Rubik's Cube Solution Method Using Real-Time Tracking Cube Approach

1st Teguh Nurhadi Suharsono
Faculty of Engineering
Universitas Sangga Buana
Bandung, Indonesia
teguh.nurhadi@usbypkp.ac.id

2nd Abdul Rozak
Faculty of Information Technology & Digital
Institut Digital Ekonomi LPKIA
Bandung, Indonesia
abdulrzk44@gmail.com

3rd Rina Mardiati
Department of Electrical Engineering
UIN Sunan Gubung Djati Bandung
Bandung, Indonesia
r_mardiati@uinsgd.ac.id

***Abstract*— Rubik's solution method is a way to solve Rubik's by using predetermined rotation steps, now there are many methods of Rubik's solution that have been created and among them there is a trend to create a Rubik's solution method with the most optimal steps starting in 1981 Morwen Thistlewaite created a Rubik's solution algorithm which has fewer steps than the solution method at that time, with 52 steps enough to solve all Rubik's randomness, this trend continued until the last recorded Kociemba algorithm which is considered the most optimal Rubik's solution algorithm. at this time with a maximum number of completion steps of 20 steps discovered by Tomas Rokicki, Herbert Kociemba, Morley Davidson, and John Dethridge in 2010. To make the optimal Rubik's solution method, a complex algorithm is needed so that it requires extra time and process in doing the Rubik's solution. Therefore, in this study, we analyze the method for the optimal Rubik's solution with the simplest possible process using a real-time Rubik's Cube tracking approach, and the results obtained with a process that is not too complex but less than optimal in terms of the number of steps produced.**

***Keywords*— rubik's solution, tracking cube, rubik's algorithm**

I. INTRODUCTION

Unstoppable technology development problems help specific problems that technology can solve[1], and Rubik's Cube is no exception. Rubik's Cube game is a puzzle game that is very popular in many countries with the distribution of physical Rubik's products in more than 30 countries in 2011, Rubik's Cube was invented by Erno Rubik a professor from Budapest in Hungary. After Erno Rubik made his first Rubiks cube he needed time more than a month to complete[2]. At this time the Rubik's solution method which is considered the simplest is the Layer-by-layer method which can be found on the official Rubik's Company website, after the Rubik's 3x3 cube was patented and published methods began to emerge a new method of solving Rubik's puzzles. Among these methods of solving Rubik's there are methods that produce fewer steps of completion than other methods and after that it became popular to find the most optimal Rubik's solution method in solving Rubik's puzzles 3x3 and s appears a term known as God's Algorithm is an algorithm that provides an optimal solution in the sense that there is no shorter solution is called God's Algorithm [3] and the maximum number of steps generated from God's Algorithm is called God's Number.

The term God's Number was coined because the thought of a person who could find the shortest sequence of moves to solve any kind of Rubik's Random must be thousands of times stronger than an ordinary human being, able to test millions of different combinations in an instant. the eye, something that mathematicians believe only God can possess, the difficulty faced in finding the most optimal solution in Rubik's 3x3 is that the number of random numbers that can be generated by Rubik's 3x3 is very large, totaling 43,252,003,274,489,856,000 Rubik's random combinations [4] that make it very difficult to generate and process even for very fast computers though.

The history of the God's Number search process dates back to 1981 when a man named Morwen Thistlewaite proved using a complex algorithm he devised himself that 52 moves was enough to solve every random of 43 quintillion different randomizations[4] and continues to emerge. a more optimal method until finally found the most optimal Rubik's 3x3 solution method to date which was discovered by Tomas Rokicki, Herbert Kociemba, Morley Davidson, and John Dethridge in 2010 and named Kociemba Algorithm with a number of steps 20 and makes God's Number is 20 to date [3]

From the cube puzzle problem above, the search for a Rubik's solution that has been going on for decades underlies the making of this research for the purpose of contributing to the search for a Rubik's solution method that focuses on optimizing the number of Rubik's solving steps and the solution process as simple as possible and supported by information technology exists at the moment.

In this research, the method approach that will be used is Tracking the Rubik's Solution Path which is carried out on the simulation results of periodic randomization of Rubik's as the main concept of the method to be implemented, this approach is taken because it is considered a simple method and will produce quite optimal completion steps according to the researcher. and with this research, it is hoped that it can add new alternatives for Rubik's game developers in implementing the Rubik's Cube solution in the Rubik's game that is made.

