Autonomous Rubik's Cube Solver

Autonom Rubiks Kub Lösare

LUDVIG BJÄRKEBACK

VIKING BJÖRK FRISTRÖM
Autonomous Rubik’s Cube Solver

LUDVIG BJÄRKEBACK
VIKING BJÖRK FRISTRÖM

Bachelor’s Thesis in Mechatronics
Supervisor: Nihad Subasic
Examiner: Martin Edin Grimheden
Approved: 2016-06-07

MDAB093 MMKB 2016:32
Abstract

Today the industry is constantly getting more automated. Every machine needs an algorithm to run and some even need to make decisions in how to perform their tasks in the best way. These problems can be as large as transportation systems, or as small as solving a puzzle. The 3D-puzzle known as Rubik’s cube have enticed millions of people since its release in mid 1970s. With 43 quintillion possible combinations the Rubik’s cube present a complex problem that requires both logical thinking and memorization. Recently Rubik’s Cube community has seen a rise of robots that can automatically solve a Rubik’s Cube.

The goal of this Bachelor’s Thesis is to investigate the effect of different solving algorithms on the energy consumption of a mechanical system. The Demonstrator system we will use is a Rubik’s cube solver that can solve any scrambled cube. This is done by optically scanning the cube and building a virtual image of it. Two different solving algorithms will be run in order to solve the cube and calculate the required permutations. The energy consumption in the two cases of the algorithms will be calculated and compared. From this data, conclusions regarding choice of algorithm will be made. A more complex algorithm that results in fewer moves will be more energy efficient, at least under the circumstances that the computation cost is ignored. When compared, the optimized algorithm only used 8% of the simpler version.
**Sammanfattning**

**Autonom Rubiks Kub Lösare**


Preface

We would like to thank Nihad Subasic, our supervisor for support and feedback. Didem Gürdür for initial feedback an encouragement to follow through with this project. Staffan Qvarnström for help with material. Finally we would like to thank all the assistants in the lab for aiding and counselling in the many issues we had with the demonstrator during the course of the project.

Ludvig Bjärkeback and Viking Björk Friström
Stockholm, Maj, 2016
Contents

Abstract iii

Sammanfattning v

Preface vii

Contents ix

Nomenclature xi

1 Introduction 1
   1.1 Background ......................................................... 1
   1.2 Purpose ............................................................. 1
   1.3 Scope ................................................................ 2
   1.4 Method ................................................................ 2

2 Theory 3
   2.1 Rubik's Cube ........................................................... 3
   2.2 Solving Algorithm ..................................................... 4
   2.3 Color Recognition ..................................................... 6
   2.4 Servo motor ............................................................ 6
   2.5 Stepper motor .......................................................... 7
   2.6 Slotted Optical Switch ............................................... 8

3 Demonstrator 9
   3.1 Problem formulation ............................................... 9
   3.2 Design and mechanics .............................................. 9
   3.3 Hardware ............................................................... 11
       3.3.1 Arduino .......................................................... 12
       3.3.2 Pixy camera ..................................................... 12
       3.3.3 Motors .......................................................... 12
   3.4 Software ............................................................... 14
       3.4.1 Scanning ......................................................... 15
       3.4.2 Robotic movements .......................................... 16
# Nomenclature

## Symbols

<table>
<thead>
<tr>
<th>Symbols</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\alpha$</td>
<td>Angel (Degrees)</td>
</tr>
<tr>
<td>$n$</td>
<td>Number of steps</td>
</tr>
<tr>
<td>$\Omega$</td>
<td>Resistance (Ohm)</td>
</tr>
<tr>
<td>$V$</td>
<td>Voltage (Volt)</td>
</tr>
</tbody>
</table>

## Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAD</td>
<td>Computer Aided Design</td>
</tr>
<tr>
<td>LED</td>
<td>Light Emitting Diode</td>
</tr>
<tr>
<td>HSV</td>
<td>Hue, Saturation and Value</td>
</tr>
<tr>
<td>GA</td>
<td>God’s Algorithm</td>
</tr>
<tr>
<td>LBL</td>
<td>Layer By Layer algorithm</td>
</tr>
<tr>
<td>PC</td>
<td>Personal Computer</td>
</tr>
<tr>
<td>USB</td>
<td>Universal Serial Port</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

This chapter shortly describes the background and procedures of this project which was carried out in the spring of 2016.

1.1 Background

The modern industry is getting increasingly automated. Between 1993 and 2007 the industrial robot density increased by 150% in 17 developed countries [Graetz and Michaels, 2015]. The automation increase may not be as quick as Moore’s law, but it is an indication that the automation may also have increased exponentially since the data was gathered. With this increase in robots it is important to control them in an environmental friendly way by minimizing the energy usage.

In this project the consumption issue will be tackled by comparing control algorithms. Different algorithms can solve an issue with different objectives. An algorithm may aim to solve a task as perfectly as possible, but this may be surplus. Other algorithms may use more movement but does it faster and can be better in that way.

When solving a Rubik’s cube there are many algorithms that can be used to get a possible solution. This project will compare two algorithms with different purposes and investigate the effect it has on the energy consumption of the robot.

1.2 Purpose

The Purpose of this project is to investigate the effect the algorithm choice may have on energy usage of a physical robot. Hence, the research question seeks out to answer the following.

How is energy consumption affected by optimizing the solving algorithm in an autonomous Rubik’s cube solver
1.3 Scope

Due to the limited time frame of this project we will base our projects upon the works of other. The camera used for scanning the cube has programmed algorithms for color recognition, we will only teach it the color signatures. Furthermore, the algorithms used for solving the cube is also sourced from the Internet. The mechanical construction concept is widely established in various projects [areyouliterte, 2010]. It is a simple but slow construction, it should however be ideal to investigate the research question. For simplicity, only the consumption of the robotic movements will be considered, not the calculation energy.

1.4 Method

A demonstrator in shape of an autonomous Rubik’s cube solver will be built. A scrambled cube will be put into the demonstrator controlled by an Arduino [Arduino LLC, 2005] and scanned by a Pixy camera [Charmed Labs, 2014]. The demonstrator will use two different solving algorithms to solve the same cube and the power consumption for each will be compared.
Chapter 2

Theory

In this chapter the theory on which this project is built is presented. To limit the scope of this section it is assumed that the reader has some basic knowledge in mathematics, computer programming, electrical engineering and mechanics.

