# Solving Rubik's Cube with Lego Mindstorms

Will Garside

03-05-2018

# I 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:

# II 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

# III Acknowledgements

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

# IV Definitions and Notations

Table IV.1: Terminology used in this document

| **Abbreviation/Word** | **Definition** |
|---|---|
| [Rubik's] Cube | A standard 3x3x3 Rubik's Cube[1]. |
| Face | One of the size faces on a Cube. Denoted by position (UDLRFB) 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. |
| Facelet | One of the fifty-four coloured squares on the sides of the 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.<br>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 IV.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 Rubik's 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

# 1 Introduction

> 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?
>
> Ernö Rubik [1]

## 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 [2]. It was later re-branded to '*Rubik's Cube*' to overcome an oversight involving patenting and copyrighting the design.

In an unpublished manuscript [1], 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, despite having been proved mathematically, is still little more than speculation: God's Algorithm. The theory behind this 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 four 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.

## 1.3 Limitations and Constraints

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 can't 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 fifteen minutes or less (an arbitrary number determined only by the attention span of the author). 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]This is an estimation based on data presented at cube20.org which states that finding optimal solutions takes 11,800 times as long as finding sequences or twenty moves or less [7]

# 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 [8]*'

## 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*' [8]. It comes from the Czech for 'forced labour' and was coined by Čapek's brother, Josef, for a short story written some years earlier [9]. 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, as is the norm in fiction surrounding Artificial Intelligence (AI), 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*' [10]. 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' [11]. 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, AI has vast potential and we have only touched the tip of the iceberg. The most advanced commercial AI systems are intended to make people's lives easier through organisation and saving precious seconds in performing simple tasks on their smartphones and more recently 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' [12].

## 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 [13], [14]. 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 Tucker's 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.

Robotics has been described as 'the most effective way of motivating and supporting the study of many areas', and has also been shown to aid the development of social and teamwork skills [15]. In the past decade or so, robots and electronic kits have been a common extra-curricular activity and are slowly being used as teaching aids and for practical sessions in actual lessons.

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 Lego's position at the forefront of edutainment [16].

## 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 [17]. Twenty-eight students were divided into six 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 other variable morphology robots - stimulates further process analysis, information selection, and to observe and experiment with the consequences of their actions [17].

The Lego Mindstorms' native programming environment is makes use of drag-and-drop code blocks 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 [18].

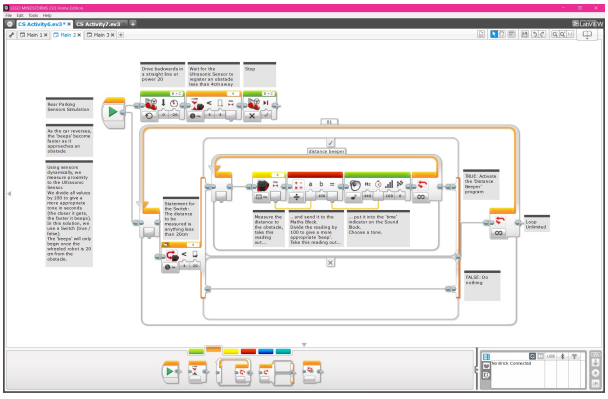


Figure 2.1: A sample program in '*EV3 Programming Software*'

## 2.4 Solving Puzzles with Algorithms

When it comes to playing games and solving puzzles, Google's '*AlphaGo Zero*' is the clear winner. Whilst Google's 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 used 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 player's performance - human or artificial [19], [20].

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 by eliminating certain elements according to a set of rules.

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] [21].

Table 2.1: Morwen Thistlethwaite's five groups for his algorithm [22]

| **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 inception of Rubik's Cube in 1974. One of the earliest instances was created by Morwen Thistlethwaite circa 1981 and is based on group theory. Thistlethwaite's algorithm splits all possible positions of the cube into five groups of decreasing size as seen in Table 2.1. David Singmaster, Thistlethwaite's 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 [22]. This was the first development of God's 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 1 Workstation and a modern computer being 64MB and 16GB respectively.

## 2.5 God's Number

The lower bound for God's Number was originally 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... given by a simple counting argument' [23].

