Learn to Track Edges

Yanghai Tsin Yakup Genc Ying Zhu Visvanathan Ramesh

Real-Time Vision and Modeling Department
Siemens Corporate Research, Princeton, NJ, 08540, USA

Abstract

Reliability of a model-based edge tracker critically depends on its ability to establish correct correspondences between points on the model edges and edge pixels in an image. This is a non-trivial problem especially in the presence of large inter-frame motions and in cluttered environments. We propose an online learning approach to solving this problem. An edge pixel is represented by a descriptor composed of a small segment of intensity patterns. From training examples the algorithm utilizes the randomized forest model to learn a posteriori distribution of correspondence given the descriptor. In a new frame, the edge pixels are classified using maximum a posteriori (MAP) estimation. The proposed method is very powerful and it enables us to apply the proposed tracker to many previously impossible scenarios with unprecedented robustness.

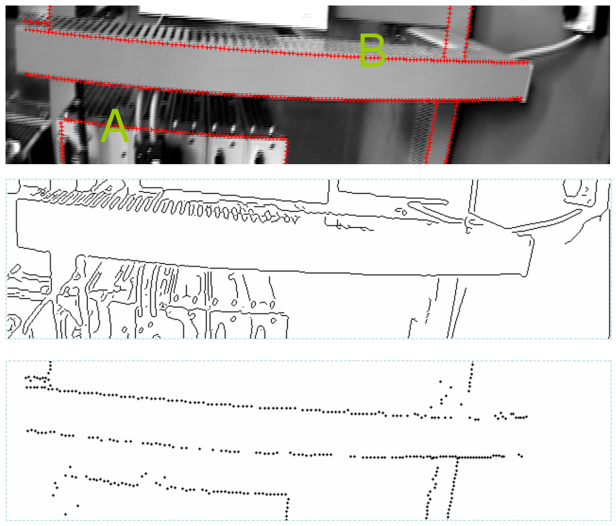

Figure 1. Top: input image with the model projection superimposed as red lines. Middle: Canny edge detector results (Matlab implementation). Bottom: the model-associated edge pixels classified using the proposed approach.

1. Introduction

The goal of this work is to design a precise and robust model-based edge tracker. Typical applications of the tracker include robot localization, robotic assembly and augmented reality. Edge tracking is an important technique in these areas largely because it is view-independent and it is resilient to moderate appearance changes. In some cases it is the only viable choice when there are no strong corner features in a scene. Edge tracking has been studied intensively since the inception of the computer vision research. However, a reliable tracker is still unavailable and researchers still deem edge trackers "brittle" [24]. They usually cannot survive the harsh conditions in real applications where clutter, large appearance change and fast motions are ubiquitous.

The main difficulty in edge tracking is to establish correspondences between points on the model edges and the edge pixels in an image. To fully appreciate the difficulty involved, we give an example in Figure 1. We would like to direct the reader's attention to regions A and B marked on the top image and their corresponding areas in the center one. Region A contains complex structures. Many irrelevant edge pixels are detected by an edge detector [6]. Identifying true correspondences becomes error-prone simply because there are too many possible candidates. The case is opposite for the line marked by B. The edge pixels corresponding to the model are missing. Almost all tracking failures can be traced back to false correspondences caused by the above two cases, clutter or missing edge pixels. In dynamic environments, the problem becomes even harder due to many factors, *e.g.*, lighting change, motion blur, scale variation and camera gain control.

Difficulties involved in binary edge detection can be partially alleviated by using intensity gradient [11]. Intensity gradient incorporates more context than a binary edge map. However, in cluttered environments these methods encounter similar complexity and they usually converge to a local minimum.

978-1-4244-1631-8/07/$25.00 ©2007 IEEE

We believe that to efficiently solve the problem, it is imperative to exploit the rich context that intensity contributes. By properly utilizing the intensity values, we can ease the correspondence problem to a large extent. There have been some work that incorporates statistical appearance models in the correspondence problem [5, 9]. However, these models are not powerful enough to handle the edge tracking problem due to several challenges: 1) the appearance distributions are much more complex than a single mode Gaussian model; 2) the distributions are dynamically changing; 3) the model needs to be estimated and updated online.

With the progress in the machine learning community, the appearance modeling problem has become feasible over the years. Statistical learning methods have been used for online tracking [3, 21] and most noticeably, the randomized forest method [1, 17] has shown promises in realtime feature matching problems. Our main contribution in this work is to use the randomized forest framework as a nonparametric model for appearance distributions around edge segments, to use the model for online classification and model updating, and ultimately to track edges. In Figure 1, bottom, we show a set of edge pixels that are classified as correspondences of an edge model. We refer to our tracker as the randomized forest (RF) tracker.

The paper is organized as following. In Section 2 we discuss the related work and address their limitations. The problem is formally formulated in Section 3. We dedicate Section 4 to describing the randomized forest model. In Section 5 details of training and classification are discussed. Experimental results are reported in Section 6 and we conclude the paper in Section 7.