2.1 Rubik’s Cube

The Rubik’s cube is the famous six-sided puzzle we are all familiar with. Made out of 26 smaller cubes, often referred to as cubies. Each cubie having one to three faces, depending on its type, with each face having one of six colors (white, yellow, blue, green, orange or red). The three types of cubies that exists are centre cubies with one face, edge cubies with two faces and corner cubies with three faces. The centre cubies cannot be moved from their position, thus their face representing the color of that side of the Rubik’s cube.

The possible positions of the cubies are as follows. 8 corner cubies which can be arranged in 8! different way, seven corners can be orientated freely but the last one depends on the rotation of the previous seven giving $3^7$ possibilities. When it comes to the edges cubies even permutations of the corners also gives even permutations of the edges, thus giving them 12!/2 arrangements. The 11 first edges can be twisted freely while the last one is dependent on the previous 11. Thereby giving them $2^{11}$ possibilities [Korf, 1997]. Which puts number of possible arrangements at.

$$8! \times 3^7 \times (12!/2) \times 2^{11} = 43,252,003,274,899,856,000$$ (2.1)

The most common way to describe a cube is what’s called Singmaster Notation. This notation considers the centre pieces of each side is fixed, that will determine the color of each side. With these faces fixed they are only able to rotate, with two directions each, producing a total of 12 moves. The Twelve possible moves that will be discussed is then U, F, D, B, L, R, Up, standing for Front, Down, Back, Left, Right Respectively, see figure 2.1. Add an apostrophe (’) to the letter will make it anti-clockwise movement. A move squared, such as $U^2$ represents doing that move...
CHAPTER 2. THEORY

twice in a row or twisting that side $180^\circ$

![Figure 2.1. picture of Singmaster notation](image)

Same set of letters can be used to describe any cube. A solved cube can be described as:

UF UR UB UL DF DR DB DL FR FL BR BL UFR URB UBL ULF DRF DFL DLB DBR.

The color each letter represents is the color of the centerpiece of that face. For example in figure 2.1, U would represent red. The UF is in this case the red and yellow edge piece. If that cubie in the scrambled cube was yellow in the top and blue to the side then the UF in the 48 letter sequence would have changed FR instead. A cube not in a solved position will have different letters in the same arrangement to describe what color we do have, instead of the desired one. This notation will be used in the programs when solving GA as it can be completed from visual inspection of the cube.

2.2 Solving Algorithm

One of the most commonly used solving algorithms for the Rubik’s cube is the so called layer by layer method (LBL). It is a quite ineffective method which often result in over 100 moves[Petruš, 2014]. However, it is popular due to its simplicity. The basic concept is that one side is picked where a cross is created by putting all the edge pieces of one side at their right position. After that, the corner pieces are put in place to create one solved layer of the cube. Secondly the corner pieces of the second layer are put in place, hence creating two solved layers. The last step is now to repeat the first one but on the last unfinished layer. As one might expect this leads to a lot of unnecessary moves but, as mentioned earlier, it is popular due to its simplicity. Due to the code for this methods size, it was omitted from the appendix.
2.2. SOLVING ALGORITHM

Taking a more academic approach, the Rubik’s cube presents a problem in what in mathematics is known as group theory. To be more specific a permutation group. In a permutation group $X$ is a given set of elements and then $G$ is a group of all the permutations or bijections of $X$ on itself [D. Dixon and Mortimmer, 1996]. In the case of the Rubik’s cube, the 48 faces of the movable cubies represent the set $X$. $G$ is the group of all possible permutations on those faces that have previously been presented $\{U, F, D, B, L, R\}$. In other words, the permutations in $G$ can only make the elements (the faces of the cubies) in $X$ change places with one another.

The first recorded attempt of setting an upper bound for the Rubik’s cube was done by David Singmaster in 1979 and was set to 277 by solving the cube using his own non-mathematical algorithm. The following year that number was lowered to 94, still not using a mathematical approach. The big breakthrough in the search for the upper bound came in 1981 by Morwen Thistlethwaite who used group theory and and computer searches to find the upper bound. The approach was to break the problem down into smaller sub problems, this by looking at the parts of the cube after a point should remain fixed. Thistlethwaite achieved this by restricting the movements of the cube by dividing $G$ in the following subgroups [Kociemba, 2010]

\[
G_0 = \{U, F, D, B, L, R\} \quad (2.2)
\]

\[
G_1 = \{U, F, D, B, L^2, R^2\} \quad (2.3)
\]

\[
G_2 = \{U, F, D, B^2, L^2, R^2\} \quad (2.4)
\]

\[
G_3 = \{U^2, F^2, D^2, B^2, L^2, R^2\} \quad (2.5)
\]

\[
G_4 = \{1\} \quad (2.6)
\]

Where $\{1\}$ is the group identity, which is no move or moves such as $U^4$ that does not change the cube. Thistlethwaite looked at the right choset space $G_1 \setminus G_0$, which means taking the cube from $G_0$ to $G_1$ and thereafter $G_2 \setminus G_1$ and so forth on each step till $G_4 \setminus G_3$. By finding upper bound for each choset an upper bound for solving the cube can be found. Using this method, it was proven the the upper bound was at maximum 29 permutations. It was later shown that these subgroups could be reduced to only 3 subgroups. These sub problems were identified by looking at the cube restricted to only the edges, only 6 edged and finally restricted by the remaining 6 edges.

Using this theory as a framework, as well as symmetry, Tomas Rokicki was able in 2010 find an upper bound. This was done using computer assisted-proof to test all possible arrangements. The 43,252,003,274,489,856,000 arrangements could be reduced to 55,882,296 sets with 19,508,428,800 positions each. With the aid of a
computer bank at Google, using 35 CPU-years Rokicki tested all the arrangements and proved that the upper bound for the Rubik’s cube is 20 moves. This number is referred to as Gods number and an algorithm solving the cube at 20 or less moves is called a Gods algorithm. Rokicki released his version and it is free for the public to use [Rokicki., 2014].

2.3 Color Recognition

For the human eye, all colors that can be perceived can be describes as a combination of three basic components: red, green and blue. The usual (RGB) notation of a color is one number between 0-255 for each of the basic colors. However, when using color recognizing software, a more useful notation is HSV. H stands for the Hue, and is the distribution of color between for example red and yellow. S is the saturation and is the amount of color present. V is the value, also known as intensity or brightness. The Value is the indication of whether a color would be dark or bright [Mbaitiga, 2015].

