TANSZÉKVEZETŐ

DIPLOMATERVEZÉSI FELADAT

Danyi Dávid
Villamosmérnök hallgató részére

Marker alapú helymeghatározás képfeldolgozással

Az információfeldolgozási kapacitás nagymértékű növekedésével párhuzamosan egyre szélesebb körben váltak alkalmazhatóvá (valós időben) képfeldolgozási, gépi látás alapú eljárások. Az ideálistól eltérően a valós képek feldolgozását számos hatás nehezíti, úgy mint zaj, optikai torzítások, megvilágítás- és színbeli különbségek, stb.

A feladat témája aktuális, az egyre gyakrabban alkalmazott mobil robotikai, kiterjesztett és virtuális valóság alkalmazásoknál minden esetben felmerül a „hol vagyok?" kérdés, azaz a nézponti paraméterek becslése, lokalizáció. A SLAM (Simultaneous Localization and Mapping) algoritmus számos szenzorforrás által biztosított információval alkalmazható, így kamerával is. Az általános megoldás tájékozódási pontok egymáshoz képesti elhelyezkedését becsli, a mérési bizonytalanságot folyamatosan csökkentve, valamint méri a tájékozódási pontokhoz képest a megfigyelő pozícióját.

Jelen munka is a fenti témához, kapcsolódik, cél megvizsgálni egy olyan mesterséges, passzív marker megvalósíthatóságát, ami bizonyos paraméterekben (azonosíthatósági tartomány, részleges láthatóság) jobbat kíván biztosítani, mint az elterjedt (ARtag, glyph, stb.) megoldások. A marker egy oldalukon nyitott különböző méretű és alakú négyszögeket tartalmaz. A diplomaterv célja megvizsgálni a javasolt marker jellemzőit és használhatóságát.

A hallgató feladatának a következőkre kell kiterjednie:

- Mutasson be pozícióbecslő algoritmusokat, röviden ismertesse ezek működését!
- Hasonlítson össze különböző nézőpontbecslési algoritmust síkban elhelyezkedő pontpárok alapján!
- Készítsen egy marker felismerő megoldást!
- Végezzen méréseket (ideális és valós képeken) a marker által meghatározott pozíció pontosságára vonatkozóan!
- Hasonlítsa össze és értékelje az eredményeket!

Tanszéki konzulens: Kovács Viktor, tanársegéd

Budapest, 2017. február 18.

Dr. Charaf Hassan
egyetemi tanár
tanszékvezető

Budapest Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Automatizálási és Alkalmazott Informatikai Tanszék

H-1117 Budapest, Magyar tudósok krt 2. Q.B.207
Telefon: 463-2870. Fax: 463-2871
www.aut.bme.hu

Budapest University of Technology and Economics
Faculty of Electrical Engineering and Informatics
Department of Automation and Applied Informatics

Marker Based Localisation and Pose Estimation Using Image Processing

Thesis

Author
Dávid Danyi

Consultant
Viktor Kovács

May 12, 2017

Contents

HALLGATÓI NYILATKOZAT

Alulírott *Dávid Danyi*, szigorló hallgató kijelentem, hogy ezt a szakdolgozatot meg nem engedett segítség nélkül, saját magam készítettem, csak a megadott forrásokat (szakirodalom, eszközök stb.) használtam fel. Minden olyan részt, melyet szó szerint, vagy azonos értelemben, de átfogalmazva más forrásból átvettem, egyértelmûen, a forrás megadásával megjelöltem.

Hozzájárulok, hogy a jelen munkám alapadatait (szerzõ(k), cím, angol és magyar nyelvû tartalmi kivonat, készítés éve, konzulens(ek) neve) a BME VIK nyilvánosan hozzáférhetõ elektronikus formában, a munka teljes szövegét pedig az egyetem belsõ hálózatán keresztül (vagy autentikált felhasználók számára) közzétegye. Kijelentem, hogy a benyújtott munka és annak elektronikus verziója megegyezik. Dékáni engedéllyel titkosított diplomatervek esetén a dolgozat szövege csak 3 év eltelte után válik hozzáférhetõvé.

