Solving a Rubik's Cube with a Traveler Constraint

Fred Henle ⟨fredhenle@gmail.com⟩
Miles Henle ⟨miles.henle@gmail.com⟩
http://fredhenle.net/g4g16

for G4G16

Abstract

We show how to solve a 3×3×3 Rubik's Cube using only moves that involve a designated cubie, which we call the traveler. We also describe a method for classifying cube states as solvable or not for particular travelers. In particular, there are some states which cannot be solved by any corner traveler, but for every state there exist at least four edge travelers which can solve it.

1 Introduction

The challenge we set ourselves was to figure out how to solve a scrambled Rubik's Cube by choosing a cubie, for example the red-white-green corner, and then only allowing moves that involve that cubie. We call the chosen cubie the **traveler**. In addition to being an interesting mathematical problem, we thought that such a method, while obviously not as efficient as existing methods, might have its own appeal. For one thing, traveler sequences might be easier to memorize for people who like to focus on something more visual, like the path that the traveler takes as it wanders and then returns home. It's also amusing instead to think of the traveler as the only cubie that *doesn't* move. Imagine mounting a Rubik's Cube on a pedestal with one corner fixed to the base and only allowing "wide" moves (moves which grab the outer two thirds of the cube instead of just one third).

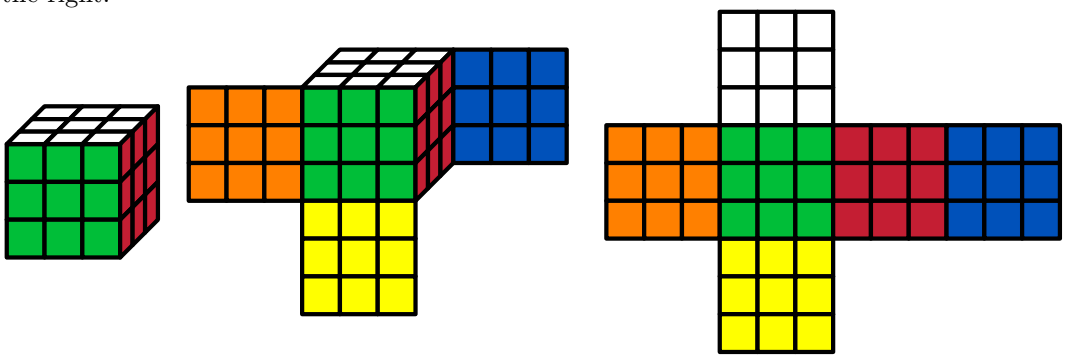
This problem is similar in spirit to puzzles such as sokoban, and to the challenge of making a continuous line drawing, where the pen never leaves the paper.

1.1 Definitions

In this paper we will consider two notions of what it means to solve a cube. Generally, we consider a cube to be solved if every face has facelets of the same color, regardless of orientation. However, we will also consider when it is possible to solve the cube for a particular desired orientation as well.

1.2 Notation and Visualization

When we talk about colors, we are using the blue-orange-yellow color scheme, with a default orientation of white on top and yellow on the bottom, green in front and blue in back, orange on the left and red on the right.



F	**F**		front face
B	**B**		back face
U	**U**		up face
D	**D**		down face
L	**L**		left face
R	**R**		right face
M	**M**		middle slice
E	**E**		equator slice
S	**S**		standing slice
			direction
			direction
			direction
x	**x**	[**x**]	whole cube
y	**y**	[**y**]	whole cube
z	**z**	[**z**]	whole cube

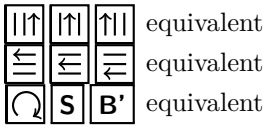


Table 1: base symbols for individual moves

F	**F**		quarter turn clockwise
Fp	**F'**		quarter turn anticlockwise
F2	**FF**		half turn

Table 2: variations for move direction or degree

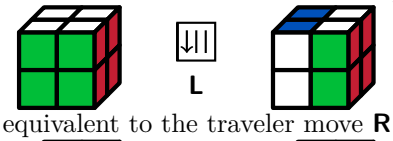
We use Singmaster notation `F B U D L R` for clockwise rotations of the six faces, as well as `M E S` for slices and `x y z` for flops (reorienting the cube as a whole). See Table 1.

For each type of move, there are three variants indication direction/degree. See Table 2.

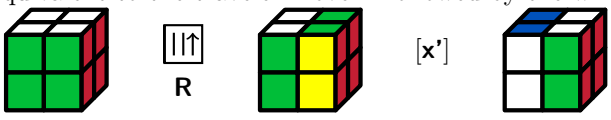
2 Solving the 2-cube with a corner traveler

The 2×2×2 cube, or 2-cube, consists of eight corner cubies. If we don't care about orientation, a 2-cube is trivially solvable by any traveler. Since every move uses exactly half the cubies, any move that does not use the traveler can be replaced by the complementary move in the opposite direction.

