Rubik examples

www.ctan.org/tex-archives/macros/latex/contrib/rubik/rubikexamples.pdf [*]

RWD Nickalls [†] & A Syropoulos [‡]

03 March 2017 (v4.0)

1 Preliminaries

These examples were generated using the TEX Rubik bundle [1] v4.0. They assume some familiarity with the three complementary packages RUBIKCUBE, RUBIKROTATION and RUBIKPATTERNS. For documentation see the files:

```
rubikcube.pdf
rubikrotation.pdf
rubikpatterns.pdf
rubikpatternsLIST.pdf
```

This file requires the following packages: `tikz, rubikcube, rubikrotation, rubikpatterns`; note that the `tikz` package must be loaded *before* the `rubikcube` package.

This file needs to be run using the `--shell-escape` command-line option; for example:

```
pdflatex  --shell-escape  rubikexample.tex
```

This is because nearly all the examples make use of the `\RubikRotation` command, which calls the Perl script `rubikrotation.pl`. If you do forget to use the command-line switch, the file will still run, but all the cubes will remain in the initial unprocessed configuration.

1.1 Environments

When using the Rubik bundle one sometimes needs to be mindful of the various LATEX environments in which Rubik commands are placed (e.g., the figure, minipage and TikZ picture environments), since these environments restrict the actions of commands they contain to the particular environment. The `\ShowCube` command is also relevant here, since it is a minipage-wrapper for the TikZ picture environment. Only Rubik `\Draw..` commands actually need to be inside a TikZ picture environment.

This issue arises because the Rubik bundle allows you to create figures showing different stages during a sequence of rotations. Consequently the effects of commands executed inside an environment (especially commands which determine the colour-state or rotations), may not be apparent to subsequent commands outside that particular environment. See Example 3 for an illustration of how to handle environments.

[*] This file is part of the Rubik bundle. To generate this file, use the following command:

```
$ pdflatex --shell-escape rubikexamples.tex
```

[†] email: dick@nickalls.org

[‡] email: asyropoulos@yahoo.com

[1] `http://www.ctan.org/pkg/rubik`

2 Examples

2.1 Sixspot

In Figure 1 we show the so-called "sixspot" configuration, generated from a solved cube using the rotation sequence **U**, **D'**, **R**, **L'**, **F**, **B'**, **U**, **D'**.

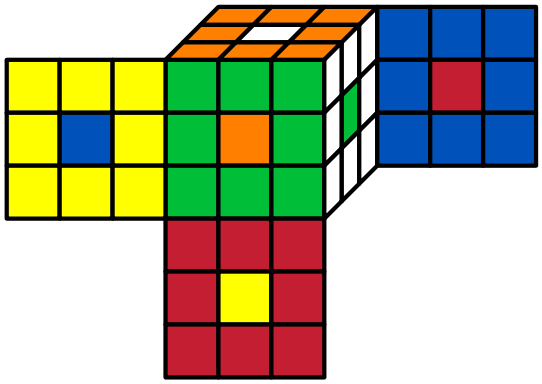


Figure 1: The 'sixspot' configuration.

Creating a macro to hold a rotation sequence greatly facilitates their use, as follows:

```
\newcommand{\sixspot}{[sixspot],U,Dp,R,Lp,F,Bp,U,Dp}
```

We can now process this sequence using its macro name as an argument for the `\RubikRotation` command. The code used for the above Figure uses the `\ShowCube{}{}{}` command for which `#1` is the minipage width, `#2` is the tikzpicture scale factor (0–1), and `#3` can include RUBIKCUBE package `\Draw..` commands, and any commands which are valid for use in a TikZ `tikzpicture` environment. The code for the above figure is as follows:

```
\begin{figure}[hbt]
   \centering
   \RubikCubeSolved
   \RubikRotation{\sixspot}
   \ShowCube{7cm}{0.7}{\DrawRubikCubeSF}
\caption{...}
\end{figure}
```

Note that the sixspot sequence is a so-called 'order 3' sequence, which means that running the 'sixspot' sequence 3 times returns the cube to its original 'solved' state. The command for processing it three times is `\RubikRotation[3]{\sixspot}`.

Note that the semi-flat form of the cube here is generated by the `\DrawRubikCubeSF` command, where the terminal SF denotes the Semi-Flat form.

2.1.1 Log-file extract

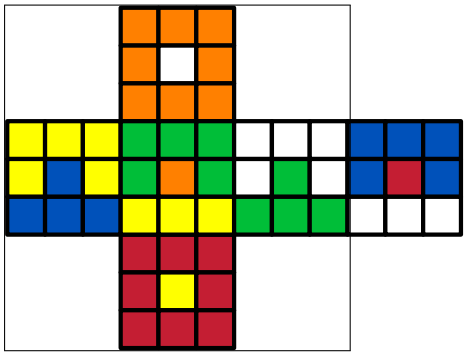
Users may find it instructive to inspect the the log-file and follow the dynamic interaction between LaTeX and the Perl script. This is easy to follow, since output by `rubikrotation.sty` is prefixed by 3 dashes (`---`), while output by the Perl script is prefixed by 3 dots (`...`). Search for the keyword 'Example', as this is written to the log-file at the start of each example.

