Registration number 100057593

2016

LEGO Robot - Rubik's Cube Solver

Supervised by Dr Ben Milner

University of East Anglia
Faculty of Science
School of Computing Sciences

Abstract

General popularity of developing automated robots to execute computationally complex functionalities is becoming increasingly popular. The purpose of this project is to develop a LEGO robot which is capable of solving a 3x3x3 Rubik's cube. Said robot is designed and constructed using a `LEGO Robot Development Kit` (specifically the *EV3 edition*) and manipulated via `Matlab`. This document reflects the process in regards to any research, design and implementation performed during the project; exploring a variety of approaches where applicable.

Acknowledgements

I would like to thank *Dr Ben Milner*, acknowledging him for his support, feedback and any clarity provided throughout the projects development.

Contents

1 Introduction & Aims

Robotics is a relatively modern field - with the term *Robotics* being coined in the 1940's by Isaac Asimov - which is consistency developing to find new applications and solutions to various problems across the globe. Robotics has naturally proven very successful when producing solutions to mathematical problems and automating processes with various applications.

The purpose of this project is to develop software in order to solve a Rubik cube using a LEGO robot. A standard 3x3x3 Rubik cube will be used for the project, along side the LEGO Mindstorm EV3 robot. Once the project has reached completion, the Rubik cube should be successfully solved by the Mindstorm EV3 from a randomised starting state. The project has many factors in which its success can be measured; initially and most importantly, successfully solving the Rubik cube. Once solved, the quality of the solution is dependent on how efficiently the Mindstorm EV3 solves the Rubik cube, both efficiency in terms of time taken (Minutes/Seconds) and least possible motor-movement related failures; something to note from these terms of efficiency is that both have direct correlation with the number of mechanical moves required by the sequence of algorithms being executed. Furthermore, this brings into question which method out of the many available is most appropriate in finding the balance between the efficiency of solution and the complexity of implementation. As stated the general aim of the project is to use a LEGO robot to solve the Rubik's cube from any randomised starting state, basing the quality of the solution on time required and consistency of completion. The project can also segregated into relevant sections, each with arguably equal importance and with their own set of aims and problem domains.

Personal motivation for the project comes from a strong interest in mathematical puzzles since a young age, being presented with a problem and being able to apply set of rules or logic to provide a solution. The Rubik's cube in particular is a puzzle which has fascinated and perplexed many. A particular quote of the creator which underpins the ideology of the mass interest into the Rubik's Cube is as follows: *'If you are curious, you?ll find the puzzles around you. If you are determined, you will solve them'*.

2 Methodology

This section outlines the approach and strategy that will be implemented in order to develop a solution for the project. It will primarily clarify the sectioning of the project, each sections purpose and desired result. Please note that this methodology is included within the predevelopment

2.1 Project Structure

The project will be separated into stages to increase productivity and rate of development, also providing an easy way to organise and manage the project. The stages are as follows: Design and Construction of the LEGO Robot, Scanning and Storage of Rubik cube variables, Cube-Movement Translation, Development of Algorithm Sequences. Additionally, a potential error checking and correction stage may be implemented to ensure the Rubik cube is solved successfully. Segregating the project in this manner will also prove useful when finding relevant research/resources regarding each stage, which will be vital to ensure the completion of the project.

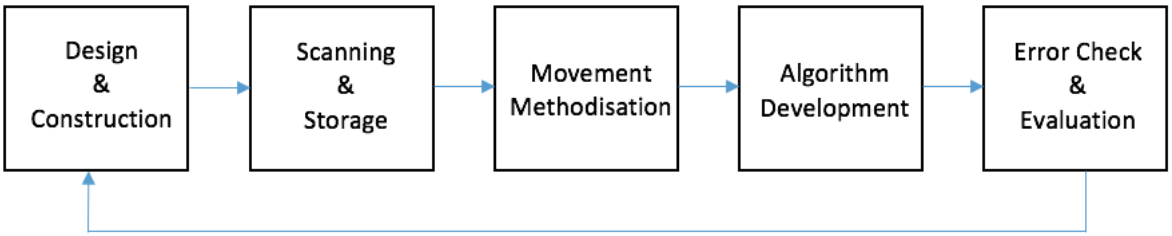

Figure 1: Project Stage Diagram

2.2 Design and Construction

The first section refers to the schematic design of the LEGO robot, allowing for all needed functionality and ensuring that the said schematics can be implemented using either the NXT or EV3 LEGO robot development kits (RDKs) released by LEGO.

2.2.1 Aims & Approach

Further in this document it will state the pre-development research reflecting upon project requirements that the LEGO robot must meet/fulfil. The aim of the LEGO design and construction is to included the requirements, ensuring that the schematics allow for the needed functionality can be executed by the LEGO robot. Additionally, the pre-development research will also present a choice between existing LEGO RDKs.

2.3 Scanning and Storage

The second section of the project refers to any methods and techniques which will aid to increase efficiency of the robots solution; for example the impacts of using image capture devices versus an RGB/colour sensor. This section also includes more generalised information regarding the data structures implemented, reflecting upon any research found about the Rubik's Cube and how some methodologies of the cube may determine how data/variables are handled.

2.3.1 Aims & Approach

The aim of any techniques or methodologies implemented which fall under the Scanning and Storage phase is to increase efficiency of both detecting and manipulating relevant cube variables such as colour and position. By increasing the efficiency and decreasing the required time of these two sections, it will have a direct and potentially major impact on the overall efficiency of the solution.

2.4 Cube-Movement Translation

The third section is the shortest and the least influenced by any pre-development research. It will consist of listing all necessary Rubik's cube operations and movements, converting these to methods which can be executed by the LEGO robot as the algorithm sequence requires. As a result of the methodised movements, implementation complexity with be reduced and code reuse will be increased.

2.5 Algorithm Sequences

The final section refers to the sequence of steps which the LEGO robot will execute in order to solve a Rubik's cube which has already been scanned and its variables stored. There are many algorithm sequences openly available and ready to be translated into code for the programmable block to execute, however some solutions have greater complexity and a higher level of required understanding of the Rubik's cube concepts.

2.5.1 Aims & Approach

One of the main aims regarding this section is to find an implementation which reflects high efficiency and consistency, reducing the number of operations required and resulting in a reduced chance of robot fault occurrence. The implementation complexity and required concept understanding also needs to be considered when deciding upon which movement-sequence to translate into computational algorithms.

3 Rubik Cube Analysis

3.1 History

Only a few puzzles and toys have remained relevant throughout generations, the Rubik's Cube is one of these few. The cube was first was developed in by the Hungarian professor Erno Rubik in 1974, in an attempt to create a visual representation of three dimensional geometryRubik's-BrandLtd (2016) - the cube could be moved in multiple directions whilst remaining intact and therefore could be used as a working model for teaching subjects such as spatial relationships. Once Erno Rubik realised the potential of his creation as a mathematical puzzle, it took him over a month to develop a solution to solve cube and as a result was quoted stating 'It was a code I myself had invented, yet I could not read it'. Both the above quote and the original purpose of the cube underline how abstract the mathematical puzzle is conceptually. At this point Erno Rubik began to push his creation into manufacturing but due to political conditions under the communist regime in Hungry at the time this process did not begin fully until 1980 when it was licensed by *The Ideal Toy Corp*. Over the following three decades the cube

saw a massive success in terms of popularity, reaching over 350 million units sold by 2010 and being regarded by many as one of the best selling products of all time. The time tested puzzle has over forty-three quintillion individual starting positions and has come into contact with approximately one in seven people alive. Despite this popularity Erno Rubik's own perception of his creation as a 'form of art and curiosity, symbolising contrast between concepts such as simplicity and complexity, stability and dynamism, order and chaos '.

3.2 Core Concepts & Approaches

Despite how initially confusing the Rubik's cube appears, certain concepts can be found which aid when developing a solution for the mathematical puzzle. Understanding the core concepts is essential to ensure the completion of the project. The figures below reflect how the cube would appear if the 3D shape was unraveled/flattened and also a way of labelling each face in order to support and simplify the development of the algorithm sequences.

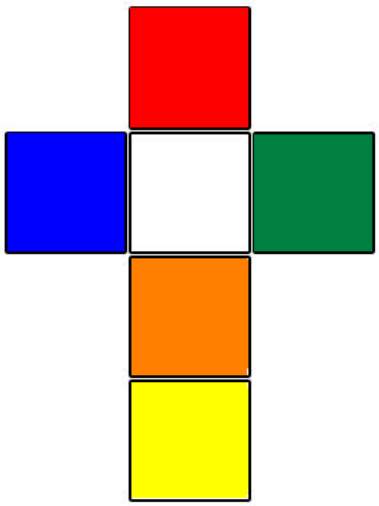

Figure 2: Unfolded Cube: Colours

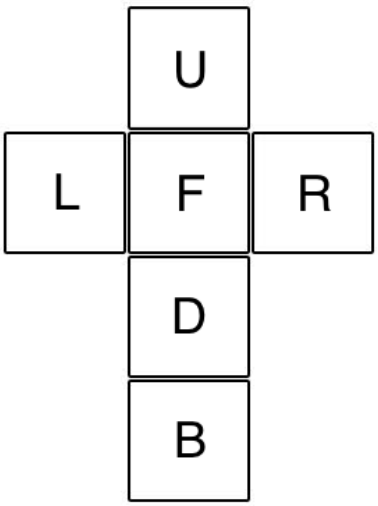

Figure 3: Unfolded Cube: Faces

Due to the design of the Rubik's cube there are only a certain amount of moves which can be implementedFerenc (2016). Each of these faces displayed in the above figure can have three different movements applied to them, a 90°clockwise motion, a 90°anti-clockwise motion and a 180°motion in either direction. The figures below reflect how the cube would appear if the 3D shape was unraveled/flattened and also a way of labelling each face in order to support and simplify the development of the algorithm sequences.

