A Framework Towards Realtime Detection and Tracking of Text

Carlos Merino
Departamento de Fisiología
Universidad de La Laguna
38071 Santa Cruz de Tenerife, Spain
cmerino@ull.es

Majid Mirmehdi
Department of Computer Science
University of Bristol
Bristol BS8 1UB, England
majid@cs.bris.ac.uk

Abstract

We present a near realtime text tracking system capable of detecting and tracking text on outdoor shop signs or indoor notices, at rates of up to 15 frames per second (for generous 640×480 *images), depending on scene complexity. The method is based on extracting text regions using a novel tree-based connected component filtering approach, combined with the Eigen-Transform texture descriptor. The method can efficiently handle dark and light text on light and dark backgrounds. Particle filter tracking is then used to follow the text, including SIFT matching to maintain region identity in the face of multiple regions of interest, fast displacements, and erratic motions.*

1. Introduction

Tracking text is an important step towards the identification and recognition of text for outdoor and indoor wearable or handheld camera applications. In such scenarios, as the text is tracked, it can be sent to OCR or to a text-to-speech engine for recognition and transition into digital form. This is beneficial in many application areas, such as an aid to the visually impaired or for language translation for tourists. Furthermore, the ability to automatically detect and track text in realtime is of use in localisation and mapping for human and robot navigation and guidance.

A review [9] and some collections of recent works [2, 1] in camera-based document analysis and recognition, highlight substantial progress in both single image and multiframe based text analysis. Overall, there have been relatively few works on general text tracking. Multiframe text analysis has been mainly concerned with improving the text in a super-resolution sense [12] or for continuous recognition of text within a stationary scene e.g. on whiteboards or in slideshows [18, 20].

A directly related work in the area of general scene text tracking is by Myers and Burns [13] which successfully tracks scene text undergoing scale changes and 3D motion. However, this work applies to tracking in batch form and is not a realtime solution. Also in [13], the text detection is done by hand, manually indicating a starting bounding box for the tracking system to follow. Another work of interest is Li *et al.*[8] in which a translational motion tracking model was presented for caption text, based on correlation of image blocks and contour based stabilisation to refine the matched position. Less directly related, in [16], seven simple specific text strings were looked for by a roving camera from a collection of 55 images in an application to read door signs.

The focus of this paper is on the development of a resilient text tracking framework, using a handheld or wearable camera, as a precursor for our future work on text recognition. The only assumption we make is that we are looking for larger text sizes on shop and street signs, or indoor office boards or desktop documents, or other similar surfaces. Our proposed method is composed of two main stages: candidate text region detection and text region tracking. In the first stage, regions of text are located using a connected components approach combined with a *texture* measure step [17] which to the best of our knowledge has never been applied to text detection; this provides candidate regions or components which are then grouped to form possible words. The method is highly accurate but not infallible to noise, however, noisy or non-text candidate regions are not detected as persistently as true text regions, and can be rejected forthright during the tracking step. In the second stage, particle filtering is applied to track the text frame by frame. Each hypothesised system state is represented by a particle. The particles are weighted to represent the degree of belief on the particle representing the actual state. This non-linear filtering approach allows very robust tracking in the face of camera instability and even vigorous shakes. SIFT matching is used to identify regions from one frame to the next. We describe the details of this framework in the next few sections.

2. Background

It should be noted that there is a significant body of work on detecting (graphical) text that has been superimposed in images and videos, as well as in tracking such text. Example works are [10, 8]. In this work we concentrate solely on text embedded in natural scenes.

Segmentation of text regions involves the detection of text and then its extraction given the viewpoint. For example, almost each one of the works published in [2, 1] present one method or another for text segmentation, usually from a fronto-parallel point of view. Example past works considering other viewpoints and recovering the projective views of the text are [4, 14, 13]. Although in this work we engage in both *segmenting and tracking* text involving varying viewpoints, actual fronto-parallel recovery is not attempted. This is a natural step possible from the tracking motion information available and will be a key focus of our future work.

An issue of note is the problem of scale. Myers and Burns [13] dealt with this by computing homographies of planar regions that contain text, and when computationally tractable, this could be useful for any (realtime) text tracking application. Here, we are detecting text dynamically, hence at some smaller scales our detector will simply not find it, until upon approach it becomes large enough.