The next refinement to the upper bound of God's Number came from Dutch mathematician Hans Kloosterman in December of 1989. Kloosterman's findings were published in a newsletter '*Cubism for Fun*' (CFF) as an article titled '*Rubik's Cube in 44 Moves*'. In this article Kloosterman discusses his existing solution to the Magic Domino, a subset of Rubik's Cube, and how he has adapted it into a more efficient Cube solution with the help of Thistlethwaite's algorithm. Kloosterman reduced the estimation of God's Number by adapting Thistlethwaite's $G_3$ 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 $\frac{|G_2|}{|G_3|}$ from 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 Thistlethwaite's algorithm. '*Rubik's Cube in 44 Moves*' ended with an editor's 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 [24].

Sure enough, a year later Kloosterman wrote an article titled '*Rubik's Cube in 42 Moves*' in the twenty-fifth issue of CFF. This stated that the new algorithm was in fact the final version of Kloosterman's 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 [25].

In the following twenty years, the upper bound was gradually refined to match the lower bound - meaning that God's Number is exactly twenty. The final development came from a group of four Rubik's 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].

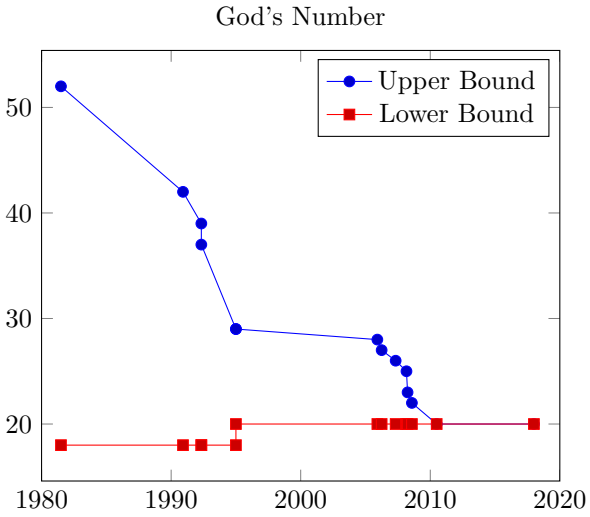



Figure 2.2: The upper bound was decreased to 20 over the course of twenty-nine years and across thirteen separate studies [5]

## 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 unhindered by on-the-fly translation/interpretation [26].

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 [26] 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 MATLAB's 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. Python's main core can be expanded massively by using any of the tens of thousands of available packages, which can provide any of MATLAB's advantages alongside extra flexibility and capabilities. One of the most popular packages, '*Numerical Python*' (NumPy), adds support for most scientific calculations. It expands Python's inbuilt data structures to use multi-dimensional homogeneous 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 [26], [27].

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 [27].

## 2.7 Conclusion

Lego Mindstorms is a strong candidate for the construction of the robot: its variable morphology allows adaptation to a range of tasks; the uniformity between pieces means that the main EV3 kit can be supplemented with other kits to extend its ability; and previous studies have shown that it is ideal for stimulating analysis and observations. Instead of using a single algorithm to solve the Cube, a range will be implemented in order to find the strongest method for finding a solve sequence. This will allow cross-comparison between, and analysis of, individual algorithms with the intention of finding the algorithm best suited to this project. Tree and group based algorithms alike will be implemented.

MATLAB is incompatible with the Lego EV3, so will not be used for this project. Python's wide compatibility, clear high-level syntax, and extendibility to compiled code for more complex calculations, makes is the ideal language for this project. The main program will be written in Python, with any intensive processing written in C++ to improve the speed of the algorithms.

# 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 [28]

## 3.1 Introduction

This chapter covers the specific requirements needed for the project to succeed: the particular tasks to complete the project; the hardware required to build the robot and run the solving program; and the software needed to write and run the program. 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

1. Build a robot which can manipulate a Cube accurately

   This is the first of the two primary objectives for this project, and is imperative to the project's 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.

2. Implement a system which successfully generates a solve sequence for any given position.

   The second primary objective ensures that any of the forty-three quintillion positions of Rubik's Cube can be solved correctly. The time taken to generate the solve sequence and the actual length of the solve sequence will both be used when measuring an algorithm's success.

3. When the robot is provided with a move sequence...

   (a) Convert any input sequence to be robot-compatible

   All of the pre-existing algorithms return a solution which a human being can follow to solve a real-world Cube. In the context of this project the manipulation is done by the robot, which cannot perform all of the standard moves, so any sequences provided must be translated to one which the robot can follow.

