2005) http://www.abc.net.au/newinventors/txt/s1300261.htm (accessed 14/10/05)

McManis,C. (2003) H-Bridges: Theory and Practice http://www.mcmanis.com/chuck/robotics/tutorial/h-bridge/

Milton Building Services, London http://www.miltonsonline.com/enquiries.html (accessed 01/11/05)

Owen, T. (1984) Flexible Assembly Systems – Assembly by robots and computerised integrated systems. US:Springer

Ram (2006) http://www.ramint.com (accessed 03/05/06)

Takenaka Corporation (2005) http://www.takenaka.co.jp/takenaka_e/robots/robots.htm (accessed 03/05/06)

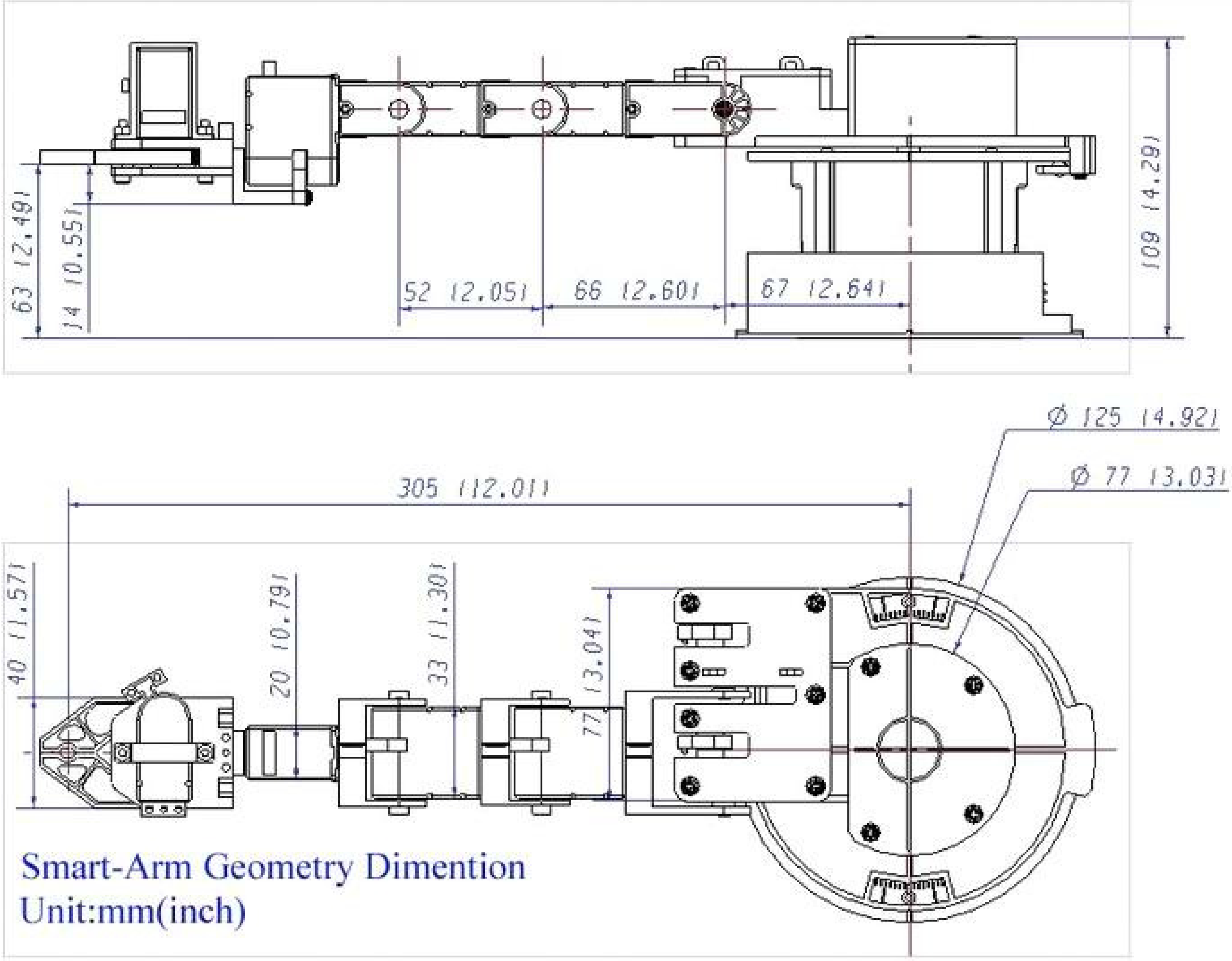
VBrick Systems (2005) http://www.vbrick.com/company/contact_us.asp (accessed 13/12/04)

Appendices

Appendix A – Letter of thanks to supporting people

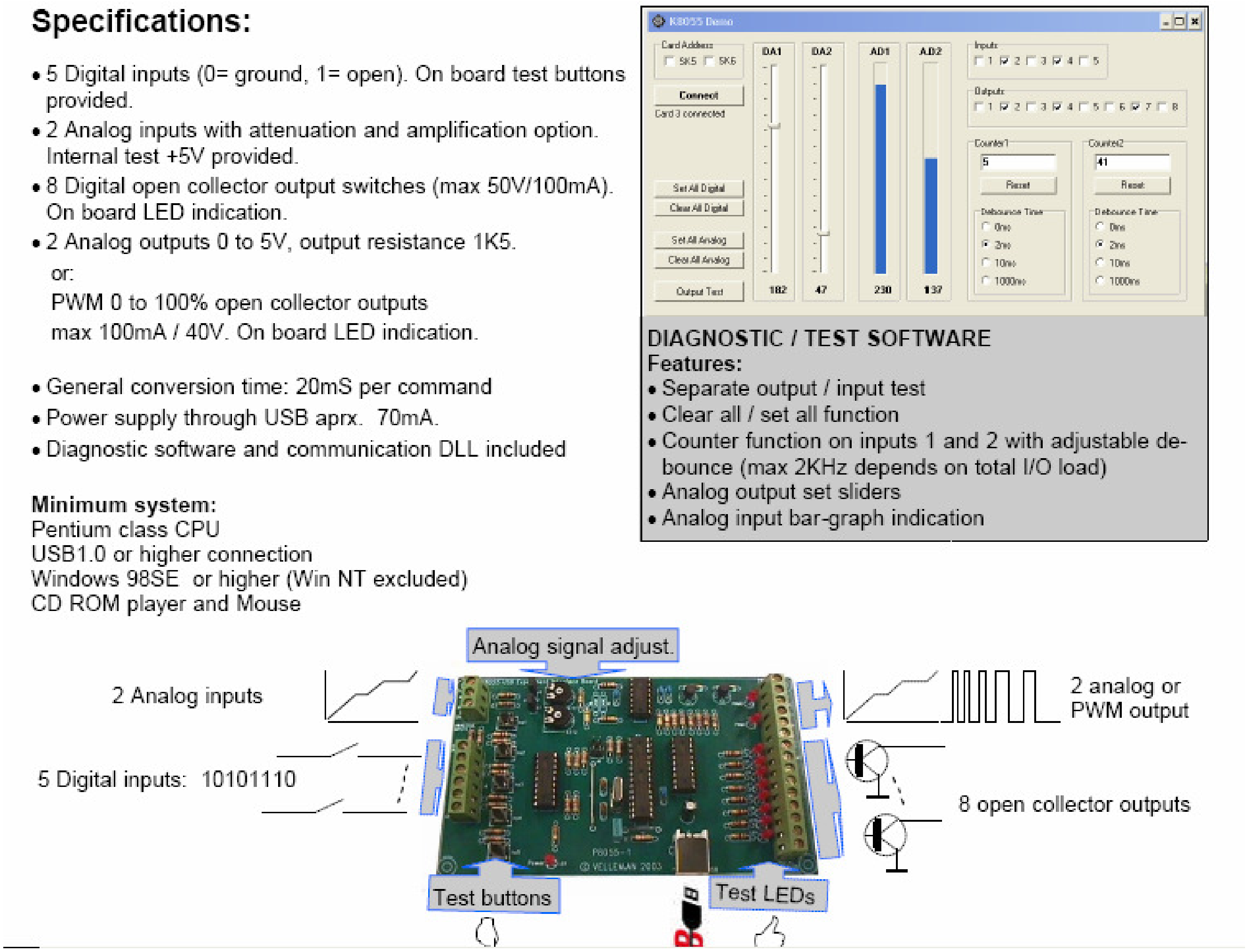
I'd like to say a special thank you to the following people who helped make this project a success. John Readle, for providing support and advice, Benson Bett for giving me an insight into custom band production and introducing me to a local drive belt specialist, Drive Design (Baldwin Lane, Halifax), for donating two ideal drive belts, VBrick Systems for the kind donation of 50 foam bricks for use throughout the project.

Appendix B – Joinmax Robotic Arm Specification



Smart-Arm Geometry Dimention
Unit:mm(inch)

Appendix C – USB Interface Board Specification



Specifications:

- 5 Digital inputs (0= ground, 1= open). On board test buttons provided.
- 2 Analog inputs with attenuation and amplification option. Internal test +5V provided.
- 8 Digital open collector output switches (max 50V/100mA). On board LED indication.
- 2 Analog outputs 0 to 5V, output resistance 1K5.
  or:
  PWM 0 to 100% open collector outputs
  max 100mA / 40V. On board LED indication.

- General conversion time: 20mS per command
- Power supply through USB aprx. 70mA.
- Diagnostic software and communication DLL included

Minimum system:
Pentium class CPU
USB1.0 or higher connection
Windows 98SE or higher (Win NT excluded)
CD ROM player and Mouse

K8055 Demo

Card Address: ☐ SK5 ☐ SK6

Card 3 connected

DA1: 182 DA2: 47 AD1: 230 AD2: 137

Inputs: ☐ 1 ☑ 2 ☐ 3 ☑ 4 ☐ 5

Outputs: ☐ 1 ☑ 2 ☐ 3 ☑ 4 ☐ 5 ☐ 6 ☑ 7 ☐ 8

Counter1: 5

Counter2: 41

Debounce Time (Counter1): ○ 0ms ● 2ms ○ 10ms ○ 1000ms

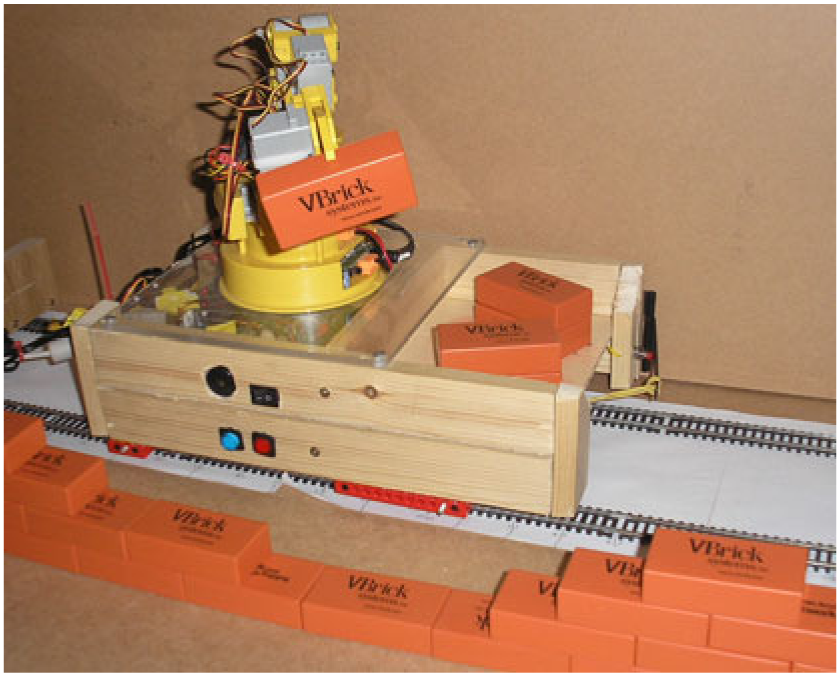
Debounce Time (Counter2): ○ 0ms ● 2ms ○ 10ms ○ 1000ms