![Figure 2.2. One way to display the HSV space](image)

The color recognition algorithm that is used in this project is taught the identity of a color by being asked to look at it and remember its HSV values. When trying to identify and recognize this color again, it will search in the hue and saturation plane around the color. This plane can be visualized as in figure 2.2. Tolerance can be set in order to more easily find the wanted color. These parameters determine the area in which colors will be accepted as valid signatures. However since there is no tolerance in the value plane, the recognition is very light sensitive.

2.4 Servo motor

There are two main types of servo motors, continuous-rotations servos and multi-turn servos. Muti-turn servos have a closed loop position control, allowing it to turn a specified amount of degrees. Most common is that the movements is limited
2.5. STEPPER MOTOR

to a rotational range of 180 degrees. A continuous-rotation servo is on the other hand not limited to a specific rotation range. It has however the drawback of poor feedback, thus not allowing for control of position. Only speed and direction of rotation can be set. [Malášek, 2011].

2.5 Stepper motor

A stepper motor is a high precision DC-motor, without any brushes. It divides one revolution of the motor into a certain amount of steeps. The motor consists of a rotor shaped like a gear in the middle, surrounded by a stator consisting of electromagnets. The electromagnets are controlled by coils that receive impulses, activating the electromagnets. Once activated the electromagnet attracts the gear-shaped rotor, twisting it one step. The next phase can be activated to keep rotating the rotor [Mosaic Industries, 2001]. There are many types of the stepper motors, the one used in this project is a 4-phase, 6-wire unipolar.

![Wiring diagram of 4-phase, 6-wire unipolar stepper motor](Mosaic Industires, 2001)

The angle which is the angle the motor twists with each step is calculated by the following equation.

\[ \alpha = \frac{n}{360} \] (2.7)
2.6 Slotted Optical Switch

A slotted optical switch consists of a LED and a phototransistor separated by a short distance, see figure 2.4. The phototransistor registers the light transmitted by the LED, signaling by sending out an analog signal of usually 5V when the light is registered. When an object covers the area between the LED and phototransistor the light will not be registered and the switch will send out a 0V signal instead. The analog signal can be converted to a binary digital signal. Slotted optical switches have many applications within engineering such as speed and direction indication. They can often be found in printers and automated doors. [Ball, 2003]

![Figure 2.4. Sideview of a slotted optical switch](Ball, 2003)
Chapter 3

Demonstrator

This chapter will describe the process of designing and constructing the demonstrator used to answer the research question.

3.1 Problem formulation

To construct a demonstrator able to automatically solve a Rubik’s cube several engineering problems needs to be addressed. The sub-problem that have to be solved are:

- Mechanical design
- Cube scanning
- Robotic movements
- Component communication

3.2 Design and mechanics

The goal with the mechanical design is to enable the robot to perform all the permutations in $G_0$, shown in equation [2.2]. Furthermore, a Pixy camera will be used to scan the cube to create a virtual cube for the software to solve, hence must the design also support space to mount the camera in a way that one side of the cube can be clearly seen.

Firstly, a CAD model of the robot was created, see figure 3.1, consisting of one arm, one holder and one support tower. A stepper-motor is connected to the holder holding the Rubik’s cube, enabling the robot to rotate the Rubik’s cube. The arm is connected a servo motor, making it possible to move the arm back and forth. This enables to robot to grip the top two layers of the cube, as seen in figure 3.1, allowing for cube moves to be made. When the arm moves forward it pushes the cube up and over, once the cubes center of gravity falls outside its base the cube
flips over. The geometry of the holder allows the cube to then slide back into the holder, but with a new side facing up. This process can be seen in figure 3.2, where the force arrow represents the force from the arm.

![Step 1](image1)
![Step 2](image2)
![Step 3](image3)

**Figure 3.2.** Sketch of the cube flip, created using Adobe Photoshop

Lastly, when moved all the way to the back, the arm lets go of the cube, allowing it to rotate freely using the stepper motor. With this fairly simple construction, we can access all the six sides of the cube, which makes all the permutations possible. The tower in the middle serves both as support of the pushing arm as well has holder for the Pixy camera.

The first step in the building process was to 3D-print both the holder for the Rubik’s cube and the grip at the end of the arm since they have the highest demand.
on precision. Additionally, an adapter for the motor to connect to the holder was 3D-printed to match the geometry of the motor axle. The rest of the parts were cut out of plywood and put together using screws and glue onto a plywood base.

3.3 Hardware

A schematic view of all the electronics used in the project and how they are connected is presented in the figure 3.3. Followed by a more detailed explanation of each part and their specific role.

Figure 3.3. Blockchart of system, created using fritzing
3.3.1 Arduino

The main component of the project is the Arduino. Arduino is a programmable open source low cost micro controller. It will serve as link between the computer where we process the data, and the physical components such as motor and camera. It has a large variety of ports and libraries, enabling endless possibilities for usage. The Arduino gets its power supplied via a USB cable connected to the PC. The USB port also serves as communication link between the Arduino and PC when running GA.

3.3.2 Pixy camera

The Pixy camera has a built in color recognition algorithm and is developed for amateurs to easily have access to color recognition and tracking. It can very easily be taught to recognize up to seven different color signatures. When one color is recognized the Pixy will create an object for it. When connected to the Arduino the camera will automatically return a vector with information of up to 135 recognized objects. The information contained is block number, the signature number, x and y coordinates as well as width and height of the object. This information is complete when trying to track objects.

The application for the Pixy camera in this project is to scan the cube and send information about the positions of the cubies for the Arduino. The Pixy is mounted on top of the support tower, see picture 3.5 Due to the light sensitivity of the color recognition a LED light had to be mounted on the support tower to give a more even and constant illumination of the cube. The Pixy camera uses a six-pin cord that was connected directly to the Arduino.

3.3.3 Motors

Two motors are used in this project. A multi-turn servo was attached to the arm of the robot. The holder was initially attached to a stepper-motor, due the the fact that only 90 degrees turns of the cube is needed to perform the permutations. However, after a few tests it was concluded that the stepper-motor did not generate sufficient torque to rotate the cube. Solutions where tried with increased current as well as a bigger stepper-motor. This however only resulted in burning the H-bridge connected between the power supply, the Arduino and and the stepper-motor. This problem was solved by replacing the stepper-motor with a continuous-rotation servo motor, which supplied sufficient torque. Consequently, the position feedback for the holder rotations was lost with the removal of the stepper-motor. To create feedback a slotted optical switch together with four small pins was included into the construction. The pins were attach in each corner of the holder. The switch was placed so that when the holder is in position for the arm to hold or flip the cube one if the pins covers the optical switch, see picture 3.4, thus creating feedback for when the cube is in the correct position. Additionally, since a pin is placed in
3.3. HARDWARE

Figure 3.4. Slotted Optical Switch mounted on the robot with a pin covering the LED each corner, a rotation between two pins in the optical switch produces a 90 degrees twist of the cube.