   (b) Correctly follow the sequence to achieve the intended output

   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 will produce an incorrect output.

4. Ensure the runtime of the system is an acceptable length

   Previous solve sequence generator algorithms have had a run-time in the order of weeks: the desired run-time of the generation by this project will be in the order of minutes. This will require careful analysis of the trade off between generator efficiency and solve sequence length.

5. Implement a program to use a range of algorithms to generate a solve sequence

   In order to effectively compare the performance of several algorithms, there must be a method of running them under the same conditions and with the same overheads (e.g. transmission time to the robot). For this reason, there will be one main program with an option for the algorithm to use.

6. Compare the performance of different algorithms, especially the difference between human and robot compatible move sequences

   The performance of each algorithm will be tracked and compared fairly to show each one's merits and faults. This will allow the choice of one algorithm to be improved to create the most efficient method - or the potential combination of algorithms to form a stronger one.

## 3.3 Analysis

### 3.3.1 Software

#### PyCharm/CLion

An Integrated Development Environment (IDE) makes development, running programs, and testing and debugging much easier than using a simple text editor and command line. PyCharm and CLion by JetBrains [29] will be used for developing in Python and C++ respectively. PyCharm and CLion provide 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.

#### Python 3.6 Runtime Environment

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

#### MinGW

C++ requires an environment to develop and compile in. MinGW (Minimalist GNU for Windows) is a popular choice when developing C++ applications.

#### ev3dev

ev3dev [30] is a custom operating system (OS) for the EV3 which contains 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 EV3's original firmware.

A Python wrapper is available on GitHub [31] for the ev3dev OS, allowing the creation of Python programs with packaged support for the actuators and sensors of the EV3.

### 3.3.2 Software Requirements

This table shows 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. Any

Table 3.1: Software requirements for the main development computer and its capabilities

| **Software** | **Requirements/Capabilities** | | | |
|---|---|---|---|---|
| | **Minimum OS** | **Processor** | **Memory** | **Connectivity** |
| PyCharm | Windows 2003 (or newer) | | 1GB minimum<br>2GB recommended | |
| CLion | Windows 7 64-bit (or newer) | | 2GB minimum | |
| Lego EV3 Programming Software[32] | Windows 2003 (or newer) | Dual Core 2GHz | 2GB | 1 USB Port |
| Python 3.6 | Windows Vista (or newer) | | | |
| MinGW | | | | |
| *Main Development Computer* | *Windows 10 Pro 64-bit* | *Quad Core i5 6500 3.6GHz* | *32GB* | *WiFi, Bluetooth, 12 × USB Port* |

