
Department of Computing Science



UNIVERSITY
of
GLASGOW

Department of Electronics
& Electrical Engineering

University of Glasgow

ESE TEAM PROJECT REPORT
LEVEL 3, 2005/2006

PCB Drilling System

by

Liam Maclsaac and Benjamin Marwick Johnstone

We hereby give our permission for this project to be shown to other University of Glasgow students and to be distributed in an electronic format. **Please note that you are under no obligation to sign this declaration, but doing so would help future students.**

Liam MacIsaac

Benjamin Marwick Johnstone

Abstract

This project implements a portable printed circuit board drilling machine which is capable of drilling a circuit board layout based on input from a CAD application. The original basis of our system was the Lego Mindstorms Robotics Invention Kit which provides a large selection of Lego bricks, gears, motors and sensors, as well as microprocessor-based programmable control units.

We chose to implement the system with a Java-based user interface on the PC, and added extra hardware to the Lego Mindstorms Kit in the form of microcontrollers, a USB interface kit and other control circuitry.

The project addresses the challenges of integrating all of these different hardware and software components with the resources of a two-man team and provides a large number of opportunities to increase our knowledge in both hardware and software engineering.

Contents

Education Use Consent	i
Abstract	ii
1 Introduction	1
1.1 Preliminaries	1
1.2 Background	2
1.3 Aims	2
1.4 Motivation	3
1.5 Overview	3
2 Project Specification	5
2.1 Risk Factors	5
2.2 Hardware Specification	5
2.3 Software Specification - Use Cases	6
2.3.1 Use Case 1 - Load File & Drill	6
2.3.1.1 Rationale	6
2.3.1.2 Actors	6
2.3.1.3 Priority	6
2.3.1.4 Status	6
2.3.1.5 Preconditions	6
2.3.1.6 Used Use Cases	6
2.3.1.7 Flow of Events	7
2.3.2 Use Case 2 - Drill PCB	7
2.3.2.1 Rationale	7
2.3.2.2 Actors	7
2.3.2.3 Priority	7
2.3.2.4 Status	7
2.3.2.5 Preconditions	7
2.3.2.6 Flow of Events	7
2.4 Non Functional Requirements / System Constraints	7

2.5	Safety Requirements	8
3	System Overview	9
3.1	Software	9
3.2	Interface	9
3.3	Hardware	9
4	Hardware & Software Interface	10
4.1	Phidgets	10
4.2	Communications Chip	11
4.3	Phidgets to Comms Chip protocol	11
4.4	Comms Chip to Control Chip Protocol	12
5	Hardware	13
5.1	Control Circuit	13
5.1.1	Main Control Hardware	13
5.1.2	Motor Control	13
5.1.3	Sensors	14
5.1.3.1	Idea: Lego Rotation Sensors	14
5.1.3.2	Idea: Lego Light Sensors	14
5.1.3.3	Idea: Home-Made Light Sensors	16
5.1.3.4	Idea: Digital Position Sensors	16
5.1.3.5	Final Idea: Potentiometer and Analogue to Digital Converter	16
5.1.4	Drill Control	20
5.2	PIC Software	21
5.2.1	General	21
5.2.2	Comms Chip	21
5.2.3	Control Chip	23
5.3	Lego Hardware	26
5.3.1	Axes	26
5.3.2	Drill	27
5.4	Lego RCX Units	27
5.4.1	Choice of Programming Language	27
5.4.2	Communication Issues	28
5.4.3	Drill Up / Down Program	29
5.4.4	X/Y Movement Program	29
6	PC Software	32
6.1	Software Overview	32
6.2	Drilling File Format	33
6.2.1	The NC File Format	33

6.2.2	Problems With the File Format	33
6.3	File Co-ordinates	34
6.4	Defining our File Format	35
6.4.1	Model File	35
6.4.2	Limitations of Our Format	36
6.4.3	Other File Settings	36
6.5	Classes	36
6.5.1	The "excellon" package	36
6.5.1.1	The "tools" Package	37
6.5.2	The "drilling" Package	38
6.6	User Interface	38
6.6.1	The Menu System	40
6.6.1.1	The File Menu	40
6.6.1.2	The Drill Menu	40
6.6.2	The Help Menu	40
6.6.3	The Layout Editor	40
6.6.4	The Drilling Interface	41
6.7	Implementation Issues	42
6.7.1	Phidgets Interface Kit	42
6.7.2	File Precision	43
6.7.3	Slow UI response during drilling	43
6.8	Testing	43
6.8.1	Hardware Simulation	43
6.8.2	Testing of Parser	43
6.8.3	JUnit Tests	44
6.8.4	Large Scale Testing	44
7	Integration & System Testing	45
7.1	Integration	45
7.2	System Testing	45
8	Conclusion	46
8.1	Current System Status	46
8.2	Achievements of the Project	47
8.3	Main Problems Encountered	47
8.4	Further Development	48
	Bibliography	50

A Contributions	51
A.1 Liam MacIsaac	51
A.1.1 Summary of Project Log	51
A.1.2 Report Sections	52
A.2 Benjamin Marwick Johnstone	52
A.2.1 Summary of Project Log	52
A.2.2 Report Sections	53
B Acknowledgements	54
C Software Testing Data and Results	55
C.1 Testing NC Files	55
C.1.1 invalid.tap	55
C.1.2 invalid2.tap	55
C.1.3 empty.tap	55
C.1.4 test.tap	56
C.2 JUnit Tests	56
C.2.1 <i>DrillCalibration</i> Class Testing	57
C.2.1.1 Rationale	57
C.2.1.2 Test Method	58
C.2.1.3 Results	58
C.2.2 <i>PointConverter</i> Class Testing	58
C.2.2.1 Rationale	58
C.2.2.2 Test Methods	59
C.2.2.3 Results	59
C.3 Large File Test	60
C.3.1 File Handling	60
C.3.2 Output Generation	60
D Glossary	64
E Related Documentation	66
E.1 NQC Reference Card	66
E.2 Communications Protocols	69

Chapter 1

Introduction

Printed circuit boards are used in virtually every modern-day electronics application. For production standard boards, all aspects of board construction are carried out by machine, based on a design created in a computer aided design package. This type of construction, however, is only economic for boards which are to be produced in large quantities. In prototype design, education and other small-scale electronics, all aspects of board construction are carried out by hand. One of the tasks involved in this process is drilling all of the holes in the board which are required to hold the components. In the production level process, this task is carried out using a drilling device known as a CNC (Computerised Numerically Controlled) machine, which drills holes based on the input from an NC (Numerical Control) drilling file created by the CAD application. Small-scale board construction could be simplified by using this NC file along with a portable drilling machine to automate the drilling process. This project will look at the design and implementation of a prototype of such a system to determine whether or not it would be a useful tool.

1.1 Preliminaries

Readers of this report will require a basic understanding of the following:

- Printed Circuit Boards
- Lego
- UML Use Case, Class and Activity diagrams
- Object Orientated Programming
- Interfacing of hardware and software
- Electronic circuits and components (including potential dividers, analogue to digital converters, microcontrollers and relays)

The BASIC language will be used to program the PIC microcontrollers and NQC (Not Quite C) will be used to program the Lego RCX units. Knowledge of any programming language is sufficient to understand these parts of the system.

1.2 Background

We have decided to implement all of our system's PC software in Java because we both have previous experience of using Java, and Java provides the system with the ability to run on many different platforms.

The hardware that will be used in this project is a combination of hardware that is readily available from the Computing Science and Electronics departments, and some additional hardware that we have acquired ourselves. We were supplied with a large selection of Lego Mindstorms hardware components which includes Lego bricks, gears, motors, cable and a number of RCX units - programmable devices that can be used to process input from various types of sensors and control the operation of Lego motors. RCX units also provide limited communication facilities through the use of an infrared link. The Lego equipment will be used to build the structure of the drilling machine, and the RCX units will be used in conjunction with motors to control the movement of the machine.

We were also supplied with a Phidgets interface kit, which provides us with the ability to connect eight digital inputs and outputs and 8 analogue inputs to a USB port. A Java programming API is also provided for the kit, which allows the easy integration of input and output functions with any software that we develop in Java. We decided to use this piece of hardware to act as an intermediate communication stage between the computer and drilling hardware, because the built in Java API eliminates the need to develop any software to communicate with the hardware via an Infrared or Serial link.

Due to the limited processing power of the Lego RCX units, we have decided to use PIC (Programmable Integrated Circuit) microcontrollers which provide slightly more processing power. The microcontrollers will be used to receive drill hole position data from the PC (via the Phidgets interface kit) and will use information from sensors on the drill axes to position the drill in the correct place and drill a hole. The Lego RCX controllers will only handle basic motor control operations.

We are using a portable hand-held PCB drill to perform the drilling in our system.

The Electronics Component Store in the Electronics Department will supply any extra electronic components that we require during the course of the project.

1.3 Aims

We aim to produce a prototype drilling machine which takes its input from the industry-standard NC drilling file format produced by most CAD applications. The machine will drill holes at the correct points on a printed circuit board, based on the input from the

file. We hope to integrate computer software, the Phidgets USB interface kit, PIC microcontrollers and a variety of different Lego Mindstorms hardware components to address the challenges involved in interfacing these different components. If the project implementation is completed with sufficient time, we also aim to carry out an evaluation of the prototype system involving electrical engineers who have experience in designing boards manually, to assess the usefulness of the system.

1.4 Motivation

One of the main factors that influenced the choice of topic for this project was the small team size. The project topic had to be manageable between two people. We were also eager to tackle a project that would present opportunities to integrate a number of different hardware and software systems to develop our skills in hardware engineering, software engineering and system integration. The chosen topic is interesting in that it addresses these issues by integrating a number of systems including Java software on the PC, Lego RCX microcontrollers and PIC microcontrollers. There are a number of hardware issues such as interfacing, communication and timing and software issues such as file formats and language choice, which must be considered. The final outcome of the project will be an interesting test of whether or not it is possible to implement a drilling machine of this type which is easy to use, affordable and accurate.

1.5 Overview

2. Project Specification

This chapter provides the technical specification of the project, detailing exactly which functions should be performed by the hardware and software, and the performance and accuracy characteristics of the system.

3. System Overview

This chapter provides an illustration of the way in which all components of the system will be connected together to create the final system.

4. Hardware & Software Interface

This chapter will discuss the issues involved in interfacing the various components of the system, and will detail the final decisions made about the choice of interfaces between each of the sections of the system.

5. Hardware

Illustrates the design, implementation and testing of the hardware components of the system.

6. **PC Software**

Details the design, implementation and component testing of all aspects of the software that will run on the PC.

7. **Integration & System Testing**

This chapter details the process of integrating all of the separate system components to produce the complete system, and any problems experienced during this process. Testing of the completed system is also addressed in this chapter.

8. **Conclusion**

Discusses the achievements of the project, problems experienced during the project and the general outcome of the project. This chapter also discusses future refinements or improvements which could be made to the system.

Chapter 2

Project Specification

2.1 Risk Factors

- Lego hardware may be inadequate to achieve the desired precision.
- Position sensors may be inaccurate or unreliable.
- Drill bits may break too often for the system to be useful.
- Developer Inexperience - This project is the first where either of the team has used the Lego Mindstorms kit, and therefore there may be inadequate time to learn about the hardware.
- Resources Risk - Due to the small size of the team, the project is at greater risk of failure due to team member illness.

2.2 Hardware Specification

The system must meet the following hardware requirements:

- Holes should be drilled on a printed circuit board, accurate to 1mm of the desired position.
- The system must be able to drill circuit boards of sizes up to at least 100x160mm (Eurocard size).
- The system should be able to turn off power to the drill during the drilling process to allow the drill bit to be changed.

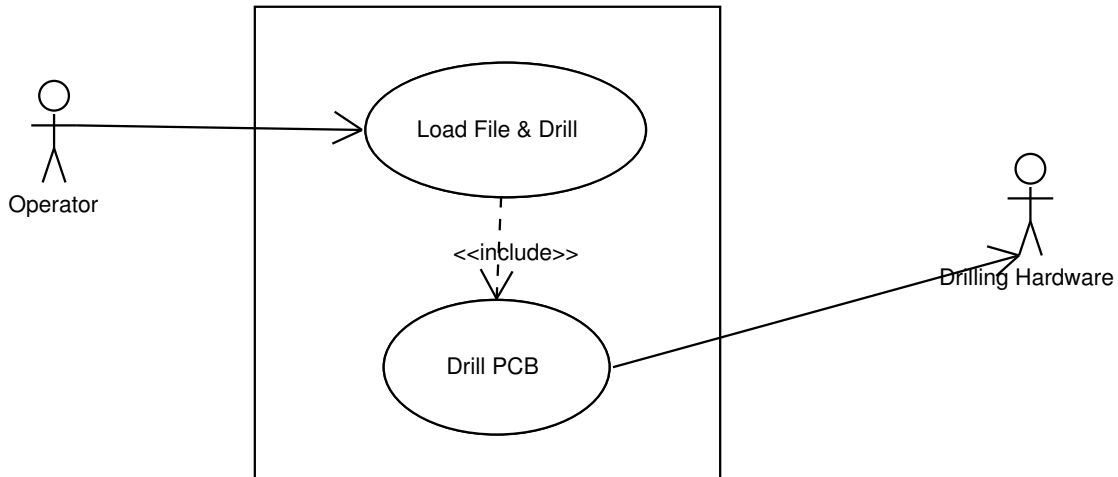


Figure 2.1: Use Case Diagram

2.3 Software Specification - Use Cases

2.3.1 Use Case 1 - Load File & Drill

2.3.1.1 Rationale

Allows the operator to load a drilling layout from a standard file and drill a PCB.

2.3.1.2 Actors

Operator

2.3.1.3 Priority

High - The system must have this function

2.3.1.4 Status

Not Started

2.3.1.5 Preconditions

An input drilling file in the required format has been previously created using a CAD application.

2.3.1.6 Used Use Cases

2. Drill PCB

2.3.1.7 Flow of Events

1. Select input file
2. Process input file and set up drilling layout within the system
3. <<Include Drill PCB Use Case>>

2.3.2 Use Case 2 - Drill PCB

2.3.2.1 Rationale

This use case will communicate with the hardware to drill a PCB using the currently-loaded drilling layout.