2. Related Work

Edge tracking for pose estimation is a special case of model-based tracking of objects in videos (see [18] for a recent review of the topic for rigid objects). Most methods [2, 5, 8, 11, 12, 13, 14, 16, 19, 20, 22, 25, 27] explore the idea of mapping the model outlines to the edge or gradient images. For computational reasons, a set of points on the model likely to create strong edges or gradients in the image are chosen as the control points. These control points are projected to the image with the given starting pose to generate hypothesis yielding the highest amount of support edges in the image. Variations of this search ranges from the basic least-squares optimization (e.g., [14]) to robust estimators (e.g., [12]) that can handle multiple candidates.

One drawback of these approaches is the possibility of getting into to a local minimum during the minimization process. This could be due to many factors including the lack of details in the available model, the large inter-frame motion or the inaccuracy of the initial pose, and the amount of clutter in the scene. Even the robust methods have their limitations [24]. Further improvement achieved using other features in combination to edges such as corners and patches [25, 24]. These, however, help only when enough texture is available. Our approach exploits any available intensity information around the tracked edges while learning the posteriori distribution of the edge and non edge pixels.

Due to the fact that visual appearance of an object changes over time, a proper likelihood model needs to be learned and adapted during tracking to describe the variation and distinguish an object or its parts from the surroundings. Online learning for visual tracking has drawn increasing attention recently. Jepson et al. proposed a mixture parametric model to describe stable and sudden changes of object appearance as well as data outliers [15], where an online EM-algorithm was used to adapt model parameters over time. In the work of Collins *et al.* [7] and Wang *et al.* [26], the features used for tracking were adjusted online for maximal discrimination. In [3, 21], tracking a foreground object was treated as a classification problem and a binary classifier was trained online to separate foreground patches from background. Compared to the aforementioned work of blob tracking, our method extends the idea of online learning to improve the accuracy and robustness of tracking objects using parametric models. In our method, specific classifiers are learned online to describe the local appearance of individual edge segments. In addition, the 3D model obtained offline provides a strong constraint to reduce the chance of drifting that often occurs in online learning.

The randomized forest method is first introduced to the vision community by Amit [1] for shape recognition. Lepetit and Fua [17] applied the technique in realtime feature matching problems and demonstrated the power of the method. Boffy *et al.* [4] proposed to adapt the forest online to handle large illumination and scale changes. Our approach differs from both [17] and [4] in the application domain and descriptors. Their feature descriptors are corner like image patches while we use a simple intensity profile.

3. Problem Definition

The object to be tracked is modeled by a set of edges $\{\mathbf{e}\}$. Sampled points $\mathbf{X}_i$ on the model edges are projected to the image plane by a transformation $T(\mathbf{X}_i, \boldsymbol{\theta})$, where $\boldsymbol{\theta}$ is the vector of transformation parameters. T can be one of the image plane transformations such as Euclidean, affine, or homography, 3D to 2D perspective projection, articulated motion or active shape models [10]. In the case of perspective projection, we assume that the camera has been internally calibrated and we only need to estimate the external parameters (translation and rotation). We are not concerned with the initialization problem, *i.e.*, finding $\boldsymbol{\theta}_0$ such that $T(\cdot, \boldsymbol{\theta}_0)$ brings the model edges to perfect alignment with the object's image in the first frame. We assume that $\boldsymbol{\theta}_0$ is given. To track the object in all subsequent images at frame t, we use the pose estimated for frame $t-1$, $\boldsymbol{\theta}_{t-1}$, as the

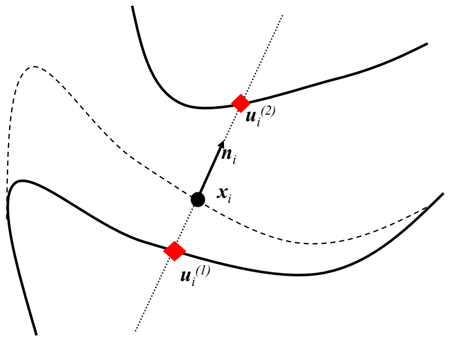

Figure 2. Illustration of edge tracking. The dashed line shows projection of a model edge. $\mathbf{u}_i^{(j)}$'s are detected edge candidates along the edge normal direction $\mathbf{n}_i$ at $\mathbf{x}_i$.

initialization. Automatic initialization algorithms are nowadays available. The interested reader is referred to [17].

3.1. Edge Detection

At the projection of a model point $\mathbf{x}_i = T(\mathbf{X}_i, \boldsymbol{\theta})$ we compute a unit length normal vector $\mathbf{n}_i$, see Figure 2. The positive direction of $\mathbf{n}_i$ is defined at modeling time and fixed. Over a search range of K pixels, we obtain a one dimensional intensity pattern, or the *edge profile*

$$\mathbf{f}_i(k) = \mathbf{I}(\mathbf{x}_i + k \cdot \mathbf{n}_i) \tag{1}$$

where $\mathbf{I}$ is the 2D image intensity function and the profile is extracted from a range $k \in [-K, K]$. To reduce the influence of noise and aliasing effects, $\mathbf{I}(\mathbf{x})$ can be convolved with a smoothing kernel before the profile is extracted.

