FAST: Fast Adaptive Silhouette Area based Template Matching

Daniel Mohr
dmoh@tu-clausthal.de
Clausthal University, Germany

Gabriel Zachmann
zach@tu-clausthal.de
Clausthal University, Germany

Abstract

Template matching is a well-proven approach in the area of articulated object tracking. Matching accuracy and computation time of template matching are essential and yet often conflicting goals.

In this paper, we present a novel, adaptive template matching approach based on the silhouette area of the articulated object. With our approach, the ratio between accuracy and speed simply is a modifiable parameter, and, even at high accuracy, it is still faster than a state-of-the-art approach. We approximate the silhouette area by a small set of axis-aligned rectangles. Utilizing the integral image, we can thus compare a silhouette with an input image at an arbitrary position independently of the resolution of the input image. In addition, our rectangle covering yields a very memory efficient representation of templates.

Furthermore, we present a new method to build a template hierarchy optimized for our rectangular representation of template silhouettes.

With the template hierarchy, the complexity of our matching method for n templates is $O(\log n)$ and independent of the input resolution. For example, a set of 3000 templates can be matched in 2.3 ms.

Overall, our novel methods are an important contribution to a complete system for tracking articulated objects.

1 Introduction

Tracking an articulated object is a challenging task, especially, if the configuration space of the object has many degrees of freedom (DOF), e.g., the human hand has about 26 DOF. Most tracking approaches require the object to be in a predefined state. If this is not desired or impractical, one has to search the whole configuration space at initialization time. Such a global search, of course, only needs to find a coarse object state, which typically consists of two parts. The first part is a similarity measure between the object in each configuration and the observed object (e.g. images from a camera). The second part is an algorithm to combine the best matching configurations, detect and eliminate false positives, potentially reduce the search space, and estimate the final object state. The focus of this paper is the first part of such a global search method.

Tracking articulated objects, there is a large number of object configurations that have to be compared with an input image. Therefore, this comparison should be extremely fast.

BMVC 2010 doi:10.5244/C.24.39

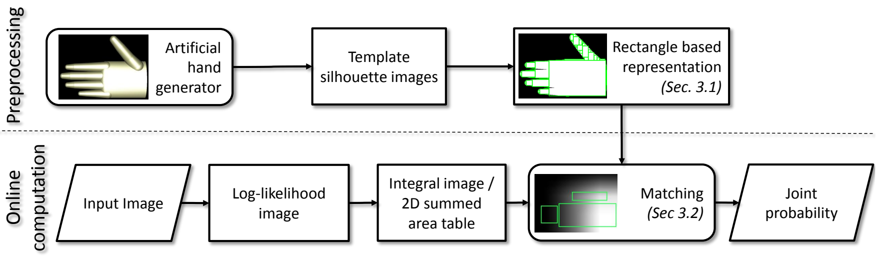
Preprocessing
Artificial hand generator
Template silhouette images
Rectangle based representation (Sec. 3.1)
Online computation
Input Image
Log-likelihood image
Integral image / 2D summed area table
Matching (Sec 3.2)
Joint probability

Figure 1: Overview of our approach using rectangle sets to approximate a silhouette. This speeds up the matching by a factor 5–30 compared to the approach proposed by Stenger *et al.* [15].

We propose a novel method for very fast approximate area silhouette comparison between model templates and input images. For one comparison, Stenger et al. [15] achieved a computation time proportional to the contour length of the template silhouette. We propose a new method, which reduces the computation time to be *independent* of the contour length and image resolution. It only depends on the desired accuracy of the template representation. This accuracy is a freely adjustable parameter in our approach. To achieve this, we first approximate all template silhouettes by axis-aligned rectangles, which is done in a preprocessing step. In the online phase, we compute the integral image [3, 18] of the segmented input image. With this, the joint probability of a rectangle to match an image region can be computed by four lookups in the integral image. Moreover, we present an algorithm to build a template hierarchy that can compare a large set of templates in sublinear time.

Our *main contributions* are:

1. An algorithm that computes a representation of arbitrary shapes by a small set of axis-aligned rectangles with adjustable accuracy. This results in a resolution-independent, very memory efficient shape representation.

2. An algorithm to compare an object silhouette in $O(1)$. In contrast the algorithm proposed by [15] needs $O(\text{contour length})$.

