A Study of 3D Reconstruction of Varying Objects with Deformable Parts Models

Bachelorarbeit

der Philosophisch-naturwissenschaftlichen Fakultät
der Universität Bern

vorgelegt von

Raoul Norman Grossenbacher

29.07.2019

Leiter der Arbeit:
Prof. Dr. Paolo Favaro
Institut für Informatik
Abstract

This work covers a new approach to 3D reconstruction. In traditional 3D reconstruction one uses multiple images of the same object to calculate a 3D model by taking information gained from the differences between the images, like camera position, illumination of the images, rotation of the object and so on, to compute a point cloud representing the object. The characteristic trait shared by all these approaches is that one can almost change everything about the image, but it is not possible to change the object itself, because one needs to find correspondences between the images. To be able to use different instances of the same object, we used a 3D DPM model that can find different parts of an object in an image, thereby detecting the correspondences between the different pictures, which we then can use to calculate the 3D model. To take this theory to practise, we gave a 3D DPM model, which was trained to detect cars, pictures of different car brands, where no pair of images showed the same vehicle and used the detected correspondences and the Factorization Method to compute the 3D point cloud. This technique leads to a completely new approach in 3D reconstruction, because changing the object itself was never done before.
Acknowledgements

First I would like to thank my supervisor, Professor Paolo Favaro. It was his seminar on computer vision that started my interest in this field. The feedback, new ideas and suggestions for improvement I received from him helped me a lot finishing this project. In addition I would like to thank Simon Jenni who let me use his 3D DPM code and helped me a lot with understanding the code. His knowledge of the code helped me in overcoming many obstacles.

I would also like to thank my friends and family for all the support and love they provided me with during this long time. If I needed some motivation I could always go talk to them and they would help anytime. (And if I needed a distraction they were there too.)

A very special thanks goes to my girlfriend Amanda. Somehow she performed the miracle of coming up with enough patience and sympathy to help me until the end of the thesis. Without her, this thesis would have never been finished.
# Contents

1 Introduction ........................................... 1  
   1.1 Motivation ............................................ 1  
   1.2 Outline .............................................. 2  

2 Background .......................................... 3  
   2.1 HOG ................................................ 3  
   2.2 DPM ................................................. 4  
   2.3 3D DPM ................................................ 5  
   2.4 Factorization Method ............................... 6  

3 Idea and Implementation ............................ 7  
   3.1 DPM Recap ......................................... 7  
      3.1.1 Sliding Window ................................ 7  
      3.1.2 Mixture Models and Deformable Parts .......... 8  
      3.1.3 Inference ...................................... 9  
      3.1.4 Training ...................................... 10  
      3.1.5 Initialisation .................................. 10  
   3.2 3D DPM ............................................. 11  
      3.2.1 Components and Viewpoints .................. 12  
      3.2.2 How to Place parts in 3D .................... 12  
      3.2.3 How to Initialise Parts in 3D ................. 14  
   3.3 Factorization Method ............................. 15  
      3.3.1 Constructing the Data Matrix ................. 15  
      3.3.2 Factorizing $D$ ................................ 16  
      3.3.3 Motion and Shape Matrices .................... 16  
   3.4 Reconstructing 3D ................................. 16  
      3.4.1 Algorithm ..................................... 16  

4 Experiments and Results ........................... 18  
   4.1 Trained Model ....................................... 18  
   4.2 Test Pictures ....................................... 18  
   4.3 Results ............................................. 19  

5 Conclusion ............................................ 25  
   5.1 Future Work ......................................... 25  

List of Figures .......................................... 26  

Bibliography ............................................. 26
Chapter 1

Introduction

1.1 Motivation

One of the main goals in Computer Vision is generating 3D models from 2D pictures. Normally one needs multiple pictures of the same object to create a 3D model, either from different angles or with varying illumination. In both methods the pictures are from the same object. But what if one would like to generate a 3D model with pictures of different objects, like images of varying cars? This thesis offers an approach to do exactly that. Normally 3D reconstruction software searches for corresponding points in the different images. Knowing these correspondences the software can calculate the rotation and translation between the different cameras and reconstruct a model of the 3D points. To find the correspondences the software needs the same object in the different images. Otherwise the software can not match two points in two images, since the objects can vary in shape and colour. So how can we bypass this problem?

The method used in this approach is to divide the problem in two different steps and solve them each by themselves. The first step is to get correspondences. The second step is to calculate a 3D model with the matrix factorization method \[18\].

If the program finds the same feature in two images, that is a correspondence. The detection of the different parts is done with a 3D Deformable Parts Model (3D DPM) \[3\], which is based on the Deformable Parts Model (DPM) \[7\]. The classic DPM detects objects in images. DPM is one of the most successful and widely used approaches to object detection. It uses HOG (Histograms of Oriented Gradients) \[3\] features, which are scale-invariant, mixture models and a deformable geometric model. This combination allows DPM a great variation of different object classes.

To match the detected features in the different images together, the program needs to know which features are the same. Normal DPM can not do that, since one needs to train one DPM per angle and it is not possible in the classic DPM to tell the software to learn the same features. Therefore we are using the 3D DPM. There the code learned more features in total than it learned per angle, so if it detects something in the image, it can tell which feature it is.

After the program detects the same features in the different images, we can use the generated data to calculate a 3D Model of the object. Since the 3D DPM
1.2. OUTLINE

only calculated bounding boxes around the detected object and its features, we only get an area where the object and its parts are.

Normally software which created 3D models from 2D images finds between hundred and thousand corresponding points. With that many points the created 3D models have many details. In this case we do not have matching points, we have matching features. Each feature will be treated like a point. Therefore the 3D model will only have a low number of points, since we only have a few features per model.

This constraint is acceptable because we use different instances of the same model class in the different images and not the same object in all the images. Therefore it is not possible to find up to thousand matching points in the pictures, since we have to look for matching features, which need multiple points themselves.

1.2 Outline

This thesis has 3 main chapters. In chapter 2 you can read about prior work like HOG features, DPM, 3D DPM and the matrix factorization method. This chapter will also give the reader a view of the research done in this field. It will additionally show how the different parts need to be combined to achieve the goal of this thesis.

