# Building a robot that can solve a 3D puzzle combining robotics and vision

Jordan Waddell

BSC Information Technology

2014/2015

The candidate confirms that the work submitted is their own and the appropriate credit has been given where reference has been made to the work of others.

I understand that failure to attribute material which is obtained from another source may be considered as plagiarism.

(Signature of student)_____

The candidate confirms that the following have been submitted*:*

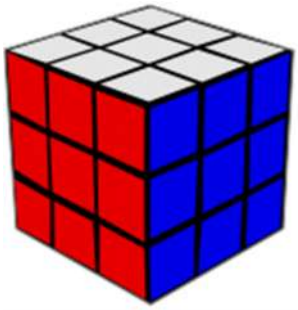
| **Items** | **Format** | **Recipient(s) and Date** |
|---|---|---|
| *Deliverable 1* | *Report* | *SSO (13/05/15)* |
| *Deliverable 2* | *Software code* | *Supervisor, assessor (13/05/15)* |
| *Deliverable 3* | *User manuals* | *supervisor (13/05/15)* |

Type of Project: Exploratory Software (ESw)

# SUMMARY

This project demonstrates how a simple robot can be built to solve 3D problems in the real world. By combining robotics and vision complex problems can be solved without the need for human interaction. These types of robots could be used for a wide range of applications such as medical examinations or rescue services.

The puzzle I choose to solve for this project is a Rubik's cube, I choose this because the Rubik's cube is a challenging puzzle and will help demonstrate how a simple robot can be more efficient then a human at solving such a complex puzzle.



# ACKNOWLEDGEMENTS

First I would like to thank my supervisor Brandon Bennett for his time and support during the course of this project. Secondly I would like to thank my tutor Andy Bulpitt for always being there to help me out even at short notice☺. Finally I would like to thank all my lecturers during my time at the University of Leeds giving me the skills and knowledge required to achieve such a project.

## 1 TABLE OF CONTENTS

# 2 INTRODUCTION

## 2.1 OVERVIEW

A 3x3x3 Rubik's cube has 6 sides, each side has 9 squares, each square has one of the following 6 colors: white, red, blue, orange, yellow or green. The Rubik's cube consists of six central pieces, each with a single colored face, twelve edge pieces with two colored faces and eight corner pieces with three colored faces. The puzzle was invented by Erno Rubik in 1974 and became one of the bestselling puzzles ever invented. (***Rubiks, 2015).***

Any face can be rotated either clockwise or anti-anticlockwise, either 90 degrees or 180 degrees. To complete the puzzle the correct faces must be turned to ensure each side of the cube contains the same color squares.

The Rubik's cube has 43 252 003 274 489 856 000 possible states. **(Mathworld.wolfram.com, 2015)**.

$$\frac{8!\quad 12!\quad 3^8\,2^{12}}{2\cdot 3\cdot 2} = 43\,252\,003\,274\,489\,856\,000$$

Every possible position of a Rubik's Cube™ can be solved in twenty moves or less. This was proved using about 35 CPU-years of idle computer time donated by Google, this is known as god's number. **(Cube20.org, 2015).** Although 20 moves is the maximal amount of moves required to solve a Rubik's cube, without Google's processing power a total optimal solution solving method is unrealistic , therefore this project will make use of predefined algorithms to find sub optimal solutions that can then be used to instruct the robot how to physically solve the Rubik's cube.

## 2.2 AIM

This project is concerned with combining vision and robotics in order to solve a Rubik's cube from any valid cube state. Vision will be used to determine the cube state by analyzing the colors of each face of a given Rubik's cube. A solution must then be generated from the cube state using a known solving algorithm. The generated solution can then be used to pass a move set to the robot which will then manipulate the cubes faces accordingly in order to solve the cube. The overall goal of the project is to combine the following features into a single easy to use application which can be used in conjunction with robotics to solve a Rubik's cube.

- ***Visual capabilities*** – A Webcam will be used to determine the colors of each square on a given Rubik's cube. This will ensure users do not have to input the cube state manually which can be quite time consuming and prone to human error.

- ***Determine solving solution*** – Determine which faces to turn to complete the Rubik's cube from a given a cube state. This will be solved programmatically using a known Rubik's cube solving algorithm. This will help minimize moves allowing the cube to be solved in the fastest time possible.

- ***Build robot*** – The robot will be used to physically rotate the cube and turn its faces accordingly. The robot will be built from scratch using a combination of different components.

- ***Graphical user interface*** – This will combine all the above functionality into a simple to use Java application.

## 2.3 EXTENSIONS

More than likely any spare time acquired will be used to refine the basic requirements, however if this project is completed earlier than planned, I may choose to use the time gained to development one or more of the following additional features.

- Attach the microcontroller to a raspberry pi instead of a desktop pc.
- Add a feature that will allow manual control of the robot from within the user interface.
- Mount the camera on the robot in order to automatically capture the Rubik's cubes current state

## 2.4 RELEVANCE TO DEGREE

This project will help demonstrate my ability to adapt to unknown problems and find solutions using the skills I have obtained during my time at the University of Leeds such as:

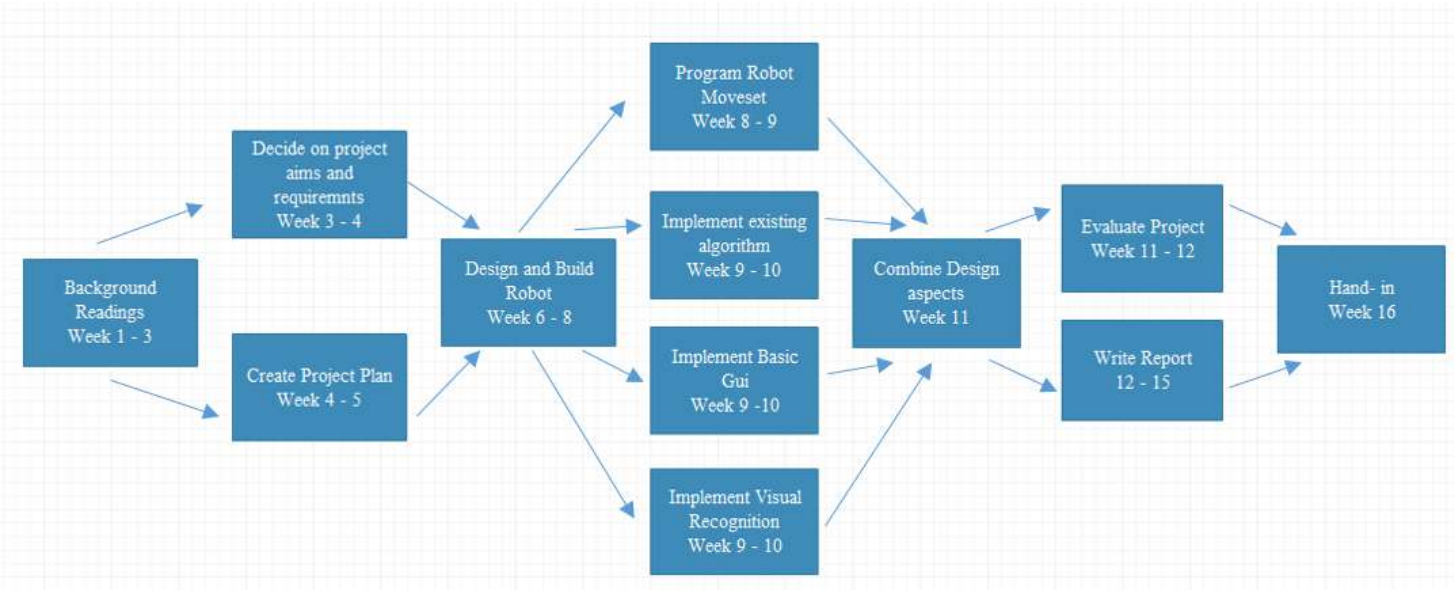
- Programming skills in both C and Java
- Machine Learning Using Weka
- Graphical user interface development
- Project management
- Software Engineering
- Electrical engineering

# 3 PROJECT PLANNING

## 3.1 GANTT CHART

| ACTIVITY | 26-Jan WEEK 1 | 02-Feb WEEK 2 | 09-Feb WEEK 3 | 16-Feb WEEK 4 | 23-Feb WEEK 5 | 02-Mar WEEK 6 | 09-Mar WEEK 7 | 16-Mar WEEK 8 | 23-Mar WEEK 9 | 30-Mar WEEK 10 | 06-Apr WEEK 11 | 13-Apr WEEK 12 | 20-Apr WEEK 13 | 27-Apr WEEK 14 | 04-May WEEK 15 | 11-May WEEK 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Background Reading | | | | | | | | | | | | | | | | |
| Aims and Requirements | | | | | | | | | | | | | | | | |
| Create project plan | | | | | | | | | | | | | | | | |
| Design robot and get required hardware | | | | | | | | | | | | | | | | |
| Build robot | | | | | | | | | | | | | | | | |
| Program robot movesets | | | | | | | | | | | | | | | | |
| Find and implement existing algorithms | | | | | | | | | | | | | | | | |
| Create basic GUI to generate a solution | | | | | | | | | | | | | | | | |
| Implement visiual recogniton feature | | | | | | | | | | | | | | | | |
| Combine all design aspects into GUI | | | | | | | | | | | | | | | | |
| Evaulate Project | | | | | | | | | | | | | | | | |
| Write Report | | | | | | | | | | | | | | | | |
| Hand-in deadline | | | | | | | | | | | | | | | | |

## 3.2 PERT DIAGRAM




## 3.3 VERSION CONTROL

For version control I used Github during the programming stage in this project. Version control can be useful at helping keep track of progress throughout a project. By keeping a record of previous commits and branches previous versions of a project can be restored if required. Having a centralized cloud based repository is useful when moving between multiple workstations allowing instant access to the most up to date version of the software being developed. I used the EGIT plugin for the eclipse java development IDE in order to connect to the Github's repository. A link to the projects repository is provided below:

https://github.com/Jstar15/RubiksPuzzleSolver

## 3.4 DEVELOPMENT METHODOLOGY

The development methodology I will use in this project will be the Extreme Programming (XP) software development methodology. This is a type of agile software development which involves developing and delivering functionality in fragments or in incremental stages. In the XP methodology changes are encouraged and are seen as natural and desirable happenings that occur during the course of the projects development cycle. This is a useful approach for this particular project where the exact requirements and potential difficulties are not always foreseeable due to the experimental nature of the project. The methodology involves splitting up the different aspects of the projects functionality into extremely simple steps giving the developer a greater degree of control over the projects development process.

The XP methodology describes four basic activities that are performed during the development process:

**Coding** - It is argued that the most important product of the system development process is code. This is because a system requires code in order to interpret the instructions necessary to achieve the project's end goal.

**Testing** - By performing unit testing on each feature implemented, each feature can be verified to be functioning correctly, this will reduce the risk of possibly overlooking any potential software errors later on in the development process. Also *acceptance tests* are required to ensure the functionality is working as intended and that it will help fulfill the software requirements.

**Listening** - Programmers must understand the technical aspects of a project well enough to in order to communicate ideas with others about what can and should not be done and also be aware of the project's limitations and constraints.

**Designing** – When designing the system simplicity is encouraged and an emphasis is placed on ensuring the programs logic is carefully thought out. Good logic can reduce system dependencies and allows for easier design modifications in the future.

Extreme programming recognizes five key values:

**Communication** – Communication is important between team members and also through the use of good software documentation.

**Simplicity** –It is encouraged that simpler functionality is implemented first and more advanced functionality added later. It is believed that investing time on future requirements which may change is a waste of time, time that could be better used later on in the project to refine and improve upon an existing software implementation.

**Feedback** – Feedback can be extremely important in identifying any possible shortcomings or design flaws. Feedback can be gained either by *unit testing* or by acquiring *formal feedback* from others whom are in a similar field of expertise. This can help inspire innovation and help see a problem space from a different perspective.

**Courage** – It is important that developers have the courage to refactor and modify their code when necessary, also developers must have the courage to be persistent in order to gain solutions for complex problems even in the face of uncertainty.

**Respect** – Having respect for yourself and others is essential. Respect for the quality of your work and respect towards other members who assist the projects development. This helps boost moral which in turn improves motivation and the overall quality of a completed project.

***Source - Wikipedia, (2015). Extreme programming.***

# 4 BACKGROUND RESEARCH

## 4.1 INTRODUCTION

The background research investigates the various hardware and software requirements needed to successfully complete this project, this also includes any background readings.