2.3.2.2 Actors

Drilling Hardware

2.3.2.3 Priority

Essential - The system must have this function

2.3.2.4 Status

Partially Complete

2.3.2.5 Preconditions

A drilling layout is ready in the system.

2.3.2.6 Flow of Events

1. Prompt the user to insert a correctly-sized PCB into the driller
2. For each hole size in the current layout
 - 2.1 Prompt the user to insert the correct drill bit
 - 2.2 For each hole of the current size
 - 2.2.1 Move drill to the correct position
 - 2.2.2 Drill hole
 - 2.3 End For
3. End For

2.4 Non Functional Requirements / System Constraints

- The system must be able to read the industry-standard NC drilling file format. Due to the fact that different CAD packages use different measurement units and root co-

ordinate values ¹, it may also be necessary to gain user input to specify some properties of the input file.

- The system must be able to operate at room temperature and under normal lighting conditions (must not be affected by direct sunlight).
- The system will be designed using the Lego Mindstorms Robotics invention kit, using any extra hardware and software that is required.

2.5 Safety Requirements

- The drill bit should be suitably guarded so that it cannot cause injury to the operator if it is broken during drilling.
- Measures should be taken to prevent the drill accidentally turning on while the bit is being changed.
- The system must incorporate an emergency stop feature.

Measures will be taken to protect against any other safety risks that become apparent during development.

¹The root co-ordinate is the co-ordinate of the bottom-left corner of the PCB

Chapter 3

System Overview

At the most simplistic level, our system has essentially three components, as shown in figure 3.1

3.1 Software

The software section of the project reads an NC drilling file, allows the user to preview it and set the accuracy and units of the co-ordinates and then performs the appropriate actions to calibrate the system and drill the holes. The software is discussed in more detail in chapter 6 .

3.2 Interface

The interface component takes Java calls for commands such as "drill hole at (x,y)" at one end and delivers these over a two-wire serial link at the other. It also carries data in the opposite direction for error-checking and calibration purposes. The software-hardware interface is discussed in detail in chapter 4 .

3.3 Hardware

The hardware component receives commands to drill holes, self test and send calibration data over the two-wire link and co-ordinates the motors, sensors and drill to carry out these commands. The hardware is discussed in detail in chapter 5.

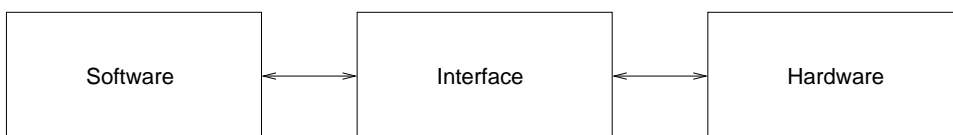


Figure 3.1: Block Diagram of Entire System

Chapter 4

Hardware & Software Interface

This section looks at the interface between the PC software and the control hardware.

A block diagram of the interface is shown in figure 4.1.

4.1 Phidgets

We were provided with a Phidgets USB interface board. This has eight digital inputs, eight analogue inputs, and eight digital outputs. This board was designed for connection to manually operated switches rather than a microcontroller, which presented some problems.

We found through experimentation that the Phidgets board cannot reliably detect a change on an input for 40ms after a previous change. This led us to the decision to use parallel data transfer over four wires to increase the transmission speed. With four wires for data, an acknowledge and a ready line, there are already six wires on the system. However, the ports on the Phidgets board are either input or output, not bi-directional, so we used six wires for each direction of data transfer, making twelve in total.

The control chip only has 16 I/O pins, so we were unable to allocate twelve of them solely for communication with the PC. This is why we chose to insert the Communications PIC, which communicates with the PC over 12 wires and communicates with the Control Chip over two wires. With hindsight, this scheme proved very hard to debug because of the long data path from PC to Control Chip, and we would have been far better to invest in a single microcontroller with more pins.

The inputs on Phidgets read logic 0 when unconnected and logic 1 when connected to ground. We interfaced the PIC to this by setting the pins to logic 0 then setting the data

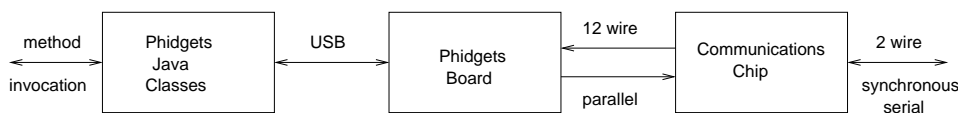


Figure 4.1: Block Diagram of the Interface between PC and Control Hardware

direction registers to input mode for logic 0 or output mode for logic 1. At one point we spent over six hours tracking down a fault caused by another routine setting the pin to logic 1, meaning 5V rather than ground was appearing on the pin when the DDR was later set to output mode.

On the software side, communication with the Phidgets board is described in section 6.5.2.

4.2 Communications Chip

This chip receives instructions from the PC and passes them to the Control chip, and vice versa. The program on this chip is discussed in section 5.2

A PIC16F628 microcontroller was used to interface between the 12-wire and two-wire data links. This flash programmable microcontroller has 16 I/O pins, an internal 4 MHz oscillator, 128 bytes of RAM and 2kwords (14 bit instruction words) of program memory. This chip was chosen mainly because I had plenty of previous experience of using both the F628 and its predecessor, the F84.

4.3 Phidgets to Comms Chip protocol

We established a layered communications protocol for communications from the Comms chip to Phidgets.

At the lowest layer, data is sent in four bit packets. The four bits are set up on the data lines, then the ready line is raised by the sender, and the acknowledge line is raised by the receiver when the data has been read.

At the next layer, a set of commands were defined and assigned numerical codes between 0 and 15. These were as follows:

Commands:

- Drill Hole
- Instruction Received
- Error
- Turn Drill Off
- Self Test
- Where Are You? (send current location of drill, for calibration purposes)
- I Am Here

Error Conditions:

- X Axis Error
- Y Axis Error
- Drill Error
- Drill Removed Error
- Unknown Error

Finally, we defined the exact sequence the commands were to be sent in. This is detailed in Appendix E.2.

4.4 Comms Chip to Control Chip Protocol

The PIC to PIC communications protocol is very similar to the one above, apart from the lowest layer. Data is sent synchronously over two wires, with one used for the actual data and one as a clock. The same pair of wires is used for communication in each direction. To simplify the programming of this, one extra command was added - an "Over" command. The over command is sent whenever one chip has finished sending and is ready to receive data from the other chip. We also made use of the over command to allow the Comms chip time to communicate an "instruction received" command to the PC without missing the "instruction completed" command from the Control chip, as shown below:

1. PC sends instruction to Comms chip
2. Comms chip sends instruction to Control chip, followed by "over"
3. Control chip sends "instruction received" to Comms chip, followed by "over"
4. Comms chip sends "instruction received" to PC
5. Comms chip sends "over" to Control chip
6. Control chip can now send "instruction completed" or other data to Comms chip when ready

Further details of the protocol are specified in Appendix E.2.

Chapter 5

Hardware

5.1 Control Circuit

This section looks at the circuitry used to control the drill.

A block diagram of the entire circuit is shown in figure 5.1.

5.1.1 Main Control Hardware

The purpose of this chip is to control the operation of motors, drills and sensors to carry out commands sent from the PC. Essentially there are four commands:

- Drill hole at (x,y)
- Turn drill off
- Return the current location of the drill (used for calibrating)
- Perform a hardware test

This chip is a PIC16F628. The program it is running is discussed in section 5.2

5.1.2 Motor Control

Due to the difficulties of interfacing with the RCX, it was decided to cut the RCX out completely and control the axis motors directly with relays. The circuit used is shown in figure 5.2. A ULN2803A Darlington array allows the PIC to drive two relays per motor, a DPDT to control polarity and hence direction, and an SPDT to turn the motor on and off. Note that when in the "off" state, the terminals of the motor are in fact shorted together, which gives a much shorter stopping time than leaving them unconnected.

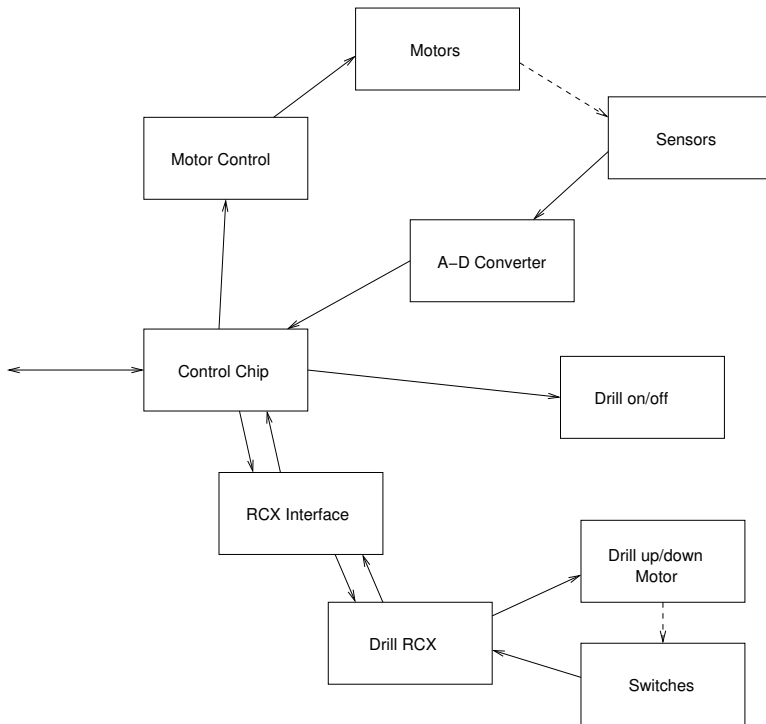


Figure 5.1: Block Diagram of Circuit

5.1.3 Sensors

5.1.3.1 Idea: Lego Rotation Sensors

The lego system comes with rotation sensors that attach directly to the RCX control units. However these have a resolution of 45° , so to attain any degree of accuracy this would have to be combined with a large gear ratio. This potential increase in accuracy would be lost again by the slippage in such a large train of gears. Whilst these sensors were readily available, we estimated a maximum possible resolution of $\pm 5\text{mm}$. Therefore the decision was taken not to use these sensors.

5.1.3.2 Idea: Lego Light Sensors

The lego system also comes with Light sensors : a combination of an LED light source and a light detector that is triggered when an object passes close to the light source. Previous projects have successfully used these to detect black lines at two inch spacings. On discussion of the matter with one of these teams, we were informed that the sensors are greatly affected by ambient light. Again we estimated that the maximum possible resolution we could achieve would be around $\pm 5\text{mm}$, so despite the fact that these sensors were readily available to us, the decision was taken not to use them in the project.

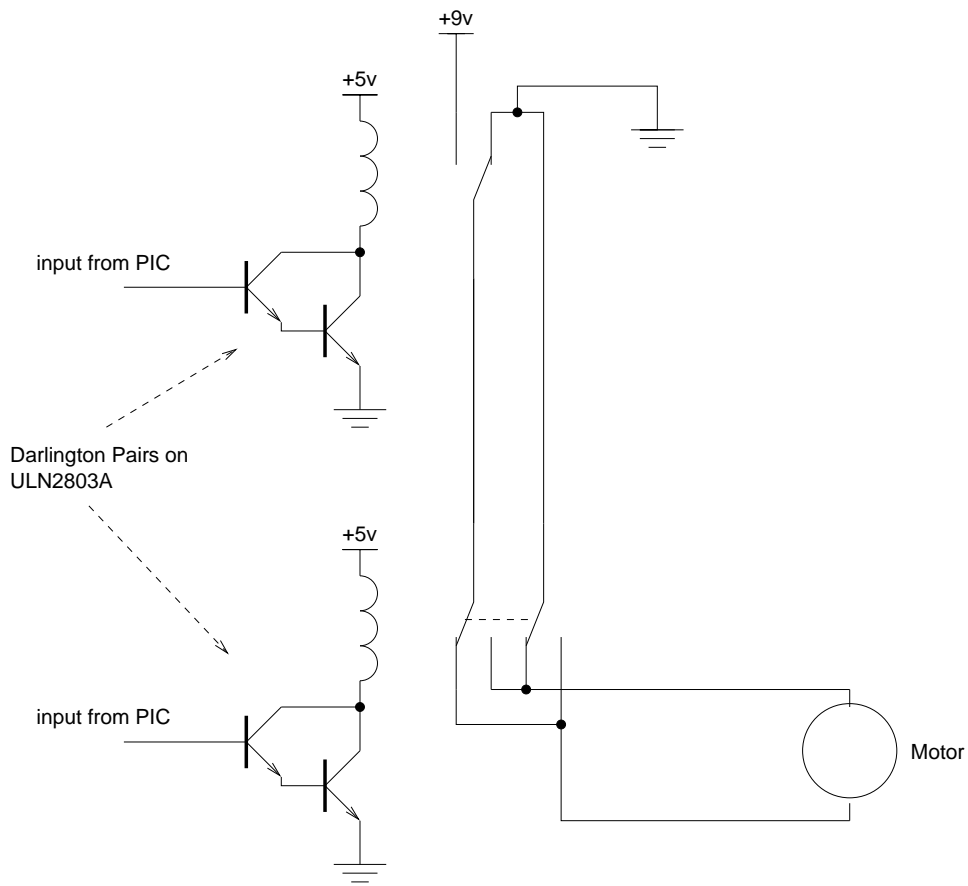


Figure 5.2: Motor Control Circuit

5.1.3.3 Idea: Home-Made Light Sensors

To counteract the problems of ambient light, we prototyped a system involving a pair of light sensors. A large difference in the signals from the two sensors would show that a nearby object was covering one of the sensors, and information as to which sensor was covered could be obtained by the sign of the output signal. However, the system would not be affected by ambient light levels as this would affect the magnitude of the signal from each sensor by the same amount, ensuring that the difference in the signals was not affected.

To create light sources to trigger the sensors, we explored various ideas involving fibre optic cables, before deciding on 2mm LEDs from Maplin. These LEDs would be in two rows staggered by 1mm, essentially creating a row of light sources with centres on a 1mm spacing. By using the pair of sensors with an $(n+0.5)$ mm spacing, half millimetre resolution could be achieved.

