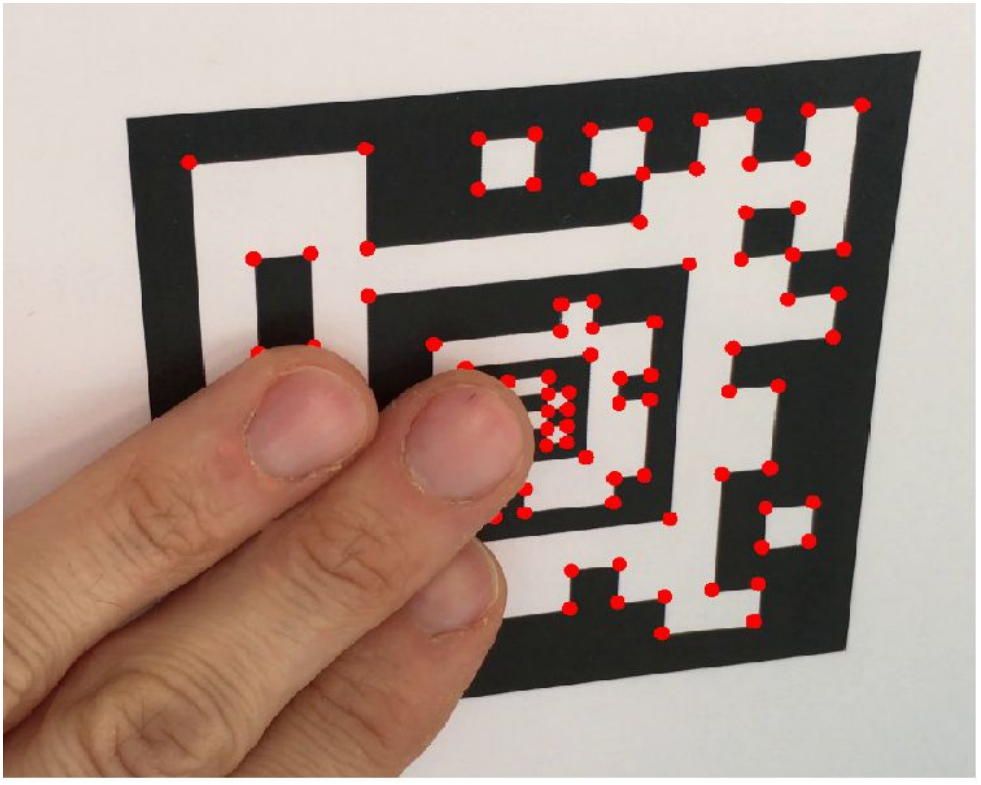
Fractal Markers: a new approach for long-range camera pose estimation under occlusion



Author. Francisco José Romero Ramírez

License and how to cite

The library is released under GPLv3. Please cite if you use it for research:

FRACTAL MARKERS

“Fractal Markers: a new approach for long-range camera pose estimation under occlusion”. Please cite our paper (under review). You can find preprint version at https://www.researchgate.net/publication/332727382_Fractal_Markers_a_new_approach_for_long-range_marker_pose_estimation_under_occlusion

ARUCO

Main Paper of Aruco version 3:
"Speeded Up Detection of Squared Fiducial Markers"
https://www.sciencedirect.com/science/article/pii/S0262885618300799
[DOWNLOAD PDF]
[DOWNLOAD Bibtex]

Paper that started Aruco project:
"Automatic generation and detection of highly reliable fiducial markers under occlusion"
http://www.sciencedirect.com/science/article/pii/S0031320314000235
[DOWNLOAD PDF]
[DOWNLOAD Bibtex]

Paper explaining the default dictionary of markers:
"Generation of fiducial marker dictionaries using mixed integer linear programming"
http://www.sciencedirect.com/science/article/pii/S0031320315003544
[DOWNLOAD PDF]
[DOWNLOAD Bibtex]

1. Introduction

Fractal markers are a new concept of marker, which is composed of several fiducial square markers of different size inside. Unlike traditional fiducial markers, the structure of this marker can be detected from a large number of distances, as well as solve problems of partial or total occlusion of the marker.

The features of this marker, together with the tools developed make it a powerful tool for camera pose estimation in a large number of applications such as robots, unmanned vehicles and augmented reality.

Fractal Marker is integrated inside ArUco's libraries, allowing a fast, robust and precise detection of the markers. ArUco is a widely used OpenSource library for detecting squared fiducial.