Budapest, May 12, 2017

Dávid Danyi
hallgató

Abbreviations

This is a complete list of the abbreviations used in this paper.

DOF Degrees of freedom

RQIM Random Quad Image Marker

Chapter 1

Markers

One of the goals of this project was to design a fiducial marker with advantageous properties for use in pose estimation. In a typical scenario the marker may be seen from largely varying viewpoints, therefore it has to have some level of scale invariability. If the observer is far from the marker, the smaller details may be lost due to the limited resolution of the camera. If the same observer moves closer to the marker, it may fill the whole field of view and some features may even slip off the image. This leads to another feature the marker needs to have: redundancy. If the observer gets too close to the marker or some obstacle partially blocks the view, the localisation still needs to provide usable results.

The intended use of the markers is spatial localisation and pose estimation. In other words: approximating the observers 3D coordinates (x, y, z) and orientation (ϕ, θ, ψ) with respect to the marker. It is supposed that the observer uses a single camera system for navigation (e.g. smartphone or robotic application with limited resources). This means the marker needs at least 6 degree of freedom.

To sum up the above discussed specifications, a suitable marker would have to:

- have at least 6 DOF
- be (to some degree) scale invariant
- have redundancy

In the following sections will be a recommendation for a marker conforming for the listed specifications. It is based on 3 connected line segments forming a quad with one missing side. The whole marker is built from quads with different side lengths and angles.

1.1 Quad

A marker is put together from quads. Figure 1.1. shows two examples. One side of the quads is left out: they are put together from three joint line segments. The middle segment, with

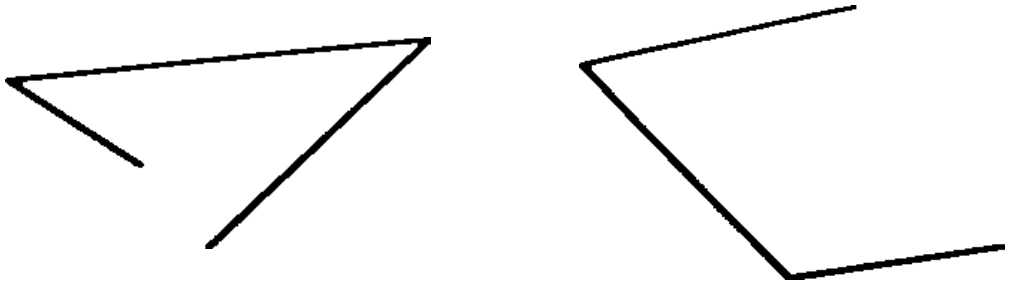

Figure 1.1: *Example for different quads*

two adjoining lines, will be referred to as the 'base' of the quad. The outer segments are going to be called 'arms'.

A quad has 6 degrees of freedom. There are 3 independent distance parameters: the length of the base and the two arm segments. There are also 3 unrelated angle parameters: the angles between each arm and the base, and the orientation of the quad.