3. We propose an algorithm to cluster templates hierarchically guided by their mutually overlapping areas. This hierarchy further reduces the matching complexity for n templates from $O(n)$ to $O(\log n)$.

We assume that the object to be tracked can be segmented in the input image. Mostly, background subtraction or color segmentation approaches are used to achieve this. The result of a segmentation is a probability map, which supplies for each pixel the probability that this pixel belongs to the foreground (target object) or background. For our approach, there is no need to binarize this probability map.

It should be obvious that our proposed methods are suitable for any kind of object. For sake of clarity, though, we will describe our novel methods in the following by the example of the human hand, since human hand tracking is our long-term goal. This includes the full 26 DOFs of the hand, not only a few poses. To achieve this challenging task, we mainly use two different features for matching: edge gradients and skin color. In this paper, we focus on the skin color feature. We use a skin segmentation algorithm that computes for each image pixel the probability to represent skin or background, respectively. We generate our templates by an artificial 3D hand model. This model can be rendered in any desired state, and it can be easily projected onto 2D and binarized to get the hand silhouette. Given an input image, the goal then is to find the best matching hand silhouette.

We use the joint probability as proposed by Stenger *et al.* [15] to compare the silhouettes with the segmentation. A simple area overlap, of course, could be used, too. The only difference is that the sum instead of the product of probabilities would have been computed. For details, see Sec. 3.

2 Related Work

A lot of object tracking approaches based on silhouette comparison have been proposed. The approaches can be divided into two classes. The first class needs a binary silhouette of both, the model and the query image. The second class compare binary model silhouette area with the likelihood map of the query image.

A simple method belonging to the first class is used in [7, 20]. The difference between the model silhouette and segmented foreground area in the query image is computed. The exponential of the negative squared difference is used as silhouette matching probability. A slightly different measure is used by Kato *et al.* [5]. First, they define the model silhouette area A_M, the segmented area A_I and the intersecting area $A_O = A_I \cap A_M$. The differences $A_I - A_O$, $A_M - A_O$ and $A_I - A_M$ are integrated in the same way, as described above, into the overall measure. In [13], the non-overlapping area of the model and the segmented silhouettes are integrated into classical optimization methods, e.g. Levenberg-Marquardt or downhill simplex. Nirei *et al.* [11] first compute the distance transform of both the input and model silhouette. Regarding the distance transformed images as vectors, they compute the normalized scalar product of these vectors. Additionally, the model is divided into meaningful parts. Next, for each part, the area overlap between the part and the segmented input image is computed. Then, a weighted sum of the quotient between this overlap and the area of the corresponding model part is computed. The final similarity is the sum of the scalar product and the weighted sum. In [1, 14] a compact description of the hand model is generated. Vectors from the gravity center to sample points on the silhouette boundary, normalized by the square root of the silhouette area, are used as hand representation. During tracking, the same transformations are performed to the binary input image and the vector is compared to the database. A completely different approach is proposed by Zhou and Huang [22]. Although they extract the silhouette from the input image, they use only local features extracted from the silhouette boundary. Their features are inspired by the SIFT descriptor [9]. Each silhouette is described by a set of feature points. The chamfer distance between the feature points is used as similarity measure.

All the aforementioned approaches have the same drawback: to ensure that the algorithms work, a binary segmentation of the input image of high quality is necessary. The thresholds, needed for the binarization, are often not easy to determine.

To our knowledge, there are much less approaches working directly on the color likelihood map of a segmentation. In [21] the skin-color likelihood is used. For further matching, new features, called likelihood edges, are generated by applying an edge operator to the likelihood ratio image. But, in many cases, this leads to a very noisy edge image. In [15, 16, 17], the skin-color likelihood map is directly compared with hand silhouettes. The product of all skin probabilities at the silhouette foreground is multiplied with the product of all background probabilities in the template background. Stenger *et al.* [16] proposed a method for the efficient computation of this joint probability. The row-wise prefix sum in the log-likelihood image is computed. The original product along all pixels in a row reduces to three lookups in the prefix sum. Thus, the complexity to compute the joint probability is linear in the number of pixels along the template border.

Nevertheless, the above mentioned approach has some disadvantages. First of all, the template representation is resolution dependent. Typically, the distance of the object from the camera is not constant, and thus different sizes of the templates need to be considered. Consequently, for each scale, an extra set of the templates has to be kept in memory. Also, the higher the resolution of the images, the higher is the matching cost.

