Window Matching using Sparse Templates

Report LiTH-ISY-R-2392

Per-Erik Forssén

Computer Vision Laboratory, Department of Electrical Engineering

Linköping University, SE-581 83 Linköping, Sweden

September 14, 2001

Abstract

This report describes a novel window matching technique. We perform window matching by transforming image data into sparse features, and apply a computationally efficient matching technique in the sparse feature space. The gain in execution time for the matching is roughly 10 times compared to full window matching techniques such as SSD, but the total execution time for the matching also involves an edge filtering step. Since the edge responses may be used for matching of several regions, the proposed matching technique is increasingly advantageous when the number of regions to keep track of increases, and when the size of the search window increases.

The technique is used in a real-time *ego-motion* estimation system in the WITAS project. Ego-motion is estimated by tracking of a set of *structure points*, i.e. regions that do not have the aperture problem. Comparisons with SSD, with regard to speed and accuracy are made.

1 Introduction

Window matching is a method to estimate local displacements between pairs of images. A local image patch is compared with patches at various displacements in the following frame, and the best match is used as an estimate of the displacement. Sub-pixel accuracy of the estimate can be achieved by combining several matches near the best match. See [6] for an overview of window matching techniques.

Window matching implicitly assumes that two consecutive frames differ only by a translation. However, camera images are in general 2D-projections of a 3D scene, and thus the images differ by a perspective change. Factors such as occlusions, and changes in lighting can also cause the frames to not fit the plain translation model. For these reasons, any matching algorithm is bound to fail in certain situations, and thus a good *measure of certainty* is an essential component of any window matching algorithm.

Some representation of the regions to keep track of are stored as *templates*, and these templates are used as long as our measure of certainty (usually the degree of match, or some function thereof) is high.

Not all image regions are well suited to window matching, most important is that the region to track cannot be modelled as a one dimensional signal. For locally one dimensional regions, the matching will suffer from the *aperture problem*, i.e. the estimated displacement will equal the projection of the true displacement onto the subspace of signal variation [8].

Applications of window matching usually keep track of several image regions simultaneously. Applications range from tracking of all moving objects in an otherwise stationary scene, to predictive video coding [14]. The intended application of the window matcher described in this report is estimation of *ego-motion* within the WITAS[1] project [9]. Ego-motion is the apparent motion of the camera as inferred from movement of local regions in a stationary scene, see for instance [11], or [5] for the present application.

Window matching is often performed directly on intensity images, by methods such as SSD[2] (Sum of Squared Difference of the pixels in both regions). We will instead make a transformation of the image into sparse data, and then apply a less computationally intensive matching technique in the sparse feature space.

2 Sparse coding

Sparse coding is a way to transform data in such a way that patterns are easy to detect. Typically inputs and outputs are represented in a *channel representation* [7]. That is, signals are limited, and *monopolar* (e.g. always positive). For non-zero values, the magnitude signifies the relevance, and thus a zero response means "no information".

David Field [3] discusses sparse coding of natural images in depth, and contrasts it to compact coding techniques such as PCA. The goal of compact coding is to minimise the number of output nodes, by concentrating the signal entropy in a small number of nodes. Sparse coding, on the other hand tries to concentrate the information content in the *active* nodes. The total number of nodes is allowed to remain the same as in the input, or even to increase.

There are several proposed optimisation schemes to find sparse transforms for a given set of input data, for instance [4, 15, 1]. The mammalian retina and primary visual cortex also seem to make use of a similar optimisation technique [3].

The result of sparse coding is a representation of the input as a combination of a few commonly occurring sub-patterns or *independent components*. For natural images, these sub-patterns consist of line and edge segments in a number of scales [15, 1], and this is our motivation for choosing edge detection as a first filtering of our input. Other kinds of sparse data, such as the divergence and rotational symmetries computed in [12] could of course also be used by the template matcher.

[1] WITAS project home page: `http://www.ida.liu.se/ext/witas/`

[2] An equivalent term is MSD (Mean Squared Difference). Another similar method is called MAD (Mean Absolute Difference).

3 Sparse template matching

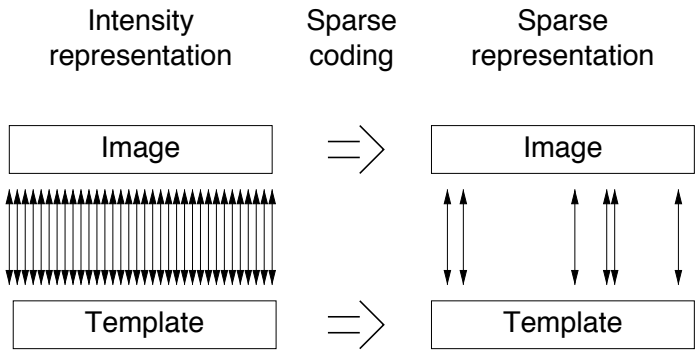


Figure 1: Sparse template matching.