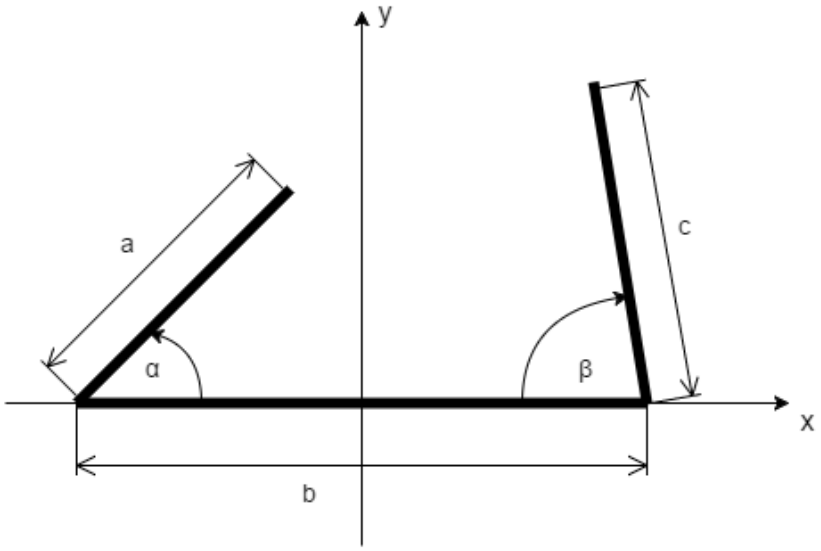

Figure 1.2: *Quad parameters*

Figure 1.2. shows the free parameters of a quad (the orientation is not shown on the image). The following notation is used:

a : The length of one arm

b : The length of the base

c : The length of the other arm

α : The angle between one arm and the base

β : The angle between the other arm and the base

γ : The angle with which the whole quad is rotated

For the sake of simplicity, figure 1.2. does not show the rotation with γ. The quad would be rotated around the origin of it's coordinate system.

The values of the length parameters are given in pixels, although they can be expressed in any unit of distance. The angles can be given is degrees or radians (in the implementation degrees are used for easier human readability).

$$a \in (0, a_{max}] \tag{1.1}$$

$$b \in (0, b_{max}] \tag{1.2}$$

$$c \in (0, c_{max}] \tag{1.3}$$

$$\alpha \in (0, 180^\circ) \tag{1.4}$$

$$\beta \in (0, 180^\circ) \tag{1.5}$$

$$\gamma \in [0, 360^\circ) \tag{1.6}$$

Equations (1.1) through (1.6) specify the range of each parameter. The maximum of the distance parameters are set by the space left on the image for the given marker, there is no theoretical limit for them. There is also no constraint for the resolution of the parameters. From the applications point of view, there are quads with continuous[1] and discrete parameter spaces.

1.1.1 Quad representation

There are several ways to represent quads, each with different advantageous properties. For this work multiple considerations were made in that regard. The most straightforward is to simply store the above mentioned parameters. This is simple and easy for human reading, which is great help in the development process.

A step forward from this is to norm the a, b and c parameter of the quad with the base segment's length. Then the following parameters are used:

s : marker size, the same as the base length

m_a : 'a' multiplier. $m_a = a/b$

m_c : 'c' multiplier. $m_c = c/b$

The α, β, γ angle parameters are not changed. This gives a scale or size parameter for the quad, which is useful for marker generation. These two representations are good for development and marker generation, but not so much for calculations.

[1] That is, only limited by the computational precision

A third option is to store the endpoints of the line segments. It requires the storage of 4 points: two endpoints (E_1, E_2) and two inner points (I_1, I_2). Equations (1.7) through (1.10) define the points' coordinates before rotating with γ, using figure 1.2.'s notation.

$$I_1' = (-\frac{b}{2}, 0) \tag{1.7}$$

$$I_2' = (\frac{b}{2}, 0) \tag{1.8}$$

$$E_1' = (-\frac{b}{2} + b * cos(\alpha), a * sin(\alpha)) \tag{1.9}$$

$$E_2' = (\frac{b}{2} - c * cos(\beta), c * sin(\beta)) \tag{1.10}$$

$$Rot(\gamma) = \begin{pmatrix} cos(\gamma) & -sin(\gamma) \\ sin(\gamma) & cos(\gamma) \end{pmatrix} \tag{1.11}$$

The point E_1, E_2, I_1, I_2 can be obtained from E_1', E_2', I_1', I_2' by a multiplication with the rotational matrix $Rot(\gamma)$.

That method is redundant for storage: it uses 8 parameters instead of 6. However this poses no practical problem in the scope of the project. The one outstanding benefit of this method is it's efficiency in calculations. Because it is based on points in a Euclidean space, linear algebraic methods (matrix multiplications) can be used for calculating the projective transformations.

