Solving Rubik's Cube with Lego Mindstorms

Will Garside

01-01-1901

Signed Declaration

All sentences or passages quoted in this report from other people's work have been specifically acknowledged by clear cross-referencing to author, work and page(s). Any illustrations which are not the work of the author of this report have been used with the explicit permission of the originator and are specifically acknowledged. I understand that failure to do this amounts to plagiarism and will be considered grounds for failure in this project and the degree examination as a whole.

Name: Will Garside
Signature:
Date:

Abstract

This should be two or three short paragraphs (100-150 words total), summarising the report. A suggested flow is background, project aims, and achievements to date. It should not simply be a restatement of the original project outline

Acknowledgements

Thanks to my parents, who raised me since I was a boy. And Rik van Grol, who raised me afterwards.

Abbreviations, Definitions and Notations

Table 1: Terminology used in this document

Abbreviation/Word	**Definition**
Rubik's Cube Cube	A standard 3x3x3 Rubik's Cube[1].
Face	One of the size faces on a Cube. Denoted by position (LRFBUD) or colour.
Slice	A central layer between two faces. Usually referenced by the faces it spans.
Quarter Turn	A clockwise rotation of a face or slice by 90°
Half Turn	A clockwise rotation of a face or slice by 180°
Quarter Turn Metric	When counting moves, a Quarter Turn is counted as a single move and a half is two moves.
Half Turn Metric[2]	Quarter turns and half turns are both counted as single moves.
Cubie	One of the twenty-six smaller cubes that make up a Cube.
Goal State	All the cubies on a given face match the colour of the centre cubie. i.e. a solved Cube.
Position	A Cube's state (mixed or solved).
Valid Position	A position that can be achieved with a real-world Cube without dismantling it.
Move Sequence	A series of moves performed consecutively. e.g. $F\ D\ F'\ D2\ L'\ B'\ U\ L\ D\ R\ U\ L'\ F'\ U\ L\ U2$
Solve (Sequence)	A move sequence which leads to the goal state.
Depth n	A Cube which has been moved n times away from the goal state.

Table 2: Notation used in this document

Symbol Notation	**Meaning**	
L	Left	This notation is used to show a quarter turn of a face. It can have a single quote or a '2' appended to show a counter-clockwise quarter turn or a half turn respectively.
R	Right	
F	Front	
B	Back	
U	Up (Top)	
D	Down (Bottom)	
X	Clockwise 90° rotation of a Cube about the relevant axis.	
Y		
Z		

[1] This Dissertation is only dealing with 3x3x3 Rubiks Cubes, and any discussion of alternative dimensions will be explicitly stated.

[2] For this Dissertation the Half Turn Metric is used unless explicitly stated.

Contents

Chapter 1

Introduction

1.1 Background

In 1974, Ernö Rubik was struggling to create a cube with independently moving parts which remain together, regardless of how much they moved. His first attempts made use of elastic, which broke and rendered the cube unusable. Rubik persevered in his attempts to hold the blocks (now called 'cubies') together - eventually concluding that the best way was to have the cubes hold themselves together. He called this design 'The Magic Cube', and it would go on to be one of the world's best-selling puzzles [1]. It was later re-branded to '*Rubik's Cube*' to overcome an oversight involving patenting and copyrighting the design.

In an unpublished manuscript [2], Rubik described first randomising his new cube,

> It was wonderful to see how, after only a few turns, the colors became mixed, apparently in random fashion. Like after a nice walk when you have seen many lovely sights you decide to go home, after a while I decided it was time to go home, let us put the cubes back in order. And it was at that moment that I came face to face with the Big Challenge: What is the way home?

It took Rubik over a month to solve this first cube - he knew intuitively that there must be a method to solving the cube, but lacked the finer methodology [3]. Since Rubik devised the first method, hobbyists and mathematicians alike have been immersed in solving the Cube as quickly and efficiently as possible. Whilst many solutions are markedly successful when it comes to optimisation, others only better them in quirkiness or internet fame [4].

There is one method which is mere speculation, despite having been proved mathematically: God's Algorithm. God's Algorithm states that an omniscient being would always make the most efficient moves and that they would be able to solve a Cube from any given position in a certain number of moves or less. This is referred to as God's Number, and was finally proved to be twenty in 2010 by a group of 4 researchers [5].