The following is the log-file extract associated with Example 1 (from the author's Debian Linux platform).

```
---Example (sixspot)
LaTeX Font Info:    Try loading font information for T1+cmss on input line 134.

(/usr/local/texlive/2016/texmf-dist/tex/latex/base/t1cmss.fd
File: t1cmss.fd 2014/09/29 v2.5h Standard LaTeX font definitions
)
---TeX process (rubikrotation.sty)-------
---NEW rotation command------------------
---command = RubikRotation[1]{[SixSpot],U,Dp,R,Lp,F,Bp,U,Dp,<(8q*, 8f*)>}
---writing current Rubik state to file rubikstate.dat
\openout7 = `rubikstate.dat'.

\ourRRcounter=\count121
---CALLing Perl script (rubikrotation.pl)
runsystem(perl rubikrotation.pl -i rubikstate.dat -o rubikstateNEW.dat)...execu
ted.

---inputting NEW datafile (data written by Perl script)
(./rubikstateNEW.dat

...PERL process.................................
...script = rubikrotation.pl v4.0 (03 March 2017)
...reading the current rubik state (from File: rubikstate.dat)
...up,W,W,W,W,W,W,W,W,W
...down,Y,Y,Y,Y,Y,Y,Y,Y,Y
...left,B,B,B,B,B,B,B,B,B
...right,G,G,G,G,G,G,G,G,G
...front,O,O,O,O,O,O,O,O,O
...back,R,R,R,R,R,R,R,R,R
...
...command=checkstate
...checking state of cube
...cubiesum = 54 (Red=9, Or=9, Ye=9, Gr=9, Bl=9, Wh=9, X=0)
...
...command=rotation,[SixSpot],U,Dp,R,Lp,F,Bp,U,Dp,<(8q*, 8f*)>
...dataline = rotation,[SixSpot],U,Dp,R,Lp,F,Bp,U,Dp,<(8q*; 8f*)>
...[SixSpot] is a label OK
...rotation U, OK
...rotation Dp, OK
...rotation R, OK
...rotation Lp, OK
...rotation F, OK
...rotation Bp, OK
...rotation U, OK
...rotation Dp, OK
...writing new Rubik state to file rubikstateNEW.dat
...SequenceName = SixSpot
...SequenceInfo = (8q*; 8f*)
...SequenceShort = [SixSpot],U,Dp,R,Lp,F,Bp,U,Dp
...SequenceLong = U,Dp,R,Lp,F,Bp,U,Dp
)
```

2.2 ShowRubikErrors

In this example we demonstrate the use of the `\ShowRubikErrors` command, which places a copy of the Perl output file `rubikstateERRORS.dat` underneath the graphic so you can see a list of all the errors, if any. Note that this example is similar to the previous one except that we have introduced several errors—e.g., bad minipage width, typos, as well as some animals—into the rotation sequence). It is important to note that the `\ShowRubikErrors` command must be placed *after* the TikZ picture environment (i.e., in this case after the `\ShowCube` command), or even at the end of the document. Note that full details of all errors are also included in the .log file.



```
%% rubikstateERRORS.dat
%% --------------------
*ERR cmd= rotation,[sixspot],U,Dp,R,Lp,F,Bp,U,Dpppp,cat,dog
*ERR      Dpppp  -- code not known ? typo or missing comma
*ERR      cat  -- code not known ? typo or missing comma
*ERR      dog  -- code not known ? typo or missing comma
```

Figure 2: The same 'sixspot' sequence of rotations as shown in Example 1, but now with some errors (wrong minipage width, typos and some animals!) in the rotation sequence (it *should* be just `U,Dp,R,Lp,F,Bp,U,Dp`).

In this example we have used the F version of the `\ShowCube` command (`\ShowCubeF`) which places an fbox around the image so you can see the extent of any white space etc. This reveals that the set minipage-width (4.5cm) in the `\ShowCubeF` command—see code below—is too narrow: it should be 5cm (10×0.5) to just include the whole image (i.e., $10\times$ the TikZ scale-factor in this case). Once fixed, we can remove the F from the `\ShowCubeF` command. Note also that only '`\Draw...`' commands really need to be inside the TikZ picture environment (i.e., inside the `\ShowCube` command). The above figure was generated by the following code.

```
\RubikCubeSolved
\RubikRotation{[sixspot],U,Dp,R,Lp,F,Bp,U,Dpppp,cat,dog}
\begin{figure}[hbt]
  \centering
  \ShowCubeF{4.5cm}{0.7}{\DrawRubikCubeF}
  \ShowRubikErrors
\caption{....}
\end{figure}
```

Even if the `\ShowRubikErrors` command is not used, it is always a good idea to check the file `rubikstateERRORS.dat` after a LaTeX run, since this file will also reveal any errors.

Note that the completely flat form of the cube here is generated by the `\DrawRubikCubeF` command, where the terminal F denotes the Flat form.

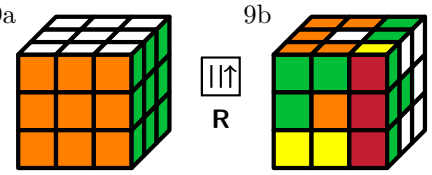
2.3 Environments

In this example we highlight the fact that Rubik commands used inside a LaTeX environment remain local to that environment, and how this can sometimes be problematic. Commands whose reach is meant to be more global need to be executed outside such environments, where they can implement global colour settings, which will then be accessible to Rubik `\Draw..` commands inside subsequent environments.