3. Methodology

The text tracking framework proposed here is based around the principle of a *tracker* representing a *text entity* - a word or group of words that appear together in an image as a salient feature, where each word comprises two or more components or regions. Trackers are dynamically created when a new text entity is detected; they follow the text frame to frame, and they get removed when the text cannot be detected anymore. Partial occlusion is dealt with, and in cases of full occlusion, a new tracker starts once the text is back in view. Our text tracking framework involves text segmentation, text region grouping, and tracking, including dynamic creation and removal of trackers.

3.1. Text segmentation

The text segmentation stage uses a combination of a connected components (CC) approach and a region filtering stage, with the latter involving the novel application to text analysis of a *texture* measure. The resulting component regions are then grouped into text entities.

3.1.1 Connected component labelling Following CC labelling in [7], León *et al* employed a tree pruning approach to detect text regions. They thresholded the image at every grey level, and built a Max-tree representation where each node stored the CC of the corresponding threshold level.

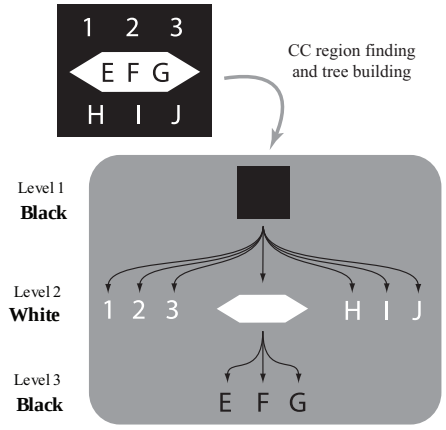

Figure 1. A synthetic sample image and its corresponding tree of connected regions.

The leaves of the tree represented the zones whose grey levels were the highest in the image. For detection of dark text over bright backgrounds, they built a different tree, a Min-tree, where the leaves represented the zones with the lowest grey levels in the image. This two pass treatment of bright text and dark text is very common in text detection algorithms.

We improve on the tree region labelling method in [7] by introducing a simple representation that allows efficient, one pass detection of bright text (white over black) and dark text (black over white) in the same tree. Initially, simple local adaptive thresholding is applied to the source frame. We empirically fixed the local threshold window size to 17×17 throughout all our experiments. The threshold was the mean grey level value of the window itself. Connected component region labelling is then performed on the thresholded image. This labelling builds a tree of connected regions, with the outermost region the root of the tree and the innermost regions the leaves. We allow the regions to toggle their label value from black to white as we go down each level of the tree. The tree represents the nesting relationship between these regions. Each node of the tree keeps only the *contour* around the border of the regions (see Figure 1).

Once the tree is built, it is walked depth-first with the objective to filter out the regions that are not text. Each node of the tree is classified as text or non-text during the walk using region filtering as described later below.

Usually, on real-world images with scene text, structural elements (such as sign borders, posters frames, etc.) can exhibit characteristics of text, such as high contrast against their backgrounds or strong texture response. These elements can be easily discarded (as long as they are not at a leaf) using the nesting relationship present in the proposed tree. When a node has children already classified as text,

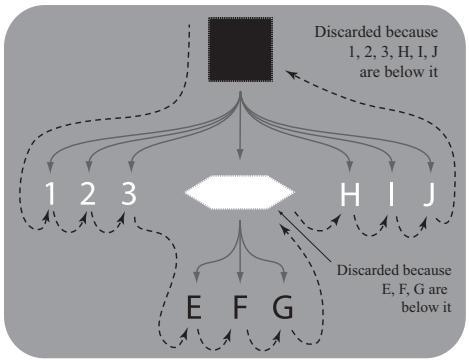

Figure 2. Parent nodes are discarded when children are classified as text.

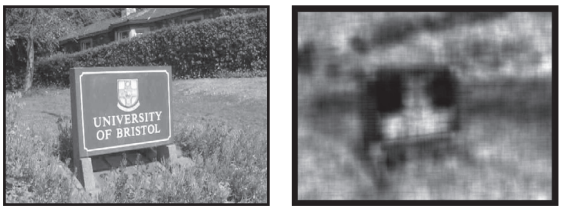

Figure 3. Original image and its Eigen-Transform response.

it is discarded as non-text, despite the text classifying functions may having marked it as text. This discards most of the non-text structural elements of the text (Figure 2).