Once the two motor were in place they were connected to the Arduino for control signals as well as connected to an external power supply of 6V. The slotted optical switch was connected only to the Arduino, so it also supplies it with power. To limit the current in the LED and phototransistor inside the slotted optical switch two resistances of $220\,\Omega$ each was added, see figure 3.4.
3.4 Software

Software is needed in order for the robot scan the cube, preform robot movements and also the solve the cube. The cube was solved using two different algorithms. Both algorithms were sourced from the Internet and therefore require different adaptations to work with the demonstrator. The slower algorithm uses a Layer by Layer technique [matt2uy, 2015]. In this algorithm all computation is done in the Arduino. The faster algorithm that we had more focus on uses Gods Algorithm [Kociemba, 2010]. It has a work flow of the data that involves many steps. Below is a flowchart for solving using God’s Algorithm in figure 3.6.

Figure 3.5. The final construction
3.4. SOFTWARE

Figure 3.6. Flowchart of the robot when solving using GA, created using http://code2flow.com/

3.4.1 Scanning

The scanning is done using a Pixy camera. The camera is already coded with color recognition algorithms. It gives a usable output of position and size of signatures with the recognized colors that its taught. This Arduino code from this can be found in Appendix A.

The Arduino code is then looking at hard coded positions in the cameras field of view where the cubies should be and checking what color it will find there. The code also tests that there are nine of each color, if any color has more or less a new scan will be automatically initiated. Since its very unlikely two colors swap places and thus remain to have nine of each color, we found this test to be very reliable and only failed twice throughout all trials and demos.

For the solver that used GA this data is then converted into the correct sequence in Singmaster notation and sent to the python program. The LBL algorithm needed
CHAPTER 3. DEMONSTRATOR

the information in a different notation. Due to lack of focus on LBL the code for converting from raw scan data to this notation was never completed.

3.4.2 Robotic movements

In the same Python program is the motor movements. There are five possible moves for the robot to make: twisting two ways making permutations, spinning the cube two ways and flipping it. In the case of the GA solver a string is received form the Python program with movements in sequence to solve the cube at hand. The string is sent to the Arduino which has robotic movement functions sending signals to the motors who executes the permutations. This code can be found in Appendix A. This was not successfully implemented with LBL algorithm. If this would have been implemented it would look slightly different as it would perform the robot movement parallel to the solving algorithm.

3.4.3 Solving algorithms

In order to answer the research questions two different solving were used.

Gods Algorithm

To solve the cube with a GA a software developed by Herbert Kociemba called Cube Explorer was used[1]. It accepts two versions of Singmaster notation in specific formats, which can be produced by the Arduino.

In order to run the solver automatically we used Python's subprocess. The communication between the Arduino and Python is done via the Serial Port. The solver program will be run as a subprocess in Python. Subprocess allows Python to run other terminal window processes and extract the printed result. With some processing the exact moves in terms of R, L’, D, etc. can be extracted.

Python script is also used to determine what motor movements is required to perform the permutations given by the solver. This is a complicated issue, because the orientation of the cube will change whenever a move is done. Depending on the orientation of the cube, different motor activities are required in order to execute an 'R' movement for example. Only two strings will be sent using the serial port, one sting from the Arduino describing the cube, and one string from the Python with all motor movements in sequence. This program can be found in Appendix B.

Layer by layer

The second algorithm used solves the cube layer by layer. After each permutation the LBL checks the positions of the cubies and then preforms the next set of permutations depending on the current positions. For each sub problem in the layer by layer method, described in the theory chapter, there are only a certain number of valid permutations that have to be repeated. Depending on the cubies positions,
these different subproblems are represented by different states in LBL code. Since
the LBL code only runs through different states when solving it does not require
the same CPU power as GA and can be run entirely on the Arduino.
Chapter 4

Results

In this chapter the results from tests on the demonstrator are presented

4.1 Data collection

The results were obtained by letting the two algorithms solve the same scrambled cube and calculate the moves required to solve it. To get more time efficient results we did not let the demonstrator perform the movement, but rather only produce the number of moves. The lack of reliability of the demonstrator further supports this decision. The number of robot moves that are presented by the programs serves as the data used for the comparison.

4.2 Numerical result

<table>
<thead>
<tr>
<th>Test #</th>
<th>Spins LBL</th>
<th>Flips LBL</th>
<th>Total LBL</th>
<th>Spins GA</th>
<th>Flips GA</th>
<th>Total GA</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>533</td>
<td>235</td>
<td>768</td>
<td>35</td>
<td>23</td>
<td>58</td>
</tr>
<tr>
<td>2</td>
<td>487</td>
<td>217</td>
<td>704</td>
<td>41</td>
<td>20</td>
<td>61</td>
</tr>
<tr>
<td>3</td>
<td>582</td>
<td>260</td>
<td>843</td>
<td>34</td>
<td>21</td>
<td>55</td>
</tr>
<tr>
<td>4</td>
<td>486</td>
<td>209</td>
<td>695</td>
<td>39</td>
<td>18</td>
<td>57</td>
</tr>
<tr>
<td>5</td>
<td>674</td>
<td>300</td>
<td>974</td>
<td>48</td>
<td>24</td>
<td>72</td>
</tr>
<tr>
<td>6</td>
<td>525</td>
<td>226</td>
<td>751</td>
<td>44</td>
<td>22</td>
<td>66</td>
</tr>
<tr>
<td>7</td>
<td>560</td>
<td>256</td>
<td>816</td>
<td>39</td>
<td>22</td>
<td>61</td>
</tr>
<tr>
<td>8</td>
<td>509</td>
<td>228</td>
<td>737</td>
<td>30</td>
<td>22</td>
<td>52</td>
</tr>
<tr>
<td>9</td>
<td>572</td>
<td>259</td>
<td>831</td>
<td>35</td>
<td>22</td>
<td>57</td>
</tr>
<tr>
<td>10</td>
<td>416</td>
<td>183</td>
<td>599</td>
<td>41</td>
<td>24</td>
<td>65</td>
</tr>
</tbody>
</table>
4.3 Energy consumption

Energy estimates for all the moves have been done by measuring the voltage supplied to the servos and the electric current while performing while solving a cube. The power was calculated and in turn the energy used.

