How to generate 3D models with the MOPED modeling tool

Projektarbeit

im Fachgebiet Informatik

vorgelegt von:	Janos Knobloch
Studienbereich:	Informatik
Matrikelnummer:	21673
Betreuer:	Tobias Fromm

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1. Introduction

One goal of of the AMSER project at Hochschule Ravensburg-Weingarten is to detect objects to interact with them. Since there is no easy method to distinguish between all sorts of objects, different approaches are taken. The MOPED system, developed at the Carnegie Mellon University, Pittsburgh, tries to detect objects based on the SIFT algorithm (Scale invariant feature transform). It has the attribute to recognize objects in realtime. But before the system can actually do real-time recognition, the objects that should be detected have to be recorded, converted into a digital model and saved in a database. This document gives a introduction on how to use the MOPED modeling tool to get good models of real objects, which then can be detected by the MOPED system.

2. Create 3D models with MOPED

In the following chapters it is assumed, that the MOPED installation has been successfully executed and MATLAB is installed. To use all the necessary commands from MOPED in MATLAB, the MEX files have to be compiled as described in the MOPED instructions.

2.1. Procedure

2.1.1. Getting the camera calibration parameters

MOPED requires the calibration parameters of the camera that is used to take the pictures for the 3D model. These parameters can be obtained using the OpenCV calib 3D module[1]. In order to get the values, the module needs about twenty images of a checkerboard from different angles taken by the camera. To use the software, follow the given instructions:

First, create a folder *checkerboard* and copy the images taken by the camera to the new folder.

```
1 cd /calibrate_path
2 rename 'y/A-Z/a-z/' checkerboard/*
3 ls /calibrate/checkerboard/*.jpg > imgs.txt
4 ./calibrate -w 6 -h 9 -o camera.yml -op -oe imgs.txt
```

[1] The required source code can be found at https://code.ros.org/trac/opencv/browser/trunk/opencv/samples/cpp/calibration.cpp?rev=3215. Compile the file and name the executable 'calibrate'

The names of the images have to be written to the *imgs.txt* file. In the next step, the tool will generate the camera parameters from the pictures. If all was done right, the tool will show the processed pictures as in figure 2.1.