3.1.2 Region filtering To classify text regions we apply three tests in cascade, meaning that if a test discards a region as non-text, no more tests are applied to it. This is in a similar fashion to Zhu *et al.* [21] who used 12 classifiers. In our case, the fewer tests are important for real time processing, and coarse, but computationally more efficient tests are applied first, quickly discarding obvious non-text regions, and slower, more discriminative tests are applied last, where the number of remaining regions is fewer. The test we apply are on *size*, *border energy*, and an eigenvector based *texture measure*.

Size - Regions too big or too small are discarded. The thresholds here are set to the very extreme. Very small regions are discarded to avoid noise. This may still drop characters, but they probably would be otherwise impossible to recognise by OCR and as the user gets closer, they are more likely to be picked up anyway. Large regions are discarded because it is unlikely that a single character occupies very large areas (over half the size) of the image.

Border energy - A Sobel edge operator is applied to all the points along the contour of each component region r and the mean value is obtained:

$$B_r = \frac{\sum_{i=1}^{P_r} \sqrt{(G_{ix}^2 + G_{iy}^2)}}{P_r} \qquad (1)$$

where P_r denotes the number of border pixels in region r, and G_x and G_y represent the Sobel gradient magnitudes. This is referred to as the *border energy* and provides a measurement of region to background contrast. Regions with border energy value below a very conservatively fixed threshold are discarded. This removes regions that usually appear in less textured and more homogeneous regions.

Jiang *et al* [6] used a three level Niblack threshold [19] in their text detection technique with good results. This introduces the local pixel values variance into the threshold calculation. However, this involves computing the standard deviation of local pixel values and we have found that doing a simpler adaptive threshold and afterwards discarding the noisy regions is faster. Also, the proposed tree walking algorithm transparently manages bright-text and dark-text occurrences on the same image without the need to apply a three level threshold image.

Texture measure - For this final decision-making step we apply a texture filter whose response at positions within the region pixels and their neighbourhoods is of interest.

We have previously combined several texture measures to determine candidate text regions, see [3]. These were mainly tuned for small scale groupings of text in the form of paragraphs. Although quite robust, the need for faster processing precludes their combined use. Here, we introduce the use of the Eigen-Transform texture operator [17] for use in text detection. It is a descriptor which gives an indication of surface roughness. For a square $w \times w$ matrix representing a pixel and its neighbouring grey values, the eigenvalues of this matrix are computed: $\|\lambda_1\| \geq \|\lambda_2\| \geq \ldots \|\lambda_w\|$. The largest l eigenvalues are discarded since they encode the lower frequency information of the texture. Then, the Eigen-Transform of the central pixel is the mean value of the $w - l + 1$ smaller magnitude eigenvalues:

$$\Gamma(l, w) = \frac{1}{w - l + 1} \sum_{k=l}^{w} \|\lambda_k\| \qquad (2)$$

The Eigen-Transform has a very good response to texture which exhibit high frequency changes, and we found in our experiments that it responds to text very well for this reason, see a simple example in Figure 3 where both the text and the background texture are picked up well. It can, however, be a fairly slow operator, but fortunately we need only apply it to the component region pixels. Indeed, we compute the Eigen-Transform only on some regularly sampled points inside the bounding box of each region of interest. A key factor in (2) is the size of w. This is determined automatically by setting it dynamically according to the height

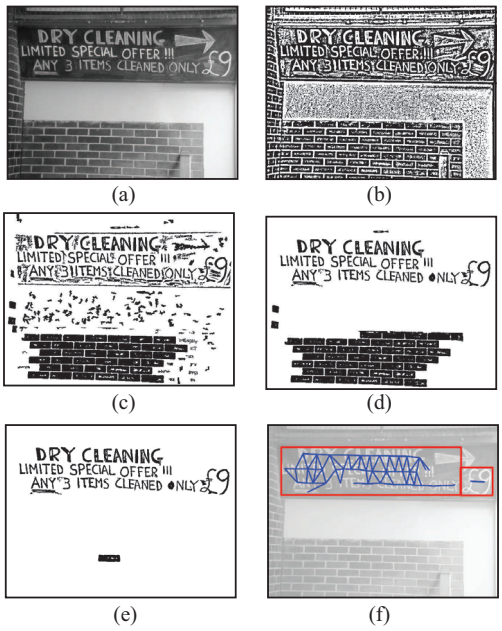

Figure 4. Steps of the text segmentation and grouping. (a) Original image, (b) Adaptive threshold, (c)–(e) result after filtering by size, border energy and Eigen-transform measure, (f) perceptual grouping.

of the region under consideration. Then l is set to be a fraction of w.

