Image Matching

Date: 10-10-2018

Author(s): Rafał Grasman, Bas Woudstra

Table of content

Task 1: Manual Selection Stitching

For task one the operator will select at least 4 matching points on each image. Then all images are stitched together according to the points selected. To make it easier to use the program some points have been preselected.

```
points1 {Point(822,685), Point(1030,690), Point(1140,450), Point(995,66), Point(820,95)};
points2 {Point(199,692), Point(405,692),  Point(518,468),  Point(427,95), Point(257,90),
         Point(907,700), Point(1113,700), Point(1063,103), Point(889,101),Point(785,470)};
points3 {Point(221,672), Point(416,660),  Point(391,94),   Point(223,69), Point(104,444)};
```

However more points can be selected. With a double click on the image the position will be added to the list, after selecting a second matching point on the second image.

```
static void onMouse (int event, int x, int y, int flags, void* param)
{
    if(event != EVENT_LBUTTONDBLCLK)
        return;

    std::vector<Point>* points = (std::vector<Point>*)param;
    nPointsSelected ++;
    points->push_back(Point(x,y));

    processImage();
    displayGraphics();
}
```

When the points are selected they are split in two parts. Here For a requirement is that the left image is selected first and then the right image. Otherwise point mismatches will occur.

```
if(points1.size()+points3.size() == points2.size())
{
    for (int i = 0; i < points1.size(); i++)
    {
        center1.push_back(points2[i]);
    }
    for (int i = 0; i < points3.size(); i++)
    {
        center3.push_back(points2[i+points1.size()]);
    }
}
```

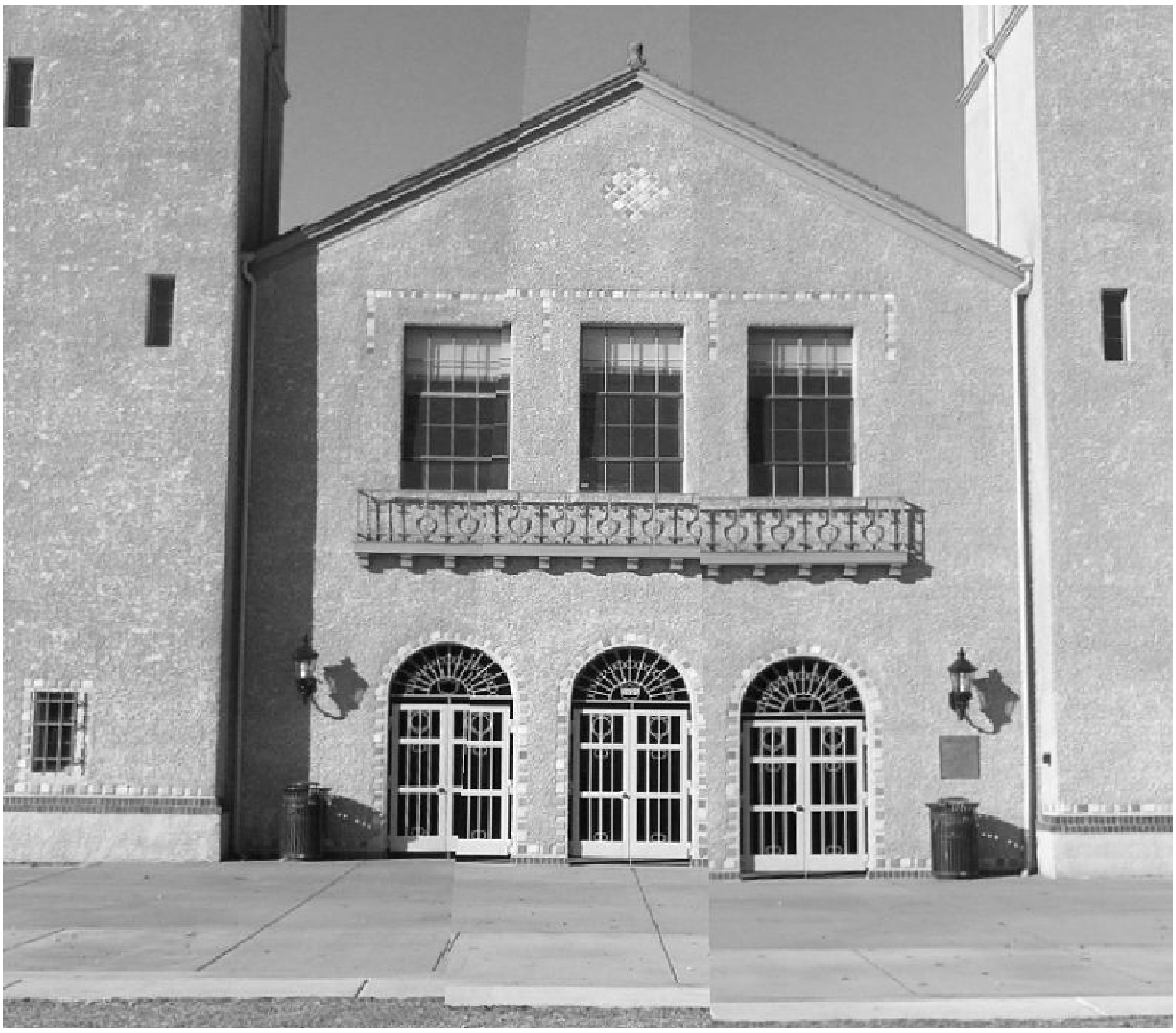
When the positions are split a offset is added to the center positions. This offset will make sure the image is moved more to the center and not cut off on the left side of the screen.

```
Point tar = Point(imgWidth,0);

for(auto& i: center1) i += tar;
```