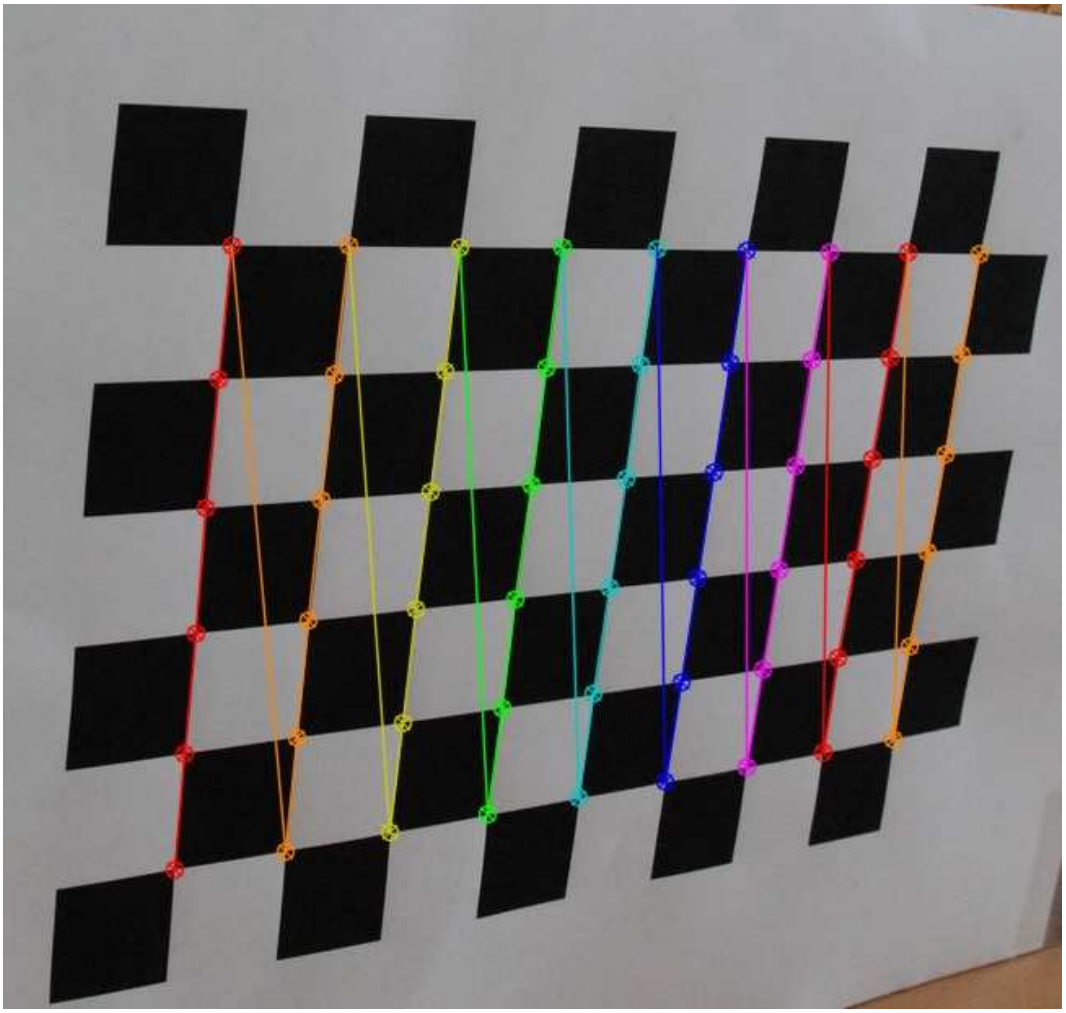

Abbildung 2.1.: Calibration tool processing a checkerboard image

As output, the files camera.xml and distortion.xml will be generated. The data values in the files have to be copied to MATLAB. To set the KK matrix write KK = [a c b d], where a-d are extracted from the camera.xml file as following:

<data> a 0 b
0 c d
0 0 1 </data>

The same has to be done with the parameters in the distortion.xml file. The values are given as following:

<data>a b c d</data>

To set the distortion matrix, write kc=[a b c d] in MATLAB. The matrices should be stored in a .mat file, so they can be easily restored later.

2.1.2. Recording the pictures

MOPED uses a software called Bundler to get a three-dimensional representation of the given pictures. Since Bundler is using a technique similar to panorama software to get a 3D model, the pictures taken have to meet some requirements.

- The quality of the pictures has to be sufficient. Especially small text or other small features of the recorded object should not be blured.
- The background should not be blurred. In order to avoid a blurred background, use a vertical boundary around the object or try to optimise the camera settings.
- The background and the ground should be textured. Bundler has to find many different characteristics to fit the images properly. A good way to achieve this is to plot a big panorama picture and put it around and respectively under the object. This also helps to avoid a blurred background.
- Use a small white paper under the object that has to be recorded.

It is absolutely necessary that the background does not move relative to the recorded object. It is impossible for Bundler to match the recorded images if, for example, the camera is always in the same position and only the object is turned around. If the object is moved, the background and underground have to be moved, too. In order to take pictures that Bundler can use, either the camera has to be moved around the object or there has to be some kind of rotatable platform, where the object is standing on (see Attachment A.1).

For a good model, according to the MOPED documentation, 40 to 60 pictures should be sufficient. However, while testing the modeling tool it turned out that sometimes there is missing a part of the model. In this case, taking about 100 pictures solves the problem.

2.1.3. Processing the pictures with MATLAB

Before using MOPED to create a 3D model, the filenames of the pictures have to be converted into lowercase. Also, the image size should be reduced. In chapter 4 the different sizes and their influence on the result are discussed.

```
cd moped/modeling/examples/salt4/
rename 'y/A-Z/a-z/' *
mogrify -resize 1600x1200 *.jpg
```

After doing the preperation steps, start MATLAB and use the following code to create a three dimensional model.

```
OBJECT_NAME='salt4'
cd /home/username/-mopedpath-/moped/modeling;
addpath(pwd);
```