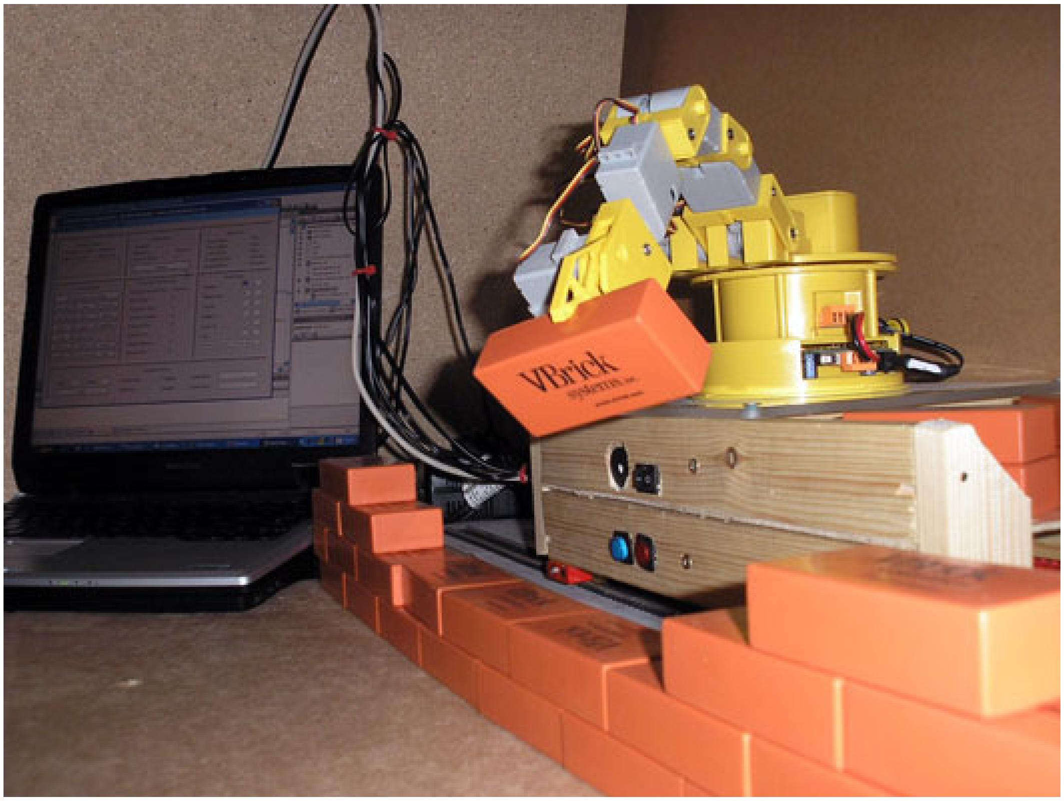
DIAGNOSTIC / TEST SOFTWARE

Features:

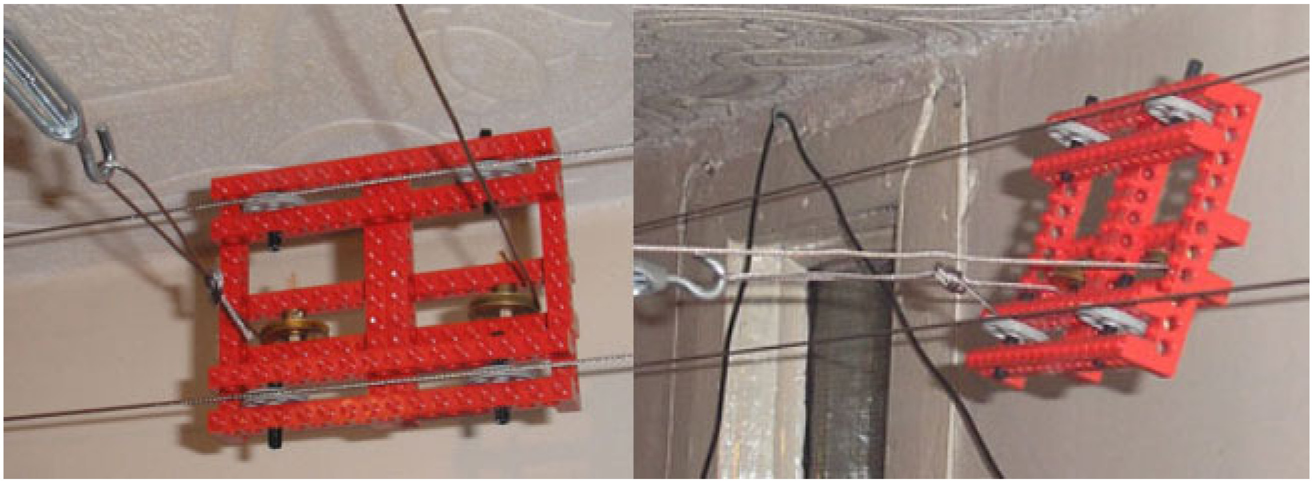
- Separate output / input test
- Clear all / set all function
- Counter function on inputs 1 and 2 with adjustable debounce (max 2KHz depends on total I/O load)
- Analog output set sliders
- Analog input bar-graph indication

Appendix D – Pictures of Track Based Approach

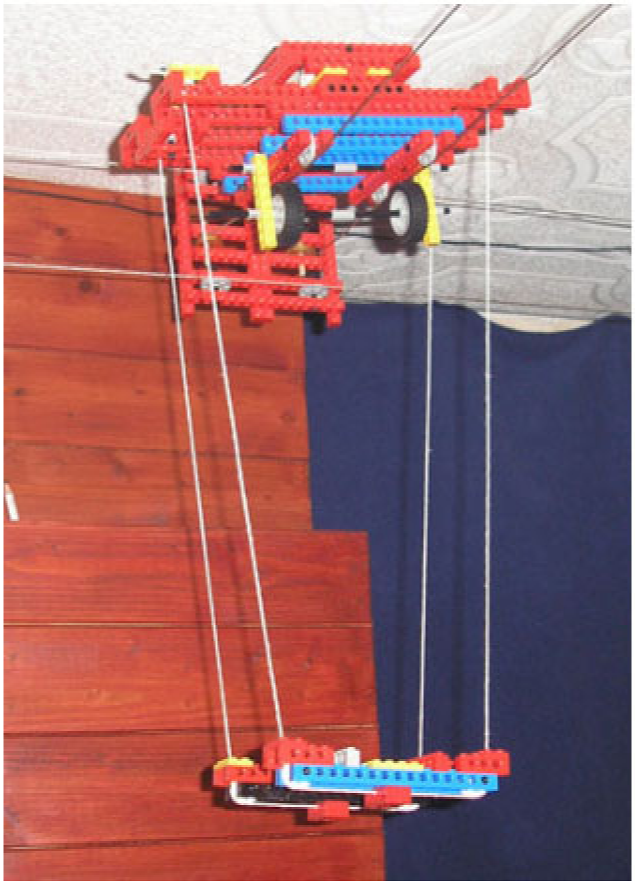


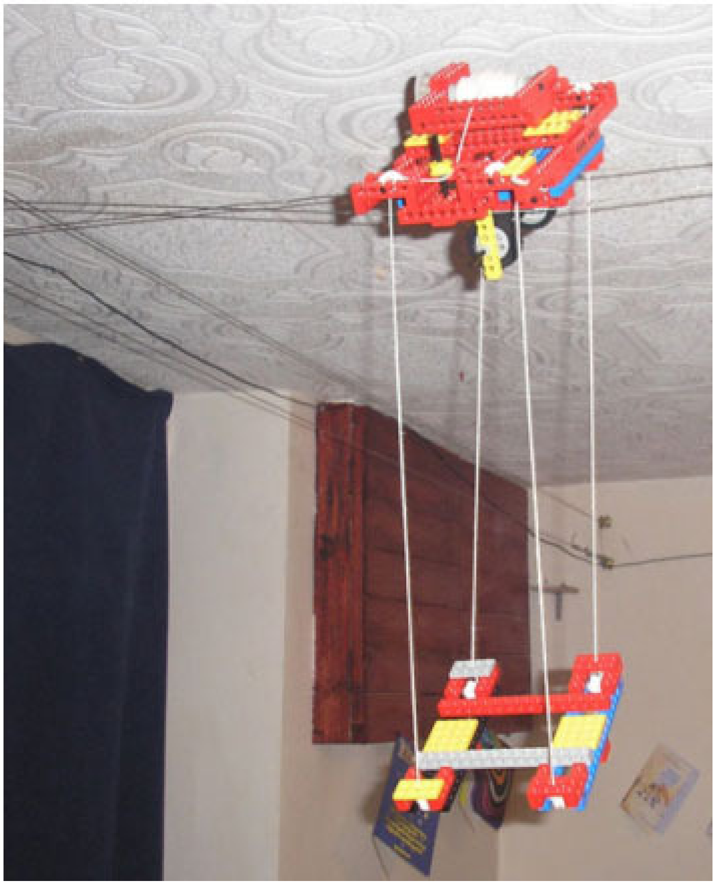


Appendix E – Pictures of Crane Based Approach



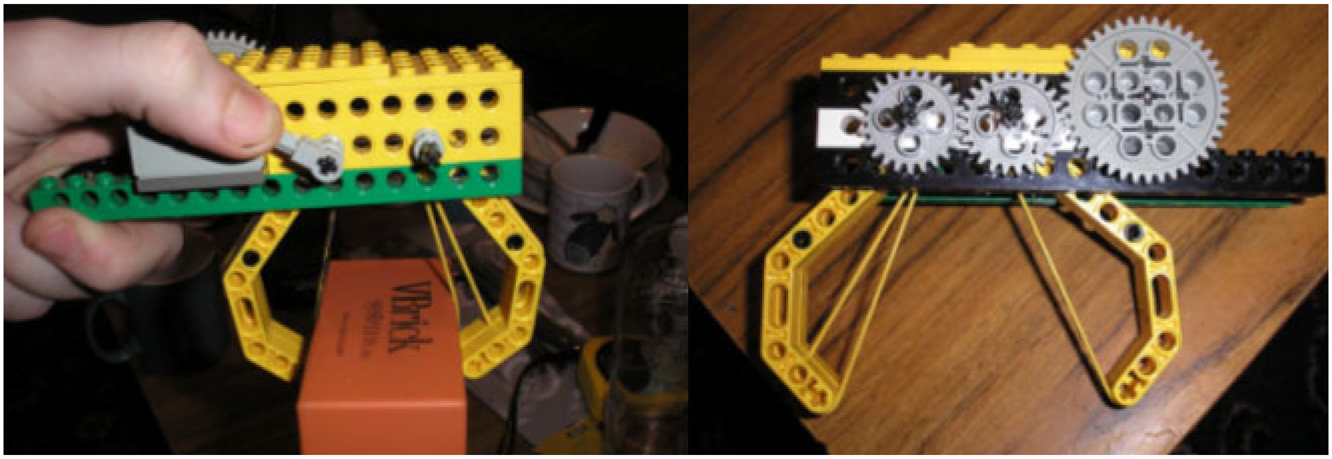
Above, the side cars can suffer tilting if the load is not balanced.