Our approach does not have all these disadvantages.

3 Silhouette representation

In the rest of this paper, we will denote the *template silhouette* simply as *template*. To avoid the issues mentioned in the previous section, we propose a novel *resolution-independent representation* of templates, which is the key to our fast matching approach. We propose to approximate a template by a set of axis-aligned mutually disjoint rectangles. With such a representation, one can perform template matching at arbitrary resolutions in constant time with respect to the template size. Figure 1 shows an overview of our approach.

We denote the integral image of a gray scale image I by $I\!I$:

$$I\!I(x,y) = \sum_{\substack{0 \le i \le x \\ 0 \le j \le y}} I(i,j) \tag{1}$$

Let R be an axis-aligned rectangle with upper left corner $\mathbf{u}$ and lower right corner $\mathbf{v}$, both inside I. The sum of the area R of all pixels in I is given by

$$\sum_R I(i,j) = I\!I(\mathrm{v}_x, \mathrm{v}_y) + I\!I(\mathrm{u}_x - 1, \mathrm{u}_y - 1) - I\!I(\mathrm{v}_x, \mathrm{u}_y - 1) - I(\mathrm{u}_x - 1, \mathrm{v}_y) \tag{2}$$

Let T with $T(x,y) \in \{0,1\}$ be a binary image representing a template. Let S and $\bar{S}$ denote the set of foreground and background pixels in T, resp. We compute a set of n mutually non-overlapping rectangles $\mathscr{R} = \{R_i\}_{i=1\cdots n}$ that cover S.

3.1 Rectangle Covering Computation

In the following, we denote a set of rectangles approximating S with $\mathscr{R}_S$. To obtain a good approximation, we minimize the non-overlapping area A of S and $\mathscr{R}_S$:

$$A = \min_{\mathscr{R}_S} \left| (S \cup \bigcup_{R_i \in \mathscr{R}_S} R_i) \setminus (S \cap \bigcup_{R_i \in \mathscr{R}_S} R_i) \right| . \tag{3}$$

Obviously, there is a trade-off between A and $n = |\mathscr{R}_S|$: the smaller the number of rectangles, the faster the matching is, but also the more inaccurate. We can utilize this to obtain an adaptive template representation.

There is a large body of work solving similar problems. One has to differentiate between rectangle covering [4, 6, 19] and partitioning problems [8, 12]: covering allows an arbitrary overlap among the rectangles in $\mathscr{R}_S$ while partitioning does not. Most covering and partitioning algorithms compute solutions under the constraint that the rectangles lie completely

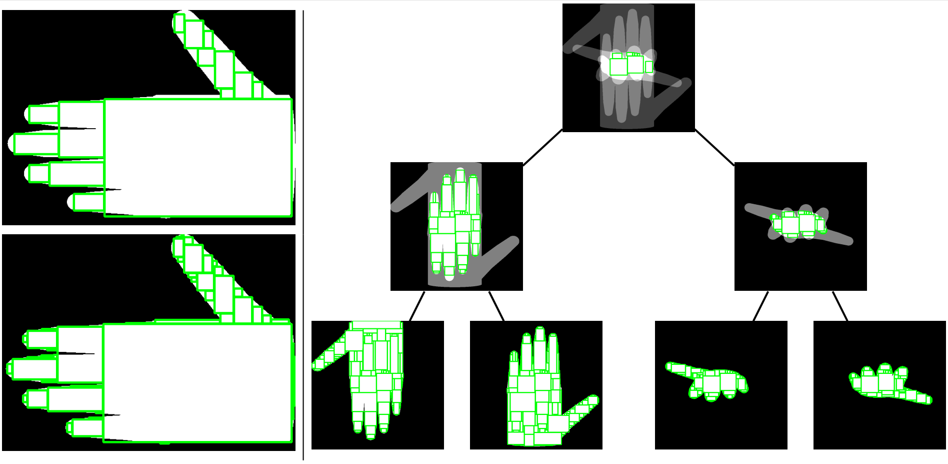

Figure 2: An example template approximated by a set of rectangles. The *left* column shows rectangles approximating the foreground at an accuracy of 85% (top) and 95% (bottom). The *right* one shows the template hierarchy generated by our approach in Sec. 3.3. For the sake of clarity, only the rectangles approximating the foreground are shown.