We compute first order intensity gradient on the profile. All local gradient extrema with magnitude $|\nabla \mathbf{f}_i| > I_t$ are considered as edge candidates, where I_t is a threshold. Throughout this work we choose a fixed threshold of $I_t = 5$ intensity levels/pixel for 8-bit gray level images. By choosing such a low threshold, we can largely avoid misdetection. These detected edge candidates are denoted

$$\mathbf{u}_i^{(j)} = \mathbf{x}_i + k_i^{(j)} \cdot \mathbf{n}_i, \tag{2}$$

i.e., the j^{th} candidate along the normal direction and $k_i^{(j)}$ is the coordinate of the edge candidate in the profile $\mathbf{f}_i(k)$.

3.2. Extracting Descriptor

We couple each edge candidate $\mathbf{u}_i^{(j)}$ with an intensity-based descriptor

$$\mathbf{d}_i^{(j)}(m) = \mathbf{f}_i\left(k_i^{(j)} + m\right) \tag{3}$$

where $m \in [-M, M]$ and $M << K$, *i.e.*, a descriptor is simply a small segment of the intensity profile taken from the two sides of an edge candidate.

3.3. Problem Definitions

The next step in edge-based tracking is to find among all edge candidates $\mathbf{u}_i^{(j)}$ the true correspondence of $\mathbf{x}_i$. We take a probabilistic approach to the problem. From some training examples, we learn a likelihood model

$$P(\mathbf{d}|c_i) \tag{4}$$

where $c_i = 1$ means that a candidate is the true correspondence of $\mathbf{x}_i$ (thus true correspondence of $\mathbf{X}_i$) and $c_i = 0$ means false correspondence. Without any other information, we assume constant prior distribution

$$P(c_i = 1) = P(c_i = 0) = 0.5. \tag{5}$$

Using the Bayes rule, posteriori probability of correspondence given descriptor $\mathbf{d}_i^{(j)}$ is

$$P\left(c_i|\mathbf{d}_i^{(j)}\right) \propto P\left(\mathbf{d}_i^{(j)}|c_i\right). \tag{6}$$

The problem of identifying the true correspondence becomes

$$j^* = \underset{j}{argmax}\, P\left(c_i = 1|\mathbf{d}_i^{(j)}\right). \tag{7}$$

That is, the best candidate is the one with the maximum a posteriori (MAP) probability.

The tracking problem is subsequently defined as

$$\boldsymbol{\theta}^* = \underset{\theta}{argmin} \sum_i \rho\left(\left\|T(\mathbf{X}_i, \boldsymbol{\theta}) - u_i^{(j^*)}\right\|^2, \sigma\right) \tag{8}$$

where we adopt a robust function such as the Huber function $\rho(\cdot, \sigma)$ to handle the possible outliers using the M-estimator approach [23], where σ is a bandwidth parameter. Since parametrization of the transformation and solution to the optimization problem is fairly standard and it is not the focus of the current work, we refer the reader to [12, 27].

3.4. Clustering Points on An Edge Segment

One can learn one posteriori probability (6) for each sampled point $\mathbf{X}_i$. However, there is a more efficient approach. Observing that neighboring profiles on an edge usually look alike, we group all points on the same model edge segment and treat them as a single class. More precisely, we define a single posteriori distribution for all points on the same edge segment

$$P(c_\mathbf{e}|\mathbf{d}) \tag{9}$$

where $\mathbf{e}$ is a model edge segment and let

$$P(c_i|\mathbf{d}) = P(c_\mathbf{e}|\mathbf{d}) \tag{10}$$

for any $\mathbf{X}_i$ on the edge $\mathbf{e}$. This approach can significantly reduce the number of probabilistic models needed because a large set of points share a single model.

Our proposed edge tracker is insensitive to the definition of an edge segment, *i.e.*, where an edge segment begins and where it ends. In all of our experiments, model edge segments are built by our colleagues with no knowledge of our tracker. We leave the study of the effect of edge segmentation to tracker performance to our future work.

4. Learn to Correspond

Intensity patterns surrounding an edge segment change dramatically due to lighting variations, perspective distortions and moving foreground/background. Descriptors that we draw from a single edge segment is inevitably dynamic and multi-modal, *e.g.*, half in complex shadows casted by tree leaves and the other half sunlit. Success of an edge tracker depends critically on understanding of these complex variations. We take the approach of dynamically modeling these changes using a randomized forest.

4.1. The Randomized Forest Model

A *randomized forest* composes of a set of *randomized trees*. A randomized tree is a special type of decision tree. Like a decision tree, it is composed of one root node, a set of internal nodes, and a set of leaf nodes. In our study, each internal node has up to three children. At each internal node, there are a set of splitting rules that decide the path of a descriptor. We adopt splitting rule generation method proposed in [17], *i.e.*, by totally random choices independent of the input data. It has been shown that when the trees are deep enough and the number of trees are large enough, its performance is comparable to more sophisticated splitting rule generation approaches [17].