The first design of the top car was top heavy, which caused the crane to tilt which is unacceptable

This is the first prototype of the grabbing mechanism which could be used to hold the bricks



Appendix F – Project Plan

	Task Description	Earliest Possible Start	Task Length	Fortnight 1	Fortnight 2	Fortnight 3	Fortnight 4	Fortnight 5	Fortnight 6	Fortnight 7	Fortnight 8	Fortnight 9	Fortnight 10
1	Research existing devices	1	2 weeks	🟦									
2	Research brick laying	1	2 weeks	🟦									
3	Research power sources	1	2 weeks	🟦									
4	Research robotic arms	1	2 weeks	🟦									
5	Write target specification	2	2 weeks		🟦								
6	Design car	2	1 week		🟦	🟦							
7	Design Robotic arm	2	2 weeks		🟦	🟦							
8	Build car	3	1 week			🟦							
9	Program car to follow rail	4	1 week				🟥	🟥	🟥				
10	Build or purchase arm	3	2 weeks			🟦	🟦						
11	Make bricks	2	1 week		🟩	🟩	🟩	🟥					
12	Program to stack bricks	5	4 weeks					🟨	🟥	🟥			
13	Test machine	4	4 weeks				🟩	🟩	🟩	🟩	🟥	🟥	
14	Make poster	1	2 weeks	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩
15	Complete final report	1	2 weeks	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩

Key

🟥 Holding project back
🟨 Urgent, Not holding project back
🟩 Not Urgent
🟦 Complete

Appendix G – Letter to obtain free bricks

Hi. I am a cybernetics student at the University of Bradford (UK) in my final year.
I believe you could help me and I could help promote your company. For my final year project, I am making a prototype of a wall building robot.

I will require 100 bricks to allow my robot to build a wall running along a track.

At the following URL, you can see the arm picking up a VBrick - which happens to be ideal for what I require.
http://retford.kicks-ass.net/public/WallBuilder/robot.mov. If this URL is broken, advise me and I will supply the video by other means.

The cybernetics project open day is a popular event where many visitors come to view the final year projects. Many companies also attend recruiting for staff. Some of the best projects win prizes and have been put in the newspapers.

If you could supply me with 100 of your promotional bricks, they would certainly get a lot of attention, and would benefit me greatly also.
You are welcome to attend the open day, in the final week of the spring semester.

Please let me know if you are interested ASAP as I will consider other possible 'sponsors' who may consider supplying the foam advertising bricks if i fail to receive a response,

Thankyou for your time

Nick Hardman
University of Bradford
Dept of Cybernetics

Appendix H – Sample Movement file for robotic arm

```
16
1
9
1000
"No.1",0,253,5,0,0
"No.2",0,253,9,0,0
"No.3",0,253,0,0,0
"No.4",0,253,0,0,0
"No.5",0,253,145,0,0
"No.6",0,253,145,5,0
"No.7",0,253,127,0,0
"No.8",0,253,127,0,0
"No.9",0,253,127,0,0
"No.10",0,253,127,0,0
"No.11",0,253,127,0,0
"No.12",0,253,127,0,0
"No.13",0,253,127,0,0
"No.14",0,253,127,0,0
"No.15",0,253,127,0,0
"No.16",0,253,127,0,0
11
101,109,243,145,5,5,229,127,127,127,127,127,127,127,127,127,
186,109,243,158,36,36,225,127,127,127,127,127,127,127,127,127,
194,112,253,142,63,63,229,127,127,127,127,127,127,127,127,127,
0,112,253,142,63,63,229,127,127,127,127,127,127,127,127,127,
0,107,253,142,14,14,229,127,127,127,127,127,127,127,127,127,
0,107,253,156,38,38,113,127,127,127,127,127,127,127,127,127,
0,107,253,146,53,53,114,127,127,127,127,127,127,127,127,127,
0,107,253,141,57,57,114,127,127,127,127,127,127,127,127,127,
186,107,253,141,57,57,114,127,127,127,127,127,127,127,127,127,
186,107,253,141,0,0,107,127,127,127,127,127,127,127,127,127,
186,107,253,0,0,0,180,127,127,127,127,127,127,127,127,127,
""
0
"EmptyMuisc"
0
```

Appendix I – Record of Finances

Item	Cost	Postage	Miles	Fuel	Total
Robot Arm	£112.00	£0.00	0	£0.00	£112.00
Axel	£0.69	£0.00	5	£1.50	£2.19
Bands	£0.99	£0.00	3	£0.90	£1.89
Robot Power Supply	£14.99	£0.00	3	£0.90	£15.89
PTM Switches	£1.38	£0.00	0	£0.00	£1.38
4 Way adapter	£3.99	£0.00	0	£0.00	£3.99
Motor	£17.63	£1.76	0	£0.00	£19.39
H Bridge	£18.80	£1.76	0	£0.00	£20.56
Line Detector	£12.93	£1.76	0	£0.00	£14.69
Motor Mount	£6.99	£1.76	0	£0.00	£8.75
Callipers	£9.99	£0.00	2	£0.60	£10.59
Nails	£0.99	£0.00	5	£1.50	£2.49
USB Interface	£29.99	£0.00	5	£1.50	£31.49
Wires	£2.97	£0.00	0	£0.00	£2.97
Solder	£7.39	£0.00	0	£0.00	£7.39
Soldering Iron	£9.99	£0.00	5	£1.50	£11.49
Soldering Iron Mount	£4.99	£0.00	0	£0.00	£4.99
Small Vice	£2.99	£4.60	0	£0.00	£7.59
Hammer	£1.99	£0.00	2	£0.60	£2.59
USB Serial Lead	£5.60	£0.00	0	£0.00	£5.60
Wood	£0.00	£0.00	1	£0.30	£0.30
Bricks	£0.00	£0.00	0	£0.00	£0.00
Wheels	£0.00	£0.00	0	£0.00	£0.00
Siren	£0.00	£0.00	0	£0.00	£0.00
USB Extension	£2.99	£0.00	4	£1.20	£4.19
Screws	£0.00	£0.00	4	£1.20	£1.20
Lab Books	£1.98	£0.00	2	£0.60	£2.58
Pin nose pliers	£1.50	£0.00	4	£1.20	£2.70
Multi meter	£9.99	£0.00	0	£0.00	£9.99
Cable Ties	£1.99	£0.00	0	£0.00	£1.99
Cable * 30m	£4.99	£0.00	0	£0.00	£4.99
Bolts * 8	£23.92	£0.00	0	£0.00	£23.92
S Hooks	£1.50	£0.00	2	£0.60	£2.10
Tensioners Large	£4.98	£0.00	0	£0.00	£4.98
Tensioners Small	£1.49	£0.00	0	£0.00	£1.49
Lego	£0.00	£0.00	0	£0.00	£0.00
4 Way adapter	£2.49	£0.00	2	£0.60	£3.09
Blue Switch	£1.99	£0.00	2	£0.60	£2.59
Emergency Stop	£2.29	£0.00	0	£0.00	£2.29
Replacement Servos	£20.76	£1.99	0	£0.00	£22.75
Replacement Servos	£20.76	£1.99	0	£0.00	£22.75
Totals	£350.14	£13.64	51	£15.30	£401.83

Appendix J – Line Sensor Test Results

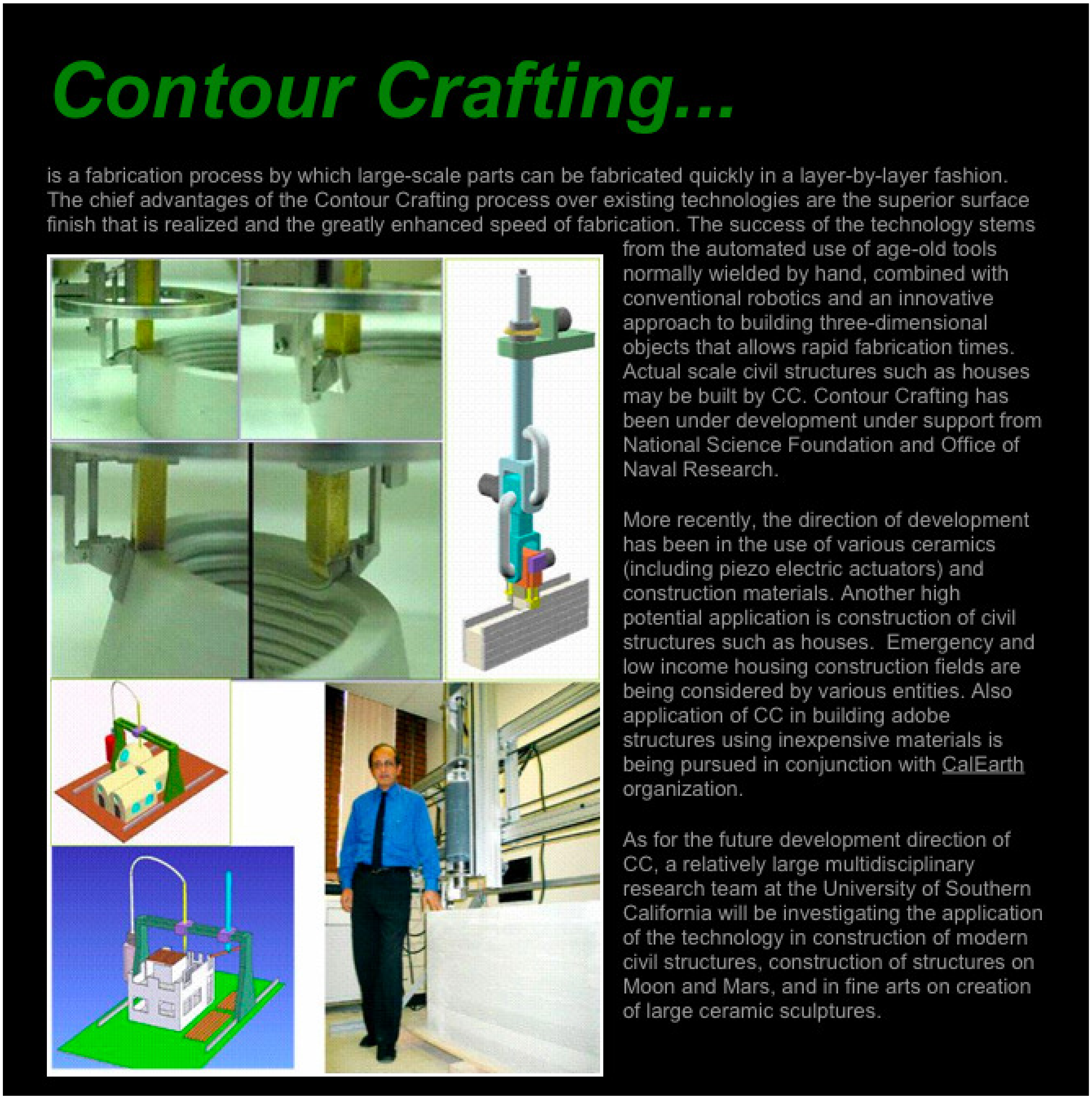
Drive Speed	On Time	Changes
120	100.00%	44
150	100.00%	8
200	100.00%	7
255	100.00%	5
Drive Speed	On Time	Changes
150	50.00%	8
150	33.00%	26
150	25.00%	73