Cube Movement List			
Face	Clockwise Notation	Anti-Clockwise Notation	Double Turn
Front	F	F'	F2
Back	B	B'	B2
Up	U	U'	U2
Down	D	D'	D2
Right	R	R'	R2
Left	L	L'	L2

To clarify there are three different block types which comprise the cube, the centre blocks on each face have a single colour and remain in a fixed position and therefore are not manipulated by any of the cube movements mentioned above. On a standard 3x3x3 Rubik's Cube there are eight edge pieces which consist of two colours in addition to the four corner pieces which are composed of three colours.

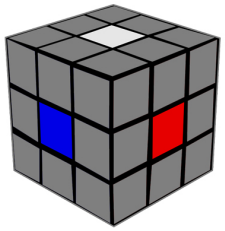

Figure 4: Center Blocks

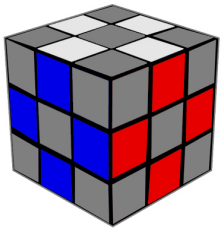

Figure 5: Side Blocks

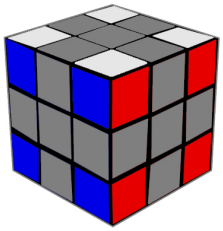

Figure 6: Corner Blocks

Additionally, a block can be described both in terms of position and orientation. The position of a block describes the location of the block on the cube, whereas the orientation of a block describes wether or not its colours are aligned correctly with the relevant faces.

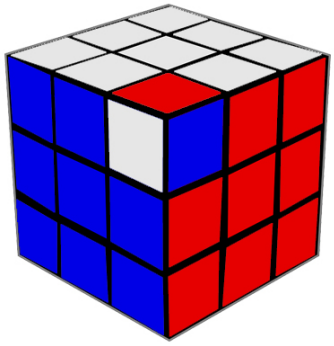

Figure 7: Corner Block Orientation

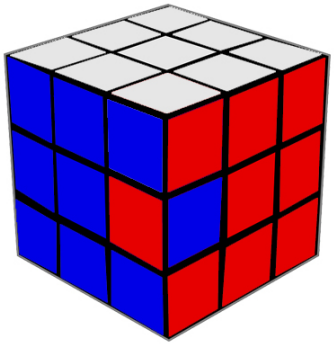

Figure 8: Side Block Orientation

Furthermore, as the robot will need to implement additional steps in terms of simply manipulating the orientation of the whole cube, these movements also require to be labeled and therefore can be discussed during further development of the project. The images below illustrate the difference between manipulating the whole cube's orientation in a horizon or vertical fashion.

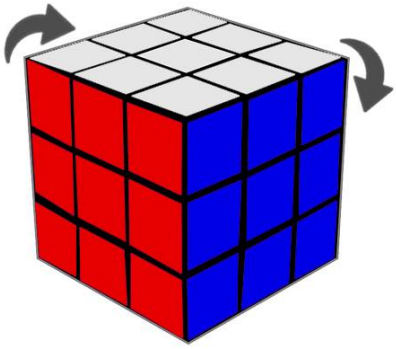

Figure 9: Pre-Vertical Orientation

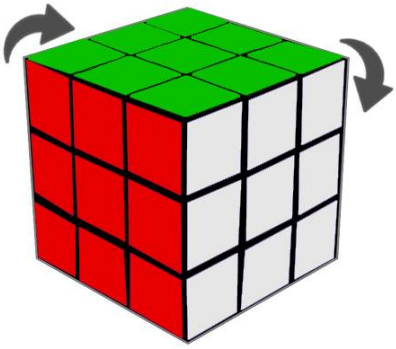

Figure 10: Post-Vertical Orientation

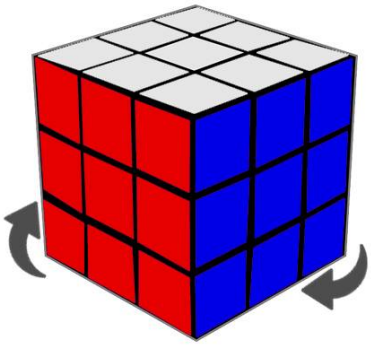

Figure 11: Pre-Horizontal Orientation

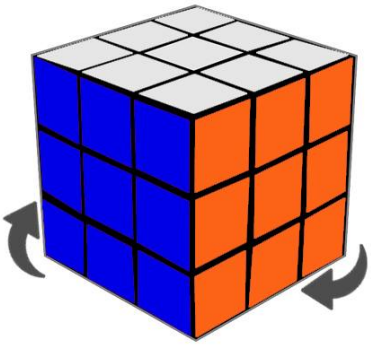

Figure 12: Post-Horizontal Orientation

4 LEGO Robot Analysis

4.1 NXT

This robot development kit (RDK) was originally released in July 2006 and replaced LEGO's original RDK titled *Robotics Invention System*. The NXT kit capabilities included three input ports used to control motors and four input ports for sensor interaction. Sensors which were available to the NXT RDK include touch, light and ultrasonic sensors. Three years after the NXT's original release, a newer version (NXT MINDSTORM 2.0) was released in August 2009 and extended the RDK's functionality by the addition of a colour sensor and other improved capabilities.

4.2 EV3

The most recent and modern RDK released by LEGO is EV3 model. As the name implies the EV3 is the third generation robotics kit, which was released in August 2013. This RDK includes an additional input port compared to the LEGO NXT and hour output ports. The EV3 development kit also includes are range of sensors from infrared sensors to gyro-sensors; additionally, the EV3 digital colour sensor is an improvement on the previous one found within the NXT RDK, including features such determining the difference between various colours, detecting the absence of colour and handles ambient light in a more appropriate manner to increase the consistency of correct functionality.

5 Pre-Development Research

Each of the previously mentioned stages have an individual problem domain which needs to be addressed during the development of the project. The first step in addressing these problem domains is to review and appraise relevant resources. However, there is very little existing academic literature in which to study. Therefore, research will be directed towards similar systems analysis, comparing existing schematics and techniques based upon efficiency; repeating the process for each stage within the project.

5.1 Similar-Systems Analysis

This section is dedicated to decomposing an existing system in order to analyse its individual components, the purpose of each component and the interaction between multiple components where relevant. Researching previously developed systems with the same application allows for an insight into potential mistakes and advances which were not previously considered. Additionally, it allows for vital functionality to be determined before any project development has begun.

5.1.1 3x3x3 Cube Solvers

Firstly, the Tilted Twister 2.0 LEGO robotAndersson (2010) schematics consists of two sensors and three motors. A colour sensor for registering the colour variables, along side an ultrasonic sensor to detect once a cube has been placed within the cradle. The motors are used for rotating the colour sensor in order for the algorithm sequence to be executed, another rotating the cradle holding the cube and a final motor to operate the tilter arm; an aspect to note about this build include the tilted angle which aids with cube rotation with the tilted arm. Secondly, the BenedetteliBenedettelli (2008) LEGO Rubik's cube solver robot schematics offer a different route regarding the construction of the robot and how the robot executes certain functionalities. The main difference is the addition of a webcam which allows for four motors to be used resulting in the capability of moving the cradle in both a rotary and horizontal motion. In addition, there are two arms, one of which is used to push/rotate the cube and the other secures the top two layers of the cube, allowing for operations to be executed. Thirdly, the MindCub3rIAssemble (2011)IAssemble (2013) LEGO robot schematics consists of similar motors to the Tilter Twister, relying on a single motorised arm and an RGB/light sensor rather than a webcam. Build quality separates these schematics from the others, providing a very solid base, motorised arm and cradle.

5.1.2 Advanced Systems

Further research into 4x4x4 and more complex LEGO robot systems may be beyond the scope of this project, yet inspiration can be found from such research. MultiCuber 3 is

a prime example of a 4x4x4 cube solver with an average solution consisting of around fifty moves. It uses a smart phone to capture an image of the cube and calculate the algorithm sequence required to solve the Rubik cube. A more recent and impressive example is the CubeStormer 3ARM (2014), developed by Mike Dobson and David Gilday. It once again uses a smart phone to capture and analyse an image of the cube, however it consists of eight motorised arms to perform operations. This means that the cube does not have to be rotated before the next operation is carried out, reducing the time required by removing additional rotary steps. The CubeStormer 3 currently holds the Guinness World Record for the solution of a 3x3x3 Rubik's cube with a time of 3.252 seconds required. As a result of this further research, a common theme emerges as the schematics reflect a huge focus not just on functionality but also robustness. Academic literature regarding robot robustness is rare and unspecific to this project, therefore examples and inspiration from these more advances models is important when attempting to achieve this quality.

5.1.3 Systems Analysis Conclusion

In conclusion of the similar systems analysis regarding the Design and Construction stage, the various robots previously described have individual and common aspects which should be taken into consideration within the development of this project. The first and simplest aspect is the tilted design of the Tilted Twister, providing aid to the rotation of the cube whilst the cradle remains motionless. The second aspect to consider is the webcam as seen in the Benedetteli cube solver, it allows for an additional motor to be implemented; the additional motor could be used for a secondary motorised arm for movements or to move the cube cradle in horizontal motion. The current MindCub3r version is the very sturdy in structure construction and is therefore the most robust of the three. The wheel components provide a solid base, while the motorised cradle and arm allow for functionality to be executed with lesser risk of robotic malfunction. Further research beyond the scope of the project show possible additions to the robot design, however, these additions increase implementation complexity during the later stages of development and will be discussed in the relevant following sections.

5.2 Scanning and Storage

5.2.1 Scanning Aspects

The LEGO EV3 kit includes a RGB/Light Sensor which can be used to detect several different colours (including those commonly found on a Rubik?s cube) or to register light intensity values. In order to determine between using the colour scanner or the light sensor, the consistency of the colour sensor in different lighting environments would be the deciding factor. When using the light intensity values, customised thresholds can be implemented to determine the colour. However, the results may still be inconsistent depending on the lighting environment during the scanning period of the solution. Both techniques using the RGB/Light sensor are reasonably simplistic to implement but as previously mentioned can be faulty under different lighting; another factor to note is the time required using this method, each face requires a minimum of nine scans which results in an increased time taken and therefore increases the chance of a fault occurrence. Lastly, due to its lack of popularity and simplicity to low level of complexity to implement, there is very little academic literature available regarding this topic. An alternative and more advanced method would be with the addition of an image capture device such as a webcam or smartphone. The image capture device is used to register the values of a whole cube face at once, resulting in only six scans being required rather than the previous 54 needed with the RGB/Light Sensor. In order to analyse the captured image, computer vision techniques are required. When looking further into computer vision techniques, there are multiple approaches to gaining the ??necessary information. Fortunately academic research is available for this section of the project, resources such as Computer Vision: A Modern Approach give an insight into techniques such as edge-line detection, colour tracking and object identification all have a valid application when analysing the cube variables. Having the academic research is extremely useful as it provides both a general understanding along side specifics regard- ing computer vision techniques. Once again, the suitability of each technique should be evaluated based upon the efficiency improvements in consideration of the implementation difficulty.