II. LITERATURE REVIEW

Rubik's Cube is a game in the form of a mechanical puzzle invented in 1974 by the Hungarian sculptor and professor of architecture, Ernő Rubik. Rubik's Cube consists of 9 sides that can be rotated on each side without breaking and each side of this cube has nine faces consisting of six different colors.

When solved/solved, each side of this cube will have one color the same on each side and a different color from the other sides[5].

Erno Rubik initially developed moving artworks for his students in the field of architecture with the initial aim of

helping his students understand three-dimensional problems, but the cube prototype created by Erno did things the world had never seen before, namely a cube prototype that had puzzles. puzzle with an iconic look. In 1975 Erno Rubik patented his cube prototype as a puzzle game[2].

The layer by layer algorithm is a Rubiks cube algorithm which is considered the simplest and easiest to understand for beginners which is recommended by the Rubiks Company as a start to start solving 3x3 Rubiks.

This kociemba algorithm was created by Herbert Kociemba to improve the existing Thistlethwaite algorithm by reducing the number of intermediate groups to two:

a. G0 = (U, D, L, R, F, B)
b. G1 = (U, D, L2, R2, F2, B2)
c. G2 = (1)

As with Thistlethwaite's algorithm, he will find the correct coset space of G1\ G0 to group the cube into group G1. Next he will find the optimal solution for Group G1. Searches in G0 and G1 are both performed using a method equivalent to Iterative Deepening A* (IDA*) Searches in G1 require a maximum of 12 moves and searches in G1 a maximum of 18 moves. Michael Reid demonstrated in 1995 that creating a suboptimal solution that takes the cube to group G1 and looking for the short solution in G1, will usually result in a shorter overall solution[5].

God's algorithm is an idea that stems from discussions about how to solve Rubik's Cube puzzles [5], but can also be applied to combinatorial puzzles and other math games[6]. It refers to any algorithm that produces a solution that has as few motions as possible, the idea being that only an omniscient being will know the optimal step of a given configuration[7].

The term God's Number was coined because the thought of a person who could find the shortest sequence of moves to solve any kind of Rubik's Random must be thousands of times stronger than an ordinary human being, able to test millions of different combinations in an instant. eyes, something that mathematicians believe can only be possessed by God, the difficulty faced in finding the most optimal solution in Rubik's 3x3 is because the number of random combinations that can be generated by Rubik's 3x3 is 43,252,003,274,489,856,000 random combinations of Rubik's[6].

There are several studies related to this rubik's. Research that performs the Rubik's Cube with the Basic Beginners algorithm and connects the Fridrich algorithm with 80 steps[8]. Research has shown that genetic algorithms are an effective method for determining the sequence of steps required to solve a Rubik's cube by randomizing the rubik's order. The selection of chromosomes that are higher than the initial-stage Rubik's solution factor leads to a better solution[9]. The study reduced the step results from 227 movements to complete the Rubik's Cube to 107 steps[10]. Other studies using the k-means method, Simple Linear Iterative Clustering and the Kociemba algorithm require a lot of time to optimize the method[11]. There is also a study that uses 3 algorithms, namely Thistlehwaite, Kociemba and Rokicki which states that the algorithm is efficient but difficult for humans to understand[12]. Another study states that the Kociemba algorithm explains the efficiency in determining each step in the Rubik's solution[13]. Research that states it is less effective if used to solve Rubik's problems because Rubik's Rubik can be solved faster than human ability and Brute Force has an ability that is less fast when compared to the IDA* algorithm because it takes a long time to get a solution[14].

III. System Purposed

A. Rubik's Prototype Flow

In this study, the Rubik's solution function which is the main focus of research will be combined with other functions so that it will produce a complete Rubik's game prototype, which includes scramble, reset, and Rubik's solving functions, along with an overview of the functions that will be designed in the Rubik's game prototype.