Since we are drawing images, this is primarily an issue with the the `\minipage`, `\figure`, and TikZ picture environments. Consequently, it is generally best when drawing a sequence of cubes to reserve the TikZ picture environment only for Rubik `\Draw..` commands and TikZ commands. Importantly, this also applies to the commonly used `\ShowCube` command, since this is a minipage-wrapper for the TikZ picture environment (see the RUBIKCUBE package documentation).

In this example the first cube (9a) uses a `\RubikCubeSolved` command *inside* a `\ShowCube` environment. However, if we now perform the rotation **R** (using the command `\RubikRotation{R}`) this results in a quite unexpected effect on cube (9b) (and obviously not correct). This is because the effect of the initial `\RubikCubeSolved` command (setting a new colour-state) is not visible outside its `\ShowCube` environment, and hence the subsequent `\RubikRotation{R}` command is unaware of this recent attempt to update the global colour-state information. It turns out that this was actually last updated following the action of the `\RubikRotation{[sixspot],....}` command used in Example 2, being the last colour-state command executed *outside* an environment (a `figure` environment in that example). Consequently, the strange form of cube (9b) is not what we expected.

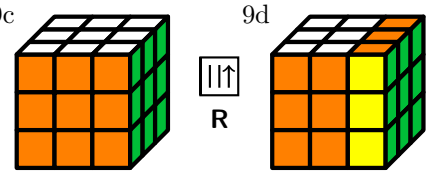
9a 9b




```
\usepackage{tikz,rubikcube,rubikrotation}
\newcommand{\cubenumber}[1]{\strut\raisebox{1cm}{#1}}
...
\cubenumber{9a}%
\ShowCube{2cm}{0.5}{%
    \RubikCubeSolved%
    \DrawRubikCubeRU%
     }%
\quad\Rubik{R}%
\RubikRotation{R}%
\cubenumber{9b}%
\ShowCube{2cm}{0.5}{\DrawRubikCubeRU}%
```

If we now bring the `\RubikCubeSolved` command out and place it before the `\ShowCube` command then its 'state' information becomes globally accessible (i.e., colour-state gets updated), and therefore gets used by the `\RubikRotation{R}` command, and hence cube (9d) is rendered correctly.

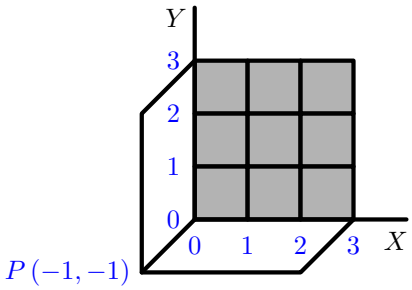
9c 9d




```
\RubikCubeSolved%
\cubenumber{9c}%
\ShowCube{2cm}{0.5}{\DrawRubikCubeRU}%
\quad\Rubik{R}%
\RubikRotation{R}%
\cubenumber{9d}%
\ShowCube{2cm}{0.5}{\DrawRubikCubeRU}%
\end{minipage}
```

2.4 Coordinates

For all cubes the origin of coordinates is defined as the bottom left corner of the FRONT face. Consequently, it is easy to determine the coordinates of points and hence draw lines, circles, and place lettering or other objects using the standard TikZ `\draw..` and `\node..` commands. Note that for convenience point P is designed to be $(-1,-1)$ on the 2D view. (The following diagram is Fig 1 from the RUBIKCUBE package documentation).



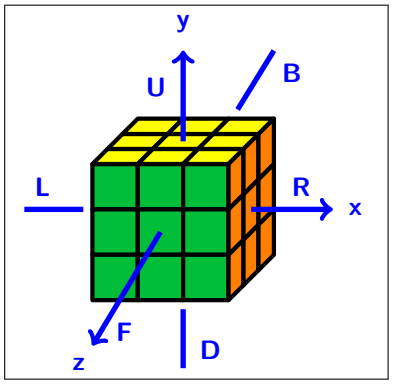
The code for the figure is given below.

We draw everything in the `\ShowCube` environment; the FRONT face in grey (colour code = `X`) using the Rubik command `\DrawFlatFront{X}`, and then draw all the lines and text using standard TikZ commands. The correct minipage-width argument (5.6cm) for the `\ShowCube` command is determined by trial-and-error, using the 'fbox' form of the command (`\ShowCubeF`), and then the 'F' is removed (→ `\ShowCube`). In order to avoid confusion, all Rubik commands start with a capital letter (e.g., `\Draw..`), while all TikZ commands start with a lower-case letter (e.g., `\draw..`).

```
\begin{figure}[hbt]
\centering
\RubikFaceFrontAll{X}% X = default non-colour (grey)
\ShowCube{5.6cm}{0.7}{%
   \DrawFlatFront
   \draw[line join=round,line cap=round,ultra thick] (0,0) -- (0,4);% Yaxis
   \draw[line join=round,line cap=round,ultra thick] (0,0) -- (4,0);% Xaxis
   \node (Ylabel) at (-0.35,  3.8)  {$Y$};
   \node (Xlabel) at ( 3.8,  -0.4)  {$X$};
   %% outline Left and Down faces
   \draw[line join=round,line cap=round,ultra thick]%
        (0,3) -- (-1,2) -- (-1,-1) -- (2,-1) -- (3,0);
   \draw[line join=round,line cap=round,ultra thick]%
        (-1,-1) -- (0, 0);
   \node (Y0) at (-0.4, 0)  [blue]{$0$};
   \node (Y1) at (-0.4, 1)  [blue]{$1$};
   \node (Y2) at (-0.4, 2)  [blue]{$2$};
   \node (Y3) at (-0.4, 3)  [blue]{$3$};
   \node (X0) at (0, -0.5)  [blue]{$0$};
   \node (X1) at (1, -0.5)  [blue]{$1$};
   \node (X2) at (2, -0.5)  [blue]{$2$};
   \node (X3) at (3, -0.5)  [blue]{$3$};
   \node (P) at (-2.4, -1)  [blue]{$P\,(-1,-1)$};
   }
\end{figure}
```