5.2.2 Storage Aspects

When exploring this section of the project, it becomes clearer that representing the variable information in the form of matrixes is the most logical option. Therefore developing the project in a language which handles matrix operations effectively would be most appropriate. Matlab is a development environment and language which is based around the use of matrix operations (the basic data element is of type matrix) and respectively is a common choice for similar projects. Furthermore, as mentioned within the project proposal, methods such as vectorisation allow for increased efficiency within code - this is simpler in Matlab as the addition of two arrays requires only a single command rather than the use of a loop. Additional research into vectorisation has proven useful; it is the process of rewriting for/while loops so that multiple elements within an array can be processed simultaneously rather than in a linear fashion. he use of vectorisation can result in more than an increase code efficiency, other benefits include reducing the volume of code and therefore increasing readability/maintainability and decreasing potential programming errors.

5.2.3 Additional Devices

Image capture devices have been previously been mentioned briefly in the *Similar-Systems Analysis* by referring towards webcams and its potential use shown within existing systems. Alternatively a smart phone device could also be used to provide the image capture capabilities; using a device such as a smart phone allows for other opportunities by providing access to its own central processing unit (CPU). The smartphones CPU operates at a higher speed than the one found within the LEGO EV3 intelligent brick and therefore, sharing or delegating the processes to the smartphones CPU would decrease the time required for a solution to be computed. However, integrating this approach into the project comes with many implementation complexities and therefore may be beyond the scope of the project.

5.2.4 Scanning/Storage Conclusion

To conclude upon the research found regarding this section there are a few factors to note. For both scanning and storage there are various techniques which will aid to increase the efficiency/reduce time taken to complete. Despite that these techniques will inevitably increase implementation complexity, the appropriate academic research will prove useful not only for the general understanding of the concepts behind techniques but also during the implementation of the techniques throughout development. Overall, the research and concepts discovered should have a drastic effect on the efficiency of the overall project in terms of time requirements.

5.3 Algorithm Development

LEGO's RDKs have been popular amongst robotic hobbyists and aspiring younger students since the release of NXT. This popularity has resulted in numerous tutorials and schematics being released and a rise in compatibility of the RDK with both development languages and environments.

5.3.1 LEGO Development Kit

The LEGO development kit is a piece of softwareWilson (2013) released by LEGO to simplify development by providing a list of prebuilt functionality such as motor movements and timers. This software is primarily aimed towards amateur robotic hobbyists and younger developers, therefore it can be seen as limited in many ways; restricting the functionality available during development. Although this environment can be used to test basic functionality of the robot during early stages of development, it lacks the capabilities required for the project.

5.3.2 NXC

Not eXactly C (commonly referred to as NXC) is one of the most popular languages amongst LEGO RDK developers. NXC is developed using the Bricx Command Centre (BricxCC) environment which compiles and translates source code into byte code which can be executed by the LEGO robotUnknown (2011). The sole purpose of the language

is regarding development of RDKs but despite this it has many restrictions and can not be considered as a general programming language. These restrictions would increase the difficulty of implementation regarding topics such as computer vision techniques.

5.3.3 Java: LeJOS

LeJos is considered replacement firmware (installed via SD card) and is available to the LEGO EV3LeJOS (2015). It allows for intelligent brick to be programmed using Java. In comparison to the previously discussed programming languages, Java allows and provides a large collection of functionality. In addition, this allows for developers to use any integrated development environment (IDE) which supports the Java programming language; some of these IDEs provide additional functionality such as error indications, refactoring and testing, potentially increasing development production. Furthermore there are an abundant amount of open source projects and libraries providing yet another layer of functionality available.

5.3.4 Matlab

The final language which will be considered is Matlab, a high-level programming language which excels when developing programs which require mathematical computation and the manipulation of array/matrix data structures. Due to these advantages MatLab can prove highly useful when implementing image processing, vectorisation or object sorting techniques. Compiled code can be transferred via USB or wifi where the program is executed on the stand-alone processor of the RDK. Basic functionalities of the RDK are already accessible and supported by MatLab packagesMathWorks (nown). There are a wide range of additional support packages available which provide functionality to interact with the hardwares input and output capacities via the communication interfaces. By combining MatLab with Simulink (A piece of sister software) further libraries and packaged allow for parameter tuning during the execution of the program which could be used for testing. A simple example of this would be adjusting light values which vary depending on external ambience levels.

6 Design and Construction

This section in an overview of the design and construction process regarding the schematic layout of the LEGO robot. Aspects of the pre-development research (Similar-Systems Analysis) provides a basis for an initial build as well as a source of inspiration. Approaches used by the similar systems analysed allows for both positive and negative aspects to be found. This section will address these aspects by carrying out the process of adopting the positive features of each similar-system in a iterative development approach; introducing a set of positive issues on each design iteration. Furthermore, changes on each iteration will be based upon increasing efficiency and reliability of the LEGO robot.

6.1 Initial Build - Light (RGB) Sensor

Similar-Systems analysis has shown that the MindCub3r Gilday (2015) proved to be the most robust and most well-rounded build and therefore the initial build will be based upon the MindCub3r system. It is also important to note that an image capture device is not included within the design schematics at this stage, this decision was made as tests will be carried out allowing for evaluation and comparison of the initial schematic layout and the revised schematic layout - showing the improvements in terms of efficiency and reliability.

6.1.1 Components & Software

The initial design/build does not focus upon achieving the highest efficiency but instead has to provide the basic functionality required to solve the Rubik's Cube. In order to provide this functionality there are various components required, mainly consisting of sensors and motors.

Firstly, the chassis is fairly basic baring close resemblance to the MindCub3r due to its high stability. It rests on the four rubber wheels faced downwards, with a connecting centre beam. Its basic structure only has the addition of a holder for the intelligent brick to be placed, this allows for the addition of future components.

Secondly, the cradle was developed as it acts as a centre piece in which the robots movement capabilities are orientated around. The sizing of the cradle became increasingly relevant as it had to be large enough for the cube to be rotated in a vertical fashion, whilst remaining as small as possible in order to ensure the reliability in the cubes position post-rotation - a design aspect which is believed to prove useful during the scanning phase (assumption applicable to both builds using the light sensor and the webcam device. This component (attached to one of three motors) will also be solely responsible for rotating the cube in a horizontal fashion.

Thirdly, the arm component is relatively conventional. It is used when manipulating the cube's vertical orientation and during core steps of the solving algorithm (any step in which the state of the Rubiks cube is altered. The arm simply latches onto the top layer of the Rubiks cube, remaining still whilst performing core steps; during vertical rotation of the cube, the arm pulls the top of the Rubiks cube until it makes contact with cradle edge forcing the cube to change orientation.

A fourth component was then introduced, a scanning device in the form of a light(RGB) sensor. The sensor is attached to a motor which helps to reposition the sensor when required. Each individual block face is scanned meaning nine scans are required per face of the cube, and a total of fifty-four for the entire cube. This scanning approach is simplistic in implementation and reliable at the expense of time efficiency.

Lastly, an infrared sensor was introduced allowing the robot to detect a cube being placed into the cradle, signalling for it to begin the solution procedure. As no algorithm development has taken place, the LEGO development kit alongside the MindCub3r solution software were used for testing purposes - these tests will allow for displaying the change in efficiency during later development.

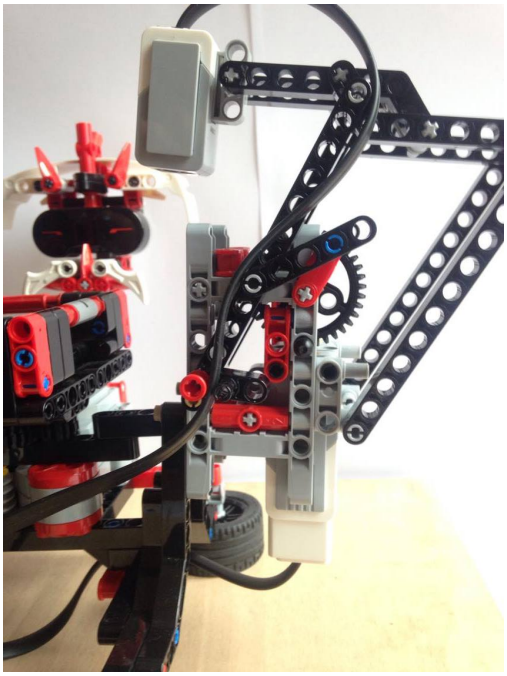

Figure 13: Light Sensor Side-View

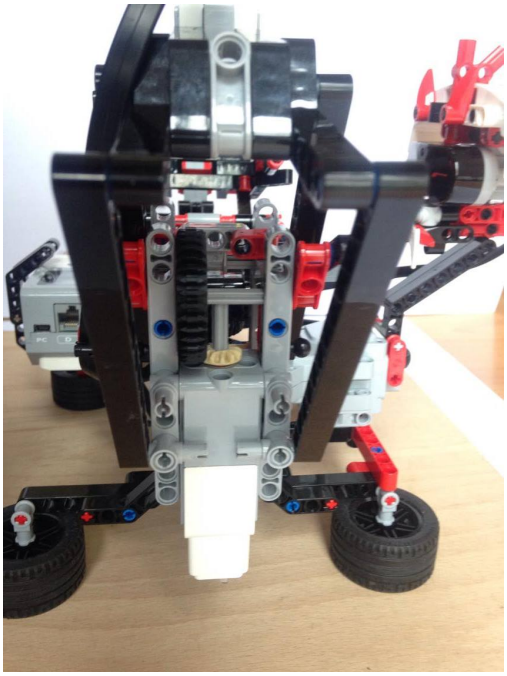

Figure 14: Light Sensor Back-View

6.1.2 Testing

In order to deduce the efficiency of the initial build, it is important to test for both time taken in addition to the robot malfunction occurrence rate. Regarding the time efficiency of the build, the overall completion time can be segregated into the time required by the individual sections of the solution process. Due to the nature of the scanning device included within the initial build, the number of scans is a constant variable; it is fair to assume that the time required for the scanning device to complete is also a consistent value. The overall time of completion, along with the number of core and cube-rotation moves required is expected to vary depending upon the cube's starting state and solution provided by the software.