The number of LEDs could be halved by using two pairs of light sensors, or halved again by using four pairs of sensors.

Calculations then showed that even with an optimal balance of sensors and leds, this idea would cost around 40 pounds to implement for the project. This is before we had even considered problems such as controlling which of the hundreds of LEDs would be turned on at any one time.

Due to the massive costs and implementation time, this idea was abandoned.

5.1.3.4 Idea: Digital Position Sensors

As we had a budget of 100 pounds, and our home-made sensor system was going to cost at least 40 pounds, we started to seriously consider spending similarly large amounts of money on commercially available sensors.

Many digital callipers are available with serial communication ports to extract the length information that is normally displayed on screen. Accuracy is normally around 10 microns. At around 20 pounds for a six inch model, this would have cost around the same as the previous idea, with far greater accuracy and significantly easier implementation.

A version with mounting holes is also available. Designed for precisely the purpose we needed - position sensing of axes in any mechanical system - these sell for around twice the price of a similarly sized pair of digital callipers.

5.1.3.5 Final Idea: Potentiometer and Analogue to Digital Converter

By this point we realised that a simple solution would be needed if we were to get the project working on time. The decision was taken to use sliding linear potentiometers, read by an analogue to digital converter.

Our original specification required axes that covered 160 mm in one direction and 100mm in the other (Eurocard size). The largest potentiometers available were 100mm in length, and a 60mm model was also available. To allow for overshoot we decided to use two of

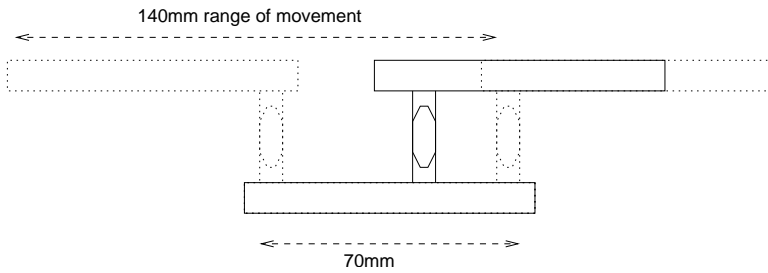
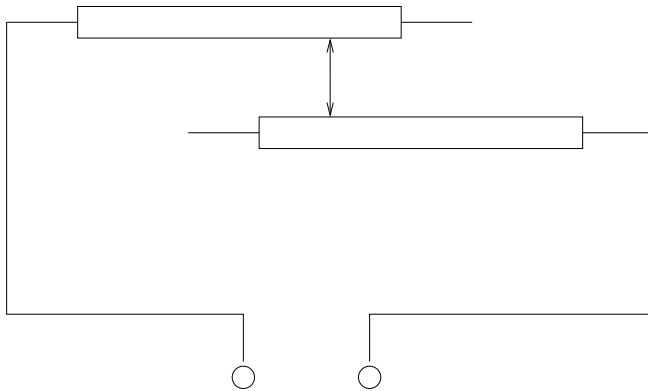


Figure 5.3: Coupling two potentiometers to double the range of movement



Resistance here is proportional to relative position of the two potentiometer bodies. It is not affected by the position of the (physically and electrically connected) wipers.

Figure 5.4: Connecting the two potentiometers electrically

the 100mm model on the long axis and two of the 60mm model on the shorter axis. These would be physically connected as shown in figure 5.3.

The original plan was to connect the potentiometers electrically as shown in figure 5.4. This would give a resistance directly proportional to the displacement of the bodies of the two potentiometers and not affected by the position of the centre section.

Unfortunately this plan did not give very accurate results due to a 2% difference in the end to end resistance of the two potentiometers. Instead it was decided to read a voltage from each potentiometer individually and then sum the corresponding lengths in software.

An additional problem was encountered with the potentiometers for the longer axis, which contrary to the description on the Farnell website have a resistance that varies logarithmically with displacement. A graph of resistance against displacement is shown in figure 5.5.

A near-linear result can be obtained from $2^{(reading)}$, as shown in figure 5.6. However, the complexity of raising two to the power of a twelve bit number with only sixteen bit integer arithmetic is prohibitive. Instead, it was decided to convert the value from each potentiometer to a displacement by breaking the graph into sixteen sections, treating each as linear, and using a lookup table to find the necessary offset and multiplicand.

For the analogue to digital converter, we chose the MCP3208 from Microchip. Operating

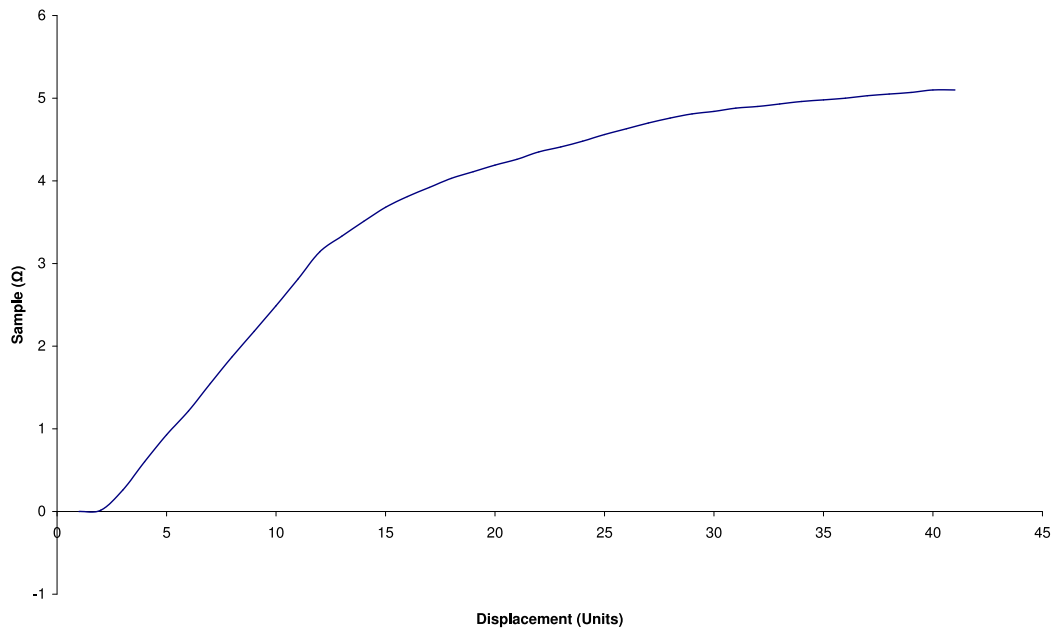


Figure 5.5: 100mm Potentiometers: Resistance versus Displacement

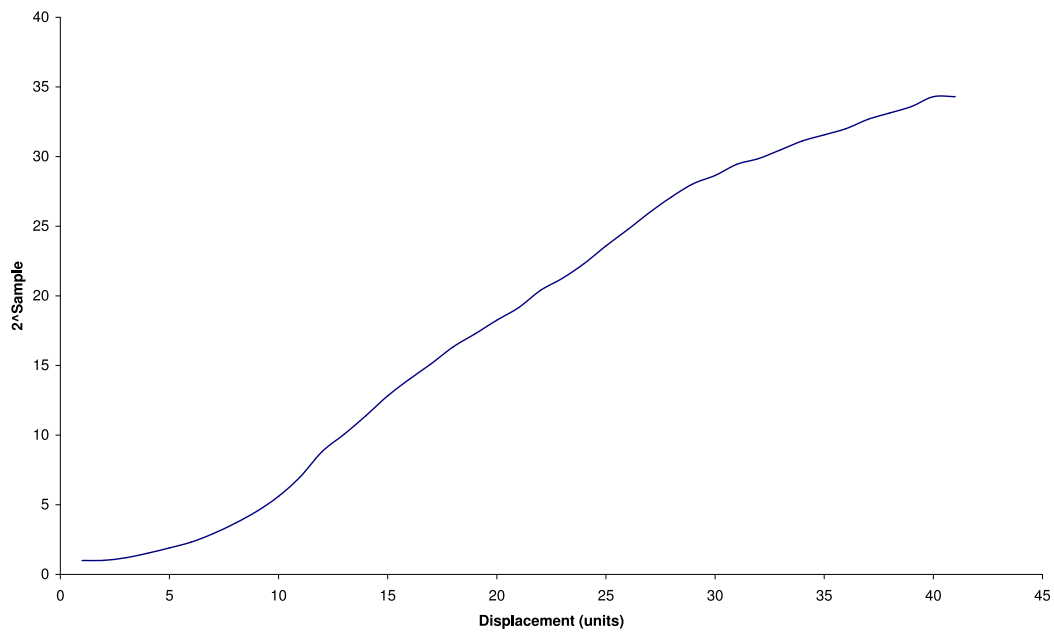


Figure 5.6: 100mm Potentiometers: $2^{\text{Resistance}}$ versus Displacement

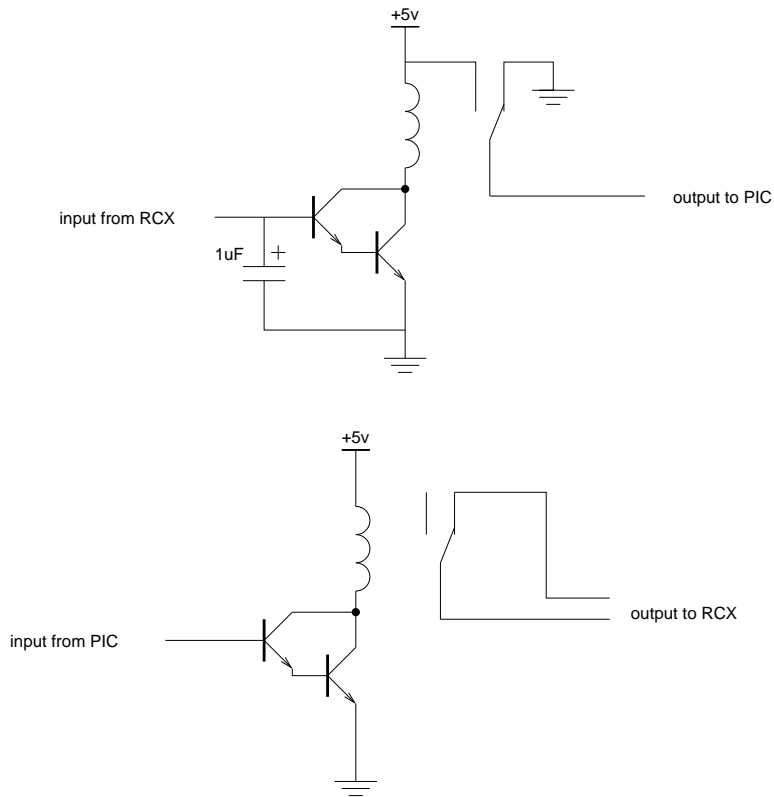


Figure 5.7: Interfacing a PIC to an RCX

on the successive approximation principle, it offers 12 bit readings from 8 channels, at a rate of up to 20KHz. An SPI-compatible serial link is used to clock data out of the device. Perhaps more importantly however, I had prior experience of interfacing the 3208 with the PIC microcontroller.

5.1.4 Drill Control

Originally, a lego RCX was used to control the up/down movement of the drill to drill a hole. While interfacing the PIC to an RCX is messy, it was judged to be simpler to keep the existing system than to replace it.

The designers of the RCX have employed a number of tricks to allow many types of sensor to be connected with only two wires. Consequently, we abandoned investigating ways in which we could trigger the RCX input with a voltage and instead replaced the standard Lego switch with a relay. This relay is controlled by the PIC through the same ULN2803A used to operate the motor control relays.

Receiving output from the RCX was even more complicated. The RCX has three output levels: high, medium and low. These all provide an output somewhere between 6 and 8 volts, using a square wave of varying duty cycles to control the power delivered to the output device (known as pulse width modulation, or PWM). We used a capacitor to smooth this wave,

which then triggers a relay through a separate ULN2803A. Finally, the relay delivers either 0 or 5 volts to the PIC.

5.2 PIC Software

There are two PIC microcontrollers used in this project. This section looks at the software running on each of them.

5.2.1 General

The two PICs in the project were programmed using the PicBasic Pro compiler from Micro-Engineering Labs. A weekend was spent trying to find a C compiler that would work with the PICF628, or at least some other PIC chip compatible with the available programmer. Unfortunately none of the free compilers were particularly well documented, so it was decided that it would be simpler to program the PIC with an existing basic compiler. While fairly limited in that arguments have to be passed to and from functions by global variables, accepting only a maximum of eight letters in variable names, and lacking not only a switch statement but also an elseif, the language is certainly very simple to use and produces fairly readable code.

The chips were physically programmed using a Velleman programmer kit. This is connected to the PC through an RS232 serial port and comes with software to load a compiled program into a chip placed into the board. The programmer board is sold as a self-assembly kit, and several hours were spent trying to fix a perfectly-working board before it was discovered that the problem lay in its incompatibility with a USB-RS232 converter. Several further hours were spent later on by attempting to fix the board when it mysteriously stopped working, while the problem was in fact that two different PIC controllers had simultaneously stopped working, possibly due to problems with static.

5.2.2 Comms Chip

The purpose of this chip is to take in commands from the PC and pass them on to the control PIC, and vice versa, as described in section 4.2.

The specification for the software on this chip is as follows:

```
while(1) {  
  
    wait for command from PC  
  
    if (command is invalid) {
```

```
    return error message

} else {

    if (command is "drill_hole") {
        get co-ordinates
    }

    send command to control pic

    if (command is "drill_hole") {
        send co-ordinates to control pic
    }

    wait for "instruction_received"

    send "instruction_received" to PC

    wait for "instruction_completed" or error code

    send the above to PC

}

}
```

However, as a workaround to pic-pic communication problems that we did not have time to solve, we produced a modified version which works as follows.

```
while(1) {

    wait for command from PC

    if (command is invalid) {

        return error message

    } else {
```

```

    if (command is "drill_hole") {
        get co-ordinates
    }

    send command to control pic

    if (command is "drill_hole") {
        send co-ordinates to control pic
    }

    send "instruction_received" to PC

    wait for 20 seconds to allow control chip to complete task

    send "instruction_completed" to PC

}

}

```

5.2.3 Control Chip

The purpose of this chip is to receive commands from the communications chip and coordinate motors, drill and sensors to carry out the commands, as described in section 5.2.3.