Appendix K – USB Interface Measured Voltage

Comparison of Desired out against Measured Out

0	0
50	0.9
100	1.8
150	2.7
200	3.61
250	4.51
255	4.6

Appendix L – Contour Crafting



Contour Crafting...

is a fabrication process by which large-scale parts can be fabricated quickly in a layer-by-layer fashion. The chief advantages of the Contour Crafting process over existing technologies are the superior surface finish that is realized and the greatly enhanced speed of fabrication. The success of the technology stems from the automated use of age-old tools normally wielded by hand, combined with conventional robotics and an innovative approach to building three-dimensional objects that allows rapid fabrication times. Actual scale civil structures such as houses may be built by CC. Contour Crafting has been under development under support from National Science Foundation and Office of Naval Research.

More recently, the direction of development has been in the use of various ceramics (including piezo electric actuators) and construction materials. Another high potential application is construction of civil structures such as houses. Emergency and low income housing construction fields are being considered by various entities. Also application of CC in building adobe structures using inexpensive materials is being pursued in conjunction with CalEarth organization.

As for the future development direction of CC, a relatively large multidisciplinary research team at the University of Southern California will be investigating the application of the technology in construction of modern civil structures, construction of structures on Moon and Mars, and in fine arts on creation of large ceramic sculptures.

Appendix M – Source code for connecting to Robot

```
            'Close port if its allready open
            If MSComm1.PortOpen Then
                MSComm1.PortOpen = False
            End If
            'Input com port settings from manufacturer
            MSComm1.Settings = "9600,N,8,1" ' ²¨ÌØÂÊ=9600, Ã»ÓÐÐ£Ñé, 8Î»Êý¾Ý, 1
bitÍ£Ö¹Î»¡£
            MSComm1.CommPort = RobotComPort
            'Connect
            MSComm1.PortOpen = True
            RoboticArmStatus.Text = "Conntected"
```

Appendix N – Source code for connecting to Controller Board

```
    Private Declare Function OpenDevice Lib "k8055d.dll" (ByVal CardAddress As
Integer) As Integer
    Private Declare Sub CloseDevice Lib "k8055d.dll" ()
    Private Declare Sub OutputAnalogChannel Lib "k8055d.dll" (ByVal Channel As
Integer, ByVal Data As Integer)
    Private Declare Sub ClearDigitalChannel Lib "k8055d.dll" (ByVal Channel As
Integer)
    Private Declare Sub SetDigitalChannel Lib "k8055d.dll" (ByVal Channel As
Integer)
    Private Declare Function ReadDigitalChannel Lib "k8055d.dll" (ByVal Channel As
Integer) As Boolean
```