To ensure clarity, the following defines the test parameters:

- **Test Run** - A number used to identify the tests carried out individually.
- **Scanning Time** - The time required for the Rubik's Cube to be scanned.

- **Algorithm Time** - The time required for the solution algorithm to execute.
- **Completion Time** - The time required for the entire solution process to complete.
- **Core Movements** - Any movements which alter the state of the Rubik's Cube.
- **Rotation Movements** - Any movements in which the cubes orientation is changed but remains in the same state.

Initial Build Testing					
Test Run	Scanning Time (Seconds)	Algorithm Time (Seconds)	Completion Time (Seconds)	Core Movements	Rotation Movements
1	30	58	95	22	32
2	30	77	108	22	37
3	30	70	104	24	35
4	30	71	112	24	46
5	30	21	50	8	13
6	30	75	110	23	42
7	30	59	88	20	32
8	30	70	116	28	45
Average	30	63	98	21	40

6.1.3 Initial Build Conclusion

When observing the test results, there are a various deductions which can be made. Firstly, no robotic failure occurred which suggests that components such as the cradle size and variables such as the cube tension have been correctly implemented. As previously assumed, the scanning time was consistent taking thirty seconds to scan the six faces (54 individual scans), requiring a total of 30 rotational movements of the cube - this section section can not be altered without altering the scanning device implemented. Furthermore, the results show a clear difference in the starting state of the Rubik's Cube, these unique states can be categorised to cases depending on the resulting completion time required. The tests have shown that average time of execution is

98 seconds. The best case result which took place in Test Run 5, taking a total of 50 seconds to complete, as well as the worst case which took place in Test Run 8, taking a total of 98 seconds to complete. When comparing the difference between these cases it becomes clear the effect in which the initial cube state has upon the overall efficiency of the executed solution. The recorded value which is most effected is the total amount of rotational movements in order to manipulate the cubes orientation; the robot must perform between one and two rotational movements before the subsequent core algorithm movement can be executed. If N equals the number of core-movements required, then the resulting number of rotational movements would be between N and Nx2. This allows for the rotational movements total to classified into boundaries:

*Let **N** equal the total number of core-movements, and **R** equal the total number of rotational movements.*

- Lower Boundary: **R** =< **N** x 1.25
- Medium Boundary: **R** > **N** x 1.25 **& R** < **N** x 1.75
- Higher Boundary: **R** => **N** x 1.75

A value which was not recorded during testing was the computational time required for the software to produce the series of solution steps. As the MindCub3r software was used to test the initial build, comparing the two different build's computational times would prove mostly pointless, it is however important to note that computational time will effect the efficiency, and is dependant on the cube's starting state meaning it is an inconsistent value.
Aspects of the initial build which are taken forward into the revised build include the chassis/general layout, cube cradle and rotary arm. The chassis provided a sturdy base which is easy to build around whilst the cube cradle and rotary arm allowed for the necessary functionality to be performed in an effective manner. Aspects which are to be developed upon during the revised build design revolve primarily around the scanning device; by introducing a different scanning device the supporting structure may also change depending on the new devices requirements.

6.2 Revised Build - Webcam Device

The purpose of the revised build is to re-use the positive aspects of the initial build and address the negative features appropriately. Testing of the initial build was not extensive enough to declare that no robotic failures will occur during the solution process, therefore developing methods to further negate the probability of a robotic failure occurring during the design of this build proves beneficial. Further within this section, each components purpose will be discussed, with relevant initial design schematics and finalised component diagrams included.

6.2.1 Cube Cradle

Purpose: To provide a holding place for the cube, whilst allowing for manipulation of the cube in terms of both rotational and core movements.

The attached motor allows for a change in horizontal orientation (as seen in Figure:16), in addition to the side barriers which aid the rotary arm during changing the vertical orientation (as seen in Figure:15). During core movements, the cradle rotates a single layer whilst the rotary arm is used to hold the remaining layers in place.

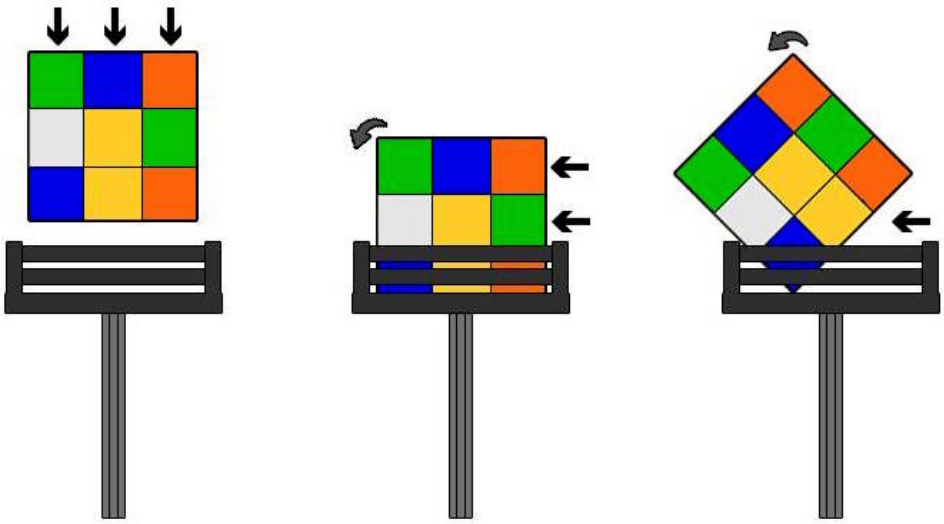

Figure 15: Cube-Cradle Side View/Vertical Cube Rotation

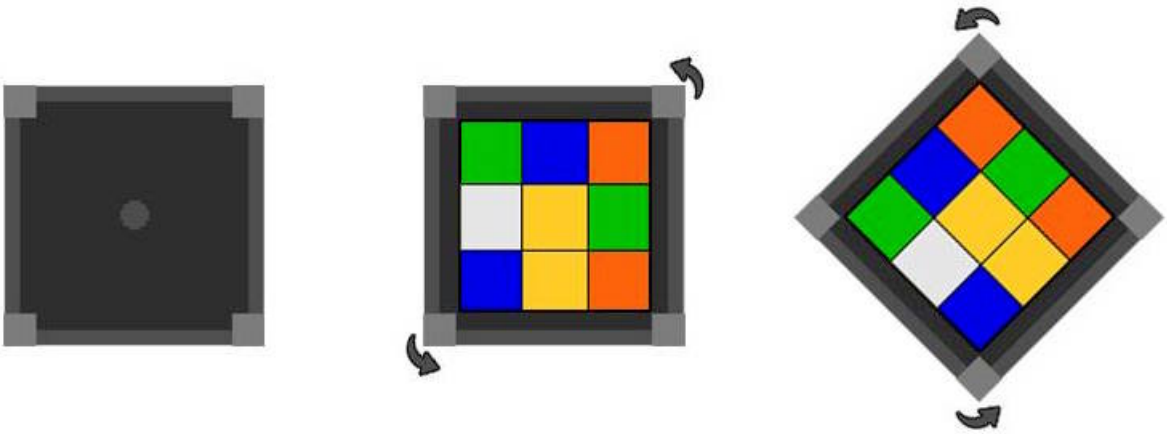

Figure 16: Cube-Cradle Top View/Horizontal Cube Rotation

The resulting build of the cube cradle component is displayed by the images below (Figure:17 & Figure:18). Overall the build follows the structure of the design images, including any previously discussed build requirements. No major changes were made between the initial build and revised build in relevance to the cube cradle.

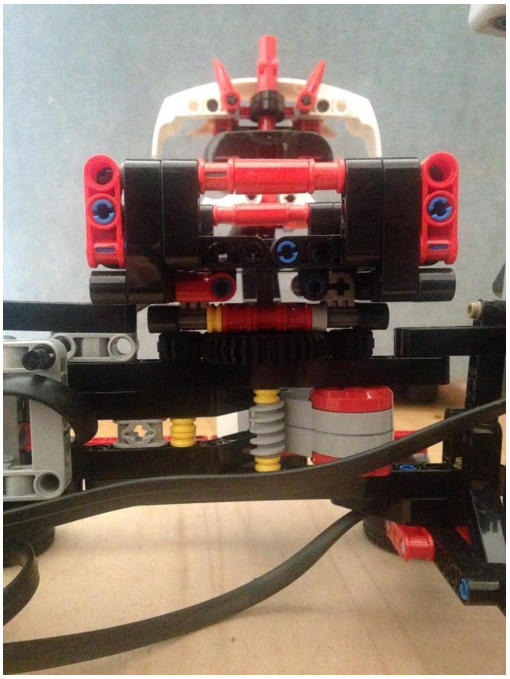

Figure 17: Light Sensor Side-View

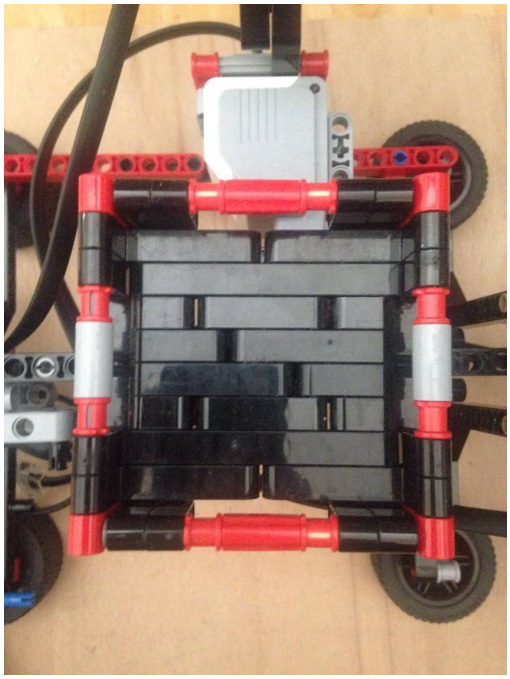

Figure 18: Light Sensor Back-View

6.2.2 Rotary Arm

Purpose: To provide functionality during the manipulation of the Rubik's Cube state and vertical orientation.