2. Fractal Markers

A fractal marker is a square fiducial marker composed of m markers recursively, see Figure 1. In a fractal marker, each internal marker is composed of a black region that facilitates the detection of the marker (s(f)), a region that identifies the marker (n(f)), and a white region where its immediate inner marker is located (k(f)).

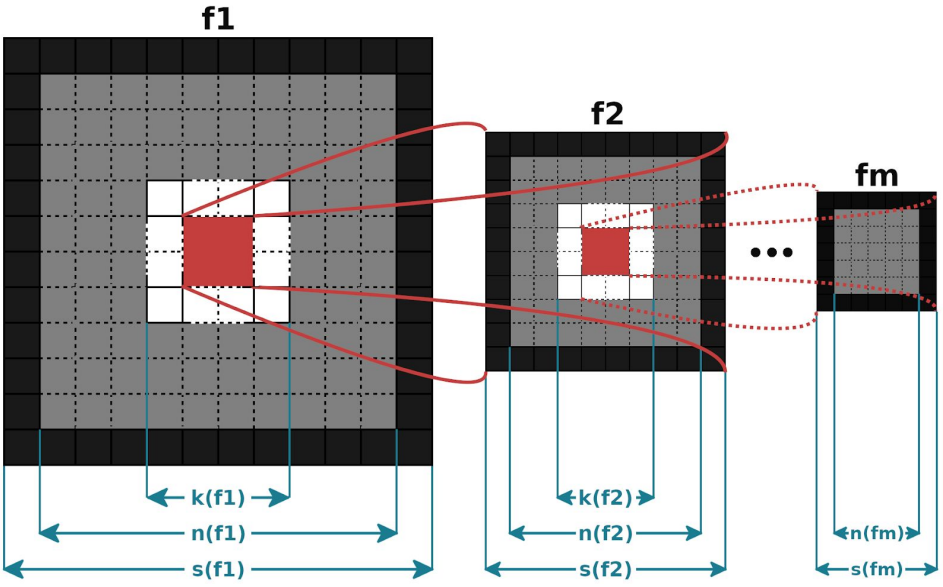


Figure 1. Generic configuration of a fractal marker

The value of the parameters n and k for each marker will determine its size as well as the number of bits that can be used for its correct identification. Thus, we can describe the configuration of a fractal marker as: n(f1):k(f1),n(f2):k(f2), … , n(fm):n(fm).

Default markers

The system has implemented four markers that can be used by default (See Figure 2). The markers have different configurations, however the user will be able to create his own marker according to his own needs as we will see next.

- **FRACTAL_2L_6.** Fractal marker composed of 2 internal markers. The configuration of the regions are: $n(f^1)=10$, $k(f^1)=6$; and $n(f^2)=6$, $k(f^2)=0$. We can summarize the configuration as 10:6,6:0 (See Figure 2a).

- **FRACTAL_3L_6.** Fractal marker composed of 3 internal markers. The configuration of the regions are: $n(f^1)=12$, $k(f^1)=8$; $n(f^2)=10$, $k(f^2)=6$ and $n(f^3)=6$, $k(f^3)=0$. We can summarize the configuration as 12:8,10:6,6:0 (See Figure 2b).

- **FRACTAL_4L_6.** Fractal marker composed of 4 internal markers. The configuration of the regions are: $n(f^1)=13$, $k(f^1)=9$; $n(f^2)=12$, $k(f^2)=8$; $n(f^3)=10$, $k(f^3)=6$; and $n(f^3)=6$, $k(f^3)=0$. We can summarize the configuration as 13:9,12:8,10:6,6:0 (See Figure 2c).

- **FRACTAL_5L_6.** Fractal marker composed of 5 internal markers. The configuration of the regions are: $n(f^1)=11$, $k(f^1)=7$; $n(f^2)=13$, $k(f^2)=9$; $n(f^3)=12$, $k(f^3)=8$; $n(f^4)=10$, $k(f^4)=6$; and $n(f^5)=6$, $k(f^5)=0$. We can summarize the configuration as 13:9,12:8,10:6,6:0 (See Figure 2d).

Custom markers

The user can create his own configuration using the program utils_fractal/fractal_create. The program generates as output a text file yml, with the configuration of the markers, it will only be necessary to specify the size of the regions previously mentioned, for each internal marker of the fractal marker.