For example, if the traveler is the red-white-green cubie RWG  then while **L** is not a traveler move because it doesn't involve RWG,

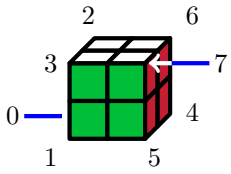


L is equivalent to the traveler move **R** followed by the whole cube reorientation **x'**



3 Permutations and Parity

Let's assign a number to each corner as follows:



The four orange corners on the left are 0, 1, 2, 3; the four yellow corners on the bottom are 0, 1, 4, 5; and so forth.

We can represent the current occupants of the eight corner positions as a string. When the cube is in the solved state, each corner is in its home position and so the corner state is simply `01234567`. After rotating the front face clockwise with **F** the new corner state is `05214763` because corner 5 is in position 1, corner 1 is in position 3, corner 7 is in position 5, and corner 3 is in position 7. We can also represent the permutation of corners in a form of cycle notation, with corners in home position (that is, unchanged by a move) outside parentheses, and each cycle of corners that have been moved within parentheses. After rotating the front face clockwise with **F** the new cycle representation is `0246(1375)` because the back corners haven't moved but the front corners are in a single cycle. After rotating the front face clockwise with an additional **F** the new cycle representation is `0246(17)(35)` because the back corners still haven't moved but the front corners are now in two pairs of opposite corners that have switched places.

It is important to note that no cube state that can be achieved with an odd number of quarter turn face rotations can also be achieved with an even number of quarter turn face rotations, and vice versa. Furthermore, one can determine from an examination of the cycle representation of a cube whether it is an even or odd number of turns from the solved state. Specifically, the number of turns is even if and only if there are an even number of cycles of even length.

We will use the term "global parity" to refer to this. The solved state of the cube, along with any state reachable by an even number of quarter turn face rotations, has even global parity, while all other states have odd global parity.

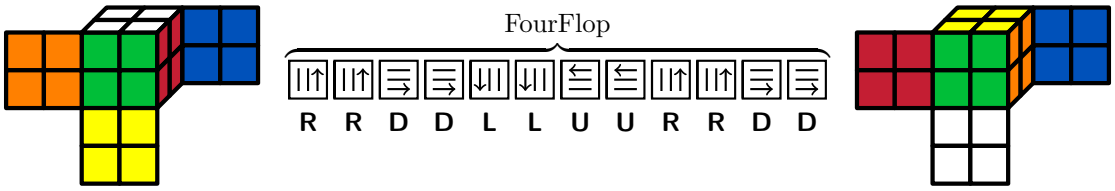
For a more mathematical discussion of Rubik's Cube permutations, see Lucas Garron's paper on the topic.[1]

3.1 Flopping the 2-cube

We generally consider whole cube reorientations (or flops), notated as [**x**], [**y**], or [**z**], to be allowable, but what if we don't? Is it possible to change the orientation of the 2-cube with a pure traveler sequence, that is, without using [**x**], [**y**], or [**z**]? The answer is yes, although only for half the possible orientations.

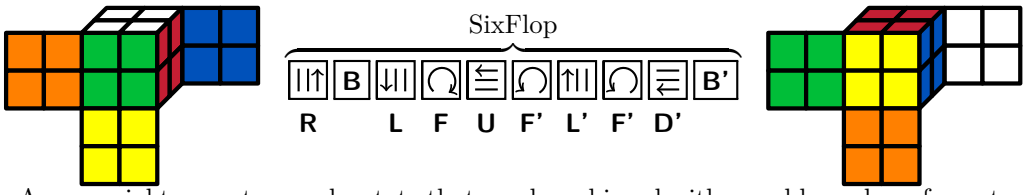
Here are two examples of flopping the cube:

`[FourFlop] R2 D2 L2 U2 R2 D2` is equivalent to `z2`.



`[SixFlop] R B L F U Fp Lp Fp Dp Bp` is equivalent to `x zp`.

[1] Lucas Garron. *The Permutation Group of the Rubik's Cube*. https://stanford.garron.us/class/math109/files/math109_conj6.pdf. 2010



As one might expect, no cube state that can be achieved with an odd number of quarter turn global rotations can also be achieved with an even number of quarter turn global rotations, and vice versa. Although we are still talking about the 2-cube it is easier to understand why this is true if we consider the face center cubies of the 3-cube and the cycle representation of those centers. The center parity of the cube is even if and only if there are an even number of cycles of even length.

To understand why only even parity flops are achievable with corner traveler moves, it is necessary to make one other observation, which is that any sequence of traveler moves that returns the traveler to its home position and orientation must take an even number of quarter turn face rotations. Let us define the term "traveler parity" for a particular traveler as even if and only if the traveler is an even number of quarter turn face rotations from its home position and orientation. Since the global parity flips for every quarter turn face rotation, and the traveler parity flips for every quarter turn face rotation, and the solved state has even global parity as well as even traveler parity for every traveler, a cube state is solvable by a traveler if and only if the global parity is the same as the traveler parity.

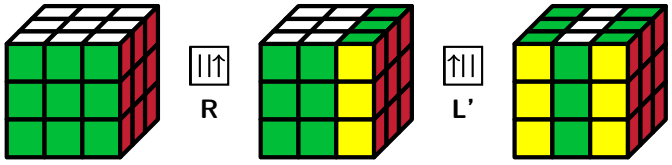
All flops have even corner parity, whether their center parity is even or odd. However, the traveler parity of every corner is the same as the center parity (for flops). Therefore, a flop may be solved by a corner traveler only if the center parity is even.