inside the polygon to be covered. Our problem is similar to standard partitioning in that we do not allow overlaps between the rectangles in $\mathscr{R}_S$, but it differs from partitioning because we can allow rectangles to cover a small part of the template background $\bar{S}$, too. In fact, we even encourage a solution where some rectangles lie slightly outside so as to keep the number of rectangles at a minimum. The reason is that, in the real world, S never perfectly matches the observed real hand anyway. Therefore, we can allow $A > 0$, which usually leads to solutions with much smaller numbers of rectangles in $\mathscr{R}_S$. To our knowledge, no algorithm has been presented that computes such an approximate rectangle covering. In the following, we present a simple and fast algorithm to obtain a solution for $A < \delta$.

First, the model (here, the human hand) is rendered at a given state, rasterized at a high resolution. and, after thresholding, a binary image T is obtained. We propose a greedy algorithm to compute the rectangle based template representation $\mathscr{R}_S \subset \mathscr{R}(T)$, where $\mathscr{R}(T)$ denotes the set of all rectangles in the image T. For an axis-aligned rectangle $R \in \mathscr{R}(T)$ we define its benefit:

$$F(R) = \sum_{\mathbf{x} \in R} (T(\mathbf{x}) - \tau) \tag{4}$$

The parameter $\tau \in [0,1]$ allows us to control the trade-off between *covering background regions* and *not covering foreground regions*. It controls the penalty for background pixels covered by a rectangle in the solution $\mathscr{R}_S$.

We initialize $\mathscr{R}_S$ with the empty set. At each iteration j in the greedy algorithm, we try to find $R_j^{\text{opt}} = \arg\max_{R \subset \mathscr{R}(T)} F(R)$. Because we have no knowledge about R_j^{opt}, we use a two-step search strategy to estimate this rectangle. Our search strategy basically works as follows:

The *first step* is a recursive search. For a rectangle/image X, we define the function

$$M(X) = \arg\max\{F(R) \mid R \in \mathscr{R}(X) \text{ with size } (\frac{\text{width}(X)}{2}, \frac{\text{height}(X)}{2})\} \tag{5}$$

At recursion level 0, we compute $R^0_{\max} = M(T)$. At recursion levels $i > 0$, we compute

$R^i_{\max} = M(R^{i-1}_{\max})$. We stop at recursion level k, if $R^k_{\max}$ is completely inside the foreground of T.

In the *second step*, we optimize the size of the rectangle $R^k_{\max}$. This is done by simply moving the rectangle borders so long as $F(R^k_{\max})$ grows. We obtain the final rectangle R^{opt}_j. Note that τ influences the optimization result. The higher τ is, the more covering a background pixel will be penalized. For example, if $\tau = 1$, then $R^i_{\max}$ will never cover any background pixel.

We add the rectangle R^{opt}_j to our final solution $\mathscr{R}_S = \mathscr{R}_S \cup \{R^{\text{opt}}_j\}$. The region R^{opt}_j is then erased from the foreground in T and the *first* and *second* step are repeated until the desired covering accuracy $A < \delta$ from Eq. 3 is achieved.

The algorithm in this form, however, has one problem. Consider, for instance, a rectangle R_1 that contains a large number of small foreground regions, i.e. $F(R_1)$ is large. Imagine another rectangle R_2, with $F(R_2) < F(R_1)$, but that contains just one fairly large region, which itself can contain a rectangle R'_2. It can happen that the area of R'_2 is larger than any of the foreground regions in R_1. However, the algorithm so far would still choose R_1 first.

To overcome this problem we extend the greedy algorithm by a *third* step. We test whether the rectangle R^{opt}_j from the *second* step is larger than a threshold r (at initialization, we set $r = \text{size}(T)$). If the test fails, we do not add the rectangle to $\mathscr{R}_S$ and further disable the region R^{opt}_j for the following iterations. If, at any time, the whole image T is disabled for search, all disabled search regions are enabled again for searching and r is set to the size of the largest rectangle found by the recursive search in the *first* step. Example coverings computed by the algorithm are shown in Figure 2.