./fractal_create <output> <regions> [-s bitsize]

Where:

- ***output***. Path of the output configuration file.
- ***regions***. Configuring marker regions. n(f1):k(f1),n(f2):k(f2),...,n(fm),k(fm)
- ***bitsize***. Size of each bit for the last level marker (fm). If not specified the fractal marker is normalized.

Example of use for the creation of a 3-level marker:

./fractal_create configuration.yml 10:8,14:10,6:0 -s 50

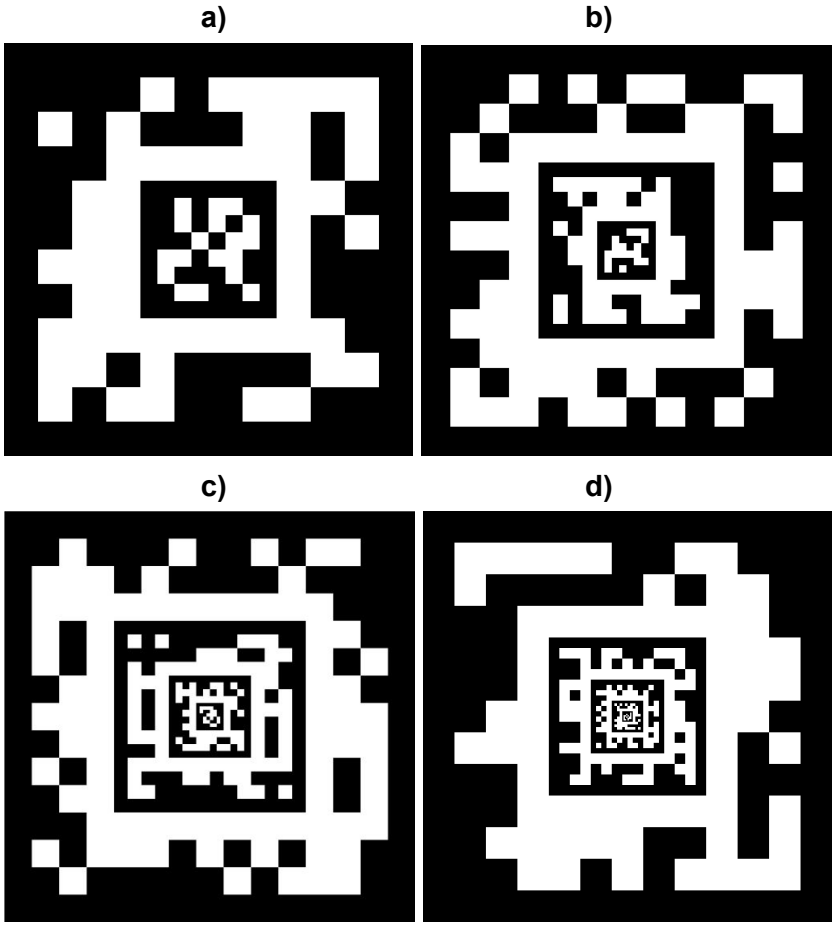


Figure 2. Default Fractal Marker configurations included in the library.

Printing markers

Now you can print the marker you want. The tool utils_fractal/fractal_print_marker has been provided.

./fractal_print_marker <output> [-c configuration] [-bs bitsize] [-noborder]

Where:

- ***output***. Path of the output image (jpg|png|ppm|bmp)
- ***configuration***. Name of the marker configuration in case you want to print a default marker (If this is not specified the system uses by default FRACTAL_2L_6), or the configuration path of the file generated by the fractal_create tool.
- ***bitsize***. Number of pixels occupied by each bit of the innermost marker (75px. by default)

Examples:

./fractal_print_marker fractalMarker.png
./fractal_print_marker fractalMarker.png -c FRACTAL_3L_6 -bs 50
./fractal_print_marker fractalMarker.png -c configuration.yml

3. Fractal marker detection and pose estimation

Fractal Marker System uses ArUco to perform markers detection. The process of calling ArUco is completely transparent to the user, and carried out using the FractalDetector class.

If the extrinsic parameters of the camera are known (camera matrix and distortion coefficients obtained by calibration, for more details on how to perform the camera calibration), then the library will be able to calculate the relative position of the fractal marker and the camera. The system will give you the rotation and translation vectors (Rvec and Tvec), which represent the transformation of the marker to the camera. The Rvec and Tvec transformation vectors are provided by performing the **poseEstimation()** implemented in the FractalDetector class.