## 3.4 Feasibility Study

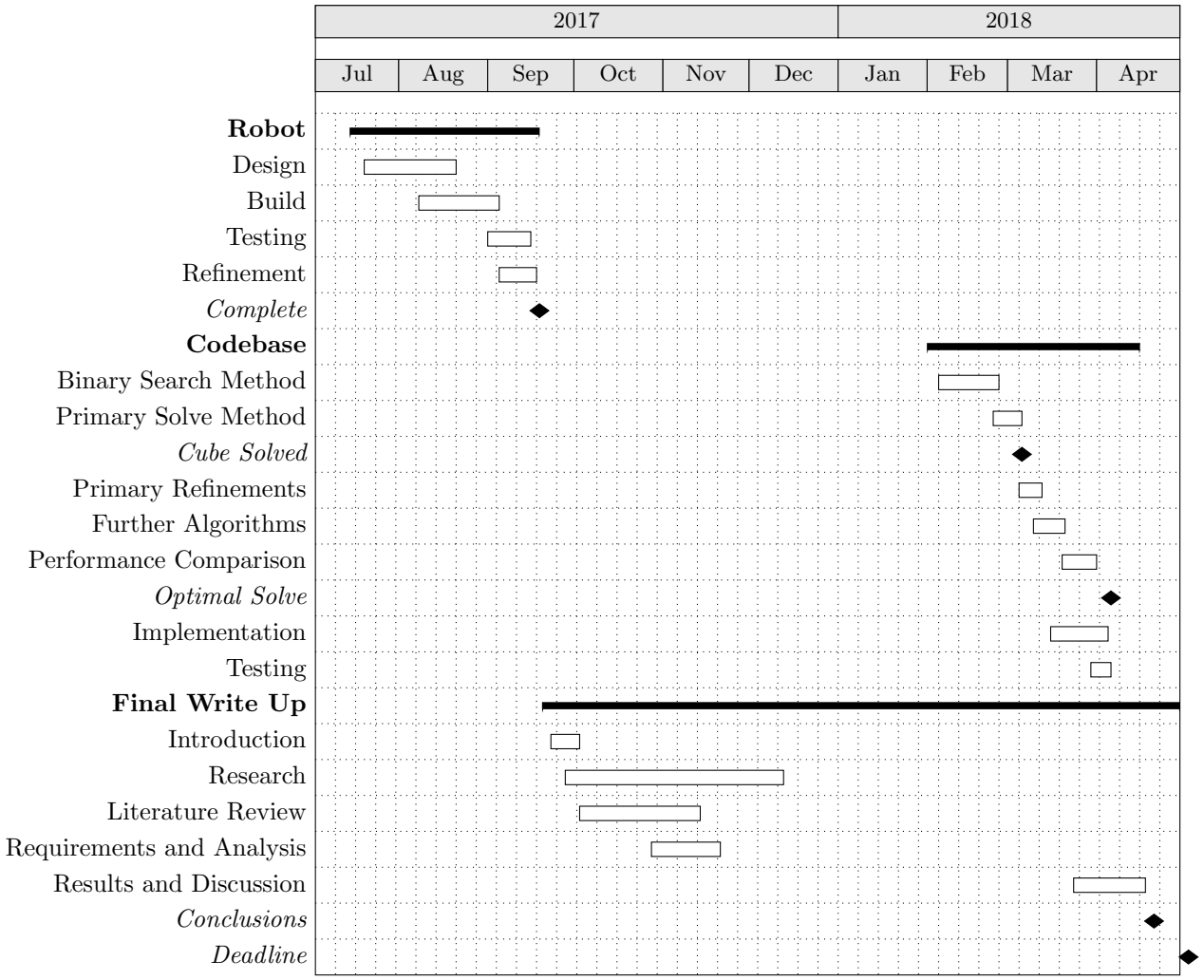
### 3.4.1 Optimal Solutions

### 3.4.2 Tree Based Methods

### 3.4.3 Group Based Methods

## 3.5 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
Codebase
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

# 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[33]

## 4.1 Robot

### 4.1.1 Sensors and Actuators

The Lego EV3 comes with three DC motors (two large (#**45502**) and one medium (#**45503**)), a colour sensor and a touch sensor amongst other peripherals. The medium motor will be used to move the colour sensor to scan the Cube, and the large motors to rotate the Cube about the horizontal and vertical axes respectively. The quantity of the motors will be the largest restriction in designing the robot: the robot will only be able to rotate a Cube about two of the three axes; and the colour sensor will only have one degree of freedom so will rely on the rotation of the Cube to scan all of the facelets.

### 4.1.2 Cube Manipulation

#### Cradle

The Cube will sit in a cradle which rotates about the Y axis. This will provide the movement needed for the moveset $\{Y, Y', D, D'\}$. This will have an inner dimension of 7 studs × 7 studs (56 mm × 56 mm) to provide a close match to the dimensions of the Valk 3: 55.5 mm × 55.5 mm. The 0.5 mm tolerance means that the Cube doesn't have to perfectly aligned when it is rotated - a difficult task considering the play in the motors. The cradle must rotate accurately to 90°increments to ensure the success of the X rotations and avoid the Cube falling out of the cradle.

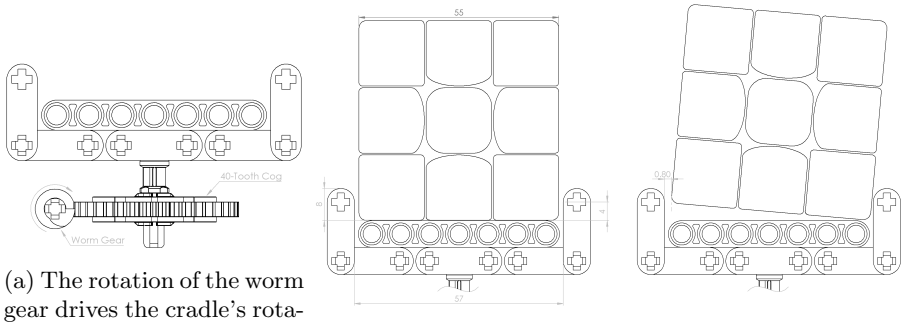
The play in the motor will reduce the accuracy of the cradle, as it effectively introduces a zero error for each rotation. If the cradle only rotates in one direction then the zero error would only have to be offset at the very start of the program because the cradle would 'sit' at one end of the region of play. However, this will reduce the moveset by 50% and thus have a worst case runtime increase of 100% (Sequentially, $|D'X| = 2 \rightarrow |DDDX| = 4$).