The specification for the software on this chip is as follows:

```

while (1) {

    get command from comms pic

    if (command is "drill_hole") {
        get co-ordinates from comms pic
    }

    send "instruction_received" to comms pic

    switch (instruction) {

        case drill_hole:

```

```
if (current_x_position < x_co-ordinate) {

    move x axis until current position > co-ordinate

} else {

    move x axis until current position < co-ordinate

}

(repeat for y axis)

turn on drill relay

signal drill RCX

wait for RCX completed signal

case drill_off:

    turn off drill relay

case where_are_you:

    read sensors
    send "i_am_here" to comms pic
    send current co-ordinates to comms pic

case self_test:

    try to move each axis in both directions

}

if (above steps completed successfully) {

    send "instruction_completed" to comms pic

} else {
```

```
    send error code to comms pic  
}  
  
}
```

However, as a workaround to pic-pic communication problems that we did not have time to solve, we produced a cut-down version which works as follows.

```
while (1) {  
  
    get command from comms pic  
  
    switch (instruction) {  
  
        case drill_hole:  
  
            move y axis for 200ms, always in same direction  
            move x axis for 400ms in an alternating direction  
  
            turn on drill relay  
  
            signal drill RCX  
  
            wait for RCX completed signal  
  
        case drill_off:  
  
            turn off drill relay  
  
    }  
  
}
```

Because of the failure of our control pic to comms pic communications, debugging the analogue to digital converter in this circuit proved impossible. Instead we constructed a separate circuit as follows:

```

while (1) {

    for each of two channels:

        read a-d 16 times, sum and divide by 16.

        repeat the above step 16 times, summing results and then
        dividing by 16.

        (the above process allows the average of 256 12-bit readings
        to be calculated with only 16-bit integer arithmetic)

    add the two averages from above, display highest 8 bits on leds.

    wait 100 ms

}

```

5.3 Lego Hardware

5.3.1 Axes

We found a conveniently sized piece of chipboard covered in lego plates left over from a previous project which we used as a baseboard. Sawtooth tracks were placed along two opposite sides of the board, and a beam was constructed with toothed wheels to fit into these tracks. This forms the basis of the first axis. Toothed tracks were then placed along each side of this moving beam to allow another similar beam to run on top of it, forming the second axis.

The wheels on each side of the beam were then linked with axles to help prevent the beam twisting when in motion, and a motor was attached to this axle through a series of gears. The gear train was constructed experimentally, and then later increased to provide slower and more accurate movement with higher torque.

Sensors were attached with a combination of superglue and silicone bathroom sealant. A simple clamp was constructed to hold the PCB in place.

No major problems were encountered although finding some combination of holes at correct spacings to build a gear train is always challenging in lego, as any mechanically-minded five-year-old knows.



Figure 5.8: The Entire Lego System

5.3.2 Drill

A cheap dremel-type drill was encased in lego secured with cable ties. Vertical saw-tooth tracks were attached to this assembly, to mesh with a series of cogs in the supporting framework. Again, a high-ratio gear train was constructed experimentally until the drill moved at what appeared to be a reasonable speed. Unfortunately, the height at which the drill must be before it can be lifted out of its holder (for changing the drill bit) is several inches above the height at which the drill bit meets the circuit board. This means that the drill takes around fifteen seconds to move down and up to drill a hole. The decision was taken to raise the drill to this height every time to avoid having to break apart the entire drill system to fit an extra sensor, which we intended to add later if time permitted.

On testing the drill mechanism, it was found that the drill bit meeting the pcb would cause the drill assembly to lift off its tracks momentarily before the drill bit went through the board. This was resolved by placing several weights (some conveniently sized battery packs) on top of the drill assembly.

5.4 Lego RCX Units

5.4.1 Choice of Programming Language

There are a number of programming options available for the Lego RCX. The RCX was originally distributed with a graphical block-diagram based system for developing software. The default firmware for the RCX, which is distributed along with this editor, also supports



Figure 5.9: The Drill Module

programming of the controller in assembly.

Other methods for programming the RCX have evolved since the original distribution of the product. The BrickOS system provides new firmware for the RCX which allows it to be programmed in C or C++. There are numerous other firmware applications similar to BrickOS which provide APIs for other programming languages, including firmware distributions which function as Java Virtual Machines. All of these third-party firmware distributions have the advantage that they allow easy programming of the RCX in a familiar language, however they generally require a much larger section of the RCX memory than the Lego firmware. Those which support Java also have the disadvantage that they greatly reduce the processing power of the RCX.

We have chosen to use a programming language called NQC ("Not Quite C") which, as the name suggests, is very similar to the C programming language. NQC also has an advantage over other third-party programming systems in that its compiler produces native Lego code which can be run on the default RCX firmware.

5.4.2 Communication Issues

The RCX provides a small number of communication commands which can be used to transmit and receive serial data using the built-in infrared link. We were advised by groups who had previously used the Lego Mindstorms kit that this communication system was difficult to integrate with Java software on the PC, and was error-prone. We therefore decided to devise a new communication method whereby we attached relays to the RCX sensor inputs which allows the external hardware to provide input to the RCX, by simulating

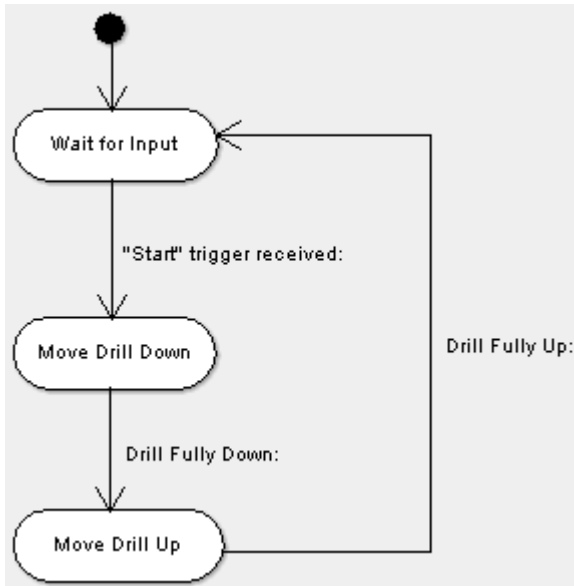


Figure 5.10: Drill Up/Down Program Activity Diagram

the signal generated by a Lego switch connected to the input. Motor output ports were used to generate signals to provide output to the external hardware.

5.4.3 Drill Up / Down Program

The purpose of this program is to control the up/down movement of the drill for drilling holes. The program should accept a "Start Drilling" trigger from the external hardware, move the drill axis motor in the "down" direction until a sensor indicates that the drill is in such a position that it has completed drilling the hole. The motor should then wait for a short time at this position before lifting the drill back up until a sensor indicates that the drill is in its "fully up" position. This process is illustrated in the activity diagram in Figure 5.10.

The starting trigger input will be provided by the external control hardware using a relay to simulate the action of pressing a lego switch sensor. The up and down detection will be carried out using Lego switch sensors. While the drill axis is moving, an extra motor port will be switched on to allow the external control hardware to detect when drilling has completed.

5.4.4 X/Y Movement Program

This program was initially developed to control the movement of the X and Y position axes, before the decision was made to build our own control circuitry for the axes. The program had to provide functionality to move an axis forwards and backwards at different speed

levels¹. It was decided that the axes should have a fast speed for moving large distances, a slow speed for finding approximate locations and a "single-step" speed for precise movement. Note that this program was designed so that one RCX had to be used for each axis controlled.

To allow the 3-input RCX to provide the required functionality, we decided to use all three inputs to provide the RCX with a 3-bit binary command, which allowed us to define up to seven distinct functions in the control program. The following functions were chosen:

- Stop movement
- Set direction to forward
- Set direction to reverse
- Move at fast speed
- Move at slow speed
- Move at single-step speed

The following pseudocode summarises the design of the program:

```
While True

    Wait until command is not "stop"

    Case Command
        When "forward"
            Set motor direction to forward
        When "reverse"
            Set motor direction to back
        When "fast"
            Set motor output to fastest speed
        When "slow"
            Set motor output to slow speed
        When "step"
            Turn on motor
            [Very Short Time Delay]
            Turn Off Motor
        When Other Cases
            Turn Off Motor
    End Case
```

¹Since this program was designed, a new gearing system has been implemented on both axes which has eliminated the need for control of the motor speed.

End While

Chapter 6

PC Software

6.1 Software Overview

The PC software for this system will consist of a simple desktop application which will allow users to import a PCB drilling layout from an industry standard drilling format file. The overall process of the application is described below and illustrated in the Activity Diagram (Figure 6.1).

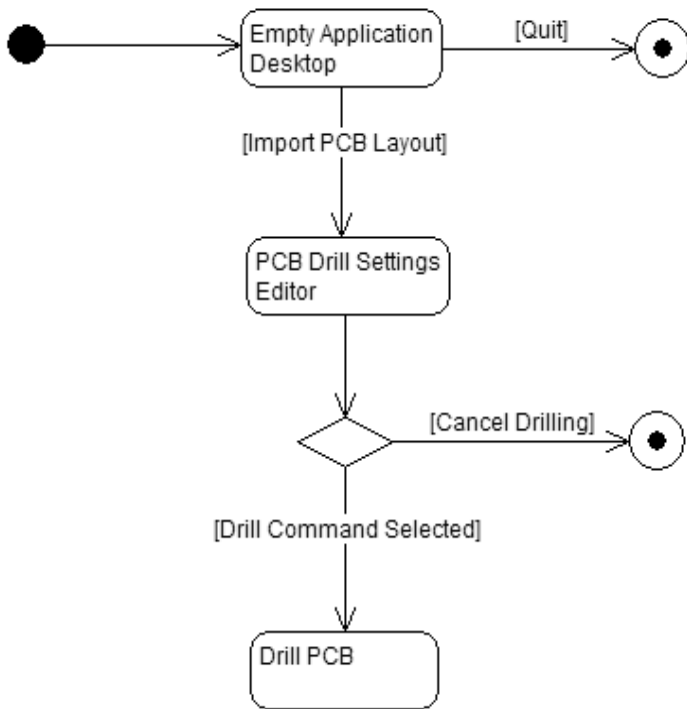


Figure 6.1: Activity Diagram

After loading the file, the user will then be presented with a PCB Drill Settings editor which will display the hole layout as it will appear on the board. This editor will also allow the user to adjust the (0,0) co-ordinate position and measurement units used in the file.

When the user has made the necessary adjustments to the drilling layout, the software will transmit the layout to the drilling hardware, notifying the user of any problems which require their attention during the process.

6.2 Drilling File Format

6.2.1 The NC File Format

It was decided that the NC drill file format (sometimes referred to as the "Excellon" format) would be used by our software to import drilling co-ordinates. This file format is an industry-standard text file format which is commonly used in both computer controlled and standalone drilling systems. Files of this format can be produced by almost all PCB design applications, and therefore tools for creating them are readily available to users who may be interested in using our system.

6.2.2 Problems With the File Format

One of the main disadvantages of using the industry-standard NC format is that the format is not strictly adhered to by PCB design applications. Figure 6.2 describes a typical "model" NC file, taken from www.pcbmilling.com.

M48	Indicate start of file
T1C0.028	Indicate that tool 1 has circumference of 0.028
T2C0.032	Indicate that tool 2 has circumference of 0.032
%	End of header
M47 File: test.th	Indicate file name
G05	Switch the drill on to drilling mode
M72	Specify that co-ordinates are imperial (M71 is metric)
T1	Select tool 1
X2.550Y1.550	Drill at specified X and Y
T2	Select tool 2
X2.050Y1.158	Drill at specified X and Y
X2.058Y1.558	Drill at specified X and Y
M30	End of file

Figure 6.2: Model NC File

If "real-life" NC files from PCB design applications are analysed, it can be seen that they differ greatly from the perfect model of an NC file.

Figure 6.3 shows a sample output from ORCAD. This file contains no header information before the start of the drilling data. The file also contains tool size, rotation speed and

feed speed on the same line as the tool selection command.

Figure 6.4 shows a sample output from EasyPC. The only header information contained in this file is the M48 ("Start of File") command. This file is also missing tool size definitions, referring to tools only by name.

These examples illustrate that our system must define its own subset of recognised NC commands to allow full compatibility with a wide variety of applications.

```

%
T1C0.021F200S100
X018750Y044750
X019663Y038000
T4C0.038F200S100
X018000Y015000
X018000Y016000
M30

```

Figure 6.3: Sample ORCAD NC File

```

M48
%
T004
X03703Y00453
X03703Y00553
T002
X04303Y02003
X04453Y02003
M30

```

Figure 6.4: Sample EasyPC NC File

6.3 File Co-ordinates

Before defining our recognised file format, there are some considerations about co-ordinate points that need to be taken into account:

- Definition of Points - Despite the fact that all of the examples shown have both X and Y co-ordinates defined for all points, it is possible to define only the X or Y co-ordinate if the other stays the same as the previous point. The code below defines points to be drilled at (100,200) and (100,300):

```

X100Y200
Y300

```

- Measurement Units - NC files can use either metric (mm) or imperial (inch/mil) measurement units for co-ordinates. As shown in the example code listings, this is not always defined.

- Precision - NC co-ordinates can use different precision levels. These levels define the number of digits before and after the decimal point in the co-ordinates. For example, if the co-ordinate 01245 is defined with 2.3 precision, then its actual value is 1.245.
- Zero Suppression - Zero suppression may be used to reduce the size of the NC file. For example, in the case where 2.3 precision with leading zero suppression is used, a co-ordinate specified in the file as 245 will be interpreted as 00245 which will provide a numerical value of 0.245. Trailing zero suppression may also be used.
- Board Offset - Some design applications do not automatically position the board's bottom left corner at the (0,0) co-ordinate.