## 4.2 CUBE MOVE NOTATIONS

Each of the 6 faces of a Rubik's cube are assigned one of the following letters, (R) Right, (L) Left, (U) Up, (D) Down, (F) Front or (B) Back. These notations are preferred instead of using colors because it is more intuitive in terms of knowing which cube face to turn without visually tracking colors on the cube. These notations can then be used to communicate a sequence of moves that can be used to solve the Rubik's cube. (**Astro.berkeley.edu, 2015).**

Modifiers are used as an extension to the face notation to give additional information to the direction the face should turn and to what degree. Examples of possible move notations are given below:

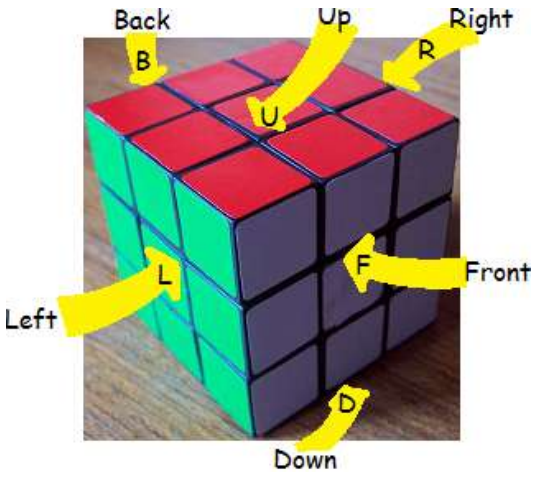
Using these notations

R = turn Right face clockwise 90° L= turn Left face clockwise 90° etc.

R' = turn right face anticlockwise 90° L'=turn left face anticlockwise 90° etc. (R' is read as `R-prime')

R2=turn right face 180 degree L2 = turn left face 180 degrees etc.

A set of these notations can describe the moves necessary to solve the Rubik's cube.




(***Commons.wikimedia.org, 2014).***

## 4.3 Solving Algorithms

There are many different Rubik's cube solving algorithms that have been invented and are optimized for solving the cube in the minimal amount of moves. I plan to integrate a piece of existing software that implements one of the following algorithms:

- Thistlethwaite's algorithm - "In 1981 Thistlethwaite showed that any configuration could be solved in at most 85 moves" **(Optimal solutions for Rubik's Cube, 2015).**
- Kociemba's algorithm- "In 1995 Michael Reid proved that by using this algorithm every position can be solved in at most 29 face turns" **(Optimal solutions for Rubik's Cube, 2015).**
- Korf's algorithm - In 1997 Richard Korf announced an algorithm which by using a method called IDA* is a depth-first search that looks for increasingly longer solutions in a series of iterations, using a lower-bound heuristic to prune branches once a lower bound on their length exceeds the current iterations bound. This method will only return an optimal solution and may take a long time to output a solution if the cube is too scrambled. **(Optimal solutions for Rubik's Cube, 2015).**

It is important that the algorithm chosen can compute a solution in an acceptable amount of time and moves, also most importantly it should be reliable. The algorithm chosen must give a solution of 30 or less moves for it to be considered an acceptable amount of moves, also the time taken to compute a solution must not exceed 2 minutes.

## 4.4 Previous cube solving robots

I have found several Rubik's cube solving robots on YouTube. The following YouTube links show 3 slightly different Rubik's cube solving robots, they all differ in their design implementations in terms of the mechanical movements used to manipulate the Rubik's cube.

https://www.youtube.com/watch?v=la2LtrQ8k_I

https://www.youtube.com/watch?v=dreTvumjNyw

https://www.youtube.com/watch?v=WG4FyoCjgdk

The previous cube solving robots may be used for inspiration when designing and building a robot for this project.

## 4.5 Development Platforms

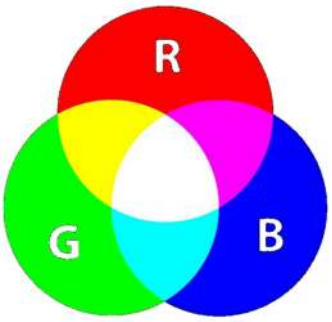
All software as part of this project has been developed and tested on the Windows 7 Operating System. All code will be written in either the C or the Java programming language. This is a personal development preference as these are the languages in which I have the most experience. The GUI for this application will be written in Java using the Jframe library due to its simplicity enabling quick development. The Eclipse IDE will be used for compiling and debugging java code. Notepad++ will be used for debugging C code. C code will be compiled using the command line in windows. Environmental Variables must be configured in windows in order to compile C code from the command line.

## 4.6 DETERMINING COLOR FROM AN IMAGE

### 4.6.1 RGB Color model

RGB is a color model which combines the colors red, green and blue, by using this model a wide variety of colors can be represented. In computing each RGB value (red, green and blue) is assigned with a numerical digit between 0-255, the numerical digit represents color intensity with 255 being the most intense. For example the color 255.0.0 will be pure red. By using the RGB color model RGB values can be extracted from each pixel in a given image which could then be used to determine the true color of an image. "The RGB color model is based on the Young–Helmholtz theory of trichromatic color vision, developed by Thomas Young and Hermann Helmholtz". **(Wikipedia *RGB color model,* 2015).**

**(Upload.wikimedia.org, 2015).**




### 4.6.2 Environmental factors in color detection

One of the biggest challenges of color detection is the capability to accurately identify the various colors in different environments. There are several environmental factors that can greatly affect the perception of color such as context, hue and brightness. As humans we automatically adjust to these environmental factors so that we can identify colors in a wide variety of environments, however even we have our limitations, as recently demonstrated by a picture of a blue and black dress which has become a global phenomenon. People continue to debate about the true color of the dress due to different people perceiving the same image of the dress in different colors. **(BBC News, 2015).** One of the greatest challenges of this project is writing a program that can use a webcam to accurately recognize the colors of each square on a Rubik's cube. Improved results could be gained by using a more expensive camera that has better hardware specifications and as a result can capture images with reduced distortion. Even better results could be gained by taking advantage of machine learning techniques such neural networks, deep learning or classification algorithms which can be used to model and predict output based on predefined training data. **(Anon, 2015).**

### 4.6.3 Machine learning

Weka is a collection of machine learning algorithms used for data mining tasks, Weka is useful at classifying data based on predefined attributes. Weka uses training data to make classification predictions on a test data, the training data consists of datasets containing a set of attributes and there classification, whereas the test data only consists of datasets that contain only a set attributes where its classification is unknown. Using the Weka library in Java, I plan to generate a training set containing attributes which consists of data extracted from the RGB values that were taken from various images, each image represents a single square on the Rubik's cube, along with its actual color as it classification. The training set will contain image data taken from different environments which in effect will improve the prediction accuracy and be less prone to error. I will then test and evaluate various machine learning algorithms with the training data in order to find the algorithm with the best prediction rate. Machine learning tends to work better when more datasets are available in the training data, more data tends to mean more accurate decision trees can be generated which in effect will yield better prediction results.

## 4.7 HARDWARE REQUIREMENTS

- The robot will require motors and a skeleton to be able to physically manipulate the cube and be able to turn each face of the Rubik's cube according to a set of given instructions. For this I decided to use the *LEGO NXT kit* to build the frame and attach the motors that came with the kit. The Lego NXT kit contains a large variety of components that can be used to easily construct mechanical mechanisms and build structures. I decided not to use the microcontroller that comes with the Lego kit because it only supports a maximal of 3 motors and can only supports a limited amount of components. My alternative to this is to use an *Arduino Uno* instead which is a more powerful microprocessor that can support up to 6 motors on digital outputs and can also support up to 6 analog inputs.
- The Arduino Uno will require an external *H-bridge* which provides the functionality to allow the motors to turn both clockwise and anti-clockwise. This will be needed to give the robot a greater variety of possible move sets.
- A single H-bridge can only support up to 2 motors, therefore multiple H-bridges will be required depending on the amount the motors used.
- A *sensor* will be required to accurately detect the motors positions when rotating a face on the Rubik's cube. This can be used to ensure the motors timing does not go out of sync. This will make it less prone to error when turning the faces of the cube due to the different level of resistance on each face of the Rubik's cube.
- A *webcam* will also be required to provide visual recognition capabilities, this will be used to capture the Rubik's cubes initial state.
- A 12v external ac to dc power supply is required to power the Arduino Uno and the various electrical components.
- Jumper cables are required in order to interconnect the various electrical components.

## 4.8 SOFTWARE REQUIREMENTS

- A Java application needs to be created to in order to read the state of a Rubik's cube using a webcam. For this I have decided to use the OpenGL library in Java. The *OpenGL* library will provide the necessary functionality to capture and enhance an image of a Rubik's cube face.
- The *Weka* library can be used to predict the color of a square in a Rubik's cube based on the RGB values extracted from an image of a Rubik's cube face. I choose to use Weka in order to try and reduce errors caused by the fluctuation of light in different environments effecting the accurate prediction of colors.
- The application also needs to be capable of generating a solution to solve the cube. For this an existing cube solving algorithm needs to be found and modified in order to work with this project.
- The application also needs to be able to generate a move set based on the solving solution, the move set can then be uploaded to the robot telling it which faces to turn in order to solve the Rubik's cube.
- An Arduino sketch file needs to be automatically generated containing the methods used to communicate with the Arduino in order to initiate a specified move set in order to solve the Rubik's cube. This also has to work in conjunction with the Java application so that a unique sketch file can be generated based on the cube solving solution.
- Pipes will be required to run both the C code for the solving algorithm program and for compiling and running an Arduino sketch file from within a Java application. The design decision of using pipes will make the application platform dependent due to it relying on functionality that is built to the specific OS it was developed for, therefore additional code would be required for it to run on multiple platforms.
- Finally a GUI needs to be created to provide a user friendly interface to provide the user with the functionality to the above features. The GUI will be written in Java from scratch using the JFrame library, I choose this as a personal development preference due to its simplicity and easy integration.

# 5 IMPLEMENTATION AND DESIGN

## 5.1 INTRODUCTION

The project can be broke down into 4 distinct steps: visual recognition, programmatically solving the cube, building the robot and finally combining these features into a single GUI application that will provide the functionality to solve the Rubik's cube.

## 5.2 VISUAL RECOGNITION

Using Vision to detect the colors of a cube can be broken down in 3 steps: capturing the image of a cube face, extracting numeral data from the image, and then finally using that data to predict the color of each square on a Rubik's cube face using Weka machine learning. The various steps where refined multiple time throughout the project in order to continually improve upon previous results.

### 5.2.1 Capturing Image

To capture the images of each face of a Rubik's cube a Logitech Quickcam Express webcam was used, its sensor resolution is 320x240 and supports up to 30fps.(note: windows drivers had to be installed). The OpenCV library in Java is then used to capture an image of a face on a Rubik's cube using the webcam. The OpenCV library contains classes that provide rich functionality to capture and enhance images that are captured on a webcam. The Java application will continuously show a preview of the webcam output image and will refresh every 20ms, the preview image will be flipped vertically so it will be easier for the user to align the Rubik's cube to the camera. The image is also programmatically enhanced by increasing its brightness and hue, this will make it easier to determine the colors of the cube in darker environments. Extracting numerical data from an image

Each image of a Rubik's cube face is then split into 9 sub images, this is so each sub image contains a single square from a cube face. Each sub image is then converted into an ArrayList containing the RGB value for each pixel in the sub image. The average RGB value, variance RGB value and median RGB value are then calculated from the ArrayList values. The average, variance and median are all useful numerical attributes that can be used to determine the true dominant color of an image.

### 5.2.2 Weka machine learning

Weka requires that you have a training set and a test set, both are required to be in .arff format. The training set contains numerous datasets each containing a set of attributes and its classification, whereas the test set only requires the attributes. These attributes will be Average red, Average blue, Average green, Variance red, Variance blue and Variance Green, Median Red, Median Green and Median Blue. Each set of attributes will be extracted from an image of a single square on the Rubik's cube. The training data will be classified with one of the following 6 colors, red, blue, green, orange, yellow or white. Weka will use the training data in order to make classification predictions on the test data.

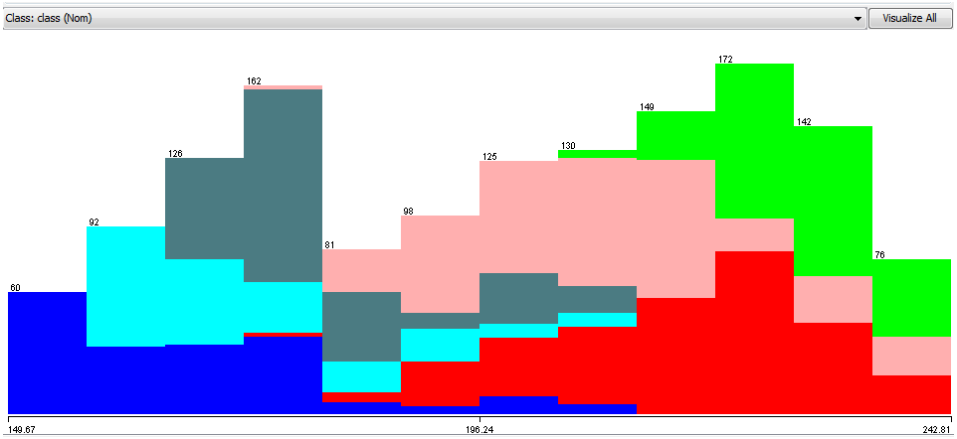
The **training set** contains datasets in the form of an .arff file, the datasets are used to train a chosen classifier which can then be used to classify dataset with an unknown classification. For this project each dataset will contain attributes about a single square on a Rubik's cube face. I evaluated various classifiers in order to identify the one that provides the best prediction rate, this can be found in the next section **(4.3).**

A new **test set** will be generated each time the user of the application requests a prediction. Weka can then be used to generate predications based on the previously generated training data.