Chapter 3 describes in detail the implementation of the 3D DPM and which parts are important for this thesis. There you can see how we get to the corresponding points and how we need them to calculate the 3D model. In section 3.2 it is described how the 3D DPM code works and how it can be used, in section 3.3 you will read about the matrix factorization method, and lastly in section 3.4 the combination of the two will be shown.

In chapter 4 the experimental results will be shown. How well can the matching features be found? How precise are the 3D model clouds? These questions will be answered and the results will be compared to similar work done in this field.

Lastly in chapter 5 we will look again over the experimental results, talk about the gained insight and knowledge and think about possible future work.
Chapter 2

Background

This thesis combines two different parts of Computer Vision. One is object/feature detection in images. The other is generating a 3D model with 2D points. In this chapter are informations about the background of these parts and some general information on this field.

2.1 HOG

Can grids of Histograms of Oriented Gradient (HOG) descriptors outperform existing feature sets? [3] studied this Question. They tested their code on the original MIT pedestrian database and achieved a near-perfect separation. Afterwards they tested their code on a more challenging dataset which contained over 1800 images with different poses and backgrounds.

Humans can adopt many different poses. To be able to detect them, one needs a robust feature set. This feature set must withstand cluttered backgrounds under difficult illuminations. Their test case was “pedestrian detection”. A linear SVM [1] was used as their base line classifier. Since their result on the MIT pedestrian test set was near-perfect, they decided to create a more challenging test set. This new set contained over 1800 images and a vast variety of poses and backgrounds.

Their method works with an extraction chain: Input image → Normalize gamma colour → Compute gradients → Weighted vote into spatial orientation cells → Contrast normalize over overlapping spatial blocks → collect HOG’s over detection window → linear SVM → person/ non-person classification. The fist element of the chain is the input, the last the output. Each image is divided in small cells and for each cell a local 1-D histogram of gradient directions or edge orientations is calculated. The cells should be invariant to illumination. Therefore, they are contrast-normalized before usage.

The well-established MIT pedestrian database [11] contains over 700 images of pedestrians. 509 are for training purposes, the remaining 200 for testing. The range of poses is relatively limited and the set contains only front or back views. Since their best detectors gave perfect results, they wanted to have a more challenging data set. “INDRIA” [3] contained 1805 different images, with a lot of diverse backgrounds.

They selected 1239 different images and their left-right reflections. The ini-
tial negative set was obtained from 1218 person-free training photos, from which they took a fixed set of 12180 patches. These were searched for false positives. Afterwards they used the augmented set (initial 12180 + hard examples) to retrain the method. The detector performance was quantified by plotting Detection Error Trade-off (DET) curves on a log-log scale.

Compared to other detectors like Generalized Haar Wavelets [9], Shape Context [2] or PCA-SIFT [19], the HOG-based detectors greatly outperformed the others. They delivered near perfect separation on the MIT set and on the INRIA they managed to reduce at least an order of magnitude.

Gamma/Colour Normalization: They used several input pixel representations like greyscale, RGB and LAB colour spaces. But these normalizations made only a small impact on the performance. Perhaps this is because of the later performed normalization. Gradient Computation: It turns out that simple schemes are the best for computing gradients. The detector performance depends on these gradients. They tested many different masks. They found out that simple 1-D masks at $\sigma=0$ work best. When they had images with colour, they calculated separate gradients for the different colour channels and took the one with the largest norm as the pixel’s gradient vector.

Spatial/Orientation Binning: The descriptor has a fundamental non-linearity. A spatial region is called a cell. For each cell an orientation bin is calculated. Therefore, for each pixel a weighted vote is calculated. The cells can be radial or rectangular. The vote is a function of the gradient magnitude at the pixel. They found out that increasing the number of orientation bins improves performance significantly.

Normalization and Descriptor Blocks: For good performance, it turns out that an effective local contrast normalization is essential. That is because the gradient strength varies over a wide range of colors and backgrounds. Most of their schemes are based on grouping cells into larger spatial blocks. Later these blocks were normalized separately. The final descriptor is then the vector of all components of the normalized cell responses from all of the blocks in the detection window. They used two different classes of block geometries. The R-HOG was rectangular and C-HOG war circular. For each of these HOG geometries they evaluated four different block normalization schemes: L2-norm, L2-Hys, L1-norm and L1-sqrt.

2.2 DPM

One of the fundamental challenges in computer vision is object recognition. [7] tried to detect and localize generic object with different categories, like cars or people. The main issue of this problem is the variability in shape of the objects. They can adopt many different poses or appear in different viewpoint or with changing illumination. [7] managed to create both an efficient and accurate system, which achieves state-of-the-art results on PASCAL VOC benchmarks [6], [4] and [5] and the INRIA person data set [3]. (Which was used in the work of [3]) They used mixtures of multiscale deformable part models. In this model, a number of object parts are detected. If they are connected to other object parts, within a defined deformable configuration, the object is detected. Before this paper, deformable part models were often outperformed by more simpler models. Felzenswalb et al. tried to overcome this performance gap. They
used mixture deformable part models because a single deformable part model is
often not expressive enough to represent an object. An image of a bike can be
taken from different viewpoints and the bike can have many different shapes.
Felsenswalb et al. wanted to use “visual grammars”. In visual grammars object
parts are defined directly or with other parts. Simpler models outperform more
complex models due the training. Simple models can easily be trained using
SVM. It is difficult to train complex models. To train their models, they used
a latent variable formulation of MI-SVM.

They used a variation of HOG features from [3] and defined a score at dif-
f erent positions and scales in an image. This led to the conclusion that higher
resolution features are essential for a good performance. This code computes an
overall score for each root location, which depends on the best possible place-
ment of the parts.

A latent SVM is a non-convex optimization problem, but once latent in-
formation is specified for the training examples, the problem becomes convex.
Bootstrapping methods use a set of negative examples and add the negative ex-
amples which were classified as the set of hard negatives. They may repeat this
procedure multiple times. In their method they start with an initial “cache”
of examples an their code alternates between training a model and updating
the cache. The easy examples are removed step-by-step in each iteration and
replaced with hard examples.

To train they used images with bounding boxes around the object of interest.
The PASCAL data sets [6], [4] and [5] consist of such images. They initialized
a structure of a mixture model and then they let the program learn all the
parameters. The training is done via latent SVM. The training model consist of
positive examples P, negative images N and an initial model Beta. The result
is a new model Beta.