By measuring the energy for a single move and multiplying it by the number of moves we go a more reliable way to calculate the energy used by the robot. It needs to be kept in mind that the computational power of the Arduino and computer is not taken into count. Based on the mean value of the energy used GA only 8% of the energy that LBL used when solving the same Rubik’s cube.

<table>
<thead>
<tr>
<th>Table 4.2. Statistics over energy consumption</th>
</tr>
</thead>
<tbody>
<tr>
<td>Energy LBL [J]</td>
</tr>
<tr>
<td>Mean</td>
</tr>
<tr>
<td>Median</td>
</tr>
<tr>
<td>Standard deviation</td>
</tr>
<tr>
<td>Mean 95% confidence interval</td>
</tr>
</tbody>
</table>
Chapter 5

Discussion and conclusions

In this chapter result and issues will be discussed, and conclusions drawn from the discussion will be presented.

5.1 Discussion

For the purpose of making a reliable demonstrator a few improvements could be made. A stepper motor was attempted for use for the spinning holder, to succeed with this would have been advantageous as it would have improved the precision in its rotational positioning. We used the servo and optical sensor, and it had the advantage of confirmation that the cube was in the desired position and torque required, but with the downside of poor precision in its turns. This compromised the reliability of the demonstrator and forced us to simulate the tests, possibly affecting the result.

As expected our result lead to that the more complex algorithm would solve the cube in a more environmentally friendly way. Unfortunately, the secondary algorithm was very poor and did not gain speed for its sacrifice in complexity. This disabled our ability to draw solid conclusions with time as an aspect. All calculation time and energy has been ignored. Both algorithms we used proved to be slow, partially because of slow hardware. When looking at a singular process such as solving a cube, calculation time will not be an issue. However, when there is a series of activities that relay on each other, such as continuous production, one machine with an efficient but slow algorithm could become a bottleneck and cause inefficiency for all other activities. This means that in certain applications a quicker algorithm with more waste may be more beneficial overall.
5.2 Conclusion

When choosing algorithm for any application the context need to be considered. For isolated tasks such as solving a Rubik’s cube an algorithm that uses the least physical movement is the best. For larger systems time needs to be considered factor. However, a number of factors limits the reliability of our result. Mainly the fact that test data came from computer simulations of the solving process, without the robot actually preforming the moves, leaving some variables unknown. Finally, the low number of test gives a quite large confidence interval for the values.

With this in mind the results are still of the magnitude to support the conclusion that energy usage is highly affected by algorithm choice.
Chapter 6

Recommendations and future work

The project is sufficient to draw initial conclusions regarding the energy consumption dependent of solver algorithm, but there are issues that could be improved and this will discussed in this chapter.

6.1 Demonstrator

The demonstrator could be improved in a few regards. Its construction should be improved in order to make moves more reliable, this will allow for more autonomous tests to be done. The need for human assistance in replacing cubes that was pushed off resulted in that this project relying mainly on simulated solutions. Improvements that could be done is to get a stepper motor to work, in order to get more precise spins. Angles of the arm should be calculated and changed so the arm could push further and stronger to flip the cube more reliably.

In order to solve the algorithms quicker a faster computer with better hardware could be used. This will help to allow more trials to be done and also get come consistent results. As mentioned in the Demonstrator chapter, the LBL algorithm was never successfully implemented to work with the physical demonstrator. If this was completed more fair tests could be conducted in order to draw stronger conclusions.

What should also be considered is that GA was developed by four professional mathematicians, while the LBL was developed by an amateur. Perhaps for a deeper analysis the LBL should be changed for a faster algorithm that still has emphasis on simple computing which results in more but quicker moves.

6.2 Improving data

To get a more statistically reliable results more test date should be gathered. Since the task of gathering data is quite time consuming a greater deal of time for the project should be dedicated to this task.
Since the research is about the energy consumption this should be measured more carefully. These improvements should include taking accounts for the computing energy used, both in the Arduino and in the computer in the case of the GA being used. More accurate energy measurements should also be made, this could include measure the total energy used by the robot over the entire solving process, as opposed to only measure the move and multiply number of moves. A complete measurement will also catch the energy used during scanning.
Bibliography


Appendix A

Arduino code

```c
#include <SPI.h>
#include <Pixy.h>

#include <Servo.h>
Servo holder;
Servo arm;

Pixy pixy; // This is the main Pixy object

int utbiten[48];
char utchar[48];
int sida[9];
int sideup=0;
int scandone=0;
int scanconfirm=0;
int i = 0;
int q=0;
int nyckel[6];
int readdone=0;

const int ForkSensor = 2;
const int servoholder = 10;
const int servoarm= 4;
const int ArmStartPos = 48;
const int ArmBackPos = 71;
const int ArmFrontPos = 18;
const int HolderStop = 93;
const int HolderRight = 88;
const int HolderLeft = 96;

int val=0;

void setup() {
  Serial.begin(9600);
```
pixy.init();

arm.attach(servoarm); // attaches the servo that moves the arm on pin 10
to the servo object
arm.write(ArmBackPos);

pinMode(ForkSensor, INPUT);
attachInterrupt(digitalPinToInterrupt(ForkSensor), Stop, FALLING);

pinMode(5, OUTPUT);
digitalWrite(5, HIGH);

}

void loop() {

i++;
if (i%60==0){ //we read every 60 cycle, if we do every the camera cant
coope
if (readdone==0){
Readcam();}

if (readdone==1){
int movies;

if (movies==108){SpinnLeft();}
else if (movies==114){SpinnRight();}
else if (movies==102){Flipp();} //this is ascii for the moves
else if (movies==104){TwistRight();}
else if (movies==118){TwistLeft();}
}