Load the camera calibration parameters and set the distortion matrix kc to 0. This is usefull, if the camera used has no distortion. Otherwise the original parameters should not be changed.

```
load nikon_nozoom_calib.mat;
kc = [0 0 0 0];
img_dir = strcat('/home/tobias/moped/modeling/examples/',
    OBJECT_NAME);
out_dir = strcat('/home/tobias/moped/modeling/examples/',
    OBJECT_NAME, '_out');
```

Correct the distortion of the images. To resize the images AND the masks, use 'imundistort_folder' with the image size (height x width) parameter. This might be useful if the model should be calculated twice, with different resolutions.

```
1
2 imundistort_folder(img_dir, KK, kc);
3 // imundistort_folder(img_dir, KK, kc, [1071 1600]);
```

Draw_mask opens a new window to draw a mask around the object. Double click or right click -> Create mask to continue with the following image.

```
1 draw_mask(img_dir);
2 cd (img_dir);
```

Now the 3D model will be created. The image size has to be adjusted. A wrong size will result in an error.

```
1 model = sfm_bundler(OBJECT_NAME, img_dir, out_dir, KK, zeros(5,1),
      [1600 1071]);
```

'Sfm_view' shows the model with all features found. Sfm_alignment_gui allows, in addition, to aligned the model in the coordinate system.

```
1 sfm_alignment_gui(model);
2 // sfm_view(model, 'all');
3 save  (['model_' OBJECT_NAME '.mat'], 'model')
4 sfm_export_xml(strcat('//home/projekt-amser/Desktop/moped/moped/
     models/', OBJECT_NAME, '.moped.xml'), model);
```

After the model has been created, there may be some points that are not part of the model itself. Often, they are scattered around the object. With the

following code the points can be filtered in MATLAB. Use the Data Cursor in the model preview to determine the threshold for the points which should be deleted.

```
1  // If the threshold should only be checked for one coordinate, a certain
       row can be selected
2  // k is the row that should be selected, e.g k=1 searches all x values
       for values bigger 10
3  // [row, column] = find(model.pts3D(k,:) > 10 | model.pts3D(k,:) < -10)
4  // If the above line was used, column will be a two dimensional matrix.
       The size for i has
5  // to be determined as following:
6  // [_foo, size_for_i] = size(column);
7
8  [row, column] = find(model.pts3D > 10 | model.pts3D < -10)
9  for i=size(column):-1:1;
10 x=column(i);
11 model.pts3D(:,x)=[];
12 end
```

Attached to this document there is a MATLAB script that automates the modeling process. However, the resizing of the pictures and the filtering of scattered points has still to be done manually. Also, camera calibration parameters for different resolutions have to be saved in a MATLAB file and the script must be modified in order to take that new file into account.

2.2. Problems that can occur while trying to create a 3D model

If the created model doesn't match the expectations, this tips should be checked out:

- The camera may be not calibrated correctly. The calibration parameters depend on the resolution of the images. If the model was created from pictures with a resolution of 1600x1200, the calibration parameters have to be obtained from pictures with the same resolution. Also, the focal length should not vary when recording the images. For the calibration parameters there is a rule of thumb: if the resolution is divided by a factor of two, the parameters have to be divided by the same factor. This rule isn't one hundred percent accurate, but for MOPED it is sufficient.
- The camera settings may not be good enough. The background must not be blured, the object should be in focus and overall, there should be enough features. To check the latter one, process the images with a 'Difference of Gaussians' algorithm (for example in 'gimp'). The detected edges are candidates for features. No edges means, that there can not be many features that MOPED needs to create a good model. Therefor, objects with now texture are not appropriate for MOPED.

3. Positioning the model to find position and rotation

MOPED is not only able to recognize an object, but can also determine its position and rotation. Before MOPED can give some meaningful values, the model has to be aligned in the coordinate system. This can be done using the *sfm_alignment_gui* script that comes with MOPED.

The alginment GUI has several options to adjust the object in the three dimensional coordinate system. With the bars the Rotation in X,Y and Z direction can be set as well as the scale of the model. With the X,Y and Z position boxes, the model can be set to a certain position. The three axes are displayed as a red (X-Axis), green (Y-Axis) and blue (Z-Axis) line. One possibility to position the model in the coordinate system is shown in the picture 3.1.