1.2 General Objectives

The primary objective of this project is to successfully implement an algorithm to solve a Cube with a Mindstorms robot. The efficiency of this algorithm is initially non-imperative, but will ideally be improved over the project time-line. The most recent iteration of the Mindstorms line, the Lego EV3 31313, will be used in the construction of the robot. This provides a wireless connectivity via Bluetooth (or WiFi with a USB dongle), three motors, a colour sensor, and a touch sensor amongst other peripherals. A custom operating system can also be installed on a microSD card to allow for greater expandability.

Secondary objectives include implementing other algorithms from various sources, devising and refining my own algorithms, and comparing the performance, efficiency and solve-length of the algorithm. The robot will have to be of sound construction, with little-to-no room for error when manipulating the Cube.

The objectives for this project are as follows:

1. Build a robot which can move a Cube to a sufficient degree of accuracy
2. Write a program which takes a move sequence as its input and moves a Cube to match that sequence
3. Implement a system which successfully generates a solve sequence for any given position
4. Ensure the runtime of the system is an acceptable length
5. Implement a program to use other algorithms to generate a solve sequence
6. Compare the performance of different algorithms, especially the difference between human-compatible and robot-compatible move sequences

1.3 Limitations and Constraints

The scope of this project is relatively local and therefore it lacks major limitations. Instead, it has several minor limitations which each present their own problems and solutions. The first problem which will be encountered will undoubtedly be during the design and build of the robot: the number of Lego pieces supplied in the EV3 set is quite limited at only six-hundred-and-one pieces - approximately a quarter of which are only for aesthetics. The design will be supplemented by sets sourced externally, thus allowing a robot of sufficient quality to be built.

Despite being of high quality and uniform across all sets, Lego is still fundamentally a toy - which will lead to errors with the precision of the robot [6]. An example of this is the motors: there is a somewhat significant degree of freedom/play in the motors, which means that they cant be relied on to move to accurate positions. A worm gear is the ideal solution to this problem: despite reducing the speed, it provides a considerable increase in motor accuracy.

One of the larger issues that this project will encounter is the run-time of the program to find a solution. Many programs have been known to take upwards of three days to find a solve sequence - this project currently aims to find a solution in approximately ten minutes or less. This will require extensive optimisation of search spaces and group theory through symmetrical elimination and set covering in order to reduce the necessary coverage of the search algorithm.

Finding the optimal solve sequence for any given position has been deliberately omitted from the objectives of this project: there are over forty-three quintillion valid positions of a Cube, each requiring an extensive amount of time to find the optimal solution for. Based on current hardware capabilities and previous studies and research, an estimated timespan for finding all optimal solutions is in the order of millennia[1].

[1] See cube20.org table of 20 move or less vs optimal and explain

Chapter 2

Literature Review

> Please forgive me, but to give birth to a machine is wonderful progress. It's more convenient and it's quicker, and everything that's quicker means progress. Nature had no notion of the modern rate of work. From a technical point of view, the whole of childhood is quite pointless.
>
> Mr. Fabry, in Karel Čapek's *Rossum's Universal Robots [7]*

2.1 Robotics

The term robot was first brought to the collective consciousness of the general public by Karel Čapek in 1921 in his play 'Rossum's Universal Robots' [7]. It comes from the Czech for 'forced labour' and was coined by Čapek's brother, Josef, for a short story written some years earlier [8]. One of the main discussions in the field of robotics has been about ethics and morality. This discussion was started in Čapek's play, when a visiting scientist tries to destroy the robot-manufacturing business, in the hope of giving the robots a soul and making them happier. This plan goes awry when the robots become sentient and start to overthrow the humans that created them. Twenty-one years later, the novelette 'Runaround' by Isaac Asimov was featured in the magazine 'Astounding Science-Fiction' [9]. It included his now-prominent Three Laws of Robotics, which were a turning point in the field of robotics and meta-ethics.

The twentieth century generated many ideas about the future of robotics, such as 'Kitchen units will be devised that will prepare "automeals", heating water and converting it to coffee' alongside grandiose aspirations such as 'An experimental fusion-power plant or two will already exist in 2014' [10]. Whilst some predictions have been fulfilled by the technology of the past two decades, others are still nowhere near completion and could be considered impossible for the next century. This is a clear demonstration that the field of Robotics has made vast improvements, but is still - in some respects - very much in its youth.

In the same way as the field of Robotics, Artificial Intelligence (AI) has vast potential and we have only touched the tip of the iceberg. The most advanced consumer-facing AI systems available currently are intended to make people's lives easier through organisation and saving precious seconds in performing simple tasks on their smartphones, and now in their homes. However, Google's Assistant, Apple's Siri, and Amazon's Alexa still fall short when it comes to the intelligence demonstrated in Asimov's book 'I, Robot'. We are still far from being unable to 'differentiate between a robot and the very best of humans' [11].