Our splitting rules are parameterized by three numbers, an intensity difference threshold t and two indices i_1 and i_2. To decide the path of a descriptor, intensities of the two entries are compared. The descriptor is dropped down to one of the three child nodes based on the following rules,

$$\begin{cases} \textit{left child} & \mathbf{d}(i_1) - \mathbf{d}(i_2) < -t \\ \textit{center child} & |\mathbf{d}(i_1) - \mathbf{d}(i_2)| \leq t \\ \textit{right child} & \mathbf{d}(i_1) - \mathbf{d}(i_2) > t \end{cases} . \tag{11}$$

We choose $t = 5$ intensity levels for 8-bit gray scale images throughout the work. The two indices are randomly chosen for each internal node independently.

4.2. Geometric Interpretation

We assume that an input image is 8-bit gray scale. As a result, a descriptor lives in a hypercube $\mathcal{C} \in \mathcal{R}^{2M+1}$ with each dimension delimited by $[0, 255]$. We illustrate a 3D example ($M = 1$) in Figure 3.

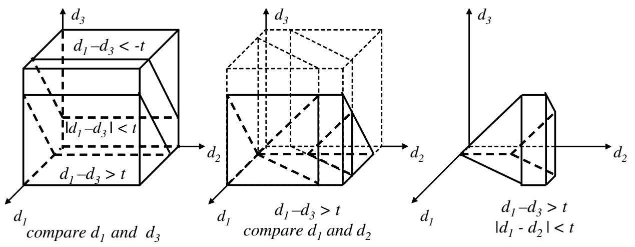

Figure 3. Geometric interpretations of the splitting rules.

At each internal node, a splitting rule divides the hypercube into three polytopes (generalization of polyhedra into higher dimensional spaces): two wedges and a slab in between, see Figure 3 left. The three polytopes are bounded by two hyperplanes and the surfaces of the hypercube. When decisions with respect to a splitting rule is made, the hypercube is reduced to one of the three polytopes. All descriptors reaching a leaf node belong to a polytope that is an intersection of all polytopes carved out by all the decisions. We denote the polytope corresponding to a leaf node as $\mathcal{D}$. A final polytope is shown in Figure 3, right.

To conclude, we see that in a randomized tree the splitting rules divide the descriptor space into ever smaller polytopes as the algorithm proceeds toward a leaf node. As the tree depth increases, the volumes of the polytopes shrink, and all descriptors in a polytope become more similar.

4.3. Leaf Node: Empirical Posteriori Distributions

Once a tree structure is determined and descriptors are defined in the training examples, we can accumulate empirical evidences in the leaf nodes. For the ease of understanding, let us assume that we train one forest for each edge segment $\mathbf{e}$. A more efficient approach will be discussed in Section 5.1.

Each training example $\mathbf{d}_i^{(j)}$ is associated with a label: $c = 1$ meaning that the descriptor describes an edge candidate of edge $\mathbf{e}$, or otherwise $c = 0$. At each leaf node, we count the true ($c = 1$) and false ($c = 0$) correspondences. Each leaf node effectively records the frequency of true candidates given that a descriptor has reached the leaf node. We denote this empirical distribution as $P(c|\mathcal{D}_n)$ where n is the leaf node index. With division of the descriptor space sufficiently fine (deep trees), it is assumed that

$$P(c|\mathbf{d}_i^{(j)}) \approx P(c|\mathcal{D}_n) \tag{12}$$

In this sense, leaf nodes store *direct estimates* of the *posteriori* probabilities (6) without resorting to the likelihood models (4). A randomized tree is a *piecewise constant* posteriori probability model, where each "piece" is a polytope. Notice that a randomized tree model is also a nonparametric model. It simply memorizes the empirical distributions.

A randomized tree naturally handles multi-modality of the descriptors. Each cluster of true descriptors ($c = 1$) are modeled by polytopes at different locations in the descriptor space, while false descriptors are likely to be scattered differently in different corners.

To reduce the granularity introduced by piecewise-constant approximation and make the posteriori distribution smoother, we average over several trees to approximate the empirical posteriori distribution,

$$P(c|\mathbf{d}_i) \approx \frac{\sum_{j=1}^{T} P(c|\mathcal{D}^{(j)})}{T} \tag{13}$$

where $\mathbf{d}_i \in \mathcal{D}^{(j)}$, $j = 1, 2, \ldots, T$, $\mathcal{D}^{(j)}$ is the polytope containing $\mathbf{d}_i$ in the j^{th} tree, and T the number of trees.

5. Online Training and Classification

5.1. Training Randomized Trees

The parameters of a randomized forest include depth of the trees and the number of randomized trees in a forest. Tree depths are empirically determined by cross validation on a large set of positive and negative descriptors. We describe the process in Section 6. Once the depth of a tree is estimated, training a classifier is very easy. It is accomplished by simply dropping all training samples down the tree and keeping counts of the sample histograms. In our experiments, we fix the number of trees in a forest to be 10.