The key idea of *sparse template matching* (STM) is illustrated in figure 1. In the intensity representation we have to verify the correspondence between image and template in all template positions, but at the sparse feature representation we only have to compare those features that are active in the template.

Note that the matching has been separated into two stages, *sparse coding*, and *feature matching*. If we want to match several regions, the result from the sparse coding stage can be reused. This means that STM is increasingly advantageous when the number of regions to keep track of increases, and when the size of the search window increases.

Unless the sparse representation is complete, the quality of STM is of course critically dependent on how well the sparse representation is able to capture image content that is important for the matching.

4 Edge filtering

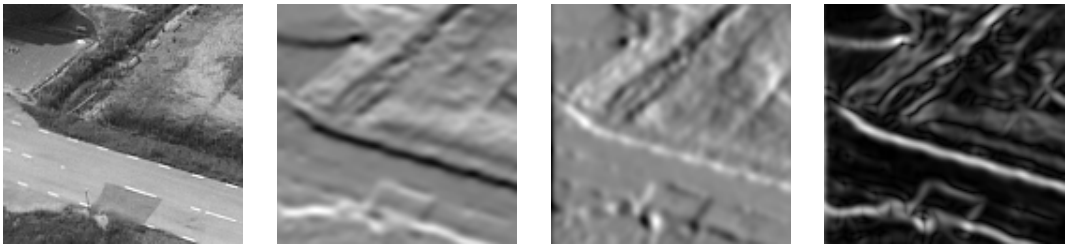
The sparse features we will use are edge filter responses. Edge filtering can be made computationally efficient if we use separable differentiating Gaussian filters. A differentiating Gaussian filter in the x-direction can be separated into a Gaussian low-pass filter[3] $\mathrm{g_x}$ and a small differentiating kernel $\mathrm{d_x} = [\ -1 \quad 0 \quad 1\ ]$. To improve the robustness of the result we also want to low-pass filter the result with a Gaussian $\mathrm{g_y}$ in the y-direction. If we make the two low-pass filters $\mathrm{g_x}$ and $\mathrm{g_y}$ the same size, we can implement edge filtering in the x and y directions as a separable Gaussian low-pass filtering followed by a small differentiating convolution for each direction:

$$\begin{aligned} \mathrm{e}_0(\boldsymbol{x}) &= (\mathrm{s} * \mathrm{g_x} * \mathrm{g_y})(\boldsymbol{x}) \\ \mathrm{e}_1(\boldsymbol{x}) &= (\mathrm{e}_0 * \mathrm{d_x})(\boldsymbol{x}) \\ \mathrm{e}_2(\boldsymbol{x}) &= (\mathrm{e}_0 * \mathrm{d_y})(\boldsymbol{x}) \end{aligned} \tag{1}$$

[3] A Gaussian low-pass filter is a truncated and discretised representation of the function $\mathrm{g_x}(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{x^2}{2\sigma^2}}$.

The standard deviation parameter, σ, of the Gaussian filters is used to adjust the scale of these edge filters. Throughout this report $\sigma = 2.0$ has been used. The resultant Gaussian filter has 15 real coefficients, and thus our edge filtering involves a total of $15 + 15 + 2 + 2 = 34$ coefficients.

The sum of the magnitudes of the edge responses, $\mathrm{e}(\boldsymbol{x}) = \mathrm{abs}(\mathrm{e}_1(\boldsymbol{x})) + \mathrm{abs}(\mathrm{e}_2(\boldsymbol{x}))$, is plotted in figure 2. As we can see from this plot, only a small fraction of the image positions will have large magnitudes. This means that if matching using e_1 and e_2 is performed as product sums, a small fraction of the template coefficients will have a dominating influence on the result. Thus, a *pruning*, or removal of low-magnitude coefficients in the templates will have little impact on the result.



$\mathrm{s}(\boldsymbol{x})$ $\mathrm{e}_1(\boldsymbol{x})$ $\mathrm{e}_2(\boldsymbol{x})$ $\mathrm{e}(\boldsymbol{x})$

Figure 2: Edge responses and the sum of their magnitudes.
e_1 *and* e_2 *have a colour map which displays positive values as bright, and negative values as dark.*

5 Template construction

Edge-image based matching is nothing new, but it has traditionally involved an additional distance map computation [6]. The purpose of the distance map computation is to obtain *metric*—a measure of how close we are to an edge in each image position. By summing the distances to edges in all positions indicated as edges in the template, we can obtain a measure that increases smoothly as we move away from the location of the best match.

However, an edge image computed at low frequency already has a *local* distance metric: The highest values are always found at the edge location, while the response magnitude slowly decreases as we move away from the edge. Thus, we can avoid the distance map computation by using low frequency edge responses instead.

