Rubik Cube Solver - F2L (First 2 Layer) and SLL (Simple Last Layer)

Introduction

The cube has 6 faces named: 1:F:Front, 2:R:Right, 3:U:Up, 4:B:Back, 5:L:Left, 6:D:Down.
And 3 middle layer faces are named: 7:X, 8:Y, 9:Z. The rotations for the 9 faces 90 degrees Clock Wise are marked as: F R U B L D X Y Z or -1 -2 -3 -4 -5 -6 -7 -8 -9.
In Counter-Clock Wise as: F' R' U' B' L' D' X' Y' Z' or +1 +2 +3 +4 +5 +6 +7 +8 +9.
For 180 degrees as: F2 R2 U2 B2 L2 D2 X2 Y2 Z2 or #1 #2 #3 #4 #5 #6 #7 #8 #9.
The rotations of the cube are marked as: -x -y -z +x +y +z #x #y #z.
The middle layer rotations may also be marked as: -X -Y -Z +X +Y +Z #X #Y #Z.
Some times, use x3 to represent -3 +3 #3. Use C1 C2 C3 for x y z. And use xC3 for -z +z #z.
This guide prefers to use -+# and 6:D may change to 3:U, and let +3=-6，-3=+6, to make some Alg easy to remember. Such as: L3.2a shown as (32)- ++-++#-# and L3.2c as x(2323)++-++---.
Then the two Alg are similar to each other. Most Alg rotates only faces 2 and 3, or the faces are repeated in a short period, so remember the sequence of the rotation sign is enough.

L2.1a:(23)+++++----. L2.1b:(23)-----++++. (a and b sign reversed in most steps)
L3.1a:(231)+++---. L3.1b:(213)+++---。L3.1c:(234)---+++.
L3.2a:(32)- ++-++#-#. L3.2b:(32)+ --+--#+#. L3.2c:x(2323)++-++---. L3.2d:x(2323)--+--+++.
L3.2e:(32)#-#++-++ -. L3.2f:(32)#+#--+-- +. (L3.2e=L3.2bR=Reverse of L3.2b)(L3.2f=L3.2aR)
L3.3a:y(2323)+++#--+##. L3.3b:y(2323)---#++-##.
L3.4a:(23)-+ -----+++#. L3.4b:(23)+- +++++---#.

For easy reference, use 1,2,3,4 to indicate the locations of the corners (1:RF,2:RB,3:LB,4:LF) and the edges (1:F,2:R,3:B,4:L).
For example, use (1+) for corner/edge 1 to indicate the other 3 corners/edges (2,3,4), make (+) rotate (2->3->4->2) or (+) corner spins.
Use (X)/(+) to indicate corners/edges 1:3 exchanged, 2:4 exchanged.
Use (=)/(\\) to indicate corners/edges 1:4 exchanged, 2:3 exchanged.
Use (||)/(//) to indicate corners/edges 1:2 exchanged, 3:4 exchanged.

There are 3 steps for each Alg: (1) Find the target piece; (2) Relocate; (3) Apply Algorithm.
The first 2 steps rotate faces 3 & 6 only to not disturb the finished layers. When applying Alg, the finished layers will be disturbed temporarily but recovered after finished.
Any mistake will destroy the finished layers, so be careful when applying Alg. To recover the disturbed layers, it is better to begin from step L1.1.
Most Alg rotates faces 2 & 3 only; use the thumb & middle finger of the **left** hand to hold the **left**-down 8 pieces of faces 1 & 4. The index finger may help to rotate layer 3. Some Alg rotate face 6 also, then do not hold the 1st layer 4 pieces.
Before explaining the F2L method, following will give a brief introduction of the Layer by Layer method.

Layer by Layer method (L123)

This method uses L1.1ab (L1.1a and L1.1b), L1.2ab, and L1.3abc to finish Layer 1. Then uses L2.1abc to finish Layer 2. And uses L3.1abc, L3.2abcdef, L3.3ab, and L3.4abef to finish Layer 3. Note that, a fast method called First 2 Layer (F2L) can be used to replace L1.3abc and L2.1abc. Orientation Last Layer (OLL) and then Permutation Last Layer (PLL) are used most often to finish Layer 3 in two steps. But OLL needs 54 Alg. and PLL needs 21 Alg. and they are hard to remember. In steady, a modified OLL needs 3 more Alg. and a modified PLL needs 4 more Alg. are introduced. A Simple Last Layer (SLL) is proposed using only the basic Alg. for Layer 3 (L3.1abc, L3.2abcdef, L3.3ab, and L3.4abef). With simple testing, this method can complete the Last Layer in 5 steps or less.

Before step L1.1, put layer 1 to layer 3 and return to layer 1 after step L1.3.

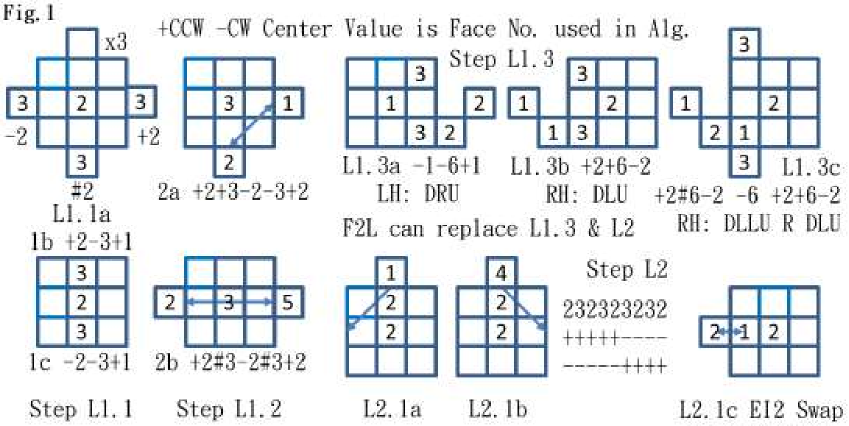
Step L1.1: (Finish face 3 colors of 4 edges)
Target: Edge marked with 3.
Relocate: For 1st target: the side not with color 3 Rot(xC3) to face 2;
For 2nd target: the side with color 3 Rot(xC3) to face 2;
For both: the target location Rot(x3) to face 2.
Algorithm: 1st:L1.1a: -2 or #2 or +2   2nd:L1.1b: +2-3+1 or L1.1c: -2-3+1

Step L1.2: (Finish side face colors of 4 edges)
Target: 2 Edges Marked with 1,2 or 5,2. (side color doesn t match the side face center color)
Rot(x3) such that at least 2 side colors match the side face center color.
Relocate: Rot(xC3) to put not matched sides on faces 1,2 or 5,2.
Algorithm: L1.2a(12): +2+3-2-3+2   L1.2b(52): +2#3-2#3+2

Step L1.3: (Finish 4 corner pieces) (May be replaced by F2L Algorithm)
Target: Corner marked with 3. Rot(xC3) to put the marked corner between face 1 and face 2.
Relocate: Color 3 on the side of Layer 1: Rot(xL12) to match a face to the other side color.
Color 3 on face 6: Rot(xL12) to cross-match color of face 1/2 to corner side 2/1.
Color 3 on the side of Layer 3: the **right** side Rot(xC3) to face 2. And use 3a/3b.
Algorithm: L1.3a   -1-6+1   L1.3b   +2+6-2   L1.3c   +2#6-2 -6 +2+6-2
3@F1:   **LH**:DRU   3@F2:   **RH**:DLU   3@F6:   **RH**:DLLU **R** DLU (by hand sense)
For the target on L3, use 3a/3b to put it to L1 only. And need to relocate and use Alg again.