}

}//viod loop closes

//----below are all robot movements, controlled by servos.----
void Stop(){
if (val == 1 ) {
holder.write(HolderStop);
holder.detach();
val = 0;
void Flipp() {
    holder.write(HolderStop);
    holder.detach();
    arm.write(ArmBackPos);
    delay(500);
    arm.write(ArmFrontPos);
    delay(740);
    arm.write(ArmBackPos);
    delay(2500);
}
void TwistLeft() {
    arm.write(ArmStartPos);
    delay(400);
    holder.attach(servoholder);
    holder.write(HolderRight);
    delay(500);
    val = 1;
    delay(750);
}
void TwistRight() {
    arm.write(ArmStartPos);
    delay(400);
    holder.attach(servoholder);
    holder.write(HolderLeft+1);
    delay(500);
    val = 1;
    delay(750);
}
void SpinnLeft(){
    arm.write(ArmBackPos);
    delay(400);
    holder.attach(servoholder);
    holder.write(HolderRight);
    delay(500);
    val = 1;
    delay(750);
}
void SpinnRight(){
    arm.write(ArmBackPos);
delay(400);
holder.attach(servoholder);
holder.write(HolderLeft);
delay(500);
val = 1;
delay(750);

void Readcan(){ //and this is the function that reads interprets the 
camera. loops may times

uint16_t blocks;
int hojd;
int bredd;
int tope;
int lefte;
int righte;
int bottome;

int key[6];
int t1=0; //top row
int t2=45;
int m1=65; //middle row
int m2=110;
int b1=135; //bottom row
int b2=200; //these are all positions of columns and rows
int l1=45;
int l2=95; //left column
int c1=115;
int c2=155; //center column
int r1=185;
int r2=225; //right column
int okay;

int column[6]={l1, l2, c1, c2 ,r1, r2};
int row[6]={t1, t2, m1, m2 ,b1 ,b2};
int Problem=0;

int sidedone=0;
blocks = pixy.getBlocks();
q++;
for(int j=0; j<blocks;j++){

-sidedone=1;

lefte=pixy.blocks[j].x-pixy.blocks[j].width/2; //position of
left edge of box
righte=pixy.blocks[j].x+pixy.blocks[j].width/2; //position of
right edge of box
bottome=pixy.blocks[j].y+pixy.blocks[j].height/2; //position
of top edge
tope=pixy.blocks[j].y-pixy.blocks[j].height/2; //position of
bottom edge

for(int cubie=0; cubie<9; cubie++){ //go though all cubies
int ve1;
int ve2;
if (cubie<3){
ve1=row[0];
ve2=row[1];}
elserf (cubie <6){
ve1=row[2];
ve2=row[3];
else{
ve1=row[4];
ve2=row[5];}

int he1=column[(cubie%3)*2];
int he2=column[(cubie%3)*2+1];

for(int h=he1; h<he2; h+=10){ //horizontal positons of
the first box/cubie

for(int v=ve1; v<ve2; v=v+10){ //vertical positons of
the first box

if(lefte<h&&righte>h&&tope<v&&bottome>v){

if
(sida[cubie]==pixy.blocks[j].signature||sida[cubie]==0){
sida[cubie]=pixy.blocks[j].signature;
else{
Problem=1;
APPENDIX A. ARDUINO CODE

```cpp
break;
}
}

if(Problem==1){
    sidedone=0;
    break; //break horizontal position go through
}
if(Problem==1){
    sidedone=0;
    break; // is going to break the going-through-all-cubies
}
if(Problem==1){
    break; // going to break for-all-blocks
}

// MOVEMENT SEQUENCE, (TOP), FLIP(FONT), FLIP(UNDER), FLIP(BACK), TWIST RIGHT, FLIP(LEFT), FLIP, FLIP(RIGHT)
if (sidedone==1){
    switch(sideup){
        case 0: // 0 is the side we start as up.
            utbiten[30]=sida[0];
            utbiten[4]=sida[1];
            utbiten[27]=sida[2];
            utbiten[6]=sida[3];
            nyckel[0]=sida[4]; // the key remembers which color(number) is
            where
            utbiten[2]=sida[5];
            utbiten[33]=sida[6];
            utbiten[0]=sida[7];
            utbiten[24]=sida[8];
            Flipp();
            sideup=1;
            break;
        case 1:// the side we consider to be the front
            utbiten[35]=sida[0];
            utbiten[1]=sida[1];
            utbiten[28]=sida[2];
            utbiten[18]=sida[3];
            nyckel[1]=sida[4]; // ngot om att detta r sida FRONT
            utbiten[16]=sida[5];
            utbiten[40]=sida[6];
            utbiten[9]=sida[7];
            utbiten[38]=sida[8];
```
//flip and switch to case 2:
Flipp();
sideup=2;
break;

case 2://what we consider to be DOWN
utbiten[39]=sida[0];
utbiten[8]=sida[1];
utbiten[36]=sida[2];
utbiten[14]=sida[3];
nyckel[2]=sida[4];//something about this beeing the DOWN side :P
utbiten[10]=sida[5];
utbiten[42]=sida[6];
utbiten[12]=sida[7];
utbiten[46]=sida[8];

//flip and switch to case 3:
Flipp();
sideup=3;
break;

case 3://now at the side we conside to be back
utbiten[44]=sida[0];
utbiten[13]=sida[1];
utbiten[46]=sida[2];
utbiten[22]=sida[3];
nyckel[3]=sida[4];//someting about being the back;
utbiten[20]=sida[5];
utbiten[31]=sida[6];
utbiten[5]=sida[7];
utbiten[29]=sida[8];

SpinnRight();
Flipp();
sideup=4;
break;

case 4: //now on the left side scanned; importnat that UPP is on
        the right side.
utbiten[43]=sida[0];
utbiten[23]=sida[1];
utbiten[32]=sida[2];
utbiten[15]=sida[3];
nyckel[4]=sida[4];//something that indicates that we are on teh
        left side
utbiten[7]=sida[5];
utbiten[41]=sida[6];
utbiten[19]=sida[7];
APPENDIX A. ARDUINO CODE

```cpp
utbiten[34]=sida[8];
//flip twice, and well reach the right side, UPP still to the
left, change to case 5
sideup=5;
Flipp();
Flipp();
bright;
case 5: //pretty much that opp there; side right
utbiten[37]=sida[0];
utbiten[17]=sida[1];
utbiten[26]=sida[2];
nyckel[5]=sida[4]; //something that this signature is RIGHT
utbiten[3]=sida[5];
utbiten[47]=sida[6];
utbiten[21]=sida[7];
utbiten[28]=sida[8];

scandone=1;
//python will be hardcoded to know that "right" is up, and
"back" is front at begining..
break;
} //switch closes
}
//if loop for one side is done CLOSING

} //for loop that goes though all blocks; closing

} //if loop that check if we have any bloxes: CLOSING

if (scandone==1){
int plus=0;
int ganger=1;

for (int nyckelplats=0; nyckelplats<6; nyckelplats++){
    plus=plus+nyckel[nyckelplats];

    if (nyckel[nyckelplats]!=0){ganger=ganger*nyckel[nyckelplats];}
    //kan sker si slint, men detta r lttast och minst kod
}
if (ganger==120&&plus==15){
int rekna[]={0, 0, 0, 0, 0, 0}; //FR Att kolla s att vi har rtt antal
    av alla frger
```
for(int pos0=0; pos0<48;pos0++){ // counting the number of each color.
    if (utbiten[pos0]==nyckel[0]){
        rekna[0]=rekna[0]+1;
        utchar[pos0]='U';
    } else if(utbiten[pos0]==nyckel[1]){ 
        utchar[pos0]='F';
    } else if(utbiten[pos0]==nyckel[2]){  
        utchar[pos0]='D';
    } else if(utbiten[pos0]==nyckel[3]){  
        utchar[pos0]='B';
    } else if(utbiten[pos0]==nyckel[4]){  
        utchar[pos0]='L';
    } else if(utbiten[pos0]==nyckel[5]){  
        utchar[pos0]='R';
    }
}