In this project the second and the third options are used. The first, naive method is omitted because it has no considerable advantage over the other two. The second method, using the size, multiplier and angle parameters is used in marker generation. The third is used during the calculations and the recognition process.

1.2 Marker

Quads are 6 DOF shapes: in theory it would be enough to use only one of them for localisation and pose estimation. However that method would have very low error tolerance and questionable accuracy even in a best case scenario. To comply with the specifications written in the beginning of this chapter, the markers are put together from multiple quads. By placing quads with different orientations and sizes the error tolerance and accuracy can be greatly improved.

An intrinsic positive quality of using multiple quads with varying sizes is the scale invariance. As mentioned, even a single quad is sufficient for the task at hand. If the smaller quads become unrecognisable because of the low resolution or too large distance, a successful measurement is still possible. The same is true on the other end of the spectrum: if the observer is too close to the marker and the larger ones leave the field of view, the position and orientation can be calculated from the smaller quads.

Figure 1.3. shows an example for a marker. It is generated with the simple algorithm described in the next section, and is not optimal in many ways. Nonetheless it is functional, even if only a fraction of the quads are registered for the measurement.

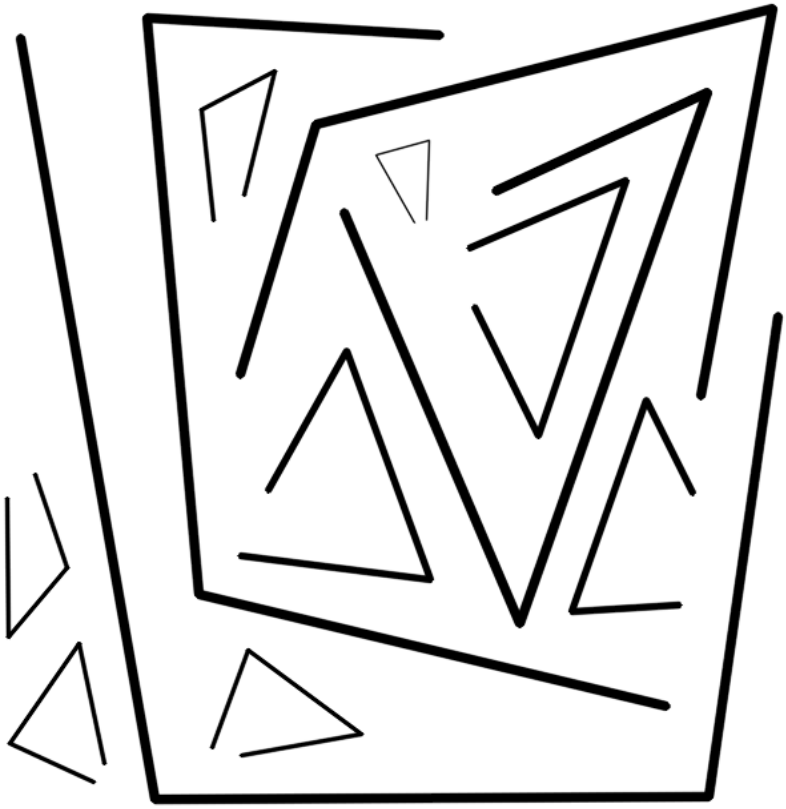

Figure 1.3: *An example for a marker*

The markers are going to be referred to as RQIM, which means Random Quad Image Marker. As the name suggests, the quads are randomly generated and placed on the markers.

Unless otherwise specified, RQIMs use quads with continuous parameter spaces. In this chapter there will also be a small introduction to discrete parameter markers and their potential applications.

1.2.1 Marker generation

In the current state of the project, markers are randomly generated using a simple algorithm. The generator routine receives the number of quads to be used in the current RQIM. The core concept is to create the desired number of random quads and place them on the image.