2.2 Robotics in Education

There has always been a clear use of edutainment style teaching in one form or another for hundreds of years. One of the earliest known uses of edutainment is in 'Poor Richard's Almanack', which was written by Benjamin Franklin and published annually between 1732 and 1758. Alongside all the usual information contained in an Almanac, Franklin (under the pseudonym 'Poor Richard') included maths exercises, puzzles, and aphorisms [12], [13]. Walt Disney was another pioneer of edutainment in the time immediately before, during, and following the World Wars. His first piece of edutainment was a short film, 'Tommy Tuckers Tooth', which was commissioned by a dental institute in 1922.

In the past few decades, there has been a marked change in the form of edutainment in line with advances in technology. We have progressed from paper-based, so called serious games in the 1970s, to early-era video games such as the infamous Oregon Trail; and then in the past decade electronic and robotic kits have made their way into classrooms. One of the leaders of modern edutainment is the Lego Mindstorms kit. The Lego Mindstorms kit has undergone two major revisions since the launch of the first generation of the Mindstorms kit in 1998 (the RCX): in July 2006, the second-generation NXT was released, updating the sensors and adding Bluetooth amongst other improvements; then in September 2013, Lego released the EV3 Home and Education sets. This release cemented Legos position at the forefront of edutainment [14].

2.3 Lego Mindstorms

A study at the University of Calabria in 2009 looked at how Lego Mindstorms worked as a learning tool when used in team-building type tasks [15]. 28 students were divided into 6 groups, each given a Lego robot and preliminary training which covered the Lego Mindstorms and programming environment. The groups were all given the same task, and their progress studied throughout the sessions. When studying the building and programming as two separate sub-tasks, three main categories of work subdivision were found: each group member had a set role in building, but all shared the programming; both sub-tasks were equally divided amongst members; each member had a set role in either sub-task, and there was a clear leader. This closely follows real-world teamwork mechanics and scenarios, and the use of robots was found to stimulate the student into exploring and sharing critical knowledge within the group. The study's conclusion was that Lego Mindstorms - and inherently variable morphology robots - stimulates further process analysis, information selection, and to observe and experiment with the consequences of their actions [15].

The Lego Mindstorms native programming environment is visual-based, with drag-and-drop code blocks used to form programs which can run natively on the main 'brick'. This provides a simplistic user experience and allows novice programmers to create complex programs with relative ease. Drag-and-drop programming can drastically reduce the amount of syntax errors in a program, allowing programmers to better focus on the functionality of their program [16].

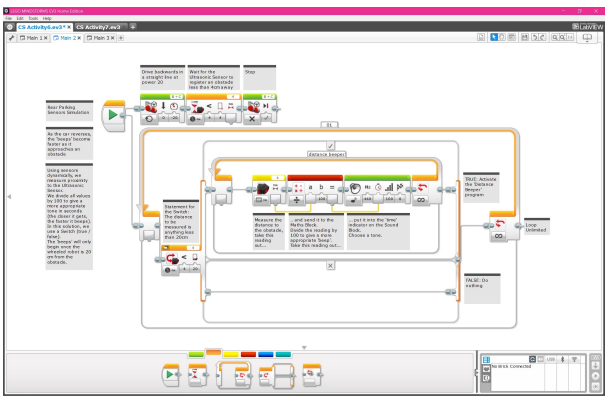


Figure 2.1: A sample program in 'EV3 Programming Software'

2.4 Solving Puzzles with Algorithms

When it comes to playing games and solving puzzles, Googles AlphaGo Zero is the clear winner. Whilst Googles first Go-playing program defeated the world champion at Go, it took months of supervised learning from experts and reinforcement learning from self-play. The new version started tabula rasa and was completely self-taught - it had no interaction with any humans, and uses no historical data. Through reinforced learning alone by simulating games against itself, it took three days to reach the level of AlphaGo and in forty days had surpassed any Go players performance - human or artificial [17], [18].

When a computer solves a Cube (or any other similar puzzle), it does so by following set algorithms and rules until the solved state is achieved. Computers cannot use humanlike instincts, so we often try to provide a heuristic to make the task simpler. These heuristics can be pre-computed data tables to reduce the amount of processing at run time, or a reduction in the original data set through elimination of certain data sets.