6.4 Defining our File Format

6.4.1 Model File

Taking into account all of the factors mentioned in sections 6.2.2 and 6.3, we have decided that our application will recognise files of the following format:

```
[Header Information]
%
T1C0.02F200S100
X123Y456
Y457
X100
T2
X50Y40
X30Y20
M30
```

The following points should be noted about our recognised file format:

- All header information before the initial % symbol will be ignored.
- Tool definitions must contain T[Name]. Adding circumference information is optional. Feed speed (F) and rotation speed (S) tags can be included, but will not be read.
- Co-ordinates should be defined in the format X[Number]Y[Number] but can be defined as X[Number] or Y[Number] to change only one co-ordinate from the previous value. Co-ordinates should not include decimal points.
- The file must end with the M30 tag.

6.4.2 Limitations of Our Format

- Files which do not have the % start of data tag will not be read.
- Files which have any commands other than co-ordinate points, tool definitions and the M30 end command after % will not be read.
- Files which have trailing zero suppression will not be read correctly. Any files with no zero suppression or leading zero suppression will be recognised correctly.

6.4.3 Other File Settings

Due to the limitations imposed to allow a greater range of files to be accepted by our program, the following file settings will need to be collected from the user to ensure that the co-ordinates are being correctly interpreted:

- Measurement Units (mm, mil)
- Precision (2.3,3.4, etc.)
- Board Offset (The X,Y co-ordinate pair defining the bottom left corner of the board)

6.5 Classes

This section describes the classes that will be used to implement the control software on the PC. User interface classes are not detailed in this section. The design of these classes will be detailed in Section 6.6.

The application classes will be split into packages. User interface objects will be contained in the "ui" package. The rest of the packages are described below.

6.5.1 The "excellon" package

The excellon package (Figure 6.5) contains classes to import NC (Excellon) drilling files and store and manipulate the drilling co-ordinates. Brief descriptions of the classes in this package are given below:

- DrillPoint - Holds the co-ordinates and Tool definition for a drilling point
- GerberPrecision - Holds gerber precision values (e.g. 2.3 precision files contain 2 integer and 3 decimal digits in the co-ordinates)
- PointConverter - Class to hold the gerber precision, measurement units and origin offset value for co-ordinates. Also provides functions to convert co-ordinates from the drilling file into co-ordinate measurements in millimetres for drilling.

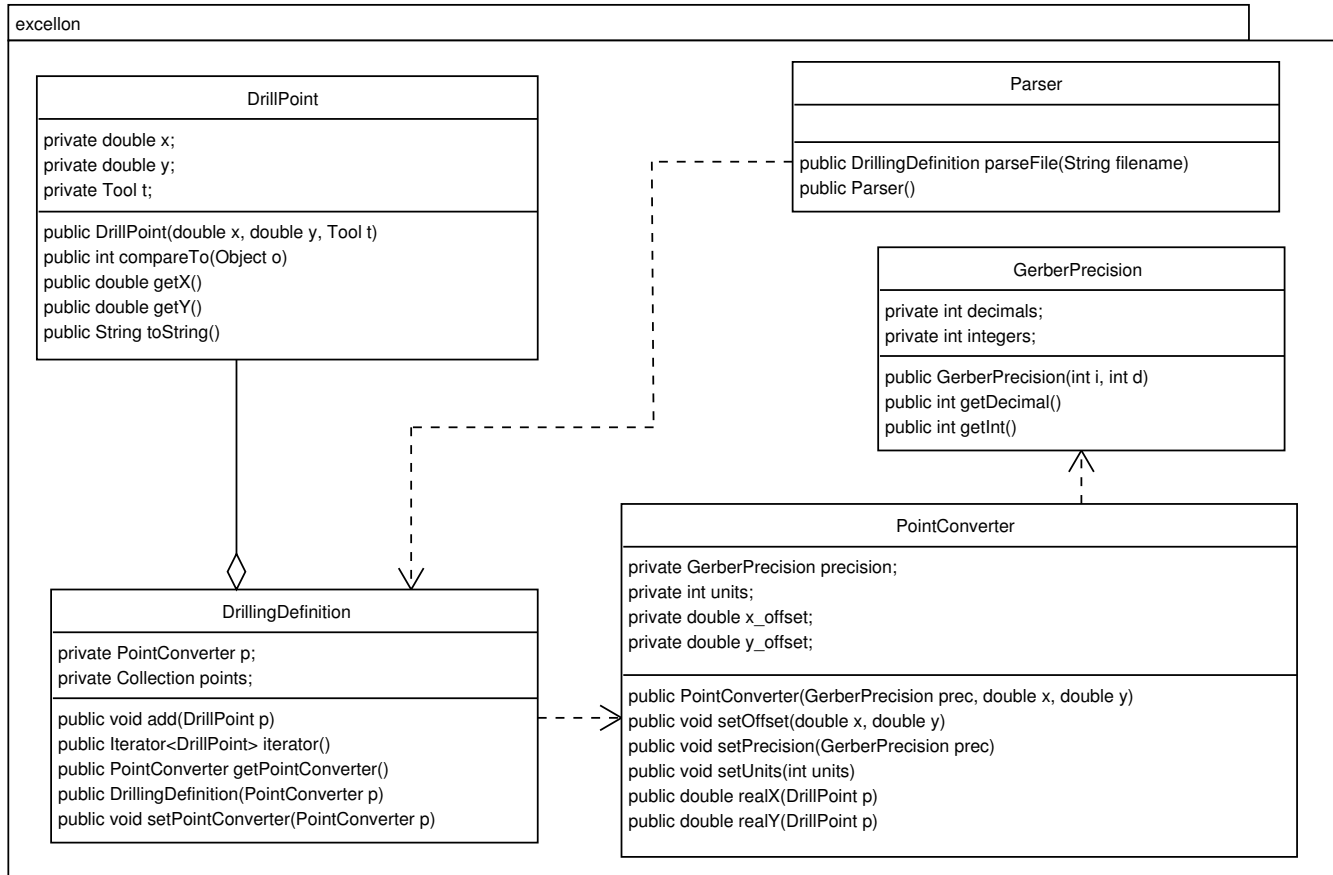


Figure 6.5: The "excellon" Package

- DrillingDefinition - Holds a PointConverter object and a set of DrillPoint values to represent the contents of an NC drilling file.
- Parser - Parses an NC drilling file into a DrillingDefinition object.

6.5.1.1 The "tools" Package

The "tools" package (Figure 6.6) is a sub-package of the excellon package. It defines an interface which specifies the methods required in a drill tool class. The package also defines two types of tool. A "PlainTool" contains only a tool name. "CircumTool" is a subclass of "PlainTool" which also holds information about the drill bit circumference. The Tool interface defines methods to retrieve the tool name and description, and comparison methods to allow the sorting of tools in alphabetical order. Sorting will allow lists of drilling points to be grouped by tool, to minimise the number of times that the user has to change the tool.

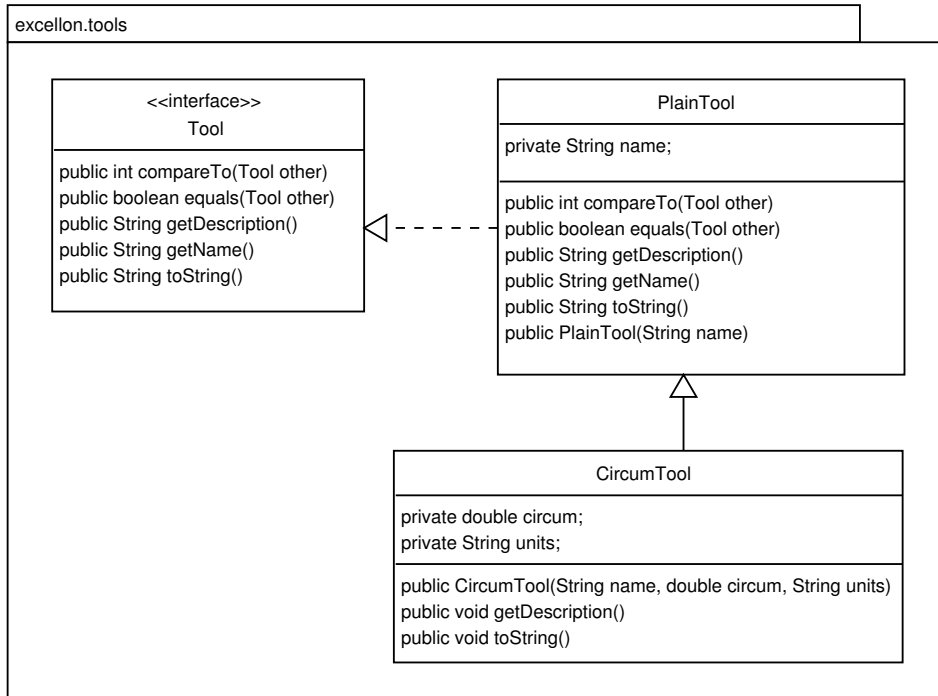


Figure 6.6: The "drilling.tools" Package

6.5.2 The "drilling" Package

The *drilling* package (Figure 6.7) contains the interfaces and concrete implementations of the hardware control classes. The interfaces are as follows:

- Hardware - Defines the lowest level of communication between the PC and the hardware device. The interface defines methods for sending and receiving 4 bit data packets, and for testing the link state.
- Drill - This interface defines the higher level methods for drilling a hole, switching off the drill, testing the drill, resetting the drill and testing whether or not the drill has been removed.

Concrete implementations of Drill and Hardware have been created in the form of the *LegoDrill* and *PhidgetsHardware* classes, which provide the exact implementation for communication with our hardware. The *DrillCalibration* class is used by *LegoDrill* to convert drill points in millimetres into integer values which are recognised by the hardware.

6.6 User Interface

The user interface to this application will be designed to be intuitive and will guide users through the use of the hardware. The main functions that the interface should support are:

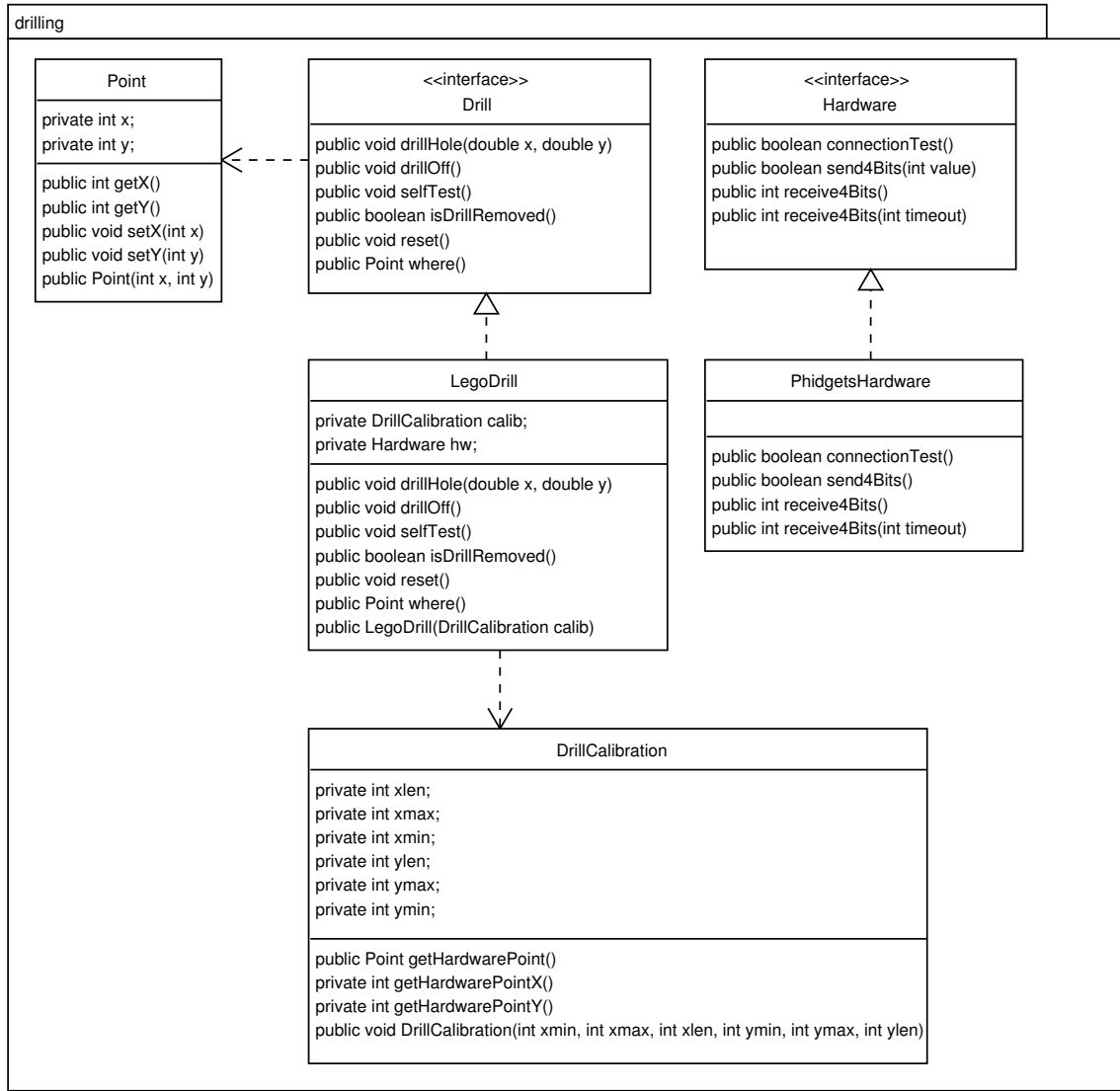


Figure 6.7: The "drilling" Package

- Loading an NC drill file, allowing the user to view the file as it will appear on the drilled board and allowing the user to make changes to the measurement units, precision and origin offset used in the file.
- Guiding the user through the process of producing the drilled board and notifying the user when errors occur or when intervention is required (e.g. changing the drilling tool).

When the application launches, the user will be presented with the empty desktop interface (Figure 6.6). This interface provides the user with no functionality other than access to the menu system.