An example of the generated .arff files can be seen in the appendix **(9.4).**

The main applications GUI will contain two modes, training mode and testing mode. This will allow training sets to be created quickly so that the visual recognition software can be quickly configured to work well in unforeseen environments. This will be explained in more detail later in section **(4.8).**

This is a visualization that was generated from within the Weka machine learning application, it shows a spatial graph depicting the various data sets inside the training data, the different colors represent the different classification given for each dataset in the training data and where it sits compared to the other colors is based on the numerical values given as attributes. It clearly shows how the various datasets can be differentiated visually and how a dataset with an unknown classification could be predicted based upon where it is positioned in the graph.




### 5.2.3 Choosing a classifier in Weka

The Weka library has various classifiers which use different algorithms to make predictions on datasets. Because they differ in their implementation they can greatly vary in their performance, therefore in order to find the most suitable classifier they must be evaluated and compared. Below is a chart showing the results from various classifiers that were tested with the training data that was previously generated as described in chapter **(4.2).** The results below are generated using 10 fold on the training set, this takes a random $10^{th}$ of the training data ten times and each time compares it with the remaining datasets. The training data being used contains 1413 instances.

| Classifier used | Correctly classified | Incorrectly classified | Time taken |
|---|---|---|---|
| **NaïveBayes** | 93.4183 % | 6.5817 % | 0.02 seconds |
| **MultilayerPerceptron** | 99.8585 % | 0.1415 % | 3.37 seconds |
| **SMO** | 98.5846 % | 1.4154 % | 0.19 seconds |
| **KStar** | 99.08 % | 0.9200 % | 0 seconds |
| **IB1** | 99.7169 % | 0.2831 % | 0 seconds |
| **RotationForest (J48)** | 99.7877 % | 0.2123 % | 0.83 seconds |
| **FT** | 99.7169 % | 0.2831 % | 0.55 seconds |

The classifiers outputted rather similar results in terms of their correct classification percentage, however there is a substantial difference in the amount of time it took to calculate a prediction. If the time taken to calculate a prediction is to long it will cause a bad user experience especially as more datasets are added to the training file. Because of this I decided to opt for the IB1 classifier. It took 0 seconds (i.e. less than 0.0001s) to calculate and performed second only to the Rotation Forest which tool much longer to calculate at 0.83 seconds. The IB1 classifier will then be integrated into the visual recognitions software and be used to predict the colors of the cube when requested by the user in real time.

IB1 or otherwise known as the “Nearest-neighbor classifier. Uses normalized Euclidean distance to find the training instance closest to the given test instance, and predicts the same class as this training instance. If multiple instances have the same (smallest) distance to the test instance, the first one found is used.” **(Weka.sourceforge.net, 2015).**

## 5.3 PROGRAMMATICALLY SOLVING THE RUBIK’S CUBE

Finding an existing cube solving algorithm that performed well and that was also compatible with Java was more difficult than I had originally anticipated. I finally found a program written in C created by Mike Reid. **(GitHub, 2010).** It is based on Kociemba's sub-optimal cube algorithm. However the program did require some modifications in order to compile and work with in conjunction with my Java application.

### 5.3.1 Modifications

The program would not originally compile on my machine due to outdated style of code. In order to compile the code I had to define data types which were not defined and replace depreciated code. And then in order to use the program in conjunction with Java I had to modify the program to take the cube state as a command line argument instead of providing input via console.

Also the program was originally designed to continue to run even after finding a solution in order to find a better solution continuously until the user exits the program. So in order to use in conjunction with the Java the program it had to be modified to complete running before its output could be passed to Java. To solve this I modified the code so it would exit after trying to find a better solution more than 14 times.

I also modified the program to return various error messages in the case of an error when calculating a solution.

### 5.3.2 Using compiled program

I then compiled the program as miker.exe, it requires a command line argument representing the cube state written in as a set of edges and corners separated by commas for example:

"UF,UR,UB,UL,LB,DB,DL,RD,FR,FL,DF,BR,LDF,RDB,DLB,RUF,LUB,DRF,FUL,BUR"

The program will output the moves required in Rubik’s cube move notations:

"Solution: F B2 U2 L' U2 F' R B2 R' D (14q, 10f)"

If the cube state given in not valid an error message will be returned instead.

### 5.3.3 Running compiled C program in Java

By using the technique of piping commands via cmd.exe in Java, C programs can be easily ran from within Java without the need for additional libraries such as JNI (Java Native Interface). However this method will make the Java application windows OS dependent unless additional code is added to support other operating systems. An example of how this is possible is shown below:

```
//cubestate = "UF,UR,UB,UL,LB,DB,DL,RD,FR,FL,DF,BR,LDF,RDB,DLB,RUF,LUB,DRF,FUL,BUR,";  //example cube state input
Process p = Runtime.getRuntime().exec("miker/miker.exe" +" "+ cubestate);
BufferedReader input = new BufferedReader(new InputStreamReader(p.getInputStream()));
```

## 5.4 Robot Design

The robots design can be split into 3 stages, cube manipulation, electronics and Programming.

### 5.4.1 Cube manipulation

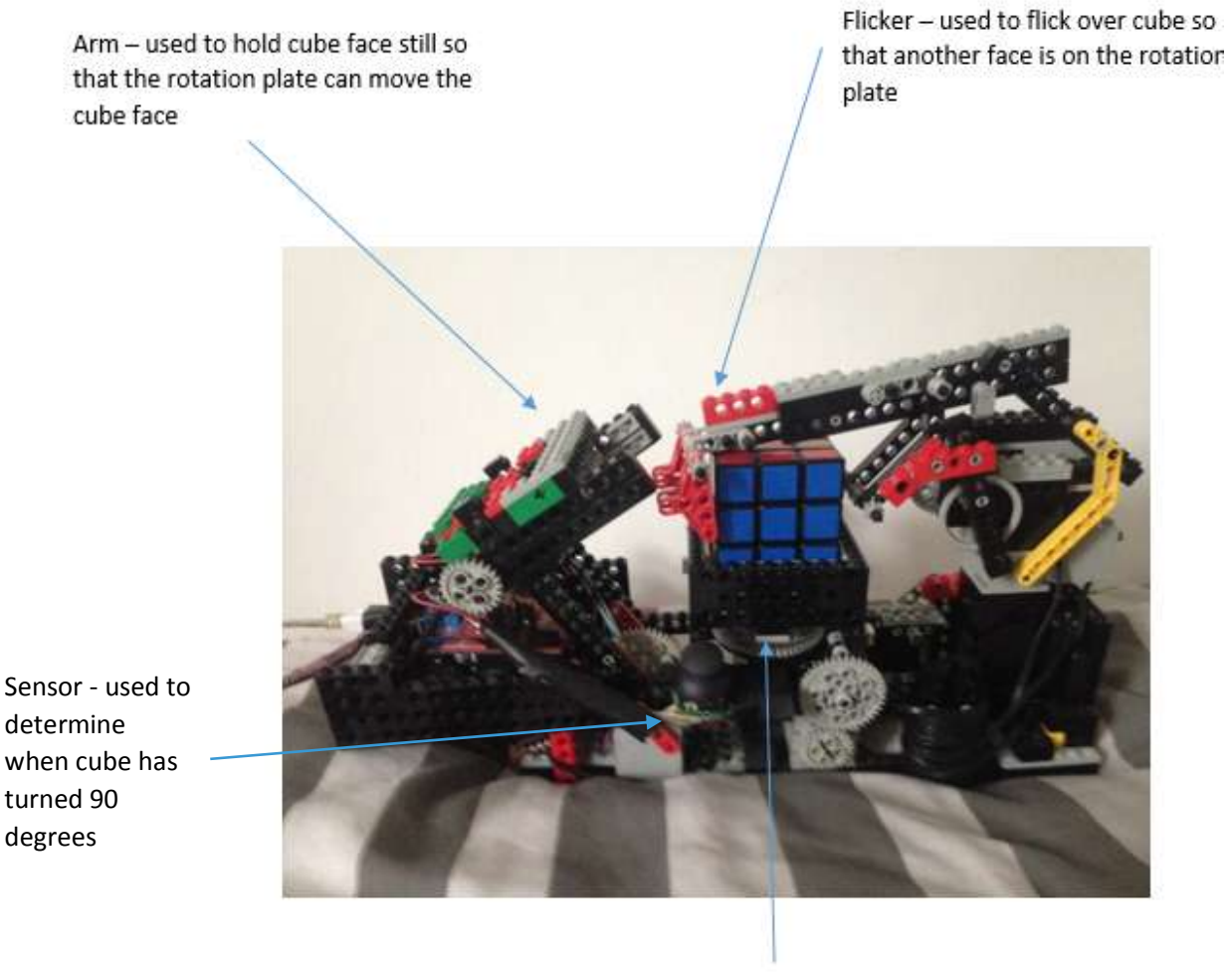
In terms of the hardware required to build the robot I decided to use the Lego RCX kit to create the skeleton and mount the motors that also came with the Lego kit. In order for the robot to manipulate the Rubik's cube to completion the following requirements had to be met:

- The robot must be able to move any face both clockwise 90 degrees, anticlockwise 90 degrees and 180 degrees.
- The robot must be able to re-orientate the entire cube without moving a face and prepare any face to be rotated next.

Based on the requirements I proposed a design made up of the following 3 mechanisms:

- Cube rotation plate – The rotation plate will be used to turn the robots faces.
- Flicker – This flicker will be used to re-orientate the entire cube so that another face is on the rotation plate
- Arm – The arm will be used to hold the cube in place in conjunction with the rotation plate which will turn a face on the cube.

The completed design can be seen below including labels for the various mechanisms used, the robot was built from scratch taking some inspiration from the YouTube videos mentioned in section **(3.4)**:



**Cube Rotation plate**

The rotation plate is where the Rubik's cube sits, below the rotation plate a cog is connected to a motor which allows the rotation plate to move 360 degrees either clockwise or anticlockwise. Custom pieces was sanded down in order to create a perfect fit for the Rubik's cube, this helped reduce the risk of errors when manipulating the Rubik's cube robotically. Multiple cogs are used in order to allow the motors to drive the rotation plate on a lower gear, this makes the rotation plate more effective when rotating a face on the cube. This principal can be compared to a manual car and the importance of using lower gears when pulling off.

**Flicker**

The flicker must not only flip the cube so that another cube face is sitting on the rotation plate, but also be able to move into resting position without manipulating the Rubik's cube. To achieve both these move sets I decided to design the flicker with 2 motors. This allowed a greater variation of move sets by attaching the motors to different areas of the flicker allowing more complex move sets to be achieved. The design took inspiration from the skeleton that attaches the wheels of a train and from an observation of the resulting movement of attaching a skeleton inside the outer radius of a wheel. This is the technique used to give the flicker its circular swiping movement which is used to flick over the Rubik's cube.

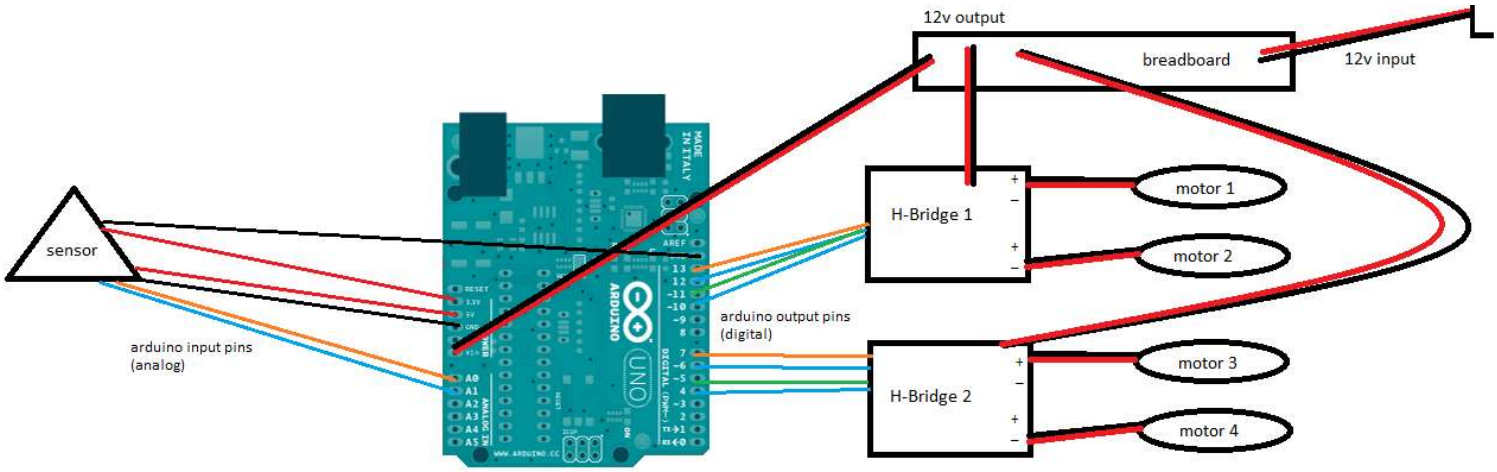
**Arm**

The Arm is attached to a cog and a single motor and can mechanically move up towards the cube to hold it in position and also back down into resting position.