We could train one random forest for each edge segment. To reduce the memory requirements, however, we use the same set of internal nodes and splitting rules for all edge segments in a model. At leaf nodes, we keep sample histograms separately for each edge segment.

5.2. Online Update

Yet another attractive feature of the randomized forest model is that updating the classifier is extremely easy and can be done online. After successfully tracking a new frame, we can update our classifiers using descriptors from the new frame. For each edge profile, edge candidates coincide with the edge projections are true correspondences. The rest are false correspondences.

In order to increase the predictive power of our classifier, we synthesize a set of random descriptors to update the randomized forest. Specifically, we randomly sample a set of points along the model edge (not necessarily coinciding with the already sampled points). At each projected sample, we randomly perturb the normal direction using an angle uniformly sampled from $[-10, 10]$ degrees, and perturb the scale of the descriptors in the range of $[0.5, 1.5]$. All generated samples are used to update the random forest. The synthesis of random samples can be viewed as sampling of the data under assumptions of prior distributions governing rotation and scale changes.

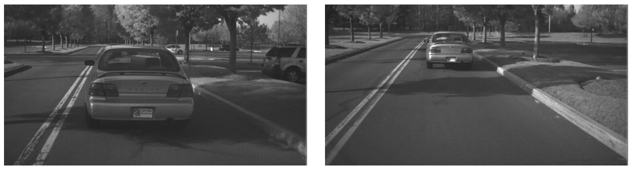
Figure 4. Some example images in the vehicle sequence. Notice the large scale change, shadows cast by trees, reflections, and constantly changing background texture.

5.3. Classifying Based on Descriptors

For each edge candidate at position $\mathbf{u}_i^{(j)}$ we associate a descriptor $\mathbf{d}_i^{(j)}$. Using the trained forest, we estimate a posteriori distribution using (13), and determine the correspondence using (7). In order to reduce false alarm, we additionally require that the maximum posteriori probability be large enough, *i.e.*, $max_i \, P(c|\mathbf{d}_i) > 0.05$, in order to report a correspondence. Otherwise, the algorithm does not report any true correspondence for $\mathbf{X}_i$.

5.4. Forgetting Old Samples

Ideally, a randomized forest should remember all samples related to an edge and recall the instance whenever the same conditions are met. Due to the capacity (the number of possible leaf nodes) of a randomized forest, this is not totally impossible but remains to be studied rigorously. However, due to the nature of our application (tracking instead of detection) we do not need to model all variations of an edge. We only need to keep enough history for reliable tracking and we can afford to forget old, irrelevant samples.

In our implementation, we fix the number of edge segment classes a leaf node can hold (in our case 10). When a new training descriptor reaches a leaf node and the edge class is not modeled by the leaf node, we erase storage of the edge class with the least number of total samples (true plus false) and replace it with that of the new edge class.

6. Experiments

6.1. Selection of Tree Depth

We choose a vehicle sequence to empirically select tree depths. The sequence is a difficult case due to a large set of boundary fragments, scale and illumination variations, see Figure 4. An edge model for the back of the vehicle contains 50 3D edge segments. We extract both positive and negative samples for each of the 50 segments using 190 frames with known poses ($\boldsymbol{\theta}$). Descriptor length is chosen as 19 and fixed for all experiments. An edge profile is extracted over the range of [-40,40] pixels. This range selection enables us to track edges with large inter-frame motions up to ± 40 pixels. On average, we collect 2800 edge profiles for each

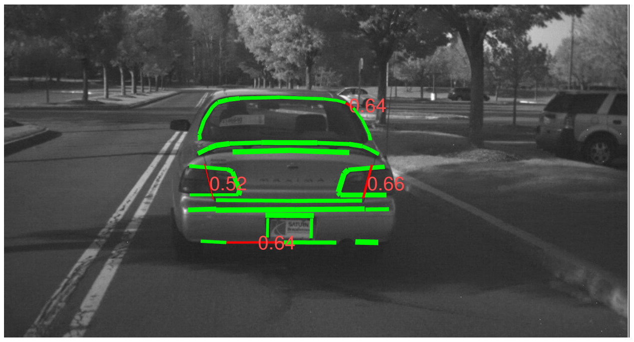

Figure 5. Classification accuracies for all 50 segments are coded by line widths. Thicker the line, better the accuracy. The worst four segments and their corresponding accuracies are drawn in red.

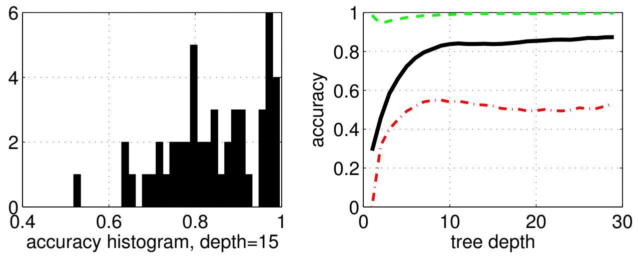

Figure 6. Accuracy of classification. Left, histogram of accuracy with tree depth 15. Right, accuracy as a function of tree depth. Black solid line: the median accuracy among all 50 edge segments. Green dash line: best accuracy. Red dash dot line: worst case.

edge segment and we extract 11 edge candidates per edge profile. Random choice will have a low chance (9%) of hitting the true correspondence.