A worm gear (#**4716**) will solve both of these problems: it will be combined with a large gear (#**3649**) to create a gear ratio of 1:40, increasing the accuracy of the motor by a factor of forty (see Figure 4.1a). This will also slow the maximum rotational velocity of the cradle to 0.209rad/s (2RPM) at 7.5V, so a 90°turn will take 7.5 seconds[1]. A compromise between maximum rotational velocity and accuracy will be found by adding more gears to reduce the ratio. The problem presented by the play in the motors will be reduced to an acceptable level due to the unidirectional nature of the gear train.

The sides of the cradle will only be 1 stud high, but will have a semi-circular profile to allow the Cube to rotate out of the base of the cradle as demonstrated in Figures 4.1b and 4.1c. The curved edge will also

[1]This figure is the result of a measurement of the maximum rotational velocity of a large motor, and will fluctuate depending on the voltage supplied by the batteries in the EV3 brick

cause the Cube to slide back into the cradle completely, rather than resting on the upper corner of the edge piece. This movement back into the cradle after an X-axis rotation will be assisted by the X-Move Arm.




(a) The rotation of the worm gear drives the cradle's rotation

(b) Resting in the cradle

(c) Starting an X rotation

Figure 4.1: A cross-section of the cradle and Cube

### X-Move Arm

To rotate the Cube about the X axis, a rotating arm will push the Cube up and out of the cradle and cause it to pivot about the edge of the cradle. The Cube will then slide back into the cradle to complete the rotation. A large motor will be directly connected to the shaft which the arm will rotate about, with no need for gears. One end of the arm will have a wheel with a rubber tire to provide enough friction to tip the Cube over, and the other side will have a set of guards which can hold the top two slices of the Cube in place when performing a $D$ or $D'$ move. The design of the swing arm and its positioning relative to the cradle can be seen in Figure 4.2.

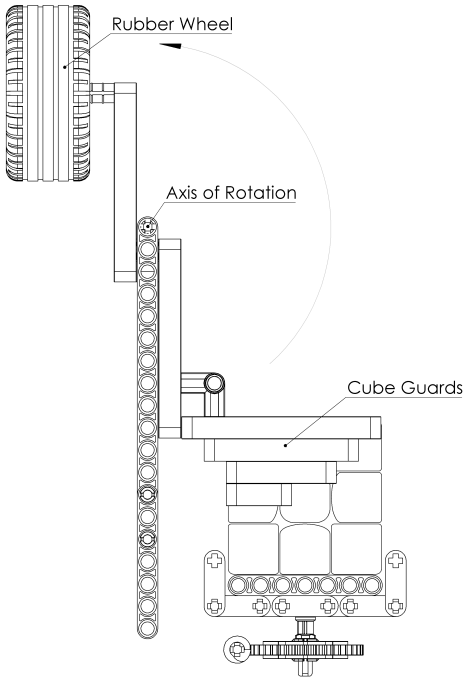



Figure 4.2: The rotating arm used to flip the Cube

## 4.1.3 Color Sensor

The colour sensor will be attached to the end of an extending arm, which will be moved by a rack and worm gear to ensure the sensor can be accurately placed above the correct facelet. The worm gears will

be driven by the medium sized motor, which produces less torque than the large motors but has a larger maximum rotational velocity - making it ideal for overcoming the speed reduction of the worm gears. The colour sensor will be in a fixed position relative to the arm, and will be centred on the Cube's x-axis. In order to scan a full Cube, the sequence $XXXYXXX$ will be followed, and the Up side scanned every time a new face is seen as shown by algorithms 2 and 1.