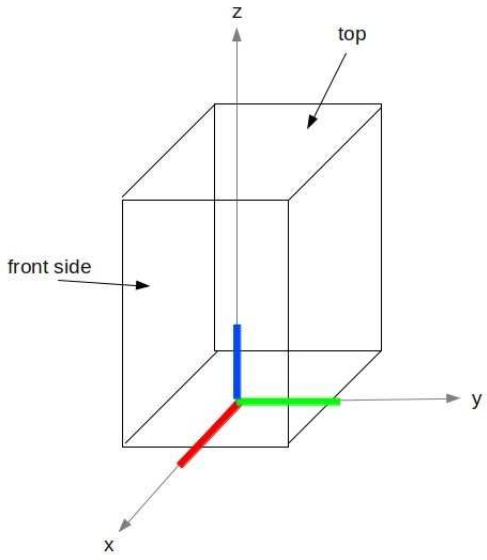

Abbildung 3.1.: Possible position of a model in the coordinate system

In order to position the model in the coordinate system, the first step is to rotate the model. It is important to save and load the model after each change of one parameter, since the sfm_alignment_gui applies all changes within a single matrix operation. This means, if the rotation in Z direction is changed directly after the rotation in X direction, without saving, the model will not rotate properly around the Z-Axis.

After the model is rotated, use the position boxes to align the model in relation to the origin. The last step is to change the scale to make the model match some metric. Measure the height and width of the real object and make the values of the coordinate system match them. For example - assuming the object is 20 cm high and 5 cm wide, the model is positioned as shown in the picture and the values on the Z-Axis range from 0 to 2 - using a scaling factor of 0,1 MOPED should give the position in meters.

4. Experimental series - how the resolution and number of images affects the model

This chapter will give a short overview of how the resolution and the number of the images used for creating a 3D model influences its quality. As said before, there was sometimes a side missing in the model, which could be solved by taking more pictures for the process. Since more images lead to a higher computing time it is quite interesting what the critical value is for a good model. The same applies for the resolution of the images. The higher the resolution, the more key points will be found in the image. On the one hand this will lead to a good model with a lot of features, on the other hand it will take much longer to match all the key points of the different images to a single 3D model.

The aim is to find a good relation between quality and computing time. In order to find the boundary values, an experimental series was created. The same set of images of a Pringles box (see attachment A.2) was used to create the same model with a different number of pictures in different resolutions. To compare the models, the MOPED recognition was used. It returns a score that indicates how well the recognition of the model was in a certain picture. Each model was tested about 100 times with the same picture (attachment A.3). The values in table 4.1 show the mean of the recorded scores.

The results indicate that a resolution of at least about 800x535 pixels is necessary to create a model that can be recognized by MOPED. It also shows, that the further increase of resolution does affect the quality of the model, but not in a large scale. However, the same model created with less pictures gets worse if there aren't enough features, thus if the resolution is too small. There is also a problem that a part of the model is missing if the amount of images

Tabelle 4.1.: Experimental series

	Resolution			
Number of pictures	**1600x1071**	**1200x803**	**800x535**	**600x402**
104	11.2969	11.3909	10.1175	0
70	10.0171	6.8118	1.1355	0
52	9.2181	5.03204	0	0

used is to small. For the Pringles box model this was the case when taking 52 pictures. The score is still quite good because the image that was used doens't show the missing side of the model. The value could be significant lower for other view points. Note that for some objects - with lesser features than the used Pringles box - a higher resultion might be necessery to get a model of good quality.

A. Attachements

A.1. Attachment 1

Rotatable platform, where the object is standing on. A camera positioned in the inner of the construct can take pictures without being moved.

Abbildung A.1.: Easier recording of an object with a rotatable platform.