The attached motor acts as a centre point in which the arm rotates around, allowing it to latch onto the cube (in the same fashion seen in Figure:19). At this point, core movements can be performed by rotating the cradle whilst the arm holds the top two layers; alternatively, the motor can be used again to pull the cube against the cradle barriers, resulting in vertical orientation taking place.

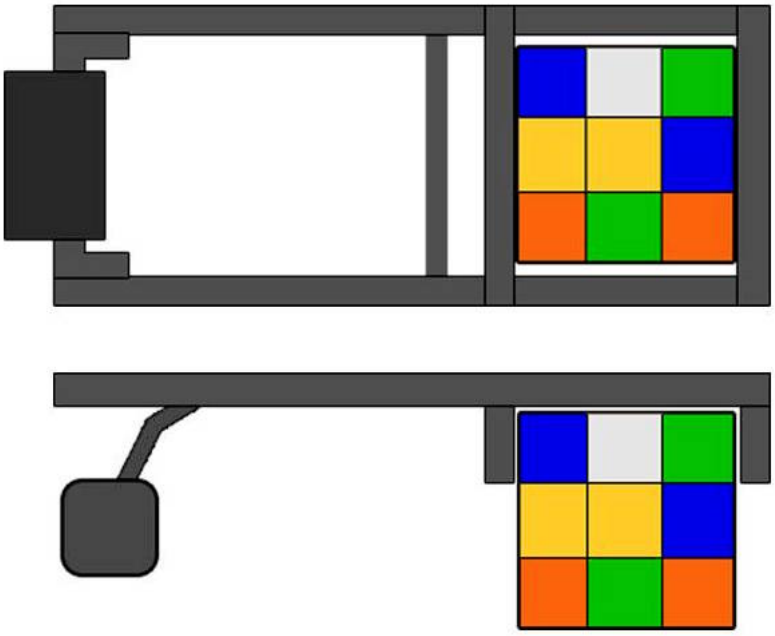

Figure 19: Cube-Cradle Top View

The resulting build of the rotary arm component is displayed by the images below (Figure:20 & Figure:21). The arm follows the structure layout of design images seen in figure19, including any previously discussed build requirements. Similarly to the previous component, no major changes were made between the initial build and revised build in regards to the rotary arm.

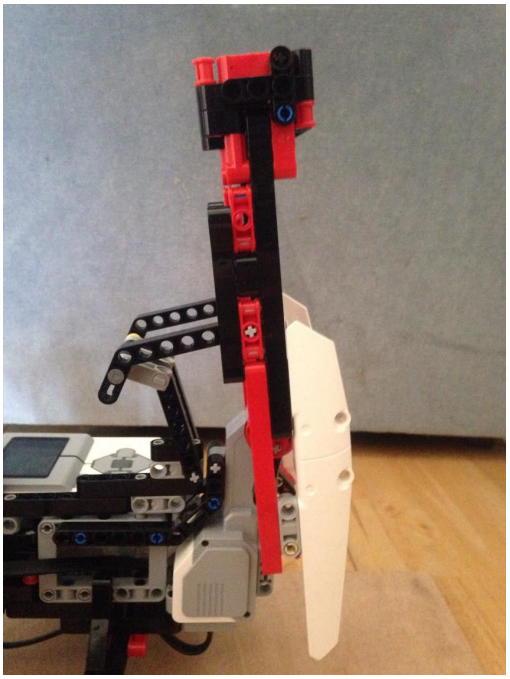

Figure 20: Rotary Arm: Side-View

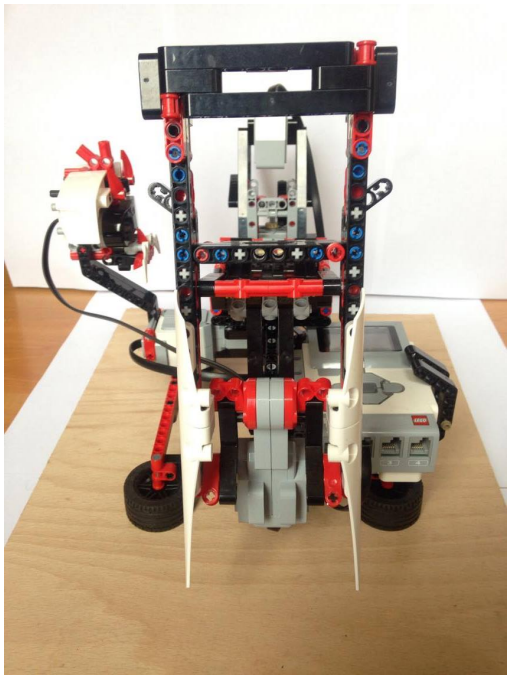

Figure 21: Rotary Arm: Top-View

6.2.3 Webcam Cradle

Purpose: To provide a secure structure to hold the webcam in an appropriate position to successfully and reliably scan the cube.

The structure is rather simple, there is not an attached motor as no alterations to the webcams position are required to perform scanning as the cube itself is rotated in order to capture an image of all six faces. As the build is constrained to the amount of LEGO pieces provided by the LEGO EV3 Kit, the structure of the webcam cradle is limited. Due to this constraint, the webcam cradle can not be placed over the cube cradle at the height required; instead the webcam cradle is to replace and take the position of the infrared sensor. The scanning device (a webcam in revised build) is placed upon the cradle facing downwards. Despite that this design requires the cube to be placed onto the same surface as the robot during the scanning phase, the height gained by the existing structure of the robot allows for the webcam to held at a position capable of capturing an appropriate image for analysing.

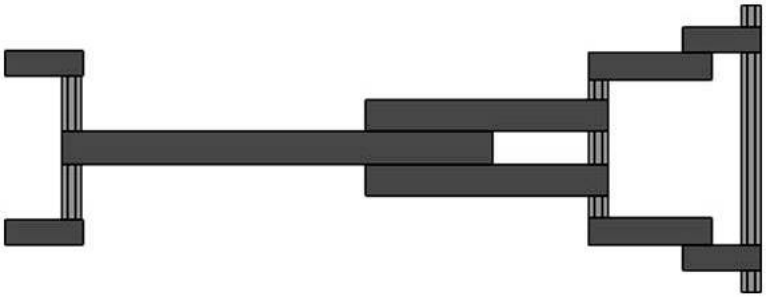

Figure 22: Webcam-Cradle Design Image

The resulting build of the webcam cradle component is displayed by figures26 & 27. These figures show that the component developed has been strongly influenced by the design image shown in figure22. In order to secure reliability and prevent unsuccessful attempts during the scanning phase, the cube should remain in relatively the same proximity after rotation has occurred; this could prove to be an issue as (with the current revised build discussed) the cube has to rotated manually, allowing the possibility of human error.

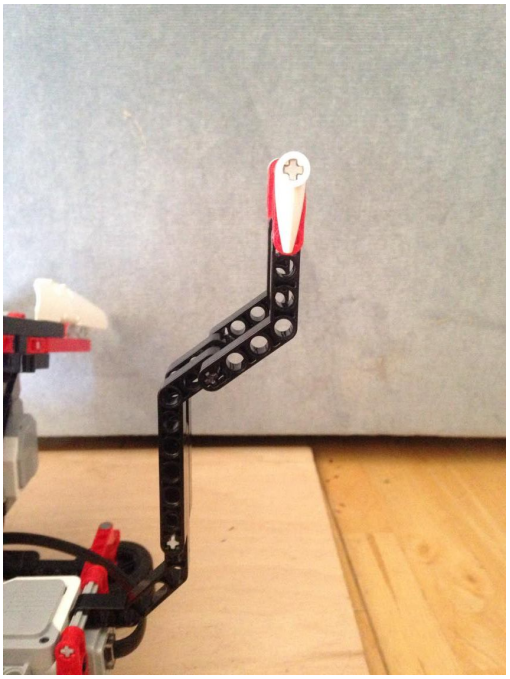

Figure 23: Rotary Arm: Side-View

Figure 24: Rotary Arm: Top-View

6.2.4 Base-Board & Adjusted/Tilted Angle

Purpose: To provide further stability and reduce potential robotic failures via small alterations/additions to the robots structure.

The first alteration involved securing the robot onto a solid structure such as a base-board to help increase stability. It is common for robotic movement to be considered relatively erratic, hence the base-board has been included to reduce unwanted movement of motorised and non-motorised components. Secondly, elevating the base-board and in effect tilting the entire robot may prove useful to further negate the potential of robotic failures. Similar systems analysis (specifically the Tilted Twister 2.0 LEGO robot) has shown that the downward force from gravity can help during rotations of the cube; helping to secure the cube is in place for further operations. The degree of elevation needs to be considered as if too steep, it could have adverse effects to the components intentions and potentially interfere with the robotic movements required during cube rotation.

Lastly, the board can provide a proximity for the cube to be placed during the scanning phase. Altering the base to incorporate a slot for the cube would also help to ensure the phase is executed successfully; allowing the cube to be re-placed appropriately after rotation.

Images displaying the base-board with the additional tilt can be found within the subsequent section.

6.2.5 Component Compilation

All key components have been discussed alongside their purpose and design requirements. Within the following section are images containing a compilation of the mentioned components, in addition to the resulting compilation of the revised build.