Another novelty in our approach is that we will not perform the matching on the compact feature map $\mathrm{e}(\boldsymbol{x})$, but instead on both edge responses separately. When combining $\mathrm{e}_1(\boldsymbol{x})$ with $\mathrm{e}_2(\boldsymbol{x})$ to form $\mathrm{e}(\boldsymbol{x})$ we remove information that could aid the matching, by ignoring the direction of edges.

So instead of producing a compact description of edge structure, we make an expansion of the dimensionality. What actually makes our approach faster in the end is the fact that we can prune our templates of most of their coefficients.

5.1 Interest-point selection

To illustrate the sparse template construction, we start from a region in the scene that has reasonably large intensity variations, and does not have the aperture problem (see figure 3).

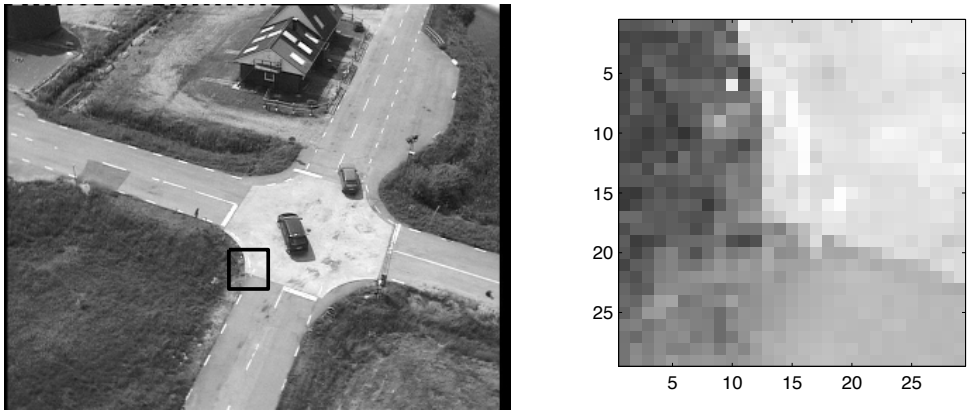


Figure 3: A region without the aperture problem.

Such a region can be found by first constructing a tensor image $\boldsymbol{T}(\boldsymbol{x})$:

$$\boldsymbol{T}(\boldsymbol{x}) = \begin{pmatrix} \mathrm{e}_1(\boldsymbol{x}) \\ \mathrm{e}_2(\boldsymbol{x}) \end{pmatrix} \cdot \begin{pmatrix} \mathrm{e}_1(\boldsymbol{x}) & \mathrm{e}_2(\boldsymbol{x}) \end{pmatrix}$$

We then average the tensor components in local image regions (preferably using separable Gaussian filters) and select points with two large eigenvalues [13]. For eigenvalues $\lambda_1 \geq \lambda_2$, such points can easily be found by looking at local peaks in a λ_2 image. This is equivalent to the Harris corner detector [10].

5.2 Pruning

The positive and negative parts of the edge filter responses for the region shown in figure 3 are shown in the top row of figure 4. As mentioned earlier, removal of low magnitude coefficients in these templates will have small impact on the matching result. Another way to prune the templates is to perform a directed *non-max-suppression* in each of the templates (an idea inspired by [2]). That is, we first loop over the vertical templates, and set all positions that are smaller than either the neighbour above or below, to zero. The same procedure is performed on the horizontal templates, but here the left and right neighbours are checked.[4]

The directed non-max-suppression operation can be seen as a crude approximation of the lateral inhibition mechanisms found in biological vision systems.

To control the sparseness of the templates, the remaining non-zero positions are sorted according to magnitude, and those belonging to the largest percentile are kept. In this

[4]Note that non-max-suppression destroys the local metric in the templates only. There still are continuous variations in the edge responses we match the templates with, and this is what gives us continuous responses.