A.2. Attachment 2

Code for the file modeling.m. Helps to create the 3D model out of the pictures.

```
% Usage: modeling(ObjectName, draw_mask_mode, imundistort)
%
% Input:          ObjectName         - Name of the object that should be modeled. The folder cont
%                 draw_mask_mode          - If the argument isn't 'template' or 'templateShow', m
%                                    If the masks are already there, you have to do nothing.
%                                    If draw_mask_mode is set to 'template', you have to create a
%                                    If draw_mask_mode is set to 'templateShow', MATLAB will show
%                                    You can control your mask to make sure, that the object isn'
%                 imundistort         - If false, imundistort_folder will not be executed.
%

% Declaration
function [] = modeling(objectName, draw_mask_mode, imundistort)

% Path to the folder that contains your moped installation
base_path = '/home/projekt-amser/Desktop/moped/moped/';

OBJECT_NAME = objectName;
cd /home/projekt-amser/Desktop/moped/moped/modeling;
addpath(pwd);
img_dir = strcat(base_path, 'modeling/examples/', OBJECT_NAME);
out_dir = strcat(base_path, 'modeling/examples/', OBJECT_NAME, '_out');


imgfiles = dir(fullfile(img_dir, '*.jpg'));
imgfiles = {imgfiles.name};

%load matching camera calibration
info=imfinfo(fullfile(img_dir, imgfiles{1}));
if (info.Width == 1600)
        display(['Using nikon_nozoom_calib_1600_horizontal.mat']);
        load nikon_nozoom_calib_1600_horizontal.mat;
elseif(info.Height == 1600)
        display(['Using nikon_nozoom_calib_1600_vertical.mat']);
        load nikon_nozoom_calib_1600_vertical.mat;
elseif(info.Width == 1200)
        display(['Using nikon_nozoom_calib_1200_horizontal.mat']);
        load nikon_nozoom_calib_1200_horizontal.mat;
elseif(info.Width == 800)
        display(['Using nikon_nozoom_calib_800_horizontal.mat']);
        load nikon_nozoom_calib_800_horizontal.mat;
elseif(info.Width == 400)
        display(['Using nikon_nozoom_calib_400_horizontal.mat']);
        load nikon_nozoom_calib_800_horizontal.mat;
else
        display(['No matching camera calibration file found']);
```

```
47          return;
48  end
49
50  kc = [0 0 0 0];
51  if (imundistort ~= false)
52          imundistort_folder(img_dir, KK, kc);
53  end
54
55  % if draw_mask_mode is given, draw 1 mask and save it as general.mat. draw_mask function will t
56  if (strcmp(draw_mask_mode, 'template') | strcmp(draw_mask_mode, 'templateShow'))
57          display(['Using template.mask']);
58
59              img = imread(fullfile(img_dir, imgfiles{1}));
60             % Create a mask for the region of interest.
61             mask = roipoly(img);
62              save(fullfile(img_dir, ['template.mat']), 'mask');
63          draw_mask(img_dir);
64
65  else
66          draw_mask(img_dir);
67  end
68
69  % Initializes idx_imgs with the number of files that should be shown with the mask overlayed.
70  nFiles = length(imgfiles);
71  stepsize = 5;
72  idx_imgs = 1:stepsize:nFiles;
73  % Shows the images with mask overlay
74  if (strcmp(draw_mask_mode, 'templateShow'))
75          mask=~mask;
76          for i = idx_imgs,
77                  img=imread(fullfile(img_dir, imgfiles{i}));
78
79                  img_withmask=imoverlay(img, mask, [0 0 0]);
80                  fig = figure, imshow(img_withmask);
81                  waitfor(fig);
82          end
83  end
84
85
86  model = sfm_bundler(OBJECT_NAME, img_dir, out_dir, KK, zeros(5,1), [info.Width info.Height]);
87  save (['model_' OBJECT_NAME '.mat'], 'model');
88  sfm_export_xml(strcat('/', base_path, 'models/', OBJECT_NAME, '.moped.xml'), model);
89  sfm_alignment_gui(model);
```