2.5 Face notation

The following diagram is Fig 2 from the RUBIKCUBE package documentation.



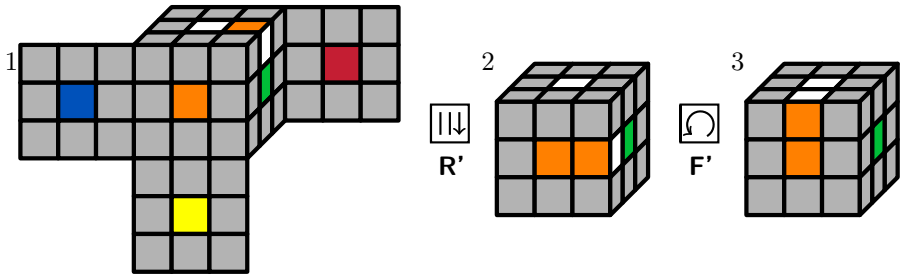
The code for the figure is as follows (the origin $(0, 0)$ is at the bottom left corner of the FRONT face).

```
\begin{figure}[htb]
\centering%
\RubikFaceUpAll{Y}
\RubikFaceFrontAll{G}
\RubikFaceRightAll{O}
\ShowCubeF{5cm}{0.6}{%
  \DrawRubikCubeRU%
  %% Right
  \draw[line width=2pt,color=blue,->] (3.5,2) -- (5.3, 2);
  \node (R) at (4.6, 2.5)  [blue]{\textbf{\textsf{R}}};
  \node (x) at (5.8, 2)  [blue]{\textbf{\textsf{x}}};
  %%Left
  \draw[line width=2pt,color=blue] (-0.2,2) -- (-1.5, 2);
  \node (L) at (-1.1, 2.5)  [blue]{\textbf{\textsf{L}}};
  %%Up
  \draw[line width=2pt,color=blue,->] (2, 3.5) -- (2, 5.5);
  \node (U) at (1.4, 4.7)  [blue]{\textbf{\textsf{U}}};
  \node (y) at (2, 6.1)  [blue]{\textbf{\textsf{y}}};
  %%Down
  \draw[line width=2pt,color=blue] (2, -0.2) -- (2, -1.5);
  \node (D) at (2.6, -1.1)  [blue]{\textbf{\textsf{D}}};
  %%Front
  \draw[line width=2pt,color=blue,->] (1.5, 1.5) -- (0, -1);
  \node (F) at (0.7, -0.7)  [blue]{\textbf{\textsf{F}}};
  \node (z) at (-0.3, -1.4)  [blue]{\textbf{\textsf{z}}};
  %%Back
  \draw[line width=2pt,color=blue] (3.2, 4.2) -- (4, 5.5);
  \node (B) at (4.4, 5)  [blue]{\textbf{\textsf{B}}};
  }
\end{figure}
```

2.6 Grey cube

When explaining elementary layer 1 moves, it can be useful to use the 'grey cube' (`\RubikCubeGrey`), as this sets up only the central cubie on each face; we have shown the first cube in Semi-Flat (SF) mode simply to show how the grey cube is configured (note that an 'all-grey' cube is also available: `\RubikCubeAllGrey`). Both of these grey cube commands will also accept the word 'gray' (to be consistent with TikZ).

In this example, we show how to position a single 'flipped' white/orange edge cubie in the top layer.



The code for the figure is given below. After setting up the first cube, we then just use the `\RubikRotation` command to generate the remaining cubes. The colours are coded as follows: R (red), O (orange), Y (yellow), G (green), B (blue), W (white), and X (grey).

```
\usepackage{tikz,rubikcube,rubikrotation}
\newcommand{\cubenumber}[1]{\strut\raisebox{1cm}{#1}}
...
\begin{figure}[hbt]
\centering
% set up the first cube
\RubikCubeGrey%
\RubikFaceUp{X}{X}{X}%
            {X}{W}{O}%
            {X}{X}{X}%

\RubikFaceRight{X}{W}{X}
               {X}{G}{X}
               {X}{X}{X}
\cubenumber{1}%
\ShowCube{5cm}{0.5}{\DrawRubikCubeSF}%
%
\quad\Rubik{Rp}\RubikRotation{Rp}
\cubenumber{2}%
\ShowCube{2cm}{0.5}{\DrawRubikCube}%
%
\quad\Rubik{Fp}\RubikRotation{Fp}
\cubenumber{3}%
\ShowCube{2cm}{0.5}{\DrawRubikCube}%
\end{figure}
```

2.7 Scramble a cube

In this example we use the `\RubikRotation` command to scramble a 'solved' Rubik cube via a sequence of 120 random rotations, using the following command (the details of the process can be seen in the `.log` file):

```
\RubikRotation{random,120}
```

On this occasion we draw the cube Flat (F) using the command `\DrawRubikCubeF`. In this example, we also make use of the `\SaveRubikState{}` command to save the final configuration (state) to a file (`rubikexampfig4.tex`) using `\SaveRubikState{rubikexampfig4.tex}`, so we can display the same cube configuration later but in a different format (we show it again in the following example (Example 2.8). Note that since we are using a random sequence, it follows that each time this file is run not only will a visually different cube be generated, but the same state will be shown in both here and in Example 2.8.