The result of the text segmentation stage is a set of candidate regions with a high likelihood of being text. For each region, the centre position of its bounding box is stored as a component $\mathbf{c}_i$ into the *observation function* $\mathbf{y}_k$ of the particle filter (see section 3.2). As a result of the CC region tree design, and taking into account only the contour and not the contents, both inverted text (light on dark) and normal text (dark on light) are detected in the same depth-first pass. Figure 4 shows an example result highlighting each of the text segmentation and grouping steps.

3.1.3 Perceptual text grouping - The text grouping stage takes the regions produced by the text segmentation step and makes compact groups of perceptually close or *salient* regions. We follow the work by Pilu [14] on perceptual organization of text lines for deskewing. Briefly, Pilu defines two scale-invariant saliency measures between two candidate text regions A and B: *Relative Minimum Distance* λ and *Blob Dimension Ratio* γ:

$$\lambda(A,B) = \frac{D_{\min}}{A_{\min} + B_{\min}} \qquad \gamma(A,B) = \frac{A_{\min} + A_{\max}}{B_{\min} + B_{\max}} \quad (3)$$

where $D_{\min}$ is the minimum distance between the two regions, and $A_{\min}$, $B_{\min}$, $A_{\max}$ and $B_{\max}$ are respectively the minimum and maximum axes of the regions A and B. Pilu's text saliency operator between two text regions is then:

$$\mathbf{P}(A,B) = N(\lambda(A,B),1,2) \cdot N(\gamma(A,B),0,4) \quad (4)$$

where $N(x,\mu,\sigma)$ is a Gaussian distribution with mean μ and standard deviation σ whose parameters were determined experimentally in [14]. To reduce the complexity of comparing all the regions against each other, we construct a planar graph using Delaunay triangulation, with the region centres as vertices. The saliency operator is then applied to each edge of this graph, keeping only the salient ones and removing the rest. This edge pruning on the graph effectively divides the original graph into a set of connected subgraphs. Each subgraph with more than two vertices is considered a text group. This additional filtering step removes a number of isolated regions (see Figure 4(f)).

3.2. Text tracking

Particle filtering, also known as the Sequential Monte Carlo Method, is a non-linear filtering technique that recursively estimates a system's state based on the available observation. In an optimal Bayesian context, this means estimating the *likelihood* of a system's state given the observation $p(\mathbf{x}_k|\mathbf{y}_k)$, where $\mathbf{x}_k$ is the system's state at frame k and $\mathbf{y}_k = \{\mathbf{c}_1, \ldots, \mathbf{c}_K\}$ is the observation function.

Each hypothesised new system state at frame k is represented by a particle resulting in $\{\mathbf{x}_k^{(1)}, \mathbf{x}_k^{(2)}, \ldots, \mathbf{x}_k^{(N)}\}$, where N is the number of particles. Each particle $\mathbf{x}_k^{(n)}$ has an associated weight $\left\{(\mathbf{x}_k^{(1)}, w_k^{(1)}), \ldots, (\mathbf{x}_k^{(N)}, w_k^{(N)})\right\}$ where $\sum_{i=1}^{s} w_k^{(i)} = 1$. Given the particle hypothesis $\mathbf{x}_k^{(n)}$, the weights are proportional to the likelihood of the observation, $p(\mathbf{y}_k|\mathbf{x}_k^{(n)})$. For a detailed explanation of particle filter algorithms and applications, see e.g. [5].

Particle filtering is the ideal method given the instability of the handheld or wearable camera in our application domain. We build on the particle tracking framework developed in [15] for simultaneous localisation and mapping (SLAM). Here we want to independently track multiple instances of text in the image, with a simple state representation. Thus, each text entity is assigned a particle filter, i.e. a *tracker*, responsible of keeping its state. The main components to now deal with in a particle filter implementation are the *state representation*, the *dynamics model* and the *observation model*.