Let the number of quads to generate be n. First, the quad sizes are picked. There is an upper and a lower limit for them, given in percent of the image size. The generated sizes

follow an exponential distribution:

$$s = e^{-x*f} \tag{1.12}$$

Where s is the quad size, x is random number between 0 and 1 with uniform distribution, and f is a scale factor. Then the n sizes are ordered in descending order.

After the scale factors are picked, the whole quads are generated by the following method. A random quad is created with the first (the largest) scale and placed on the image. Then another quad is created with the next largest size. After every new quad a check is performed whether or not it can be placed on the marker. If it cannot, then a new quad is generated with the same scale factor until it can be placed or the algorithm reached the limit of retries.

With this simple logic n quads are placed on the RQIM and the creation process is finished. Below is the pseudo-code of the algorithm.

```
n_max = number of quads to create
f = scale factor for exponential distribution
lowlim = lower size limit
uplim = upper size limit
n = 0
while n < n_max
        size = exp(-rand() * f)
        if size > lowlim and size < uplim
                store size
                n = n+1
        endif
end
sort(sizes, descending)
n = 0
while n < n_max
        while quad placed or max tries
                quad = create_random_quad(sizes(n))
                if quad can be placed
                        place quad
                        n = n+1
                end
        end
end
return marker
```

This method is not optimal and is based on trial and error, but it gives usable markers for the development process.

1.2.2 Discrete RQIM

There are experiments in progress with discrete parameter space quads. It may be advantageous to quantize the parameter space in order to decrease the error probability in the pose estimation process.

Quads with finite possible states can be stored using much less resources than their continuous counterpart. As an example let us take a look at the following quantisation.

Angles: $15°, 30°, 45°, 60°, 75°, 90°, 105°, 120°$

Multipliers: $0.40, 0.60, 0.80, 1.0, 1.25, 1.50, 1.75, 2.0$

Orientations: $0°, 22.5°, 45°, 67.5° \ldots 270°, 292.5°, 315°, 337.5°$

Sizes: $1, 0.8, 0.6, 0.5, 0.4, 0.3, 0.25, 0.2, 0.1, 0.08, 0.06, 0.05, 0.04, 0.03, 0.025, 0.02$

In this example, there are 8 possible values for the angle parameters, also 8 for the multipliers, 16 for the orientation and also 16 for the sizes. If the possibilities are stored in a lookup table, it is enough for the quad to store the index at which the value is accessible. A quad is defined by two angle parameters, two multipliers, an orientation and a size. The angles (in this case) require at least 3 bits each, the multipliers also. The orientation and the size need 4 bits each. That gives a sum of 20 bits per quad, which is significantly less than the space required to store 6 floating point numbers per quad.

This 20 bit word is also usable as an ID for the quad. It may be possible to code information in these ID-s, so the marker could provide additional information. This information could be related to and used by the localisation process, or be totally unrelated, general data. These possibilities have not yet been extensively researched.

The discrete RQIMs are usually less dense than the continuous ones, due to the limited angle possibilities. This means fewer quads per marker, which leads to decreasing redundancy. An optimum must be found between the number of quads per RQIM and distance between quads in the parameter space.

Chapter 2

Marker recognition

In this chapter there will be a summary of the image processing algorithms tried and used for the recognition of the fiducial markers. From a computer vision point of view the task is to detect joint line segments. This is a relatively easy task in image processing, there are many well tried algorithms for it.

In this chapter will be a short summary of the algorithms used for testing and performance comparison. The preprocessing steps used for preparing the images for the line fitters (segmentation, thresholding, filtering etc.) will also be discussed.

The general flow of processing is the same for every line fitting solution. The input is the raw image taken[1] by the observer. The first problem is finding the RQIM on the picture. When the marker area is located, it is necessary to discard the only partially visible and/or unrecognisable quads. At this point there is an image or set of images containing potentially good quads. The process here diverges depending on which line fitting algorithm is used. They all need differently conditioned input images for optimal performance. The line fitter routines not necessarily have the same output format[2], so conversion may be needed. This is the end of the marker recognition phase. This step of the process takes the raw input image and initiates quad structures based on the observed picture.