Appendix O – ABLE Source Code

```
Module SharedFunctions
    'This structure will hold the motor values of a fixed position
    Structure RobotPosition
        Dim Name As String
        Dim Motor1 As Integer
        Dim Motor2 As Integer
        Dim Motor3 As Integer
        Dim Motor4 As Integer
        Dim Motor5 As Integer
        Dim Motor6 As Integer
        Dim Motor7 As Integer
    End Structure
    Structure Motor
        Dim LowerLimit As Integer
        Dim UpperLimit As Integer
        Dim CurrentPosition As Integer
        Dim IntendedPosition As Integer
        Dim MovementsPerSample As Integer
    End Structure
    Structure FuzzyInputTable
        Dim LowerMembership As Single
        Dim MiddleMembership As Single
        Dim UpperMembership As Single
    End Structure
End Module


Option Explicit On
Imports System.Math 'Includes the maths class required by the inverse kinematics
code
Public Class Form1
    'Declare functions which talk to the interface board
    Private Declare Function OpenDevice Lib "k8055d.dll" (ByVal CardAddress As
Integer) As Integer
    Private Declare Sub CloseDevice Lib "k8055d.dll" ()
    Private Declare Sub OutputAnalogChannel Lib "k8055d.dll" (ByVal Channel As
Integer, ByVal Data As Integer)
    Private Declare Sub ClearDigitalChannel Lib "k8055d.dll" (ByVal Channel As
Integer)
    Private Declare Sub SetDigitalChannel Lib "k8055d.dll" (ByVal Channel As
Integer)
    Private Declare Function ReadDigitalChannel Lib "k8055d.dll" (ByVal Channel As
Integer) As Boolean
    Dim BricksOnRobot As Integer
    Dim WaitingTime As Integer
    Dim MaxWallHeight As Integer
    Dim RobotMoving As String 'Will be set to Stopped or name of a position it is
moving to
    Dim RobotDriving As String 'Will be set to Stopped or name of a position it is
driving to
    'Declare global variables
    Dim MaxWallLength As Integer 'Maximum number of bricks long the machine can
build
    Dim BrickInterval As Integer 'the number of counts to move the distance of a
single brick
```

```
    Dim Origin As Integer 'the number of counts to move to the origin from the far
end
    Dim DriveDirection As String 'Specifies the direction the car should be moving
in
    Dim DriveDistance As Integer 'Specifies the distance the car has remaining to
move
    Dim CardComPort As Integer 'specifys which com port the card will use
    Dim RobotComPort As Integer 'specifies which com port the robot will use
    Dim FrontTouchSensor As Integer 'holds the value of the last reading
    Dim RearTouchSensor As Integer 'holds the value of the last reading
    Dim LineSensor As Integer 'holds the value of the last reading
    Dim AlarmOn As Boolean 'Specifies if the alarm in enabled
    Dim SafeDecision As Boolean 'Is used to stop the robot if an instruction is
supplied which can cause dammage
    Dim Motor1 As Motor 'A motor on the robot arm
    Dim Motor2 As Motor 'A motor on the robot arm
    Dim Motor3 As Motor 'A motor on the robot arm
    Dim Motor4 As Motor 'A motor on the robot arm
    Dim Motor5 As Motor 'A motor on the robot arm
    Dim Motor6 As Motor 'A motor on the robot arm
    Dim Motor7 As Motor 'A motor on the robot arm
    Dim BlueButtonPressed As Integer
    Dim Alternator As Integer 'To ensure the robot only moves 1 out of 4 time
sequences
    Dim AngleToPileA As Integer 'motor2
    Dim AngleToPileB As Integer 'motor2
    Dim AngleToWall As Integer 'motor2
    Dim TwistForPileA As Integer 'motor7
    Dim TwistForPileB As Integer 'motor 7
    Dim TwistForWall As Integer 'motor7

    Dim Floor1 As Integer 'height in cm of brick layer 1 off ground
    Dim Floor2 As Integer 'height in cm of brick layer 2 off ground
    Dim Floor3 As Integer 'height in cm of brick layer 3 off ground
    Dim Floor4 As Integer 'height in cm of brick layer 4 off ground
    Dim Floor5 As Integer 'height in cm of brick layer 5 off ground
    Dim Floor6 As Integer 'height in cm of brick layer 6 off ground
    Dim Brick1 As Integer 'height in cm of brick 1 off robot
    Dim Brick2 As Integer 'height in cm of brick 2 off robot
    Dim Brick3 As Integer 'height in cm of brick 3 off robot
    Dim HoldBrick As Integer 'position of gripper when holding brick
    Dim DropBrick As Integer 'position of gripper when dropping brick
    Dim WallDistance As Double 'distance from robot to wall in cm
    Dim PileADistance As Double 'distance from robot to pilea in cm
    Dim PileBDistance As Double 'distance from robot to pileb in cm
    'function to allow program to sleep whilst the robot grabs or drops a brick
    Declare Sub Sleep Lib "kernel32" (ByVal dwMilliseconds As Long)


    'This sub will set each of the motors intended position to a given value
    Public Sub AssignNewPositionNoKinematics(ByVal PositionName As String)

        If RobotMoving <> "Stopped" Then
            Pause(1)
            wait()
        End If

        RobotMoving = PositionName
        FeedbackBox.Text = RobotMoving
```

```
Dim m1, m2, m3, m4, m5, m6, m7 As Integer
Select Case PositionName
    Case "Drop1"
        m1 = Motor1.CurrentPosition
        m2 = AngleToWall
        m3 = 0
        m4 = 66
        m5 = 130
        m6 = m5
        m7 = TwistForWall
    Case "Drop2"
        m1 = Motor1.CurrentPosition
        m2 = AngleToWall
        m3 = 0
        m4 = 90
        m5 = 106
        m6 = m5
        m7 = TwistForWall
    Case "Drop3"
        m1 = Motor1.CurrentPosition
        m2 = AngleToWall
        m3 = 0
        m4 = 108
        m5 = 84
        m6 = m5
        m7 = TwistForWall
    Case "AboveWall"
        m1 = Motor1.CurrentPosition
        m2 = AngleToWall
        m3 = 21
        m4 = 165
        m5 = 33
        m6 = m5
        m7 = TwistForWall
    Case "AboveBricksA"
        m1 = Motor1.CurrentPosition
        m2 = AngleToPileA
        m3 = 21
        m4 = 165
        m5 = 27
        m6 = m5
        m7 = TwistForPileA
    Case "AboveBricksB"
        m1 = Motor1.CurrentPosition
        m2 = AngleToPileB
        m3 = 21
        m4 = 165
        m5 = 27
        m6 = m5
        m7 = TwistForPileB
    Case "BrickA1" 'pile a base
        m1 = Motor1.CurrentPosition
        m2 = AngleToPileA
        m3 = 21
        m4 = 145
        m5 = 89
        m6 = m5
        m7 = TwistForPileA
    Case "BrickA2" 'pile a brick 2
```

```
                m1 = Motor1.CurrentPosition
                m2 = AngleToPileA
                m3 = 21
                m4 = 165
                m5 = 69
                m6 = m5
                m7 = TwistForPileA
            Case "BrickB1" 'pile b base
                m1 = Motor1.CurrentPosition
                m2 = AngleToPileB
                m3 = 21
                m4 = 139
                m5 = 92
                m6 = m5
                m7 = TwistForPileB
            Case "BrickB2" 'pile b brick 2
                m1 = Motor1.CurrentPosition
                m2 = AngleToPileB
                m3 = 21
                m4 = 165
                m5 = 65
                m6 = m5
                m7 = TwistForPileB
            Case Else
                m1 = Motor1.CurrentPosition
                m2 = AngleToPileA
                m3 = 51
                m4 = 8
                m5 = 7
                m6 = 7
                m7 = TwistForPileA
        End Select

        Motor1.IntendedPosition = m1
        Motor2.IntendedPosition = m2
        Motor3.IntendedPosition = m3
        Motor4.IntendedPosition = m4
        Motor5.IntendedPosition = m5
        Motor6.IntendedPosition = m6
        Motor7.IntendedPosition = m7

        Motor1.MovementsPerSample = (Motor1.IntendedPosition -
Motor1.CurrentPosition) / 15
        Motor2.MovementsPerSample = (Motor2.IntendedPosition -
Motor2.CurrentPosition) / 15
        Motor3.MovementsPerSample = (Motor3.IntendedPosition -
Motor3.CurrentPosition) / 15
        Motor4.MovementsPerSample = (Motor4.IntendedPosition -
Motor4.CurrentPosition) / 15
        Motor5.MovementsPerSample = (Motor5.IntendedPosition -
Motor5.CurrentPosition) / 15
        Motor6.MovementsPerSample = (Motor6.IntendedPosition -
Motor6.CurrentPosition) / 15
        Motor7.MovementsPerSample = (Motor7.IntendedPosition -
Motor7.CurrentPosition) / 15
        RobotTimer.Enabled = True
        wait()
    End Sub
    'This sub will set each of the motors intended position to a given value
    Public Sub AssignNewPosition(ByVal PositionName As String)
        RobotMoving = PositionName
        FeedbackBox.Text = RobotMoving
        Dim m1, m2, m3, m4, m5, m6, m7 As Integer
        Select Case PositionName
            Case "Drop1"
                m1 = HoldBrick
                m2 = AngleToWall
                InverseKinematics(WallDistance, Floor1, m5, m4, m3)
                m6 = m5
                m7 = TwistForWall
            Case "Drop2"
                m1 = HoldBrick
                m2 = AngleToWall
                InverseKinematics(WallDistance, Floor2, m5, m4, m3)
                m6 = m5
                m7 = TwistForWall
            Case "Drop3"
                m1 = HoldBrick
                m2 = AngleToWall
                InverseKinematics(WallDistance, Floor3, m5, m4, m3)
                m6 = m5
                m7 = TwistForWall
                '            Case "Drop4"
                '                m1 = HoldBrick
                '                m2 = AngleToWall
                '                InverseKinematics(WallDistance, Floor4, m5, m4,
m3)
                '                m6 = m5
                '                m7 = TwistForWall
                '            Case "Drop5"
                '                m1 = HoldBrick
                '                m2 = AngleToWall
                '                InverseKinematics(WallDistance, Floor5, m5, m4,
m3)
                '                m6 = m5
                '                m7 = TwistForWall
                '            Case "Drop6"
                '                m1 = HoldBrick
                '                m2 = AngleToWall
                '                InverseKinematics(WallDistance, Floor6, m5, m4,
m3)
                '                m6 = m5
                '                m7 = TwistForWall
            Case "AboveWall"
                m1 = HoldBrick
                m2 = AngleToWall
                m3 = 51
                m4 = 8
                m5 = 7
                m6 = 7
                m7 = TwistForWall
            Case "AboveBricks"
                m1 = HoldBrick
                m2 = AngleToPileA
                m3 = 51
                m4 = 8
                m5 = 7
                m6 = 7
                m7 = TwistForPileA
            Case "BrickA1" 'pile a base
                m1 = HoldBrick
                m2 = AngleToPileA
                InverseKinematics(WallDistance, Brick1, m5, m4, m3)
                m6 = m5
                m7 = TwistForPileA
            Case "BrickA2" 'pile a brick 2
                m1 = HoldBrick
                m2 = AngleToPileA
                InverseKinematics(WallDistance, Brick2, m5, m4, m3)
                m6 = m5
                m7 = TwistForPileA
            Case "BrickA3" 'pile a brick 3
                m1 = HoldBrick
                m2 = AngleToPileA
                InverseKinematics(WallDistance, Brick3, m5, m4, m3)
                m6 = m5
                m7 = TwistForPileA
            Case "BrickB1" 'pile b base
                m1 = HoldBrick
                m2 = AngleToPileB
                InverseKinematics(WallDistance, Brick1, m5, m4, m3)
                m6 = m5
                m7 = TwistForPileB
            Case "BrickB2" 'pile b brick 2
                m1 = HoldBrick
                m2 = AngleToPileB
                InverseKinematics(WallDistance, Brick2, m5, m4, m3)
                m6 = m5
                m7 = TwistForPileB
            Case "BrickB3" 'pile b brick 3
                m1 = HoldBrick
                m2 = AngleToPileB
                InverseKinematics(WallDistance, Brick3, m5, m4, m3)
                m6 = m5
                m7 = TwistForPileB
            Case Else
                m1 = HoldBrick
                m2 = AngleToPileA
                m3 = 51
                m4 = 8
                m5 = 7
                m6 = 7
                m7 = TwistForPileA
        End Select

        Motor1.IntendedPosition = m1
        Motor2.IntendedPosition = m2
        Motor3.IntendedPosition = m3
        Motor4.IntendedPosition = m4
        Motor5.IntendedPosition = m5
        Motor6.IntendedPosition = m6
        Motor7.IntendedPosition = m7

        Motor1.MovementsPerSample = (Motor1.IntendedPosition -
Motor1.CurrentPosition) / 15
        Motor2.MovementsPerSample = (Motor2.IntendedPosition -
Motor2.CurrentPosition) / 15
```

```
        Motor3.MovementsPerSample = (Motor3.IntendedPosition -
Motor3.CurrentPosition) / 15
        Motor4.MovementsPerSample = (Motor4.IntendedPosition -
Motor4.CurrentPosition) / 15
        Motor5.MovementsPerSample = (Motor5.IntendedPosition -
Motor5.CurrentPosition) / 15
        Motor6.MovementsPerSample = (Motor6.IntendedPosition -
Motor6.CurrentPosition) / 15
        Motor7.MovementsPerSample = (Motor7.IntendedPosition -