They used HOG features. A pixel-level feature map that specifies a sparse
histogram of gradient magnitudes was defined. To get some invariance to small
deformations and to reduce the size of a feature map, they created a cell based
feature map by aggregating pixel-level features. The simplest way to do this is
to map each pixel in a cell and calculate the feature vector for the cell. This
can be accomplished by summing or averaging the pixel-level features.

It is not entirely clear what the desired output of an object detection should
be. In the PASCAL challenges ([6], [4] and [5]) the code should predict the
bounding boxes. In this system, they used the complete configuration they
received from the object to predict a bounding box for it. They used a scoring
system to eliminate double detections, since the same object could be detected
multiple times due the overlapping.

They evaluated their system with the PASCAL VOC 2006, 2007, 2007 comp3
challenge data sets and protocol ([6], [4] and [5]). To pass this test the code
should be able to provide bounding boxes of a given class in an image. The
calculated bounding box is considered correct if it overlaps with at least 50
percent of the ground truth bounding box.

2.3 3D DPM

For this thesis the 3D DPM code of [8] was used. To turn the DPM code
into 3D DPM code, one needs special 3D training sets. In this case it was
viewpoint annotated. The training data used here consisted of synthetic and real examples of cars. To create the synthetic pictures, two different approaches were used. Either a CAD model was rendered onto a negative training image or onto a single-coloured image. This image colour was the average pixel value of a training image.

The first change to the classical DPM was to assign mixture components to particular viewpoints. Therefore component training sets were initialized and the training data was spread on these sets accordingly. The components which can be seen depend on the viewpoint. For example, if we see the right side of a car on an image, it is not possible to detect a component from the left side.

2.4 Factorization Method

Once the 3D DPM code has gathered the corresponding points in the different images, one needs to calculate the 3D points, to get a point cloud of the object. One possible way to calculate this cloud is the Factorization Method [18]. In chapter [5] you will read more about the mathematical part of this method. Here is a general idea of it.

One takes all the corresponding points and creates a data Matrix $D$. With this data matrix $D$ two other matrices are calculated. The first is a so called motion Matrix $M$, the second a structure Matrix $S$. The matrices are calculated, so that $D = MS$. In the structure Matrix $S$ are the calculated 3D points. In the motion Matrix $M$ are the different camera matrices, which turn the 3D points into 2D points per image. Each column in $S$ is a 3D point. Therefore $S$ is the 3D point cloud we are looking for. Figure 2.1 illustrates this.
Chapter 3

Idea and Implementation

To get the 3D DPM to work with the Factorization Method, one needs to understand how they work. This chapter gives a more detailed look on the formulas and how we can use the 3D DPM to find 2D points, which the Factorization Method can use to calculate the 3D points.

3.1 DPM Recap

The DPM belongs to the family of sliding window detectors. A feature pyramid is computed from the input image, then a set of linear filters is applied. There are two kinds of filters. Multiple higher resolution part filters and a root filter per DPM. The part filters model smaller parts of the object, and the root filter models the whole object.

3.1.1 Sliding Window

To get an idea of the sliding window approach, one can imagine a frame, which is smaller than the image. This frame is moved over all possible positions on the image. For each position, one checks if the searched object is present or not.

Instead of using the images themselves, feature maps are used, which are computed from the images. A feature map $F$ consists of a two-dimensional array of d-dimensional feature vectors. Each feature vector represents a small image patch and the feature map is a grid of feature vectors. Here HOG features are used. Let us look at a model with only a root filter (or in other words: a DPM without parts). A root filter consists of d-dimensional weight vectors, which are stored in a two-dimensional array. One can interpret the root filter as the “window” that moves over the feature map to detect objects.

We use a feature pyramid to detect objects at different scales. A parameter $\lambda$ determines the subsampling factor from level to level which is constant. The number of levels in an octave is fixed with $\lambda$. To compute the feature map an octave beneath in the pyramid, twice the resolution is used. For each placement of the filter in the feature pyramid a score is assigned. To compute this score, the dot-product of the filter and the related sub-array of the feature map at the specific level is calculated. More precisely: If we have a filter $F$ of size $w \times h$, we can define a filter placement in the feature pyramid by a three-tuple
3.1. DPM RECAP

\[ p = (x, y, l). \] (x, y) determines the location of the upper-left corner and l is the level of a feature pyramid H. Additionally let \( \phi(H, p, w, h) \) stand for the vector which can be obtained by concatenating the feature vectors in H, correlating to a window of size \( w \times h \) with position \( p \) in H. The score is then determined by

\[ \text{score}(p) = F' \cdot \phi(H, p), \] (3.1)

where \( \phi(H, p) \) is a short form for \( \phi(H, p, w, h) \), since \( w \) and \( h \) are implied from \( F \) and \( F' \) defines the concatenation of the weight vectors in \( F \).

3.1.2 Mixture Models and Deformable Parts

Now it is time to extend the root-only model with a deformation model and parts. The case we will look at is of a DPM model for cars. The whole object (in this case the whole car) is modelled in the root filter, which covers approximately all of the model. In this case, the root filter can be seen as a detection window. To cover more detailed and smaller parts of the car, for example windows of tires, one uses part filters with a higher resolution. To compute the part filter with twice the resolution, they are placed an octave (\( \lambda \) levels) below the root filter in the feature pyramid. Each part is located relative to the root filter’s position at a fixed anchor-position. This way we get a star-shaped model. There can be variations in the positions of a part with different cars. To model this, parts can differ from their anchor position. This difference is restricted by a deformation cost.