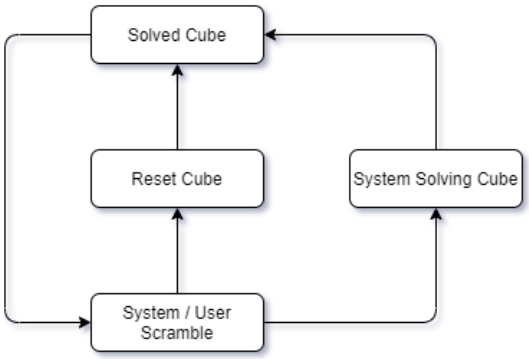

Fig. 1. Rubik's Prototype Flow

B. Data Design

In the implementation of the Rubik's solution, it requires a Rubik's object which is the only object in solving the Rubik's cube, therefore it requires a data format that can represent a 3-dimensional Rubik's object into a data type that can be executed by Programming Languages starting with breaking up the Rubik's into small parts and on Rubik's 3x3 there are 3 types of parts, namely:

1. Corner / Corner: is part of the rubik's which has 3 color sides and is located at each end of the rubik's cube and has a total of 8 corners.

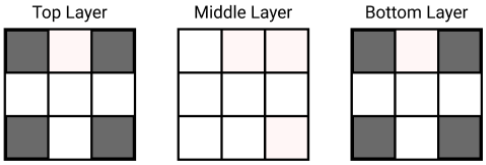

Fig. 2. Corner Section

2. Edge / edge: is part of the rubik's which has 2 sides of color and lies on each edge of the rubik's and has a total of 12 edges.

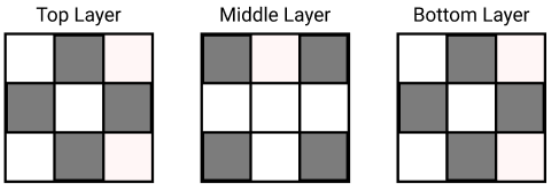

Fig. 3. Edge Section

3. Center / Center: is part of the rubik's which only has 1 color side and is in the middle on each side of the rubik's and has a total of 6 centers.

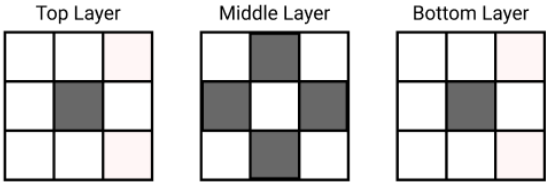

Fig. 4. Center section

As before, the data structure above is able to represent all existing edges from edge 1 to edge 12 which are represented by numbers 00 to 11, which is slightly different from the data format in the corner where each part is represented by 1 digit number, while at the edge the number of numbers that represent each 1 edge has 2 digits because there are numbers 10 and 11 which have 2 digits, then all edges are represented by 2 digits to equalize all edge data formats, and then assign the condition values to each edge as follows:

C. Algorithm Purposed

The method used in this study comes from the initial concept of God Number search which seeks the shortest number of steps to solve the Rubik's cube in all types of randomness which makes it necessary to have all existing Rubik's randoms and test them, in this case when all randoms have been successfully simulated and stored in the computer. and map the random paths that are interrelated then the Rubik's solution method can be created, but as already explained that to achieve all Rubik's randomness is very difficult because the number is very large, namely 43,252,003,274,489,856,000 random combinations which make it almost impossible to achieve by computers at this time. And in this study, the simulation method of all randomness will be reduced to only performing simulations on steps that are only traversed by random steps that are inputted in real-time so that it is expected to reduce the number of processes and time used.

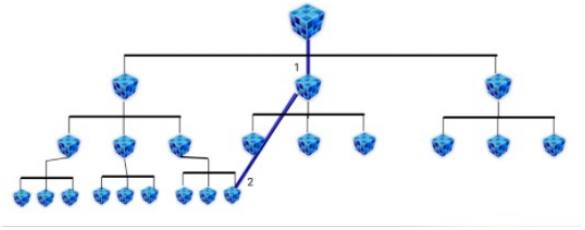

Fig. 5. Algorithm before changing lanes

The picture above illustrates a random simulation that is made in stages according to the random input received then the completion step that will be generated is in the form of a random tracing in reverse towards the top so that the rubiks will return to normal (see the path in the image with a thicker line), but there are times when the simulation The newly created random has been created before, resulting in data redundancy and a higher number of steps.

The newly created random has been created before, resulting in data redundancy and a higher number of steps. For that reason, a tracking stage is needed to check data redundancy and move the scramble path to the shortest path as shown in the following figure.