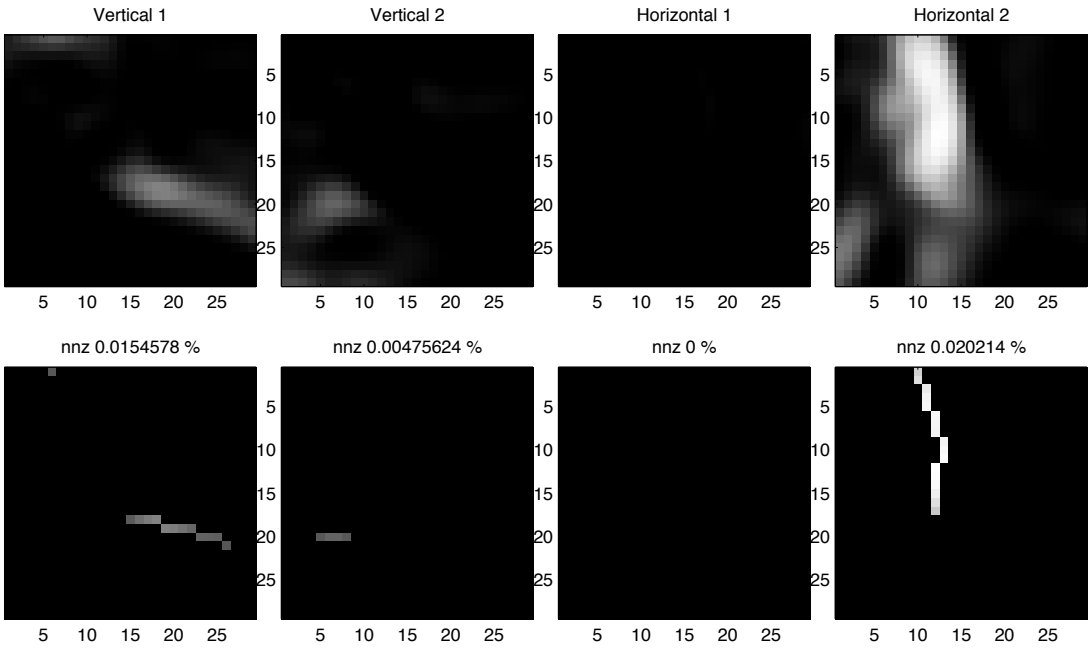


Figure 4: Pruning of a template.

process the vertical and horizontal templates are sorted separately. This makes it easy to ensure that after the pruning we have the same number of coefficients in both templates and thus good accuracy in both horizontal and vertical directions.

The bottom row of figure 4 shows the positive and negative parts of the templates after directed non-max-suppression and removal of all but the largest 2% of coefficients. As can be seen, the operation will produce templates in which the spatial extent of the ridges are kept, even for very sparse templates.

This degree of sparsity works fine for smaller templates as well (see section 8), and reduces the total number of coefficients by a factor 25 compared to full template matching techniques such as SSD. The total execution time is of course also dependent on the speed of the edge filtering. However, as mentioned in section 3, the execution time is no longer as sensitive to the size of the search window as in SSD. Nor is the number of windows to match as important, and this fact is especially important for estimation of ego-motion, where more displacement estimates increase the robustness.

6 Matching procedure

From e_1 and e_2 we extract the template regions t_1 and t_2. To formulate the matching procedure we also add a conceptual index n, that loops over the template coefficients left after the pruning. We can now write the product-sum matching as:

$$\mathrm{m_v} = \max(0, \sum_{n=1}^{N_v} \mathrm{t_1}(\boldsymbol{x}_n)\, \mathrm{e_1}(\boldsymbol{x}_0 + \boldsymbol{x}_n)) \tag{2}$$

$$\mathrm{m_h} = \max(0, \sum_{n=1}^{N_h} \mathrm{t_2}(\boldsymbol{x}_n)\, \mathrm{e_2}(\boldsymbol{x}_0 + \boldsymbol{x}_n)) \tag{3}$$

The compound match is computed as:

$$\mathrm{match} = \mathrm{m_v} \cdot \mathrm{m_h} \tag{4}$$

This compound match requires a high degree of match for features in both vertical and horizontal direction. As we can see, the max operations in equations 2 and 3 are needed, since large mismatches in both horizontal and vertical directions would otherwise result in a high total match.

The result of this matching technique with the templates in the bottom row of figure 4 is shown in figure 5. The leftmost image in the figure shows the frame in which the matching has been performed (not the same as the one from which the template was extracted), and the centre and right images show the $\mathrm{m_v}$ and $\mathrm{m_h}$ responses respectively.

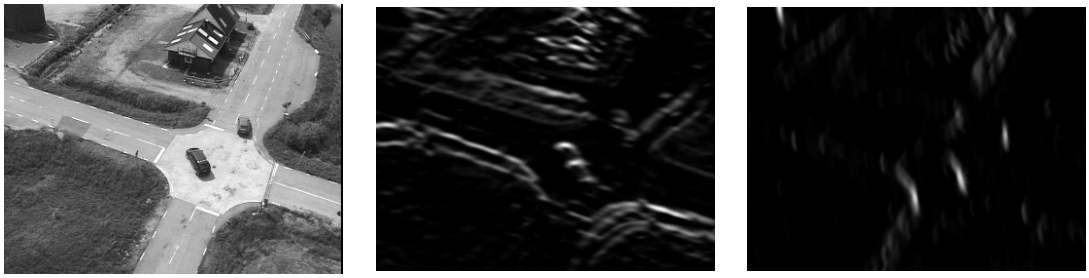


Figure 5: Matches in vertical and horizontal directions.