In his 1997 paper on the use of pattern databases to increase solve efficiency, Richard Korf uses different heuristic functions and characterises how effective they are in reducing the number of moves in a solve sequence. His first method used an Iterative-Deepening A* algorithm, combined with the heuristic of the Manhattan distances from the edge cubies current position and orientation to that which is desired. This reduced the time required to solve a Cube at depth-14 to three days[1], however when increased to depth-18 the time increased exponentially to two hundred and fifty years. After a re-evaluation, Korf modified the heuristic by pre-computing the Manhattan distance of each cubie from all of its possible positions and orientations and storing the table in memory at run-time. This created a table of nearly ninety-million entries - which would have been bigger but was restricted by the available memory (the table was approximately forty-two megabytes). The newer heuristic reduced the time to search at depth-18 to less than four weeks - a reduction of approximately 99.97%. Korf suggested that the speed of the algorithm used would increase linearly with the amount of memory available, meaning that if it were run today the depth-18 search would take under three hours[2] [19].

Table 2.1: Morwen Thistlethwaite's five groups for his algorithm [20]

Group Denotation	**Mathematical Representation**	**Summary**
G_0	$\langle L,\ R,\ F,\ B,\ U,\ D\rangle$	All valid positions
G_1	$\langle L,\ R,\ F,\ B,\ U2,\ D2\rangle$	All positions that can be reached with quarter turns of the $L,\ R,\ F,\ B$ faces and half turns of the $U,\ D$ faces
G_2	$\langle L,\ R,\ F2,\ B2,\ U2,\ D2\rangle$	Positions reachable from quarter turns of $L,\ R$ faces and half turns of the $F,\ B,\ U,\ D$ faces
G_3	$\langle L2,\ R2,\ F2,\ B2,\ U2,\ D2\rangle$	Positions only reachable from half turns of any face
G_4	$\{I\}$	The goal state

There have been many historical landmark algorithms since the Rubiks Cubes inception in 1974. One of the earliest instances was created by Morwen Thistlethwaite circa 1981 and is based on group theory. Thistlethwaites algorithm splits all possible positions of the cube into five groups of decreasing size as seen in Table 2.1. David Singmaster, Thistlethwaites colleague at London South Bank University (then called Polytechnic of the South Bank), describes the methodology of the algorithm: 'Once in G_i, one only uses moves in G_i to get into G_{i+1}. The ratio $\frac{|G_i|}{|G_{i+1}|}$ is called the index of G_{i+1} in G_i.' Using this algorithm, Thistlethwaite proved that the maximum number of moves required to solve any valid position is fifty-two [20]. This was the first development of Gods Number, so named because it is the number of moves that an omniscient being would use to solve a Cube in the most efficient manner without failure.

[1] The simulation was run on a Sun Ultra-Sparc Model 1 workstation

[2] This is based on the memory capacity of the Model Workstation and a modern computer being 64MB and 16GB respectively.

2.5 God's Number

The lower bound for Gods Number was recognised to be eighteen moves through analysis of the number of sequences of seventeen moves or fewer, and the discovery that there were more Cube positions than seventeen-moves-or-fewer sequences. In 1995, the lower bound was increased to twenty by Michael Reid upon discovering that the 'Superflip' position requires twenty moves to solve. He included his findings in an email to the Cube-Lovers mailing list, 'superflip is now known to require 20 face turns . . . this is the first improvement to the lower bound (18f) given by a simple counting argument' [21].

The next refinement to the upper bound of Gods Number came from Dutch Mathematician Hans Kloosterman in December of 1989. Kloostermans findings were published in a newsletter 'Cubism for Fun' (CFF) as an article titled 'Rubiks Cube in 44 Moves'. In this article Kloosterman discusses his existing solution to the Magic Domino, a subset of the Rubiks Cube, and how he has adapted it into a more efficient Cube solution with the help of Thistlethwaites algorithm. Kloosterman reduced the estimation of Gods Number by adapting Thistlethwaites G_3 into a subgroup of G_2 where all of the U-cubies are on the U face and all the D-cubies are similarly on the D face. This reduces the index $\frac{|G_2|}{|G_3|}$ form 15 to 8 (whilst also reducing the $(G_1 : G_2)$ index by 3 and increasing the $(G_3 : G_4)$ index by 1), creating a net decrease of 8 moves when compared to Thistlethwaites algorithm. 'Rubik's Cube in 44 Moves' ended with an editors note stating that not all of the details of the algorithm were certain and more (including a more efficient algorithm) would be revealed in a later issue [22].