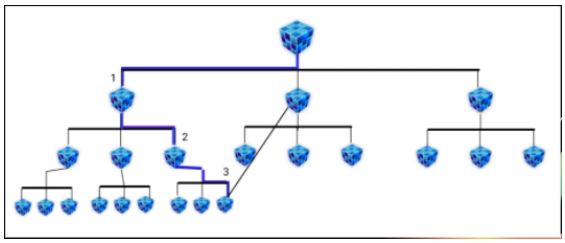

Fig. 6. Algorithm after changing lanes

The line that the scramble path changes makes the steps one less step.

D. Flowchart Purposed

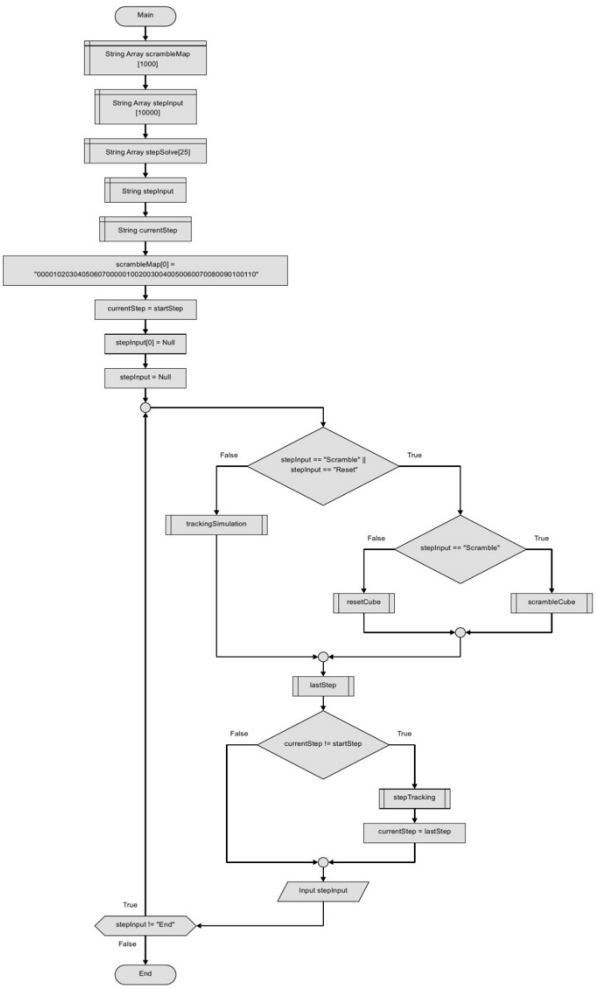

Fig. 7. Rubik's Simulation Function

The System Scramble method or automatic randomization is basically the same as the manual randomization method using the same steps (F, F', B, B', L, L', R, R', U, U', D , D') but what makes the difference is in the creation of a maneuver or a collection of Rubik's rotation steps which are run automatically by the computer in this study using the Random function.

E. Software Design

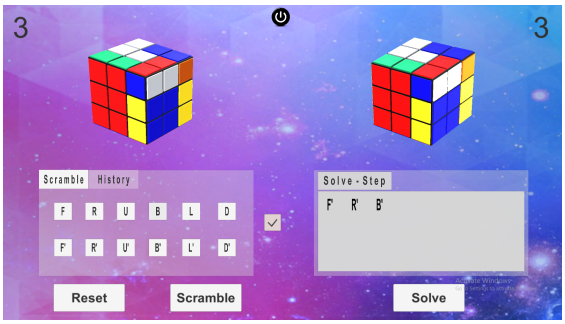

Fig. 8. Rubiks prototype interface

In the prototype interface above, it only has one display that is able to accommodate scramble and solving functions with the left window as an interface for scramble and the right window for solving, to scramble the user can press the button containing the Rules letters in the rubik or press the "Scramble" button. which will shuffle 20 random Rubik's cubes, and to do the solving by pressing the "Solve" button on the right window, the completion steps will appear in the "Solve-Step" city.