So, while a 2-cube may trivially be solved by any corner traveler if we don't consider global orientation of the cube, if we want to solve the cube with a corner traveler for a particular orientation without using [**x**], [**y**], or [**z**], then a given corner traveler can only solve half of all cube states (those where the global parity matches the traveler parity of the particular corner). Furthermore, we can't always pick another corner to be the traveler, because for certain cube states, in particular flops of odd center parity, no corner has the same traveler parity as the global parity.

4 Solving the 3-cube with a corner traveler

This is the perfect starting point for a discussion of the 3-cube and corner travelers. Because the 3-cube has center cubies, and those cubies have a fixed relationship with each other, and because corner traveler moves are simple face rotations that do not affect centers, it only makes sense to reorient a scrambled cube into a reference orientation (such as white on top and green in front) and then consider solvability. The task of solving the 3-cube will then be to get the twelve edges and the eight corners into the correct positions and orientations relative to the centers. Leaving aside the problem of solving the edges, the problem of solving the corners of the 3-cube is identical to the problem of solving the 2-cube without allowing global orientation changes.

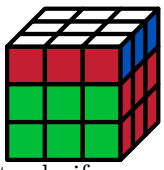
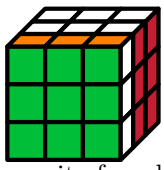
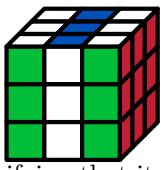
This means that for a given corner traveler, at least half of all cube states will not be solvable by that traveler, and that certain cube states will not be solvable by any corner traveler, in particular states such as



where the cube has even global parity but every corner has odd traveler parity. Understanding this somewhat disappointing result, we next consider what it would mean to use an edge cubie as a traveler.

5 Solving the 3-cube with an edge traveler

A corner traveler can move in any of three ways, all three being face rotations for the three faces that intersect at that corner. An edge traveler can also move in any of three ways, two of them being face rotations for the two faces that intersect at that edge, with the third way being a slice such as **M** [|↓|], **E** [⇉], or [**S**]. For the white-green cubie WG in home position the base moves are **U** [⇇], **F** [↻], and **M** [|↓|].

We need to clarify our definition of traveler parity for edge travelers by specifying that it is even if and only if the edge traveler can reach its home position and orientation relative to the centers in an even number of quarter turn face rotations. In other words, for the purpose of determining traveler parity of edge travelers, ignore slices (which are equivalent to a pair of face rotations and in any case do not change the edge's position relative to the centers).

The things we said about corner travelers also apply to edge travelers. In particular, a cube state may be solved by an edge traveler only if the traveler parity of that edge is the same as the global parity, i.e., the corner parity relative to the centers. This means that there are cube states that cannot be solved by certain edge travelers. However, unlike corner travelers, there is no cube state that cannot be solved by any edge traveler. This is because the only way for all edges to have the same traveler parity is if that parity is even and the global parity is also even. For any cube state there are at most eight edges that cannot solve the cube, and there are always at least four edges that can solve it. There is surely a way to prove this, but we were satisfied with observing the results of a compelling number of simulations.

6 An Edge Traveler Algorithm

As of this writing, we did not have time to fully flesh out the algorithm to the point that a computer program could execute it without human decision-making, but we are confident that we have found all the necessary elements, and as a proof of concept we have used those elements to solve a scrambled cube.

The overall structure is as follows. First, identify an edge traveler and move it into home position and orientation. Thereafter, use only edge traveler sequences that return the traveler to its home position and orientation. Start with the corners, moving them into position, then solve the corner orientation. Next, move centers into positions that match the corners. Finally, move the edges into position and solve their orientation using traveler sequences that do not affect the corners or centers.

6.1 The Traveler

Orient the cube so the centers are in the desired positions. Let us assume that is white on top, yellow on the bottom, green in front, blue in back, orange on the left, and red on the right.

Determine the global parity by counting the number of corner cycles of even length.

Choose an edge traveler whose traveler parity matches the global parity, and move it into the correct position and orientation relative to the centers. We'll assume that it is the white-green edge, but for any other edge, just orient the cube such that the traveler is between the top and front faces, and the sequences described below will work the same.

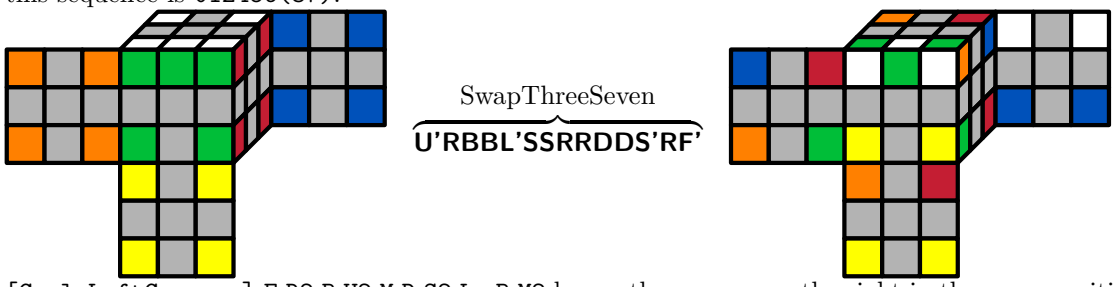
6.2 The Corners

The traveler sequences described from this point onward begin and end with the edge traveler in its home position and orientation.