### 5.4.2 Electronics

Due to the limitations of the LEGO RCX micro controller of only being able to control up to three motors per micro controller, I decided to modify the LEGO RCX kit to work with a more powerful microcontroller. I did this by modifying 2 Lego cables with some jumper cables in order to create 4 Lego socket to male pins cable connecters which can then be used to connect the Lego motors to other components.

Below is a diagram showing the peripherals and electronics used in the robot and how they interconnect with each other.




***Power supply***

In order to provide power to the H-bridges an external power source will be required, a Universal power supply that provides 12v will be used to power the *external H-bridges, motors, sensor and the Arduino*. The 12v input is supplied from the mains using a 12v 600mA power supply. This will provide enough power to drive the motors. The original microcontroller for the Lego motors was supplied 9v 700mA so I had to ensure the power supply I am using has enough watts to drive the motors effectively. Power is defined as WATTS = volts * amps. So the Lego was running at 10*700 =7000mA whereas the power supply I am using for the project runs at 12*600=7200mA. The slight increase in wattage may even lead to a slight increase in the speed of the motors can drive.

**H-bridges**

In order to give the motors bidirectional capability's an H-bridge needs to be connected to each motor. Each H-Bridge can support up to 2 motors.The H-bridges receive input from the Arduino Uno instructing it on which motors to drive and in which direction. There are two different types of H-bridges used in my design. First is a simple L298, the other is a Ardumoto which in addition to the bidirectional control feature, provides multiple speed functionality to the motors. A 12v input is supplied to both H-bridges via the breadboard.

**Motors**

The motors used are taken from the Lego RCX 1.0 kit and are just 4 simple DC Motors. The rotation movement of the motors is used to manipulate the robots skeleton which in turn will manipulate the Rubik's cube. In order to move the skeleton with precision a combination of multiple motors are required. This combination creates mechanical movements which are required in order to create robotic move sets that are capable of manipulating the Rubik's cube to completion. The motors are used in conjunction with cogs producing lower gears, this will increase the torque and provide enough power to efficiently move the motors under the tension of the Rubik's cube.

**Sensor**

A sensor will be required to detect the position of the rotation plate. This can be used to ensure the motors timing does not go out of sync. This will make it less prone to error when turning the faces of the cube due to the different level of resistance on each face. For the sensor I decided to use an analog stick from old PlayStation controller requiring some soldering to make this component compatible the Arduino. **(Instructables.com, 2015).** The sensor is mounted on the corner of the robot so that when the rotation plate reaches a 90 degree angle it will hit the sensor, this can help improve the robots accuracy by using the sensor to track orientation instead of relying on pure timing. The sensor requires 2 times 5v lines, this will be provided from the Arduino's output pins. The PlayStation controller analog is basically a potentiometer. The analog has two output pins, one for horizontal readings and the other for vertical readings. These are connected to the Arduino through its analog input pins. The analog also requires two lines of power, for this I used the Arduino 3.3v output and 5v output. I was unsure about the power requirements but both voltages seemed to power the sensor effectively, this eliminated the need for a second breadboard to distribute 5v power because 3.3V was sufficient from the remaining power output pin on the Arduino.

**Arduino Uno**

The microcontroller I choose is an Arduino Uno microcontroller, I choose this because it is more powerful in terms of hardware specifications in comparison to the outdated microcontroller that came with the Lego RCX kit, it also moderately easy to program. It also supports up to 12 separate digital input peripherals such as motors and can support up to 6 analogue inputs which can be used for sensors. Finally it supports up to 12v power input which will provide enough voltage and current to drive the motors effectively. The Arduino is used to store and execute instructions informing the various components on what they should be doing, the instructions will be uploaded to the Arduino using the cube solving program discussed in the following chapter **(5.4.3)**.

**COM Cable**

A COM cable will also be connected to the Arduino from the computer in order for the cube solving program to be able to communicate with the Arduino instructing it on the moves required in order for the robot to manipulate the Rubik's cube to completion. The COM cable will connected to a computer running the cube solving program via USB.

***Breadboard***

The breadboard receives power input from the 12v power supply and is then used to distribute the 12v input to both the Arduino Uno and both H-Bridges.

### 5.4.3 Programming

In order to communicate and write code for the Arduino I had to first install the drivers and download the Arduino IDE. The Arduino IDE requires that you compile and write code in .ino format, however it is basically a set of C / C++ functions. "All standard C and C++ constructs supported by avr-g++ should work in Arduino." **(Arduino.cc, 2015)**. In order to configure the Arduino with the IDE you must also select the COM port the device is using and also its model name. In my case it was COM3 using an Arduino UNO.

**Writing Code**

For this I began by writing a function for every possible move set the robot required for example:

*Armup();* //moves arm up to keep cube in place

*ArmDown();* // moves arm keeping cube in place down

*Rotate90Clockwise(); Rotate90AntiClockwise; RotateCube180();* //rotate the faces of the cube using arms

*Move90Clockwise(); Move90Anticlockwise(); MoveCube180();* //move cube without rotating faces

*FlickoverCube();* Use flicker to flip over cube facer // use to reorient the cube

Using these move sets it is possible to meet all the requirements necessary to manipulate a Rubik's cube to completion. Each function required tedious programming to ensure that the timing, speed and direction of the motors moves the skeleton of the robot in a way that makes the each move set possible. By combining timing with the sensor helped reduce the risk of error by not relying on pure timing to determine when the cube plate has turned exactly 90 degrees. Below is an example of how this is possible:

```
//rotate a face on the cube 90 degrees clockwise
void RotateCube90Clockwise(){
  ArmUpClockwise();  //move arm up to hold cube in place
  digitalWrite(dir_b, LOW);   //set direction of cube face to clockwise
  bstart(); // start cube face movement
  delay(7000); //wait
  while(true){
    val = analogRead(analogPin);    // read the analog input pin
    val2 = analogRead(analogPin2);    // read the analog input pin
    if (val > 800 || val2 > 600) { //when sensor reaches these values stop rotation cube
      bstop();
      delay(500);
      ArmDown();
      delay(500);
      break; //break once move is finished
    }
  }
}
```

For more code referring to the functions used by my Arduino Sketch code follow this link:

https://github.com/Jstar15/RubiksPuzzleSolver/blob/master/RubiksPuzzleSolver/robot/SketchTemplate.txt

## 5.5 Generating an Arduino Sketch File

Once the solution is returned from the cube solving program (miker.exe) the solution string must then be translated into a sketch file (.ino) to be able to run on the Arduino. I created a Java class (CreateArduinoClass.java) that would determine the robotic moves required to solve the cube based on the solution string. It must keep track of the Rubik's cubes current orientation and generate move sets that the robot is capable of achieving. The robotic moves determined are then used to generate/parse an Arduino sketch file (.ino) which can be later uploaded to the Arduino microcontroller. An example of a generated Arduino sketch file can be seen in the appendix **(9.2)**.

## 5.6 Running Arduino sketch file from within a Java application

By using this technique of piping commands via command line in windows eliminates the need for additional Java library's or frameworks on programs that are already installed on your operating system, however this is at the cost of making it windows operating system dependent and additional code will be required to make the application run on other operating systems. By using the ProcessBuilder() class in conjunction with command line arguments it is possible to run and build a sketch file onto the Arduino, is this case it would be the generated sketch file as discussed in chapter. **(5.5).** below contains code showing how a process can be built in Java using pipes to upload a sketch file on an Arduino.

```
//get directory of created sketch file
String currentsketchdir = System.getProperty("user.dir") + "\\robot\\robot.ino";

//run sketch from command line on specified port
ProcessBuilder builder = new ProcessBuilder("cmd.exe", "/c",
        "cd \"C:\\Program Files (x86)\\Arduino\" && arduino --board arduino:avr:uno --port " + comport +" --upload "+ currentsketchdir);
builder.redirectErrorStream(true);
```

## 5.7 COMBINING DESIGN ASPECTS

Below is a list of all the functionality that was required to achieve the goals of this project, following the data flow from start to finish, below examines the different design aspects of this project in terms of the functionality required and how it all fits together. Users can gain full access to all of the program features using a graphical user interface which will be discussed in more detail in the next chapter.

***Using a webcam in Java***

First an image is taken using a webcam. The images contrast and brightness are modified in order to improve visibility between the various colors of the cube faces. As discussed in chapter **(5.2.1).**

***Extracting numerical data from image***

The image is then split into 9 equal quadrants representing a square for a face on the cube. The RGB values for each quadrant is then extracted and the average RGB, variance RGB and median RGB are calculated. As discussed in chapter **(5.2.1).**

***Running Weka Classifiers from Java***

If in training mode, the numerical values representing each cube square are appended to the training data file, for predicting later in normal mode.

In normal mode the data will generated to create a file called test.arff. Once a test.arff file is generated machine learning is used to make a classification on the test file based on the existing training data. A prediction is made for each square on a cube face in relation to its color classification. As discussed in chapter **(5.2.2).**

***Running cube solving program from within the Java application***

When all the colors of each square on each face of the cube is known it can be parsed into a string known as the cube state string. The cube state string is then passed to the C program where the output is the cube solution written in cube move notation. In order to run the C program from Java I executed it using an exec command, this allows a command line argument to be ran from within Java and then returns the output. In this case I ran miker.exe, the existing cube solving program I had previously compiled as discussed in chapter **(4.3)**.

***Create Arduino sketch from cube solving solution***

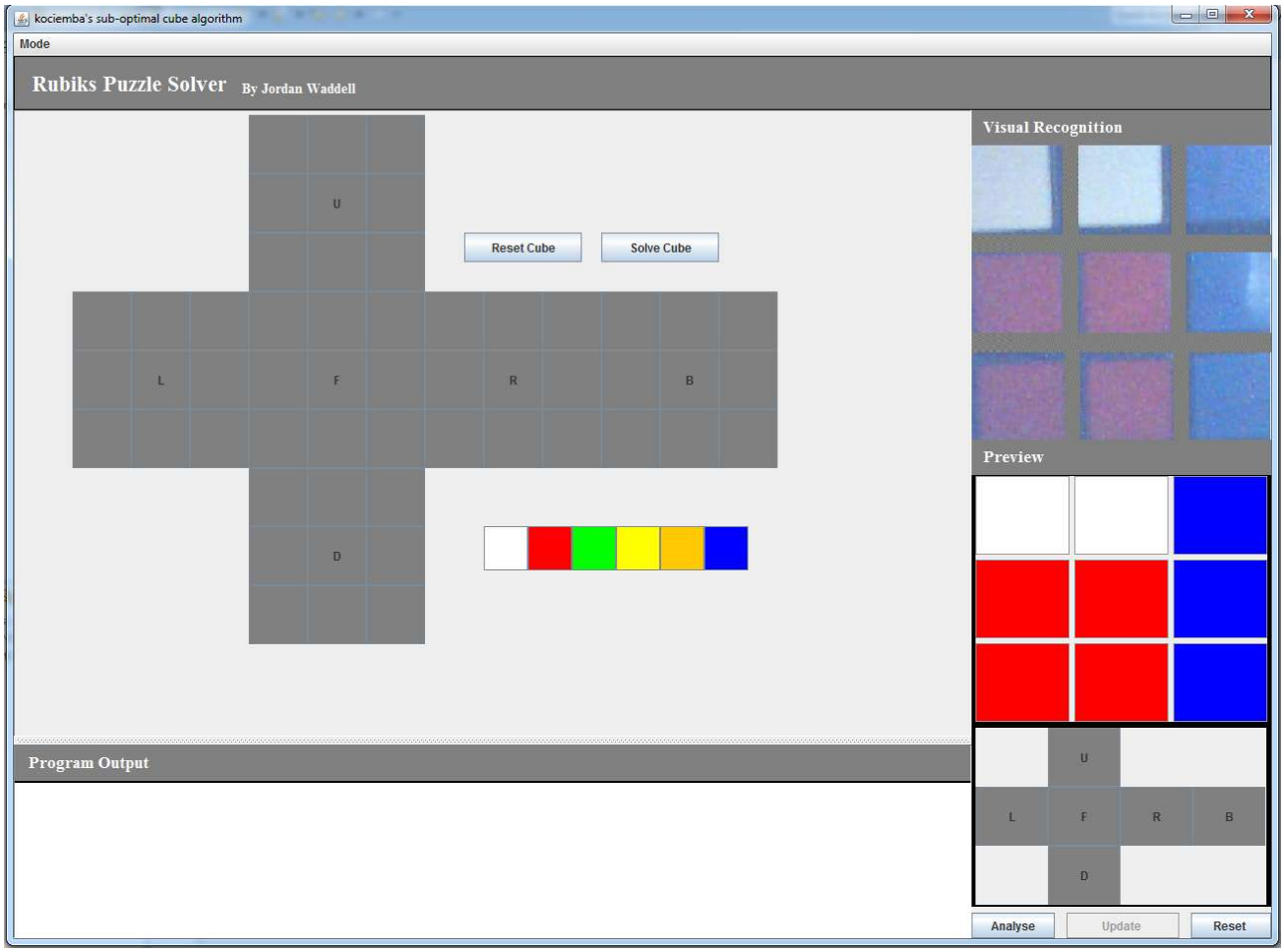
The cube solution string is then used to generate an Arduino sketch file containing the move sets required to solve the Rubik's cube as discussed in chapter **(5.5)**.