We define a model with \( n \) parts as \((F_0, P_1, ..., P_n, b)\) with \( F_0 \) for the root filter, \( P_i \) for a model of part \( i \) and \( b \) for a bias term. Each \( P_i \) is defined by \((F_i, v_i, d_i)\) with \( F_i \) as the part filter, \( v_i \) describes the anchor position, and \( d_i \) models the deformation cost with a four-tuple of coefficients. We call a placement \( z = (p_0, p_1, ..., p_n) \) a hypothesis. To compute the relative displacement to their anchor position of the parts, we use the formula

\[ (dx_i, dy_i) = (x_i, y_i) - 2((X_0, y_0) + v_i). \] The factor of two is necessary, since the parts are computed at double the resolution of the root. If we have the displacements, we can set the deformation feature to \( \phi_d(dx_i, dy_i) = (dx^2, dx, dy^2, dy) \). To give a hypothesis a score we use the formula

\[ \text{score}(z) = \sum_{i=0}^{n} F_i' \cdot \phi(H, p_i) - \sum_{i=1}^{n} d_i \cdot \phi_d(dx_i, dy_i) + b. \] (3.2)

The second term of the sum is concerned with the deformation and the first term with the appearance. If we combine multiple DPMs, as we just described, we get a richer object model, which we call a mixture model. Each mixture model is an \( m \)-tuple \( M = (M_1, ..., M_m) \) with \( M_c \) standing for the model for mixture-component \( c \). To compute an object hypothesis for a specific mixture model we use \( z = (c, p_0, ..., p_n) \) where \( n_c \) defines the number of parts in \( M_c \). To get the score of the hypothesis \( h \) we simply use the score of \( z' = (p_0, p_1, ..., p_{n_c}) \) for \( M_c \).

If we join all the model parameters of \( M_c \) in a vector

\[ \beta_c = (F_0, F_1, ..., F_n, d_1, ..., d_n, b) \] (3.3)
we can express the whole mixture model with the vector \( \beta = (\beta_1, ..., \beta_m) \). Similarly we can join all the features of hypothesis \( z' \) into the vector

\[
\Phi(H, z') = (\phi(H, p_0), ..., \phi(H, p_n), -\phi_d(x_1, y_1), ..., -\phi_d(x_n, y_n), 1).
\]

This way, we can construct a sparse feature vector for hypothesis \( h \) by setting

\[
\Phi(H, z) = (0, ..., 0, \Phi(H, z'), 0, ..., 0).
\]

We can use the dot product to declare the score of the hypothesis \( h \) as

\[
\text{score}(z) = \beta \cdot \Phi(H, z).
\]

3.1.3 Inference

If we find high scoring root placements, the problem of object detection is solved, since the root filters can be interpreted as detection windows. To compute the score of a root placement for each mixture component we can use:

\[
\text{score}(p_0) = \max_{p_1, ..., p_n} \text{score}(p_0, p_1, ... p_n).
\]

The goal is to find the best placements for each part for all root placements. With the formula \( R_{i, l}(x, y) = F'_i \cdot \phi(H, (x, y, l)) \) we can pre-compute the array \( R_{i, l}(x, y) \) which stands for the filter responses. In this case \( l \) is the level in the feature pyramid and \( i \) stands for the filter. We can transform the filter responses with

\[
D_{i, l}(x, y) = \max_{dx, dy}(R_{i, l}(x + dx, y + dy) - d_i \cdot \phi(dx, dy)).
\]

\( D_{i, l}(x, y) \) describes the maximum contribution of the part \( i \) compared to a root filter. This filter is located in the feature pyramid in the way that we can see the anchor position of part \( i \) at \((x, y, l)\). Now we can express the score in equation 3.7 as

\[
\text{score}(x_0, y_0, l_0) = R_{0, l_0}(x_0, y_0) + \sum_{i=1}^{n} D_{i, l_0}(2(x_0, y_0) + v_i) + b.
\]

Bounding Box Prediction

We can use a so-called bounding box regression technique to raise the accuracy of the detected bounding boxes. Basically we use the new information which is provided by the part placements to clarify the predicted bounding boxes. There are two advantages: First of all, if a model has parts, we can model intra-class changes better (like is it a small car or a big car) and secondly since we have the double resolution of the root filter, we can match the parts more precisely.

Once a model is trained, we train a new bounding box predictor. Let’s use \( g(z) \) for a feature vector which we can obtain from the object hypothesis \( z \). \( z \) contains the upper left corners of all filters in image coordinates and the width of the root filter in image coordinates. The bounding box predictor is defined with the linear function \( b(g(z)) = (x_1, y_1, x_2, y_2) \) of \( g(z) \). We get the lower-right \((x_2, y_2)\) and the upper-left \((x_1, y_1)\) corners as outputs for the predicted bounding box. We use least-squares regression to learn the function \( b \).
3.1. DPM RECAP

Non-Maximum Suppression

If we use the bounding box prediction procedure like it is right now, we would get multiple overlapping detections. This is due to the model detecting the same object on positions that have a similar scale and are really close to each other. Now we can not simply choose the detection with the highest score since that would discard several correct detections. Therefore we need a new method to reduce detections.

We will use non-maximum suppression. Here the detections are first sorted depending on their score. Then the detection with the highest score is chosen. Now we greedily chose more detections, while we skip the detections that have an overlap bigger than 0.5 with one of our previously chosen detections.

3.1.4 Training

Now let’s talk about training. We have a set $D = \{(x_1, y_1), \ldots, (x_n, y_n)\}$ of training examples. In this case the $x_i$’s are images and $y_i = (y_l, y_b)$ stands for the labels. Here $y_b = (x_{min}, y_{min}, x_{max}, y_{max})$ defines a bounding box around the object and $y_l \in \{-1, 1\}$ declares if the object is there or not.

For the example $x$ and the labels $y$ we can define $Z(x, y)$ the set of possible latent hypothesis values. If we have a hypothesis which leads to a root placement that is overlapping with the bounding box label at least 0.7, we speak of a positive example. For the negative examples we do not set any restrictions on $Z(x)$. If we have a model $\beta$, we can use equation 3.6 to train a linear classifier with the formula:

$$f_\beta(x) = \max_{z \in Z(x,y)} \beta \cdot \Phi(x,z).$$

(3.10)

Here a formalism called Latent-SVM [7] is used. It is similar to MI-SVM [1]. Latent-SVM is used to minimize the objective function

$$L_D(\beta) = \frac{1}{2} \|\beta\|^2 + C \sum_{i=1}^{n} \max(0, 1 - y_i f_\beta(x_i))$$

(3.11)

similarly to standard linear SVMs with standard hinge loss. If this optimisation problem is given fixed latent variables for positive examples, it will become convex. Therefore it is semi-convex.