A difference to be noted between the design image and resulting images is that the light/RGB sensor is present within the resulting build. This is for demonstration purposes so that both builds can be displayed during any later presentations.

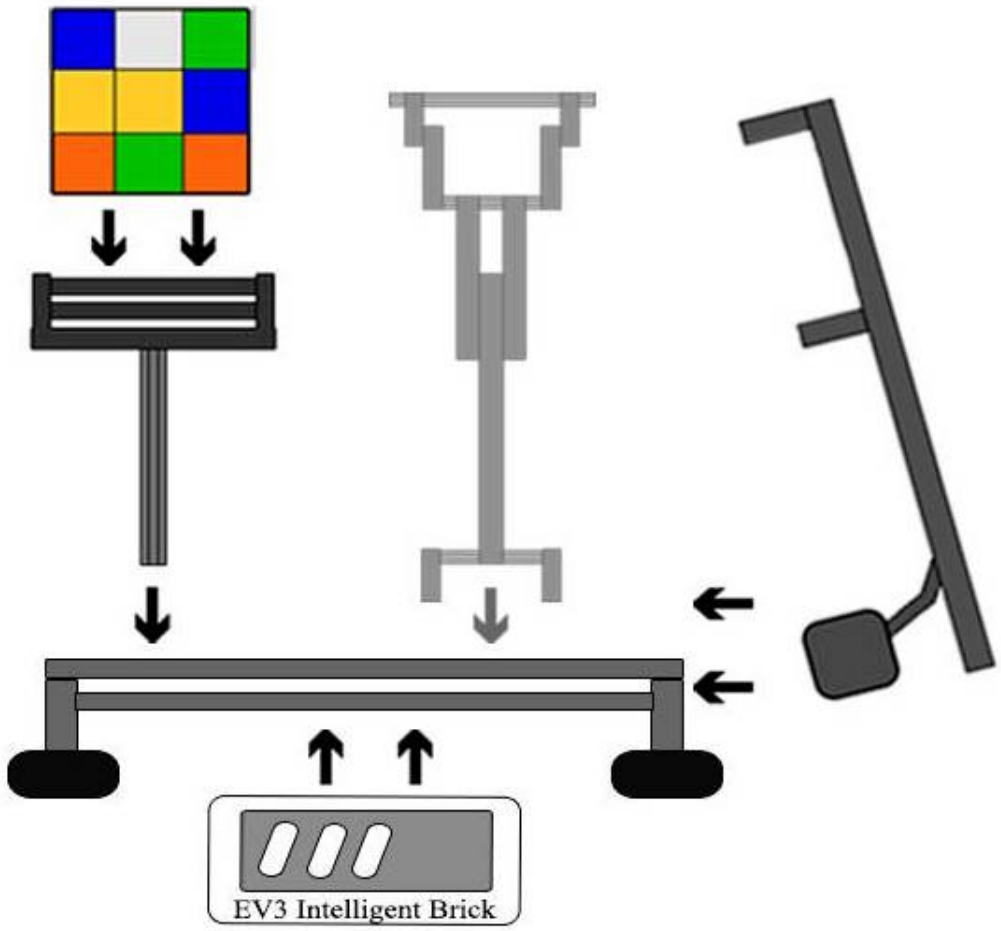

Figure 25: Component compilation of the revised build: Design Image

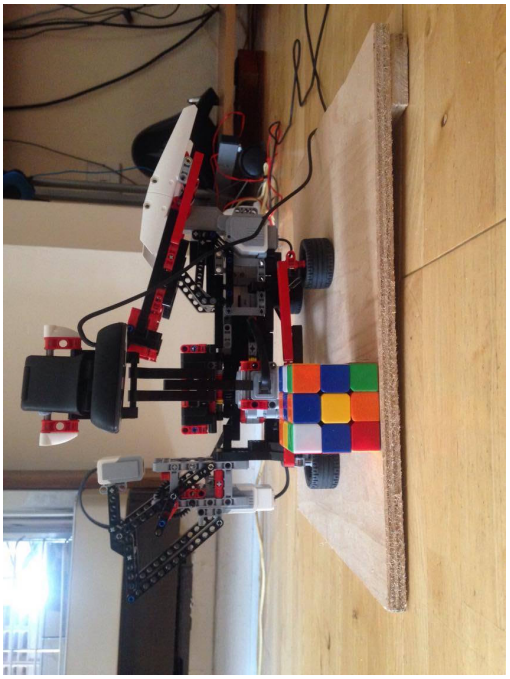

Figure 26: Revised Build: Side-View

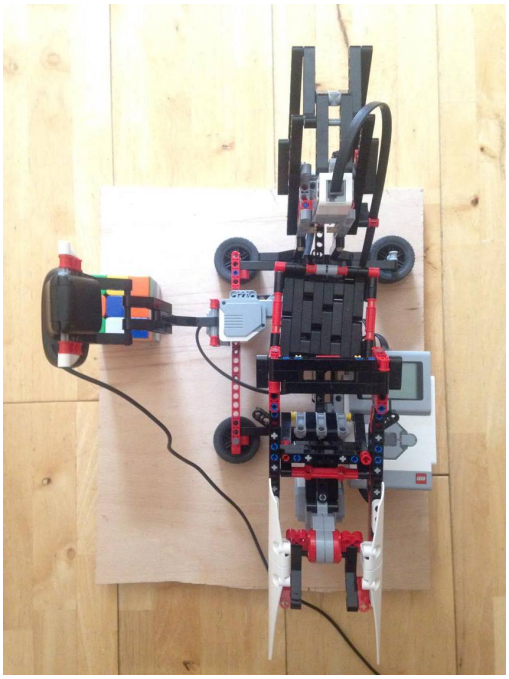

Figure 27: Revised Build: Top-View

6.2.6 Revised Build Conclusion

When assessing the produced robot from the design build, its capabilities allow for the required functionality to be executed with either scanning device (image capture device or RGB sensor); as well as all functionality required to perform both core-movements and rotation movements. In particular, the similar systems analyse has proven useful. The initial build used the similar system with the most stability (the MindCub3r by David Gilday), allowing for the evaluation of individual components and the entire robot. This evaluation allowed for both positive and negative aspects of the build to be outlined, later to be addressed by the following design iteration, the revised build.
Despite documented development between iterations, the revised build still retains original aspects of the MindCub3r such as the rotary arm or cube cradle, which provide the cube manipulation required. However, additions such as the elevated baseboard are influenced by other similar systems analysed (in this case the tilted twister). A defining difference between the produced revised build and any similar systems analysed during research is the scanning device used - an image capture (webcam). The image capture

device is an addition which does increase complexity of implementation in comparison to its predecessor found within the initial build. Despite this, the webcam can help to reduce the scanning time required by the system which previously required a consistent thirty seconds to execute (as seen in the results of the initial build testing). To allow for both scanning devices to remain as components, the IR sensor has been removed to provide enough space. The infrared sensor was incorporated within software used by the initial build, detecting once a cube has been placed into the cradle and for solution procedure to begin execution. This software will need to be edited, initialising the process on a button press instead of object detection, in order to allow for later demonstration of the initial build.

Prior to the following sections of scanning, cube movement translation and algorithm development, evaluating the build is rather unspecific as no tests are able to be performed. However, some limitations can still discussed; as the webcam cradle does not allow for the cube to be placed within the cube-cradle during scanning (due to a lack of LEGO pieces remaining from the LEGO EV3 Kit), the cube has to be rotated manually. Automating the entire scanning process (including cube rotation) would allow for a faster time and leave no opportunity for human error. The process of manually turning the cube, may prove to hinder the time required to execute the scanning phase, this considered during later testing.

Evaluating possible future developments is important as if the permitted time allows for further iterations of the revised build, these developments will become a key focus point of said iteration. When disregarding demonstration purposes, there is no need for a RGB sensor to be incorporated (hence why it was not shown during design images) and therefore an additional motor slot is available. During further development, an additional motor could be used for another rotary arm, one which holds only the top layer of the cube, allowing for it to be rotated independently of the two below. By providing this capability to the robot, the amount of cube rotation movements required can be reduced rather drastically depending on the starting state.

In conclusion, the revised build develops upon the initial design build appropriately. Positive aspects remain in the build, such as the arm, cradle and chassis of the robot; whilst compensating for any negative aspects via small adjustments/additions (such as the base-board) or by implementing a different approach, such as the scanning method applied. Overall, the revised build has been a successful iteration of the previous.

7 Scanning Phase - Image Capture Device

As discussed during the design phase of the revised build, the RGB sensor has been replaced by an image capture device. This section of the report will discuss aspects regarding the research and development of software to operate the image capture device, alongside any required image processing/analysing. The purpose of the software developed during this section is to capture images of the cube, retrieve information regarding cube variables (colours of each block) and finally classify the blocks by colours. The final output should be compatible with solution software which is used; the solution software will be formally discussed during the *Algorithm Sequence Development* section.

7.1 Research

Before software development is discussed, research regarding underlying concepts and relative techniques is to take place. The following section discusses the major underlying concepts required and techniques which can be applied to solve a set problem/issue. By undergoing the research procedure to discover and discuss possible routes which can be taken, software development may prove to be more efficient (in terms of time and processing required) and reliable.

7.1.1 RGB - Colour Model

A colour model is a method of defining/modelling colour which is based upon the human perception of colour. There are numerous colour models which exist and each model has its own adequacy depending on the application in question. RGB is com-

monly used for digital representation of colour images amongst software application. Consisting of three separate colour values (Red, Green and Blue)**?**, the RGB colour model is defined as additive; combining different levels of the three stated channels to produce a singular colour.

Each channel has a value (also known as an intensity) ranging from a minimum of 0, to a maximum of 255. Once a value for each colour channel has been declared, a resulting colour can be determined by combining the intensity of each colour in relevance to one another. Due to the large range of possible values for each channel, there are a wide amount of possible colours which can be represented.

Within the context of this project, the RGB colour model can be used when analysing images captured of the Rubik's cube. Depending on the RGB values found on each block, a colour can be determined; reproducing this action for each block will allow for the cube's state to be retrieved. Further explanation of how RGB values are manipulated for this project is discussed during the *Software Development* section.