**Robotically solve the Rubik's cube**

The Rubik's cube then simply needs to be placed on the rotation plate of the robot where the sketch file can be uploaded to the Arduino as discussed in chapter **4.5.** This will then initiate the robot and the cube will be solved using robotics using the robot created as discussed in chapter **(4.4.1)**.

## 5.8 GRAPHICAL USER INTERFACE

The GUI is what allows the user to interact with all the functionality that has been developed during the scope of this project.

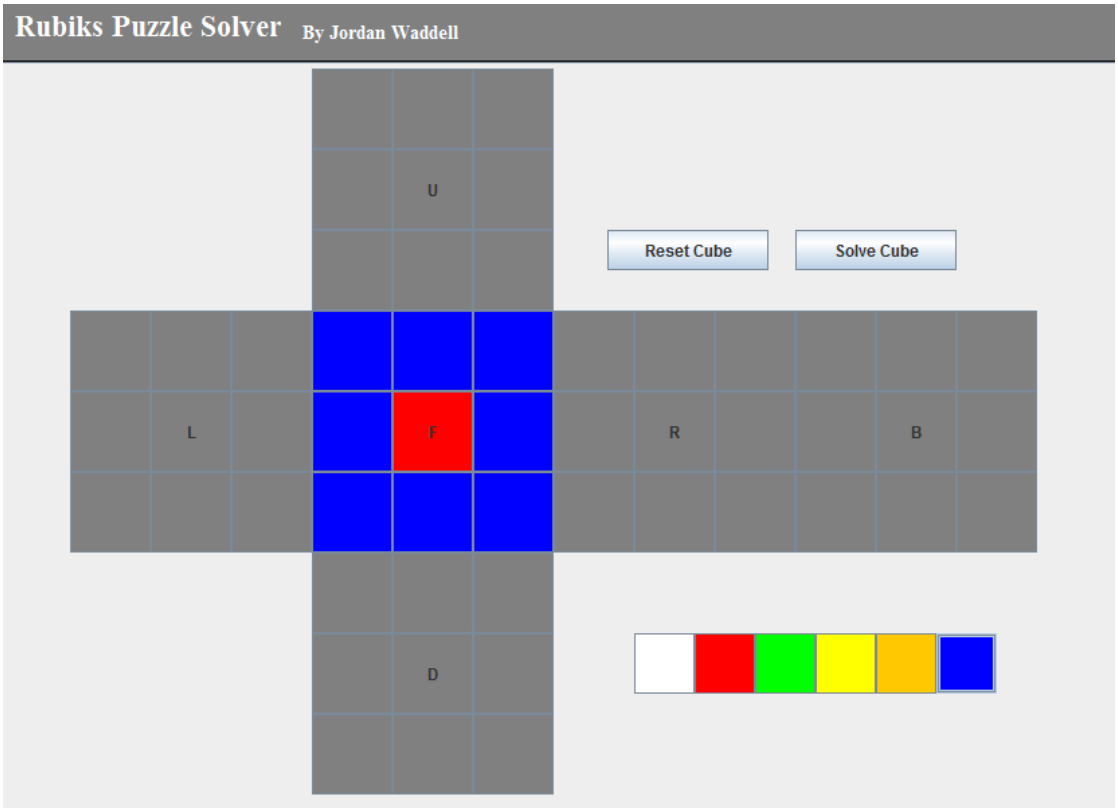



My aim was to create a simple GUI that provides the following functionality:

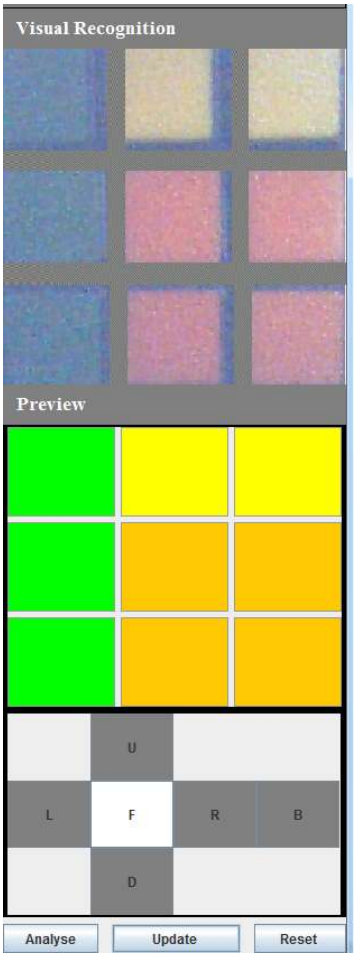
- Input cube state manually
- Input cube state using vision
- Training mode to train visual recognition in new environments.
- Return the solution to solve a Rubik's cube
- Ability to run the solution on the robotic cube solver

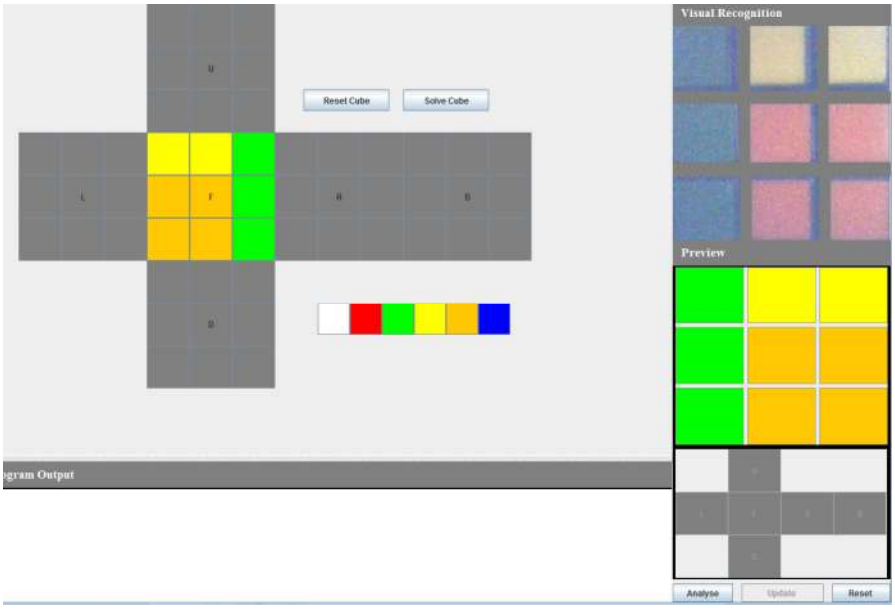
### 5.8.1 Input cube state manually

To input the cube state manually, simply select the color and select a square. Do this for every square so that all squares on all faces are identical to the Rubik's cube you are trying to solve.



### 5.8.2 Input cube state using vision

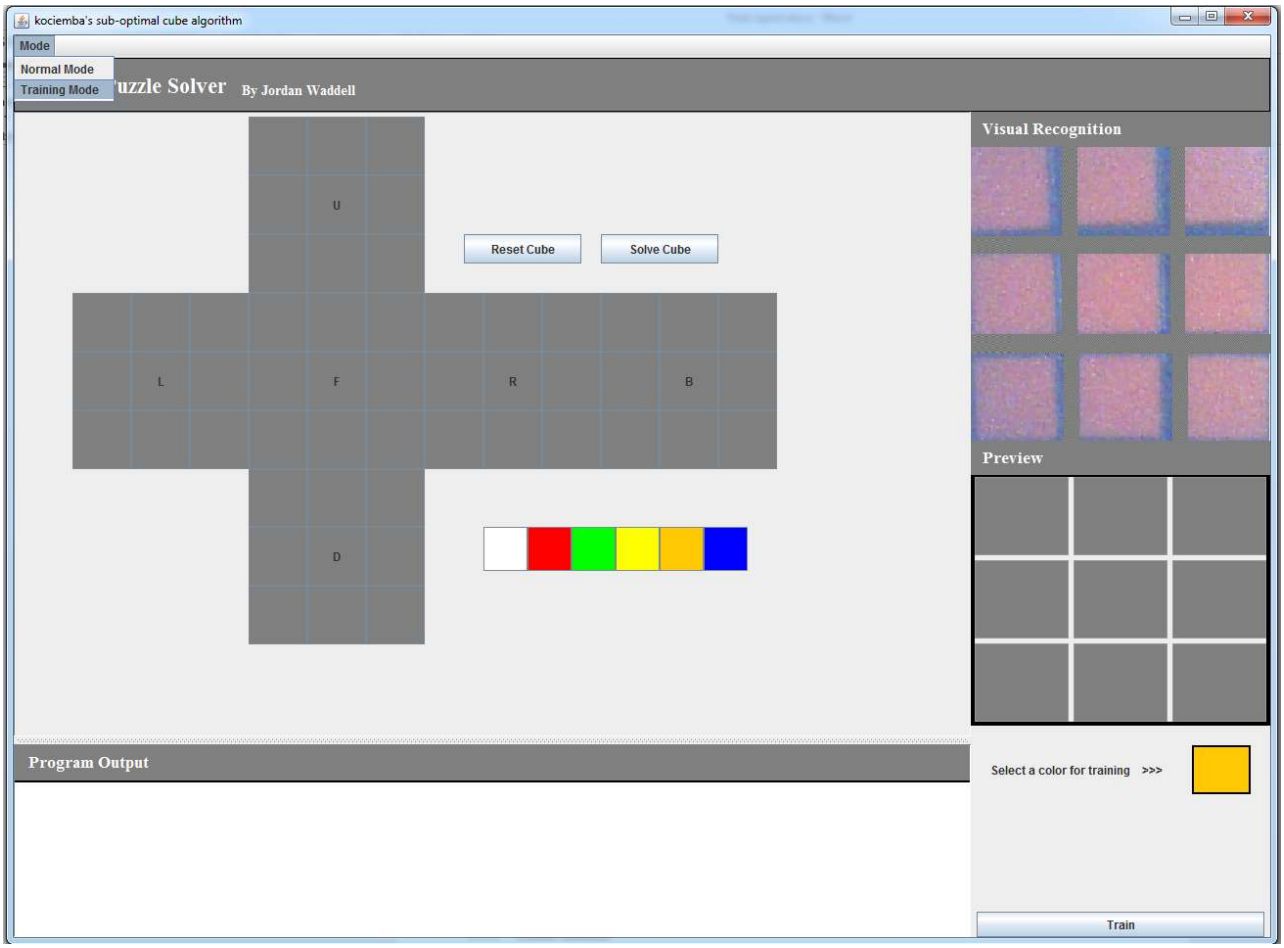



First align cube in front of webcam and select analyze. A preview will be displayed, you may need to analyze multiple times to get the desired output. Then select the face you have captured and select update. The preview will then be copied to the cube. (Note the preview had to be flipped because it's a mirror image).

The reset button will simply change the preview back to its default display.

### 5.8.3 Training Mode




Training mode allows you to add datasets to the Weka training file. This will help improve results in normal mode.

