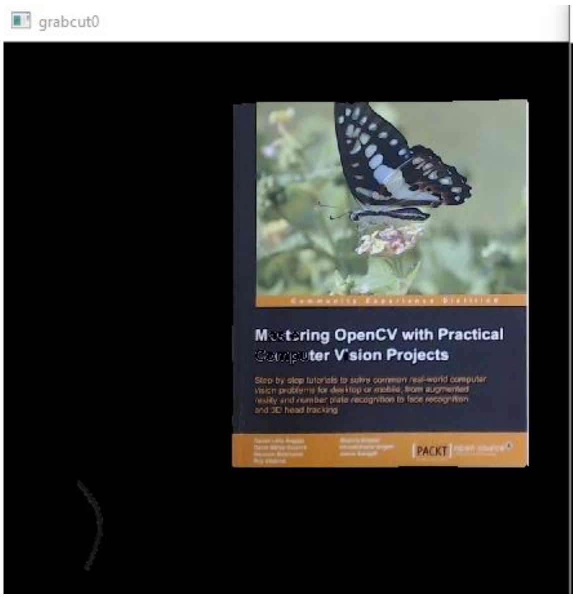
Computer Vision

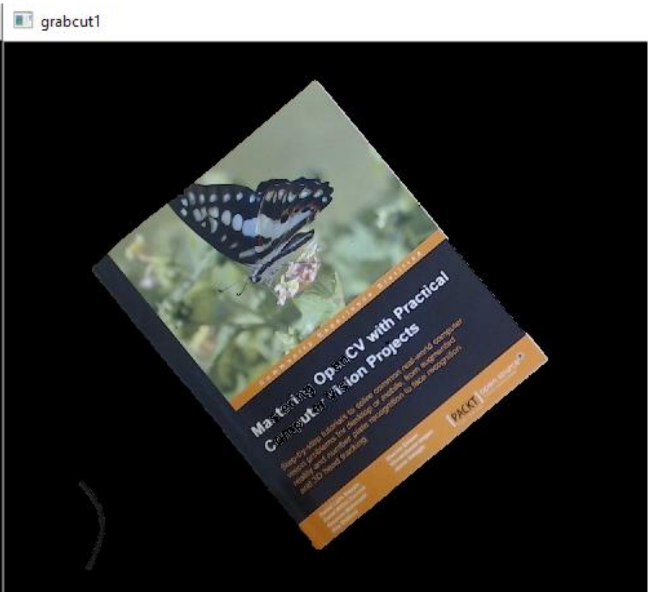
Assignment: 4
Author(s): Rafał Grasman, Bas Woudstra
Date: 28-09-2018
Subject: Image Transformations
Required pre-knowledge: OpenCV, OpenCV documentation, previous assignments (including used code)

Contents

Task 1, 2 and 3

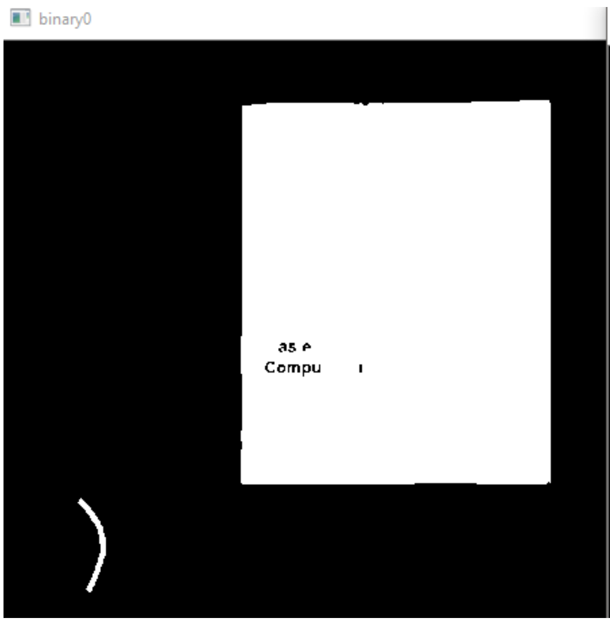
For task 1, to determine the 4 points of the book the background has been removed by using grabCut function from opencv:

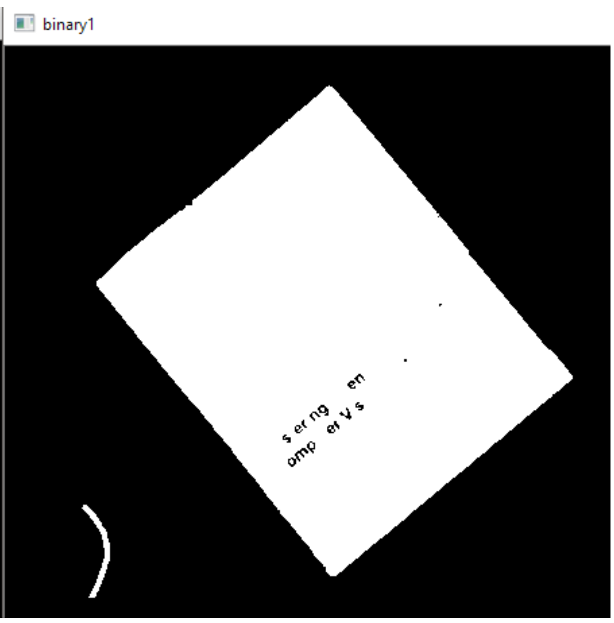




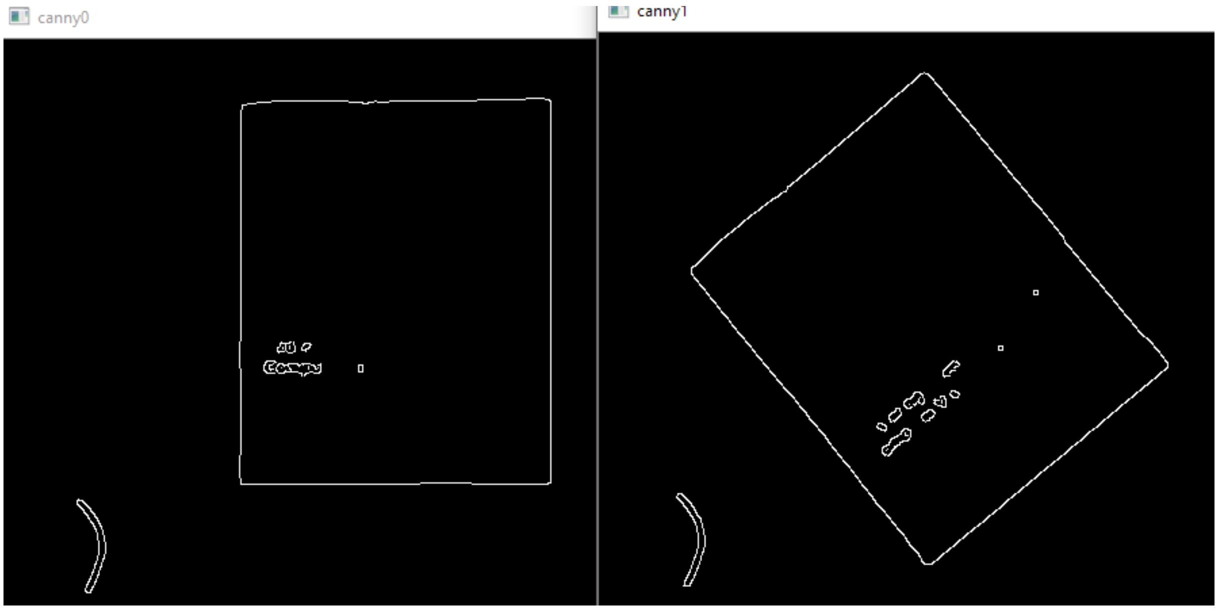
```
		dst[i] = Mat::zeros(src[i].size(), src[i].type());

		Rect rect;
		rect.x = ((double)src[0].size().width) * 0.075;
		rect.y = ((double)src[0].size().height) * 0.050;
		rect.width = ((double)src[0].size().width) * 0.50;
		rect.height = ((double)src[0].size().height) * 0.900;

		grabCut(src[i], mask, rect, bgdModel, fgdModel, 1, GC_INIT_WITH_RECT);
```