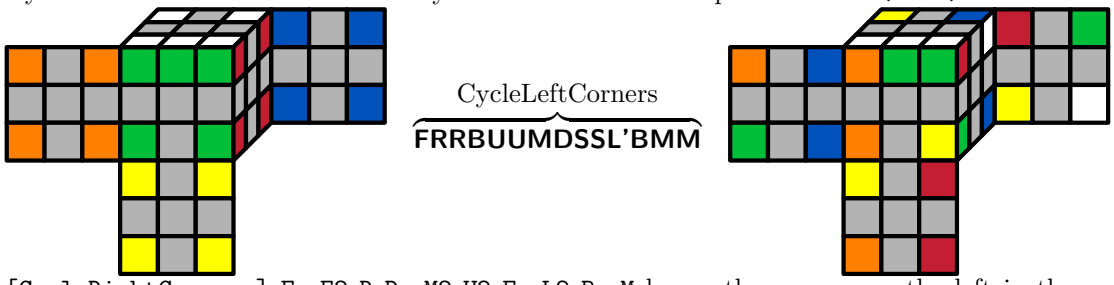
6.2.1 Corner Positions

The following three sequences are sufficient to move the corners into the correct position relative to the edge traveler, but have side effects on corner orientation and the positions and orientation of the edges and centers.

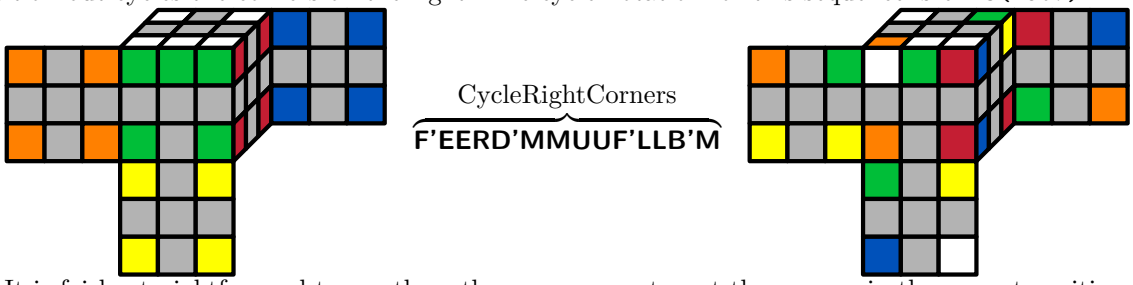
`[SwapThreeSeven] Up R B2 Lp S2 R2 D2 Sp R Fp` leaves all corners in the same position except for swapping corners 3 and 7, which are the two corners on either side of the traveler. The cycle notation for this sequence is `012456(37)`.



`[CycleLeftCorners] F R2 B U2 M D S2 Lp B M2` leaves the corners on the right in the same position but cycles the corners on the left. The cycle notation for this sequence is `4567(0123)`.



`[CycleRightCorners] Fp E2 R Dp M2 U2 Fp L2 Bp M` leaves the corners on the left in the same position but cycles the corners on the right. The cycle notation for this sequence is `0123(4567)`.



It is fairly straightforward to use these three sequences to put the corners in the correct positions. See Table 3 for an example. Start by moving all corners to the correct half (orange to the left and red to the right) by cycling the left corners until there is a red corner in position 3 which belongs on the right, and cycling the right corners until there is an orange corner in position 7 which belongs on the left, then swap them. Repeat as necessary.

Once all the corners are in the correct half, adjust their order by swapping pairs of corners on the same side. For example, to swap corners 0 and 2, cycle the left until corner 2 is in position 3, swap positions 3 and 7, then cycle the left until corner 0 is in position 3, swap positions 3 and 7, then cycle the left until the borrowed red corner is back in position 3, and swap positions 3 and 7 for a third time.

Finally, cycle the left and right until the corners are not only in the correct order but the correct positions.

6.2.2 Corner Orientations

`[RotateOneFourFive] F R2 B2 Lp D F L2 B2 Rp U` performs a clockwise rotation of corners 1, 4, and 5, but has side effects on the positions and orientation of the edges.