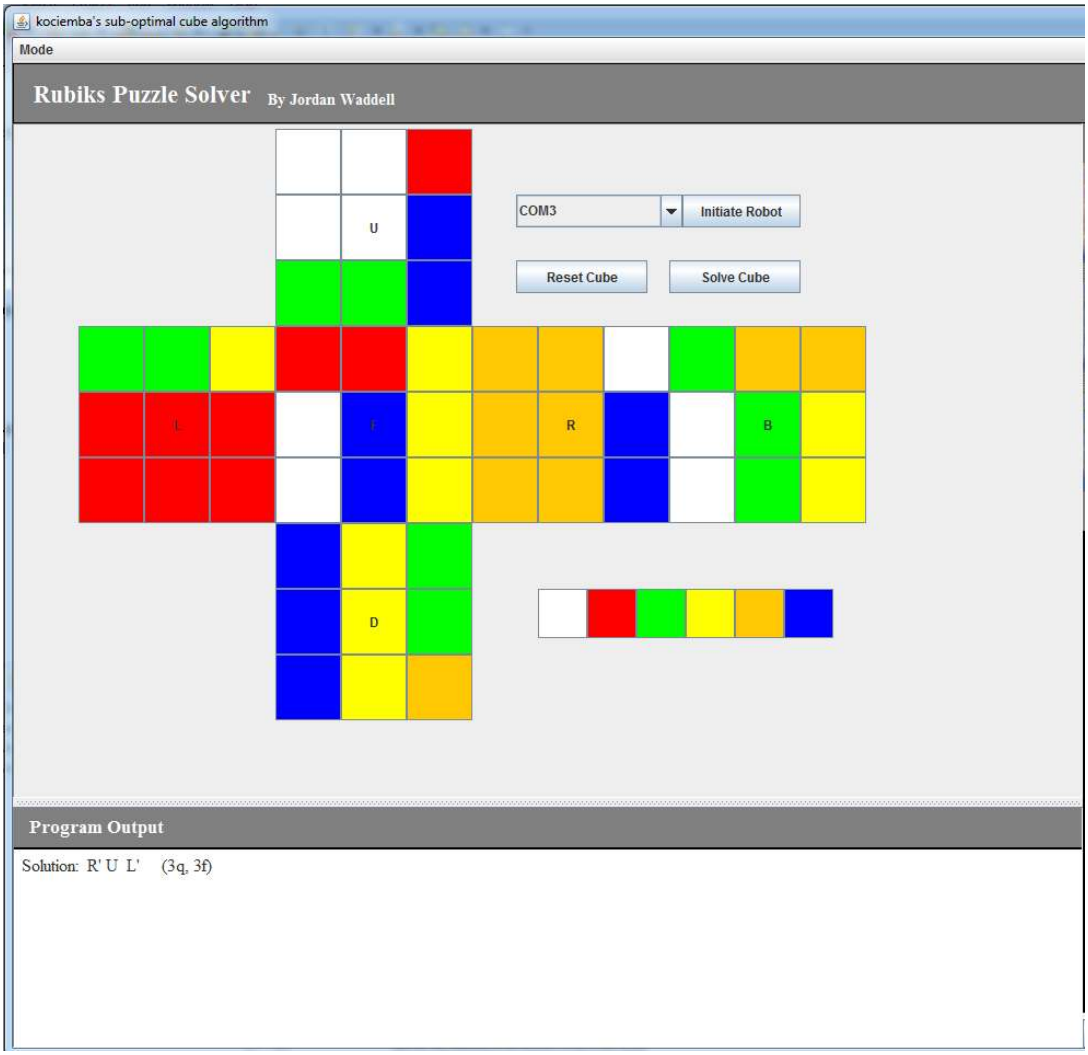
To get into Training mode select the training tab from the mode menu in the top left hand corner of the application.

Select the color you want to train, align cube showing a cubes face with only the specified color in the camera preview and select train.

NOTE: It is important that you only train the desired color with a cube face that is only that color. This ensures that false data is not added to the training set. This would ultimately damage the prediction rate.

### 5.8.4 Program Output

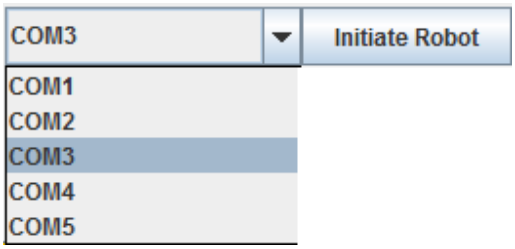
Once all faces of the cube have been inputted select the solve button. Output will then be generated in the programs output panel, the output will be in the form of a string containing the moves required to solve the Rubik's cube written in cube move notation as discussed in chapter **(3.2).**




Note: If the programs output is successful an initiate robot option will appear. Else an error message will be displayed informing the user with useful feedback on what went wrong.

### 5.8.5 Initiate Robotics

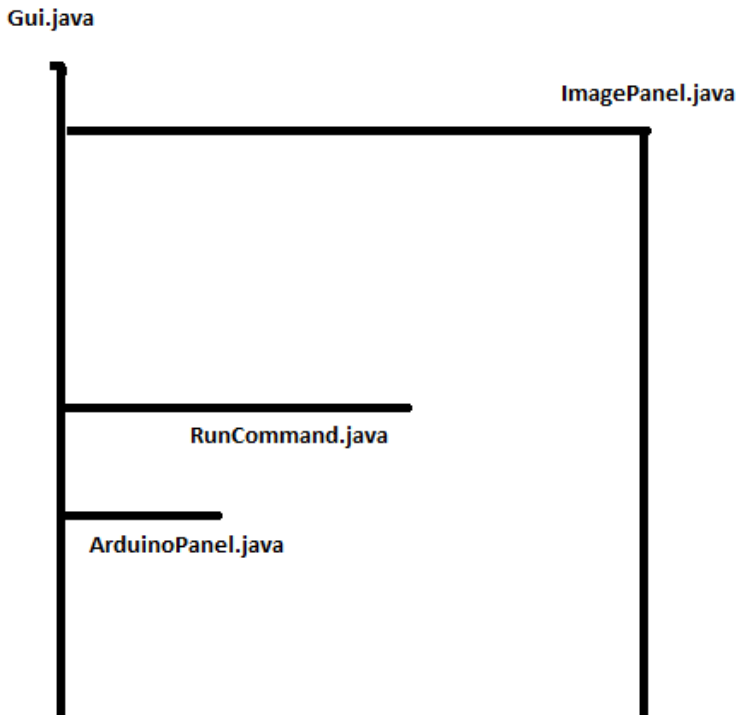
First put the Rubik's cube into the robots rotation plate, select the com port number that connects to the Arduino microcontroller and select the initiate robot button. The Arduino file generated will then be uploaded to the robot where upon completion will robotically solve the Rubik's cube.




NOTE: When physically placing the Rubik's cube into the robot side (U) must face up and side (F) must face towards the flicking mechanism of the robot.

## 5.9 Multiple Threads

Multiple threads are required in this application to allow multiple functionality to run concurrently with each other. Below is a diagram showing the multiple threads that are initiated during the applications life cycle and including a small description of the purpose of each thread.



***Gui.java***

This Graphical user interface is initiated on the main thread. This will always run-in its own thread to avoid lag to the user.

***ImagePanel.java***

This class is used to continuously refresh the image displayed as a preview. The user would not be able to interact with the application if this was not in its own thread.

***RunCommand.java***

This class is called to run the cube solving program that is written in C to run in Java, the program can sometimes take up to 30-50 seconds before returning an output depending on the cube state inputs solving difficulty.

***ArduinoPanel.java***

This class is used to upload the sketch to the Arduino microcontroller.

## 5.10 Error Handling

In order to reduce the risk of errors in the application and provide useful output to users errors had to be captured and displayed to user were appropriate.

***Webcam detection***

If a webcam is not detected the vision panel will not be displayed and an error will be displayed to the user. This stops errors when a user try's to use the vision functionality when a webcam is not plugged in.

**Cube input incorrect**

If the cube state input is incorrect an error message will be displayed to the user in the output panel alerting the user of the error type.

**File error**

There should never be any file errors as data streams are closed when appropriate, in the rare case of any possible errors exceptions have been added to the code where necessary.

## 5.11 Documentation

**ReadMe file**

Aside from this report I have provided a readme file with the software implementation that provides user requirements and direction for software use. The readme file has been added to the appendix.

**Video file**

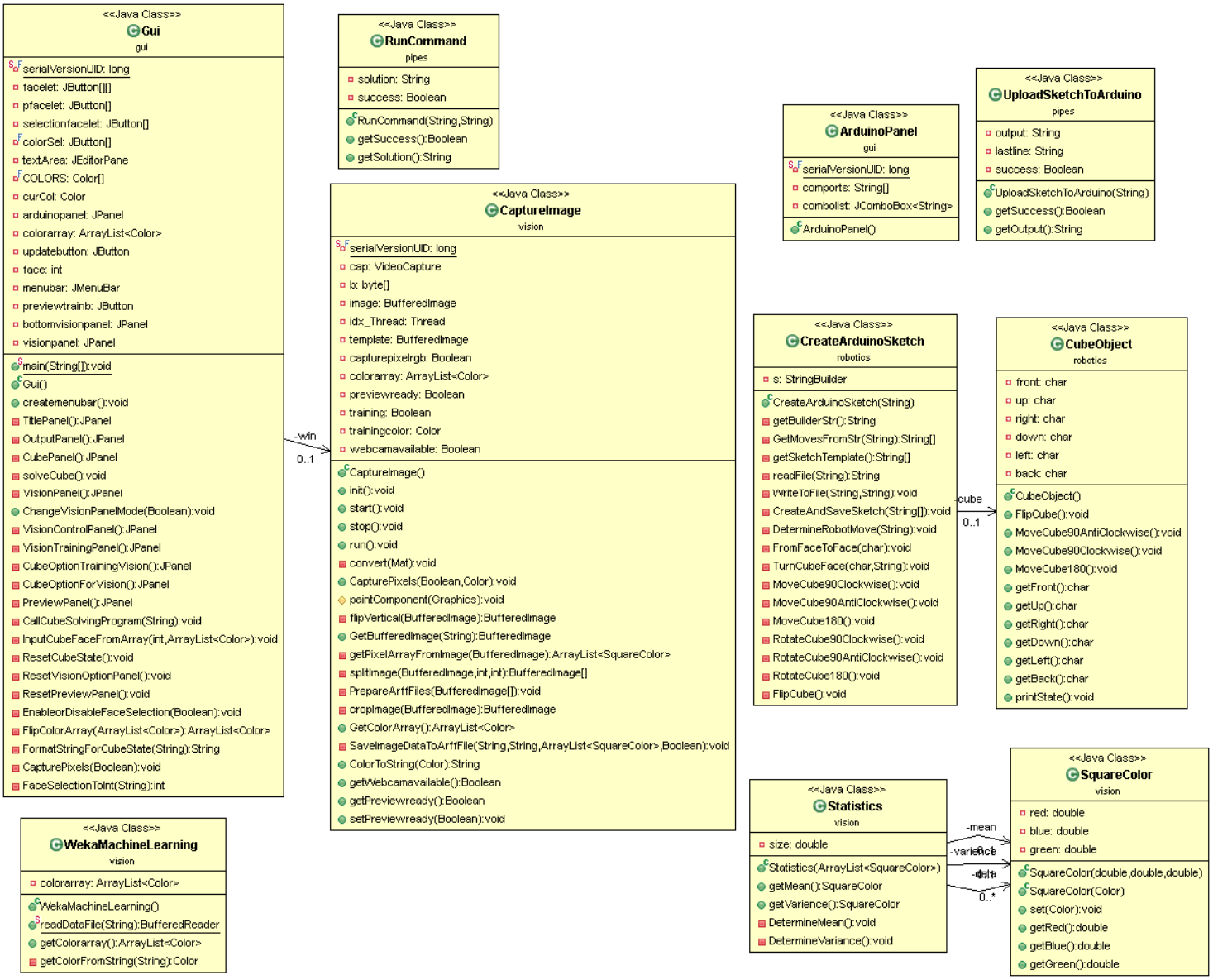
The project lifecycle is demonstrated in the following video I uploaded to YouTube, It shows the visual recognition software in use and also demonstrates how the robot works.

https://www.youtube.com/watch?v=VPhksyqhGTY

## 5.12 Software Architecture Diagrams

### 5.12.1 UML Class diagram

This diagram shows all Java classes, functions and variables used in the finished application.