One can solve equation 3.11 by repeating the steps below:

1. Create a training set $D(Z')$ with $Z'$ for the fixed latent variables for both negative and positive examples and $\beta$ is kept fixed. If we find $\arg \max_{z \in Z(x,y)} \beta \cdot \Phi(x,z)$ we can fix latent variables of positive examples. We use hard negative mining for negative examples. Explicitly, we look for examples with $\beta \cdot \Phi(x,z) > -1.05$.

2. Then we update $\beta$ by solving equation 3.11 with the constrained set. Therefore we use a gradient descent algorithm which uses sub-gradients similar to [17].

3.1.5 Initialisation

We want to train root filters for each component. Therefore we first split the positive training examples with respect to aspect ratio statistics in $n$ training-sets. This way we can initialise a $n$-component model. We choose the root
3.2 3D DPM

To be able to train a 3D DPM we absolutely need training data which is 3D annotated. For the training of the model that was introduced in [13] and [12] and used for this work, both synthetic and real training examples of cars were used. To receive the synthetic examples, CAD models were rendered onto backgrounds of mean pixel colour of the training images or negative training images. (See figure 3.1). There were experiments with both perspective and orthographic projections for the rendering.

In the original DPM [7] the symmetry constraints did not allow to distinguish different views. Therefore they were eliminated (if you have a mirrored right and left side view of a car, they look identical). In figure 3.2 you can see the different evolution steps of one model view while the model is trained.

Figure 3.1: Perspective projection was used to render both of these images. On the left is a CAD example which was rendered on a background of mean pixels. On the right is a CAD example which was rendered with its annotated bounding box on a negative training image. These images are from [8].

filter dimensions in a way that they do not cover more than 80% of the positive bounding boxes. To train root filters we warp positive bounding boxes to filter dimensions. We train against negative examples that can be obtained by extracting features from regions of negative training images, which we choose randomly. At the time that all the root filters are initialised, we train the model by using multiple iterations of hard negative mining. Interestingly, at this stage only the root placement and the component variable is considered latent.

Regions with high Euclidean norm of positive root filter weights can be seen as high “energy” regions. To initialise the part filters, they are placed on those. To find possible start shapes, we take a pool of shape-templates and add them to the model greedily. We choose new parts to add until we added a fixed number of \( k \) parts. If we covered a region, it is zeroed out. We set the constrain that the root filters have to be symmetric when they are lying on the centre vertical axis, as well as parts that are on the vertical axis. The parts that are not in the centre must have a symmetric partner.
3.2. 3D DPM

(a) Initialised root filter after training against random negatives
(b) Root filter after training against hard negatives
(c) Mixture components after parts have been initialised
(d) Mixture component after training against hard negatives

Figure 3.2: Visualisation of the evolution of a mixture component of a 3D DPM with eight viewpoints during the training procedure. These images are from [8].

3.2.1 Components and Viewpoints

The splitting of the training data is done with 3 variables: \( n_c, n_a, n_e \). \( n_c \) is the number of mixture components, \( n_a \) is the number of different azimuth angles and \( n_e \) is the number of different elevations. These variables have to fulfil the following equation: \( n_c = n_a \cdot n_e \). Viewpoints are distributed equally, meaning a model with \((n_c,n_a,n_e) = (8,4,2)\) would have four azimuth angles lying in \( \{0°,90°,180°,270°\} \).

One way to ensure that the components are labelled correctly during training is the usage of a penalty for incorrect viewpoint predictions. There are 3 labels that are used in this method: \( y_l, y_b, y_c \). Label \( y_l \in \{-1,1\} \) declares if the object is in the picture or not. Label \( y_b = (x_{\text{min}}, y_{\text{min}}, x_{\text{min}}, y_{\text{min}}) \) declares a bounding box around the object in image coordinates. During latent positive search, the following formula is used for the latent variables

\[
z' = \operatorname{argmax}_{z \in Z(x,y)} (\beta \cdot \Phi(x,z) - l(y_c,c)),
\]

where \( l(y_c,c) = \mathbb{1}[y_c \neq c] \cdot \gamma \) is the penalty term. \( \gamma \) was set to \( \gamma = \max(1, \beta \cdot \Phi(x,z)/2) \). So if a very confident hypothesis leads to wrong viewpoint estimates, it is penalised more.

3.2.2 How to Place parts in 3D

In 2D the anchor positions were placed in the 2D root filter. In 3D we will place them in the 3D object space. Here we need a reference coordinate system. Therefore we create a 3D object box whose origin is located at the centre of the 3D box (for more details, take a look at section 3.2.3). We define parts boxes \( b_i = (s_x, s_y, s_z) \) which are co-aligned with the object box. These part boxes are
used to parametrise the parts. To specify the centre of the part box in object
coordinates we define the new anchor position as \( v_i = (s_i, y_i, z_i) \).

If we look from a mixture component we can see a 3D part box. To be able
to tightly cover this box, we let the part filters remain 2D templates of weight
vectors and size them accordingly. The viewpoint of a component \( c \) gives us an
orthographic projection \( P_c \) which we can use to map the 3D anchor position to
root filter coordinates.

We have to ensure that part filters, which are learned, are consistent across
views. Therefore we let the latent part positions be inferred in 3D while being
simultaneous for each view of a given car model. To achieve this, we make use of
the CAD training data. Therefore if we have a positive CAD example, we label
it with a model-ID \( y_0 \) and create a set \( D(i) = \{ (x, y) \in D : y_0 = i \} \). We know
that CAD examples provide us with perfect annotations. Therefore we can fix
the root placement and component variable. Now we choose the root placement
\( p_c \) of component \( c \) in the way that we receive a maximum overlap with bounding
box \( y_b \) which is annotated. Now we can define an object hypothesis in 3D by
\( h = (q_1, \ldots, q_n) \) where the position of part \( i \) is specified by \( q_i = (x_i, y_i, z_i) \). Here
you can see that we get for each model component \( c \) the connection to the 2D
hypothesis \( z = (c, p_c, P_c(q_1), \ldots, P_c(q_n)) \). We still fix the parts in the way that
they are an octave below the root filter in the feature pyramid. Now we define
\( \Psi(x, y, h) = \Phi(x, z(y, h)) \), with \( z(y, h) = (y_c, p_c, P_c(q_1), \ldots, P_c(q_n)) \). Now we
want to find
\[
\hat{h'} = \arg \max_h \sum_{(x, y) \in D(i)} \beta \cdot \Psi(x, y, h).
\] (3.13)

Here you can see that we still model part displacements in 2D and they only
depend on other views with equation (3.13). With the 3D deformation model,
performance for both viewpoint estimation and detection are decreased, but
performance on ultra-wide baseline matching experiments is increased.