3.2.1 State representation - The *tracker* represents the evolution over time of a text entity. It has a state that tries to model the apparent possible changes that the text entity may experience in the image context. The model has to be rich enough to approximate the possible transformations of the text but at the same time simple enough to be possible to estimate it in real time.

The state of a tracker at frame k is represented by a 2D translation and rotation: $\mathbf{x}_k = (t_x, t_y, \alpha)$. We found this simple state model provides sufficient accuracy given the degree of movement within consecutive frames, but is also important in computational savings towards a real-time model[1]. This state defines a relative coordinate space, where the x-axis is rotated by an angle α with respect to the image, and its origin is at (t_x, t_y) in image coordinates.

Let's say a text entity contains M components. Its tracker preserves a list of M features $\mathbf{Z} = \{\mathbf{z}_1, \ldots, \mathbf{z}_M\}$ where each feature $\mathbf{z}_i$ is a 2D position lying in the tracker's relative coordinate space. Each feature represents a text component being tracked, and it is fixed during tracker initialization. We define the transformation function $\Psi(\mathbf{z}_i, \mathbf{x}_k)$ as the coordinate transform (translation and rotation) of a feature position from the state's coordinate space to image coordinates. This is used during weighting. Additionally, each feature is associated with a set of SIFT descriptors [11] computed only once during the tracker initialization. They give the ability to differentiate between candidate text components, providing a degree of multiscale and rotation invariance to the feature matching as well as resilience to noise and change in lighting conditions[2].

Figure 5 shows the current state representation $\mathbf{x}_k$ of a tracker at frame k which has $M = 4$ features $\mathbf{Z} = \{\mathbf{z}_1, \mathbf{z}_2, \mathbf{z}_3, \mathbf{z}_4\}$. For ease of exposition, all the features are visualised to lie along the x-axis of the tracker's coordinate space. Further, the figure shows another particle $\mathbf{x}_k^{(1)}$ representing an alternative state hypothesis. The four features $\mathbf{z}_i \in \mathbf{Z}$ are mapped to the particle's relative coordinate space to show the same set of features from a different reference frame. The observation function $\mathbf{y}_k$, with $\mathbf{y}_k = \{\mathbf{c}_1, \mathbf{c}_2, \mathbf{c}_3, \mathbf{c}_4\}$ representing the center points of the candidate text components is also shown.

3.2.2 Dynamics model - The dynamics model defines the likelihood of a system state transition between time steps as $p(\mathbf{x}_k|\mathbf{x}_{k-1})$. It is composed of a deterministic part - a prediction of how the system will evolve in time, and a stochastic part - the random sampling of the particles around the predicted position. Examples of prediction schemes are constant position, constant velocity and constant acceleration. Examples of stochastic functions are uniform and Gaussian random walks around an uncertainty window of the predicted position.

The selection of an appropriate dynamics model is crucial for a tracking system to be able to survive unpredictable movements, such as those caused by wearable or hand-held camera movements. Pupilli [15] concluded that for such scenarios a constant position prediction model with a uniform or Gaussian random walk would provide better results, due to the unpredictable nature of erratic movements. Here, we follow this advice to use a constant position model with random Gaussian walk around the last state, i.e. $p(\mathbf{x}_k|\mathbf{x}_{k-1}) = N(\mathbf{x}_{k-1}, \Sigma)$. The covariance matrix Σ defines the *particle spread* which is empirically set to a generous size, and automatically reduced via an annealling process as in [15].

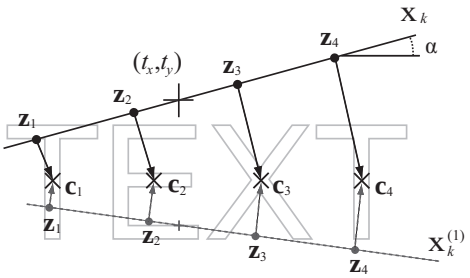

Figure 5. State model of one *tracker*, $\mathbf{x}_k = (t_x, t_y, \alpha)$, with 4 tracked features $\mathbf{Z} = \{\mathbf{z}_1, \mathbf{z}_2, \mathbf{z}_3, \mathbf{z}_4\}$. A particle, $\mathbf{x}_k^{(1)}$, shows a different state hypothesis.