7.1.2 Classification

Classifying images is a vast topic with many varying approaches. Image classification is a form of classification based upon statical data of an image, or set of images. The statical data of the image is analysed in order to retrieve information allowing it to be labelled with a category. Despite the abundance of approaches, it is common for image classification techniques to use training/reference images; these images allow the system to develop some understanding of the desired categories and the defining characteristics of each.

In relation to this project, the colour associated with each of the six face can be used as the categories. As discussed in the previous section (*RGB-Colour Model*) it was discussed how colour can be determined via the three colour channels. Similarly, colours can be labeled or categorised based upon the RGB colour intensities found within the reference images. Therefore, cube variables obtained via scanning can be used in com-

parison to set an appropriate label based upon (category specific) thresholds.

7.1.3 Lighting Correction/Consideration

When analysing a set of images and performing tests based upon the present colour intensities, the lighting environment can manipulate the resulting colours which are determined. Therefore, the following section describes various techniques for lighting correction/consideration and the applicability of each within this project.

Firstly, homomorphic filteringButtkus (1975) (a form of image processing) is method used to remove visual noise from an image. The process is based upon the image consisting of two components which are reflectance and luminosity values. Despite that both components are caused in effect by the lighting environment, the reflectance values are considered as visual noise, as it can distort how the image is perceived. Homomorphic filtering is the process of removing the reflective component and increase the contrast found within the luminosity values. Within the context of this project, this technique could be suitable in regards to the resulting image. However, the processing required is relatively excessive for the application of it within this project. Additionally, the process is commonly performed on images with reduced lighting, allowing for the image to be brought forward and image details visible.

Image normalisation is a process which results in changing the range of pixel values found within an image. Uses for the normalisation of images commonly aim towards removing visual noise within an image (such as glare), but also used for allowing any further image processing to be independent of the lighting environment present. Although not as extensive as homomorphic processing, image normalisation also requires a certain level of computation to occur; within the context of the project, this may require a set of time/processing which is too high when aiming to deliver efficiency.

Lastly, an approach known as reference imaging is a method of managing the effect of the lighting environment via the use of predefined images. By providing a set of reference images which vary in lighting environments, the closest reference can be found

and used to aid analysis of the image. In the instance of this project, reference imaging can be used to set more appropriate threshold values, rather than consistent/constant values. In relation to the previously discussed topic of image classification, this method can prove beneficial by allowing for each lighting environment to effect the statistical data which defines the categories. This approach does not completely remove the effect of lighting, but rather attenuates its effect and helps to adjust the process to suit the lighting environment of an image.

7.2 Software Development

Prior to the development of the scanning software, the requirements of the system need to be specified. Avoiding specifics regarding set algorithms, the system is required to capture multiple images of the cube (a image per cube face). The images are then analysed to retrieve values representative of each blocks colour, which cumulatively can be used to define the initial state of the Rubik's cube. The values regarding the cubes state are then to be passed to external software in order to produce a solution sequence (in the form of cube notation discussed previously during *Rubik Cube Analysis*).

Various possible programming languages/environments were discussed and evaluated during the pre-development research (specifically the *Algorithm Development* section). Based upon said evaluation and the requirements of the system, Matlab has been chosen as the development language due to its applicability to the application and also the level of support it provides for development using the LEGO RDK (Robot Development Kit).

The first stage is to capture an image of each cube face, within Matlab this procedure if simple. The scanning device (webcam) is automatically detected by the programming environment, once initialised and assigned to a variable the device can be used by various functions. As discussed during the constraints of the revised build, it was unable to facilitate the scanning procedure being performed whilst the cube is within the cube-cradle; resulting in manual cube manipulation being required during the scanning phase. To compensate for the manual rotation of the cube, the system must allow for the cube rotation process before taking each image. This was simply implemented us-

ing the pause command within Matlab, causing the system to wait for user input before capturing an image. Once all images are captured, they are stored within an array for later manipulation.

Each image currently represents an individual face of the cube, the images are processed in a singular fashion. Processing for each image begins by calling a function which obtains the colour values of a specified section of the image (this function is very similar to the block methods later described). The specified section is required to be consistent within all 6 images (a section of the baseboard in this instance) as it is used to determine thresholds which define the colour categorisation. Once obtained, the RGB values of the selected section are averaged parsed to the *refImages* function.

The *refImages* function takes an RGB value as the input (values retrieved from the function discussed by the previous paragraph), this value is then compared to a predetermined reference chart. The reference chart holds numerous RGB values of the same specified section and have been retrieved from a set of training images. Once compared to the reference chart, the index of the RGB values found with the least distance/difference are stored; the index is then used as a condition to various conditional statements which set the colour thresholds. For example, if the least distance values found within the reference chart at index 15 then the thresholds are set based upon reference image 15.
This approach using a reference set not only allows for colour classification labels to be assigned (discussed further within this section), but also for the lighting environment to be considered. Each reference image was captured under different lighting conditions. By matching the closest reference image and setting relevant thresholds to suit, accuracy of the scanning phase is increased; under a single set of constant thresholds, wrongly labeled colours occur much more frequently.

At this point, all images have been obtained and depict the whole cube state. Each image is parsed to 9 separate block methods, each of which are designed to extract the

RGB values for that image within the relevant block by specifying the relevant pixels coordinates of each block. Within this approach all of the blocks of each face are numbered in the fashion depicted by figure 28.

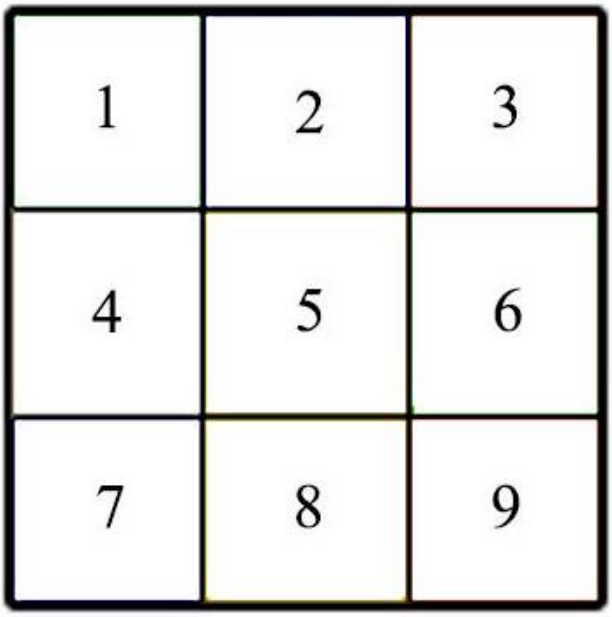

Figure 28: Component compilation of the revised build: Design Image

The first block method (named scanBlock1) retrieves the RGB value representing the colour found upon the first block, the second block method (named scanBlock2) does the same but in regards to the second block, and so on for all 9 block methods. Each of the block methods take an image of an entire face as an input and then outputs the RGB intensities representing the desired blocks colour. In order to retrieve the RGB values, the method first crops the image to remove all unwanted areas and leaves only the desired block remaining. The block is then analysed to retrieve its three colour channels, representing the colour value of each pixel. Values of each channel are stored in individual matrixes (which are of size 50x50 as a total of 2500 pixels are left after cropping), which are then averaged (becoming a 51x1 size matrix) and finally averaged again to produce a singular value for each colour channel. Once each colour channel average has been obtained, it is stored into a 3x1 matrix and returned as the function

output. Figures 29 & 30 show the initial image captured by the webcam and the regions of each block retrieved after block methods are applied.

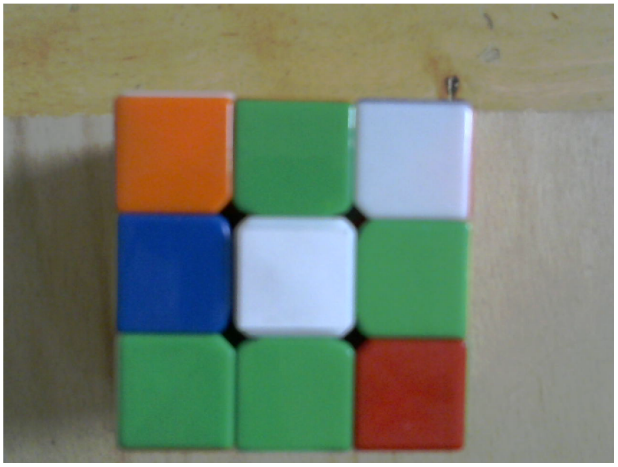

Figure 29: Image captured by webcam

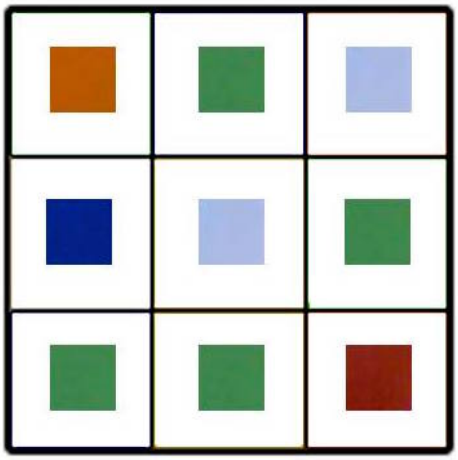

Figure 30: Post-cropping Block Areas

Once a blocks average RGB values have been determined, they are parsed to the colour classification method (*colourFind*) alongside the appropriate colour thresholds found previously. The function compares the RGB values with those found within the colour thresholds, storing the index of the least distant values. Regardless of which reference image is used to determine the thresholds, the indexes constant in the sense that a set index will always represent a set colour (as seen in figure 31). Once all blocks of a face have been classified, the values are stored into 9x6 matrix before the process repeats with the following image.