Sure enough, a year later Kloosterman wrote an article titled 'Rubiks Cube in 42 Moves' in the twenty-fifth issue of CFF. This stated that the new algorithm was in fact the final version of Kloostermans forty-four move algorithm. In the same way as its predecessor, this article is split into four stages to solve a Cube. The first three stages are based on 'solving' a Cube with only certain cubies coloured - the positions of the other remain irrelevant until a later stage. The final stage is solving the Cube in a non-particular manner, and is calculated to take no more than eighteen moves [23].

In the following twenty years, the upper bound was gradually refined to match the lower bound - meaning that Gods Number is exactly twenty. The final development came from a group of four Rubiks Cube enthusiasts in July of 2010, and was only achieved through a brute force method: the four-man team first took every possible position of a Cube and reduced it from forty-three quintillion positions down to just over one quintillion - still a vast number, but a reduction by a factor of forty-three - by using rules of symmetry and mirroring. They then wrote a program to find solve sequences of length twenty or less, and ran it on a 'large number of computers at Google' to find all solve sequences in a few weeks. The parallel computation took the equivalent of thirty-five years [5].

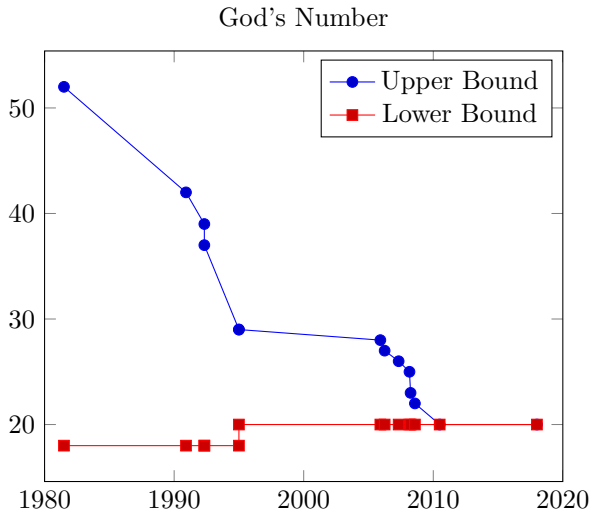


Figure 2.2: The upper bound was decreased to 20 over the course of twenty-nine years and across thirteen separate studies [5]

2.6 Python as a Scientific Scripting Language

Traditionally when performing large calculations and processes, compiled languages (C, C++, Fortran etc.) are the preferred choice over scripting languages such as Ruby, JavaScript, (pure) Python, etcetera. This is because compilation is done before run-time, allowing the program to run un-hindered by on-the-fly translation/interpretation [24].

For general purpose applications, scripting languages are preferable because they allow small changes to be made without re-compiling large classes before testing. In the thirteenth issue of 'Scientific Programming', Cai, Langtangen and Moe [24] explain how MATLAB is often a preferable choice for less intensive scientific computation due to its many features, such as: integrated simulation and visualisation tools; clear syntax; immediate feedback of commands; impressive documentation. They describe the use of MATLAB in computational science as paradoxical, as it is a scripting language which inherently is interpreted at run-time.

One of MATLABs biggest selling points, the IDE with integrated support for matrices, graphs and other mathematical features, can be replicated in a few Python IDEs or with the addition of Python packages. Pythons main core can be expanded massively by using any of the tens of thousands of available packages, which can provide any of MATLABs advantages alongside extra flexibility and capabilities. One of the most popular packages, Numerical Python (NumPy), adds support for most scientific calculations. It expands Pythons inbuilt data structures to use multi-dimensional homogenous arrays of any Python data type. NumPy takes Python from a well-structured general-purpose language to a powerful mathematical language which can be applied to many different tasks [24], [25].

As well as the multitude of available packages, Python has another major tool in its arsenal: it can be extended with a compiled language by design. This means that processor-intensive tasks like nested for loops can be migrated to a compiled language such as C++ or Fortran, increasing run speed up to a factor of ten. This extension process is now a trivial matter thanks to automated tools such as 'f2py', which allows for automated calls to compiled Fortran code [25]. In conclusion, given Pythons wide compatibility, clear high-level syntax, and extendibility to compiled code for more complex calculations, it is the ideal language for this project.

2.7 Conclusion

Chapter 3

Requirements and Analysis

> I have yet to see any problem, however complicated, which when you looked at in the right way, did not become still more complicated.
>
> Paul Anderson, New Scientist [26]

3.1 Chapter Introduction

This chapter covers the specific requirements needed for the project to succeed: the hardware required to build the robot and run the solving program; the software needed to write and run the program; and the specific tasks to complete the project. As well as the requirements, an in-depth analysis of the project and its goals is made, discussing their feasibility, difficulty, and how necessary they are for the success of this project.