F can be evaluated by four lookups in the integral image IT of T (see Eq. 2), which can be precomputed at initialization. Let w and h be the width and height of T then the complexity of the *first step* can be described by the following recursive formula:

$$T(w \cdot h) = c \cdot \frac{w}{2} \cdot \frac{h}{2} + T(\frac{w}{2} \cdot \frac{h}{2}) \tag{6}$$

which is in $O(w \cdot h)$. It is trivial to see that the complexity of the *second step* is $O(w \cdot h)$. The update cost of IT after erasing R^{opt}_j in T is also linear in the size of T. Therefore, the overall complexity of our rectangle covering algorithm is $O(|\mathscr{R}_S| \cdot w \cdot h)$.

For *step two*, we also tried the well-known Nelder-Mead optimization (Numerical Recipes implementation). In our experience, however, the quality of the resulting rectangles was never better and sometimes much worse, while the computation time was about a factor *1000* higher.

3.2 Matching Templates

In the previous section, we have developed an algorithm to compute for each template a resolution-independent compact representation consisting of axis-aligned rectangles. In the following, this representation will be used for fast template matching.

Our goal is to compare a template S with an input image I at a given position $\mathbf{p}$ using the joint probability (see Stenger *et al.* [16]). The first step is the foreground/background segmentation. We use the color likelihood instead of the binary segmentation due to its higher robustness against noise and imperfect segmentation. In the following, the color likelihood image of an input image I is denoted with $\tilde{L}$ with $\tilde{L}(x,y) \in [0,1]$. To convert the product in the joint probability into sums, we take the pixel-wise logarithm: $L(x,y) = \log \tilde{L}(x,y)$.

Utilizing Eq. 2, we can compute the joint probability at position $\mathbf{p}$ by:

$$P_S(\mathbf{p}) = \sum_{R_i \in \mathscr{R}_S} \left(\mathit{IL}\left(\begin{pmatrix} \mathrm{v}_x^i \\ \mathrm{v}_y^i \end{pmatrix} + \mathbf{p}\right) + \mathit{IL}\left(\begin{pmatrix} \mathrm{u}_x^i \\ \mathrm{u}_y^i \end{pmatrix} + \mathbf{p}\right) - \mathit{IL}\left(\begin{pmatrix} \mathrm{v}_x^i \\ \mathrm{u}_y^i \end{pmatrix} + \mathbf{p}\right) - \mathit{IL}\left(\begin{pmatrix} \mathrm{u}_x^i \\ \mathrm{v}_y^i \end{pmatrix} + \mathbf{p}\right) \right) \qquad (7)$$

IL denotes the integral image of the log-likelihood image L. The rectangle set $\mathscr{R}_S$ approximates only the template foreground. To get the appropriate match probability for a template, one has to take into account the background distribution, too.

Fortunately, the set of background pixels $\bar{S}$ of a template image, obviously, can be approximated by a set of rectangles with the same algorithm described in the last section. Having computed $\mathscr{R}_{\bar{S}}$, we can compute $P_{\bar{S}}$. P_S and $P_{\bar{S}}$ are resolution-dependent and need to be normalized appropriately. The final joint probability is

$$P = \exp\left(\frac{1}{2}(P_S + P_{\bar{S}})\right) \qquad (8)$$

where $P_{\bar{S}}$ is the background joint probability. Treating the joint probabilities for the foreground and background equally takes care of the fact that different template shapes have different area relative to their bounding box used in the template: in a template with fewer foreground pixels, the matching of the background pixels should not have a bigger weight than the foreground pixels and vice versa.

To determine the size of the target object one has to match the templates at different scales. This can be done easily by scaling the corner values for all rectangles accordingly. No additional representation has to be stored. Comparability between the same template at different sizes is ensured by the normalization.

3.3 The Template Hierarchy

In the previous section, we have described a novel method to match an arbitrary template T to an input image I. In a typical tracking application, especially when dealing with articulated objects, a huge number of templates must be matched. A suitable approach to reduce the complexity from $O(\#\text{templates})$ to $O(\log(\#\text{templates}))$ is to use a template hierarchy. However, building a well working one is still a challenging task.

We propose an approach to build a hierarchy that naturally fits with our representation of the templates by rectangles. In addition, it even further reduces the computational effort per template matching. We build the tree structure by utilizing a hierarchical clustering algorithm. Vectors, describing the similarity between templates, are computed and used as input for the clustering algorithm [2]. The output are k disjoint clusters, where k defines the number of children per tree node. At each node in the template tree, rectangles covering the intersection of all templates of all children are pre-stored.