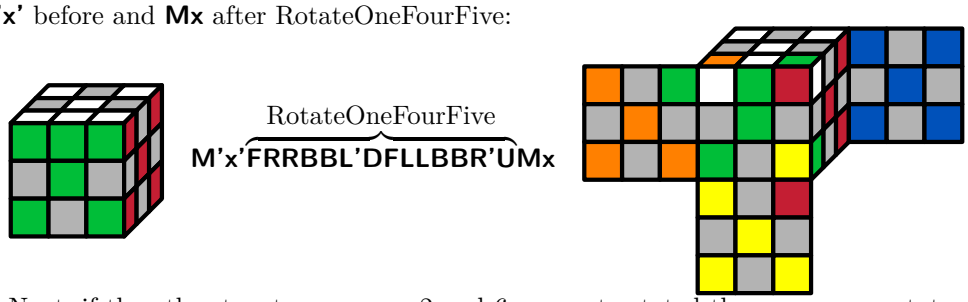
corner state	move	result	goal
01234567 01234567	**RBD**	02734165 0346(1572)	mess up the corners for this example
02734165 0346(1572)	left	30274165 246(01573)	move 0–3 to the left and 4–7 to the right
30274165 246(01573)	right	30275416 2(01673)(45)	
30275416 2(01673)(45)	right	30276541 25(0173)(46)	
30276541 25(0173)(46)	swap	30216547 257(013)(46)	
30216547 257(013)(46)	right	30217654 2(013)(47)(56)	swap 4 and 6
30217654 2(013)(47)(56)	swap	30247651 2(01743)(56)	
30247651 2(01743)(56)	right	30241765 26(0143)(57)	
30241765 26(0143)(57)	right	30245176 2(01543)(67)	
30245176 2(01543)(67)	swap	30265174 2(0154763)	
30265174 2(0154763)	right	30264517 2457(0163)	
30264517 2457(0163)	right	30267451 2(0174563)	
30267451 2(0174563)	swap	30217456 2(013)(4567)	
30217456 2(013)(4567)	right	30216745 2(013)(46)(57)	put 4567 in the correct positions
30216745 2(013)(46)(57)	right	30215674 2(013)(4765)	
30215674 2(013)(4765)	right	30214567 24567(013)	
30214567 24567(013)	swap	30274561 2456(0173)	swap 1 and 2
30274561 2456(0173)	left	73024561 456(02317)	
73024561 456(02317)	swap	73014562 456(027)(13)	
73014562 456(027)(13)	left	17304562 456(03271)	
17304562 456(03271)	left	01734562 013456(27)	
01734562 013456(27)	left	30174562 456(01273)	
30174562 456(01273)	swap	30124567 4567(0123)	
30124567 4567(0123)	left	23014567 4567(02)(13)	put 0123 in the correct positions
23014567 4567(02)(13)	left	12304567 4567(0321)	
12304567 4567(0321)	left	01234567 01234567	

Table 3: corner solving example

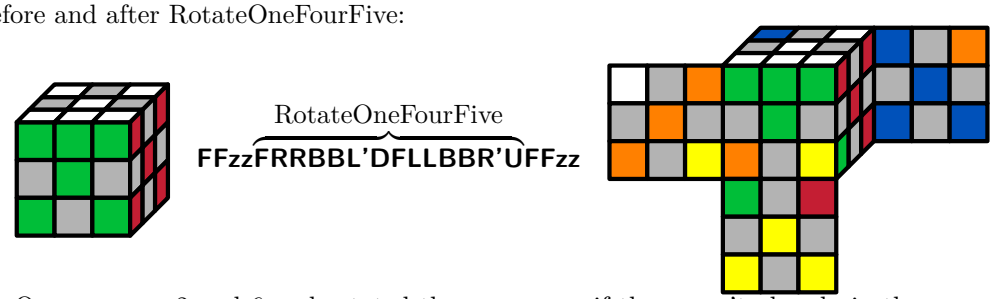
There are many other sequences that rotate other sets of corners, but RotateOneFourFive is sufficient in combination with setup moves to put the traveler in another location before the sequence, and the inverse setup moves afterwards.

Let's start by solving the rotation of the corners on either side of the traveler, in other words the other corners with white and green, corners 3 and 7. If they are already in the correct orientation, there's nothing to do. If they are not rotated the same way (because exactly one is in the correct orientation, or they are both incorrect but one is rotated clockwise and the other one anticlockwise) then we can rotate corner 7 (with side effects of rotating corners 1 and 4) by performing **MMxx** before and after RotateOneFourFive:

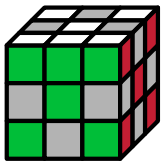
Once corners 3 and 7 and rotated the same way, if they aren't already in the correct orientation, rotate them both either once or twice as necessary (with side effect of rotating corner 5) by performing **M'x'** before and **Mx** after RotateOneFourFive:



Next, if the other two top corners, 2 and 6, are not rotated the same way, rotate corner 2 until it matches the rotation of corner 6 (with side effects of rotating corners 1 and 5) by performing **FFzz** before and after RotateOneFourFive:

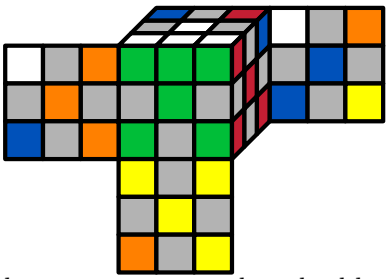


Once corners 2 and 6 and rotated the same way, if they aren't already in the correct orientation, rotate them both either once or twice as necessary (with side effect of rotating corner 0) by performing **FFzzMx** before and **M'x'FFzz** after RotateOneFourFive:



RotateOneFourFive

FFzzMxFRRBBL'DFLLBBR'UM'x'FFzz

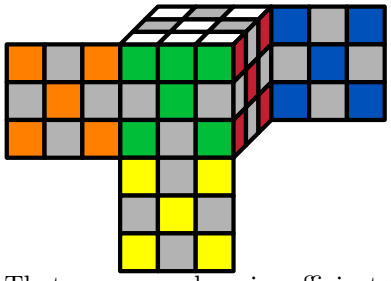


Once the top corners are all in the correct orientation, the bottom corners can be solved by using multiples of **Uy'** for setup to choose which three of the bottom corners will be rotated. If all four bottom corners are correct, there is nothing to be done. If one bottom corner is correct, then the other three will be in the same orientation and can be solved in one round. If two bottom corners are correct, then the other two will be in opposite orientations, in which case rotate the two solved corners along with one of the rotated corners until exactly one corner is solved, at which point you are in the prior case and can solve the other three together.

6.3 The Centers

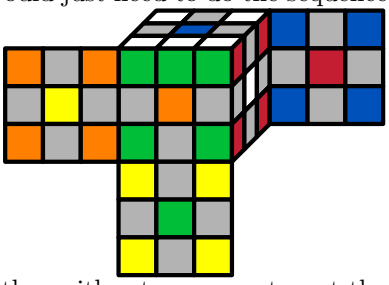
Since solving corner orientations doesn't move the centers around, we could actually have done this step after solving corner positions, which is the last step that moves centers around as a side effect.

There are many traveler sequences for cycling centers, but one of the easiest to remember is `[CycleTwoTriples] U S Up M`, which cycles two triples of centers while also cycling five edges. Since we haven't solved the edges yet, it doesn't matter that it has a side effect of cycling edges, but if you want to move the centers without moving the edges you would just need to do the sequence five times.



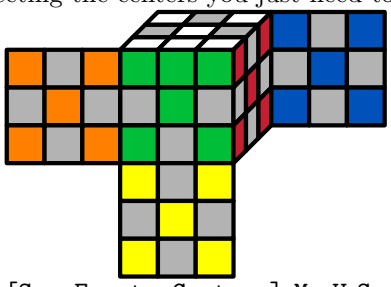
CycleTwoTriples

USU'M



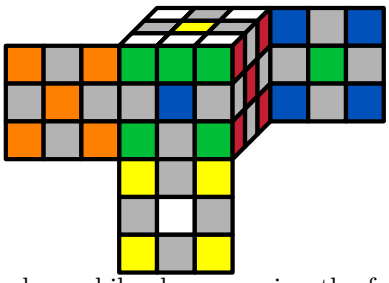
That sequence alone is sufficient to solve centers, together with setup moves to put the traveler in the correct position and orientation to effect the desired result. Of course if the centers are already in the correct positions there's nothing to do. If none are in the correct positions, then one setup sequence plus one or two applications of CycleTwoTriples is all that is needed. If two centers are in the correct position, then two setups and applications of CycleTwoTriples are needed, but for convenience, here are two different sequences that swaps two pairs of opposite centers.

`[SwapMiddleCenters] Mp U2 Mp U2` cycles three edges while also swapping the top and bottom (white and yellow) and front and back (green and blue) centers. Since we haven't solved the edges yet, it doesn't matter that it has a side effect of cycling edges, but if you want to move the centers without moving the edges you just need to do the sequence three times. If you want to cycle the edges without affecting the centers you just need to do the sequence twice.



SwapMiddleCenters

M'UUM'UU



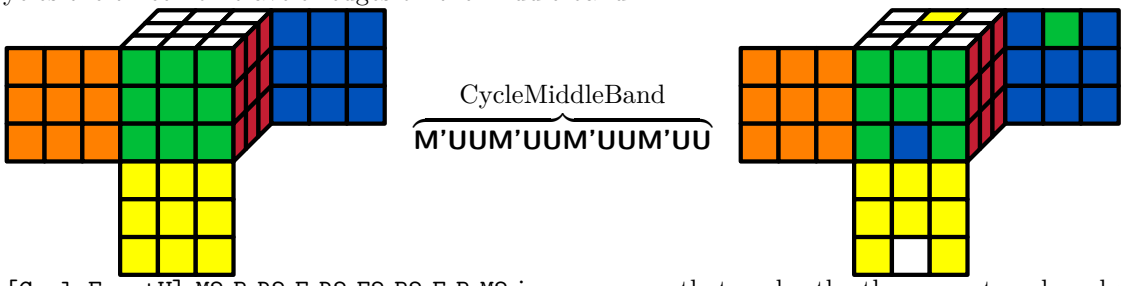
`[SwapEquatorCenters] Mp U Sp U2 Sp U M` cycles three edges while also swapping the front and back (green and blue) and the left and right (orange and red) centers. Since we haven't solved the

edges yet, it doesn't matter that it has a side effect of cycling edges, but if you want to move the centers without moving the edges you just need to do the sequence three times. If you want to cycle the edges without affecting the centers you just need to do the sequence twice.