Step L2.1: (Finish 2nd layer 4 edge pieces) (May be replaced by F2L Algorithm)
Target: Edge mark with 1,2 or 4,2. (both side colors not with color 3)
Relocate: Side color 2 Rot(x3) to match side face color and Rot(xC3) to face 2.
For 2x1, may Rot(x3) an L3 edge to E34 for color sweep to reduce 1 next step.
Algorithm: 12: 232323232   42: 232323232   2x1: 23232323131 (This Alg is based
L2.1a: +++++----   L2.1b: -----++++   L2.1c: -#+--#+-++- on F2L Algorithm)

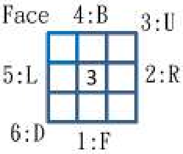
Fig. 1



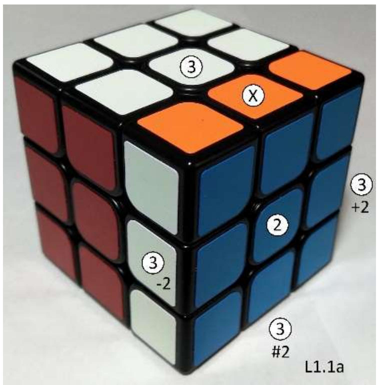
+CCW -CW Center Value is Face No. used in Alg.

2323 2323 131
-#+- -#+- ++-
(E34 Ex.-> E32)
(From F2L)

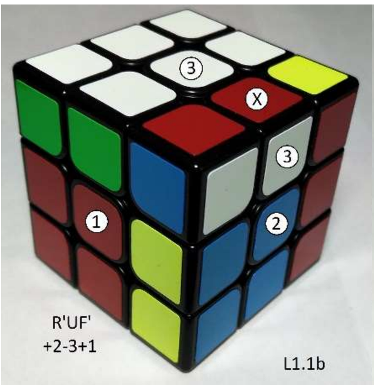
Two useful Alg: (1)L3 2 corner pieces Rot(x3)@V123:
L_turn: LH:DLU RH:DLU -1+6+1 +2+6-2
R_turn: RH:DRU LH:DRU +2-6-2 -1-6+1



(2)L3 2 corner pieces A,B Rot(x3)@V123 with C@V126(CABC):
A@V123: RH:DLU B@V123: RH:DRU +2+6-2 +2-6-2 C1A3B31C Bu=2
A@V123: LH:DRU B@V123: LH:DLU -1-6+1 -1+6+1 C2A3B32C Bu=1

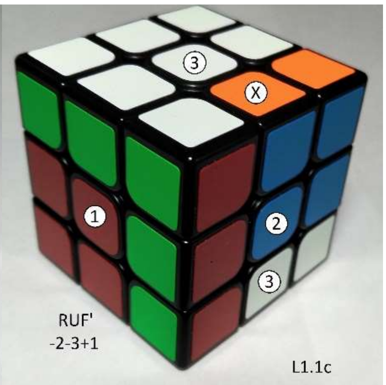


L1.1a



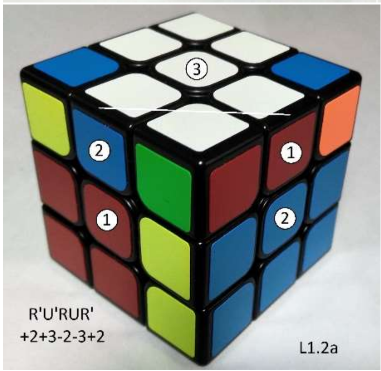
R'UF'
+2-3+1

L1.1b



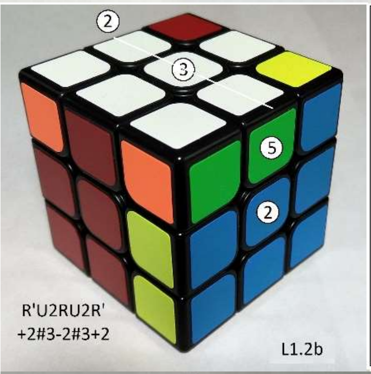
RUF'
-2-3+1

L1.1c



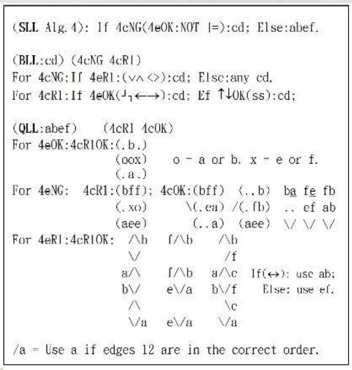
R'U'RUR'
+2+3-2-3+2

L1.2a

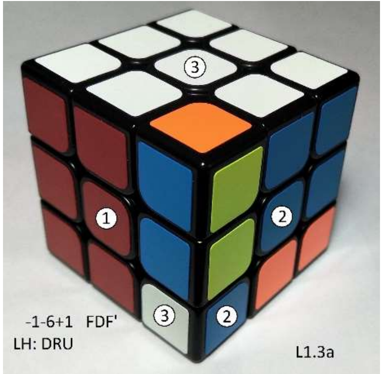


R'U2RU2R'
+2#3-2#3+2

L1.2b

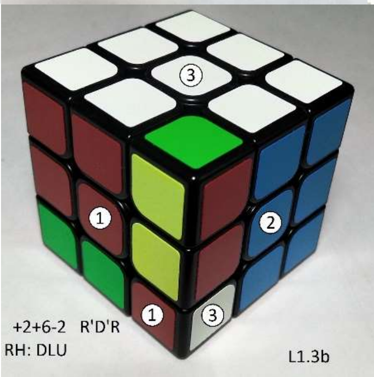


```
(SLL Alg.4): If 4cNG(4eOK:NOT |=):cd; Else:abef.

(BLL:cd) (4cNG 4cR1)
For 4cNG:If 4eR1:(∨∧<>):cd; Else:any cd.
For 4cR1:If 4eOK(┘┐←→):cd; Ef ↑↓OK(ss):cd;

(QLL:abef)    (4cR1 4cOK)
For 4eOK:4cR1OK:(.b.)
                (oox)    o = a or b. x = e or f.
                (.a.)
For 4eNG:  4cR1:(bff); 4cOK:(bff)  (..b)  ba fe fb
                (.xo)       \(.ea) /(.fb)  .. ef ab
                (aee)         (..a)  (aee)  \/ \/ \/
For 4eR1:4cR1OK:  /\b   f/\b    /\b
                  \/             /f
                 a/\    f/\b   a/\c  If(↔): use ab;
                 b\/    e\/a   b\/f    Else: use ef.
                  /\             \c
                  \/a   e\/a    \/a

/a = Use a if edges 12 are in the correct order.
```



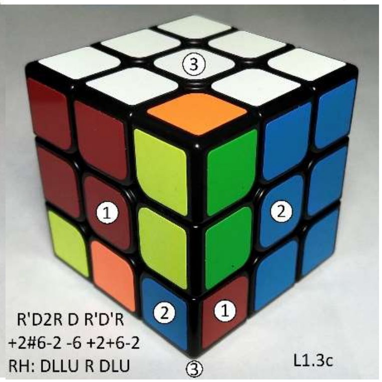
-1-6+1 FDF'
LH: DRU

L1.3a



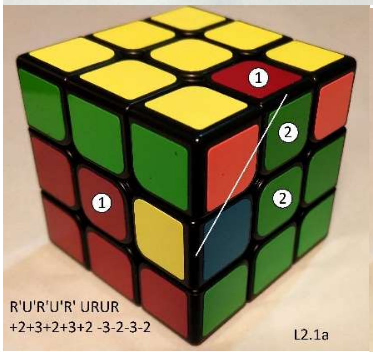
+2+6-2 R'D'R
RH: DLU

L1.3b



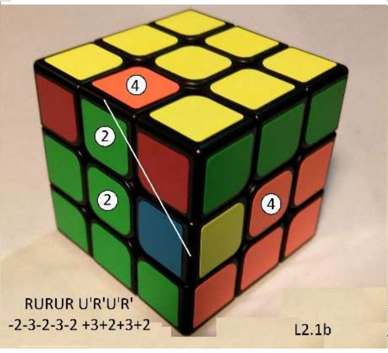
R'D2R D R'D'R
+2#6-2 -6 +2+6-2
RH: DLLU R DLU