int antalratt=0;
for (int pos1=0;pos1<6;pos1++){
    if (rekna[pos1]>8){
        scandone=0;
        sideup=0;
        memset(nyckel, 0, sizeof(nyckel));
        memset(utbiten, 0, sizeof(utbiten));
    } else{antalratt++; 
    if (antalratt==6){
        Serial.print(utchar); //this is where we sent to the python the result
        readdone=1;
    }
}
}
} //if loop cheking the sum and product of key is right
else{ //this is if wefail to have the right key

scandone=0;
sideup=0;
memset(nyckel, 0, sizeof(nyckel)); // reset the stuff so do again
memset(utbiten, 0, sizeof(utbiten));
}
} // closing "if scan is done"
Appendix B

Python code

```python
import time
import serial
from subprocess import PIPE, STDOUT
import subprocess

ser = serial.Serial("com4",9600,timeout=1)  # Establish the connection on a specific port
counter = 0

print("waiting for cube")
time.sleep(2)

while True:  #waits to recieve cube info
    ardo=ser.readline()
    ut=ardo.decode("utf-8")
    lol=list(ut)
    if len(ut)>20:
        break

    #formats the info the way we want it. spaces and all
    for j in range(1,8):
        lol.insert(48-j*3,' ',)

    for q in range(0,12):
        lol.insert(24-q*2,' ',)

    if lol[-1]!="L" and lol[-1]!="D" and lol[-1]!="F" and lol[-1]!="U" and lol[-1]!="R" and lol[-1]!="B":
        lol.pop(-1)
    conected=''.join(map(str,lol))
    print(conected)
```

37
def Solver(idata):
    cubeExplorerPath = 'C:\Users\Filip\Desktop\optiqtm.exe'  # all this
    # is to start the solver
    p = subprocess.Popen([cubeExplorerPath], stdout=PIPE, stdin=PIPE)

    out, err = p.communicate(input=idata.encode())

    p.kill()

    count = 0
    deezashit = []
    for acii in out:
        char = ''.join(chr(acii))

        if char == 'd':  # so that we dont read the depths
            count = 0  # all below is so that we extract the right piece of info
            # on the cube
            if count == 5:
                if char == '(':
                    break
                deezashit.append(char)

        elif char == 't':
            count += 1

        elif char == 'e':
            count += 1

        elif char == 's':
            count += 1

        else:
            count = 0

    deezashit.pop()
    deezashit.pop()
    deezashit.pop(0)
    deezashit.pop(0)
    while " " in deezashit:
        deezashit.remove(" ")
    return(deezashit)

print("the size of conected is", len(conected))
res = Solver(conected)
print("vi kommer singmaster som ska gras: ", res)
fjomp=0
for j in res:
    if j=='''':
        fjomp+=1

for i in range(0,len(res)-fjomp+1): #fixar tillres, connecting the fjoms
    if res[-i]=='''
        res[-i-1]=res[-i-1]+'''
        res.pop(-i)

#this works like a dice (number up). 1 is top at start. 2 facing u, 4 to the left
#in otder to determine orientation. these work like states.
moves=[] #vector that will contain all moves

def spinleft(top,front): #left is clockcwise. determines the ointaion
    #after move is done. appends Moves
    global moves
    moves.append('l')
    # ------..-------- i

    if top==1:
        if front==2:
            front=3
        elif front==3:
            front=5
        elif front==4:
            front=2
        elif front==5:
            front=4
    elif top==6:
        if front==3:
            front=2
        elif front==5:
            front=3
        elif front==2:
            front=4
        elif front==4:
            front=5
    elif top==2:
        if front==1:
            front=4
        elif front==4:
            front=4
```python
front=6
elif front==6:
    front=3
elif front==3:
    front=1
elif top==5:
    if front==4:
        front=1
    elif front==6:
        front=4
    elif front==3:
        front=6
    elif front==1:
        front=3
elif top==3:
    if front==1:
        front=2
    elif front==2:
        front=6
    elif front==6:
        front=5
    elif front==5:
        front=1
elif top==4:
    if front==2:
        front=1
    elif front==6:
        front=2
    elif front==5:
        front=6
    elif front==1:
        front=5

return top,front
def twistleft():
global moves
moves.append('v')  # append the vector with all moves
pass
def spinright(top,front):  # right is anti-clockwise "have the cube in front
    # of you, and flick with right thumb."
global moves  # determines the orientation of cubes, and append with
    # the move
moves.append('r')

#---------..---------..---------..---------..---------..---------..---------

if top==6:
    if front==2:
        front=3
```
elif front==3:
    front=5
elif front==4:
    front=2
elif front==5:
    front=4
elif top==1:
    if front==3:
        front=2
    elif front==5:
        front=3
    elif front==2:
        front=4
    elif front==4:
        front=5
elif top==5:
    if front==1:
        front=4
    elif front==4:
        front=6
    elif front==6:
        front=3
    elif front==3:
        front=1
elif top==2:
    if front==4:
        front=1
    elif front==6:
        front=4
    elif front==3:
        front=6
    elif front==1:
        front=3
elif top==4:
    if front==1:
        front=2
    elif front==2:
        front=6
    elif front==6:
        front=5
    elif front==5:
        front=1
elif top==3:
    if front==2:
        front=1
    elif front==6:
        front=2
    elif front==5:
        front=6
    front=6
elif front==1:
    front=5