The Rvec and Tvec vectors can be used by cv::projectPoints() to project points on the image using the fractal marker reference system. Notice, that the fractal marker is normalized during the tracking process indicating that (0,0) will be the shared center of all the markers and (-1,1),(1,1),(1,-1),(-1,-1),(-1,-1) correspond to the four external corners of the fractal marker.

The program utils_fractal/fractal_tracker allows in an easy and automatic way the detection of markers in the scene, as well as the camera pose estimation. It is also possible to track the marker in a video sequence.

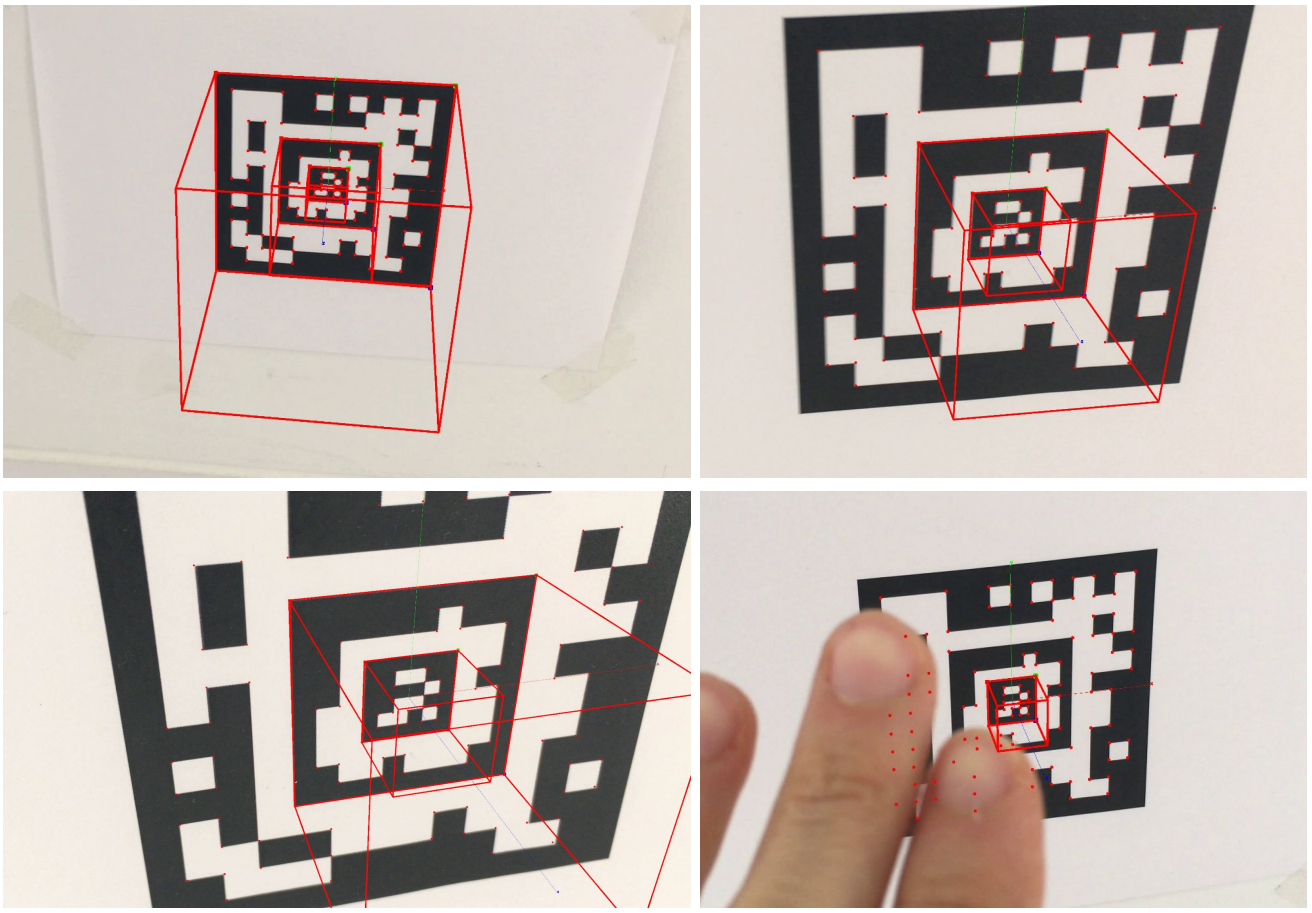
./fractal_tracker <input> [-cam cameraParams] [-c configuration]

Where:

- ***input***. Path of the input video
- ***cameraParams.*** *Camera calibration parameters.*
- ***configuration***. Name of the marker configuration in case you want to print a default marker (If this is not specified the system uses by default FRACTAL_2L_6), or the configuration path of the file generated by the fractal_create tool.