Then the homography is found and the image is warped into its final position.

```
Mat leftMatch  = findHomography(points1, center1, 0);

warpPerspective(frame2, result, centMatch,  size, INTER_LINEAR, BORDER_TRANSPARENT);
```

Result:
the result is a nicely stitched image, if the points need to be reselected. Some misalignment is visible, but thats because of inaccuracies of the selected points.



Task 2: Automatic Stitching

For task 2 the selection is done automatic. This is done with SURF. Firstly the key points and descriptions are found.

```
std::vector<KeyPoint> keyPoints[nPics];
Mat discriptors[nPics];

for (int i = 0; i < nPics; i++)
{
    int minHessian = 400;
    Ptr<SURF> surfer = SURF::create(minHessian);
    surfer->detect(frame[i], keyPoints[i]);

    surfer->compute(frame[i], keyPoints[i], discriptors[i]);
}
```

When all points are found the matches are selected. This is done with every picture compared to the middle image (frame[1]).

```
for (int i = 0; i < nPics; i++)
{
    BFMatcher matcher;
    std::vector< DMatch > matches;
    matcher.match(discriptors[i], discriptors[1], matches);
```

To find the good matches the minimum distance of a match is found and then that distance is multiplied by 3 to select all accurate matches. The +0.000001 is added to include the matches of the image which have a very small distance but bigger then 0, (the minimum distance).

```
for (int j = 0; j < discriptors[i].rows ; j++)
{
    if(matches[j].distance < 3*minDist+0.000001)
        goodMatches.push_back(matches[j]);
}
std::cout<<"min distance: "<< minDist << std::endl;
std::vector<Point> pointsRef;

for (int j = 0; j < goodMatches.size(); j++)
{
    points[i].push_back(keyPoints[i][goodMatches[j].queryIdx].pt);
    pointsRef.push_back(keyPoints[1][goodMatches[j].trainIdx].pt);
}
```

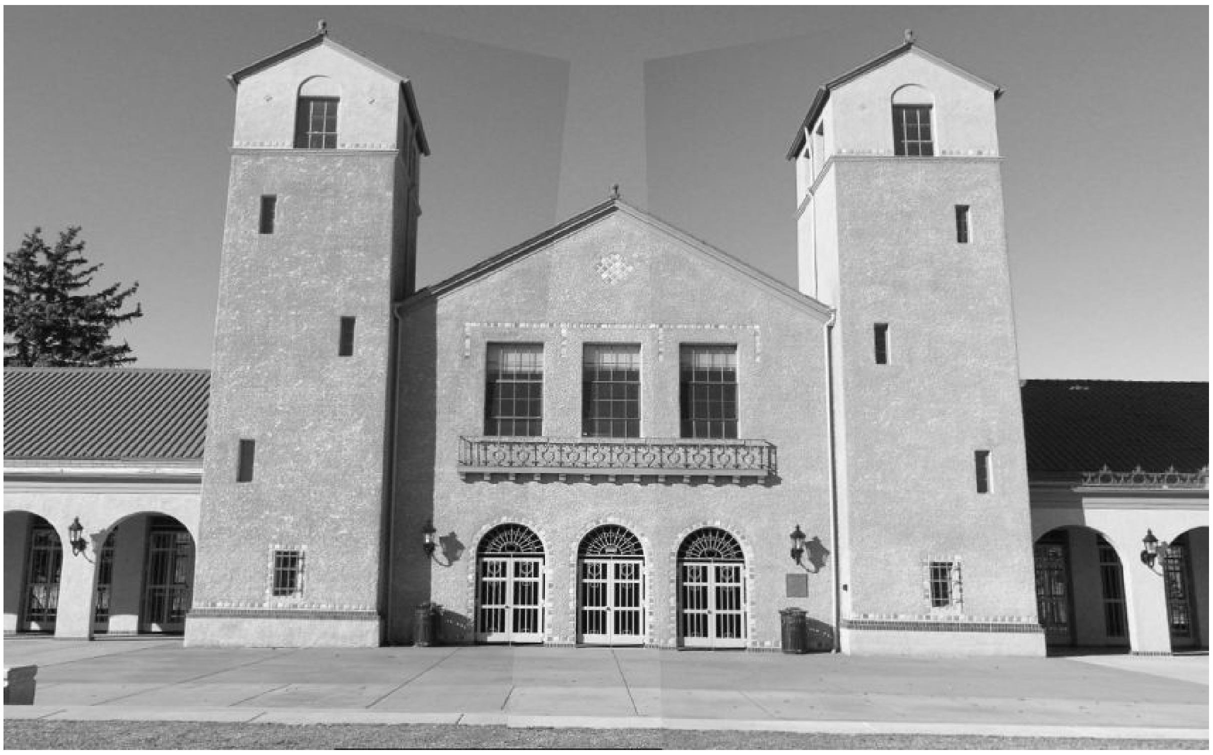
The reference points are translated to the right by the image width and then the homography is found.

```
Point tar = Point(imgWidth,0);
for(auto& j: pointsRef) j += tar;
transformMatrix[i] = findHomography(points[i], pointsRef,CV_RANSAC);
```