L1.3c



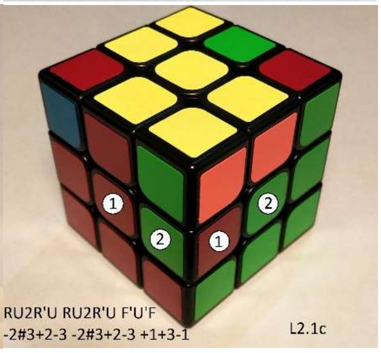
R'U'R'U'R' URUR
+2+3+2+3+2 -3-2-3-2

L2.1a



RURUR U'R'U'R'
-2-3-2-3-2 +3+2+3+2

L2.1b



RU2R'U RU2R'U F'U'F
-2#3+2-3 -2#3+2-3 +1+3-1

L2.1c

Both 2D and 3D pictures are shown for most algorithms.

Step L3.1: (Finish face 3 colors of 4 edges)
Target: 2 correct edges marked with 3 on top face3.
Relocate: For 2OKe: Rot(xC3) such that color 3 is in location (@LB or @BF or @LF) as shown.

```
Algorithm:   2OKe:@LB: 231312      2OKe:@BF: 213132      2OKe:@LF: 234342
            L3.1a:@LB: +++---     L3.1b:@BF: +++---     L3.1c:@LF: ---+++
```

(**SLL** Alg.2): Count the number N for faces marked as “o” with color 3 on them.
Choose Alg 1a,1b,1c base on N in the order (4,1,0,2).

For 0OKe: Need 2 steps: Step 1 uses SLL Alg.1 below. Step 2 uses SLL Alg.2 above.

(**SLL** Alg.1): The Simple Last Layer Alg uses 8 cases from the modified OLL Alg.
Use L3.1a for 1OKc:@(3+) or 1OKc:@(2-)
Use L3.1b for 2OKc:@(<,<,/) or 4OKc:@(X) or 0OKc:@(ii)

The Simple Last Layer Alg.1 (SLL Alg.1) uses 8 cases from the modified OLL Alg(Fig.4):
3 cases for 0OKe inside 2 regions bounded by solid lines (First row:2,3,8):
1a(1OKc:3+,2-),1b(0OKc||)=1b(0OKc:ii)
5 cases for 0OKe inside the M-region bounded by dashed lines (Last row):
1b(2OKc:<,<,/),1b(4OKc:X),1b(0OKc:ii)

Step L3.2:(Finish face 3 colors of 4 corners) (3c Spin plus 3e Rot(2abef) or 3c Rot(2cd))
Target: NG corners marked with 3 on the side faces. At first, consider 3NGc as follows:
Relocate: (3cSpin)@(3-): If 3NGc need (-) Spin, put 1OKc @3: Use L3.2bdf.
(3cSpin)@(4+): If 3NGc need (+) Spin, put 1OKc @4: Use L3.2ace.

```
Algorithm:(3eRot) 3 23232323    (3cRot)  x 23232323 x    (3eRot)  32323232 3 (Last3 may omit)
 @(4+) L3.2a(1+): - ++-++#-#  L3.2c(4-): - ++-++--- +  L3.2e(4+): #-#++-++ - (=L3.2bR)
 @(3-) L3.2b(3-): + --+--#+#  L3.2d(3+): - --+--+++ +  L3.2f(4-): #+#--+-- + (=L3.2aR)
```

Each location has 3 Alg to choose base on the rules below:

(**SLL** Alg.3): If ccOK, use L3.2cd. Ef ssNG, use L3.2ab. Ef 4eOK, use L3.2cd. Else use L3.2ef.
Note: ccOK = 2 corner faces marked with “c” have the same color.
ssNG = 2 edges marked with “s” are NOT in the correct order.
4eOK = 4 edges are in the correct order.

2NGc/4NGc: Need 2 steps: Step 1 uses SLL Alg.4 below. Step 2 uses SLL Alg.3 above.

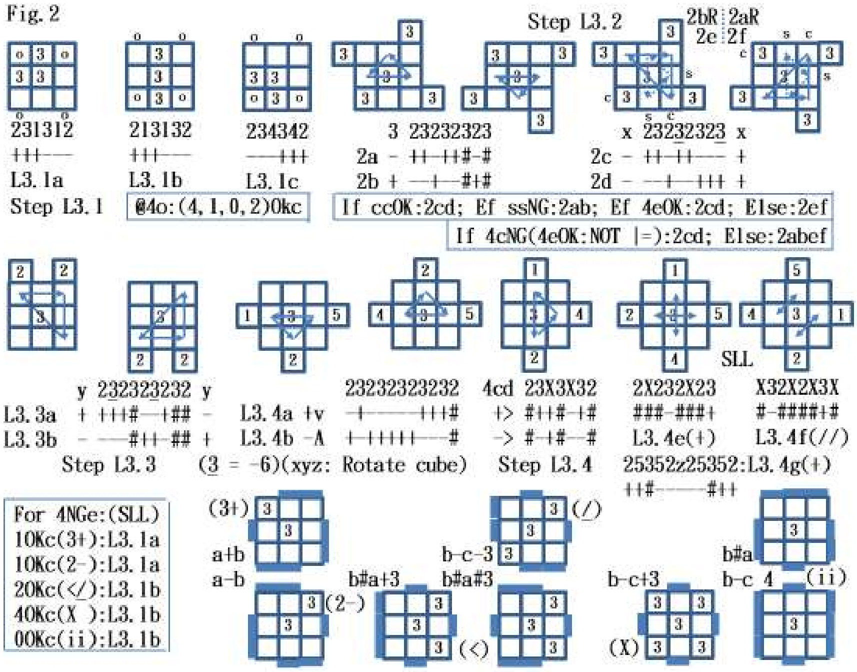
Fig.2 gives the (+-) Rot of 3c/3e by the Alg 2cd/2abef on the piece for step 1.
Location for L3.2cae: 4NGc@FF:LBF/2NGc@LFL.   Location for L3.2dbf: 4NGc@BB:LBF/2NGc@LBL.
Remember this (or by picture) for step 1 to obtain 3NGc, such that step 2 can use SLL Alg.3.

Use (4cOK:4cR1:4cNG) to indicate the order of 4 corners (correct:Rot-1:Reversed).
This order can be checked by: Rot (x3) for any corner to the correct position.
And check whether the opposite corner is at the correct position: If not, it is 4cR1.
Else check any other corner the same thing: If not, then it is 4cNG, else 4cOK.

(**SLL** Alg.4): If 4cNG: use any cd (but NOT use |= for 4eOK); Else use any abef.