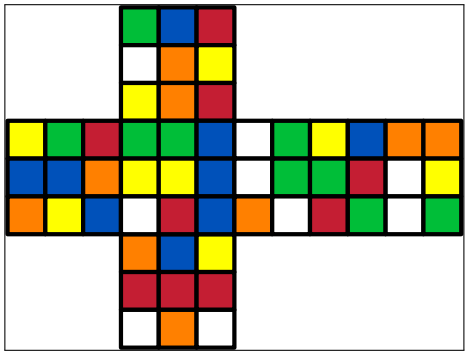


Figure 3: This shows a cube generated by 120 random rotations

```
\usepackage{tikz,rubikcube,rubikrotation}
...
\begin{figure}[hbt]
  \centering
  \RubikCubeSolved
  \RubikRotation{random,120}
  \SaveRubikState{rubikexampfig4.tex}
  \ShowCubeF{6cm}{0.5}{\DrawRubikCubeF}
\caption{....}
\end{figure}
```

Q: How do we determine the minipage-width (7.2cm) in the `\ShowCube` command?

A: The object is 12 cubie squares wide. Since the TikZ scale-factor argument of the `\ShowCube` command (cms/unit length; default = 1) in this case is set to 0.5, then the true width of the image will be $12 \times 0.5 = 6$ cm. Note that here we have used the `\ShowCubeF` command and so we can see that this is correct. Changing the scale-factor will change the image size and hence a new width argument will be required to just fit the image.

Note that in this particular case (where there is only a single image in the 'figure' environment), since the `\ShowCube` command places the image (in a TikZ picture environment) centrally inside a minipage, the image will in fact be centrally placed in the `\textwidth` provided the image is *smaller* than the fbox—i.e., if we used instead a minipage-width of, say, 12 cm the image would still appear centred in the `\textwidth` in this case. However, when there are several images in the 'figure', then the spacing may appear strange unless each image closely fits its own minipage-width etc. It is often useful, therefore, to check the size of the fbox (using the `\ShowCubeF` command) as we have done here.

2.8 SaveRubikState

In this example we display a cube having the same state as that shown in the previous example (Example 2.7). The configuration state was saved from Figure 3 using the command `\SaveRubikState{rubikexampfig4.tex}`, and then input here using `\input{rubikexampfig4.tex}`. These commands therefore allow the state of a previous cube to be saved to a file, and then displayed again later in a different format.

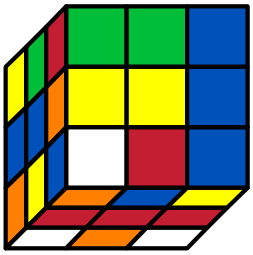


Figure 4: This shows a Rubik cube in exactly the same state as the one shown in Figure 3

```
\usepackage{tikz,rubikcube,rubikrotation}
...
\begin{figure}[hbt]
  \centering
  \input{rubikexampfig4.tex}
  \Showcube{4cm}{0.8}{\DrawRubikCubeLD}
\caption{....}
\end{figure}
```

2.9 Series of cubes

Here we show a convenient way of displaying a series of small cubes showing a sequence of rotations (**U**, **R**, **F**).

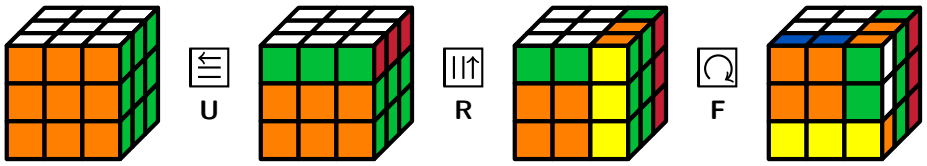


Figure 5: The rotations **U**, **R**, **F** on a solved cube.

The code for the above sequence is as follows:

```
\usepackage{tikz,rubikcube,rubikrotation}
...
\begin{figure}[hbt]
  \centering%
  \RubikCubeSolved%
  \ShowCube{2cm}{0.5}{\DrawRubikCubeRU}%
  \quad\Rubik{U}\quad%
  \RubikRotation{U}\ShowCube{2cm}{0.5}{\DrawRubikCubeRU}%
  \quad\Rubik{R}\quad%
  \RubikRotation{R}\ShowCube{2cm}{0.5}{\DrawRubikCubeRU}%
  \quad\Rubik{F}\quad%
  \RubikRotation{F}\ShowCube{2cm}{0.5}{\DrawRubikCubeRU}%
\caption{The rotations \rr{U}, \rr{R}, \rr{F}\ on a solved cube.}
\end{figure}
```

Note that we are starting with the default white-opposite-yellow (WY) solved cube, using the command `\RubikCubeSolved`, which is functionally the same as the more explicit `\RubikCubeSolvedWY` (if you forget the terminal two letters then at least you will get a 'solved' cube). A white-opposite-blue (WB) solved cube is available as `\RubikCubeSolvedWB`.