return top,front

def twistright(): #the lower platform twists right
    global moves
    moves.append('h')
    pass

def flip(top,front):
    global moves
    moves.append('f')

    oldtop=top
    top=front
    front=7-oldtop

    return top,front

def makemove(top,front,moves):
    if top+front==7 or front>6 or top>6:
        print("Ngot r kaoz med sidonumrerna")
        return

    if moves=='R' or moves=='R'": #to execute this we need side 3 to face down.
        if top==4:
            pass
        else:
            if top==3:
                top, front=flip(top,front)
            elif top==1: #kanske hlla koll p vilken sida som kollar frammt
                if front==5: #is front is 5, its quicker to move left
                    top,front=spinleft(top,front)
                else:
                    while front!=4: #move right until we have 4 facing, when we flip!
                        top,front=spinright(top,front)

            elif top==5:
                if front==6:
                    top,front=spinleft(top,front)
                else:
                    while front!=4:
elif top==6:
    if front==2:
        top,front=spinleft(top,front)
    else:
        while front!=4:
            top,front=spinright(top,front)

eelif top==2:
    if front==1:
        top,front=spinleft(top,front)
    else:
        while front!=4:
            top,front=spinright(top,front)

top,front=flip(top,front) #does the flipping twist

if moves=="R":
    twistright() #perform the actual twist.
else:
    twistleft()

eelif moves=='L' or moves=="L'": #nu vill vi flippa ner fyra.
    if top==3:
        pass

eelse:
    if top==4:
        top,front=flip(top,front)
    if top==1:
        if front==2:
            top,front=spinleft(top,front)
        else:
            while front!=3:
                top,front=spinright(top,front)
    elif top==2:
        if front==6:
            top,front=spinleft(top,front)
        else:
            while front!=3:
                top,front=spinright(top,front)
    elif top==6:
        if front==5:
            top,front=spinleft(top,front)
        else:
            while front!=3:
APPENDIX B. PYTHON CODE

top, front = spinright(top, front)

elif top == 5:
    if front == 1:
        top, front = spinleft(top, front)
    else:
        while front != 3:
            top, front = spinright(top, front)

    top, front = flip(top, front)

if moves == "L":  # perform the actual twist, dependant of weather we do
    twistright()
else:
    twistleft()

elif moves == 'U' or moves == "U'":  # nu vill vi flippa ner ett.
    if top == 6:
        pass
    else:
        if top == 1:
            top, front = flip(top, front)
        if top == 2:
            if front == 4:
                top, front = spinleft(top, front)
            else:
                while front != 6:
                    top, front = spinright(top, front)

        elif top == 4:
            if front == 5:
                top, front = spinleft(top, front)
            else:
                while front != 6:
                    top, front = spinright(top, front)

        elif top == 5:
            if front == 3:
                top, front = spinleft(top, front)
            else:
                while front != 6:
                    top, front = spinright(top, front)

        elif top == 3:
            if front == 2:
                top, front = spinleft(top, front)
            else:
                while front != 6:
                    top, front = spinright(top, front)

44
top, front = flip(top, front)

if moves == "U": # perform the actual twist, dependant of weather we do L or anti
    twistright()
else:
    twistleft()

elif moves == 'D' or moves == "D'": # nu vill vi flippa ner sex.
    if top == 1:
        pass
    else:
        if top == 6:
            top, front = flip(top, front)
        if top == 2:
            if front == 3:
                top, front = spinleft(top, front)
            else:
                while front != 1:
                    top, front = spinright(top, front)
        elif top == 3:
            if front == 5:
                top, front = spinleft(top, front)
            else:
                while front != 1:
                    top, front = spinright(top, front)
        elif top == 5:
            if front == 4:
                top, front = spinleft(top, front)
            else:
                while front != 1:
                    top, front = spinright(top, front)
        elif top == 4:
            if front == 2:
                top, front = spinleft(top, front)
            else:
                while front != 1:
                    top, front = spinright(top, front)

        top, front = flip(top, front)

if moves == "D": # perform the actual twist, dependant of weather we do L or anti
    twistright()
else:
    twistleft()
elif moves=='F' or moves=="F'": #nu vill vi flippa ner tv.
    if top==5:
        pass

    else:
        if top==2:
            top,front=flip()
        if top==1:
            if front==3:
                top,front=spinleft(top,front)
            else:
                while front!=5:
                    top,front=spinright(top,front)
        elif top==3:
            if front==6:
                top,front=spinleft(top,front)
            else:
                while front!=5:
                    top,front=spinright(top,front)
        elif top==6:
            if front==4:
                top,front=spinleft(top,front)
            else:
                while front!=5:
                    top,front=spinright(top,front)
        elif top==4:
            if front==1:
                top,front=spinleft(top,front)
            else:
                while front!=5:
                    top,front=spinright(top,front)

    top,front=flip(top,front)

elif moves=="F": #perform the actual twist, dependant of weather we do
     L or anti
    twistright()
else:
    twistleft()

elif moves=='B' or moves=="B'": #nu vill vi flippa ner fem.
    if top==2:
        pass

    else:
        if top==5:
            top, front=flip(top,front)
        if top==1:
if front==4:
    top,front=spinleft(top,front)
else:
    while front!=2:
        top,front=spinright(top,front)
elif top==4:
    if front==6:
        top,front=spinleft(top,front)
    else:
        while front!=2:
            top,front=spinright(top,front)
elif top==6:
    if front==3:
        top,front=spinleft(top,front)
    else:
        while front!=2:
            top,front=spinright(top,front)
elif top==3:
    if front==1:
        top,front=spinleft(top,front)
    else:
        while front!=2:
            top,front=spinright(top,front)

top,front=flip(top,front)

if moves=="B": #perform the actual twist, dependant of weather we do
    twistright()
else:
    twistleft()

return(top,front)

for sing in res: #loops though all the moves the solver want to.
    top,front=makemove(top,front,sing)

flipp=0
spinn=0

print(moves)
for i in moves:
    if i=='f': #all this is a conter to see wat hasz been done
        flipp+=1
    else:
        spinn+=1

print(len(moves))
APPENDIX B. PYTHON CODE

```python
print("antal flips r: ",flipp)
print("antal snurrar r: ",spinn)

for i in moves:
    ut=i.encode()  # send the moves ID to the arudino.
    ser.write(bytes(i, 'UTF-8'))
```

48