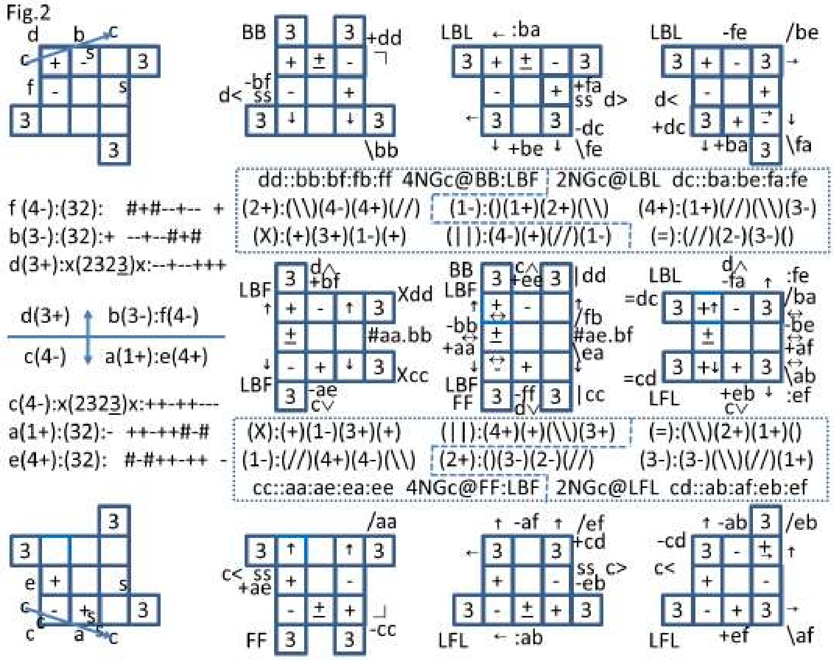
```
(BLL:cd) (4cNG 4cR1)
For 4cNG:If 4eR1:(∨∧<>): use cd; Else: use any cd.
For 4cR1:If 4eOK(┘┐←→): use cd; Ef ↑↓OK(ss): use cd; (Else: use any abef.)

(QLL:abef)   (4cR1 4cOK)
For 4eOK:4cR1OK:(.b.)
                (oox)    o = a or b, x = e or f.
                (.a.)
For 4eNG:  4cR1:(bff); 4cOK:(bff)  (..b)   ba fe fb
                (.xo)      \(.ea) /(.fb)   .. ef ab
                (aee)       (..a)  (aee)   \/ \/ \/
For 4eR1:4cR1OK:  /\b   f/\b    /\b
                  \/             /f
                 a/\    f/\b   a/\e    If(↔): use ab;
                 b\/    e\/a   b\/f      Else: use ef.
                  /\             \e
                  \/a   e\/a    \/a

Note: /a = Use a if edges 12 are in the correct order.
      ↑↓OK,←,→ = Color matched at ↑↓←→.   ↔ = Color matched at 3 ↔.
      ss,<,>,∨,∧ = Edge nearby ss,<,>,∨,∧ must be one of the 2 edges in the correct order.
      ┘,┐ = (Edge,Corner,Edge) must be in the correct order.

Step L3.3: (Finish L3 side colors of 4 corners) (Only 3 corners Rot(4-)(3+))
Target: 2 corners with color 2 on the same side. Otherwise, do any 3a,3b is OK.
Relocate: Rot(xC3) such that: color 2 face to Back or color 2 face to Front.
          For both cases, color 2 will turn to the Right face 2 after applying the Alg.
          If possible, the edge color on face 2 is also color 2.
Algorithm: 3c(-)Rot@c4  y 232323232 y      3c(+)Rot@c3  y 232323232 y
           L3.3a:(4-):  + +++#--+## -      L3.3b:(3+):  - ---#++-## +

Step L3.4：(Finish L3 side colors of 4 edges) (Only 3 edges Rot(3+)(1-))
Target: 1OK edge with the same color as 2 corners. Otherwise, use L3.4e(+) or L3.4f(//).
Relocate: 1OKe Rot(x3) to Back (3+)(+v) for other 3e (+) Rot, otherwise to Front (1-)(-A).
Algorithm: 3e(+)Rot@e3     23232323232     3e(-)Rot@e1     23232323232
           L3.4a:(3+)(+v): -+-----+++#     L3.4b:(1-)(-A): +-+++++---#

May use 4 short Algorithms. Where X is the middle layer faces to face_1 (hard to rotate).
 3eRot@e4        23X3X32     2X232X23     X32X2X3X     25352 z 25352  (4cdg may replace 4abe)
 L3.4c:(4+)(+>): #++#-+#     ###-###+     #-####+#     ++#-- - --#++  (SLL adds 2 Alg : 4ef)
 L3.4d:(4-)(->): #-+#--#    L3.4e:(+)    L3.4f:(//)      L3.4g:(+)  (4efg = 2 steps of 4abcd)
```

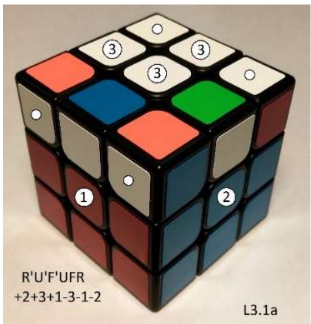
Fig. 2



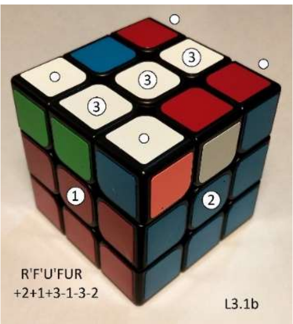
SLL Alg adds 4 check locations as shown inside solid lines.

Fig.2

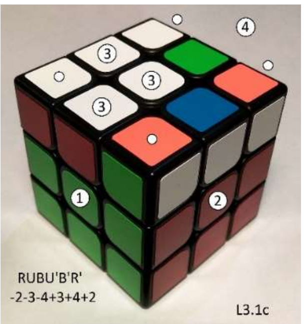




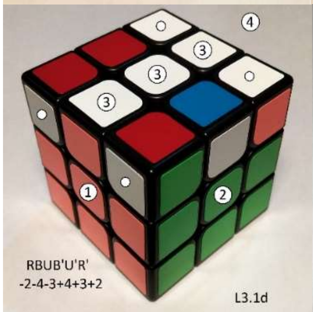
R'U'F'UFR
+2+3+1-3-1-2
L3.1a



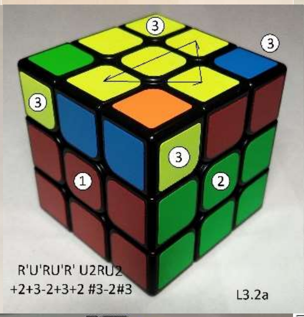
R'F'U'FUR
+2+1+3-1-3-2
L3.1b



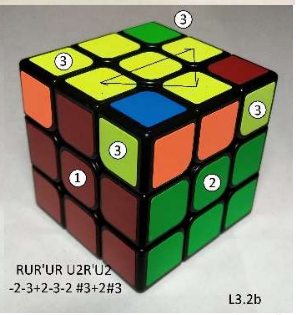
RUBU'B'R'
-2-3-4+3+4+2
L3.1c



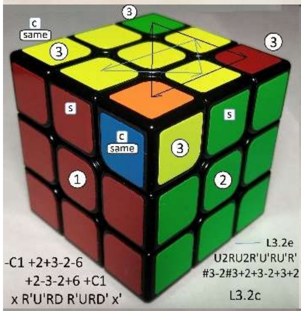
RBUB'U'R'
-2-4-3+4+3+2
L3.1d



R'U'RU'R' U2RU2
+2+3-2+3+2 #3-2#3
L3.2a



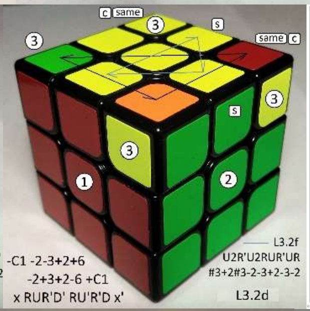
RUR'UR U2R'U2
-2-3+2-3-2 #3+2#3
L3.2b



-C1 +2+3-2-6
+2-3-2+6 +C1
x R'U'RD R'URD' x'