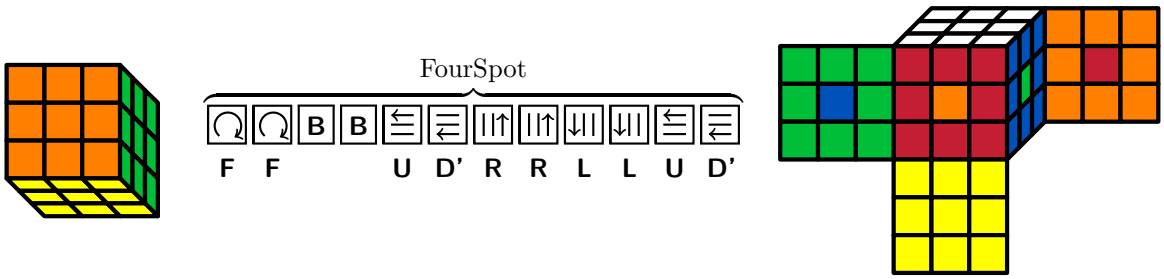
2.10 Rotation sequence

We now explore using the named Rubik cube rotation sequences and associated patterns available in the RUBIKPATTERNS package—a small macro database (see the RUBIKPATTERNS documentation, and also its companion file `rubikpatternsLIST.pdf`). Having the sequences available as macros is very convenient since (a) it avoids making errors when typing them out, and (b) allows the easy application of software tools.

A Rubik pattern is the configuration generated by a sequence of rotations (or 'moves') from some initial starting configuration (typically a 'solved' configuration). For example, FourSpot is a well known pattern which we can generate from a solved Rubik cube using the macro `\FourSpot`; it is defined as follows:

```
\newcommand{\FourSpot}{[FourSpot],F2,B2,U,Dp,R2,L2,U,Dp,<(12q*, 8f*)>}
\newcommand{\fourspot}{\FourSpot}
```

Note that for convenience the macros names in the RUBIKPATTERNS package are defined in both upper and lower-case (i.e., the macros `\FourSpot` and `\fourspot` are identical). The following figure shows the FourSpot sequence and pattern.



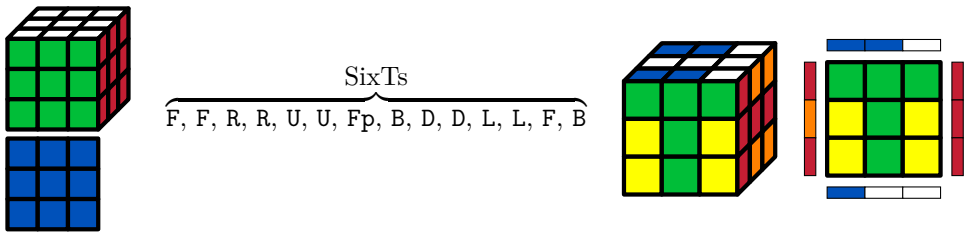
The code for the above figure is as follows:

```
\usepackage{tikz,rubikcube,rubikrotation,rubikpatterns}
...
\noindent%
\RubikCubeSolvedWY
\ShowCube{2cm}{0.5}{\DrawRubikCubeRD}
\RubikRotation{\FourSpot}
\quad
\SequenceBraceA{FourSpot}{%
                          \ShowSequence{}{\Rubik}{\SequenceLong}%
                         }
\quad
\ShowCube{2cm}{0.5}{\DrawRubikCubeSF}
```

Note that we have spread the code slightly here in order to emphasise that the `\ShowSequence` command is being used as an argument for the `\SequenceBraceA` command (the 'A' in the command `\SequenceBraceA` denotes that the annotation is placed Above the sequence.) We have used a solved cube with the WY (White opposite Yellow) configuration (`\RubikCubeSolvedWY`). The first cube is drawn from the RD (Right-Down) viewpoint (`\DrawRubikCubeRD`). The second cube is drawn from the SF (Semi-Flat) viewpoint (`\DrawRubikCubeSF`) so we can see all the faces.

2.11 SixTs

A more interesting cube pattern is the SixTs configuration (from the RUBIKPATTERNS package), which we now show in a slightly different way (adding an extra face), as follows:



This time we have started with a solved cube having the WB configuration (White opposite Blue), which is generated using the command `\RubikCubeSolvedWB` (we have added the DOWN face (blue) below to reveal the colour of this face—see note below).

The rotation sequence is in 'long-format' (expanded into separate rotations), comma-separated and space, typewriter font, using the command `\ShowSequence{,\ }{\texttt}{\SequenceLong}`.