Motor7.CurrentPosition) / 15
        RobotTimer.Enabled = True
        wait()
    End Sub
    'close the grabber
    Public Sub grab()
        Pause(2)
        wait()
        MoveRobot("Grab", HoldBrick)
        Motor1.CurrentPosition = HoldBrick
        BricksOnRobot = BricksOnRobot - 1

        'Sleep(1000)
    End Sub
    'open the grabber
    Public Sub drop()
        MoveRobot("Grab", DropBrick)
        Motor1.CurrentPosition = DropBrick

        Pause(1)
        wait()
        'Sleep(1000)
    End Sub
    'gets the next brick or asks the user to reload
    Public Sub GetNextBrick()
        'choose position
        Select Case BricksOnRobot
            Case 0
                MsgBox("Please ensure the robot is fully loaded then press the blue
button")
                If BricksOnRobot = 0 Then
                    Advice.Text = "Please reload the robot then press the blue
button"
                    While (BricksOnRobot = 0)
                        'allow windows to process other events whilst waiting
                        Advice.Text = "Please reload"
                        System.Windows.Forms.Application.DoEvents()
                    End While
                    Advice.Text = ""
                    GetNextBrick()
                    Exit Sub
                End If
            Case 1
                AssignNewPositionNoKinematics("AboveBricksB")
                wait()
                AssignNewPositionNoKinematics("BrickB2")
                wait()
                AssignNewPositionNoKinematics("BrickB1")
                wait()
                'grab brick
                grab()
                wait()
                AssignNewPositionNoKinematics("BrickB2")
                wait()
                AssignNewPositionNoKinematics("AboveBricksB")
            Case 2
                AssignNewPositionNoKinematics("AboveBricksB")
                wait()
                AssignNewPositionNoKinematics("BrickB2")
                'grab brick
                wait()
                grab()
                wait()
                AssignNewPositionNoKinematics("AboveBricksB")
            Case 3
                AssignNewPositionNoKinematics("AboveBricksA")
                wait()
                AssignNewPositionNoKinematics("BrickA2")
                wait()
                AssignNewPositionNoKinematics("BrickA1")
                wait()
                grab()
                wait()
                AssignNewPositionNoKinematics("BrickA2")
                wait()
                AssignNewPositionNoKinematics("AboveBricksA")
            Case 4
                AssignNewPositionNoKinematics("AboveBricksA")
                wait()
                AssignNewPositionNoKinematics("BrickA2")
                wait()
                grab()
                wait()
                AssignNewPositionNoKinematics("AboveBricksA")
            Case Else
                MsgBox("There are only 4 brick in the source pile")
        End Select

        'go to safe position
        AssignNewPositionNoKinematics("Safe")
    End Sub
    'lay a brick at the given height
    Public Sub LayBrick(ByVal Height As Integer)
        Select Case Height
            Case 0
                MsgBox("Cannot lay a brick at level zero")
            Case 1
                AssignNewPositionNoKinematics("Drop1")
            Case 2
                AssignNewPositionNoKinematics("Drop2")
            Case 3
                AssignNewPositionNoKinematics("Drop3")
            Case 4
                AssignNewPositionNoKinematics("Drop4")
            Case 5
                AssignNewPositionNoKinematics("Drop5")
            Case 6
                AssignNewPositionNoKinematics("Drop6")
            Case Else
```

```
                MsgBox("Cannot build higher than 6 bricks high")
        End Select
        drop()
        AssignNewPositionNoKinematics("Safe")
    End Sub
    'Outputs all the variables to the display
    Private Sub RefreshOutput()
        'display robots speed
        TextBox12.Text = Motor1.MovementsPerSample
        TextBox11.Text = Motor2.MovementsPerSample
        TextBox9.Text = Motor3.MovementsPerSample
        TextBox10.Text = Motor4.MovementsPerSample
        TextBox8.Text = Motor5.MovementsPerSample
        TextBox7.Text = Motor7.MovementsPerSample

        Label41.Text = BricksOnRobot
        Label35.Text = RobotDriving
        Label33.Text = RobotMoving
        If FrontTouchSensor = -1 Then
            FrontCollisionStatus.Text = "Clear"
        Else
            FrontCollisionStatus.Text = "Collision"
            crash()
        End If
        If RearTouchSensor = -1 Then
            RearCollisionStatus.Text = "Clear"
        Else

            RearCollisionStatus.Text = "Collision"
            crash()
        End If
        If LineSensor = -1 Then
            TrackSensorStatus.Text = "Black"
        Else
            TrackSensorStatus.Text = "White"
        End If
        TextBox1.Text = Motor1.CurrentPosition
        TextBox2.Text = Motor2.CurrentPosition
        TextBox3.Text = Motor3.CurrentPosition
        TextBox4.Text = Motor4.CurrentPosition
        TextBox5.Text = Motor5.CurrentPosition
        TextBox6.Text = Motor7.CurrentPosition
        CalculateFuzzyVoltage(DriveDistance * 10, ShowMotorSpeed.Text)
        MovementStatus.Text = DriveDirection
    End Sub
    'initialises settings and moves robot to origin
    Private Sub DoStartUp()
        BrickInterval = 5
        WaitingTime = 0
        MaxWallLength = 5
        MaxWallHeight = 3
        AngleToPileA = 58 'motor2
        AngleToPileB = 190
        AngleToWall = 2
        TwistForWall = 187
        TwistForPileA = 100
        TwistForPileB = 50
        Floor1 = -6
        Floor2 = -5
        Floor3 = -4
        Floor4 = -3
        Floor5 = -2
        Floor6 = -1
        Brick1 = 2
        Brick2 = 3
        Brick3 = 4
        PileADistance = 5
        PileBDistance = 5
        WallDistance = 5
        HoldBrick = 101
        DropBrick = 180
        Alternator = 0
        BricksOnRobot = 4
        CardComPort = 0
        RobotComPort = 4
        InitialiseMotors()

        ConnectToRobot(False)
        ConnectToControllerBoard(False)
        ConnectToRobot(True)
        ConnectToControllerBoard(True)
        RobotMoving = "Stopped"
        RobotDriving = "Stopped"

    End Sub
    Public Sub MoveToOrigin()
        'drive to origin
        RobotDriving = "Origin"
        MoveCar(100, "Reverse")

        'move to origin
        RobotMoving = "To Origin"
        AssignNewPositionNoKinematics("Safe")

        'wait till robot is in the origin position
        wait()
    End Sub
    'This sub routine will move all motors on the robot by one sample
    Private Sub RefreshRobotPosition()
        'If the robot is turning, sound the alarm
        If Motor7.CurrentPosition <> Motor7.IntendedPosition Then
            AlarmTimer.Enabled = True
        Else
            AlarmTimer.Enabled = False
            ClearDigitalChannel(7)

        End If

        'if the motor is at the intended position - ie it is within range of the
end then
        'stop make the current position equal the inteded and make movements zero
        'this applies to all motors

        Dim tempvalue As Integer
        Dim Range As Integer
        Range = 10

        'if motor 1 is within range of end point, make it equal end
        tempvalue = Motor1.CurrentPosition - Motor1.IntendedPosition
        If tempvalue < Range And tempvalue > -Range Then
            Motor1.CurrentPosition = Motor1.IntendedPosition
            Motor1.MovementsPerSample = 0
        Else
            Motor1.CurrentPosition = Motor1.CurrentPosition +
Motor1.MovementsPerSample
        End If

        'if motor 2 is within range of end point, make it equal end
        tempvalue = Motor2.CurrentPosition - Motor2.IntendedPosition
        If tempvalue < Range And tempvalue > -Range Then
            Motor2.CurrentPosition = Motor2.IntendedPosition
            Motor2.MovementsPerSample = 0
        Else
            Motor2.CurrentPosition = Motor2.CurrentPosition +
Motor2.MovementsPerSample
        End If
        'if motor 3 is within range of end point, make it equal end
        tempvalue = Motor3.CurrentPosition - Motor3.IntendedPosition
        If tempvalue < Range And tempvalue > -Range Then
            Motor3.CurrentPosition = Motor3.IntendedPosition
            Motor3.MovementsPerSample = 0
        Else
            Motor3.CurrentPosition = Motor3.CurrentPosition +
Motor3.MovementsPerSample
        End If
        'if motor 4 is within range of end point, make it equal end
        tempvalue = Motor4.CurrentPosition - Motor4.IntendedPosition
        If tempvalue < Range And tempvalue > -Range Then
            Motor4.CurrentPosition = Motor4.IntendedPosition
            Motor4.MovementsPerSample = 0
        Else
            Motor4.CurrentPosition = Motor4.CurrentPosition +
Motor4.MovementsPerSample
        End If
        'if motor 5 is within range of end point, make it equal end
        tempvalue = Motor5.CurrentPosition - Motor5.IntendedPosition
        If tempvalue < Range And tempvalue > -Range Then
            Motor5.CurrentPosition = Motor5.IntendedPosition
            Motor5.MovementsPerSample = 0
        Else
            Motor5.CurrentPosition = Motor5.CurrentPosition +
Motor5.MovementsPerSample
        End If
        'if motor 6 is within range of end point, make it equal end
        tempvalue = Motor6.CurrentPosition - Motor6.IntendedPosition
        If tempvalue < Range And tempvalue > -Range Then
            Motor6.CurrentPosition = Motor6.IntendedPosition
            Motor6.MovementsPerSample = 0
        Else
            Motor6.CurrentPosition = Motor6.CurrentPosition +
Motor6.MovementsPerSample
        End If
        'if motor 7 is within range of end point, make it equal end
        tempvalue = Motor7.CurrentPosition - Motor7.IntendedPosition
        If tempvalue < Range And tempvalue > -Range Then
            Motor7.CurrentPosition = Motor7.IntendedPosition
            Motor7.MovementsPerSample = 0
        Else
            Motor7.CurrentPosition = Motor7.CurrentPosition +
Motor7.MovementsPerSample
        End If


        'Move robot to current position
        MoveRobot("Grab", Motor1.CurrentPosition)
        MoveRobot("Rotate Wrist", Motor2.CurrentPosition)
        MoveRobot("Top Lift", Motor3.CurrentPosition)
        MoveRobot("Middle Lift", Motor4.CurrentPosition)
        MoveRobot("Extra Bottom Lift", Motor6.CurrentPosition)
        MoveRobot("Bottom Lift", Motor5.CurrentPosition)
        MoveRobot("Rotate Base", Motor7.CurrentPosition)

        If Motor1.MovementsPerSample + Motor2.MovementsPerSample +
Motor3.MovementsPerSample + Motor4.MovementsPerSample + Motor5.MovementsPerSample +
Motor6.MovementsPerSample + Motor7.MovementsPerSample = 0 Then
            Pause(2)
            RobotTimer.Enabled = False
        End If
    End Sub
    'Sets the waiting time to the value passed
    Sub Pause(ByVal WaitFor As Integer)
        WaitingTime = WaitFor
        RobotWaiting.Enabled = True
    End Sub
    'will build a wall of the given length and height
    Public Sub BuildWall(ByVal WallLength, ByVal WallHeight)
        Dim rows, columns As Integer
        'for each row of bricks (up to the top row)
        For rows = 1 To WallHeight
            If WallLength > 0 Then
                'Move the car to the origin
                MoveCar(100, "Reverse")
                MoveCar((BrickInterval * rows / 2), "Forwards")
                For columns = 1 To WallLength
                    Progress.Text = "Row " & rows & " Brick " & columns
                    Progress2.Text = WallLength - columns + ((WallHeight - rows) *
WallLength) & " Bricks remaining"
                    GetNextBrick()
                    LayBrick(rows)
                    MoveCar(BrickInterval, "Forwards")
                    Progress.Text = "Row " & rows & " Brick " & columns
                    Progress2.Text = WallLength - columns + ((WallHeight - rows) *
WallLength) & " Bricks remaining"
                Next
                WallLength = WallLength - 1
            End If
        Next
        MoveCar(100, "Reverse")
        MsgBox("Construction Complete")
    End Sub
    'will stick in an infinate loop until the robot is in the correct position and
so is the car
    Sub wait()
        While (RobotDriving <> "Stopped")
            Advice.Text = "Waiting for car to arrive at destination"
            'allow windows to process other events whilst waiting
            System.Windows.Forms.Application.DoEvents()
        End While
        Advice.Text = ""
        While (RobotMoving <> "Stopped")
            Advice.Text = "Waiting for robotic arm to reach its destination"
            'allow windows to process other events whilst waiting
            System.Windows.Forms.Application.DoEvents()
        End While

        Advice.Text = ""
        While (BricksOnRobot = 0)
            Advice.Text = "Waiting to reload"
            ReloadButton.BackColor = Color.Red
            ReloadButton.Text = "Please reload then click this button"