3.2 Specific Aims and Objectives

Table 3.1: The objectives, with relative difficulty and value

Objective	**Weightings**	
	Difficulty	**Value**
1. Build a robot which can move a Cube to a sufficient degree of accuracy. This is the primary objective for this project, and is imperative to the projects success. As such it will be the first objective to be completed. This task, despite being of a medium difficulty, is fairly straightforward by nature: the robot simply needs to manipulate a Cube in any valid move.	10%	20%
2. Convert human-compatible move sequences to robot-compatible sequences. This objective is non-essential to the projects success because all of the analysis could be performed using the robots capabilities as a constant frame of reference. This, however, would require a manual translation of the pre-existing algorithms into robot-compatible algorithms and as such the translator is of high value in this project.	20%	10%

Objective	Weightings	
	Difficulty	**Value**
2.1 Move the Cube in a given (robot-compatible) sequence. Once a robot-compatible move sequence has been generated, the robot must move the Cube in this exact sequence with no errors or move-failures. If a single move is performed incorrectly then the entire move sequence is inherently moot.	5%	10%
3. Implement a system which successfully generates a solve sequence for any given position. This is the core goal of the project, and take precedence above all others. The program will take certain arguments to define the parameters of the algorithm and output a solve sequence. This may take a substantial amount of time to complete and this must be a consideration on completing this task.	40%	30%
4. Ensure the runtime of the system is an acceptable length. Some previous attempts at finding optimal solve-sequences have had estimated run times in the order of tens of weeks. Clearly this is unacceptable or the scope of this project and thus the runtime of the project will have a target of sub-fifteen minutes, with the intention of bringing it under five minutes.	10%	10%
5. Implement a program to use other algorithms to generate a solve sequence. This objective makes up a significant part of this project. Different algorithms presented by notable persons will be implemented and measured by the robot.	10%	10%
6. Compare the performance of different algorithms, especially the difference between human-compatible and robot-compatible move sequences. Once all algorithms have been run and measured, their performance will be analysed and compared. In this context, performance comprises the efficiency in finding the solve sequence and the length of the solve sequence. Algorithms for generating a human-compatible sequence will also be compared against those that generate robot-compatible sequences.	5%	10%

3.3 Analysis

3.3.1 Software Requirements

Python 3.6 Runtime Environment

Python is the ideal language for this project, as discussed in the Literature Review, and requires a runtime environment for programs to be run. This consists of the Python interpreter, libraries, and any packages used in the development process.

ev3dev

ev3dev [27] is a custom operating system (OS) for the EV3 which s contain a low-level framework for the peripherals of an EV3. It allows users to write their own programs in a multitude of languages (including Python) to create complex functions and models. It is installed by flashing the ev3dev OS to a microSD card which is then inserted into the EV3, avoiding affecting the EV3s original firmware.

A Python wrapper is available on GitHub [28] for the ev3dev OS, allowing the creation of Python programs with packaged support for the actuators and sensors of the EV3.

PyCharm

Python doesnt require an Integrated Development Environment (IDE) for developing or running programs, however using an IDE makes both of these tasks, and testing, much easier. The program(s) for this project will be written using PyCharm from JetBrains [29]. PyCharm provides strong code completion abilities, project-wide refactoring, and software development kit (SDK) modification amongst many other abilities. This will allow more time to be allocated to tasks of higher value rather than struggling with relatively minor issues.

Software-Hardware Comparisons

The below tables show the requirements of the software to be used in the creation of this project, and the capabilities of the computer that they will be running on. This includes both the Main Development Computer and also the Lego EV3.