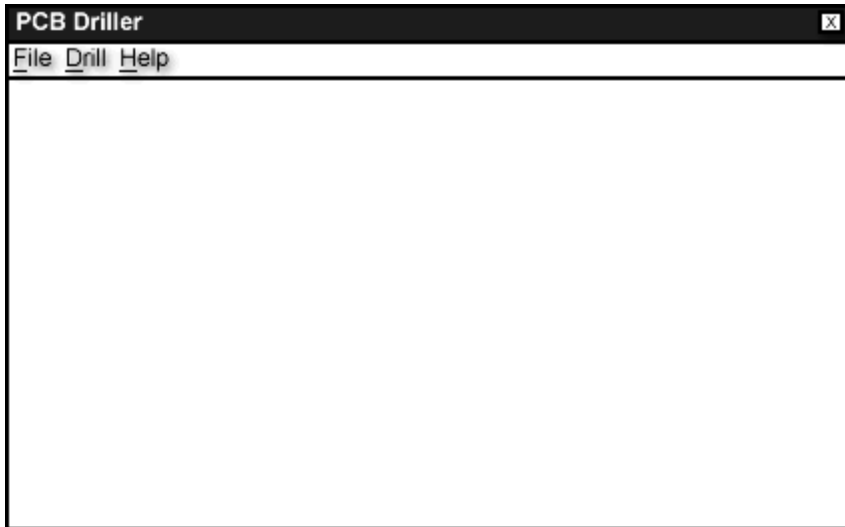


Figure 6.8: The Desktop Interface

6.6.1 The Menu System

6.6.1.1 The File Menu

The File menu will provide the following commands:

- Import NC File - Allows the user to locate and import an NC file
- Close - Close the file which is currently open
- Exit - Exit the application

6.6.1.2 The Drill Menu

- Drill PCB - Drills the currently loaded layout on a PCB
- Calibrate Drill - Allows the user to calibrate the drill so that the drilling hardware positions drill points correctly.

6.6.2 The Help Menu

The help menu has been included to allow an online help facility to be added to the application if required.

6.6.3 The Layout Editor

After successfully loading an NC file, users will be presented with the "Layout Editor" interface (Figure 6.6.3) which displays the loaded NC as it will appear on the PCB, and provides options which allow the co-ordinate precision, measurement units and origin offset.

The estimated size of PCB required to drill the layout will also be displayed, allowing the settings to be verified before drilling.

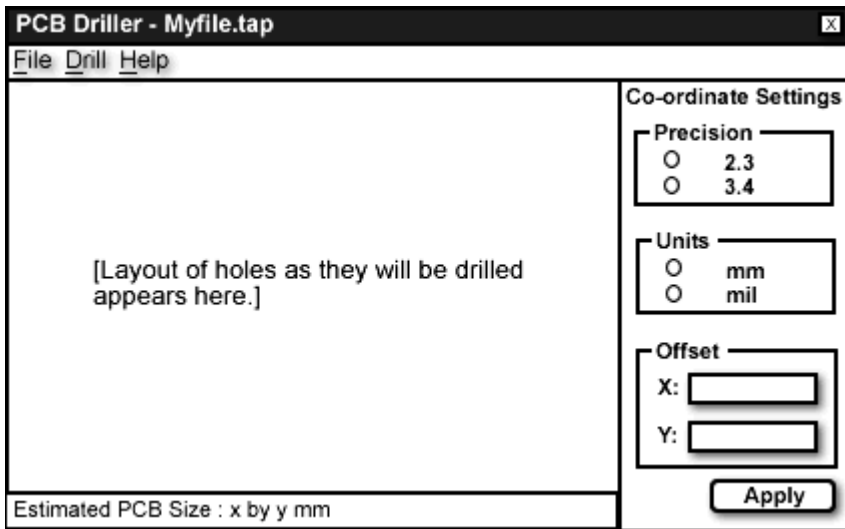


Figure 6.9: The Layout Editor

6.6.4 The Drilling Interface

The drilling interface (Figure 6.6.4) will be displayed when the user chooses to drill the loaded layout onto PCB. This interface will guide the user through the drilling process, explaining how to load the PCB and providing updates on the progress of the drilling process. The user will also be informed when errors occur, or when their intervention is required during the process. This process is illustrated in Figure 6.6.4.

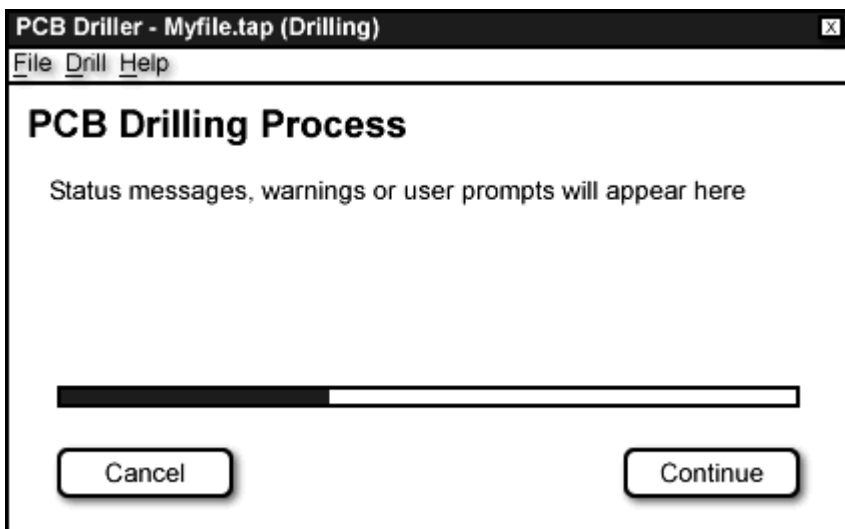


Figure 6.10: The Drilling Interface

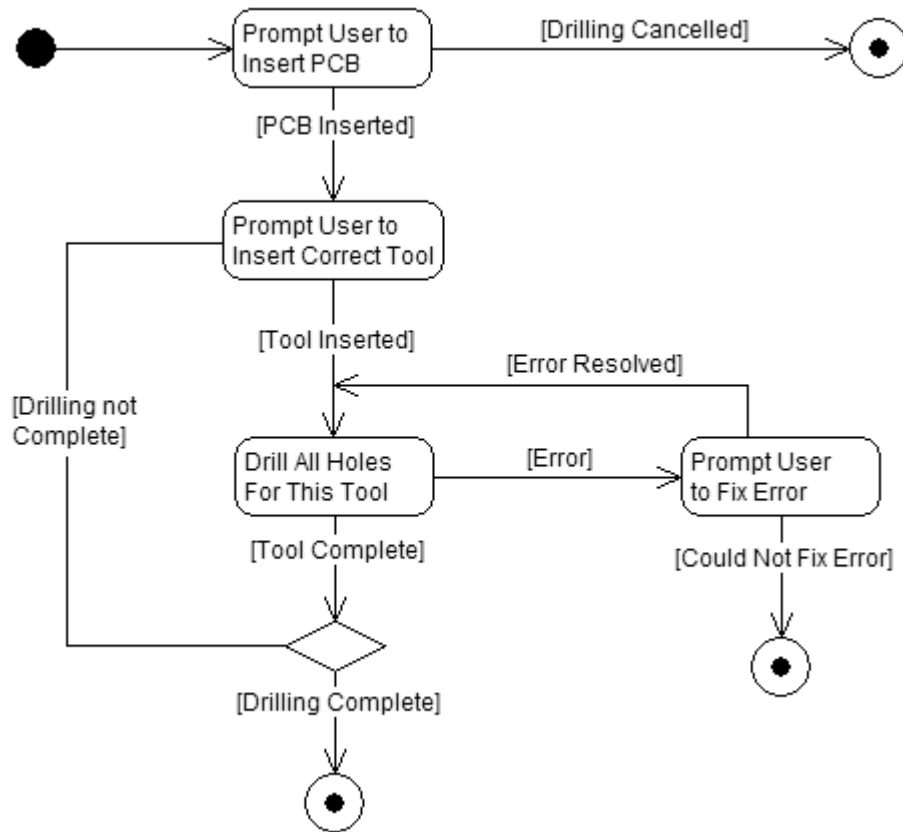


Figure 6.11: The Drilling Process - User Interaction

6.7 Implementation Issues

6.7.1 Phidgets Interface Kit

During the initial development of the *PhidgetsHardware* class (See Section 6.5.2), problems were experienced with reading values from the Phidgets' digital inputs. The inputs always provided a zero reading, regardless of whether or not an input was being received. Due to the lack of documentation for the Phidgets Java API, we had to experiment ourselves to solve the problem. It was eventually discovered that the Phidgets Interface class had a thread which had to be started to poll the inputs for readings, and this resolved the problem.

Another problem with the Phidgets API was that the inputs initially gave incorrect values when read. The conclusion was reached that this was due to unsettled voltage levels at the inputs as the system started up. This problem was resolved by reading the inputs a number of times when the connection to the interface was made, and then pausing the program for a short time to allow the inputs to settle.

6.7.2 File Precision

In the original design of the user interface, it was decided that precision values 2.3 and 3.4 would be provided for the user to make their precision selection from. However, when testing the system with drilling files from OrCAD we discovered that despite selecting 2.3 or 3.4 precision in OrCAD, the application always outputs drill files using 4.1 precision. We therefore added this precision option to our interface.

6.7.3 Slow UI response during drilling

Due to the fact that during the drilling process, communication with the hardware was controlled by the thread which was also responsible for handling user input events, the system was often slow in responding to user input. For example, when a user selected the cancel button while a point was being drilled, the button input was not processed until the drilling completed. We have included a "Please Wait..." message which appears when a user selects cancel, so that the user is notified that their input has been received.

6.8 Testing

6.8.1 Hardware Simulation

To ensure that the hardware interfaces were following the correct processes, testing classes which implemented the *drilling.Hardware* and *drilling.Drill* (Section 6.5.2) interfaces were created. These classes obtained user input to simulate responses from the hardware. They were extremely useful because they allowed the implementation of the software to be carried out independently of the hardware implementation.

The drill simulation class was extended at a later date so that the drill co-ordinates which would be sent to the hardware were stored in a CSV spreadsheet. This allowed other tests to ensure that the drilling co-ordinates were being calculated correctly by the software.

6.8.2 Testing of Parser

The NC file parser was tested using a set of NC files, which were specially designed to ensure that the parser would read correctly-formatted files and reject files containing invalid data. During this process, the on-screen preview of the data contained in the files was also checked to verify that the co-ordinates were being read correctly from the file. Correctly-formatted files which were accepted by the parser were also "drilled" using the drilling simulator, to ensure that the output produced by the drilling stage was also as expected. More information about these tests is available in Appendix C.1.

6.8.3 JUnit Tests

JUnit tests were used for the *drilling.DrillCalibration* (Section 6.5.2) and *excellon.PointConverter* (Section 6.5.1) classes, which handle the numeric processing of drill points in the application, to ensure that the results produced were correct. We have chosen to use JUnit tests on these classes because the correct functionality of these classes is essential if the software is to produce the correct output. Only one problem was uncovered during the testing process: the `setUnits()` method of the *excellon.PointConverter* had a small bug which caused it to ignore the "mils" measurement unit setting. All other tests provided the correct results.

More information about the JUnit tests is included in Appendix C.2.

6.8.4 Large Scale Testing

To ensure that the program was able to handle large-scale designs, a drilling simulation was carried out using one of the sample library files supplied with OrCAD. This file contains over 1400 drill points. We also compared the output produced by the simulator with the original PCB dimensions in OrCAD to confirm that the software was calculating the correct locations for all points. This tests was also successful. More information is available in Appendix C.3.

Chapter 7

Integration & System Testing

7.1 Integration

We took the approach whereby we integrated the hardware and software from an early stage in the system development. We decided that this approach would be best because we could ensure that the two were working together from the start of the development process.

7.2 System Testing

System testing was carried out at all stages of the system development. This took many forms throughout the process, as different parts of the system were implemented. Software testing was relatively straightforward, as verbose testing output could be provided from the software development environment, and simulations could be used to test the software's interaction with the hardware.

Once the software was complete, and we began to implement the hardware, we used a wider variety of testing approaches. The PC interface could often be used to obtain feedback from the hardware to trace the point of execution in the code, which was useful for debugging the microcontroller software. When this was not possible, we used other approaches including slowing the hardware down and measuring the voltages on the microcontroller pins, and connecting LEDs to the microcontroller pins which allowed us to read the pin values easily.

When we had a partially-working system, we were able to test the system using the full PC software suite to control the parts of the hardware which were working. Hardware functions which were not implemented in any of these testing stages were simulated by the communications PIC so that the drilling process would work correctly on the PC.

Unfortunately, due to the problems experienced, we were unable to complete the system, but if the system had been completed, an evaluation with potential users would have been carried out after formal testing of the system's accuracy, to assess the system from an external point of view.

Chapter 8

Conclusion

8.1 Current System Status

The following parts of the system have been implemented completely:

- PC Software and Related I/O Interface
- Communications chip for interfacing hardware with PC
- RCX Drill Up/Down Control
- Drill Power Control
- Motor Movement Control

The following functions have not been completely implemented:

- Feedback communication from the control chip - In the current prototype system, no feedback is provided from the control chip due to a problem with the serial link to the communications chip. The communications chip receives a command from the PC (e.g. "Drill Hole"), relays the command to the control chip, waits for sufficient time for the instruction to be completed and returns "instruction completed" to the PC.
- Position Sensors - The development of the position sensors relied on the feedback communication to the PC to allow us to easily read the sensor values during development. Without this functionality, we were unable to successfully implement the sensors.

The current prototype system consists of the final implementation of the software, which would have been used with the fully-functioning system, along with an incomplete hardware implementation.

The hardware has been adjusted so that it fully demonstrates the functions which are currently working. Instead of drilling holes based on the software input, the hardware moves the position motors a set distance and drills a hole each time it receives a "Drill

Hole” command. This demonstrates the working motor control and drill control systems. ”Drill Off” commands are responded to correctly. The hardware always provides the correct feedback to the software, to simulate a fully-working system.

In addition to this, we have also connected one set of axis position sensors to a PIC which displays the current position value as a binary value on a set of LEDs, to demonstrate the way in which the system would read the sensors.

8.2 Achievements of the Project

We feel that we have achieved a great deal during the course of this project and are satisfied with the final outcome, despite the failure to produce a completely working system. The outcome of the project - all aspects functioning correctly excluding two related parts - and the fact that with a small amount of further implementation work, the system could be complete, was extremely satisfying to us.

The project has been very useful for developing skills in project management as well as the technical aspects such as hardware and software design, implementation, testing and problem solving. The experience has also been very useful in developing our skills in debugging code, as we had no debugger available for the PIC software.