The compound matching result is shown in figure 6 together with the response from a full edge-image[5] product sum, and SSD. The SSD result has a colour-map with low values shown as bright in order to aid comparison. Details of this figure are shown in figure 7. This will hopefully illustrate that sparse template matching (STM) produces results that are at least as useful as SSD. Both methods produce results that gradually increase near the peak–a sign of graceful degradation. The smoothness of the peak in figure 7 is controlled by the σ parameter of the initial low-pass filters $\mathrm{g_x}$ and $\mathrm{g_y}$. The figures in this report all use $\sigma = 2.0$.

In a region tracking situation, we would like to keep our templates as long as possible, since each time we select a new template we introduce an error. If we do not use sub-pixel resolution coordinates, this error is in the range $[-0.5, 0.5]$ pixel in each dimension. Graceful degradation is very useful here, since we can use the degree of match as an

[5]The edge image used here is the sum of the magnitudes of the edge responses defined in equation 1.

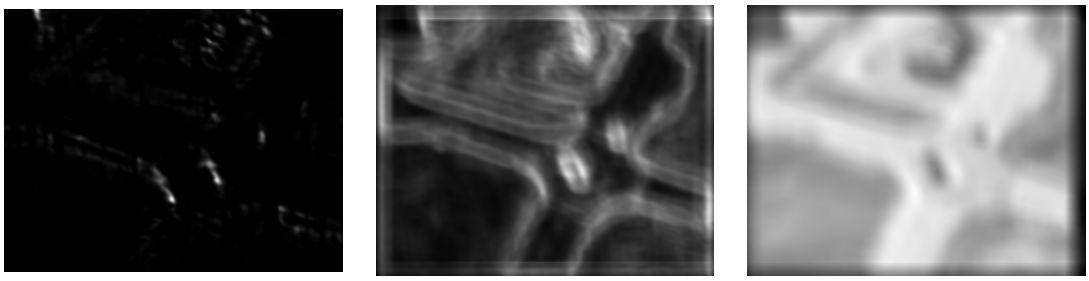


Figure 6: STM response compared with edge-image product sum, and SSD.

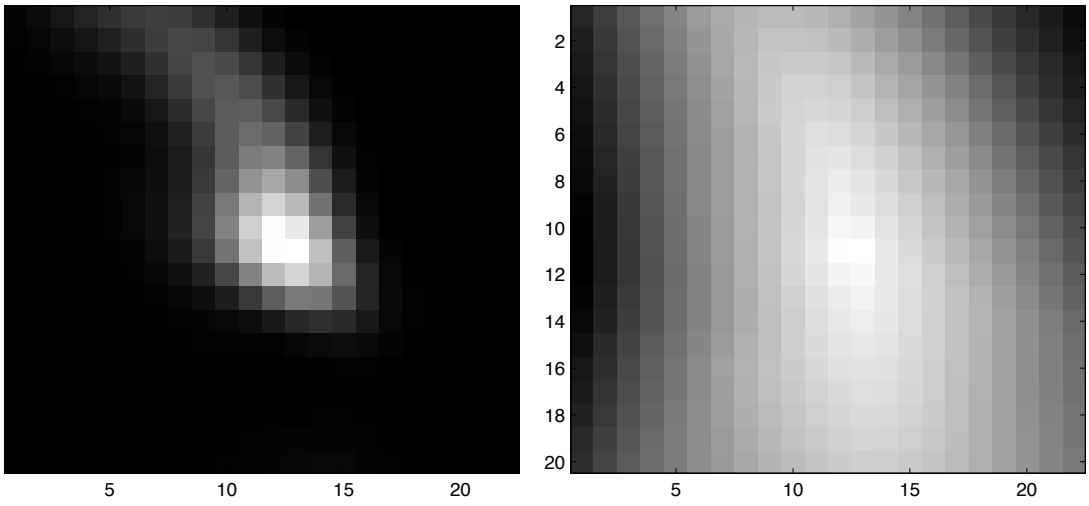


Figure 7: Detail of figure 6. STM and SSD responses.

indication of when to select a new template. In the current implementation new templates are selected when the degree of match deviates too much from the expected value, i.e.:

$$\text{Keep template if} \quad \left| \frac{\text{match}(\mathrm{T}, \mathrm{I})}{\text{match}(\mathrm{T}, \mathrm{T})} - 1 \right| < \epsilon \tag{5}$$

Where $\epsilon = 0.1$ or similar. Since $\text{match}(\mathrm{T}, \mathrm{T})$ is constant for each template, we could simply divide all template coefficients with this value beforehand to simplify the comparison.

The normalisation described here is an approximation of a *normalised product sum*. Ideally we should divide the match by the coefficient sums of the template and of the image region, but since the image content varies, this is computationally expensive. Since the matching is not fully normalised, an upper threshold on the match is needed in order to remove false matches in high contrast regions.

The gradual increase near the peak in the response image implies that we could use the response image to find the displacement with sub-pixel resolution [13, 6], for instance by fitting a quadratic function to the response image samples.