Then the image is converted to a binary image:



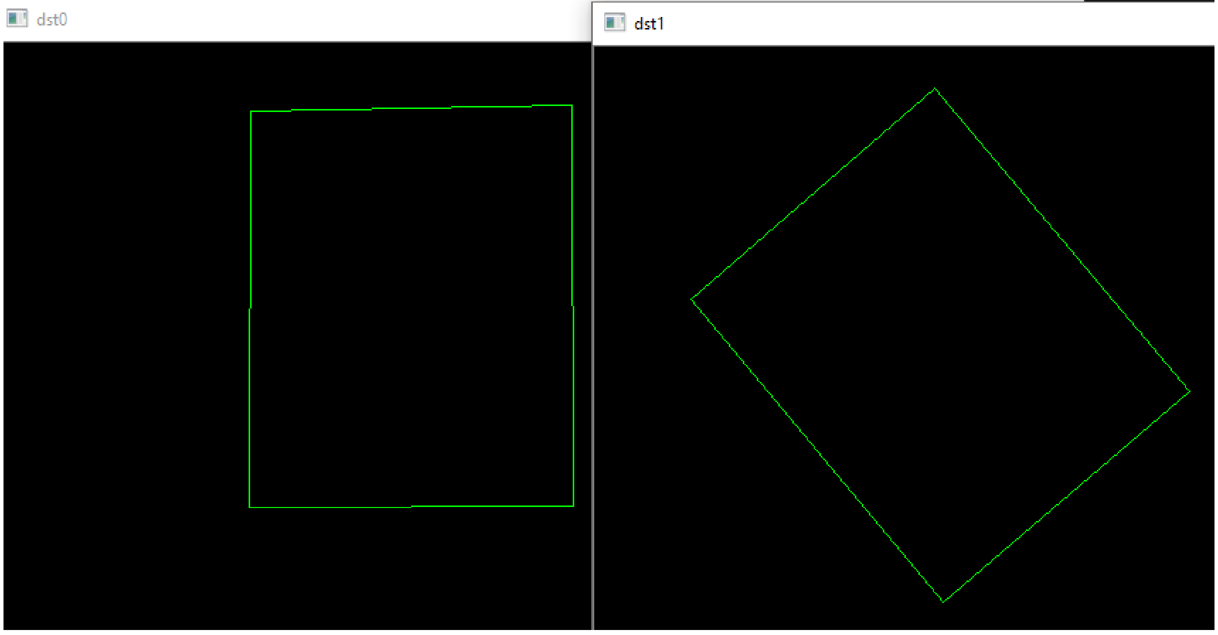


Then it is blurred and canny edge detection is performed:



After that the `findContours` with `approxPolyDP` functions are used to get the book rectangle per image:

```
cv::findContours(canny.clone(), contours, CV_RETR_EXTERNAL, CV_CHAIN_APPROX_SIMPLE);
//(…)
cv::approxPolyDP(cv::Mat(contours[j]), approx, cv::arcLength(cv::Mat(contours[j]),
true)*0.02, true);
//(…)
drawContours(dst[i], contours, contours.size() - 1, Scalar(0,255,0));
```



After which the transformation can be calculated from the points:

```
Mat transform = getAffineTransform(rectangles[0].data(), rectangles[1].data());
std::cout << transform << std::endl;
```