L3.2e
U2RU2R'U'RU'R'
#3-2#3+2+3-2+3+2

L3.2c



-C1 -2-3+2+6
-2+3+2-6 +C1
x RUR'D' RU'R'D x'

L3.2f
U2R'U2RUR'UR
#3+2#3-2-3+2-3-2

L3.2d

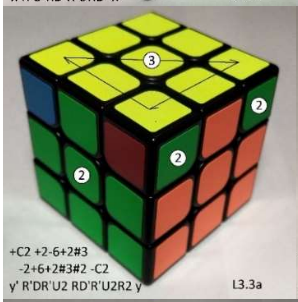
Simple Last Layer (SLL)

For L3.1 4NG-edges:
Use L3.1a for 1OK-corner @(3+) or @(2-).
Use L3.1b for (0,2,4)OK-corner:
0OKc:@(ii) or 2OKc:@(<∠) or 4OKc:@(X).

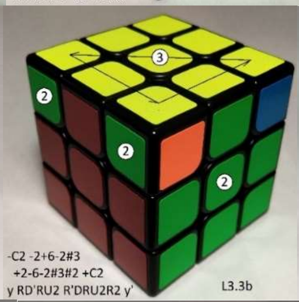
For L3.1 2NG-edges: @4o:(4,1,0,2)OKc.

For L3.2 3NG-corners:
If ccOK use L3.2cd. Ef ssNG use L3.2ab.
Ef 4eOK use L3.2cd. Else use L3.2ef.

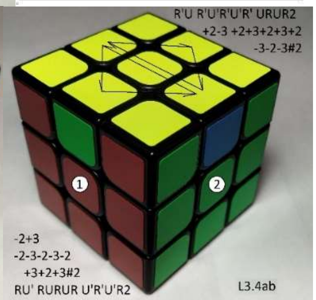
For L3.2 2NG/4NG-corners:
If 4cNG(4eOK:X |=):L3.2cd; Else:L3.2abef.
Use L3.2dbf(-): 4NGc@BB:LBF / 2NGc:LBL
Use L3.2cae(+): 4NGc@FF:LBF / 2NGc:LFL



+C2 +2-6+2#3
-2+6+2#3#2 -C2
y' R'DR'U2 RD'R'U2R2 y
L3.3a

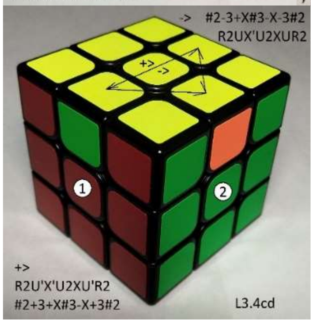


-C2 -2+6-2#3
+2-6-2#3#2 +C2
y RD'RU2 R'DRU2R2 y'
L3.3b



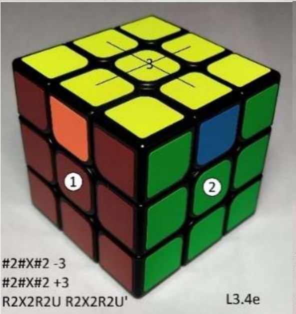
R'U R'U'R'U'R' URUR2
+2-3 +2+3+2+3+2
-3-2-3#2

-2+3
-2-3-2-3-2
+3+2+3#2
RU' RURUR U'R'U'R2
L3.4ab

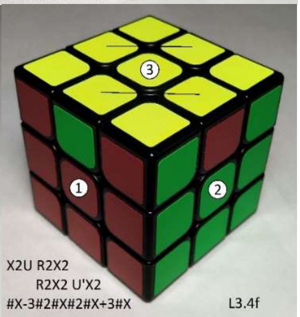


-> #2-3+X#3-X-3#2
R2UX'U2XUR2

+>
R2U'X'U2XU'R2
#2+3+X#3-X+3#2
L3.4cd



#2#X#2 -3
#2#X#2 +3
R2X2R2U R2X2R2U'
L3.4e



X2U R2X2
R2X2 U'X2
#X-3#2#X#2#X+3#X
L3.4f

Both 2D and 3D pictures are shown for most algorithms.
But the locations and the moves (+-) for 4NGc/2NGc are shown only in 2D.

F2L Algorithm (A fast Alg for a competitor) (Seems complicated, Actually easy) (See Fig.3)

F2L Alg combines an Edge piece and a Corner piece (E & C) and puts it into a Slot. Identify 2 faces of Edge as (EU,ES) for (Up,Side) faces. Identify 3 faces of the Corner as (CU,CS,C6) for they have the same color as (EU,ES,face6). Identify 2 faces of the cube as (FU,FS) for their center color (Fc:=Face-color) the same as (EU,ES). Identify the Slot location as (s).

(1) E3C1：Edge piece at Layer3 (E@L3), Corner piece at Layer1 (C@L1):
(See Up-Left 3 Cases in Fig.3, it is better to do these cases first)
(a) C6 at Down-face: Rot-face is CU. Rot x3 for ES to the opposite side of Rot-face CU.
(b) C6 & CU at Side-face: Rot-face is CU. Rotate x3 for ES to Rot-face CU.
(c) C6 & CS at Side-face: Rot-face is C6. Rotate x3 for ES to Rot-face C6
Rot cube about face3 (xC3) for Rot-face to Right. (Fig.3 shows the E & C Location)

Cases (a) & (b) use the same Alg (+2-3-2) to joint E & C (EjC).
Case (c) use Alg (+2+3-2) to disjoint E & C in good position (E%C).
The Alg (+2-3-2) (+2+3-2) is called a basic Alg. Each basic Alg has 3 steps:
The first step is to rotate about Rot-face (always on the Right side) (+2, -2, or +-2).
The second step is to rotate about face 3 (+3,-3, #3, or x3).
The last step is the reverse of the first step (-2, +2, or -+2).

After the basic Alg, both E & C move to Layer 3 and use the method for E3C3 as follows.
(2) E3C3：Both Edge & Corner pieces at Layer3: based on the required number of basic Alg:

(a) Need 1 basic Alg: As shown in Up Right 2 cases:
1. Edge joint to Corner (EjC): Basic Alg (+2-3-2).
2. Edge disjoint to Corner (E%C): Basic Alg (+2+3-2).
Fig.3 shows the locations of the pairs and the basic Alg to be used. After doing the 3 steps, the (Edge, Corner) pair will be in the Slot location. It must do first; otherwise, the pair may be destroyed.
(b) Need 2 or 3 base rotations: As shown in Lower 6 cases[1-6] (need 2 basic Alg) and case[0] in Center-Right (need 3 basic Alg).
Case[0] needs 3 basic Alg all the same Alg(+2+3-2) with 2 x3(#3,-3) in between. The Alg is easy to remember as ++-# ++-- ++- or --+# --++ --+.
Cases[1-6] need to know only the first basic Alg. Since after the first basic Alg, the second basic Alg is obvious and can follow item (a).

Following steps are easy and important to put Edge and Corner in place and identify the Slot location for the 3rd step (-2 or +2) to rot Slot Up-ward. Each Case is numbered as [0-6].