7 Sigmoid-like function

A common problem with product sums on edge images is that very sharp edges will get very high responses, and will thus tend to dominate the result completely. This problem can be dealt with by using normalised product sums, but these have the disadvantage of increasing the computational load. It is also dealt with to some extent by the directed non-max-suppression treatment of the templates (see section 5.2), since this will tend to make the shape of the edge ridges more important than their actual values.

An approach that is less computationally demanding than the normalised product sums is to apply a sigmoid-like function on the edge responses before they are used. An example of such a function that has been found to be satisfactory is:

$$\begin{aligned} \mathrm{n}(\boldsymbol{x}) &= \mathrm{e}_1(\boldsymbol{x})^2 + \mathrm{e}_2(\boldsymbol{x})^2 \\ \mathrm{f}_1(\boldsymbol{x}) &= \mathrm{sign}(\mathrm{e}_1(\boldsymbol{x}))\, \mathrm{e}_1(\boldsymbol{x})^2/(500 + \mathrm{n}(\boldsymbol{x})) \\ \mathrm{f}_2(\boldsymbol{x}) &= \mathrm{sign}(\mathrm{e}_2(\boldsymbol{x}))\, \mathrm{e}_2(\boldsymbol{x})^2/(500 + \mathrm{n}(\boldsymbol{x})) \end{aligned}$$

The parameter 500 in the denominator applies to images in the range $[0 \ldots 255]$, and Gaussian filters g_x and g_y that sum to 1.

The main advantage with this function is that it preserves the symmetrical shape of the matching response near the peaks (see figure 7), contrary to most non-linear functions operating on e_1 and e_2 separately.

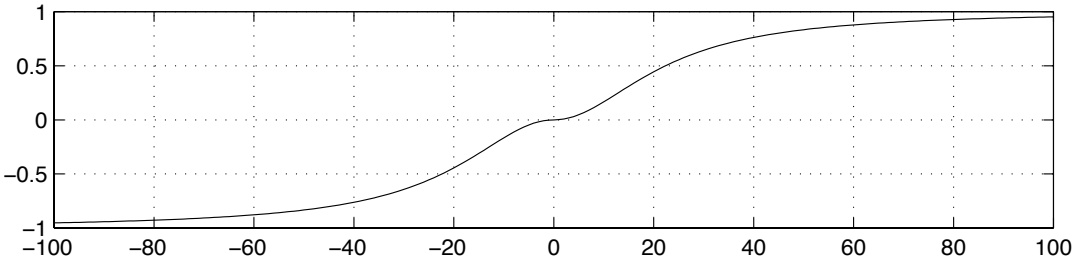


Figure 8: Sigmoid-like function ($\mathrm{f}_1(\mathrm{e}_1)$ for $\mathrm{e}_2 = 0$).

The sigmoid-like function can be seen as a soft threshold of the edge data, i.e. once the edge response is above a certain threshold it does not matter much how much above the threshold it is (see figure 8). Since the variations in edge image magnitude will become smaller when a sigmoid is applied, it might be advantageous to lower the decision threshold for changing templates (see equation 5).

In the current implementation the sigmoid computation time is approximately 20% of the time needed to compute the edge responses.

8 Performance

The performance of a stand-alone window matching algorithm is difficult to assess for a number of reasons.

- Firstly we know a priori that only *structure points* i.e. regions without the aperture problem are well suited to tracking. A fair evaluation should thus evaluate the joint system of point-selection and window matching.

- Secondly, the implicit assumption of pure translation is not valid in general (see section 1). This implies that the performance evaluation should also evaluate the measure of certainty that is used to detect when the match is no longer valid.

Since these requirements make a fair evaluation very difficult, we will settle for a test of the pure translation case, but we will incorporate the structure point selection described in section 5, and use the measure of certainty defined in equation 5.

In the evaluation, three different constant scenes of 360×288 pixels, were subjected to sub-pixel translations in both horizontal and vertical directions. The translations spanned the range $[0, 1]$ with steps of 0.1 pixel, giving a total of 121 translations per image. The sub-pixel translations were computed with *bicubic interpolation* using the `INTERP2` routine in MATLAB.

8.1 Accuracy

In the first experiment we investigate the accuracy of the matching under varying amount of pruning.

In this experiment we use 13×13 templates, and the maximum shift checked is 7 pixels in each direction. A total of $3 \times 24 = 72$ structure points were selected, but only those which had valid matching values were kept. This typically resulted in $1 - 5$ points being discarded in each frame for non-zero-coefficient ratios below 0.1. Figure 9 shows average absolute errors with and without the sigmoid function, and with and without sub-pixel estimations. The sub-pixel estimation used fits a quadratic function to a 3×3 neighbourhood around the peak. As can be seen, the curves obtained without the sigmoid are more noisy. This is due to spurious false matches that passed equation 5 undetected. As a comparison, the average absolute errors for SSD are 0.325 without sub-pixel estimation, and 0.143 with.

