This is currently a collection of notes and work in progress.

# Architecture

In reality, the core architecture is just the simple “Model-View-Controller” pattern, or for this application, more logically phrased as a “Controller-Model-View” pattern. Specifically:

## Controller

This lion-share of the application complexity in both code and CPU consumption is in this domain. Specifically, each incoming Camera Image Frame is fully analyzed in an attempt to recognize a Rubik Face. Then, additional processing is performed in class StateMachine to determine if the Model (or State) should be changed, and, when all six sides of the Rubik Cube have been recorded, the TwoPhase Rubik Cube Logic solution is called.

## Model (or State)

Rubik Cube state is captured in the simple class StateModel.java. Besides the cube six faces, a small amount of additional data (state) is held in this class. Otherwise, this represents pretty much the full state of the system.

## View

There are actually four separate “Views,” they are:

### Diagnostic Annotations

A wide range of diagnostic information is available through the Menu->Annotation user interface. This is rendered using OpenCV on the frame image itself. Information broader that what is in state is offered.

### Pilot Cube

At the time of this writing, the Pilot Cube is immature, but its long term purpose is to represent the state of the cube both in tiles and faces, and eventually to show requested Cube Rotations. This is rendered using OpenGL.

### Requested Text Directions

Text instructions to user to rotate entire cube or rotate an edge can be optionally displayed as a text message. This is rendered using OpenCV on the frame image itself.

### Requested Arrow Directions

Graphic instructions in the form of 3D arrows are rendered in a fashion overlaying the cube itself with proper position, orientation and size.

# Notes

* Prefix Rubik on all allows for easy identification and use of common names like Util and Controller.
* Significant use of public key word on member variables.

# Event Model

## Thread Model

Nearly all code is executed on the OpenCV Frame thread.

## Tracking and Feedback

For simplicity, no frame to frame information path exists. That is, each frame is processed without any information from the previous frame.

# Architecture

## Image Processing

## Face Recognition

## Cube Model (i.e., State)

## Gesture Recognition

# Image Analysis Design

The main part of this application is image recognition

The top-level design description is according to the Sonka, Hlavac and Boyle text.

## Boarders

More specifically, in this application, “contours” are obtained and used.

## Curves

## Segments

## Syntactic Description

## Syntactic Recognition

## High Level Image Representation

## Image Understanding

# Class Diagram

**+**

Activity

Menu and Params

Image Recognizer

Controller

State Model

Face

Instructions View

Overlay View

Pilot Cube View

Rhombus

Polygon

Contour

# Controller State Diagram