6.4 The Edges

6.4.1 Edge Sequences

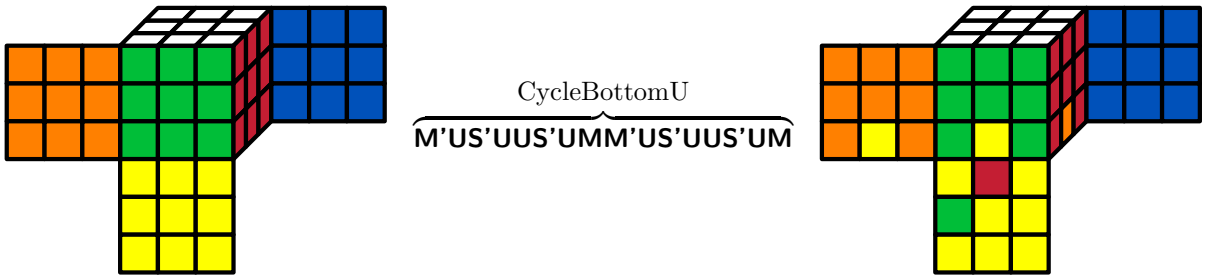
`[CycleMiddleBand]` `Mp U2 Mp U2 Mp U2 Mp U2` is exactly what the previous paragraph described, and it cycles the three non-traveler edges on the middle band.



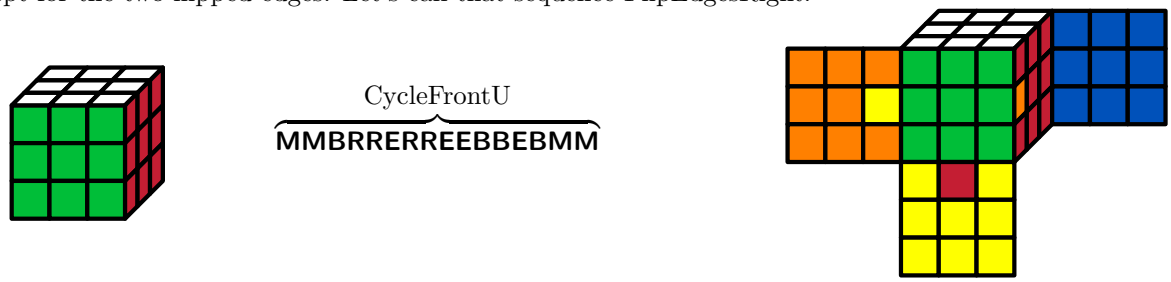
`[CycleFrontU]` `M2 B R2 E R2 E2 B2 E B M2` is a sequence that cycles the three non-traveler edges on the front face:

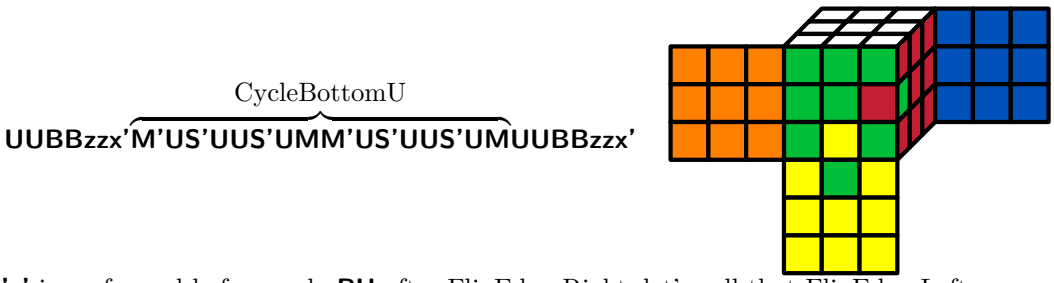
`[SwapEquatorCenters]` `Mp U Sp U2 Sp U M` (previously seen) cycles three edges while swapping centers:

`[CycleBottomU]` `Mp U Sp U2 Sp U M Mp U Sp U2 Sp U M` performs SwapEquatorCenters twice in order to put the centers back to their home positions; it cycles three bottom edges but also flips two of them:



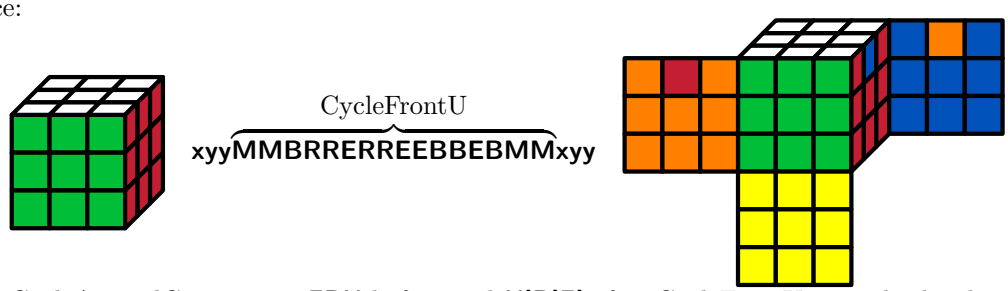
When **UUBBzzx'** (`U2 B2 z2 xp`) is performed before and after CycleBottomU, it cycles the same front edges but in the opposite direction. Put together with CycleFrontU, everything cancels out except for the two flipped edges. Let's call that sequence FlipEdgesRight:





When **U'R'z'** is performed before and **zRU** after FlipEdgesRight, let's call that FlipEdgesLeft:

CycleTopU uses **xyy** before and after CycleFrontU to cycle the three non-traveler edges on the top face:



CycleAroundCorner uses **FRU** before and **U'R'F'** after CycleFrontU to cycle the three edges next to the red-yellow-green corner RYG.