It is interesting to note that the average error in STM actually drops when coefficients are removed. This is probably due to reduced influence of low-template-value-high-image-value type matches. However, the uncertainty in detection also increases, which is reflected in the increased errors for extremely sparse templates.

The next experiment tests non-max-suppression as an alternative and/or a complement to the sigmoid. As can be seen in figure 10, non-max-suppression appears to do more harm than good to the accuracy. But note that this is just the accuracy, practical experience on other than plain translation cases seem to indicate that non-max-suppression reduces the chance of false matches. We can also see that the optimal pruning level appears to be around 2-3%.

8.2 Speed

The next experiment compares STM with 2% pruning level with SSD in terms of speed. In this experiment 15×15 templates are used, and the maximum shift checked is 12 pixels

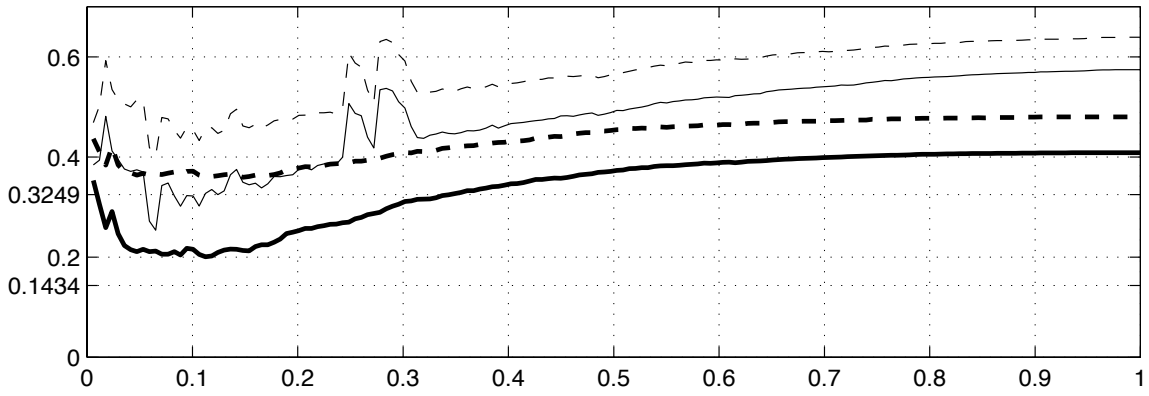


Figure 9: Test of sigmoid. Absolute errors against fraction of non-zero coefficients.
Thin: Without sigmoid. Thick: With sigmoid.
Solid: With sub-pixel estimation. Dashed: Without sub-pixel estimation.

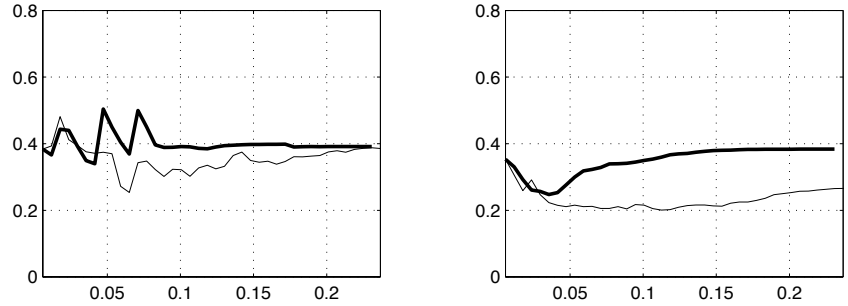


Figure 10: Test of non-max-suppression.
Left: Without sigmoid. Right: With sigmoid.
Thick: With non-max-suppression. Thin: Without.

in each direction. The timings are computed as averages over 121 executions.

The sparse template matching (STM) system can be divided into three parts as shown in figure 11. The timings of the first two boxes depend only on the image size. In the reference implementation, which is written in JAVA and executed on a 300 MHz Sun Ultra 60, they average at 323 ms for the edge filtering, and 59 ms for the sigmoid computation.

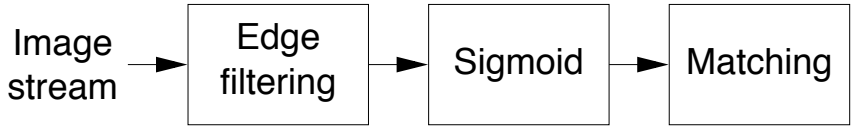


Figure 11: STM system parts.

In the experiment we test matching with the number of templates at 10, 18, 28, 37, 45, and 52. The left part of figure 12 shows the absolute timings of STM and SSD. Since edge filtering is an integral component of most vision systems, the STM timing without the edge filtering step is also shown, and for completeness also the timing of the matching

stage alone. The right part of figure 12 shows the SSD timing divided by the three variants of STM timings.