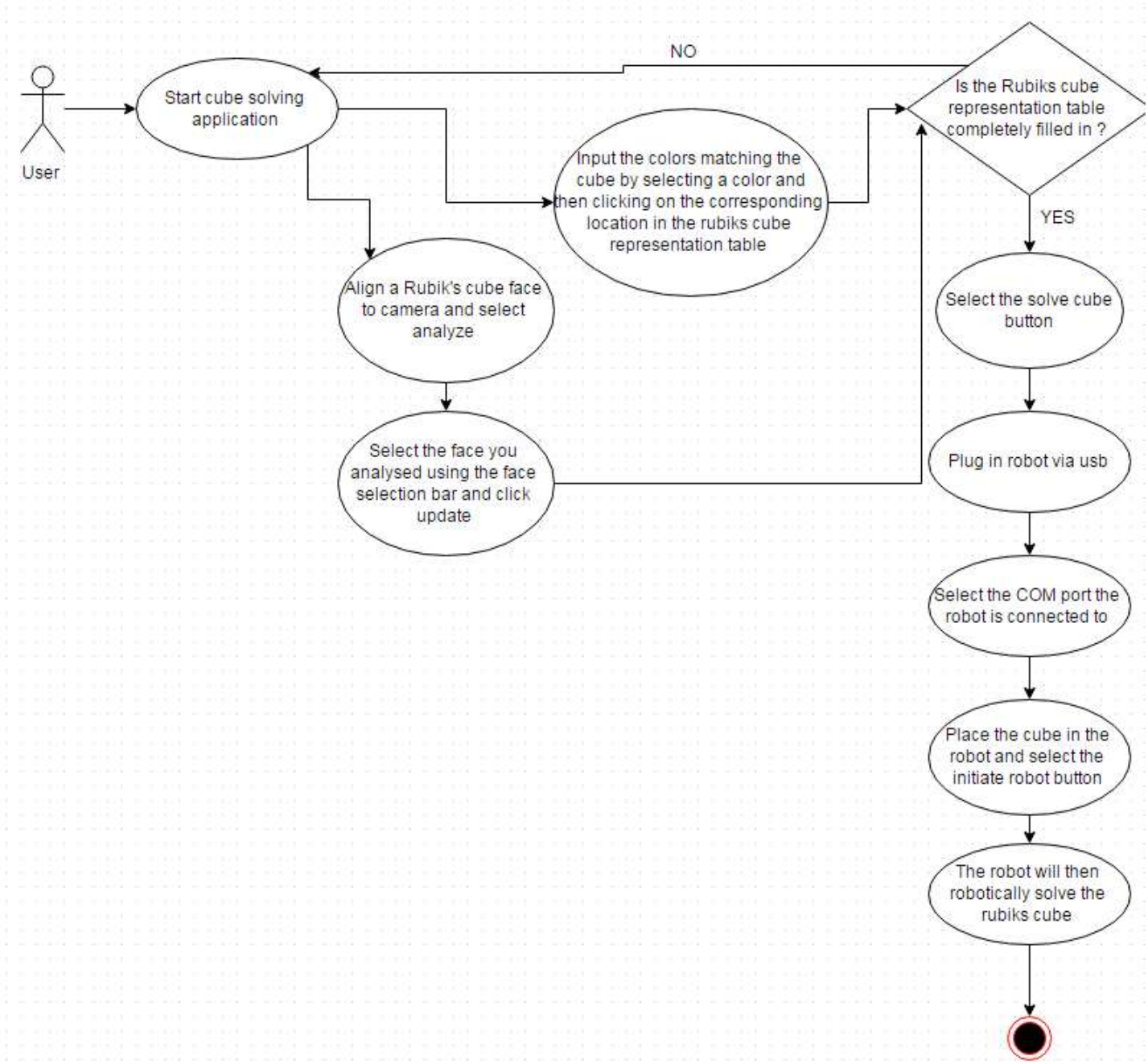



A full scale class diagram can be found at:

https://github.com/Jstar15/RubiksPuzzleSolver/blob/master/RubiksPuzzleSolver/images/classdiagram.gif

### 5.12.2 UML Case Diagram

This UML diagram describes the applications life cycle for robotically solving a Rubik's cube.



# 6 EVALUATING AND TESTING

## 6.1 EVALUATION OVERVIEW

This Evaluation will be split into the 4 following criteria:

1. Minimal requirements evaluation
2. Visual recognition evaluation
3. Robot performance evaluation
4. Effectiveness of chosen development methodology

Each criterion will help evaluate the success of the different aspects of this project:

## 6.2 MINIMUM REQUIREMENTS EVALUATION

I feel that all the following functional requirements of this project have been satisfied:

- ***Visual capabilities*** –Using the application the cube state can be input using the webcam to detect the colors for each face on the Rubik's cube. By using vision the cube state can be input into the cube solving application much quicker than if inputting manually. The color detection also works well in a wide variety of lighting environment due to the implementation of machine learning.

- ***Determine solving solution*** – The program is capable of providing a solution for a given cube state and then able to determine which instructions to give to the robot in order to solve the cube. This functionality works as intended.

- ***Robot capabilities*** – The robot works as intended and is capable of manipulating the Rubik's cube to completion without any human interaction.

- ***Graphical user interface*** – The graphical user interface meets the requirement of providing the user with access to the above functionality in a simple easy to use application.

## 6.3 VISUAL RECOGNITION EVALUATION

In order to evaluate the effectiveness of the visual recognition software I decided to test the visual recognition on a Rubik's cube in 3 different environments with each environment containing a different intensity of light. The 3 environments will be following:

- In front of the window next to sunlight. (Bright)
- In a dimly lit room using a lamp (dim)
- In a room with the curtains closed (dark)

For this test a Rubik's cube will be randomly scrambled and each face of the Rubik's cube will be captured using the visual recognition feature in the GUI application. This will be repeated 3 times for each environment and the results will be recorded in regards to prediction accuracy. The prediction accuracy will be calculated by counting the correct color classification given for each square on the Rubik's cube. This number will then be divided by 9, 9 is the total number of each colors occurrence on a Rubik's cube. That number is than multiplied by 100 to get a percentage.

The following condition are met:

- Before testing in each new environment, training sets will be generated and will be added to the training data from images captured from a Rubik's cube in that environment. This will increase prediction accuracy and help ensure that any particular environment won't get an advantage based on the previously generated prediction datasets in the training data.
- When capturing and predicting the squares from each face of a Rubik's cube a maximal of 5 recaptures are allowed. This is because the Rubik's cube may need to be moved around in order to align the Rubik's cube to the camera properly.
- The Rubik's cube will be scrambled in the exact same way for all tests to ensure any particular test instances does not gain an advantage, this could be due to the possibility that some cube states are harder to visually recognize then others because of the varying levels of distortion between neighboring colors on a Rubik's cube.

The test results have been recorded and can be seen in the following table:

| Test no | Lighting Condition | Red accuracy (%) | Blue accuracy (%) | Orange accuracy (%) | Green accuracy (%) | Yellow accuracy (%) | White accuracy (%) |
|---|---|---|---|---|---|---|---|
| **1** | Bright | 100% | 100% | 100% | 100% | 100% | 100% |
| **2** | Bright | 100% | 100% | 100% | 100% | 100% | 100% |
| **3** | Bright | 100% | 100% | 100% | 100% | 100% | 100% |
| **4** | Dim | 100% | 100% | 66.66% | 88.88% | 88.88% | 77.77% |
| **5** | Dim | 100% | 100% | 66.66% | 100% | 88.88% | 100% |
| **6** | Dim | 100% | 100% | 55.55% | 100% | 100% | 88.88% |
| **7** | Dark | 88.88% | 33.33% | 0% | 100% | 0% | 0% |
| **8** | Dark | 88.88% | 0.11% | 0% | 100% | 0% | 0% |
| **9** | Dark | 77.77% | 33.33% | 0% | 100% | 0% | 0% |

The visual recognition works well and if the lighting conditions are not too dark and a training set is created beforehand. Results of 100% prediction accuracy can be achieved if the lighting is sufficient. Under darker conditions the visual recognitions accuracy decreases as it can sometimes struggle between the colors green and blue or red and orange. However this is to be expected as even we humans have our perception thresholds when detecting similar colors in environments with harsh lighting conditions.

## 6.4 ROBOTICS PERFORMANCE EVALUATION

In order to evaluate the robots performance I decided to run a small experiment. The experiment will begin by first randomly scrambling the Rubik's cube, the cube will be scrambled incrementally from 1 to 12 moves. Each time the scrambled cube will then be input into the cube solving program where the robot will be initiated to solve the cube. The robot will be timed how long it takes it to solve the cube. If the robot requires any human assistance it will be counted as an error. Also the amount of moves required to solve the cube will be noted.

| Test instance no | Randomly Scrambled cube By x moves. | Solving solution moves required Where q is quarter turns | Time Taken (MM.SS.ss) | error count |
|---|---|---|---|---|
| **1** | 1 | 1q | 00:17:51 | 0 |
| **2** | 2 | 2q | 00:24:90 | 0 |
| **3** | 3 | 3q | 01:01:75 | 0 |
| **4** | 4 | 4q | 01:25:07 | 0 |
| **5** | 5 | 7q | 01:45:23 | 1 |
| **6** | 6 | 6q | 01:48:38 | 0 |
| **7** | 7 | 8q | 02:12:05 | 5 |
| **8** | 8 | 8q | 02:40:23 | 3 |
| **9** | 9 | 9f | 02:59:16 | 3 |
| **10** | 10 | 11q | 03:20:50 | 2 |
| **11** | 11 | 11q | Failed to complete cube | 1 |
| **12** | 12 | 12q | 3:21:56 | 2 |

All errors in this evaluation was caused by the cube sometimes not locking into rotation plate after being flicked over, the fix is to simply push cube back into rotation plate so it can continue as normal. The Rubik's cube was completed by the robot in all test instances except on test instance 11 where I was unable to recover the cube in time and the robot got out of sync therefore failed at completing the cube.

## 6.5 EFFECTIVENESS OF CHOSEN DEVELOPMENT METHODOLOGY

I feel the Xtreme Programming methodology chosen for this project was very suitable. By splitting the project up into small manageable chunks that can be independently unit tested helped minimize the risk of errors throughout the project. This helped keep the class structures clean by separating code that isn't relevant to each other. This allowed easier debugging because different features could be tested independently of each other. This was particularly useful when implementing the visual recognition software as there was many logical steps required to achieve the end goal.

In comparison to the more popular development approaches such as the waterfall model where going back to rewrite code is not encouraged, XP prefers to implement simpler functionality first and more advanced functionality later. When building the robot skeleton the exact design and structure required was unknown due to the experimental nature of this project, therefore designing and building the robot was a continuous ongoing process following only rough design concepts in order to come up with a solution, rigorous testing, redesigning and innovation was required throughout the project in order to continuously improve on the robots performance. The XP methodology allowed for me to anticipate this is my initial project plan and as a result reduce the risk of falling behind schedule.

The project management techniques used to manage my time effectively throughout the course of this project such as the Gantt chart and PERT diagram helped to keep track of progress which in effect allowed me to reach my deadlines and as a result stay on schedule.

# 7 CONCLUSION AND FUTURE IMPROVEMENTS

## 7.1 FURTHER IMPROVEMENTS

Below lists the various ways in which the results of this project could be improved given more time or resources.

***Visual Recognition improvements***

- The visual recognition could have been improved if a better camera was used which could adjust to the various lighting conditions more effectively and as a result the colors of the cube squares would be more consistent and could have been more accurately determined.
- By adding more training sets to the training data prediction rates will continue to improve. However more time would be required in order to capture training data from images taken from the Rubik's cube in a wider variety of environments.
- Prediction rates could have been improved if more advanced machine learning strategies were used techniques such as Boosting, Bagging or blending Methods. **(Brownlee, J. 2014).**

**Robotics improvements**

- By using a more powerful power supply the motors would have been faster and as a result solve the cube in a faster amount of time. Too much voltage however could damage the DC motor so various testing would be required.
- If more time was spent perfecting the robots original skeleton, the end product may have been able to totally eliminate the need for human interaction due to the robot making fewer mistakes.
- The Lego pieces available for construction was limited and therefore added to the difficulty of implementing a design. The newer Lego kits provide the opportunity to create more innovative designs due to the newer kits containing a larger variety of pieces which could have been used to simplify the designing and building process.
- If servo motors was used instead of regular dc motors then a sensor would not have been required, this is because a servo motor provides feedback allowing accurate control over the angular positioning of the motor. Using a servo motor instead would have simplified the programming process as getting robotic moves to work exactly the same each time would be simplified.

**Extensions**

- Adding the training mode to the application was an extension I made to enable the user to easily add training sets to the training data. This allows the user to quickly create training sets taken from new environments allowing the application to evolve and give more accurate predictions with time as more training sets are added to the training data.

## 7.2 CONCLUSION

I feel the visual recognition has performed well and demonstrates how machine learning can be used to build evolving applications were future performance is improved through collecting additional data in order to make better decisions. Considering the webcam that was used for this project is over 15 years old I was very happy with the positive results. I feel the visual recognition makes it much quicker and easier to input the Rubik's cube current state and makes the application much less intensive on the user. As more datasets get added to the training set the prediction accuracy will improve due to the machine learning algorithms having more data which in turn enables better decision making.

The robotics could have performed better in terms of minimizing errors in order to completely eliminate the need for human intervention. I feel that if the flicker was redesigned at an earlier stage in the project error could have been dramatically reduced. On the other hand I feel the robot clearly demonstrates the ability to mechanically move the faces of the cube according to instructions and I feel it meets the aims and requirements of this project. The robot however is quite reliable and although it is not the fastest robot in the world it is indeed fully capable of manipulating a Rubik's cube to completion.

Although there was a lot of interconnectivity going on between different programs on multiple platforms (pipes), I feel the GUI was well designed in terms of bringing all the functionality together into simple easy to use user friendly application.

Overall I am happy with result of this project and I feel it has demonstrated how a simple robot can be built in combination with vision to solve a 3D puzzle. I feel this project has definitely fulfilled the goals and requirements that was defined at the beginning of this report.

# 8 BIBLIOGRAPHY

**Rubiks (2015).** The History of the Rubik's Cube. [online] Available at:
http://uk.rubiks.com/about/the-history-of-the-rubiks-cube
[Accessed 23 Feb. 2015].