We conduct 100 trials for each edge segment at each tree depth level. For each segment, we randomly split all edge profiles into training (two thirds) and testing (one third) data. A single randomized tree is trained for each edge segment. We then test the classification accuracy by MAP estimation (Section 5.3) using the testing data . If j^* is indeed the true correspondence, we deem the MAP estimation correct, otherwise incorrect. The ratio of correct classifications is used as an accuracy measure for the segment and the values are averaged over 100 trials. The accuracy values are visualized in Figures 5 and 6.

In Figure 5, the accuracy of the classifier for a particular edge is coded by the width of its projected line. The thicker the line, the better the accuracy. The four edge segments with the lowest accuracies are drawn in red, together with the ratios. It is not surprising to see that the easiest edges are mostly internal edges with unique patterns. Boundary edges are difficult due to varying background. However, the performance of the classifier is still quite remarkable, see Figure 6, left. The median accuracy is approximately 85%, and about 25% of the edge segments can be easily classified with accuracies above 95%. The locally robust M-estimators is sufficient to handle the false correspondences for our tracking purpose.

Figure 6, right figure shows the median, best, worst classification accuracy over all 50 segments as functions of tree depths. It is interesting to note that for some of the easy edges, very shallow randomized forests (depth 2 to 4) can achieve accuracies close to 100%. The median accuracy rises very quickly in the range of depth $[1, 10]$. After depth 10, the accuracy increases slowly. As a tradeoff between classification accuracy and storage requirement, we adopt tree depth from 10 to 15. When compared with those trees adopted in [17], our trees are fairly shallow. The reason is that our descriptors are much shorter. Our descriptors are length 19 vectors, compared to their 32×32 patches.

6.2. Benchmark Trackers

To compare the performance of our tracker, we use two baseline trackers. In the first one the edge candidate closest to the model projection is chosen as true correspondence. We call it the ICP tracker. The second tracker is a high performance edge tracker that has been developed by the authors. The tracker uses the gradient of the edge profile as a template. An edge candidate in a new frame is weighted by normalized cross correlation between template gradient profile and gradient profile surrounding the candidate edge pixel. The tracker framework is similar to that of [27] but different in optimization and correspondence. We call this tracker gradient profile correlation (GPC) tracker.

Since the main contribution of this paper is an edge candidate correspondence scheme, we use the same edge sampling and detection strategy, pose parametrization and optimization method for all three trackers. Also note that parameter settings for GPC have been chosen to maximize its performance. All three trackers deal with the same outliers and in difficult cases a large amount of clutters are unavoidable one way or another.

6.3. Tracker Performance Comparisons

We test the trackers on three objects. Some exemplary images can be found in Figure 8. We name the objects *max3*, *bmw* and *newcab* respectively.

The *max3* sequence contains 728 frames. Due to the low edge detection threshold, edge detection already gives the ICP tracker much more clutter than it can handle. The ICP tracker establishes wrong correspondences whenever there is moderate motion. It fails very quickly and never recovers. The GPC tracker performs very well on this sequence. It tracks without any problem until frame 557, where large inter-frame motion makes it very difficult for the weak GPC correspondence method to identify the true correspondence.