As can be seen from these plots, the actual matching in STM is approximately 10 times faster than SSD. The break-even point for STM in this setup is at 17 templates, but for smaller templates, and smaller search windows, a larger number of templates will be needed for break-even. The maximum recorded speedup for STM is a factor 2.7 at 52 templates, but if the edge filtering is excluded this rises to a factor 7.2.

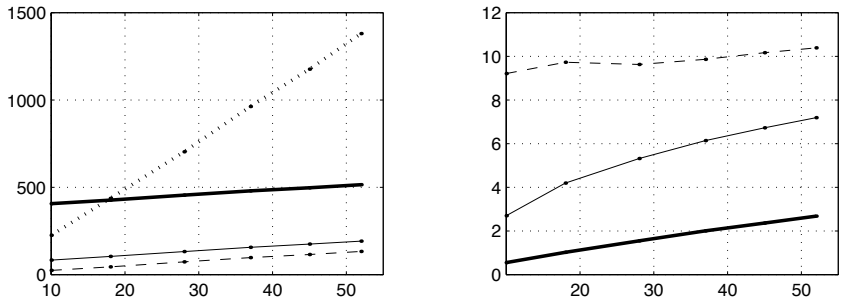


Figure 12: Test of speed.
Left: Absolute timing. Right: Speedup factors.
Dotted: SSD timing.
Thick, thin, and dashed: STM, STM without edge filtering, STM matching stage only.

9 Conclusions

In this report we have shown that STM has an accuracy that is in the same range as SSD (STM with sub-pixel accuracy is better than SSD without), while being significantly faster. If the total system already uses an edge filtering stage for other purposes, the additional cost of the matching stage is much smaller in the STM case (the effective speedup is a factor 3 to 7 depending on the application).

Acknowledgements

The work presented in this report was supported by WITAS, the Wallenberg laboratory on Information Technology and Autonomous Systems, which is gratefully acknowledged.

References

[1] A. J. Bell and T. J. Sejnowski. Edges are the 'independent components' of natural scenes. *Advances in Neural Information Processing Systems*, 9, 1996.

[2] J. Canny. A computational approach to edge detection. PAMI-8(6):255–274, November 1986.

[3] D. J. Field. What is the goal of sensory coding? *Neural Computation*, 1994.

[4] F. Földiák. Forming sparse representations by local anti-hebbian learning. *Biological Cybernetics*, 1990.

[5] Per-Erik Forssén. Updating Camera Location and Heading Using a Sparse Displacement Field. Technical Report LiTH-ISY-R-2318, Dept. EE, Linköping University, SE-581 83 Linköping, Sweden, November 2000.

[6] A. Giachetti. Matching techniques to compute image motion. *IVC*, 18(3):247–260, February 2000.

[7] G. H. Granlund. An Associative Perception-Action Structure Using a Localized Space Variant Information Representation. In *Proceedings of Algebraic Frames for the Perception-Action Cycle (AFPAC)*, Kiel, Germany, September 2000.

[8] G. H. Granlund and H. Knutsson. *Signal Processing for Computer Vision*. Kluwer Academic Publishers, 1995. ISBN 0-7923-9530-1.

[9] Gösta Granlund, Klas Nordberg, Johan Wiklund, Patrick Doherty, Erik Skarman, and Erik Sandewall. WITAS: An Intelligent Autonomous Aircraft Using Active Vision. In *Proceedings of the UAV 2000 International Technical Conference and Exhibition*, Paris, France, June 2000. Euro UVS.

[10] C.G. Harris and M. Stephens. A combined corner and edge detector. In *4th Alvey Vision Conference*, pages 147–151, September 1988.

[11] M. Irani, B. Rousso, and S. Peleg. Recovery of ego-motion using region alignment. *IEEE Trans. on PAMI*, pages 268–272, March 1997.

[12] Björn Johansson and Gösta Granlund. Fast Selective Detection of Rotational Symmetries using Normalized Inhibition. In *Proceedings of the 6th European Conference on Computer Vision*, volume I, pages 871–887, Dublin, Ireland, June 2000.

[13] Anders Moe. Passive Aircraft Altitude Estimation using Computer Vision. Lic. Thesis LiU-Tek-Lic-2000:43, Dept. EE, Linköping University, SE-581 83 Linköping, Sweden, September 2000. Thesis No. 847, ISBN 91-7219-827-3.

[14] K. N. Ngan, T. Meier, and D. Chai. *Advanced Video Coding: Principles and Techniques*. Elsevier Science B.V., 1999.

[15] B. A. Olshausen and D. J. Field. Emergence of simple cell receptive field properties by learning a sparse code for natural images. *Nature*, 381:607–609, 1996.