8.3 Main Problems Encountered

Only two major problems were encountered during the design of the project. The choice of file format (Section 6.4) was a problem during the software design due to a lack of information and the failure of CAD applications to conform to the set standards. During the hardware design, we encountered difficulties in choosing sensors (Section 5.1.3). The problems arose both because of the fact that the sensors had to be suitable to attach to the Lego and because we needed to find affordable sensors that would provide the required accuracy.

The Lego equipment placed large constraints on our project. The basic infrared communication facilities offered by the RCX could not be easily incorporated into our Java software, resulting in the need for external communication circuitry which had to be interfaced with the RCX. The spacing of the Lego brick connectors and the sizes of the gears were also difficult to work with sometimes, as we occasionally required to fit the lego construction around other hardware which was not of the standard lego size (for example, the drill).

Most of the problems encountered during the implementation of the project were involved in the hardware implementation stage, and in particular interfacing the different hardware and software components. Most of these problems arose due to the fact that we were using a number of different pieces of hardware, all of which introduced the possibility of errors.

The hardware development was also constrained by the fact that debugging errors in the hardware system was much more time-consuming than debugging software, where the

exact position of execution can be followed using a debugger. Hardware debugging involved firstly determining whether the problem was related to the circuitry or to the microcontroller software, and then taking the necessary steps to resolve the problem. Another complication was added by the fact that the only way to debug the microcontroller software was to add signals at certain points in the code to allow us to follow the execution. These signals generally involved turning an output pin on or off and measuring the voltage, or observing the behaviour of a connected component such as a relay. The positive outcome of this process was that we greatly developed our skills in debugging code without the aid of tools.

8.4 Further Development

From the current system state, the following additions and refinements could be made to improve the system:

- **Correct Current Problems** - If the current problems with the feedback communication system were resolved, we believe implementation of a fully working system would become trivial, as the sensor systems already exist for integration into the overall system.
- **Increasing the range of file formats available** - As well as the NC format, the ability to create a drilling layout from an image file would also be useful, as a number of free PCB design systems produce graphical rather than data output. This could be achieved by displaying the graphic on screen and allowing the user to indicate the positions of holes on the graphic using the mouse.
- **Automatic precision detection** - The system currently requires file precision information from the user in order to correctly calculate the co-ordinates for the drill holes. An extension to this would be to prompt the user for the dimensions of the circuit board being drilled, and calculate the precision settings so that the loaded layout fits sensibly on the board. This would remove the need for the user to know the exact settings for every layout that they create.
- **Increasing drilling speed** - The drilling speed could be increased greatly by only lifting the drill to its "fully up" position when the bit needs to be changed. This would save large amounts of time that are spent waiting for the drill to be lifted. Slight speed increases could be made to the communication system, but these would have negligible effect on the overall drilling speed.
- **Increasing drilling accuracy** - The accuracy of the system could be improved by using sensors with a higher resolution (resolution is currently 12 bits, giving 4096 possible values for each axis). Using gears with less slippage than the Lego gears currently in use would also improve the accuracy of the system.

- **Eliminating the need for an RCX controller** - The RCX controller for the drill up/down movement was left in the system due to time constraints. The system could be made more efficient by removing this controller, and using a PIC to control the drill axis directly. This would also reduce the variety of hardware devices being used and therefore decrease the risk of interfacing problems.
- **Improve the circuitry and power supply** - The system is still currently implemented as a prototype with all circuitry on veroboard and separate power supplies for the different voltage and current levels. This could be improved by creating a single printed circuit board for all of the system's components and creating a power supply circuit which is capable of accepting a single power input and providing all of the separate power supplies required.
- **Use a single PIC** - The system currently has two 18-pin PIC microcontrollers because there are not enough I/O pins on one to accommodate all of the connections required. This problem could be solved by either using a PIC with more I/O pins, or by altering the external circuitry (for example, the communications link to the PC) to require less I/O pins. Most of our major implementation problems were encountered in the PIC to PIC link, and therefore the use of a single PIC would eliminate these problems.

Bibliography

- [1] Examples of Gerber and Excellon Data Files, December 2005, <http://www.pcbmilling.com/ExamplesofGerberandExcellonDataFiles.htm>
- [2] "Computer Numerical Control for Drillers and Routers", APCircuits: NC Drill File Format, January 2006, http://www.apcircuits.com/resources/information/nc_introduction.html
- [3] APCircuits: NC Sample File, January 2006, http://www.apcircuits.com/resources/information/nc_sample.html
- [4] Java Docs, Phidget Documentation and Examples, January 2006, <http://www.phidgets.com/documentation/JavaDoc.zip>
- [5] "PhidgetInterfaceKit 8/8/8", Phidget Documentation and Examples, December 2005, <http://www.phidgets.com/documentation/1013.pdf>
- [6] "Help On LaTeX Commands", University of Cambridge, March 2006, <http://www.eng.cam.ac.uk/help/tpl/textprocessing/teTeX/latex/latex2e-html/ltx-2.html>
- [7] (Author Unknown), *PicBasic Pro Compiler Reference Manual*. microEngineering Labs, (Year of Publishing Unknown)
- [8] ULN2803A Data Sheet, March 2006, <http://us.st.com/stonline/books/pdf/docs/1536.pdf>
- [9] MCP3208 Data Sheet, March 2006, <http://ww1.microchip.com/downloads/en/devicedoc/21298c.pdf>
- [10] PIC16F628 Data Sheet, March 2006, <http://ww1.microchip.com/downloads/en/devicedoc/40300c.pdf>
- [11] LM78L05 Data Sheet, March 2006, <http://cache.national.com/ds/LM/LM78L05.pdf>

Appendix A

Contributions

A.1 Liam MacIsaac

A.1.1 Summary of Project Log

October

- Choosing Project Topic
- Shown various aspects of hardware

November

- Wrote some standard functions in NQC for axis control.
- Sampled different types of sensors.
- Designed a small number of java classes for the PC software.
- Wrote full NQC control software for axes.

December

- Created relay board to connect phidgets to lego switches and created some sample code to test Phidgets controlling Lego.
- Wrote an initial prototype of a drill point preview control and a basic application to hold the control.

January

- Designed revised classes for the application for a better structure than the prototype.
- Investigated Gerber and NC file specifications and decided on file format.
- Completed final software design.
- Carried out some implementation of the PC software.

February

- Phidgets interface kit stopped working, ordered new component and got the board repaired.
- Developed lower level communication classes and tested.
- Implemented the drilling process user interface.
- Created a drill calibration interface.
- Programmed the upper level of the communication system on the PC to meet the agreed protocol.
- Performed JUnit testing of numeric handling classes.
- Performed testing with sample input files.

March

- Wrote test applications for testing hardware functionality.
- Fixed small bugs in the PC software.
- Created drill simulator for further software testing.
- Carried out final software testing.
- Implemented parts of the microcontroller software.
- Final report sections written.

A.1.2 Report Sections

- Introduction (1)
- Lego RCX Units (5.4)
- PC Software (6)
- Conclusion (8)

A.2 Benjamin Marwick Johnstone

A.2.1 Summary of Project Log

October

- Shown available hardware
- Chose project topic

November

- Built axes

- Started investigating types of sensor
- Built prototype of paired light sensor system
- Investigated light sources for sensors

December

- Decided to abandon light sensor system and use potentiometers instead.
- Created PIC to Phidgets comms protocol
- Started experimenting with PIC to Phidgets comms.

January

- Built PIC Programmer
- Tested PIC C Compilers
- Debugged problems with Programmer Comms
- Prototyped A-D converter

February

- Prototyped Motor Control subsystem
- Added more gears to Axes
- Other minor refinements to Lego hardware.
- Created simulator for Comms Chip
- Created PIC-PIC comms protocol

March

- Implementation and Testing of PIC-PIC comms protocol
- Built hardware circuitry on veroboard
- Implemented Comms chip
- Created simulator chip to run on control board
- Wrote report sections on hardware, system overview and hardware-software interface

A.2.2 Report Sections

- System Overview (3)
- Hardware & Software Interface (4)
- Hardware (5) - excluding (5.4)

All report sections which are not listed above were written jointly by both team members.

Appendix B

Acknowledgements

We would like to thank the following people for their help during our project:

- **Ray Welland** and **Martin MacAulay** for supervising our project.
- **The 2004/2005 ESE Project Team** for their advice on using the Lego Mindstorms Kit and providing links to software and tutorials for the NQC language.
- **Phil Gray** for help with swing.
- **Shona Ballantyne** in the Electronics Department Stores for ordering our components.
- **The EEE Technical Staff** for helping us to repair the Phidgets board and providing PCBs to drill.
- The staff from **Maplin Electronics** for their assistance in locating components.

Appendix C

Software Testing Data and Results

C.1 Testing NC Files

C.1.1 invalid.tap

%	File comments:
T001	
X30	• File is not valid because there is not previous
M30	Y co-ordinate defined before X30.

Result : parser error was correctly displayed for file.

C.1.2 invalid2.tap

T001	File comments:
X30Y40	
M30	• File is not valid because there is no starting % tag.

Result : parser error was correctly displayed for file.

C.1.3 empty.tap

%	File comments:
M30	• File should produce a parse error because it contains no points.

Result : parser error was correctly displayed for file.

C.1.4 test.tap

File comments:

%	• Tool T001 should appear on screen with no tool size information, only the name, when the user is prompted to insert the tool.
T001	
X100Y100	
Y90	
Y80	• Points defined Y90...Y50 should use the X co-ordinate of 100 defined previously.
Y70	
Y60	
Y50	• Tool T002 should display the circumference of 0.054 on screen when prompting the user to insert the tool.
T002C0.054	
X60	
X70	• Points defined X60...X100 should use the Y co-ordinate of 50.
X80	
X90	
X100	• Points X200Y200 and Y400X400 should both be read correctly, despite the co-ordinates being defined in a different order.
T003	
X200Y200	
Y400X400	• The file is missing the M30 closing tag, but this is not essential and should not generate an error.

Result: The file was successfully loaded. The file was also tested with the drill simulator, using the following settings:

Precision: 4.1

Units: mm

Offset: (0,0)

The simulator calibration settings were set to produce an output of 1 point per mm. The results of the simulation are illustrated in Figure C.1.

C.2 JUnit Tests

JUnit testing was used for the *excellon.PointConverter* and *drilling.DrillCalibration* classes because these the correct functionality of these classes is extremely important in the process of transforming file co-ordinates into physical drilling co-ordinates.

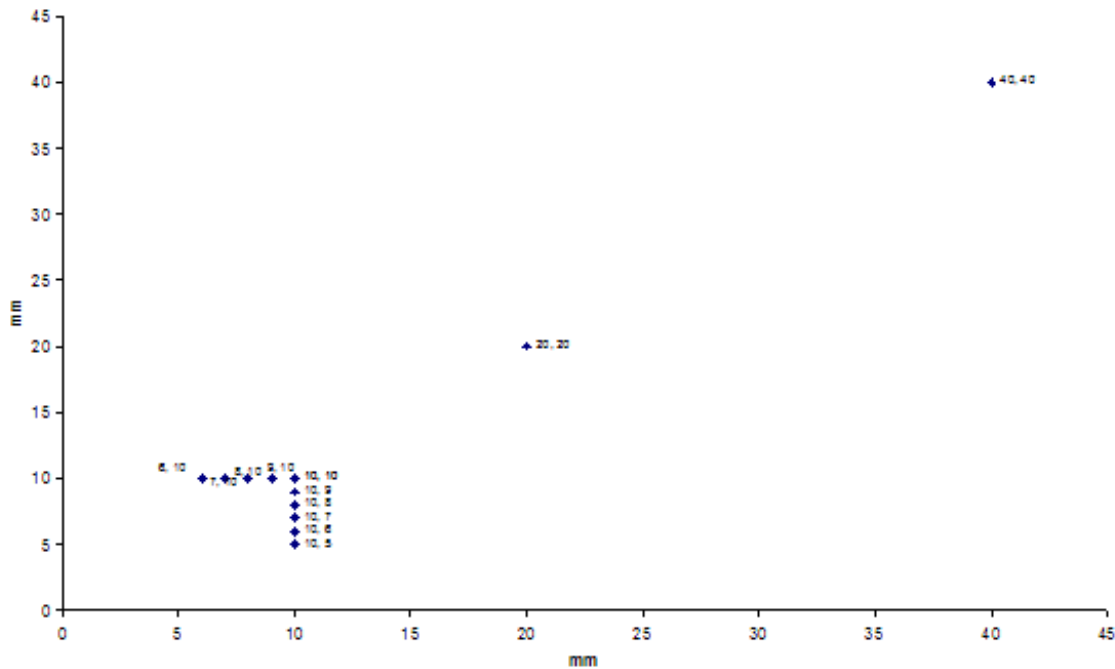


Figure C.1: Output from drill simulator for "test.tap"

C.2.1 *DrillCalibration* Class Testing

C.2.1.1 Rationale

The *DrillCalibration* class is responsible for converting hole positions in mm into 12-bit X and Y co-ordinates which are recognised by the hardware. To perform this conversion, the class requires the following data:

- XMIN and YMIN - The X and Y hardware co-ordinates of the bottom-left corner of the drilling area.
- XMAX and YMAX - The X and Y hardware co-ordinates of the top-right corner of the drilling area.
- XLEN and YLEN - The horizontal and vertical lengths in mm of the drilling area.

Given this data, the class calculates the actual X and Y drilling points using the following formulae:

$$[X \text{ Drilling Point}] = ((XMAX - XMIN) / XLEN) * [X \text{ Co-Ordinate in mm}] + XMIN$$

$$[Y \text{ Drilling Point}] = ((YMAX - YMIN) / YLEN) * [Y \text{ Co-Ordinate in mm}] + YMIN$$

The aim of the JUnit test on this class is to ensure that the hardware co-ordinates returned by the `getHardwarePoint()` method of this class are correct.