A.3. Attachment 3

Code for the file clean.m. Helps to clean scattered points around the model, which are not part of the object.

```
% Usage: modeling(ObjectName, draw_mask_mode, imundistort)
%
% Input:         model_file        - name of the file where the model is saved (*.mat file)
%                coordinate         - [x, y or z ] determines on which axis the threshold shoul
%                                  is specified, the script will look for x values that are bigge
%                                  It will then delete all the points.
%                threshold         - specifiy threshold with smaller (>) or greater (<) sign. Fo
%                                  If the operator is not given, the script will look for the exa
%

% Declaration
function [] = clean(model_file, coordinate, threshold)

% preperation
coordinate_value=0;
if (strcmp(coordinate, 'x'))
        coordinate_value=1;
elseif(strcmp(coordinate, 'y'))
        coordinate_value=2;
elseif(strcmp(coordinate, 'z'))
        coordinate_value=3;
else
        display(['coordinate should be x, y or z']);
        return;
end

load (model_file);
threshold_size=(size(threshold,2));
threshold_value=threshold(2:threshold_size);
value = str2num(threshold_value);

if(strcmp(threshold(1), '>'))
        [row, column] = find (model.pts3D(coordinate_value,:) >  value);
elseif(strcmp(threshold(1), '<'))
        [row, column] = find (model.pts3D(coordinate_value,:) < value);
else
        [row,column] = find (model.pts3D(coordinate_value,:) == value);
end
for i=size(column,2):-1:1;
        x=column(i);
display(column(i));
        model.pts3D(:,x)=[];
end
save (strcat('filtered_', model_file));
```

A.4. Attachment 4

Abbildung A.2.: Front view of the model used for the experimental series in Chapter 4.

A.5. Attachment 5

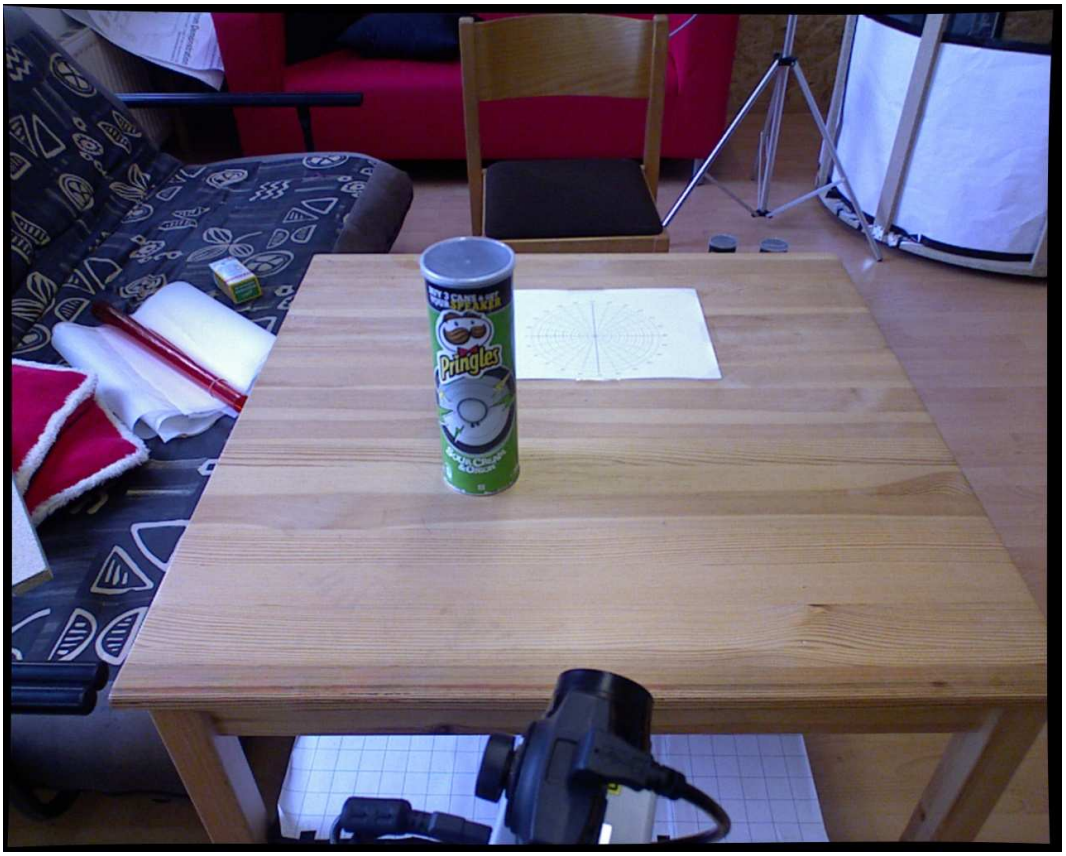

Abbildung A.3.: Test picture used to detect the object.