The final image shows just the FRONT face together with all the side-bars indicating the colours of the adjacent facelets, using the `\DrawFaceFrontSide` command. The images are separated using `\quad`. The code for the above figure is as follows:

```
\usepackage{tikz,rubikcube,rubikrotation,rubikpatterns}
...
\noindent\hfil
\RubikCubeSolvedWB
\ShowCube{1.6cm}{0.4}{%
                      \DrawRubikCubeRU%
                      \DrawFlatDown{0}{-3.3}%
                      }
\RubikRotation{\SixTs}
\quad\SequenceBraceA{SixTs}{%
                      \ShowSequence{,\ }{\texttt}{\SequenceLong}%
                      }
\quad\ShowCube{2cm}{0.5}{\DrawRubikCubeRU}
\quad\ShowCube{2cm}{0.5}{\DrawFaceFrontSide}
\hfil
```

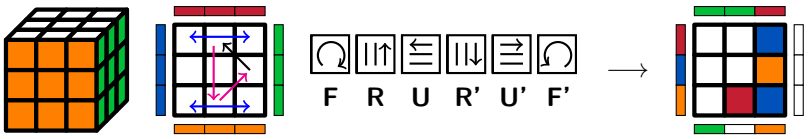
Notes

1. We have drawn the DOWN face of the first cube using the command `\DrawFlatDown{0}{-3.3}` where the two arguments are the x and y coordinates of the *bottom left* corner of the DOWN face (blue). Note that the grid origin of all cube images coincides with the *bottom left* corner of the FRONT face (green in this case).

2. The first image is really just 4 units wide. This is because the 2D width of the *side* face (red) is designed to measure 1 unit wide in the oblique view (similarly, the 2D height of the *top* face also measures just 1 unit). Consequently, since the TikZ scale factor used is 0.4, then the (minimum) width argument for its `\ShowCube{}{}{}` command is $4 \times 0.4 = 1.6$cm., hence we have `\ShowCube{1.6cm}{0.4}{...}`.

2.12 Three-edge cycle

The following example shows a sequence often used in solving the final layer. The black (no flip) and magenta (flip) arrows indicate the UP face cubie movement associated with the 'three-edge cycle sequence **F**,**R**,**U**,**R'**,**U'**,**F'**. The blue arrows indicate so-called collateral damage (two pairs of corner cubies swap positions) which can be fixed at a later stage. (This diagram is from Section 13 in the RUBIKCUBE package documentation).



The code for the figure is as follows

```
\bigskip%
\noindent\hfil%
\RubikCubeSolved%
\ShowCube{1.6cm}{0.4}{\DrawRubikCubeRU}%
\quad\ShowCube{1.6cm}{0.4}{%
   \DrawFlatUpSide%
   \draw[thick,->,color=magenta] (1.5,0.5) -- (2.4, 1.4);
   \draw[thick,->] (2.5,1.5) -- (1.6, 2.4);
   \draw[thick,->,color=magenta] (1.3, 2.3) -- (1.3, 0.5);
   \draw[thick,<->,  color=blue] (0.5,2.6) -- (2.5, 2.6);
   \draw[thick,<->,  color=blue] (0.5,0.3) -- (2.5, 0.3);
   }%
\RubikRotation{F,R,U,Rp,Up,Fp}%
\quad\ShowSequence{}{\Rubik}{\SequenceLong}\quad$\longrightarrow$\quad%
\ShowCube{1.6cm}{0.4}{\DrawFlatUpSide}%
\hfil%

\bigskip
```

2.13 Superflip

Once you can solve Rubik's cube, then an interesting exercise is to generate the so-called 'superflip' configuration, in which all the corners are correctly solved, while all the edges are flipped [2].

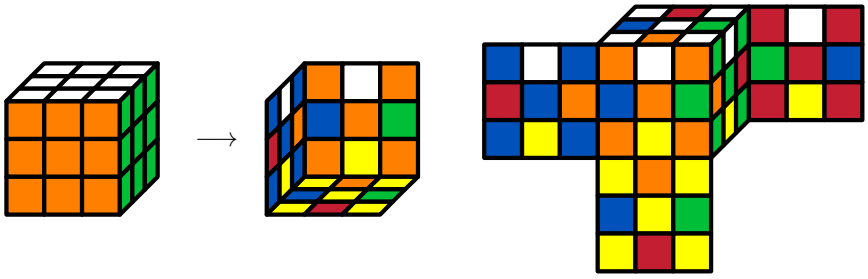


Figure 6: Two representations of the superflip configuration.

A superflip sequence converts the solved cube on the left into the form on the right, using the command `\RubikRotation{\superflip}`. The full code for the above figure is as follows:

```
\usepackage{tikz,rubikcube,rubikrotation,rubikpatterns}
...
\begin{figure}[hbt]
  \centering
  \RubikCubeSolved
  \ShowCube{2cm}{0.5}{\DrawRubikCubeRU}
  \quad$\longrightarrow$\quad%
  \RubikRotation{\superflip}%
  \ShowCube{2cm}{0.5}{\DrawRubikCubeLD}
  \quad\quad%
  \ShowCube{5cm}{0.5}{\DrawRubikCubeSF}
\caption{...}
\end{figure}
```

The following superflip sequence [3] has just 20 HTM rotations (Half Turn Metric: counting 180 degree turns as just one 'rotation'). Note that the RUBIKPATTERNS package contains this particular superflip sequence as the macro `\superflip` (see `rubikpatterns.pdf`). Consequently the code `\ShowSequence{,}{\large\texttt}{\superflip}`, will typeset the sequence as follows:

```
[Superflip],Dp,R2,Fp,D2,F2,U2,Lp,R,Dp,R2,B,F,Rp,U2,Lp,F2,Rp,U2,Rp,Up,<(20f*)>
```

Note that for convenience, the RUBIKPATTERNS package includes the sequence name (in square brackets) as the first element of the associated macro. This is possible since the contents of a comma-separated square bracket is not actioned as a rotation when it appears as part of the argument of the `\RubikRotation` command.