IV. Results and Analysis

At the Data Analysis stage the Rubik's section has been divided into 3 types of sections and only 2 types of sections will be used for the Rubik's data structure in this study, namely the Corner section and the Edge section, at this data design phase the Rubik's sections will be arranged by number into string data type and given the value of miss in each part, the following is the arrangement of the data structure:

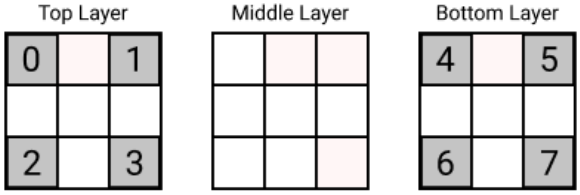

Fig. 9. Rubik's Corner Number

* Data = ("01234567")

The data structure above is able to represent all corners from corner 1 to corner 8 which are represented by the numbers 0 to 7, but as explained in the data analysis stage that each part has the possibility of a miss value which in the corner there are 3 conditions are true, miss(1), and miss(2), therefore it is necessary to give a condition sign in the form of a value that indicates the condition of each corner in the data structure and the data is generated as follows:

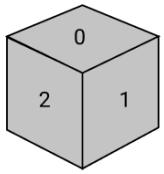

Fig. 10. Miss Corner Rubik's number

* Data = ("0010203040506070")

The data structure above increases from the previous 8 digits, then increases to 16 digits. The additional digits are adding 1 digit number to each number representing the corner, the number 0 indicates that all corners are true, the number 1 indicates the corner is miss (1), and number 2 shows the corner is miss(2). All digits added to the data above are all 0 which means all corners are true and this value will change to 1 if the side that should be above changes position to the right and will change to 2 if the side that should be above changes position to the left like shown in the image above.

After the data structure that represents the position and condition of each corner is arranged, it is necessary to create a data structure that can represent the position of each edge and also its condition with the same concept as the corner data structure, namely in the form of numbers that are

entered into the string data type and the resulting data is as follows :

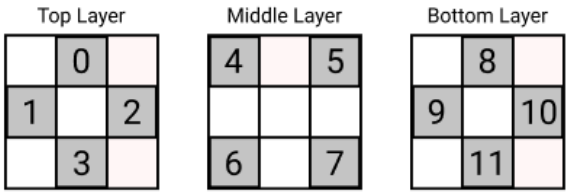

Fig. 11. Rubik's Corner Number

*Data = (“000102030405060708091011”)

As before, the data structure above is able to represent all existing edges from edge 1 to edge 12 which are represented by numbers 00 to 11, which is slightly different from the data format in the corner where each part is represented by 1 digit number, while at the edge the number of numbers that represent each 1 edge has 2 digits because there are numbers 10 and 11 which have 2 digits, then all edges are represented by 2 digits to equalize all edge data formats, and then assign the condition values to each edge as follows:

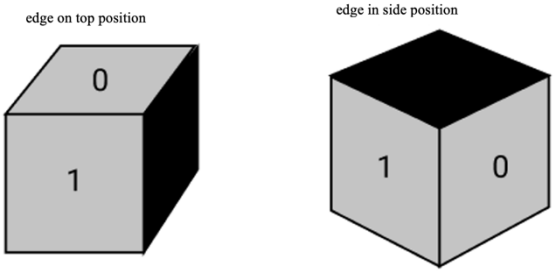

Fig. 12. Miss Edge Rubik's number

*Data = (“000010020030040050060070080090100110”)

Same as before the condition value of each edge is added after the value representing the edge is written and in the edge case there are 2 conditions, namely true and miss conditions with a value of 0 representing true and a value of 1 representing miss in the data structure above all conditions are represented by a value of 0 which means all conditions edge is true and this value will change to 1 if the side that should be on top moves down or if the side that should be on the right changes its position to the left side as shown in the picture above, this makes 1 edge represented by 3 digit number (first 2 digits indicate edge position and last 1 digit indicates miss value on edge).

After the corner and edge data structures are arranged, the next step is to combine the edge and corner data into 1 complete string and produce the following data:

Data = (“00102030405060700000100200300400500600700800901 00110”)

The result of the data structure above represents a 3-dimensional Rubik's cube that has the correct position and condition (the position is solved) into 1 string consisting of 52 digits (16 digits first represents the corner and the remaining 36 digits represent the edge).