For matching n templates, we traverse the hierarchy. The rectangles from the root to one leaf constitutes a covering of that template, which is thus being matched incrementally during traversal. At the same time, we prune large parts of the hierarchy (i.e. large numbers of templates), because we descend only into those sub-trees with largest probability. If, at any state in a tracker, there is no need or not possible to perform an accurate object detection, e.g. if the object moves extremely fast, one can stop to traverse the hierarchy at any depth. This is reasonable, because any inner node in the template tree is a coarser representation of the templates in the sub-tree. Figure 2 illustrates the basic idea of our template hierarchy. For a detailed description of the hierarchy generation and matching, please take a look into our technical report [10]

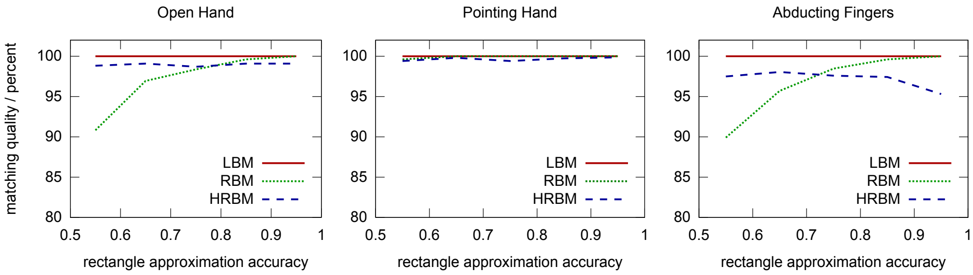

Figure 3: The matching quality of our two methods (RBM and HRBM) Just for reference, the quality of LBM [15] is also plotted (LBM has no notion of template accuracy).

4 Results

In our experiments, we have analyzed two aspects. The first one is the quality of our algorithm from Sec. 3.1 to compute a representation of templates by sets of axis-aligned rectangles. The second one measures and compares the quality and computation time of the template matching itself. In all experiments we have set the control parameter $\tau = 0.95$ (see Eq. 4).

4.1 Quality of our rectangle representation

First, we evaluated the quality of our approach approximating templates by axis-aligned rectangles. As quality measure we used the ratio of the benefit of the covering to the benefit of a perfect covering:

$$q = \frac{\sum_{R \in \mathscr{R}_S} F(R)}{F(\text{foreground}(T))} \tag{9}$$

In our experiments, we tried to cover a representative set of postures and orientations. We observed that, on average, for an accuracy of $q = 0.55/0.65/0.75/0.85/0.95$, we need about 5/9/20/132/768 rectangles respectively. (The images had a resolution of 1024×1024)

4.2 Evaluation of the matching quality

We compare our approach with a state-of-the-art approach proposed by Stenger *et al.* [15], because our approach was inspired by theirs and the application (hand tracking) is the same.

In the following, we will denote the algorithm from [15] as *line-based matching (LBM)*, ours as *rectangle-based matching (RBM)*, and ours including the hierarchy *hierarchical RBM (HRBM)*. It is not quite fair to compare a hierarchical approach to non-hierarchical ones, but we add the results of the hierarchical match to our plots to analyze the potential of the hierarchy.

In the following, we will evaluate the difference between the methods with regard to resolution-independence, computation time, and accuracy. We generated templates with an artificial 3D hand model. We used the templates also as input images. There are two reasons to use such synthetic input datasets. First, we have the ground truth and, second, we can eliminate distracting influences like differences between hand model and real hand, image noise, bad illumination, and so on.

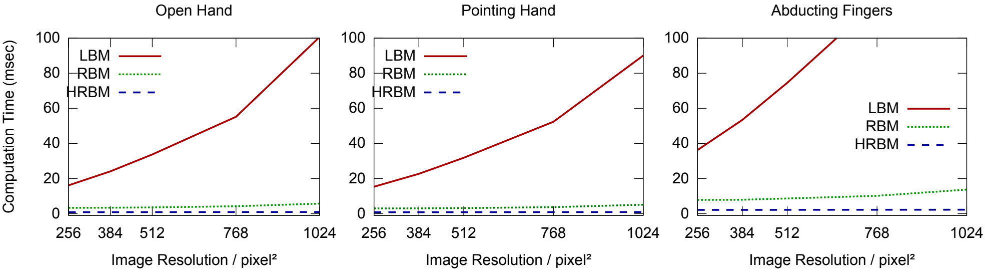

Figure 4: Each plot shows the average computation time for all three approaches: LBM [13], RBM (our approach), HRBM (our approach incl. hierarchy). Clearly, our approaches are significantly faster and, even more important, resolution independent.