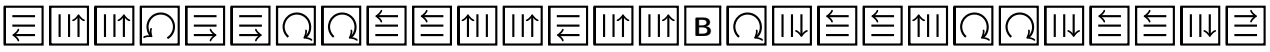
[2] See the 'superflip' entry in *Wikipedia*, and also the Kociemba website (`www.kociemba.org/cube.htm`); particularly the page `http://kociemba.org/math/oh.htm`

[3] This particular superflip sequence (in the RUBIKPATTERNS package) is due to Reid (1995); for details see the RUBIKPATTERNS package documentation, and also `http://kociemba.org/math/oh.htm`. Another 20-move superflip sequence (due to H Kociemba), is designated as K32466 in `www.nickalls.org/dick/papers/tex/RUBIK20moves.zip`.

Next we present the sequence without commas in the form of hieroglyphs using the `\Rubik` font, for which we require the 'expanded' `\SequenceLong` form (since the 'short form' includes trailing digits—see § 9 in `rubikcube.pdf`), using the code

```
\usepackage{tikz,rubikcube,rubikrotation,rubikpatterns}
...
\RubikCubeSolved
\RubikRotation{\superflip}
\noindent\strut\hspace{-8mm}\ShowSequence{}{\Rubik}{\SequenceLong}
```

which gives

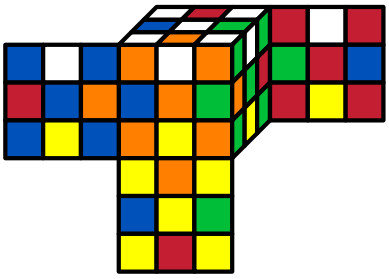


Equivalent sequence

Interestingly, the superflip sequence is actually equivalent to $\left\{\left(\text{Rm' U'}\right)4,\ [\text{y'}],\ [\text{x}]\right\}3$. Furthermore we can readily demonstrate this, as we can process this novel form of the sequence using some useful features of the `\RubikRotation` command, as follows:

```
\RubikCubeSolved%
\RubikRotation[3]{[superflip],(Rmp,Up)4,yp,x}%
\ShowCube{4cm}{0.4}{\DrawRubikCubeSF}%
```

which generates the following



which is exactly the same configuration as before. Note that to do this we had to make use of the 'repeat' option `[3]` as well as the `(Rmp,Up)4` 'repeat-block' in the argument of the `\RubikRotation` command above.

2.14 Inverse sequence

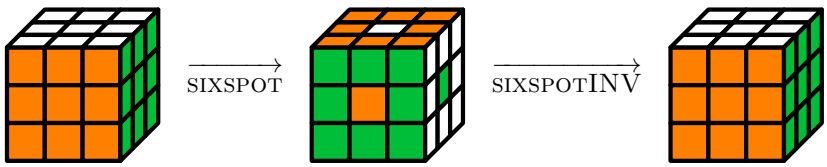
Generating the inverse of a Rubik sequence involves (a) reversing the order of the sequence, and (b) inverting each rotation in the sequence (see Sections 5.1 and 5.11 in the RUBIKROTATION package documentation).

From the grey-cube example (2.6) we saw that the sixspot sequence is: `U,Dp,R,Lp,F,Bp,U,Dp`; its inverse is therefore readily determined as `D,Up,B,Fp,L,Rp,D,Up`. Note that this is easy to check since the sequence generated by the `\RubikRotation` command is held by the macro `\SequenceLong`. For example, the output of the following commands

```
\fbox{\strut\ %
The inverse of the sixspot sequence is:
\RubikRotation{\sixspot,<inverse>}
\ShowSequence{,}{\texttt}{\SequenceLong}.
}
```

is The inverse of the sixspot sequence is: D,Up,B,Fp,L,Rp,D,Up.

A sequence and its inverse will annihilate each other when applied consecutively. For example, in the following figure we start with a solved cube and apply the sixspot sequence. Applying the inverse of the sixspot sequence then results in the solved cube configuration again.



The code for the above figure is as follows:

```
\usepackage{tikz,rubikcube,rubikrotation,rubikpatterns}
...
\newcommand{\sixspotINV}{[sixspotINV],D,Up,B,Fp,L,Rp,D,Up}
\newcommand{\sixspotarrow}{$\quad\overrightarrow{\strut\textsc{sixspot}}\quad$}
\newcommand{\sixspotINVarrow}{$\quad\overrightarrow{\strut\textsc{sixspotINV}}\quad$}

\begin{figure}[hbt]
\centering
\RubikCubeSolved%
   \ShowCube{2cm}{0.5}{\DrawRubikCubeRU}\sixspotarrow%
\RubikRotation{\sixspot}%
   \ShowCube{2cm}{0.5}{\DrawRubikCubeRU}\sixspotINVarrow%
\RubikRotation{\sixspotINV}%
   \ShowCube{2cm}{0.5}{\DrawRubikCubeRU}
\end{figure}
```

A significant property of the superflip configuration is that it is its own inverse. Consequently we can achieve a similar result simply by applying the superflip sequence *twice in succession*, as follows:

—— END ——