Four separate line fitters are profiled in this experiment.

- Hough-transformation
- Corner detection
- Skeletoning detector
- Gradient detector (enhanced Hough transformation)

[1] In the development phase there were rendered pictures used for better repeatability

[2] Some return line segments defined by their endpoint, others use the polar representation of a lin, etc.

The first one uses the classic Hough-transformation for line detection. In the OpenCV framework there is another, probabilistic implementation of the transformation. It will also be tested.

The second detector is based on corner recognition. There are more variants of this method to try out, too. The corner metrics of a feature can be calculated differently with (Harris metric, eigenvalues, etc.) varying results. It is also needed for the solution to be scale invariant, which also can be achieved in a number of ways.

The third alternative uses skeletoning for line detection. It is a RANSAC-like method, which makes it very robust, though quite computationally expensive. The core concept is to thin the observed marker (band) down to single pixel lines and try to find two point with the most inliers.

The gradient detector is a bit Hough-like in it's nature. It uses the image gradient to calculate the angle in the Hough-space, and the pixels only vote in their distance parameter.

A typical marker shot with partial visibility is shown figure 2.1..

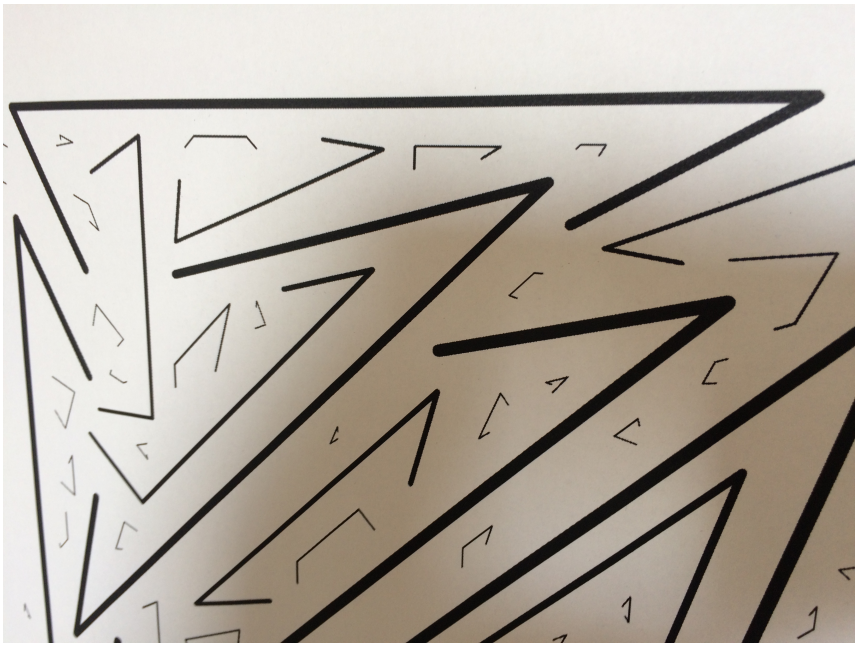

Figure 2.1: *Partially visible marker (taken with commercial smartphone)*

2.1 Preprocessing

The preprocessing is done in two stages. The first is the segmentation, when quad-like blobs are found. The second step is the preparation of the aforementioned blobs for the line fitting algorithms' needs.

2.1.1 Segmentation

The segmentation process is carried out on images roughly like the one shown in figure 2.1.. First the photos are converted to binary format by applying a threshold. The image is inverted in the process, because it makes more sense for the objects to be marked with non-zero elements than vice-versa. The threshold's value is determined using Otsu's method, which maximises the inter-class variance of the clusters[3]. The implementation is provided by the OpenCV framework.