In solving the rubik's cube, the rubik's object to be solved must be a randomized rubik's so that the previous rubik's must go through a scramble or randomization process either manually or automatically, in the manual scramble method the user is required to press the rotation steps button manually. manual so as to produce a maneuver / collection of rotational steps, in carrying out rotational steps there are limitations in the form of possible rotations, namely F, F', B, B', L, L' ,R ,R' ,U ,U ' ,D ,D' with the following explanation:

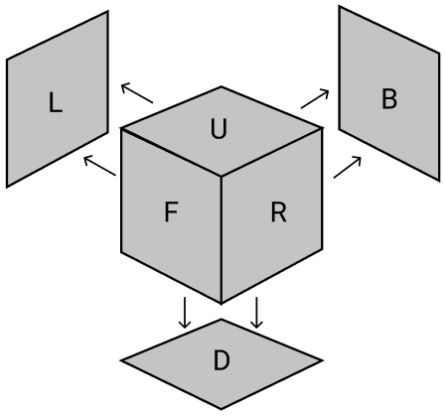

Fig. 13. Rubik's Sides

F = Front / Front (90º rotation clockwise)
F' = Accent Front (90º rotation counterclockwise)
B = Back / Back (rotation 90º clockwise)
B' = Back Accent (90º rotation counterclockwise)
L = Left / Left (90º rotation clockwise)
L' = Left Accent (90º rotation counterclockwise)
R = Right (rotation 90º clockwise)
R' = Right Accent (90º rotation counterclockwise)
U = Up / Up (90º rotation clockwise)
U' = Up Accent (90º rotation counterclockwise)
D = Down / Down (90º rotation clockwise)
D' = Down Accent (90º rotation counterclockwise)

The scramble mechanism is in the form of manual scramble input by the user by pressing the buttons for the Rubik's rotation steps which include: F, F', B, B', L, L', R, R', U, U', D ,D' and the concept of changing the data structure in the rubik's which is conditionally solved into the rubik's data structure which is in a randomized condition by each step of the scramble (F, F', B, B', L, L', R, R', U, U', D , D') and the following results are obtained:

Table I. Results Changes To The Rubiks Data Structure

Randomization Rules	Initial Condition	Condition After Scramble Position	Conditions After Adding Miss
Front (F)	0010203040506070000010020030 0400500600708 0090100110	0010602040507030000010020060 0400501100308 0090100070	0010622140507132000010020061 0400501100308 0090100071
Front Accent (F')	0010203040506070000010020030 0400500600708 0090100110	0010307040502060000010020070 0400500301108 0090100060	0010327140502162000010020070 0400500311108 0090100060
Back (B)	0010203040506070000010020030 0400500600708 0090100110	1050203000406070050010020030 0000800600704 0090100110	1152203002416070051010020030 0000800600704 1090100110
Back Accent (B')	0010203040506070000010020030 0400500600708 0090100110	4000203050106070040010020030 0800000600705 0090100110	4102203052116070040010020030 0810010600705 0090100110
Left (L)	0010203040506070000010020030 0400500600708 0090100110	4010003060502070000040020030 0900500100708 0060100110	4210013061502270000041020030 0900500100708 0061100110
Left Accent (L')	0010203040506070000010020030 0400500600708 0090100110	2010603000504070000060020030 0100500900708 0040100110	2210613001504270000060020030 0110500910708 0040100110
Right (R')	0010203040506070000010020030 0400500600708 0090100110	0030207040106050000010070030 0400200601008 0090050110	0031207240126051000010071030 0400200601008 0090051110
Right Accent (R')	0010203040506070000010020030 0400500600708 0090100110	0050201040706030000010050030 0401000600208 0090070110	0051201240726031000010050030 0401010600208 0090070110
Up (U')	0010203040506070000010020030 0400500600708 0090100110	2000301040506070010030000020 0400500600708 0090100110	2000301040506070010030000020 0400500600708 0090100110
Up Accent (U')	0010203040506070000010020030 0400500600708 0090100110	1030002040506070020000030010 0400500600708 0090100110	1030002040506070020000030010 0400500600708 0090100110
Down (D')	0010203040506070000010020030 0400500600708 0090100110	0010203050704060000010020030 0400500600710 0080110090	0010203050704060000010020030 0400500600710 0080110090
Down Accent (D')	0010203040506070000010020030 0400500600708 0090100110	0010203060407050000010020030 0400500600709 0110080100	0010203060407050000010020030 0400500600709 0110080100

For Example:

Scramble *Front* (F)

- Initial Conditions (*Solve Cube)*:

S = ("0010203040506070000010020030040050060070080090100110")

- Condition After Scramble Position F:

F = ("0010602040507030000010020060040050110030080090100070")

- Condition After the miss value at position F is added:

+2,+1 +1,+2 +1 +0, +0 +1

F = ("0010622140507132000010020061040050110030080090100071"

From the results of the scramble process above, it is known that the randomization process is divided into 2 stages, namely solving and rearranging the corner and edge data positions according to the rules of each randomized step and secondly adding the miss values on the corners and edges according to the rules of each randomization.