We generated three datasets for evaluation. Dataset 1, consisting of 1536 templates, is an open hand at different rotation angles. Dataset 2 is a pointing hand rendered at the same rotation angles as dataset 1. In dataset 3, consisting of 3072 templates, we used an open hand with abducting fingers (moving fingers in the hand plane). Additionally, for each position of the fingers, we rendered the model at different rotations.

First, we compared the matching quality of the three approaches. RBM and HRBM were evaluated at five different template approximation accuracies (see Eq. 9). We expected LBM to work best on the artificial datasets because, for each input image, there is one exactly matching template. For evaluation we used an input image resolution of 256 × 256 and compared each template at 5 different scalings (from 70 × 70 up to 200 × 200). All three approaches always found the correct location of the hand in the input image. Thus, for evaluation, we define the matching quality as the ratio of

$$\frac{\text{\# frames where the correct template was ranked among the top 10}}{\text{\# frames in the input sequence}} \qquad (10)$$

at the correct position in the images. Please see Fig 3 for the results.

The quality of RBM is as expected: the higher the rectangle approximation accuracy is, the higher the matching quality is. One notices that the matching quality in the *pointing hand* dataset is very high even at low rectangle approximation accuracy. We have taken a closer look at the datasets and found that the *pointing hand* has fewer similar 2D template shapes at different hand state parameters (i.e. less information loss during projection from 3D to 2D).

The results for HRBM at higher accuracy are slightly lower compared to RBM. The reason is that, in contrast to utilizing a “flat” set of templates, the matching algorithm never considers *all* templates because the tree traversal prunes large portions of the set of templates, which is the purpose of a hierarchy. Thus, utilizing any kind of hierarchical matching increases the likelihood of missing the best match, because that could happen to reside in a branch that was pruned. The quality of HRBM also depends on the clustering algorithm and the rectangles used to approximate the templates. This is the reason for the decreasing matching quality at the *abducting fingers* dataset at higher rectangle approximation accuracy.

Second, we examined the dependence between the input image resolution and computation time. We have decided to use RBM and HRBM at a rectangle approximation accuracy of 0.75 because the plots in Figure 3 show that the matching quality is at most 3% lower than in the LBM method. We used input images at 5 different resolutions. We averaged the time to compute the joint probability for all frames at 49 positions each. The result is shown in Figure 4. Clearly, LBM's computation time depends linearly on the resolution, while our approaches exhibit almost constant time.

5 Conclusions

In this paper, we have presented a fast, adaptive template matching approach based on silhouette area matching. It works by approximating the template silhouettes by a set of axis-aligned rectangles. The accuracy of this representation can be adjusted by a parameter at this stage. We have also proposed a novel greedy algorithm to compute such a rectangle covering. Additionally, we have presented a template hierarchy, which utilizes our representation of the templates. This hierarchy reduces the computational complexity of the matching algorithm for a set of templates from linear to logarithmic time. Again, we would like to point out that our contributions constitute just one of the many pieces of a complete hand tracking system.

Overall, we need about $4.5\mu s$ on average to compare one template to one position in an input image at an arbitrary resolution (without using the hierarchy). This is about a factor 15 faster than the state-of-the-art approach from [15] at a resolution of 1024×1024. Furthermore, the template representation is very memory efficient. For example, a template set consisting of 3000 templates needs less than 1.5 MByte storage space at an accuracy of 0.75.

In the future, we plan to implement our approach in a massively parallel programming paradigm. Furthermore, we will extend our hierarchical approach to a random forest approach, which we expect to improve the template matching quality significantly. To get different classifiers at each node, one can choose a random subset instead of all covering templates to cluster a tree node for further subdivision.

References

[1] Akihiro Amai, Nobutaka Shimada, and Yoshiaki Shirai. 3-d hand posture recognition by training contour variation. In *IEEE Conference on Automatic Face and Gesture Recognition*, pages 895–900, 2004. ISBN 0-7695-2122-3.

[2] Marie Cottrell, Barbara Hammer, Alexander Hasenfuß, and Thomas Villmann. Batch neural gas. In *5th Workshop On Self-Organizing Maps*, 2005.