For a car model \( i \) we use the following procedure to solve equation (3.13):

**Step 1** Pad the 3D object box to all sides to construct a 3D grid of possible
part placements. With padding we allow parts to be outside of the 3D
object box.

**Step 2** For each example \( (x, y) \in D(i) \) feature maps for part and root filters are
computed. With the fixed root placement we want to obtain maximum
bounding box overlap and we compute the feature maps in the way that
if we see a 3D grid from viewpoint \( y_c \) it is covered completely.

**Step 3** Now for all the views sum up the scores for placing part \( p_j \) at position
\( q_i \) by iterating each position \( q_i \) of the 3D grid and each part \( p_j \). If we
have a view which is related to component \( c \), we can compute the score
by placing it at \( P_c(q_i) = (s_i, y_i) \) in root filter coordinates. With the
formula \( \text{score}(p_j, q_i, c) = F'_j \cdot \phi(H, P_c(q_i)) - d_j \cdot \phi_d(dx_j, dy_j) \) this can be
done. Here we have to note that \( H \) denotes not a feature pyramid that
as described in section 3.1.2, but instead the feature maps computed in
step 2. To get the total score for placing \( p_j \) at \( q_i \) we can use the formula:
\[
\text{score}(p_j, q_i) = \sum_c \text{score}(p_j, q_i, c).
\]

**Step 4** Choose the position with the highest overall score for each part. We
define \( h' = (\arg \max_q \text{score}(p_1, q), \ldots, \arg \max_q \text{score}(p_n, q)) \).
3.2. 3D DPM

If we have a positive training example which does not have a provided model-ID, we treat it as before. Meaning we project the 3D anchor position to 2D and then we search for the best hypothesis in 2D.

3.2.3 How to Initialise Parts in 3D

Now we want to initialize the parts in 3D. To do this, we use a similar heuristic as we used in 2D DPM. We greedily chose high energy regions in 3D and place the parts there. Let’s say we have a model which has \( m \) parts. Then we can use the following procedure:

**Step 1** First we need to set up a grid of possible part placements. To get this, we create the 3D object box. We model the box with a 3D array of zeros. To initialize the dimensions of the box, we use the dimension of right and front view root filters. We double the height of the front filter and set this as the new height. Then we warp the right and front view dimensions in a way that they fit the height of the box, at the same time we preserve the aspect ratios.

**Step 2** Then we create a pool of \( b_i = (s_{xi}, s_{yi}, s_{zi}) \) which are possible 3D part boxes. All part boxes have a size such that their volume is approximately \( 1/m \) of the total object box volume. To be exact: \( V_{part} = 0.8 \cdot \frac{V_{object}}{m} \).

**Step 3** Now we convolve all the part boxes with the 3D object box. We do this to have a grid of possible start placements. Then we move the part box through the grid and assign an energy to all the grid positions. Let’s say we have a grid position \( q = (x, y, c) \). To obtain the energy we project \( q \) onto the root filters of each component. Then we sum up the norm of positive root filter weights. The filter weights have to be covered by the part box for all the components.

**Step 4** To get the new 3D part we chose the combination of highest scoring position and shape. We do not want overlapping parts. Therefore once we have a new part, the regions that it covers are set to \(-100\) in the 3D object box. This way, if we calculate the energy in step 3, we get a negative result for overlapping parts.

**Until \( m \) parts are chosen, repeat Sep 3 and 4.**

It is not possible to see each part from all the views. Therefore we only assign a maximum of \( k \) parts to each view. We choose the parts greedily for all the views based on both their depth in the 3D object box if we look from this view (we prefer parts in front) and their energy on the root filter. For each 3D part that gets chosen, we zero out the region it covers in the root filter energy. We stop choosing parts once we have \( k \) parts or it is not possible to chose another part any more because there is no region left in the root energy that can possibly be covered by another part (as a side note there is to mention that now different components may have a different number of parts that are assigned to them). We choose the part filter in a way that they tightly cover the 3D part boxes which are seen from that view (see figure 3.3).
3.3. FACTORIZATION METHOD

Figure 3.3: Visualization of part segmentations with their part bounding boxes. The green 2D box stands for the part filter template and the blue box represents a 3D part box from the given viewpoint.

3.3 Factorization Method

Another big problem in computer vision is reconstructing the camera positions and recovering the scene geometry. There exist solutions that work well if you have perfect images. But they react very sensitive to noise. For this thesis a solution was implemented which was introduced in [18]. This method is called the “factorization method”. It is able to robustly reconstruct motion and shape from different images under orthographic projection. Due to the orthography the camera translations on the optical axis will not be recovered. For this thesis this is not important, since the camera will not move along this axis.

3.3.1 Constructing the Data Matrix

In this method they use a data matrix $D$ with dimensions $2m \times n$. Where $m$ is the number of cameras (or in this case images) and $n$ is the number of corresponding points. This matrix will then be factorized (this is why it is called the factorization method) into two matrices $M$ (Motion) and $S$ (Shape). $D$ is factorized in the way that $D = MS$. Therefore the dimensions of $M$ are $2m \times 3$ and the dimensions of $S$ are $3 \times n$. They found out that the results are way better when the centre of the coordinates is in the centre of the object. Therefore the corresponding points are first centred. So for each point $x_{ij}$ in the data matrix $D$ we calculate the centred point $\tilde{x}_{ij}$. To do so, we simply subtract the mean from each point:

$$\tilde{x}_{ij} = x_{ij} - \frac{1}{n} \sum_{K=1}^{n} x_{ik}. \quad (3.14)$$

We can use this formula because each row of the data matrix $D$ contains one coordinate of the projections of all the $n$ points in image $i$. If you subtract the average of the coordinates from all the coordinates, the centre (average point) will be at the centre of the coordinate system.
3.3.2 Factorizing $D$

We factorize $D$ with the singular value decomposition (SVD). This will give us the matrices $U$, $W$ and $V$ with the property: $D = U W V^T$. In [18] they derived the rank theorem. This theorem says that without noise, the data matrix with the centred points is at most of rank three. This is due to the fact, that the data matrix $D$ is highly redundant. But in reality we do have noise which leads to a data matrix with a rank $> 3$. Therefore we need to reduce the rank of our SVD. To do this, we simply set all the singular values to 0 except for the first 3. Doing this, we get the matrices $U_3$, $W_3$ and $V_3^T$. Where $U_3$ is the matrix we get, if we simply take the first three columns of $U$, for $W_3$ we only take the upper left $3 \times 3$ block of $W$ and for $V_3^T$ we only take the first three rows of $V^T$.