            'TakeSample()
            'allow windows to process other events whilst waiting
            System.Windows.Forms.Application.DoEvents()
        End While
        Advice.Text = ""
        'allow other processes to complete
        System.Windows.Forms.Application.DoEvents()
        If WaitingTime <> 0 Then wait()

    End Sub
    'Old Inverse kinematics experimental code. This is not used
    Sub InverseKinematics(ByVal DesiredDistance As Double, ByVal DesiredHeight As
Double, ByRef Bottom As Integer, ByRef Middle As Integer, ByRef Top As Integer)
<ref>
        'REFERENCE TO CODE SOURCE:MICROSOFT EXCEL SPREADSHEET
        'By Hoon Hong (hong@math.ncsu.edu)

        'It solves the problem posed in
"http://www.lynxmotion.com/ViewPage.aspx?ContentCode=wanted&CategoryID=54"


        'These measurements are in cm
        Dim RobotHeight, L1L2Avg, ToolLength, ServoRange, DesiredX, DesiredZ,
AngleFromGround As Double

        RobotHeight = 8.5 'Specifies the distance from the perspex base to the
shoulder joint
        L1L2Avg = 5.8 'Average length of link1 and link2
        ToolLength = 10.2 'Length from final pivot motor to tool tip
        DesiredX = DesiredDistance
        DesiredZ = DesiredHeight
        AngleFromGround = 270
        ServoRange = 255
        'declare variables
        Dim Xb, X, L3, P, L1, Zb, Z, H, Q, P1, P2, T1, T2, T3, Pi, S As Double

        'specify inputs
        L1 = L1L2Avg 'link 1 length
        H = RobotHeight 'robot height
        L3 = ToolLength 'tool length
        X = DesiredX 'desired distance
        Z = DesiredZ 'desired height
        P = AngleFromGround 'angle from ground
        Pi = 3.14159
        S = ServoRange
        'do calculations
        Xb = (X - L3 * Cos(P * Pi / 180)) / (2 * L1)
        Zb = (Z - H - L3 * Sin(P * Pi / 180)) / (2 * L1)
        Q = Sqrt(1 / (Xb ^ 2 + Zb ^ 2) - 1)
        P1 = Atan2(Zb + Q * Xb, Xb - Q * Zb) * 180 / Pi 'This was switched around
as the Atan2 function is reversed in vb
        P2 = Atan2(Zb - Q * Xb, Xb + Q * Zb) * 180 / Pi
        T1 = P1 - 90
        T2 = P2 - T1
        T3 = P - P2

        'normalise results
        T1 = Int(T1 Mod (127) + 127)
        T2 = Int(T2 Mod (127) + 127)
        T3 = Int(T3 Mod (127) + 127)

        If (Abs(T1) > S Or Abs(T2) > S Or Abs(T3) > S) Then MsgBox("Impossible due
to limited servo range!")
        If (Xb ^ 2 + Zb ^ 2 > 1) Then MsgBox("Impossible due to limited arm
length!")

        'output results
        MsgBox("Motor 1=" & T1 & " Motor2=" & T2 & " Motor 3=" & T3)

        'Test if the resulting desired position is within reach
        'If (Xb ^ 2 + Zb ^ 2 > 1) Then
        ' MsgBox("Impossible due to limited arm length!")

        'Test if the resulting desired position is within motor limits
        'ElseIf (Abs(T1) > ServoRange) Or (Abs(T2) > ServoRange) Or (Abs(T3) >
ServoRange) Then

        'MsgBox("Impossible due to limited servo range!")

        'If the move is valid move to that position
        'Else
        Bottom = T1
        Middle = T2
        Top = T3
        'End If
    End Sub
    'Takes the user input and validates it
    Public Sub ValidateInput(ByVal BricksLong As Integer, ByVal BricksHigh As
Integer)
        If BricksHigh > MaxWallHeight Then
            MsgBox("I cannot build a wall higher than " & MaxWallHeight & "
bricks")
        Else
            If BricksLong <= MaxWallLength Then
                If BricksHigh > BricksLong Then
                    MsgBox("The wall must be more bricks long than high")
                Else
                    BuildWall(BricksLong, BricksHigh)
                End If
            Else
                MsgBox("I cant build a wall that long. Please specify a value up to
" & MaxWallLength)
            End If
        End If
    End Sub
    'This will stop the robot from driving into a wall and move it to just outside
the sensors reach
    Private Sub crash()
        'Commented out to allow testing without the robot present
        If ControllerBoardStatus.Text = "Connected" Then
            If FrontTouchSensor = False Then
                MoveCar(25, "Reverse")
            ElseIf RearTouchSensor = False Then
                MoveCar(1, "Forwards")
                'MsgBox("Rear Crash")
            Else
                MoveCar(0, "Stop")
                RobotDriving = "Stopped"
            End If
        End If
    End Sub
    'Will update global variables and enable the motor timer
    Sub MoveCar(ByVal Distance As Integer, ByVal Direction As String)
        RobotDriving = Direction
        DriveDirection = Direction
        DriveDistance = Distance
        MotorSpeedTimer.Enabled = True
        wait()
    End Sub
    'Will power on or off the motor depending on how far is remaining
    Private Sub MotorSpeedTimer_Tick(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MotorSpeedTimer.Tick
        'Every nth call to this function will turn on the motor and decrease the
distance
        If Alternator = 0 Then
            'If robot is at target, stop and disable timer
            If DriveDistance < 1 Then
                Drive("Stop")
                DriveDistance = 0
                MotorSpeedTimer.Enabled = False
                RobotDriving = "Stopped"
            Else 'If distance is still remaining
                Drive(DriveDirection) 'Drive in the direction
                DriveDistance = DriveDistance - 1 'And decrease the distance
            End If
        Else 'Every call which isnt n, the robot motor will switch off
            Drive("Stop")
        End If
        'Increase the count
        Alternator = Alternator + 1
        'If the count > n then set alternator to 0
        If Alternator > 2 Then
            Alternator = 0
        End If
    End Sub
    'Set robot position to origin and put in movement constraints
    Sub InitialiseMotors()

        Motor1.CurrentPosition = DropBrick
        Motor1.IntendedPosition = DropBrick
        Motor1.LowerLimit = 100
        Motor1.UpperLimit = 253
        Motor1.MovementsPerSample = 0
        Motor2.CurrentPosition = AngleToPileA
        Motor2.IntendedPosition = AngleToPileA
        Motor2.LowerLimit = 0
        Motor2.UpperLimit = 253
        Motor2.MovementsPerSample = 0

        Motor3.CurrentPosition = 51
        Motor3.IntendedPosition = 51
        Motor3.LowerLimit = 0
        Motor3.UpperLimit = 253
        Motor3.MovementsPerSample = 0

        Motor4.CurrentPosition = 7
        Motor4.IntendedPosition = 7
        Motor4.LowerLimit = 0
        Motor4.UpperLimit = 253
        Motor4.MovementsPerSample = 0

        Motor5.CurrentPosition = 8
        Motor5.IntendedPosition = 8
        Motor5.LowerLimit = 0
        Motor5.UpperLimit = 253
        Motor5.MovementsPerSample = 0

        Motor6.CurrentPosition = 8
        Motor6.IntendedPosition = 8
        Motor6.LowerLimit = 0
        Motor6.UpperLimit = 253
        Motor6.MovementsPerSample = 0

        Motor7.CurrentPosition = TwistForPileA
        Motor7.IntendedPosition = TwistForPileA
        Motor7.LowerLimit = 0
        Motor7.UpperLimit = 253
        Motor7.MovementsPerSample = 0

    End Sub
    'Will turn on or off the motor which drives the car
    Private Sub Drive(ByVal Direction As String)
        'output the command to the user
        MovementStatus.Text = Direction
        Select Case Direction
            Case "Reverse"
                'Output sequence to drive backwards
                OutputAnalogChannel(1, 0)
                OutputAnalogChannel(2, 210)
                ClearDigitalChannel(8)
            Case "Forwards"
                'Output sequence to drive forwards
                OutputAnalogChannel(1, 210)
                OutputAnalogChannel(2, 0)
                ClearDigitalChannel(8)
            Case "Stop"
                'Turn off the motor
                SetDigitalChannel(8)
                OutputAnalogChannel(1, 0)
                OutputAnalogChannel(2, 0)
        End Select
    End Sub
    'When the alarm timer ticks, this sub will either turn on or off the buzzer
    Private Sub AlarmTimer_Tick(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles AlarmTimer.Tick
        If AlarmOn = True Then
            ClearDigitalChannel(7)
            AlarmOn = False
        Else

            SetDigitalChannel(7)
            AlarmOn = True
        End If
    End Sub
    'This sub routine will move the specified joint name to the specified position.
    Private Sub MoveRobot(ByVal JointName As String, ByVal Position As Integer)
        On Error GoTo Err_MoveRobot

        'declare byte for data to be stored
        Dim OutData(2) As Byte

        'Convert the motor name to the corresponding number
        Dim MotorNumber As Integer
        Select Case JointName
            Case "Rotate Base"
                MotorNumber = 6
            Case "Bottom Lift"
                MotorNumber = 5
            Case "Extra Bottom Lift"
                MotorNumber = 4
            Case "Middle Lift"
                MotorNumber = 3
            Case "Top Lift"
                MotorNumber = 2
            Case "Rotate Wrist"
                MotorNumber = 1
            Case "Grab"
                MotorNumber = 0
        End Select

        'check to see if the movement is within the constraints of the machine
        SafetyCheck(MotorNumber, Position)
        If SafeDecision = True Then
            'standard marker
            OutData(0) = &HFFS '  µÚÒ»Byte£º·¢ËÍÍ¬²½ÐÅºÅ (¹Ì¶¨Îª Ê®Áù½øÖÆFF) ¸ø
            'motor number
            OutData(1) = MotorNumber '  µÚ¶þByte£ºÑ¡ÔñËÅ·þÂí´ïÐòºÅ£¨0µ½16£©
            'position
            OutData(2) = CByte(Position) '
µÚÈýByte£ººÑËÅ·þÂí´ïµÄÎ»ÖÃÊýÖµ·¢ËÍµ½¿ØÖÆºå
            'output to com port
            MSComm1.Output = OutData 'VB6.CopyArray(OutData) '
ºÑÊý¾ÝÍ¨¹ý´®¿Ú·¢ËÍ³öÈ¥
        Else
            'warn the user about the bad command
            Advice.Text = "This is not a wise idea! You will dammage the robotic
arm! Please step away from the computer!"
        End If

Exit_MoveRobot:
        Exit Sub

Err_MoveRobot:
        If Err.Description = "Exception from HRESULT: 0x800A1F52" Then
            RoboticArmStatus.Text = "Not Connected"

        Else
            FeedbackBox.Text = Err.Description

        End If
        GoTo Exit_MoveRobot
    End Sub
    'This is used to ensure the robot cannot be moved into a position which will
cause damage
    Private Sub SafetyCheck(ByVal MotorNumber As Integer, ByVal IntendedPosition As
Integer)
        SafeDecision = False
        'MsgBox("Attempting to move " & MotorNumber & " to " & IntendedPosition)
        Select Case MotorNumber
            Case 0
                If IntendedPosition > Motor1.LowerLimit And IntendedPosition <
Motor1.UpperLimit Then
                    SafeDecision = True
                End If
            Case 1
                If IntendedPosition > Motor2.LowerLimit And IntendedPosition <
Motor2.UpperLimit Then
                    SafeDecision = True
                End If
            Case 2
                If IntendedPosition > Motor3.LowerLimit And IntendedPosition <
Motor3.UpperLimit Then
                    SafeDecision = True
                End If
            Case 3
                If IntendedPosition > Motor4.LowerLimit And IntendedPosition <
Motor4.UpperLimit Then
                    SafeDecision = True
                End If
            Case 4
                If IntendedPosition > Motor5.LowerLimit And IntendedPosition <
Motor5.UpperLimit Then
                    SafeDecision = True
                End If
            Case 5
                If IntendedPosition > Motor6.LowerLimit And IntendedPosition <
Motor6.UpperLimit Then
                    SafeDecision = True
                End If
            Case 6
                If IntendedPosition > Motor7.LowerLimit And IntendedPosition <
Motor7.UpperLimit Then
                    SafeDecision = True
                End If
        End Select
    End Sub
    'This will exit the program when the user clicks ok to confirm
    Private Sub ExitToolStripMenuItem_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles ExitToolStripMenuItem.Click
```

```
        If MsgBox("Are you sure you wish to exit the program", MsgBoxStyle.YesNo, "Exit Software") = MsgBoxResult.Yes Then
            End
        End If
    End Sub
    'This will call the take sample function every time the controller board timer ticks
    Private Sub ControllerBoardTimer_Tick(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles ControllerBoardTimer.Tick
        TakeSample()
    End Sub
    'Disconnects the robot arm and the controller board then re initialise
    Private Sub ResetToDefaultsToolStripMenuItem_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles ResetToDefaultsToolStripMenuItem.Click
        ConnectToRobot(False)
        ConnectToControllerBoard(False)
        DoStartUp()
    End Sub
    'Will connect to or dis connect from the robot
    Private Sub ConnectToRobot(ByVal Connection As Boolean)
        On Error GoTo Err_ConnectToRobot
        Advice.Text = ""

        'if a connection is requested