**Mathworld.wolfram.com (2015)**. Rubik's Cube -- from Wolfram MathWorld. [online] Available at:
http://mathworld.wolfram.com/RubiksCube.html
[Accessed 23 Feb. 2015].

**Commons.wikimedia.org, (2015).** File:Rubiks 18.svg - Wikimedia Commons. [online] Available at:
http://commons.wikimedia.org/wiki/File:Rubiks_18.svg?uselang=en-gb
[Accessed 23 Feb. 2015].

**Cube20.org, (2015).** God's Number is 20. [online] Available at:
http://www.cube20.org/
[Accessed 24 Feb. 2015].

**Wikipedia, (2015).** *RGB color model*. [online] Available at:
http://en.wikipedia.org/wiki/RGB_color_model
[Accessed 7 Apr. 2015].

**Optimal solutions for Rubik's Cube, (2015).** *Wikipedia*. [online] Available at:
http://en.wikipedia.org/wiki/Optimal_solutions_for_Rubik%27s_Cube
[Accessed 8 Apr. 2015].

**Instructables.com, (2015).** *PS1 Controller Joysticks with Arduino*. [online] Available at:
http://www.instructables.com/id/Extracting-the-PS1-Controller-Joysticks/
[Accessed 8 Apr. 2015].

**Arduino.cc, (2015).** *Arduino - FAQ*. [online] Available at:
http://arduino.cc/en/Main/FAQ
[Accessed 8 Apr. 2015].

**Commons.wikimedia.org, (2014).** *File:Named Rubiks cube.png - Wikimedia Commons*. [online] Available at:
http://commons.wikimedia.org/wiki/File:Named_Rubiks_cube.png
[Accessed 9 Apr. 2015].

**Astro.berkeley.edu, (2015).** *Basic Notation - Understanding the Rubik's Cube - Joe Converse*. [online] Available at:
http://astro.berkeley.edu/~converse/rubiks.php?id1=basics&id2=notation
[Accessed 9 Apr. 2015].

**Anon, (2015).** [online] Available at:
https://www.cs.sfu.ca/~mark/ftp/Icip2010/icip2010.pdf
[Accessed 10 Apr. 2015].

**BBC News, (2015).** *Optical illusion: Dress colour debate goes global*. [online] Available at: http://www.bbc.co.uk/news/uk-scotland-highlands-islands-31656935 [Accessed 10 Apr. 2015].

**Wikipedia, (2015).** *Extreme programming*. [online] Available at: http://en.wikipedia.org/wiki/Extreme_programming [Accessed 15 Apr. 2015].

**Weka.sourceforge.net, (2015).** *IB1*. [online] Available at: http://weka.sourceforge.net/doc.stable/weka/classifiers/lazy/IB1.html [Accessed 17 Apr. 2015].

**Brownlee, J. (2014).** Improve Machine Learning Results with Boosting, Bagging and Blending Ensemble Methods in Weka - Machine Learning Mastery. [online] Machine Learning Mastery. Available at: http://machinelearningmastery.com/improve-machine-learning-results-with-boosting-bagging-and-blending-ensemble-methods-in-weka/ [Accessed 19 Apr. 2015].

**GitHub, (2010).** jdkoftinoff/mb-linux-msli. [online] Available at: https://github.com/jdkoftinoff/mb-linux-msli/blob/master/uClinux-dist/user/games/rubik/miker.c [Accessed 20 Apr. 2015].

**Upload.wikimedia.org, (2015).** [online] Available at: http://upload.wikimedia.org/wikipedia/commons/c/c2/AdditiveColor.svg [Accessed 28 Apr. 2015].

# 9 APPENDIX

## 9.1 PERSONAL REFLECTION

This project was quite enjoyable and I am happy with my choice of project. It was my first time working with robotics so enjoyed learning a bit more about how it all comes together. Building the robot from Lego was probably the longest part of the project, quite a few design iteration were built before finalizing the design. Programming the motors in conjunction with sensor was a bit fiddly, time was wasted on a failed attempt to build a sensor made out of tin foil which I unfortunately could not get to work 100% of the time.

Overall I am happy with the results of the project and am glad I picked an interesting project that had plenty of material to cover ensuring there was plenty to talk about in this report. This is by far the longest project I have ever undertaken and although the end product won't actually benefit anybody I was still ultra-satisfying the first time the robot actually solved the Rubik's cube. Of course there are things I would change if I was doing the project again in terms of the design implementation, however I would not change the way I approached the problem throughout the course of this project in terms of both the methodology used and the way the project was managed.

## 9.2 EXAMPLE OF A GENERATED SKETCH FILE

Below is an example showing the generated moves required for the robot to solve the cube showing only the function calls that were generated as a sketch file.

```
34 void loop(){
35 MoveCube90Clockwise();FlickOverCube();RotateCube90AntiClockwise();MoveCube90Clockwise();FlickOverCube();
36 RotateCube90AntiClockwise();FlickOverCube();RotateCube90AntiClockwise();FlickOverCube();RotateCube90AntiClockwise();
37 MoveCube90AntiClockwise();FlickOverCube();RotateCube90AntiClockwise();MoveCube90AntiClockwise();FlickOverCube();RotateCube90AntiClockwise();
38
39 while(1);
40 }
```

A full detailed description of the functions called can be seen on the sketch template at:

https://github.com/Jstar15/RubiksPuzzleSolver/blob/master/RubiksPuzzleSolver/robot/SketchTemplate.txt

## 9.3 SOFTWARE README FILE

README.md

# RubiksPuzzleSolver

Solve a Rubik's cube using vision and robotics. Final year project.

## System Requirements:

Operating System: Microsoft Windows 7 (x64)

## Software Requirements:

Eclipse IDE (or alternative)

OpenCV 3.0 beta (environmental variables must be set)

Weka 3.6.12

Java VM 1.7

Arduino 1.6.4 IDE software

## Hardware Requirements:

A Webcam: software tested using a Logitech QuickCam Pro 3000

A Rubik's cube solving robot (For robotic capabilities)

## Prerequisites

User libraries must be added to the Java project for both openCV and Weka so they can be added to the build path on runtime. The JRE system library must also be added to the build path.

## Usage

First start by opening the project in eclipse and then run Gui.java located in the src/gui/ package

### Normal Mode

First align cube in front of webcam and select analyse. A preview will be displayed, you may need to analyse multiple times to get the desired output. Then select the face you have captured and select update. The preview will then be copied to the cube. (Note the preview will be flipped because it's a mirror image). The reset button will simply change the preview back to its default display.

Alternatively you can input the cube state manually. This can be done by simply selecting a cube colour and then selecting the corresponding square on the cube representation diagram. Do this for every square so that all squares on all faces are identical to the Rubik's cube you are trying to solve.

Once all faces of the cube have been input into the cube representation diagram select the solve button. Output will then be generated in the programs output panel, the output will be in the form of a string containing the moves required to solve the Rubik's cube.

Then put the Rubik's cube into the robots rotation plate, select the COM port number that connects to the Arduino microcontroller and select the initiate robot button. The Arduino file generated will then be uploaded to the robot where upon completion will robotically solve the Rubik's cube.

### Training Mode

Training mode allows you to add datasets to the Weka training file. This will help improve results in normal mode. To get into Training mode select the training tab from the mode menu in the top left hand corner of the application. Select the colour you want to train, align cube showing a cubes face with only the specified colour in the camera preview and select train. NOTE: It is important that you only train the desired colour with a cube face that is only that colour. This ensures that false data is not added to the training set. This would ultimately damage the prediction rate.

## Credits

The cube solving algorithm implemented in this application was developed by Mike Reid. The program is written in C and was modified to work with this project and was found here.

https://github.com/jdkoftinoff/mb-linux-msli/blob/master/uClinux-dist/user/games/rubik/miker.c

## 9.4 GENERATED .ARFF FILES

These files were both generated from within the Java application and are used by Weka to make classification predictions.

**Training set (.arff file)**

colortrain.arff

```
@relation color
@attribute avgblue numeric
@attribute avgred numeric
@attribute avggreen numeric
@attribute varblue numeric
@attribute varred numeric
@attribute vargreen numeric
@attribute midblue numeric
@attribute midred numeric
@attribute midgreen numeric

@attribute class {red,blue,green,orange,yellow,white,unknown}

@data

232.795245398773,219.96993865030674,223.3638036809816,1563.6508669645991,3486.810752756071,2973.386665287312,255.0,255.0,255.0,white
231.09769938650308,217.96319018404907,221.32423312883435,1731.2927554421756,3726.5587677367425,3241.5157317740704,255.0,255.0,255.0,white
224.93765337423312,205.3717791411043,211.66487730061348,1915.1322325286378,4539.71653486967,3718.1076314279417,255.0,255.0,255.0,white
236.68159509202454,227.70306748466257,229.3256134969325,1283.9512256767953,2663.7661255594976,2375.1833930280955,255.0,255.0,255.0,white
221.17039877300613,208.604754601227,210.2019938650307,1635.960841559016,2704.6847319943554,2642.9176340605995,244.0,226.0,232.0,white
227.52507668711655,213.60950920245398,217.69670245398774,1878.4711503002093,3847.966535343403,3309.1608480221375,255.0,255.0,255.0,white
222.13527607361962,205.06564417177916,208.77101226993864,2218.482313880451,4556.765936241396,4119.942810019388,255.0,254.0,255.0,white
225.27975460122698,210.414263803681,213.45644171779142,2053.086768038575,4365.6141216992255,3837.557305130243,255.0,255.0,255.0,white
217.9476226993865,198.80828220858896,204.16786809815952,2446.126321036422,5211.307722813944,4447.334320302837,255.0,255.0,255.0,white
232.73466257668713,220.06625766871164,223.2187116564417,1582.1506083219815,3484.1416221905724,2989.8625946588522,255.0,255.0,255.0,white
231.17638036809817,218.13128834355828,221.59984662576687,1720.5865280018318,3686.8431928184104,3185.9050613249688,255.0,255.0,255.0,white
224.84792944785275,205.74777607361963,211.6674846625767,1925.4663684120085,4476.341521124781,3721.5290041029757,255.0,255.0,255.0,white
236.94969325153374,227.3302147239264,229.23926380368098,1243.060199292613,2713.6513876841586,2392.3407589675994,255.0,255.0,255.0,white
221.21104294478528,208.29923312883435,210.18205521472393,1590.0724854151613,2706.2414411302175,2645.0456902552137,244.0,226.0,232.0,white
227.57223926380368,213.7678680981595,217.3752300613497,1881.6579716714348,3800.3254859476306,3354.1405674310354,255.0,255.0,255.0,white
222.20736196319018,204.54539877300613,208.89601226993864,2192.3349151263337,4610.019411342924,4092.372315385457,255.0,252.0,255.0,white
225.33236196319018,210.0638036809816,213.26947852760736,2035.9614680399393,4423.243168355233,3864.9640377656465,255.0,255.0,255.0,white
218.13765337423314,198.5424846625767,204.36717791411044,2433.150913512782,5232.148195054435,4400.124536207606,255.0,255.0,255.0,white
```

**Test set (.arff file)**

colortest.arff

```
@relation color
@attribute avgblue numeric
@attribute avgred numeric
@attribute avggreen numeric
@attribute varblue numeric
@attribute varred numeric
@attribute vargreen numeric
@attribute midblue numeric
@attribute midred numeric
@attribute midgreen numeric

@attribute class {red,blue,green,orange,yellow,white,unknown}

@data
159.74432515337423,115.50306748466258,96.37699386503067,139.84291258137304,308.0076593022081,71.67259955210517,160.0,120.0,96.0,unknown
158.05046012269938,112.58220858895706,97.25306748466258,132.18487708889555,358.52176935524443,72.50712249237901,158.0,118.0,96.0,unknown
167.0616564417178,118.54831288343559,103.13328220858895,246.98070768565248,578.8153039020673,216.56260395101458,166.0,120.0,100.0,unknown
155.82377300613496,115.36073619631902,94.0861963190184,115.67599926135745,392.8753908690659,65.76373583876499,156.0,120.0,94.0,unknown
155.79815950920246,112.75337423312884,93.56411042944785,129.03886164328233,584.385188001051,67.69036838044629,156.0,120.0,92.0,unknown
156.59647239263805,113.12852760736196,96.96226993865031,137.06293234126608,537.19513709582,87.05808564492418,156.0,120.0,96.0,unknown
158.52269938650306,110.79340490797546,94.4707055214724,145.51927001391627,564.6419810446622,71.61739336727562,158.0,118.0,94.0,unknown
159.66303680981596,113.07070552147239,96.46395705521472,147.30271347716865,631.6238350850576,69.4157254460043,160.0,120.0,96.0,unknown
159.584509202454,111.08773006134969,97.56687116564417,163.67552690634224,555.1511991418685,81.99460800180799,160.0,118.0,98.0,unknown
```