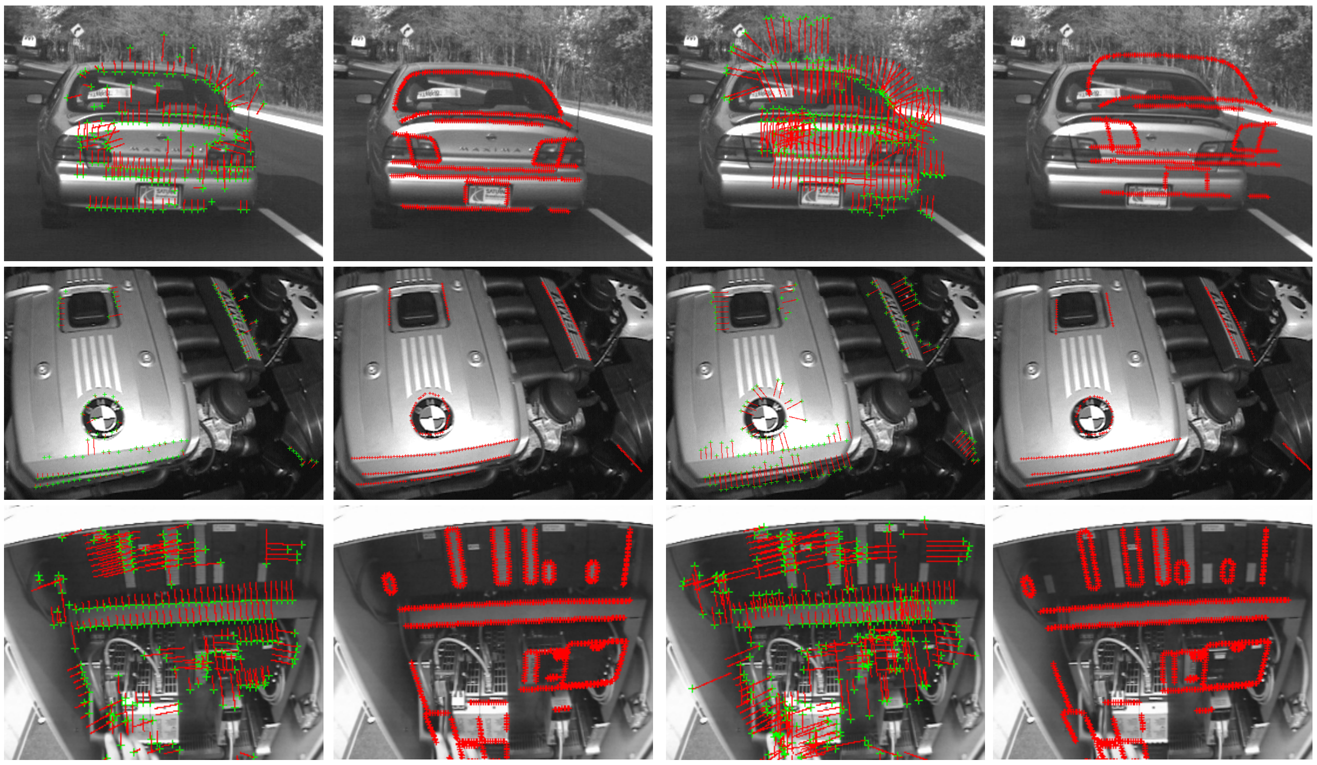

Figure 8. Tracking results for the three test sequences: *max3*, *bmw* and *newcab*. Each row shows a frame from the corresponding sequence. First column: the estimated correspondences using the proposed method. Second column: successful tracking results by our method. Third column: the correspondences established by GPC. Last column: failed frame by using the GPC tracker.

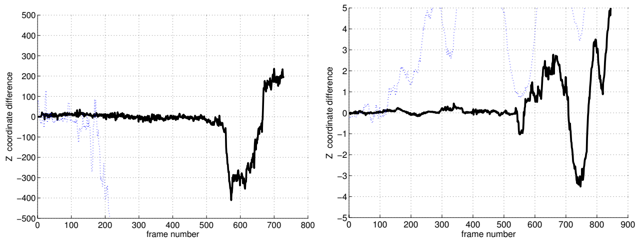

Figure 7. Tracking results comparison, shown as differences between the Z coordinates of two estimated camera centers. Left: the *max3* sequence. Right: the *bmw* sequence. Solid black line: difference between GPC and RF tracker. Dotted blue line: ICP and RF.

In contrast, the RF tracker easily establishes correspondences and tracks the whole sequence without any problem. Frame 557 of the sequence is shown in Figure 8, first row. The sequence contains complex motion trajectories that would not be quite informative if drawn in 3D. To visualize the tracking accuracies, we plot a difference value in Figure 7, left figure. The difference is the Z component of camera centers estimated by two trackers. We do not have ground-truth data for this sequence. Visual examination of the tracked results provided clear indication that the RF tracker performed the best. Thus, we use the pose estimated by the RF method as reference. From Figure 8 we see that this difference is a good indication of tracking quality.

The case for the *bmw* sequence is similar. ICP tracker loses track very quickly. The GPC tracker fails in frame 543 and never recovers, see Figure 8, second row and Figure 7, right figure.

The *newcab* sequence contains 294 frames. It is difficult because it contains very complex textures, poor camera quality, a combination of low and high contrast edges, and abrupt motions. The ICP tracker performs very poorly and we will only compare the GPC tracker with the RF tracker. Since the GPC fail quite often in this sequence while the RF tracker tracked the full sequence without a problem, we take a different approach to comparing the two trackers. After the GPC tracker fails, we re-initialize it using the correct pose computed by the RF tracker. We count the number of re-initializations that is needed in order for GPC to track the 294 frames. For the sequence shown in Figure 8, last row, we have to re-initialize the tracker 8 times. We compared the trackers on a second sequence of a similar object with a better quality camera. The second sequence contains 500 frames. The RF method tracks the whole sequence successfully, while the GPC tracker needs 4 additional re-initializations. Note that the performance of the trackers are functions of the geometric models chosen. Our choice of these models for the examples was done for optimizing GPC tracker performance in order to be fair to GPC. The comparisons clearly demonstrate the superiority of the RF method.

GPC compares less favorably to the RF tracker because it tries to capture appearance similarity from frame to frame for the same model points on an edge, *i.e.*, it spans temporally only between two consecutive frames and spatially just a single point. In contrast, RF tracker models all un-forgotten frames and spatially along the whole edge. Past variations of neighboring points are strong indications of possible appearance changes of a select point. This broad range (both in time and space) appearance modeling is important but difficult for all previous models. However, the proposed method handles it with ease.