Then the image is warped to be stitched correctly.

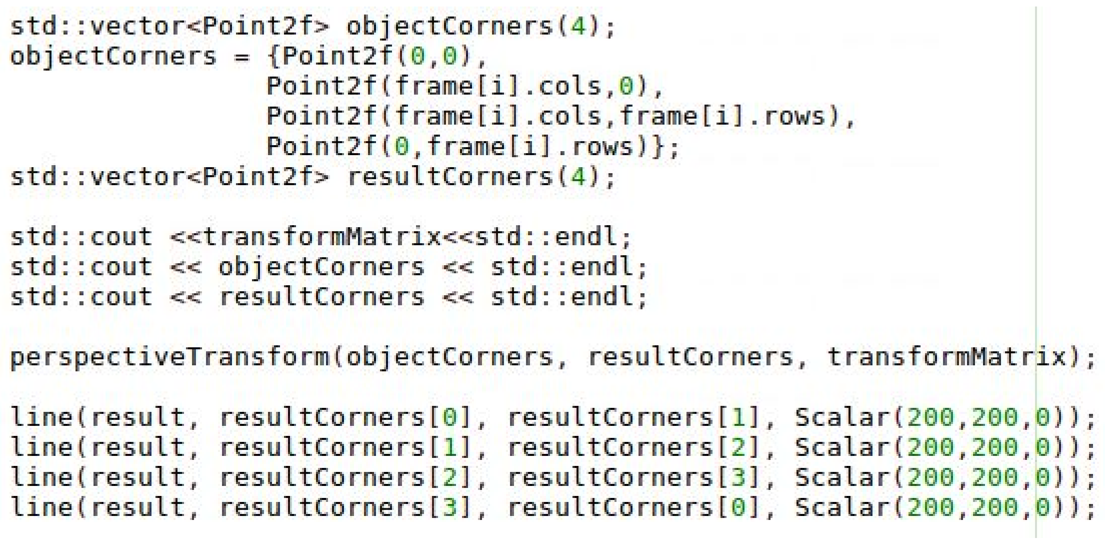
```
warpPerspective(frame[1], result, transformMatrix[1], size, INTER_LINEAR, BORDER_TRANSPARENT);
warpPerspective(frame[0], result, transformMatrix[0], size, INTER_LINEAR, BORDER_TRANSPARENT);
warpPerspective(frame[2], result, transformMatrix[2], size, INTER_LINEAR, BORDER_TRANSPARENT);
```

Result:
The result is a near perfect stitching of the image.



Task 3: object detection

For the 3rd task the same is done as in task 2, but instead of warping some points are warped. And the points are used to create a square around the object.



```
std::vector<Point2f> objectCorners(4);
objectCorners = {Point2f(0,0),
                 Point2f(frame[i].cols,0),
                 Point2f(frame[i].cols,frame[i].rows),
                 Point2f(0,frame[i].rows)};
std::vector<Point2f> resultCorners(4);

std::cout <<transformMatrix<<std::endl;
std::cout << objectCorners << std::endl;
std::cout << resultCorners << std::endl;

perspectiveTransform(objectCorners, resultCorners, transformMatrix);

line(result, resultCorners[0], resultCorners[1], Scalar(200,200,0));
line(result, resultCorners[1], resultCorners[2], Scalar(200,200,0));
line(result, resultCorners[2], resultCorners[3], Scalar(200,200,0));
line(result, resultCorners[3], resultCorners[0], Scalar(200,200,0));
```

Result:
The result is 2 squares around the detected objects.