[3] Franklin C. Crow. Summed-area tables for texture mapping. In *SIGGRAPH: Proceedings of the 11th annual conference on Computer graphics and interactive techniques*, volume 18, 1984.

[4] Laura Heinrich-Litan and Marco E. Lübecke. Rectangle covers revisited computationally. In *ACM Journal of Experimental Algorithmics*, volume 11, 2006.

[5] Makoto Kato, Yen-Wei Chen, and Gang Xu. Articulated hand tracking by pca-ica approach. In *International Conference on Automatic Face and Gesture Recognition*, pages 329–334, 2006. ISBN 0-7695-2503-2.

[6] V.S. Anil Kumar and H. Ramesh. Covering rectilinear polygons with axis-parallel rectangles. In *Annual ACM Symposium on Theory of Computing*, pages 445–454, 1999. ISBN 1-58113-067-8.

[7] John Y. Lin, Ying Wu, and Thomas S. Huang. 3D model-based hand tracking using stochastic direct search method. In *International Conference on Automatic Face and Gesture Recognition*, page 693, 2004. ISBN 0-7695-2122-3.

[8] W.T. Liou, Jimmy Jiann-Mean Tan, and R. C. T. Lee. Minimum rectangular partition problem for simple rectilinear polygons. In *IEEE Transactions on Computer-Aided Design*, volume 9, pages 720–733, 1990.

[9] David G. Lowe. Object recognition from local scale-invariant features. In *IEEE International Conference on Computer Vision*, volume 2, pages 1150–1157, 1999.

[10] Daniel Mohr and Gabriel Zachmann. Fast adaptive silhouette area based template matching. In *IfI Technical Report Series*, 2010. URL http://www.in.tu-clausthal.de/forschung/technical-reports/.

[11] Kenichi Nirei, Hideo Saito, Masaaki Mochimaru, and Shinji Ozawa. Human hand tracking from binocular image sequences. In *22th International Conference on Industrial Electronics, Control, and Instrumentation*, pages 297–302, 1996. ISBN 0-7803-2775-6.

[12] Joseph O'Rourke and Geetika Tewari. Partitioning orthogonal polygons into fat rectangles in polynomial time. In *In Proc. 13th Canadian Conference on Computational Geometry*, pages 97–100, 2001.

[13] Hocine Ouhaddi and Patrick Horain. 3D hand gesture tracking by model registration. In *Workshop on Synthetic-Natural Hybrid Coding and Three Dimensional Imaging*, pages 70–73, 1999.

[14] Nobutaka Shimada, Kousuke Kimura, and Yoshiaki Shirai. Real-time 3-d hand posture estimation based on 2-d appearance retrieval using monocular camera. In *IEEE International Conference on Computer Vision*, page 23, 2001. ISBN 1530-1044.

[15] Bjorn Stenger, Arasanathan Thayananthan, Philip H. S. Torr, and Roberto Cipolla. Model-based hand tracking using a hierarchical bayesian filter. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, volume 28, pages 1372–1384, 2006.

[16] Bjorn Dietmar Rafael Stenger. Model-based hand tracking using a hierarchical bayesian filter. In *Dissertation submitted to the University of Cambridge*, 2004.

[17] Erik B. Sudderth, Michael I. Mandel, William T. Freeman, and Alan S. Willsky. Visual hand tracking using nonparametric belief propagation. In *IEEE CVPR Workshop on Generative Model Based Vision*, volume 12, page 189, 2004.

[18] Paul Viola and Michael Jones. Rapid object detection using a boosted cascade of simple features. In *IEEE Conference on Computer Vision and Pattern Recognition*, volume 1, pages I–511–I–518, 2001.

[19] S.Y. Wu and S. Sahni. Covering rectilinear polygons by rectangles. In *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, volume 9, pages 377–388, 1990.

[20] Ying Wu, John Y. Lin, and Thomas S. Huang. Capturing natural hand articulation. In *International Conference on Computer Vision*, volume 2, pages 426–432, 2001. ISBN 0-7695-1143-0.

[21] Hanning Zhou and Thomas Huang. Tracking articulated hand motion with eigen dynamics analysis. In *IEEE International Conference on Computer Vision*, volume 2, pages 1102–1109, 2003.

[22] Hanning Zhou and Thomas Huang. Okapi-chamfer matching for articulated object recognition. In *IEEE International Conference on Computer Vision*, volume 2, pages 1026–1033, 2005.