V. Conclusion

Based on the results of research that has been carried out using the Rubik's Solution method using a real-time cube tracking approach, the following conclusions can be drawn;

1. The method applied results in the number of steps for solving the rubik's cube which is still less than optimal.
2. The process that is run when solving the problem is quite simple with the special condition that the randomization path must be available making it have a lack of flexibility in solving Rubik's which does not have a randomization trace.

VI. Future Work

Suggestions for the method of solving Rubik's in this study to be better in the future.

1. Reviewing the existing Rubik's solution method and increasing the efficiency of the Rubik's solution step.
2. Improved process optimization when performing simulations so that there are fewer processes.

Reference

[1] T. N. Suharsono, D. Anggraini, Kuspriyanto, B. Rahardjo, and Gunawan, "Implementation of Simple Verifiability Metric to Measure the Degree of Verifiability of E-Voting Protocol," in *2020 14th International Conference on Telecommunication Systems, Services, and Applications (TSSA*, Nov. 2020, pp. 1–3. doi: 10.1109/TSSA51342.2020.9310915.

[2] Rubik's Company, "Rubik's Official Website," *Rubik's Official Website*, 2019.

[3] H. Kociemba, "Kociemba," 2018.

[4] D. Ferenc, "Rubiks Cube Wiki God Number," 2019.

[5] H. Agung, J. Jessica, and R. Januar, "Implementasi Algoritma Kociemba pada Rekomendasi Penyelesaian Langkah Permainan Rubik," *Jurnal Sistem dan Teknologi Informasi (JUSTIN)*, vol. 7, no. 3, p. 139, Jul. 2019, doi: 10.26418/justin.v7i3.30197.

[6] X. Chen and Z. J. Ding, “Solving extra-high-order Rubik’s Cube problem by a dynamic simulated annealing,” *Computer Physics Communications*, vol. 183, no. 8, pp. 1658–1663, Aug. 2012, doi: 10.1016/j.cpc.2012.03.003.

[7] J. Jery, “New State Based Algorithm For Rubiks Cube,” *International Journal of Computer Sciences and Engineering*, vol. 7, no. 2, pp. 40–45, Feb. 2019, doi: 10.26438/ijcse/v7i2.4045.

[8] P. R. Chandre, A. Raut, Riya, S. Terkar, and J. Shah, “Rubik’s Cube Solver,” *International Journal for Scientific Research & Development*, vol. 5, no. 01, 2017.

[9] A. Tyagi and P. Tyagi, “GEN-R: Genetic Algorithm Based Model for Rubik’s Cube Solution Generator,” *ijsat*, 2011.

[10] K. S. Anil, “Solution For Rubik’s Cube by Using Genetic Algorithm,” *International Journal of Engineering Sciences & Research Technology*, vol. 4, 2015.

[11] J. Hack and K. Shutzberg, “Rubiks Cube Localization, Face Detection, and Interactive Solving,” *Stanford University*, 2015.

[12] N. El-Sourani, S. Hauke, and M. Borschbach, “An Evolutionary Approach for Solving the Rubik’s Cube Incorporating Exact Methods,” 2010, pp. 80–89. doi: 10.1007/978-3-642-12239-2_9.

[13] H. Sawhney, S. Sinha, A. Lohia, P. Jalan, and P. Harlalka, “Autonomous Rubik’s Cube Solver Using Image Processing,” *INTERNATIONAL JOURNAL OF ENGINEERING RESEARCH & TECHNOLOGY*, vol. 2, no. 10, 2013.

[14] C. R. Gunawan, A. Ihsan, and Munawir, “Optimasi Penyelesaian Permainan Rubik’s Cube Menggunakan Algoritma IDA* dan Brute Force,” *Jurnal Infomedia*, vol. 3, no. 1, 2018.