	1	2	3	4	5	6
Red Chanel	0-255	0-255	0-255	0-255	0-255	0-255
Green Channel	0-255	0-255	0-255	0-255	0-255	0-255
Blue Channel	0-255	0-255	0-255	0-255	0-255	0-255
	Red	Blue	Orange	Green	White	Yellow

Figure 31: Colour Classification Thresholds

The final requirement of the scanning software is to pass the classified colours to the external solving software (this software will be credited and discussed further during the *Algorithm Sequence* section). The software allows for cube state to be input in the form of a 3x3x6 matrix; to achieve this format, each face representation (a single row of the 9x6 matrix) is retrieved and manipulated into a 3x3 matrix. The resulting 9 3x3 matrixes are then concatenated to produce the desired format. Depending on the order in which faces were scanned, the face-representative value may need to be ordered, which can occur before or after the desired format is achieved.

7.3 Scanning Conclusion

Assessment of the scanning software implemented is based upon a comparison to applicable approaches for each scanning component, accuracy of approach and how successfully the components interact.

Image classification can be a very powerful tool depending on the application in question. Due to efficiency being a priority throughout the project, many image classification techniques require a level of processing which is too intensive. The approach implemented gives a consistent result in terms of labelling each colour based upon the set colour thresholds. The use of reference images allows for a middle ground to be found between the resulting accuracy and processing intensity. Although, in comparison to alternative image classification approaches the implementation does not give a accurate

result; this can be related to the lighting compensation applied.

Light compensation is also applied via the usage of reference imagery. The approach implemented attempts to match the current light conditions (depicted by the images captured) to the light conditions found within reference image data set. In the context of this project, the data set collected can be considered rather large. Despite the data set size, confusion between certain colours still occurred; this is caused by a lack of variance between reference images. Although the number of images found within the data set is relevant to the overall accuracy, a smaller set with a broader range of light intensities returned may prove more successful in this regard.

There are also multiple approaches to other aspects of the scanning procedure. An instance of this would be a scanning order; currently the colour is being determined without a specific scanning order, and instead on thresholds set by reference images. By introducing a specific scanning order, the central colour value of each face can be assumed and analysed to set the thresholds its relevant category.

In conclusion, the scanning software developed is reliable but not accurate. Images are captured, processed and analysed in order to assign a colour category; whilst consecutively allowing for lighting consideration. Despite this, the colours assigned may not be a true representation of the actual block colours due to the limited amount of reference images, or variance within reference imagery. Finally, the approach implemented allows for interaction between scanning components (image classification & reference imaging), therefore altering the reference data set will allow for better results for both aspects.

8 Algorithm Sequence

As mentioned at various points throughout this document, an external software (titled as Rubik's Cube Simulator and Solver by Joren HeitHeit (2011)) has been used develop a solution sequence required to solve the cube from a given state. The software requires

a format representative of the cubes scanned state (3x3x6 matrix) and outputs a solution. The solution consists of a list of sequence movements in the form of standard cube notation as described within *Core Concepts & Approaches*. The notation describes the sequence of steps to follow in order to solve the cube from a scrambled starting state, to a complete state.

The software contains numerous solving algorithms for different cube sizes (2x2x2, 3x3x3, 4x4x4). Due to the context of this project, only the 3x3x3 solving algorithms are applicable; the most efficient of the two being the *Thistlewaite 45* algorithm (also known as the T45Unknown (2008)). The upper boundary of required moves with the T45 algorithm is 45, with an average solution sequence count of 31.

The *Thislewaite 45* algorithm is relatively complex and a full understanding is beyond the scope of this project, therefore a general understanding of the ideology and approach behind the algorithm is satisfactory. T45 stems from the idea of dividing the overall solution process into smaller processes. Applying this idea to the cube orientated around group theory, specifically the group of moves which can be applied to the cube. T45 introduces the idea that at each stage there are a number of blocks which can not be moved/manipulated; reducing the group of moves which can be implemented at each stage. The remaining manipulations for each stage can then be assessed regarding the moves required, implementing the solution that requires the least moves. By doing so the upper boundary of moves required for the total solution is reduced.

Post-application of the T45 algorithm, the *Rubik's Cube Simulator and Solver* outputs a solution in the form of standardised cube notation which describes the solution process required from any starting state to a state of completion. Once obtained, the solution requires translation to robotic movements capable of being performed by the revised build.

9 Cube Movement Translation

Movement translation software that is developed is required to translate a solution from standardised Rubik's cube notation to a set of instructions of which the revised build can facilitate. Standardised notation is primarily purposed towards human interaction with the cube and therefore does not compensate for limitation of the robot. The revised build does not have the capacity to manipulate a face which is not on the bottom layer, therefore performing sequential moves requires for additional robot-specific steps between core movements. Cube orientation is also very relevant within this task; human-interaction with the cube is mostly subconscious regarding cube orientation, where as the robot must track the orientation consistently to maintain awareness of the cubes position and successfully perform a sequence of solving movements.

A limitation of the movement translation is that the post-translation moves has to be applicable to the robot and its functionalities. The revised build is capable of horizontal and vertical orientated rotation, as well as providing core movement functionality by rotating the bottom layer independently. Methodising these functionalities/movements is important during the translation phase as these movements will be used in combination to manipulate the orientation and state of the cube.

As shown within the *Cube Movement List* table there are 18 possible notations for cube movements. Therefore developing the translation algorithm consists of creating functions which can perform each notation based upon the current orientation. There are 6x4 orientation states possible as there are 4 different orientation possibilities for each face. This approach to the movement translation is relatively simple at the expense of the large quantity of methods which require development.

It can also be noted that this section could allow for different algorithm solutions to be developed as it the input is so standardised. Additionally, other algorithms may not produce a start to finish solution, but instead behave in the same way as a human interaction and solving the cube via a series of steps. To do this the initial state of the cube needs to be stored and altered in relevance to any core movements performed, reflecting

the result of the movement.

10 Project Evaluation & Conclusion

To conclude upon the project, there are various aspects which were successful and others which limited the development. In particular, the research gathered in regards to both pre-development and during development is closely relevant to the problem domain of the project. Approaches which have been discussed for each section aided during the various design and implementation phases, resulting in an overall system where the individual components complement and collaborate with one another. In particular, the scanning phase involved a number of components with overlapping functionality which aided each other to achieve the desired purpose.

Regarding limitations within the project, there are aspects at each stage which are relevant. Firstly, further pre-development research discussing the possible solution algorithms would allow for a greater understanding of the external algorithm solution software or potentially original development of a solution algorithm. Secondly, the lack of lego pieces which remained after the initial build resulting in a rather unstable webcam cradle. Stability of the webcam is considered important within this implementation as images are analysed based on pre-set pixel coordinates; unpredicted movements of the webcam could result in unsuccessful scanning results. Thirdly, the reference data set which was implemented within the scanning software does not contain enough varied lighting environments. As well as variety between images, the reference data set could be increased - image classification techniques typically use hundreds of training/reference images to set a classification categories defining characteristics. The most influential limitation within the project would be the incomplete movement translation. Although the approach was discussed, the algorithm has not been implemented. Therefore no extensive testing can be executed and comparisons between the build iterations can not be made; despite this, estimated timings can be gathered based upon the number of core movements and rotary movements required by the solution - as the builds cube movement functionality has not changed from the initial design, similarities can

be drawn.

There are many potential areas which could be considered for further development of the project. Topics mentioned among the project limitations such as additional reference data and redevelopment of components (such as webcam cradle) which provide clear routes for future development. Additionally, further iterations of the *Design & Construction* section may be explored; the processing unit which manages operations such a motor/sensor control has additional slots. These slots could be used to implement more components such as an addition rotary arm. An additional arm would potentially reduce the number of rotational movements required by allowing for the top layer to be rotated without any changes to the cubes vertical orientation (similar to the current rotary arm implementation). Lastly, a number of alternative approaches were discussed where relevant throughout the project. Further development would allow the development of these alternatives, of which tests/comparisons can be assessed to determine which approach is most viable to ensure the success of the project.

Summatively, although it is prosperous to consider possible improvements, the project was a success in regards to the overall research, design and implementation throughout. There are many individual and contrasting approaches to solving the cube for both the conventional and unconventional patterns; a creative puzzle which is guaranteed to continue being referred to as one of the greatest and most popular toys/puzzles within modern times.

References

Adobe (2000). The rgb (cmy) color model. *http://dba.med.sc.edu/price/irf/Adobe_tg/models/rgbcmy.html*.

Andersson, H. (2010). Tilted twister 2.0 - lego mindstorms rubik's cube solver. *https://www.youtube.com/watch?v=w3f-WyDqOUw*.

ARM (2014). Cubestormer 3 smashes rubik's cube speed record. *https://www.youtube.com/watch?v=X0pFZG7j5cE*.

Benedettelli, D. (2008). Danny's rubik's cube solver. *https://www.youtube.com/watch?v=rzio92VYUIQ*.

Buttkus, B. (1975). Homomorphic filtering, theory and practice.

Ferenc, D. (2016). Rubik cube notation. *http://ruwix.com/the-rubiks-cube/notation/advanced/*.

Gilday, D. (2015). Mindcub3r for lego mindstorms ev3. *http://mindcuber.com*.

Heit, J. (2011). Rubik's cube simulator and solver. *http://www.mathworks.com/matlabcentral/fileexchange/31672-rubik-s-cube-simulator-and-solver*.

IAssemble (2011). Lego® mindcuber. *https://www.youtube.com/watch?v=Uw1BgD72vaA*.

IAssemble (2013). Mindcub3r featuring lego® mindstorms® ev3. *https://www.youtube.com/watch?v=MQvG6lgrgYQ*.

LeJOS (2015). Java for lego mindstorms. *http://www.lejos.org*.

MathWorks (Unknown). Lego mindstorms ev3 support from matlab. *http://uk.mathworks.com/hardware-support/lego-mindstorms-ev3-matlab.html*.

Rubik's-BrandLtd (2016). The history of the rubiks cube. *https://uk.rubiks.com/about/the-history-of-the-rubiks-cube*.

Unknown (2008). A detailed example of the thistlethwaite algorithm. *http://cubeman.org/thistle.txt*.

Unknown (2011). Bricx command center 3.3. *http://bricxcc.sourceforge.net*.

Wilson, D. (2013). Lego mindstorms ev3 software overview. *https://www.youtube.com/watch?v=9HiNF1ry6x0*.