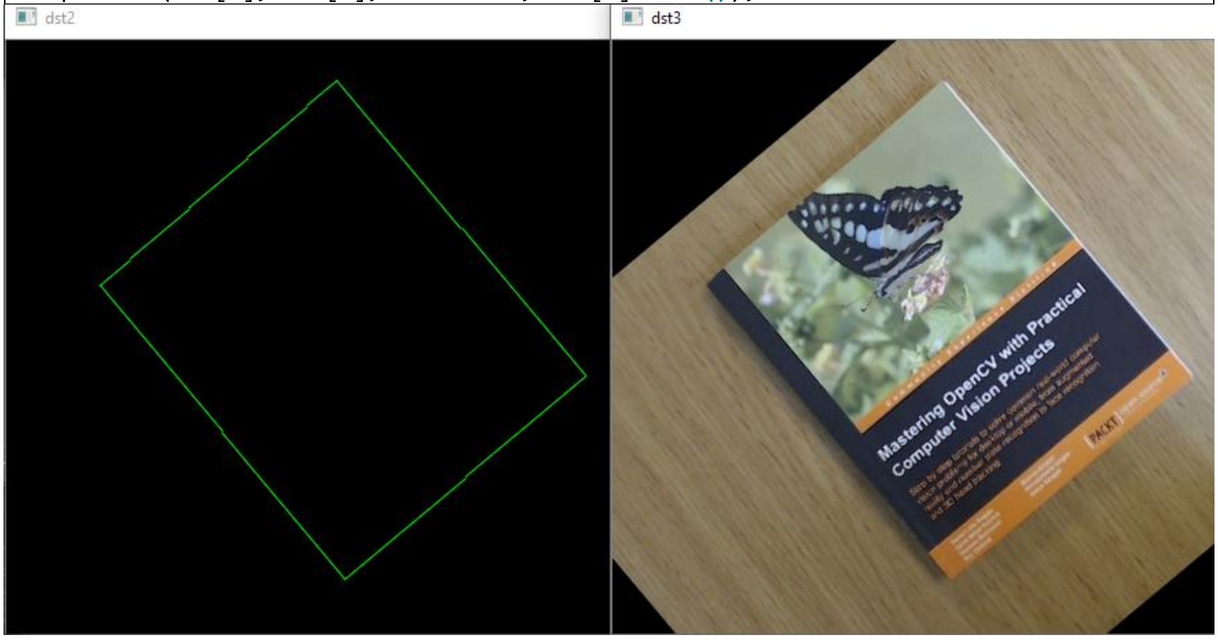
Which yields:

[**0.7689596576247719, 0.6384605702058028**, *-100.2898860847127*;
 -0.6429605387374916, 0.7619894266473659, *256.5144439549375*]

With the rotation, sheer and scaling data being **bold** and translation data being *italic*.

The calculated transformation matrix can be re-applied onto the source image to check if the transformation yields the destination image:

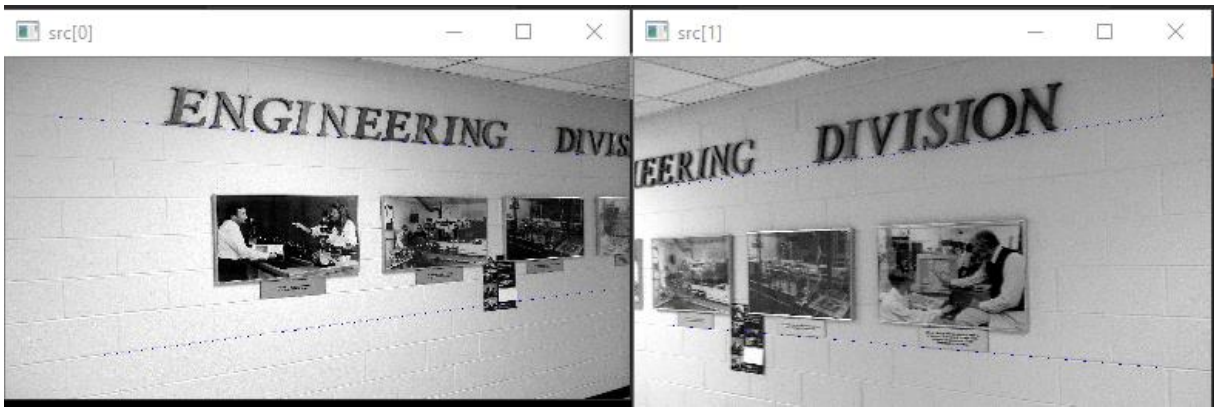
```
dst[2] = Mat::zeros(src[0].size(), src[0].type());
dst[3] = Mat::zeros(src[0].size(), src[0].type());
warpAffine(dst[0], dst[2], transform, dst[2].size());
warpAffine(src[0], dst[3], transform, dst[3].size());
```