3.3.3 Motion and Shape Matrices

Now we know that $D = U_3 W_3 V_3^T$. One possible decomposition to get the motion matrix $M$ and the shape matrix $S$ is: $M = U_3 W_3^{1/2}$; $S = W_3^{1/2} V_3^T$. This way $D = M S$. Now we have the problem that this decomposition is not unique. If we take any invertible $3 \times 3$ matrix $Z$ and transform $M \mapsto MZ$ and $S \mapsto Z^{-1} S$ we get the same $D$. This is due to the fact that we have here an affine transformation and therefore we have to enforce some Euclidean constraints (for example we could force the image axes to be perpendicular).

We want the images to be orthographic. Therefore the image axes have to be perpendicular and the scale has to be normalized. So if we have the axis $a_1$ and $a_2$ this translates to the two equations: $a_1 \cdot a_2 = 0$ and $|a_1|^2 = |a_2|^2 = 1$. This leads to $3m$ equations in $L = C \times C^T$:

$$A_i L A_i^T = I_d,$$  (3.15)

where $I_d$ is the identity matrix and $i = 1, \ldots, m$. Now we solve this for $L$. Then we can compute $C$ with the Cholesky decomposition: $L = C C^T$. Now we just have to update $M = MC$ and $S = C^{-1} S$. This way we have eliminated the affine ambiguity.

3.4 Reconstructing 3D

Until now we saw the DPM, the 3D DPM and the factorization Method. Now it is time to put them all together to reconstruct a 3D model of images of different instances of the same object type. Therefore we use the 3D DPM to get the correspondences in the different images. Then we can use the factorization method to calculate a 3D model.

3.4.1 Algorithm

We have a set of images which show different cars from different view points. To get a 3D model of the car, we can use the following procedure:

- Use the pre-trained 3D DPM to detect the components of the car on the different images and get their correspondences. This will give us $n$ features for the $m$ images.
3.4. RECONSTRUCTING 3D

- Now centre the feature coordinates using equation 3.14.

- Then construct the data matrix \( D \). \( D \) has the dimensions \( 2m \times n \). In the rows \( 2i - 1 \) and \( 2i \) we have the coordinates of the projections of all the \( n \) points in image \( i \). In the column \( j \) we have the projection of point \( j \) in all the views.

- Now we can factorize \( D \). We do it as it is described in section 3.3.2. This way we get \( U_3 \), \( V_3 \) and \( W_3 \).

- Compute the shape and motion matrices with the formulas: \( M = U_3W_3^{1/2} \) and \( S = W_3^{1/2}V_3^T \).

- Eliminate the affine ambiguity.
In this chapter you will read about the experiments and the results of this thesis. First it is described hot the 3D DPM car model was trained. You will find information about the number of viewpoints and the training data set. Then the test set which was used to get the 3D models of cars will be described. Finally you will see the resulting 3D models that were gained.

4.1 Trained Model

The goal was to find as many corresponding points between the different pictures as possible and then calculate a 3D model of an object with pictures of different objects which are all of the same type. In this case: pictures of different cars. Normally if one tries to get a 3D model from different images, the images show all the same object. Therefore there are a lot of correspondences that can be found. In this case one can not directly find corresponding points between the images, since they show different cars. The cars can vary in shape, colour and size. Therefore a 3D DPM model was trained to be able to detect cars and their parts in images. This 3D DPM model has 16 different viewpoints trained. Each part of the car that is detected, will be one corresponding point. Since there are way less parts in the 3D DPM model, than in classical 3D reconstruction, the model will have way less details. But nonetheless, we should be able to see a car in the point cloud.

4.2 Test Pictures

I used the EPFL GIMS08 dataset [10] as test images, because it was not used during the training of the 3D DPM model. This dataset contains images taken at the Geneva International Motor Show 2008. The cars were positioned on rotating platforms. The camera (a Nikon D70) was positioned on a stationary tripod. Then they took pictures approximately every 3-4 degrees. The lens in the camera is a Nikkor 12-24mm DX F/4. They kept the focal length constant while taking pictures of a single car, but they let it vary from sequence to sequence. They set the focus to manual, approx. at the hyperfocal distance.

The images from figures 4.1 and 4.2 are all from this dataset. Since the 3D DPM model had 16 different views trained, 16 of the 20 sequences from the
4.3. RESULTS

The calculated 3D point clouds of cars are shown in the figures 4.3 and 4.4. In the figures 4.1 and 4.2 you can see all images that were chosen from the dataset by the algorithm for one calculated 3D model, in this case from model 14 (Figure 4.4g).

First let us talk about the chosen car images. For each view the model had to choose the image with the highest score. As you can see from the car images (car 01 - 16), the cars have all different angles to the camera. And if you go from picture to picture, it looks like the cars are slowly turning around their axis. We have an exception in picture 4.1g. There the detection of the car is completely wrong. That is why the car’s angle does not fit in the rotating order of the others. There are two possible explanations why the algorithm chose this image, either the detection we see in the top left corner has an extremely high detection score, or the score is low but in the images where the angle would be approximately correct the neighbouring views scored higher detections.