Example

./fractal_tracker video.avi -cam cameraParams.yml -c FRACTAL_3L_6



4. Using Fractal Markers in your project

Here we show you how to create your own project using the Aruco and fractal markers libraries. The code is available for use in SourceForge. In the same way, the data and the example project described is available for use in the following link. https://mega.nz/#F!4gAwjQ4L!LyWCbB7Fewd6fR43x-_5_Q

We will use cmake to manage our project. First, we create a directory in which we will place the following file CMakeLists.txt:

```
cmake_minimum_required(VERSION 3.0)
project(examples LANGUAGES CXX)

set(CMAKE_INCLUDE_CURRENT_DIR ON)
set(CMAKE_CXX_STANDARD 11) # C++11...
set(CMAKE_CXX_STANDARD_REQUIRED ON) #...is required...
set(CMAKE_CXX_EXTENSIONS ON) #...with compiler extensions like gnu++11
```

```
find_package(OpenCV REQUIRED)
include_directories(${OpenCV_INCLUDE_DIRS})
find_package(aruco REQUIRED)

add_executable(sample_detection sample_detection.cpp)
target_link_libraries(sample_detection          aruco opencv_calib3d)
```

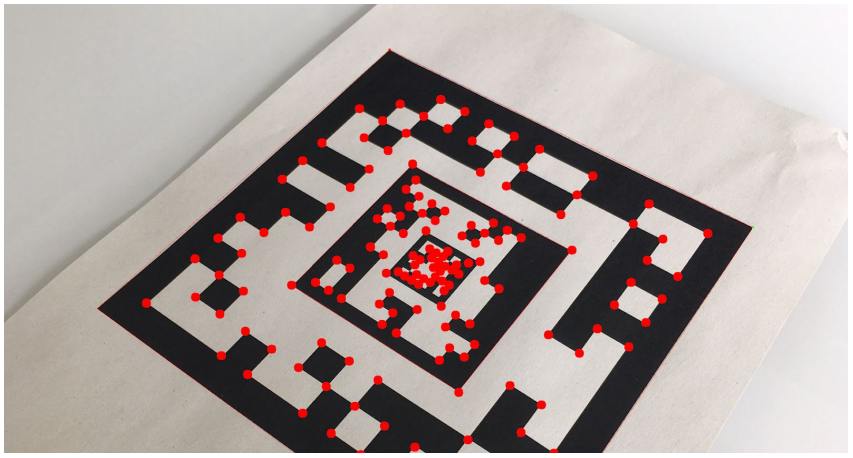
Example1:

Then, create the program file sample_detection.cpp. We will be able to detect fractal markers (eg. **FRACTAL_5L_6**) at the scene.

```
#include <opencv2/highgui.hpp>
#include <aruco.h>
using namespace std;
int main(int argc,char **argv)
{
   cv::Mat im=cv::imread(argv[1]);
   aruco::FractalDetector FDetector;
   FDetector.setConfiguration("FRACTAL_3L_6");
   if(FDetector.detect(im)){
        FDetector.drawMarkers(im);
   }
   cv::imshow("image",im);
   cv::waitKey(0);
}
```

Finally, create a directory for building (e.g. build), go in, compile and execute

```
mkdir build
cd build
cmake .. -Daruco_DIR=<path2arucoConfig.cmake> -DOpenCV_DIR=<path2OpenCVConfig.cmake>
make
./sample_detection image_with_markers.jpg
```



Example2:

The following example shows a simulation of a drone flight. In the video, the fractal marker **FRACTAL_5L_6** is detected in the scene and its four corners are used to project an image over it.

The example could be described in two stages:

1. Detection of markers (same as previous example).
2. The program uses the corners of the detected markers to obtain the homography and thus be able to transform the four 3D corners of the fractal marker into the image, 2D corners.

```
./sample_landing landing.mov landingPad.png
```