With getPerspectiveTransform all 4 points can be used to get a 3x3 transformation matrix, which will be shown in the next task.

Task 4

The original images have a dotted line drawn (by mouse selected points) in the order from top-left to top-right and then bottom-right to bottom-left:



After those points are known they are saved and the transformation matrix is calculated by making the y position of the lines the same (so same height makes the line straight on screen):

```
		rectangles[i].target.clear();
		for (size_t j = 0; j < 4; ++j)
		{
			Point2f point = rectangles[i].source[j];
			if (j % 2 == 1)
			{
				point.y = rectangles[i].source[j - 1].y;
			}

			rectangles[i].target.push_back(point);
		}
```

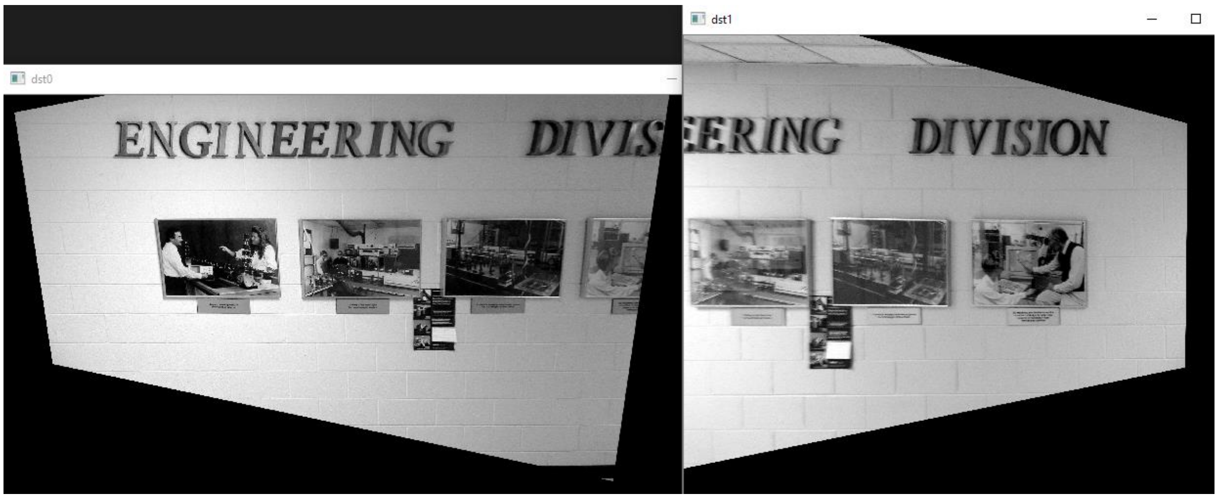
After that the transformation is calculated:

```
Matx33f transform = getPerspectiveTransform(rectangles[i].source.data(),
rectangles[i].target.data());
```

And then the image is warped and shown:

```
		warpPerspective(src[i], dst[i], transform, dst[i].size());

		namedWindow("dst" + std::to_string(i), CV_WINDOW_NORMAL);
		imshow("dst" + std::to_string(i), dst[i]);
```