I let the algorithm calculate multiple different car models. You can see some of them in the figures 4.3 and 4.4. As one can see, the models have quite a variation between them. This is due to the fact that the cars in each image have very different shapes. In normal 3D reconstruction one does have multiple images of the same instance of the object (in this case it would be the same car). These 3D reconstructions normally have way more points, since you get a point in the cloud for each corresponding point the algorithm can find. Since they have images of the same object, state of the art software can find hundreds to thousands of points (for example [13], [16], [15]). This reconstruction can not find correspondences in a conventional way. The images that were used contain
Figure 4.1: Images of cars taken at the Geneva International Motor Show 2008. These images were not used during the training of the 3D DPM. Each image already shows the bounding boxes that were calculated for the cars. The green bounding boxes contain the cars, the blue ones parts of the cars.
Figure 4.2: Images of cars taken at the Geneva International Motor Show 2008. These images were not used during the training of the 3D DPM. Each image already shows the bounding boxes that were calculated for the cars. The green bounding boxes contain the cars, the blue ones parts of the cars.
4.3. RESULTS

Figure 4.3: Calculated 3D models of the cars. Each image shows a different model calculated by the code. See section 4.3 for a detailed description of the calculated models.
4.3. RESULTS

Figure 4.4: Calculated 3D models of the cars. Each image shows a different model calculated by the code. See section 4.3 for a detailed description of the calculated models.
cars that vary in shape, colour and car brand. Therefore the 3D DPM model
searches the car in the image and its parts belonging to the view. For the car
and each part of the car we only get a bounding box. Each part of the car is
somewhere in its bounding box. There is no way to tell where exactly that is
and we do not get many bounding boxes per view. A simple approximation is
to take the centre of the bounding box. This approximation costs us a lot of
precision and it can vary a lot from car model to car model.

The other reason that it is hard to see a car in these 3D car models is the
following: Imagine how we as humans check if there is a car in an image. We
have an idea (or model) of cars in our head. These models vary from person to
person in depth of details and shapes and so on. But in general we can say that
most people would describe a car like this: It has four tires, three to five doors,
a roof, a car front and back, an exhaust pipe and so on. On the other hand,
computers can not see the pictures. They only see a lot of numbers. We use a
3D DPM model to “find” the car in the picture (or numbers in the view of the
computer). The 3D DPM model was not told which parts it should learn. So
we do not have a guaranty that the model has the same car parts in mind as a
human would. One would expect to see four points for the four tires of a car,
but maybe the trained model does not have a part for each tire. The same goes
for windows and doors.

Of course the models created by this code are not perfect. But after all
there is a reason that they use in 3D reconstruction different images of the same
object and not only same object class. But nonetheless it was an interesting
idea and the generated models have a vague car shape. So this thesis shows
that it is possible to create a 3D model if you have different images of different
instances of the same object type.
Chapter 5

Conclusion

In this thesis a new concept for reconstructing 3D models of objects was implemented. In classical 3D reconstruction one uses multiple images of the same object. The images can vary in camera location, rotation of the object or illumination of the scene. All these varying conditions can be used to gain the necessary information for a 3D reconstruction. This thesis contains a new approach for 3D reconstruction: Changing the object. More specifically we use different instances of the same object type. In other words: Is it possible to reconstruct a 3D model from different images of different cars? The implemented method showed that it is possible to get a 3D reconstruction, but the results are coarse. This comes from the fact that we have a lot less points in the 3D model than normal 3D reconstruction and since we use different cars in each image, it is not possible to get a perfect reconstruction of a car.

So the answer to the question, whether one can reconstruct a 3D model from different images of different cars, is a clear yes and no. In these experiments we got different point clouds which did not really look like cars. But all these point clouds only have between ten and twenty points. It is difficult to see a car in just eleven points. If there were more points in the cloud, or in other words more parts in the model of the 3D DPM, we would probably get better results.

5.1 Future Work

There are some possibilities how one could improve the result, but they are something for a future thesis: If one could improve the the 3D DPM in a way that it would learn more parts for one model, then we would have more points and therefore a more detailed 3D model. An other possibility would be to create a model, that has parts per part. Then one would not have to take the centre of the bounding boxes of the parts, but could instead use the location of the part of parts bounding boxes to locate the part better. This would improve the accuracy of the calculated 3D model.
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Illustration of the final outcome of the factorization method. For the measurements matrix $D$, motion matrix $M$, and shape matrix $S$ are calculated.</td>
</tr>
<tr>
<td>3.1</td>
<td>Perspective projection was used to render both of these images. On the left is a CAD example which was rendered on a background of mean pixels. On the right is a CAD example which was rendered with its annotated bounding box on a negative training image. These images are from [8].</td>
</tr>
<tr>
<td>3.2</td>
<td>Visualisation of the evolution of a mixture component of a 3D DPM with eight viewpoints during the training procedure. These images are from [8].</td>
</tr>
<tr>
<td>3.3</td>
<td>Visualization of part segmentations with their part bounding boxes. The green 2D box stands for the part filter template and the blue box represents a 3D part box from the given viewpoint.</td>
</tr>
<tr>
<td>4.1</td>
<td>Images of cars taken at the Geneva International Motor Show 2008. These images were not used during the training of the 3D DPM. Each image already shows the bounding boxes that were calculated for the cars. The green bounding boxes contain the cars, the blue ones parts of the cars.</td>
</tr>
<tr>
<td>4.2</td>
<td>Images of cars taken at the Geneva International Motor Show 2008. These images were not used during the training of the 3D DPM. Each image already shows the bounding boxes that were calculated for the cars. The green bounding boxes contain the cars, the blue ones parts of the cars.</td>
</tr>
<tr>
<td>4.3</td>
<td>Calculated 3D models of the cars. Each image shows a different model calculated by the code. See section 4.3 for a detailed description of the calculated models.</td>
</tr>
<tr>
<td>4.4</td>
<td>Calculated 3D models of the cars. Each image shows a different model calculated by the code. See section 4.3 for a detailed description of the calculated models.</td>
</tr>
</tbody>
</table>
Bibliography


Erklärung

gemäss Art. 28 Abs. 2 RSL 05

Name/Vorname: ........................................................................................................

Matrikelnummer: ........................................................................................................

Studiengang: ........................................................................................................

Bachelor □    Master □    Dissertation □

Titel der Arbeit: ........................................................................................................

........................................................................................................

........................................................................................................

LeiterIn der Arbeit: ........................................................................................................

........................................................................................................


..................................................................

Ort/Datum

..................................................................

Unterschrift