        If Connection = True Then
            RoboticArmStatus.Text = "Attempting to connect to robotic arm"
            'Close port if its allready open
            If MSComm1.PortOpen Then
                MSComm1.PortOpen = False
            End If
            'Input com port settings from manufacturer
            MSComm1.Settings = "9600,N,8,1" ' ²¨ÌØÂÊ=9600, Ã»ÓÐÐ£Ñé, 8Î»Êý¾Ý, 1 bitÍ£Ö¹Î»¡£
            MSComm1.CommPort = RobotComPort
            'Connect
            MSComm1.PortOpen = True
            RoboticArmStatus.Text = "Conntected"

        Else
            RoboticArmStatus.Text = "Attempting to disconnect from robotic arm"
            MSComm1.PortOpen = False
            RoboticArmStatus.Text = "Not Conntected"
        End If
Exit_ConnectToRobot:
        Exit Sub
Err_ConnectToRobot:
        'MsgBox("Error Connecting to Robot: " & Err.Description)
        Select Case Err.Description
            Case "Exception from HRESULT: 0x800A1F4C"
                FeedbackBox.Text = "Cannot Disconnect: Port not active"
                Advice.Text = "You should check the robotic arm has power, and a good connection to the computer"

            Case "Exception from HRESULT: 0x800A1F52"

                RoboticArmStatus.Text = "Not Connected"
                Advice.Text = "Connect the robot and re-establish the connection"
            Case "Exception from HRESULT: 0x800A1F42"
                Advice.Text = "You should check the robotic arm has power, and a
good connection to the computer"
                FeedbackBox.Text = "Cannot Connect: Port not active"
            Case Else
                FeedbackBox.Text = Err.Description
        End Select

        Resume Exit_ConnectToRobot
    End Sub
    'Will connect to or disconnect from the controller board
    Private Sub ConnectToControllerBoard(ByVal Connection As Boolean)
        On Error GoTo Err_ConnectToControllerBoard
        If Connection = True Then
            ControllerBoardStatus.Text = "Attempting to connect to Controller
Board"
            Dim Feedback As Integer
            Feedback = OpenDevice(CardComPort)
            Select Case Feedback
                Case 0, 1, 2, 3
                    ControllerBoardStatus.Text = "Connected"
                Case -1
                    ControllerBoardStatus.Text = "Not Connected"
                Case -2
                    ControllerBoardStatus.Text = "K8055E" + Mid(Str(CardComPort),
2, 1) + ".EXE not found"
            End Select
            If Feedback >= 0 Then ControllerBoardTimer.Enabled = True
        Else
            ControllerBoardStatus.Text = "Attempting to disconnect from Controller
Board"
            CloseDevice()
            ControllerBoardStatus.Text = "Not Connected"
        End If
Exit_ConnectToControllerBoard:
        Exit Sub
Err_ConnectToControllerBoard:
        MsgBox("Error Connectiong to Controller Board: " & Err.Description)
        Resume Exit_ConnectToControllerBoard
    End Sub
    'When connect to controller board is selected from the menu
    Private Sub ConnectToolStripMenuItem1_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles ConnectToolStripMenuItem1.Click
        ConnectToControllerBoard(True)
    End Sub
    'When disconnect from controller board is selected from the menu
    Private Sub DisconnectToolStripMenuItem1_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles DisconnectToolStripMenuItem1.Click
        ConnectToControllerBoard(False)
    End Sub
    'When connect to robot is selected from the menu
    Private Sub ConnectToolStripMenuItem_Click(ByVal sender As System.Object, ByVal
e As System.EventArgs) Handles ConnectToolStripMenuItem.Click
        ConnectToRobot(True)
    End Sub
    'When disconnect from robot is selected from the menu
    Private Sub DisconnectToolStripMenuItem_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles DisconnectToolStripMenuItem.Click
        ConnectToRobot(False)
    End Sub
    'Reads all inputs and saves them to variables
    Private Sub TakeSample()
        FrontTouchSensor = ReadDigitalChannel(3)
        RearTouchSensor = ReadDigitalChannel(4)
        LineSensor = ReadDigitalChannel(5)
        'channel 2 is emergency stop
        'BlueButtonPressed = ReadDigitalChannel(1)
        RefreshOutput()
    End Sub
    'This is the code which runs when the program loads
    Public Sub New()
        ' This call is required by the Windows Form Designer.
        InitializeComponent()
        ' Run my start up procedure to reset the robot to an origin
        DoStartUp()
    End Sub
    'Calculates the truth of a each statement
    Private Sub GetFuzzyValue(ByRef x As Integer, ByRef TemporaryFuzzyValue As
FuzzyInputTable)
        If x < 20 Then
            'Only Lower input is true
            TemporaryFuzzyValue.LowerMembership = 1 - 0.05 * x
            TemporaryFuzzyValue.MiddleMembership = 0
            TemporaryFuzzyValue.UpperMembership = 0
        ElseIf x < 40 Then
            'Lower input drops to 0 over period of 20 values
            'middle input rises to 1 over period of 20 values
            TemporaryFuzzyValue.LowerMembership = 1 - 0.05 * (x - 20)
            TemporaryFuzzyValue.MiddleMembership = 0.05 * (x - 20)
            TemporaryFuzzyValue.UpperMembership = 0
        ElseIf x < 60 Then
            'only the middle input is 1
            TemporaryFuzzyValue.LowerMembership = 0
            TemporaryFuzzyValue.MiddleMembership = 1
            TemporaryFuzzyValue.UpperMembership = 0

        ElseIf x < 80 Then
            'Middle input drops to 0 over period of 20 values
            'Upper input rises to 1 over period of 20 values
            TemporaryFuzzyValue.LowerMembership = 0
            TemporaryFuzzyValue.MiddleMembership = 1 - 0.05 * (x - 60)
            TemporaryFuzzyValue.UpperMembership = 0.05 * (x - 60)
        Else
            'only the upper input is 1
            TemporaryFuzzyValue.LowerMembership = 0
            TemporaryFuzzyValue.MiddleMembership = 0
            TemporaryFuzzyValue.UpperMembership = 1
        End If

    End Sub
    'calculates the output voltage dependent on distance remaining
    Private Sub CalculateFuzzyVoltage(ByVal DistanceRemaining As Integer, ByRef
OutputVoltage As Integer)
        Dim Speed As FuzzyInputTable
        Dim Distance As FuzzyInputTable
        GetFuzzyValue(DistanceRemaining, Speed)
        GetFuzzyValue(DistanceRemaining, Distance)
        SpdSlow.Text = Speed.LowerMembership
        SpdMed.Text = Speed.MiddleMembership
        SpdFast.Text = Speed.UpperMembership
        DistClose.Text = Distance.LowerMembership
        DistMed.Text = Distance.MiddleMembership
        DistFar.Text = Distance.UpperMembership
        Dim Numerator As Integer
        Dim Denominator As Integer
        Dim AreaSlow As Integer = 60 * Speed.LowerMembership
        Dim AreaMed As Integer = 40 * Speed.MiddleMembership
        Dim AreaFast As Integer = 60 * Speed.UpperMembership

        Numerator = AreaSlow * 1 + AreaMed * 50 + (AreaFast * 100)
        Denominator = 150
        'MsgBox(Numerator.ToString + " / " + Denominator.ToString + " Slow " +
AreaSlow.ToString + " Med " + AreaMed.ToString + " Fast " + AreaFast.ToString)

        If DistanceRemaining = 0 Then
            OutputVoltage = 0
        Else
            OutputVoltage = (Numerator / Denominator) ' - 1
        End If
    End Sub
    'Shows my carreer page on my website in a new window
    Private Sub MyCarreerToolStripMenuItem_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles MyCarreerToolStripMenuItem.Click
        WebBrowser.Show()
    End Sub
    'show the about window
    Private Sub AboutThisProgramToolStripMenuItem_Click(ByVal sender As
System.Object, ByVal e As System.EventArgs) Handles
AboutThisProgramToolStripMenuItem.Click
        About.Show()
    End Sub
    'show a message box with some instructions
    Private Sub HelpToolStripMenuItem_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles HelpToolStripMenuItem.Click
        MsgBox("To build a wall manually, " & Chr(13) & Chr(13) _
              & "      Press the buttons in the manual control box as needed" &
Chr(13) & Chr(13) _
              & "To build a wall automatically," & Chr(13) & Chr(13) _
              & "      Enter the wall dimensions into the automated build box and
click build")
    End Sub
    'every time the robot timer ticks, update the robot position
    Private Sub RobotTimer_Tick(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles RobotTimer.Tick
        RefreshRobotPosition()
    End Sub
    'Manual Control, Get Next Brick
    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
        AssignNewPositionNoKinematics("BrickA2")
    End Sub
    'Manual Control, Go to safe position
    Private Sub Button10_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button10.Click
        AssignNewPositionNoKinematics("Safe")
    End Sub
    'Manual Control, lay brick at specified height
    Private Sub Button3_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button3.Click
        drop()
    End Sub
    'reset the car to the origin position
    Private Sub Button2_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button2.Click
        MoveToOrigin()
    End Sub

    'Manual Control Button
    Private Sub ForwardsMotor_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles ForwardsMotor.Click
        MoveCar(InputBox("Enter Distance to travel forwards. Please note, this is
in intervals, not bricks.", , BrickInterval), "Forwards")
    End Sub

    'Manual Control Button
    Private Sub ReverseMotor_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles ReverseMotor.Click
        MoveCar(InputBox("Enter Distance to travel backwards. Please note, this is
in intervals, not bricks.", , BrickInterval), "Reverse")
    End Sub

    'Manual Control Button
    Private Sub StopMotor_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles StopMotor.Click
        MoveCar(0, "Stop")
    End Sub
    'This button resets the brick count on the robot to 4
    Private Sub ReloadButton_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles ReloadButton.Click
        BricksOnRobot = 4
        ReloadButton.Text = "I'm not empty! If you want to load me up anyway,
please click the button"
        ReloadButton.BackColor = Color.Empty
    End Sub
    'Take user build coordinates and validate it is within the constraints of the
machine
    Private Sub Button4_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button4.Click
        ValidateInput(TextBox13.Text, TextBox14.Text)
    End Sub
    'Decrease waiting time on each tick
    Private Sub RobotWaiting_Tick(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles RobotWaiting.Tick
        If WaitingTime > 0 Then
            WaitingTime = WaitingTime - 1
            RobotMoving = "Waiting"
        Else
            RobotMoving = "Stopped"
            RobotWaiting.Enabled = False
        End If
    End Sub

    'Manual Control Button
    Private Sub Button6_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button6.Click
        AssignNewPositionNoKinematics("BrickA1")
    End Sub

    'Manual Control Button
    Private Sub Button7_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button7.Click
        AssignNewPositionNoKinematics("BrickB2")
    End Sub

    'Manual Control Button
    Private Sub Button8_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button8.Click
        AssignNewPositionNoKinematics("BrickB1")
    End Sub

    'Manual Control Button
    Private Sub Button9_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button9.Click
        AssignNewPositionNoKinematics("AboveBricks")
    End Sub

    'Manual Control Button
    Private Sub Button5_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button5.Click
        grab()
    End Sub

    'Manual Control Button
    Private Sub Button11_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button11.Click
        AssignNewPositionNoKinematics("AboveWall")
    End Sub

    'Manual Control Button
    Private Sub Button12_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs)
        AssignNewPositionNoKinematics("Drop4")
    End Sub

    'Manual Control Button
    Private Sub Button17_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button17.Click
        AssignNewPositionNoKinematics("Safe")
    End Sub

    'Manual Control Button
    Private Sub Button16_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button16.Click
        AssignNewPositionNoKinematics("AboveWall")
    End Sub

    'Manual Control Button
    Private Sub Button15_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button15.Click
        AssignNewPositionNoKinematics("Drop1")
    End Sub

    'Manual Control Button
    Private Sub Button14_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button14.Click
```

```
        AssignNewPositionNoKinematics("Drop2")
    End Sub

    'Manual Control Button
    Private Sub Button13_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button13.Click
        AssignNewPositionNoKinematics("Drop3")
    End Sub
End Class
```