As can be seen in the following picture the perspective transformation has straightened out the pictures:

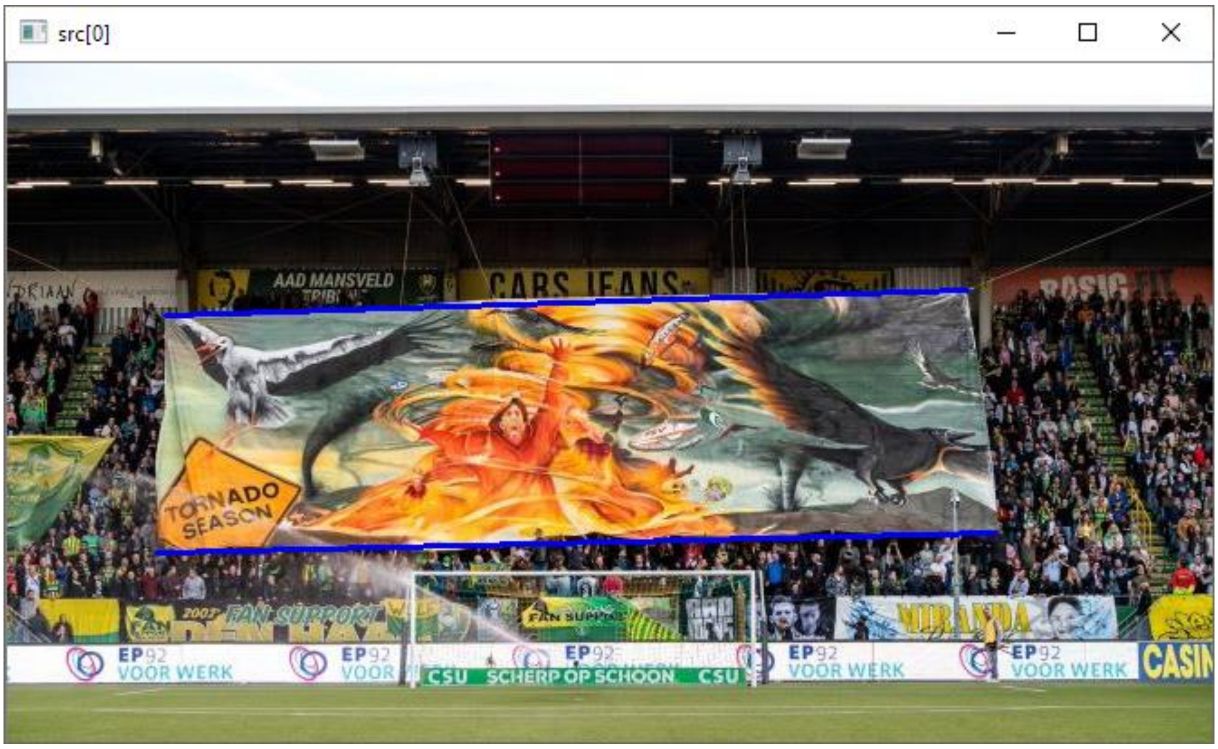


The calculated 3x3 transformation matrixes are:

```
transform0:
[0.510548, 0.096822344, 40.219765;
 -0.14131102, 0.78065056, 81.195084;
 -0.00022838234, 7.5927703e-05, 1]
transform1:
[2.0013189, 0.0007334752, -118.19073;
 0.48749441, 1.3977752, -257.98288;
 0.00044634411, 5.8678015e-06, 1]
```

Task 5

First the outlines of the banner that needs to be replaced are selected:



After which the saved points (in order top-left, top-right, bottom-right, bottom-left) are transformed into a straight rectangle:

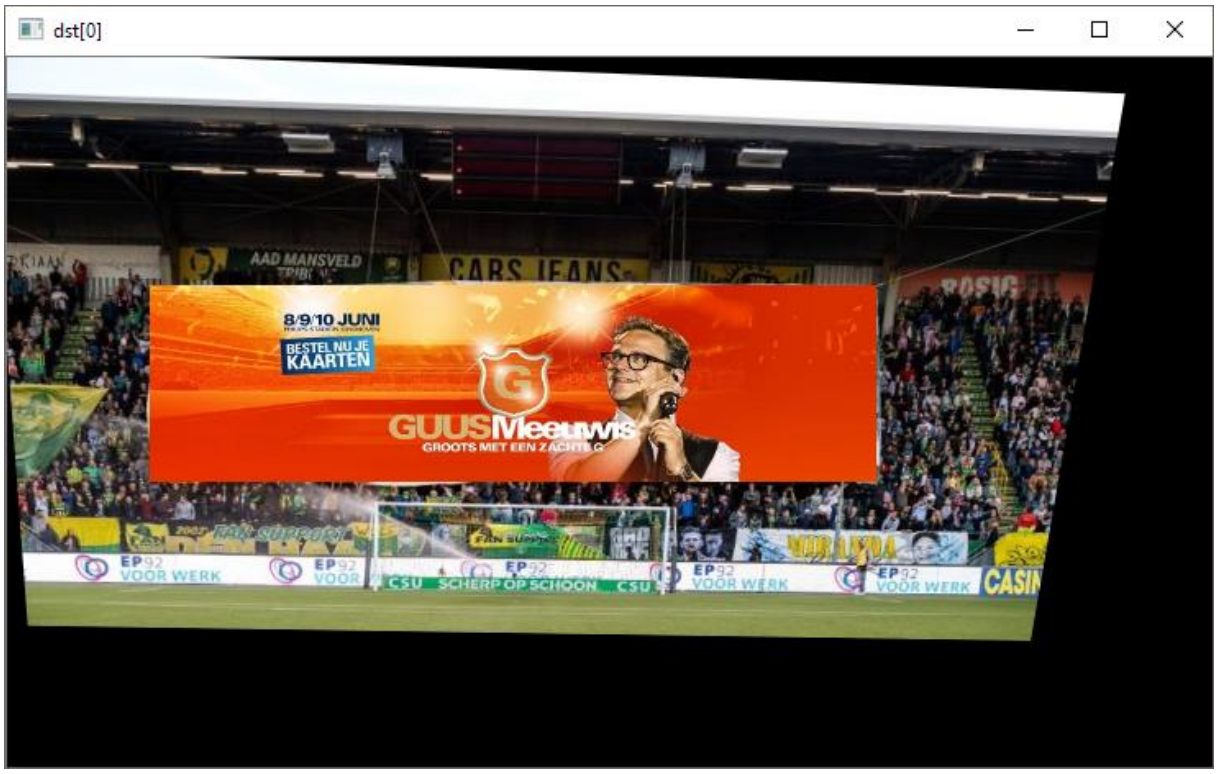
```
std::vector<cv::Point2f> transformfix = rectangles;
transformfix[1].y = transformfix[0].y;
transformfix[3].y = transformfix[2].y;
transformfix[3].x = transformfix[0].x;
transformfix[2].x = transformfix[1].x;
```

This is then used to get the perspective transform:

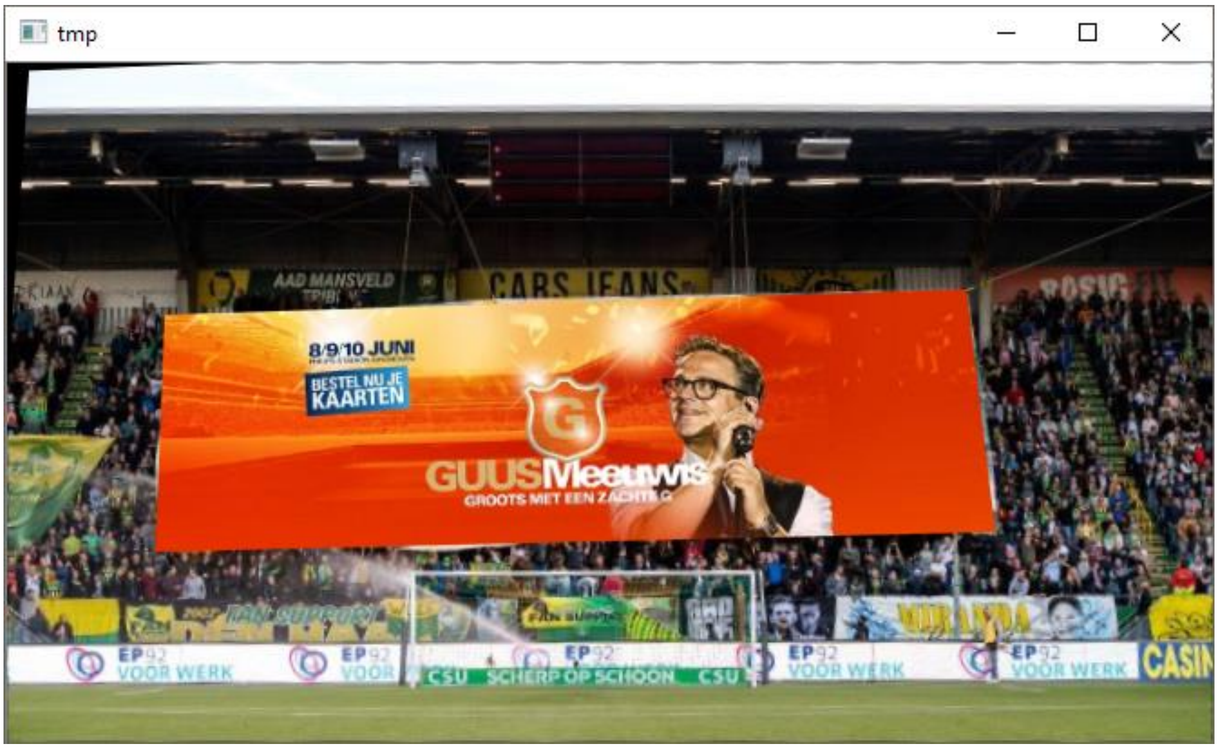
```
Matx33f transform = getPerspectiveTransform(rectangles.data(), transformfix.data());

std::cout << "transform:" << std::endl;
std::cout << transform << std::endl;
```

```
transform:
[1.09539, 0.074461691, -13.501223;
 0.042667266, 1.0690696, -5.0879583;
 7.5850279e-05, 0.00037821592, 1]
```

Then the image is "straightened" out and at the top-left point the new (and resized to the rectangle) banner is inserted:



After that the inverse transform is applied to get the original image back with the inserted banner:



The steps can also be done differently to get rid of the artefact in the top-left corner, for example applying the transform to the banner and putting that onto the image.

The code to perform the above steps:

```
dst[1] = Mat::zeros(src[1].size(), src[1].type());
resize(src[1], dst[1], Size(abs(transformfix[1].x - transformfix[0].x),
abs(transformfix[3].y - transformfix[0].y)));

namedWindow("dst[1]", CV_WINDOW_AUTOSIZE);
imshow("dst[1]", dst[1]);

dst[1].copyTo(dst[0](cv::Rect(transformfix[0].x, transformfix[0].y, dst[1].cols,
dst[1].rows)));

namedWindow("dst[0]", CV_WINDOW_AUTOSIZE);
imshow("dst[0]", dst[0]);

Mat tmp(Mat::zeros(src[0].size(), src[0].type()));
warpPerspective(dst[image], tmp, transform.inv(), tmp.size());

namedWindow("tmp", CV_WINDOW_AUTOSIZE);
imshow("tmp", tmp);
```