Step 1: xC3 to put Edge or Corner to Right:
If CE connected & CU@end：rot CU to Right[06]. {C6@top is [0]; C6@side is [6].}
Else: if C6@top: rot ES to Right[12]. {CE disconnect is [1]; CE connect is [2].}
If C6@side: rot C6 to Right[345]. {ES to R[3]; CU@top[4]; CU@side[5].}
Mark as **1: xC3：{(E|C & CU@end)：CU->R}06 {C6@top：ES->R}12 {C6@side：C6->R}345**

Step 2: xL12 to fix Side rot face: (And mark the slot location as (s) as shown in Fig.3)
If [01246]: rot L12 for FS to R. Slot location is between FU & FS.
If [35]: rot L12 for FU to R. Slot location is between FU & FS.

As a check: If [023]: Slot under C; If [456]: Slot not under C; If [123]: ES->R.
Mark as **2: x12：{[01246]：FS->R}　{[35]：FU->R}**
check：{[023]：slot@C}　{[456]：slot@C̲}　{[123]：ES->R}

The above 2 steps only need one rot3. After these steps, the locations of (Edge,Corner,Slot) for all Cases[0-6] are shown in Fig.3. The Case numbers have been arranged carefully for easy remembering. They can be modified if you want. The next 3 steps are as simple as the first 3 steps (a basic Alg) of the complete Alg (two basic Alg) underneath:

1. Rot Slot about Side-face(R:2) Up-ward (+-2).
2. Rot x3 following the 4th step below (x3).
3. Rot Slot about Side-face(R:2) Down-ward (-+2).

These 3 steps will arrange Edge and Corner to a good position (EjC or E%C) such that the next basic Alg can follow item (a). Therefore, no need to remember any long Alg and ignore the explanations for steps 6-10.

Step 3: As simple as rot Slot Up-ward about Right-face (+2 or -2).

Step 4: rot about Up-face (x3)=(+3 or -3 or #3).

```
    If C is still at Laye3 [0123]: If E at Layer3: rot x3 for C6 to Right [0]
                                   If E at Layer2: If C6@top: rot x3 for C to joint E [1].
                                                   Else: rot x3 for C6 to F or B face [23].
    If C rot to Layer1 [456]: If C6 at Down-face:  rot x3 for ES to Left [6].
                              If C6 on Right-side: rot x3 for ES to F or B face [45].
                                           Ct=EU: ES to the same side of C (EjC) [4].
                                           Ct=ES: ES to the opposite side of C (E%C) [5].
    Where Ct:= top side color of Corner when at Layer3.
```

Mark as **4: C@L3：{E@L3：C6->R}0　{E@L2：{C6@top：EjC}1　{else：C6->FB}23}123**
C@L1：{C6d：E->L}6　{C6r：E->FB}45　{Ct=EU：EjC}4　{Ct=ES：E%C}5

Step 5: As simple as rot Slot Down-ward about Right-face (-2 or +2).

Step 2 may put the Slot location in (), but be sure that () has not been finished yet.

Steps 6-10 can ignore since they can be following item (a).

Step 6: If EjC: rot x3 for EU->FU [14].
If E%C: rot x3 for C->Slot [2356].

Step 7: rot Cube about face3 (xC3) for C6 to Right [35].
Mark as **6: {EjC：[x3：EU->FU]14}　{E%C：[x3：C->st]2356 + 7: [xC3：C6->R]35}**

Step 8: rot Slot Up-ward (+2 or -2). (Say: Open door)

Step 9: rot x3 for the (Edge, Corner) pair to Right (x3). (Say: Get on)

Step 10: rot Slot Down-ward (-2 or +2). (Say: Close door)

Important hint:

Remember Step 1: The sequence & logic for the decision of Case numbers: [06] [12] [345].

Remember Step 2: For [023]: The slot is under C; For [456]: The slot is not under C.
For [123]: ES is always to Right.

Fig.3 shows (Edge, Corner) before Step 3 and after Step 5 (inside ()) and the Slot location (marked as (s)). I hope that Fig.3 is a complete guide for the F2L Algorithm.
F2L Algorithm does not need to remember long Alg and uses little steps. Therefore, it is the best choice for cube competitors.

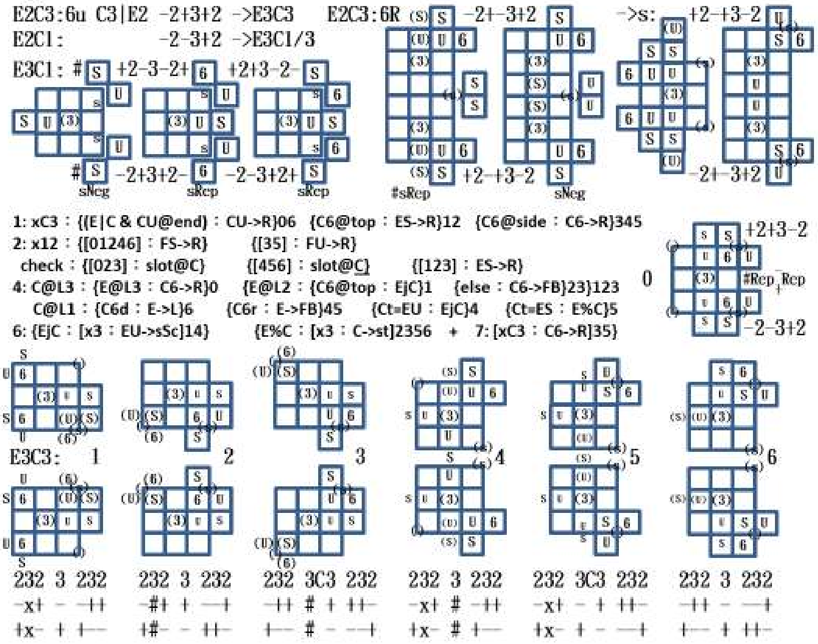
(3) E2C3 or E2C1: Edge piece at L2 and Corner piece at L3/L1. (do it last, it may not appear)

(a) Corner at Layer3 and C6 at Side-face:
See Upper-Middle 2 Cases for the locations of (Edge, Corner) and the basic Alg.
Where 6R:= rot x3 for C6 to Right.

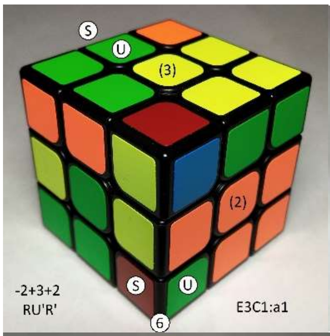
(b) Corner at Layer3 and C6 at top-face:
Rot x3 for Corner to join Edge & move (EjC) to L3. Then use the method for E3C3.

(c) Corner at Layer1: Move Edge to Layer3. Then use the method for E3C1 (or E3C3).