C.2.1.2 Test Method

The following algorithm will be used to test the class:

1. Repeat 1000 times:
 - 1.1 Select Random Values For XMIN,XMAX,YMIN,YMAX,XLEN,YLEN ensuring that XMAX>XMIN and YMAX>YMIN. Only positive values will be selected.
 - 1.2 Create a new DrillCalibration class with the chosen values
 - 1.3 Repeat 1000 times:
 - 1.3.1 Select random values for X and Y co-ordinates in mm
 - 1.3.2 Calculate Expected Hardware Co-ordinate points
 - 1.3.3 Compare with value given by `getHardwarePoint()` method
 - 1.3.4 If Equal Test Passed, Else Test Failed

C.2.1.3 Results

This JUnit test passed successfully with no errors.

C.2.2 *PointConverter* Class Testing

C.2.2.1 Rationale

The *PointConverter* class is responsible for converting the raw co-ordinate values read from an NC file into co-ordinates in mm. This is done using the following information:

- Board Offset - The X and Y distance (positive or negative) of the board's bottom left corner from the (0,0) co-ordinate. (This value is measured in native file co-ordinates)
- Measurement Units - mm or mils (1/1000th inch)
- Precision - 2.3, 3.4 or 4.1. These values represent how many integer and decimal digits are in the file's co-ordinates.

The JUnit test for this class will test the following methods:

- `realX()` and `realY()` - These methods return the X or Y co-ordinates in mm for a given *DrillPoint* object.
- `setOffset()` - Sets the board offset co-ordinates
- `setUnits()` - Sets the measurement units

C.2.2.2 Test Methods

The following test algorithm was used to test the `realX()` method:

1. Create new `PointConverter` Class
2. For units = {mm,mils}
 - 2.1 For Precision = {2.3,3.4,4.1}
 - 2.1.1 Repeat 100 times:
 - 2.1.1.1 Choose Random Point for X Offset
 - 2.1.1.2 Repeat 100 times:
 - 2.1.1.2.1 Choose random point for Y offset
 - 2.1.1.2.1.1 Choose random points for file X and Y co-ordinates
 - 2.1.1.2.1.2 Set offset of `PointConverter`
 - 2.1.1.2.1.3 Calculate correct value for output point
 - 2.1.1.2.1.4 Compare with value from `PointConverter`, fail test if not equal

A similar algorithm to the one above was used to test the `realY()` method. The following algorithm was used to test the `setOffset()` method:

1. Create new `PointConverter` Class
2. Repeat 1000 times:
 - 2.1 Select random values for X and Y offset
 - 2.2 Set X and Y offset of `PointConverter`
 - 2.3 Check that X and Y offset of `PointConverter` are equal to the values specified, if not fail test

The following algorithm was used to test the `setUnits()` method:

1. Create new `PointConverter` class with units set to mm in constructor
2. Verify that units are set to mm, if not fail test
3. Set Units to mils
4. Verify that units are set to mils, if not fail test
3. Set Units to mm
4. Verify that units are set to mm, if not fail test

C.2.2.3 Results

The following problems were encountered during testing:

- The tests for the `realX()` and `realY()` methods initially failed. This was due to the fact that double precision floating point numbers were being used to store the coordinate values and accuracy was being lost in calculations. To resolve this problem,

all values were typecast to single precision floating point numbers before comparison to eliminate the insignificant decimal places in the result. The tests then passed with no errors.

- The `setUnits()` method was validating its input parameter incorrectly and was only allowing the measurement units to be set to mm. This problem was resolved and the test then passed with no errors.

The test for the `setOffset()` method passed with no errors.

C.3 Large File Test

This test uses a large-scale design from the OrCAD samples library which contains 1479 drill points. The main aim of this experiment is to ensure that our system can handle large input files, however we have also decided to use this test to ensure that the system calculates the correct output for a given input file.

C.3.1 File Handling

The system successfully loaded the file with no noticeable time delay when processing the file, or adjusting the file's precision properties.

C.3.2 Output Generation

To ensure that the file has been processed correctly for output to the drilling hardware, we will use a simple test where four drill points will be recorded from the original OrCAD layout (Figure C.2) and the same points will be recorded from the simulator output (Figure C.3). The distance between these points will then be calculated from both sets of data to ensure that they are equal.

Notes:

- The OrCAD point measurements were taken by hand with a cursor and may therefore be only approximations.
- The calibration settings for the drill were chosen so that the simulator generated an output of exactly 1 unit per millimetre.
- The simulator output diagram was generated by using a spreadsheet package to produce a scatter chart of the CSV file generated by the simulator.

Table C.1 shows the results of the measurements taken between the four data points A,B,C and D for both the input and output. It can be concluded from these results that the system is producing the correct drilling data for the hardware.

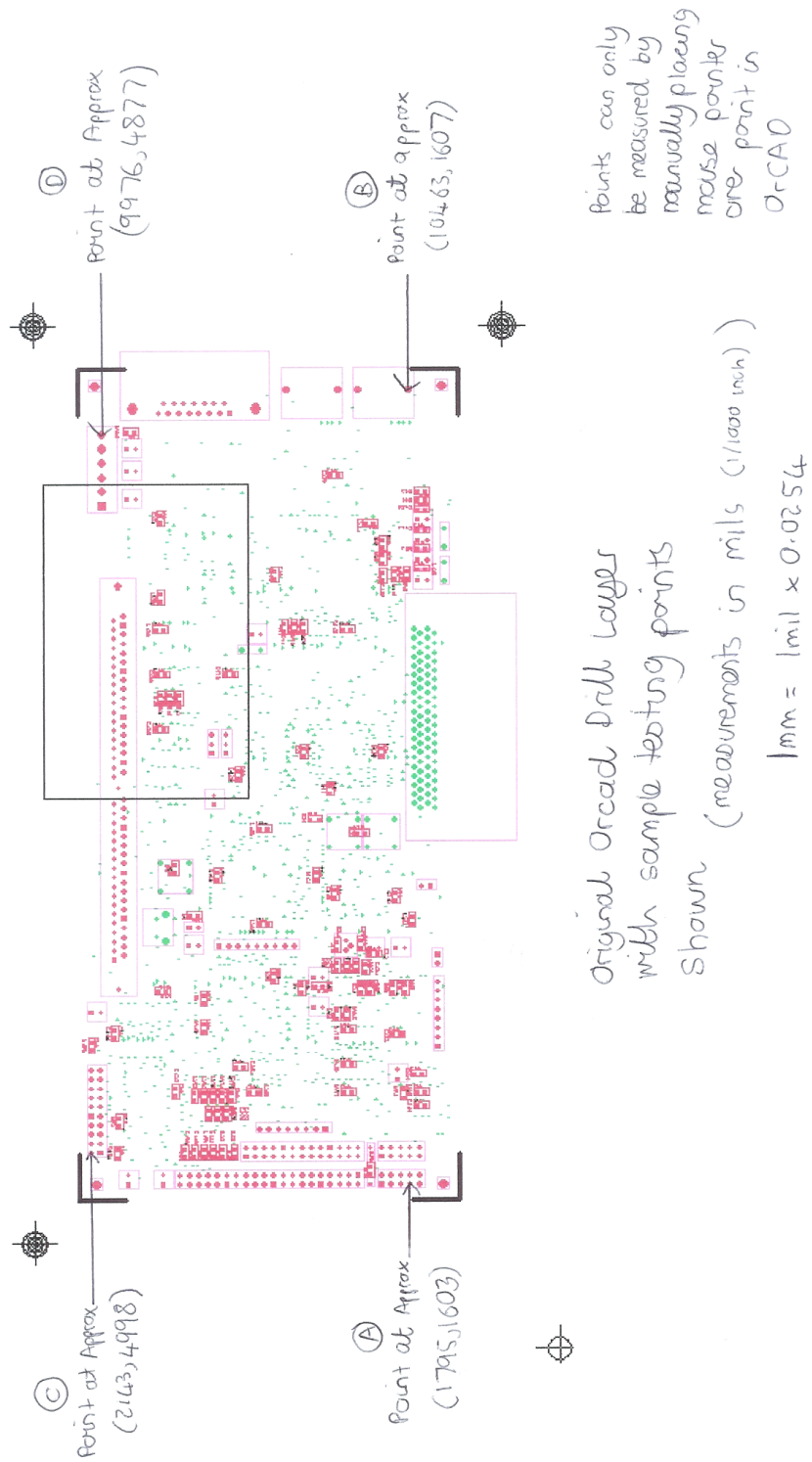


Figure C.2: OrCAD Input File Layout

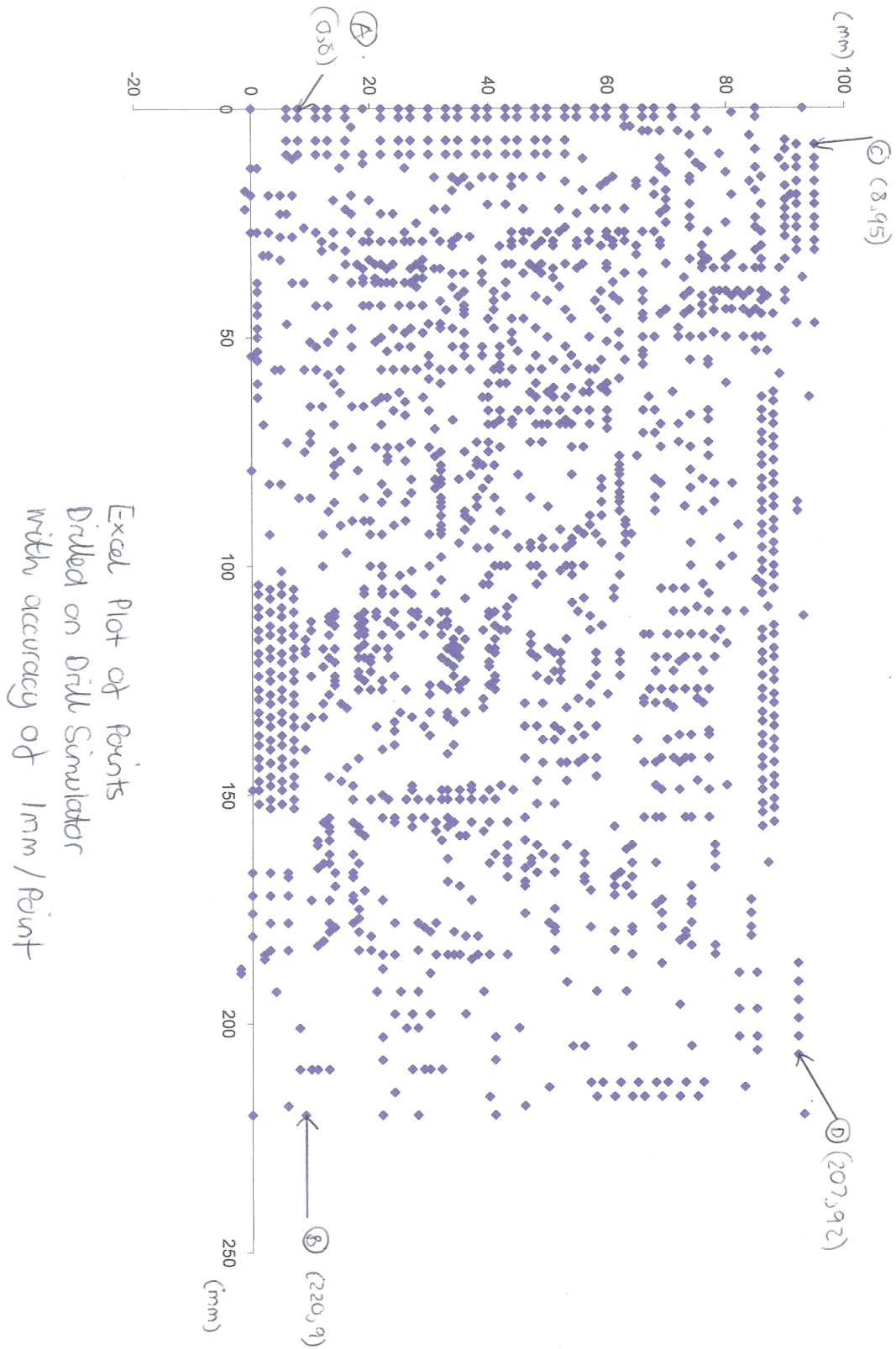


Figure C.3: Plot of Simulation Output Data

Start Point	End Point	Input Measurement (mils)	Input Measurement (mm)	Output Measurement (mm)
A	B	8668	220	200
B	D	3306	84	84
D	C	7834	199	199
C	A	3413	87	76
B	C	8985	228	229
A	D	8812	224	223

Table C.1: Simulator Test Results : Values Rounded to Nearest Unit

Appendix D

Glossary

BrickOS An operating system for the Lego RCX which allows the controller to be programmed in C or C++

CNC Machine Computer Numerical Control - A machine which is controlled by a computer file containing numerical co-ordinates.

CSV Comma Separated Values - A text-only file format for storing tabular data, one row per line. Columns within a row are separated by commas.

EasyPC An application for designing PCB layouts

Excellon A manufacturer of CNC machines. NC files are sometimes referred to as excellon files.

Gerber A file which defines the track layout of a PCB. NC files are based on Gerber files.

Lego Mindstorms Robotics Invention Kit A kit produced by Lego which provides Lego bricks, gears, motors, sensors and RCX units for creating robotic devices using Lego.

NC Numerical Control - Refers to any form of numerical control for a hardware device, but in our case refers to the Drilling Co-ordinate File that is used as the input to our system.

NQC Not Quite C - A C-like programming language which can be used to program the Lego RCX.

OrCAD An electronics design suite with a wide selection of features, including the ability to create printed circuit board layouts.

PCB Printed Circuit Board

Phidgets Interface Kit A USB PC interface which allows 8 digital inputs, 8 digital outputs and 8 analogue inputs to be connected to the PC.

PIC Programmable Integrated Circuit - A microcontroller chip

RCX A control unit provided with the Lego Mindstorms Robotics Invention Kit which has 3 sensor inputs, 3 motor outputs and contains a microcontroller which can be programmed to control Lego hardware.

Relay, DPDT - Double pole, double throw relay - A switch with two inputs, each of which can be connected to one of two outputs.

Relay, SPDT - Single pole, double throw relay - A switch where one input can be connected to one of two outputs.

SPI - Serial Peripheral Interface - A 4-line serial communications protocol.

Appendix E

Related Documentation

E.1 NQC Reference Card

E.2 Communications Protocols