Table 3.2: Software requirements for the EV3 and its capabilities

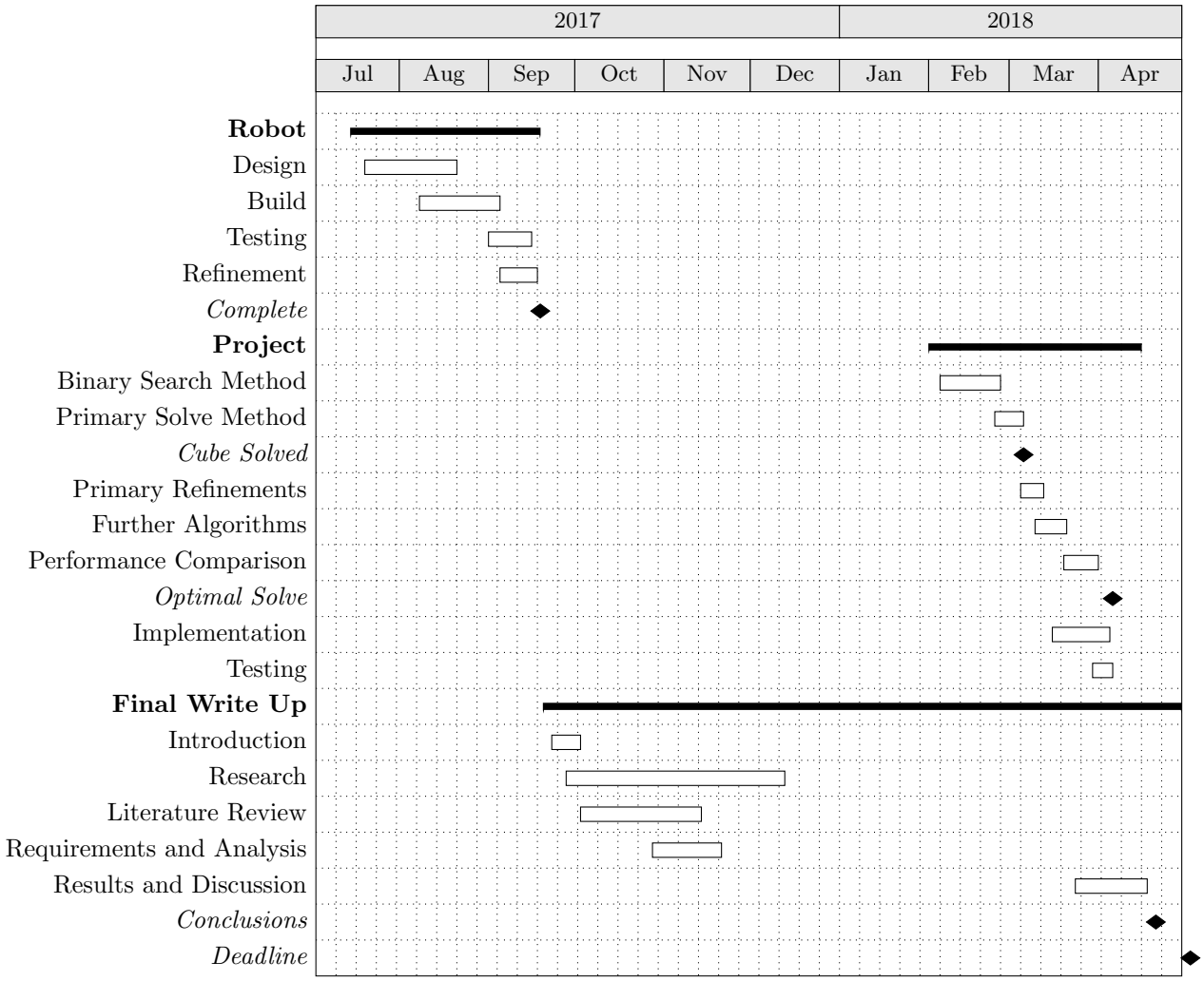
Software/Hardware	**Requirements/Capabilities**		
	Processor	**Memory**	**Connectivity**
ev3dev	300MHz	64MB	WiFi/Bluetooth/USB
Lego EV3	*300MHz*	*64MB*	*1 x USB Port, Bluetooth, WiFi (via USB dongle)*

Table 3.3: Software requirements for the main development computer and its capabilities

Software/Hardware	**Requirements/Capabilities**			
	OS	**Processor**	**Memory**	**Connectivity**
Lego EV3 Programming Software[30]	Windows 2003 (or newer)	Dual Core 2GHz	2GB	1 USB Port
Python 3.6	Windows Vista (or newer)	Program Dependent	Program Dependent	
PyCharm	Windows 2003 (or newer)		1GB minimum 2GB recommended	
Main Development Computer	*Windows 10 Pro*	*Quad Core i5 3.6GHz*	*32GB*	*WiFi, Bluetooth, 12 x USB Port*

Evaluative Techniques

The progress of this project is estimated to follow the below Gantt Chart. For the duration of its lifespan, there will be a continual measurement of the actual progress against predicted.