Fig. 3

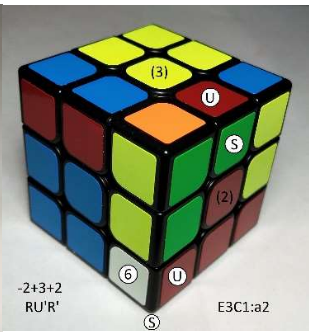


```
1: xC3 : {(E|C & CU@end) : CU->R}06   {C6@top : ES->R}12   {C6@side : C6->R}345
2: x12 : {[01246] : FS->R}        {[35] : FU->R}
 check : {[023] : slot@C}        {[456] : slot@C}        {[123] : ES->R}
4: C@L3 : {E@L3 : C6->R}0     {E@L2 : {C6@top : EjC}1     {else : C6->FB}23}123
   C@L1 : {C6d : E->L}6      {C6r : E->FB}45      {Ct=EU : EjC}4      {Ct=ES : E%C}5
6: {EjC : [x3 : EU->sSc]14}          {E%C : [x3 : C->st]2356   +   7: [xC3 : C6->R]35}
```



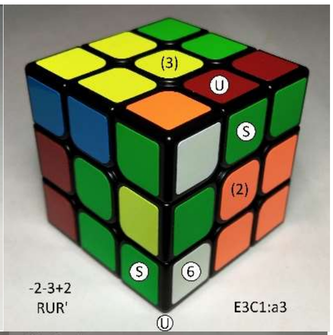
-2+3+2
RU'R'

E3C1:a1



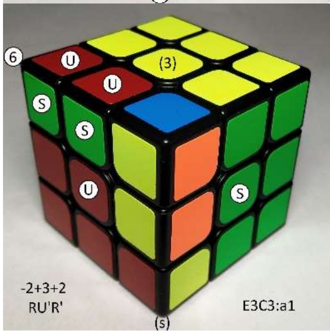
-2+3+2
RU'R'

E3C1:a2



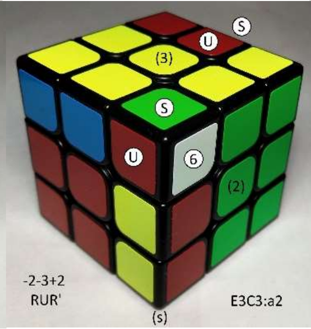
-2-3+2
RUR'

E3C1:a3



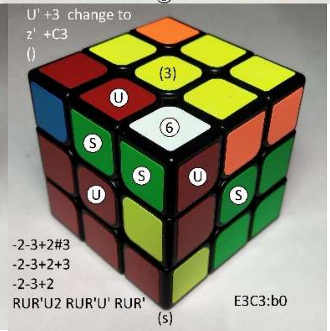
-2+3+2
RU'R'

E3C3:a1



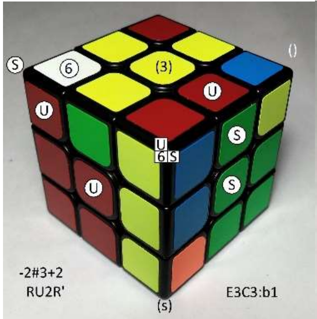
-2-3+2
RUR'

E3C3:a2


U' +3 change to
z' +C3
()

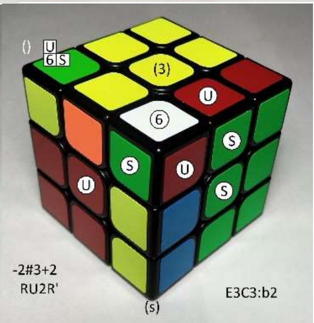
-2-3+2#3
-2-3+2+3
-2-3+2
RUR'U2 RUR'U' RUR'

E3C3:b0



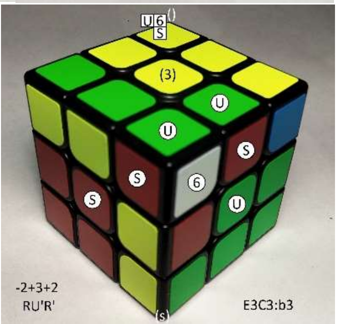
-2#3+2
RU2R'

E3C3:b1



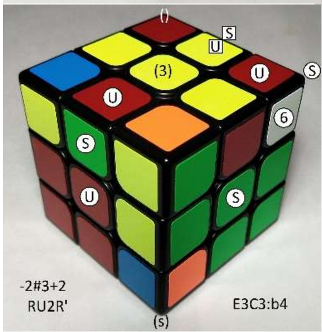
-2#3+2
RU2R'

E3C3:b2



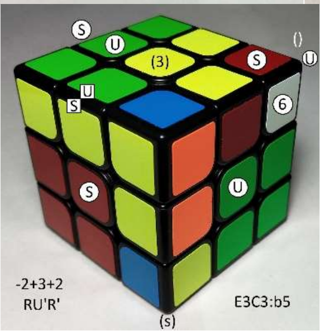
-2+3+2
RU'R'

E3C3:b3



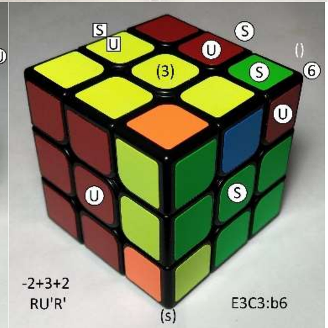
-2#3+2
RU2R'

E3C3:b4



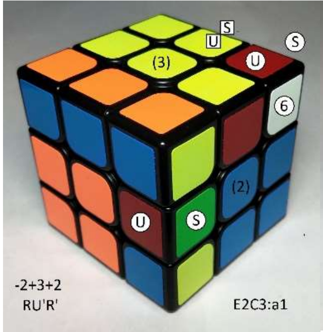
-2+3+2
RU'R'

E3C3:b5



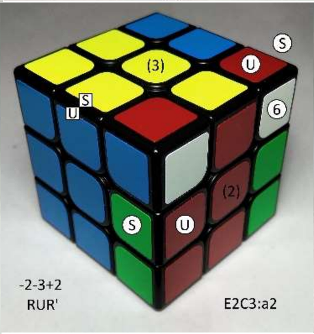
-2+3+2
RU'R'

E3C3:b6



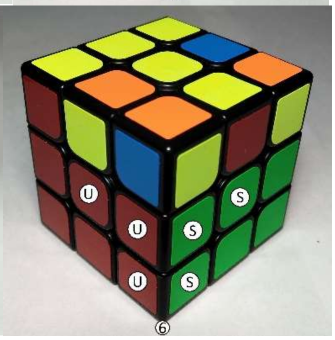
-2+3+2
RU'R'

E2C3:a1



-2-3+2
RUR'

E2C3:a2



Modified OLL algorithm for easy remembering

The original OLL algorithm requires 57 Alg, which is hard to remember. The modified OLL Alg (mOLL) requires only 8 new Alg and can be completed by 2 Alg at most. (4 for Basic Alg)

1. Fifteen cases inside 3 regions bounded by solid lines can be completed by 2 Alg of L3.1. Each region uses the same first Alg (A,B,D) for easy memory. After doing the first one, the second Alg is obvious, and there is no need to remember.

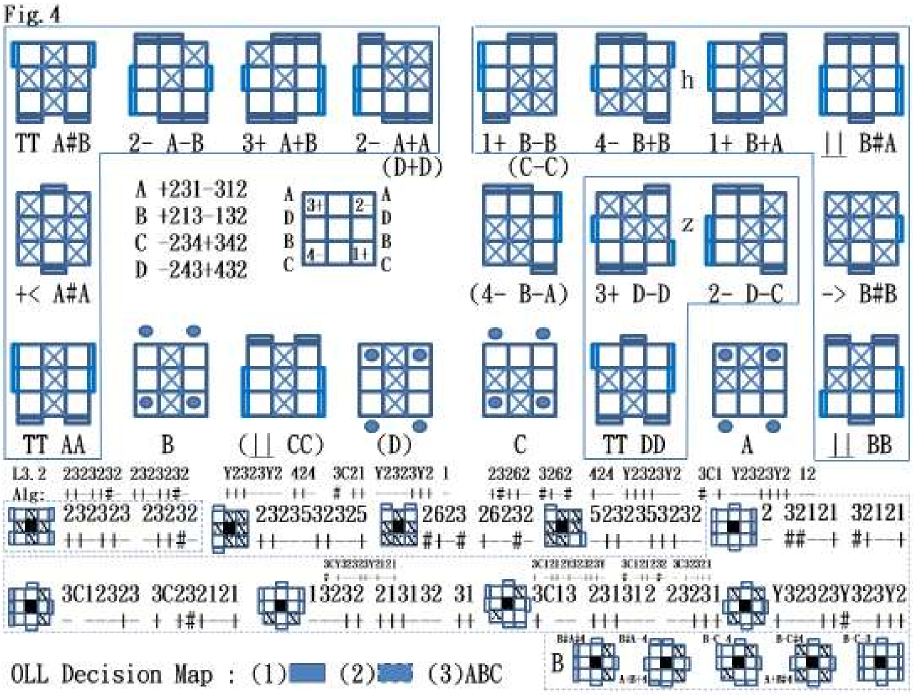
2. Five cases inside a Large region and a Middle region by dashed lines have two choices: Use one New Alg in L-region or 3 old Alg (two L3.1 plus one L3.2) in M-region. The first Alg in M-region is the same B for easy memory, and the next 2 Alg are obvious.