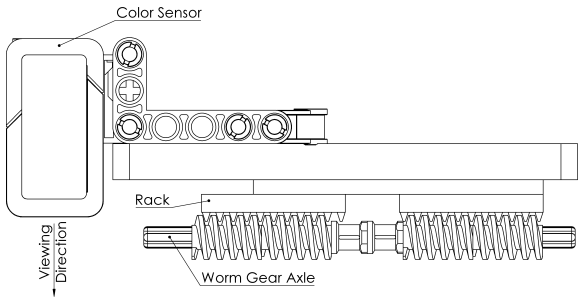



Figure 4.3: The colour sensor mounting arm is driven by a set of worm gears

**Algorithm 1** Scanning the Up-Face of a Cube

```
1:  function ScanUpFace
2:      faceletList ← [ ]                          ▷ Initialise as empty list
3:      for i ← 0 to 4 do                          ▷ 4 Corner-Edge pairs
4:          MoveColorSensor(edgeCubie)
5:          faceletList ← colorValue
6:          RotateCradle(45)
7:
8:          MoveColorSensor(cornerCubie)
9:          faceletList ← colorValue
10:         RotateCradle(45)
11:     end for
12:     MoveColorSensor(centreCubie)
13:     faceletList ← colorValue
14:     return faceletList
15: end function
```

**Algorithm 2** Scanning an entire Cube

```
1:  function ScanCube
2:      sides ← [ [ ], [ ], [ ], [ ], [ ], [ ] ]   ▷ Initialise as 6 empty lists
3:      for i ← 0 to Length(sides) do
4:          sides[i] ← ScanUpFace
5:          if i == 3 then
6:              RotateCradle(90)                   ▷ Allows Left/Right sides to be scanned after X-Move
7:          end if
8:          if i < 5 then
9:              XMoveCube                          ▷ X-Move the Cube to get a new Color on the Up-Side
10:         end if
11:         if i == 4 then
12:             XMoveCube                          ▷ Extra X-Move to get to Down-Side
13:         end if
14:     end for
15:     return sides
16: end function
```

### 4.1.4 3D Model

The drawings above show how each component will work and be built out of Lego, however a design for the whole robot is difficult to represent in 2-dimensional drawings. For this reason, a 3-dimensional model of the robot was created in BrickLink's Stud.io software [34] to create a full model representative of the final design. The components drawn above are rendered below for comparison and clarification on specific details. Some parts may be hidden in the renders to show a clearer view.

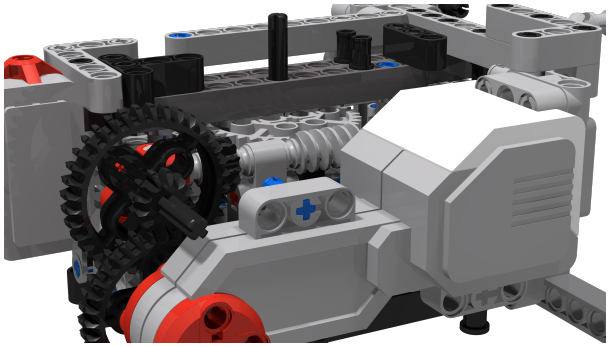


Figure 4.4: A render showing how a large motor will rotate the cradle and the gears used

Figure 4.4 shows how the cradle will be driven by a large motor. A large 36-tooth gear (#**32498**) is directly attached to the rotor with two small pins to reduce the need for an extended shaft which will flex when placed under strain (i.e. at the start of a rotation). This gear drives a 12-tooth gear (#**32270**) to triple the rotational velocity at that point of the gear train. This connection is immediately repeated and used to create a 90°turn onto the shaft with the worm gear. At this point the speed of the worm gear has a ratio of 9:1 when compared to the motor's output, which is then decreased forty times by the large 40-tooth gear to create a final ratio of 1:4.444. This means the maximum rotational velocity of the cradle will be 1.843rad/s (17.6 RPM) at 7.5V and the original torque is increased from 17.3Nm to 76.8Nm, which will allow a face of the Cube to be rotated with ease. The vertical black shaft seen at the top of the image provides a mounting point for the cradle.

The cradle's is a solid platform of 7 studs × 7 studs × 2 studs (length × breadth × height) with a surrounding wall of height 1 stud. This ensures that the weight of the cradle uniformly distributed and the centre of gravity is low. These are important factors in the design of the cradle because it is mounted on a singular vertical shaft, which is quite flexible under relatively low amounts of strain[2].