Afterwards, the binary image is conditioned with a *close* morphology operator. The closing removes the gaps from the large connected areas (possible quads) and removes the *salt and pepper*-like noise. In the current implementation the kernel size of the morphology operator is constant, however it could be beneficial to calculate it from the global or local image parameters[4].

The segmentation is based on finding continuous contours on the binary image. The OpenCV framework provides great functionality for this. The implementation is based on calculating the 8-neighbour chain code for the binary blobs on the image. The functions returns a list of list of points for the borders os each distinct contour.

The next step is the filtering of the found blobs. First the surely partial quads are discarded. This is done by calculating the bounding box of the contours, and if one of it's sides are touching the image border, the blob is marked as partial. With this approach it is possible that some fully visible quads that only touch the image border with one of their corner are lost. This problem can be easily fixed by checking the neighbourhood of the contact point, but this is not yet implemented.

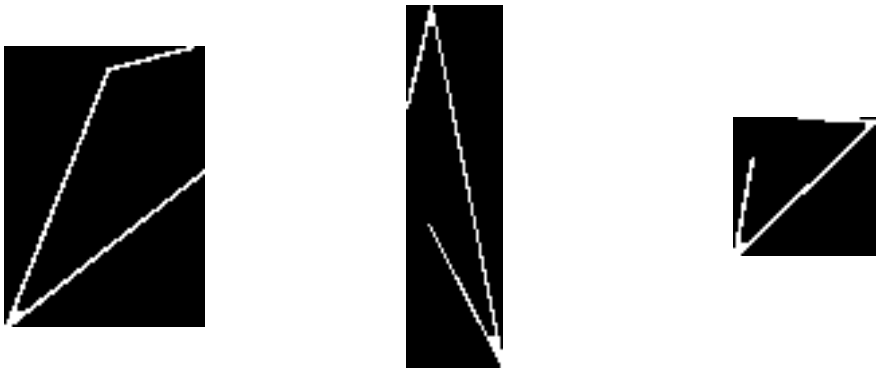

Figure 2.2: *Quad candidates after segmentation*

The next blob-filtering step is to filter out the false-positive contours. These false hits can be caused by the light conditions or the scene around the marker. For this purpose a simple metric is used to measure how likely a blob is to be a quad. This metric is the ratio of a blob's area and circumference. By experimentation this ratio for quads is found to be in the range of 10 and 50. The contours with ratios outside these limits are discarded.

[3] Foreground and background

[4] e.g. image size, area of the connected region, etc.

The segmentation processes output is available as a singe image with colour-coded[5] blobs or as a list of separate images each containing a quad candidate. Figure 2.2. shows an example for the output of the segmentation process.

2.1.2 Conditioning

Before running the line detection algorithms some conditioning steps are done in order to improve their effectiveness. These processes are not uniform - every fitting method needs it's own.

The Hough transformation traditionally works best on thin lines. The easiest way is to generate an edge image with high-pass filtering. The OpenCV framework offers a wide variety of features for this task. The best result are obtained by using the Canny edge detector.

The skeletoning detector does not need a conditioning step, as it performs the band thinning on it's own.

The methods based on image gradients and corner detection both require some level of smoothing on the picture. In case of the corner detection the smoothing is useful for removing the false positive matches caused by the jagged edges, or in case os JPEG images the artefacts caused by the compression. The gradient detector simply gives a more spread-out and easier to analyse result on a smoothly changing gradient than it would on strict edge. The smoothing is also implemented using the OpenCV framework, which provides easy access to Gaussian filtering. The OpenCV implementation is based on convolution with a configurable Gaussian kernel. The kernel size and the deviation in x and y direction can be set. The Gaussian kernel and the convolution itself is handled in th framework.

2.2 Line fitters

2.2.1 Hough transformation

2.2.2 Gradient detector

2.2.3 Corner detector

2.2.4 Skeletoning detector

[5] Gray level, to be exact