Our current implementation of the tracker has not been optimized. It tracks at a rate of about 1 frame per second. The frame rate depends on the number of samples we use for tracking and for online training.

7. Conclusion

To make model-based edge trackers more robust, we adopt the randomized forest to learn to establish correspondences. The simple yet powerful learning method clearly outperforms many established methods. It advances the model-based edge tracker one step further by incorporating latest development in learning approaches to visual trackers. It also provides a different perspective for the online tracking problem. A geometric edge model provides very strong feedback for an online tracker and ensures that the tracker does not drift. In our future research, we plan to study the capability of the proposed descriptors in detecting object instances. The ultimate goal in this project is to extend the proposed method to instance recognition and tracking in a single framework.

References

[1] Y. Amit and D. Geman. Shape quantization and recognition with randomized trees. *Neural Computation*, 9(7):1545–1588, 1997.

[2] M. Armstrong and A. Zisserman. Robust object tracking. In *ACCV'95*, volume 1, pages 58–61.

[3] S. Avidan. Ensemble tracking. In *CVPR'05*.

[4] A. Boffy, Y. Tsin, and Y. Genc. Real-time feature matching using adaptive and spatially distributed classification trees. In *BMVC'06*.

[5] P. Bouthemy. A maximum likelihood framework for determining moving edges. *TPAMI*, 11(5):499–511, 1989.

[6] J. Canny. A computational approach to edge detection. *IEEE TPAMI*, 8(6):679–698, 1986.

[7] R. Collins, Y. Liu, and M. Leordeanu. On-line selection of discriminative tracking features. *TPAMI*, 27(10):1631–1643, 2005.

[8] A. I. Comport, E. Marchand, M. Pressigout, and F. Chaumette. Real-time markerless tracking for augmented reality: The virtual visual servoing framework. *IEEE Trans. on Visual. and Comp. Graphics*, 12:4:615–628, 2006.

[9] T. Cootes and C. Taylor. Statistical models of appearance for computer vision. Technical report, Imaging Sci. and Biomedical Eng., University of Manchester, March 2004.

[10] T. F. Cootes, C. J. Taylor, D. H. Cooper, and J. Graham. Active shape models - their training and application. *CVIU*, 61(1):38–59, January 1995.

[11] H. Dahlkamp, H.-H. Nagel, A. Ottlik, and P. Reuter. A framework for model-based tracking experiments in image sequences. *IJCV*, 73(2):139–157, 2007.

[12] T. Drummond and R. Cipolla. Real-time visual tracking of complex structures. *TPAMI*, 24(7):932–946, 2002.

[13] D. B. Gennery. Visual tracking of known three-dimensional objects. *IJCV*, 7(3):243–270, 1992.

[14] C. Harris. *Tracking with Rigid Objects*. MIT Press, 1992.

[15] A. Jepson, D. Fleet, and T. El-Maraghi. Robust online appearance models for visual tracking. *IEEE TPAMI*, 25(10):1296–1311, 2003.

[16] D. Koller, K. Daniilidis, and H.-H. Nagel. Model-based object tracking in monocular image sequences of road traffic scenes. *IJCV*, 10:257281, 1993.

[17] V. Lepetit and P. Fua. Keypoint recognition using randomized trees. *TPAMI*, 28(9):1465–1479, 2006.

[18] V. Lepetit and P. Fua. Monocular model-based 3d tracking of rigid objects. *Foundations and Trends in Computer Graphics and Vision*, 1(1):1–89, 2006.

[19] D. G. Lowe. Fitting parameterized three-dimensional models to images. *TPAMI*, 13(5):441–450, 1991.

[20] D. G. Lowe. Robust model-based motion tracking through the integration of search and estimation. *IJCV*, 8(2):113–122, 1992.

[21] L. Lu and G. Hager. A nonparametric treatment of location/segmentation based visual tracking. In *CVPR'07*.

[22] E. Marchand, P. Bouthemy, and F. Chaumette. A 2D-3D model-based approach to real-time visual tracking. *Image and Vision Computing*, 19(13):941–955, 2001.

[23] P. Meer. Robust techniques for computer vision. *Emerging Topics in Computer Vision*, pages 107–190, 2004.

[24] E. Rosten and T. Drummond. Fusing points and lines for high performance tracking. In *ICCV '05*, volume 2, pages 1508–1515. IEEE Computer Society, 2005.

[25] L. Vacchetti, V. Lepetit, and P. Fua. Combining edge and texture information for real-time accurate 3d camera tracking. In *ISMAR '04*, pages 48–57. IEEE Computer Society, 2004.

[26] H. Wang, X. Chen, and W. Gao. Online selecting discriminative tracking features using particle filter. In *CVPR'05*, pages 1037–1042.

[27] H. Wuest, F. Vial, and D. Stricker. Adaptive line tracking with multiple hypotheses for augmented reality. In *ISMAR '05*, pages 62–69, 2005.