[2]The cause of any excess strain is discussed in the Implementation and Testing section

## 4.2 Codebase

### 4.2.1 Object Classes

**Cube**

**Position**

### 4.2.2 Solve Methods

**Group Based**

**Tree Based**

### 4.2.3 Database

### 4.2.4 Move Sequence Translation

The algorithms considered and discussed in the Literature Review (as well as others used in determining God's Number) are designed to work with a standard moveset $\{L\ R\ U\ D\ F\ B\}$ (as well as half turns and counter-clockwise turns). The design of the robot means that it is restricted to permutations of the moveset $\{X\ Y\ D\}$ - and will therefore require any incompatible sequences to be translated to the robot's moveset.

When the entire Cube is rotated, the **colour** of the sides will change but the sides' **orientations** will not (analogous to when one turns on the spot: left and right have changed, but east and west have not). This means that the frame of reference for the moves will have to remain invariant throughout a sequence by compensating for any full Cube rotations.

This is best explained with an example: the first step in performing the sequence $U\ F$ would be to do the sequence $X2\ D$ in order to complete the $U$ move; but now the original $F$ face has moved (due to the $X2$) - it must be located, the Cube must be rotated to get the original $F$ face into the cradle, and then the sequence can be completed. The simplest method of overcoming this complication is to convert the original move sequence to an equivalent sequence of colour-based moves: $U\ F \equiv W\ G$.

Once the original sequence has been converted to a colour-based sequence, a deep copy of the original Cube is made and the coloured sequence is iteratively translated and applied to this Cube. On each element of the sequence, the colour is converted back to a side move (allowing preservation of the correct frame of reference). Once this new side-based move is found, it is converted to a short move sequence that will move the face into the cradle and rotate the side correctly to match the original move. This short sequence comprises moves compatible with the robot.

Each of the moves in the short sequence is applied to the copied Cube, and appended to a list of moves which is returned once the original sequence is processed. The application of the moves to the Cube allows following moves to be translated correctly (as the Cube will have changed orientation in real life so needs to in this simulation). Continuing the previous example of $U\ F$, the translation is as follows:

$$
\begin{aligned}
U\ F &\rightarrow \underline{W\ G} \\
\underline{W} &\rightarrow U \\
U &\rightarrow X2\ D \rightarrow \boldsymbol{X\ X\ D} && \Longrightarrow\ U\ F \rightarrow \boldsymbol{X\ X\ D}\ \underline{G} \\
\underline{G} &\rightarrow B \\
B &\rightarrow \boldsymbol{X\ D} && \Longrightarrow\ U\ F \rightarrow \boldsymbol{X\ X\ D\ X\ D}
\end{aligned}
$$

This final move sequence will then be transmitted to the robot and used to control the motors to manipulate the Cube.

# 5 Implementation and Testing

> To those who say that 'if you need #testing at the end, you're doing it wrong', would you prefer a Boeing, or are you going Air Icarus?
>
> Michael Bolton [35]

## 5.1 Building the Robot

### 5.1.1 The Colour Sensor

### 5.1.2 The Cradle

### 5.1.3 X-Move Arm

## 5.2 Coding a Solver

### 5.2.1 Group Theory Method

#### Group Generation

### 5.2.2 Tree Method

#### Tree Generation

# 6 Results and Discussion

# 7 Conclusions

# List of Figures

# List of Tables

# Bibliography


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

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

[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*. John Wiley & Sons, 2017.

[7] T. Rokicki, H. Kociemba, M. Davidson, and J. Dethridge, "cube20," 2010.

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

[9] Etymonline, "Robot," 2017.

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

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

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

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

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

[15] J. Johnson, "Children, Robotics, and Education," *Artificial Life and Robotics*, vol. 7, no. 1, pp. 16–21, 2003.

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

[17] 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.

[18] 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.

[19] 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.

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

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

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

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

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

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

[26] 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.

[27] T. E. Oliphant, “Python for Scientific Computing,” *Marine Chemistry*, pp. 10–20, 2006.

[28] P. Anderson, “unknown,” *New Scientist*, 1969.

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

[30] Ev3dev.org, “ev3dev Home.”

[31] Ev3dev and R. Hempel, “ev3dev-lang-python.”

[32] Lego, “EV3 Programming Software,” 2017.

[33] C. A. R. Hoare, “The Emperor’s Old Clothes,” *Communications of the ACM*, vol. 24, no. 2, 1981.

[34] BrickLink, “Stud.io,” 2016.

[35] M. Bolton, “Michael Bolton Twitter Feed,” 2016.