6.4.2 White Edge Positions

1. If the red-white edge RW is on the top (white) face, move it to the back with CycleTopU and then use CycleMiddleBand to send it to the bottom face.
2. If the white-blue edge WB is on the top (white) face, move it to the back with CycleTopU and then use CycleMiddleBand to send it to the bottom face.
3. If the orange-white edge OW is not already on the top face, perform **Uy'** until OW is on the front face, then perform CycleFrontU until OW is on the bottom face, then use CycleMiddleBand to put OW on the top face.
4. Use CycleTopU to move OW to the right side.
5. Perform **Uy'** until RW is on the front face, then perform CycleFrontU until RW is on the bottom face, then use CycleMiddleBand to put RW on the top face.
6. Use CycleTopU to move RW to the right side. This will simultaneously move OW to the left side.
7. Perform **Uy'** until WB is on the front face, then perform CycleFrontU until WB is on the bottom face, then use CycleMiddleBand to put WB on the top face.

6.4.3 Equator Edge Positions

1. Find the red-blue edge RB and, if it is not already in its home position, use **Uy'** to bring it to the front face. Use CycleFrontU to bring it closer to its home position. Repeat until RB is in its home position. We will need to choose which faces we use CycleFrontU to avoid disturbing the edges we already solved.
2. Find the orange-blue edge OB and if it is on the bottom of the red or blue faces, use CycleBottomU to bring it to the bottom of the green or orange face. If necessary, navigate the traveler with **Uy'** to choose which 3 bottom edges to cycle. As before use CycleFrontU on the green and/or orange face to bring OB to its home position.
3. Find the orange-green edge OG and if it is on the bottom of the red, blue, or orange faces, navigate with **Uy'** and use CycleBottomU to bring it to the green face. Use CycleFrontU on the green face to put OG in its home position.
4. Find RG and if it is on the bottom, use CycleBottomU to bring it to the bottom of the green or red face, navigating the traveler with **Uy'**. Return the traveler to the green face and use CycleAroundCorner until RG is in its home position.

6.4.4 Yellow Edge Positions

1. Use CycleBottomU, navigating the traveler with **Uy'** as necessary, to bring the yellow-green edge YG into its home position.
2. By now, either the other three yellow edges will be all in the correct position, or they will be in a 3-cycle. Use **Uy'** until the blue face is in front, then use CycleBottomU until the last 3 edges are also solved.

6.4.5 White Edge Orientations

1. Use **xyy** to make white the front face, and use the FlipEdgesRight to orient OW if needed. This may leave BW in the wrong orientation. Do **not** reverse the setup yet.
2. Use FlipEdgesLeft to orient WB if needed. This may leave RW in the wrong orientation.
3. Use **Uy'** to make RW the front-right edge, and use FlipEdges to orient it if needed. Then undo that setup with **U'y**, and make white the top face again with **xyy**.

6.4.6 Other Edge Orientations

1. Use FlipEdgesLeft to orient YG if needed. This may leave OG in the wrong orientation.
2. Use **Uy'** to make orange the front face. Use FlipEdgesRight to orient OG if needed. This may leave OY in the wrong orientation
3. Use FlipEdgesLeft to orient OY if needed. This may leave OB in the wrong orientation.
4. Repeat this process, solving the orientation of one edge at a time, such that the second edge being flipped is further to the left. This creates an expanding safe zone of edges that are in the correct orientation.
5. The last step is to have red be the front face, and use FlipEdgesLeft to orient RY if needed. Because there will always be an even number of edges with the wrong orientation at the same time, this step is guaranteed to leave the last edge RG in the correct orientation as well.

7 Future Work

7.1 Center Traveler

We considered using a corner traveler but ended up using an edge traveler. What about a center traveler? Like the other two, there are three types of moves. Whereas a corner traveler has three face rotations and an edge traveler has two face rotations and one slice, a center traveler has one face rotation and two slices.

7.2 Shape Mods

There are puzzles that are mechanically identical to the Rubik's Cube but are shaped or decorated in such a way that center cubie orientation matters. Can such puzzles be solved with a traveler constraint?

7.3 Other Sizes

Is there a traveler that could solve a 4-cube? What about even larger cubes? What about rectangular twisty puzzles, such as the 3×3×2 or the 2×2×4?

7.4 Dodecahedra and Other Forms

Can the Megaminx (or Kilominx) be solved with a travel constraint? What about the Skewb?

References

Garron, Lucas. *The Permutation Group of the Rubik's Cube*. https://stanford.garron.us/class/math109/files/math109_conj6.pdf. 2010.