2017
2018
Jul
Aug
Sep
Oct
Nov
Dec
Jan
Feb
Mar
Apr
Robot
Design
Build
Testing
Refinement
Complete
Project
Binary Search Method
Primary Solve Method
Cube Solved
Primary Refinements
Further Algorithms
Performance Comparison
Optimal Solve
Implementation
Testing
Final Write Up
Introduction
Research
Literature Review
Requirements and Analysis
Results and Discussion
Conclusions
Deadline

Chapter 4

Design

> There are two ways of constructing a software design: One way is to make it so simple that there are obviously no deficiencies, and the other way is to make it so complicated that there are no obvious deficiencies. The first method is far more difficult.
>
> *C.A.R.Hoare[31]*

4.1 Robot

4.1.1 Cube Manipulation

4.1.2 Move Sequence Translation

4.2 Codebase

4.2.1 Object Classes

Cube

Position

4.2.2 Solve Methods

Group Based

Tree Based

4.2.3 Database

Chapter 5

Implementation and Testing

5.1 Building the Robot

5.1.1 The Color Scanner

5.1.2 The Cradle

5.1.3 Z-Move Arm

5.2 Coding a Solver

5.2.1 Group Theory Method

Group Generation

5.2.2 Tree Method

Tree Generation

Chapter 6

Results and Discussion

Chapter 7

Conclusions

List of Figures

List of Tables

Bibliography

[1] O. Waxman, "The 13 Most Influential Toys of All Time," 2014.

[2] E. Rubik, "The Perplexing Life of Erno Rubik," *Discover Magazine*, p. 81, mar 1986.

[3] Rubik's Cube, "Rubik's UK Website," 2017.

[4] C. Chan, "This Guy Can Solve 3 Different Rubik's Cube While Juggling Them at the Same Time," 2016.

[5] T. Rokicki, H. Kociemba, M. Davisdson, and J. Dethridge, "cube20," 2010.

[6] R. T. Cook and S. Bacharach, *Lego and Philosophy: Constructing Reality Brick by Brick*. 2017.

[7] K. Čapek, *Rossum's Universal Robots*. ebooks@Adelaide, 1921.

[8] Etymonline, "Robot," 2017.

[9] I. Asimov, "Runaround," *Astounding Science-Fiction*, pp. 94–103, 1942.

[10] I. Asimov, "Visit to the World's Fair of 2014," aug 1964.

[11] I. Asimov, *I , Robot*. 1970.

[12] G. Beato, "Turning to Education for Fun," 2015.

[13] B. Franklin, *Poor Richard's Almanack*. 1732.

[14] K. Becker, "The Edutainment Era - A Look at What Happened and Why."

[15] E. Bilotta, L. Gabriele, R. Servidio, and A. Tavernise, "Edutainment robotics as learning tool," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 5940 LNCS, pp. 25–35, 2009.

[16] C. Kelleher, D. Cosgrove, and D. Culyba, "Alice2: programming without syntax errors," *Proceedings of the 15th Annual Symposium on the User Interface Software and Technology*, pp. 3–4, 2002.

[17] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, and L. Sifre, "Article Mastering the game of Go without human knowledge," *Nature Publishing Group*, vol. 550, no. 7676, pp. 354–359, 2017.

[18] R. Cellan-Jones, "Google DeepMind: AI becomes more alien," oct 2017.

[19] R. E. Korf, "Finding Optimal Solutions to Rubik's Cube Using Pattern Databases," *American Association for Artificial Intelligence (AAAI)*, pp. 700—-705, 1997.

[20] D. Singmaster, *Notes on Rubik's 'Magic Cube'*. Enslow, 1981.

[21] M. Reid, "Cube Lovers Mailing List," tech. rep., 1995.

[22] H. Kloosterman, "Rubik's Cube in 44 Moves," *Cubism For Fun*, vol. December, no. 22, pp. 9–11, 1989.

[23] H. Kloosterman, "Rubik's Cube in 42 Moves," *Cubism For Fun*, vol. December, no. 25, pp. 19–22, 1990.

[24] X. Cai, H. P. Langtangen, and H. Moe, "On the performance of the Python programming language for serial and parallel scientific computations," *Scientific Programming*, vol. 13, no. 1, pp. 31–56, 2005.

[25] T. E. Oliphant, "Python for Scientific Computing," *Marine Chemistry*, pp. 10–20, 2006.

[26] P. Anderson, "No Title," *New Scientist*, 1969.

[27] Ev3dev.org, "ev3dev Home."

[28] Ev3dev and R. Hempel, "ev3dev-lang-python."

[29] JetBrains, "PyCharm: Python IDE for Professional Developers by JetBrains."

[30] Lego, "EV3 Programming Software," 2017.

[31] C. A. R. Hoare, "The Emperor's Old Clothes," *Communications of the ACM*, vol. 24, no. 2, 1981.