3. The other cases left only need one Alg to complete L3.1. Before doing the Alg (A,B,C,D), check the 4-positions marked with “o” , count +1 if its-color is face 3 color. Then choose the case (one out of two) by the count numbers (N) in the order 4,1,0,2.

   (a) If N=4: one L3.1 Alg will complete mOLL.
   (b) If N=1: one L3.1 Alg plus one L3.2 Alg will complete mOLL.
   (c) If N=0: two identical L3.2 Alg are used with 3 rot canceled.
   (d) If N=2: Use 3 New Alg as shown in Fig.4. Two Alg are provided for each case.
       It is better to use large-size Alg.

If you give up on remembering 8 (5+3) new Alg, the total number of Alg to complete mOLL Alg is 3 instead of 2.

Fig. 4



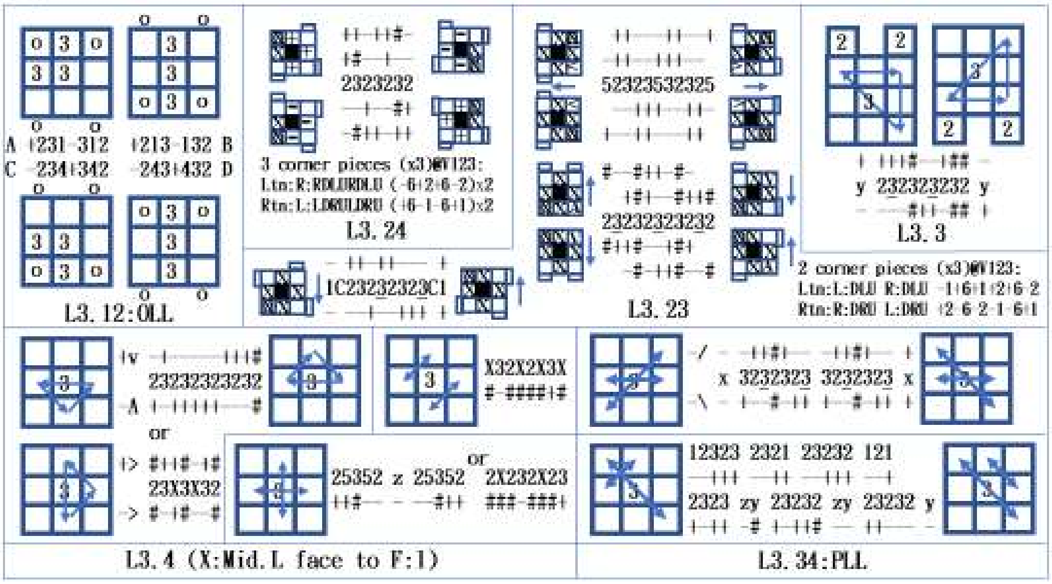
Modified PLL algorithm for easy remembering

The original PLL algorithm requires 21 (4 old, 17 new) Alg, which are hard to remember. The modified PLL Alg (mPLL) requires only 5 new Alg and can be completed by 2 Alg at most.

1. If L3.3 is already completed or only one L3.3 is needed (10 out of 21), complete L3.3 and then L3.4 by one of (two new & one old) Alg as shown in Lower-Left of Fig.5.
2. Otherwise, check the 4 sides of Layer 3, and count +1 if the corner piece has the same color as the edge piece. The total number (N) will be 4,2 or 0.
   (a) If N=4 (fan:-/,-\) or N=2 (Arrow or Kite), use one of four new Alg as shown in Lower-Right of Fig.5 to complete mPLL.
   (b) If N=0, check whether the 4 edges are in the correct sequence; if so, Rot(x3) to put them in place, else do nothing. Then, use two L3.3 to complete mPLL. Otherwise, repeat the process from step 1.

Special pattern (for fun):
(+Y-Z-Y+Z)=(+Z-X-Z+X)=(+X-Y-X+Y):make +c123 & -c456
Rotate centerpiece on up/right face 90-degree ccw/cw:
(+2+3+2+3+2-3-2-3-2-3) 3 times
Supper-flip:
(-Y-3-Y-3-Y-3-Y-3-x-z) 3 times or (-2-5#3 -1+3 -6#1#2#4-5#3 +1+4-3 #2-6#1-3 #2-3)

Fig.5



Compare the number of steps for SLL, mOLL+mPLL, OLL+PLL

SLL guarantees the number of the Alg used is 5 or less, and 4 in most cases.
For L3.1+L3.2+L3.34 the number of the Alg used is estimated as:
0OKe:2+0+3,2+1+2, 2OKe:1+0+3,1+1+2,1+2+2, 4OKe:0+0+3,0+1+2,0+2+2.
Although the best Alg can be decided based on the results shown in Fig.2 inside the regions bounded by dashed lines. The combined cases are too many to remember. SLL does not use it.
But with the additional **BLL** (Better Last Layer) checks, it may catch some better cases.
Use **QLL** (Quick Last Layer) checks, it may further reduce the numbers of the Alg used.

Compare the 3 Alg for the last layer with the basic Alg:

Basic Alg	At most 8 Alg. (2 for each step) (For L3.34: 4 basic Alg: 3-,1+,+v,-A)
Basic + 2 new Alg	At most 7 Alg. Add 0+2 Alg.(Add 2: +,\\)(Reduce L3.4 to <= 1 Alg)
SLL	At most 5 Alg. Add 0+2 Alg.(Add 2: +,\\)(Easy check reduce 2 Alg)
mOLL+mPLL	At most 4 Alg. Add 8+6 Alg.(Add 6: +,\\,Arrow,Kite,-/,-\)
OLL+PLL	At most 3.5 Alg.(2 Long-Alg) Add 57+17 Alg.(Add 6+11 for L3.34)

The numbers of added Alg for L3.12 are (0, 0, 8, 57) for (Basic+2, SLL, mOLL+mPLL, OLL+PLL).
The numbers of added Alg for L3.34 are (2, 2, 6, 17) for (Basic+2, SLL, mOLL+mPLL, OLL+PLL).
Only 6 added Alg are shown as above.

OLL Long_Alg At most 14 turns, most >=11, Equal to 1.5~2 (Basic)Alg. (6~8 turns)
PLL Long_Alg At most 17 turns, most >=13, Equal to 1.5~2 (Basic)Alg. (7~9 turns)
Therefore, 2 long Alg in OLL+PLL is counted as 3.5 (Basic)Alg.

SLL only adds 2 Alg and guarantees the number of Alg used is 5 or less (4 in most cases).
So, it is recommended to use SLL (Simple Last Layer) to replace OLL, and no need to use PLL.
It is also better to use BLL (Better Last Layer) with some additional simple checks.
QLL checks are not as simple as BLL checks, so it is not highly recommended.
QLL checks for 4eOK and 4eNG@4cR1 are not so hard to apply and it is worth a try.