3.2.3 Observation model - Given a particle state hypothesis, the observation model defines the likelihood of the observation, $p(\mathbf{y}_k|\mathbf{x}_k^{(n)})$. The weight of each particle is calculated based on the comparison from projected features' positions and actual text components found in the image. An inlier/outlier likelihood proposed by Pupilli [15] is used.

For each tracked feature $\mathbf{z}_i \in \mathbf{Z}$, a set of candidate components $\mathbf{y}_{ki} \subseteq \mathbf{y}_k$ $\{(\mathbf{z}_1, \mathbf{y}_{k1}), (\mathbf{z}_2, \mathbf{y}_{k2}), \ldots, (\mathbf{z}_M, \mathbf{y}_{kM})\}$ is computed, based on their matching to the SIFT descriptors previously stored for each feature. This reduces the search space of the particles and gives robustness to the tracking process.

The weight of a particle is proportional to the number of observed candidate components inside a circular region of radius ε around each tracked feature. First an *inlier threshold* function $\tau(\mathbf{a}, \mathbf{b})$ is defined:

$$\tau(\mathbf{a}, \mathbf{b}) = \begin{cases} 1 & \text{if } d(\mathbf{a}, \mathbf{b}) < \varepsilon \\ 0 & \text{otherwise} \end{cases} \quad (5)$$

where $d(\mathbf{a}, \mathbf{b})$ is the distance between two points. Then, the likelihood is:

$$p(\mathbf{y}_k|\mathbf{x}_k^{(n)}) \propto \exp\left(\sum_{\mathbf{z}_i \in \mathbf{Z}} \sum_{\mathbf{c}_j \in \mathbf{y}_{ki}} \tau\left(\Psi(\mathbf{z}_i, \mathbf{x}_k^{(n)}), \mathbf{c}_j\right)\right) \quad (6)$$

[1] However, we intend to investigate more complex motion models in future while ensuring the realtime aspects of the work are not compromised

[2] Note to Reviewers: We have found the SIFT matching to grossly slow our system down. By the time of this Workshop we will have implemented and hope to report faster invariant feature matching using e.g. the Hessian Affine or MSER which will additionally give a greater degree of affine invariancy

where $\Psi(\mathbf{z}_i, \mathbf{x}_k^{(n)})$ is the transformation function defined in subsection 3.2.1. Figure 6 shows the weighting process of one feature $\mathbf{z}_2$ for two different hypothesis, $\mathbf{x}_k^{(1)}$ and $\mathbf{x}_k^{(2)}$. The latter is nearer to the actual state of the system and gets a greater weight. Note that for illustration purposes we are considering here that the candidate group components for feature $\mathbf{z}_2$ is all the observation: $\mathbf{y}_{k2} = \mathbf{y}_k = \{\mathbf{c}_1, \mathbf{c}_2, \mathbf{c}_3, \mathbf{c}_4\}$.

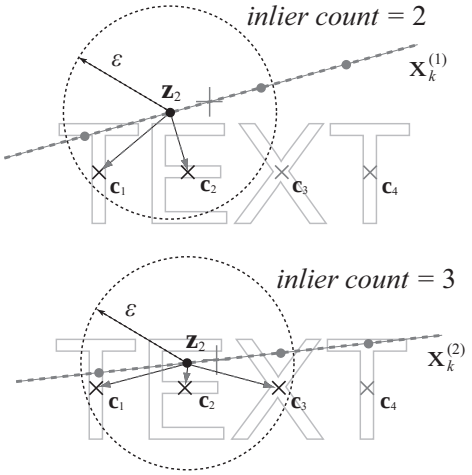

Figure 6. Inlier count of feature $\mathbf{z}_2$ for two different particles $\mathbf{x}_k^{(1)}$ and $\mathbf{x}_k^{(2)}$.

3.2.4 Bounding box computation - Bounding box computation is crucial towards next possible stages such as extraction, recognition or superresolution. Thus, it is important that it is as stable and tight as possible. Once a posterior state is established by the particle filter, each feature $\mathbf{z}_i \in \mathbf{Z}$ is assigned a *Most Likely Estimate* (MLE), that is the text component $\mathbf{c}_j \in \mathbf{y}_k$ that most likely matches it. In Figure 5, the MLE of each feature is marked with an arrow. Not all tracked features will have a MLE each frame, as sometimes they are not found due to blur, clutter or occlusion.

After perceptual text grouping, each observed text component belongs to a group, and thus the MLE of each tracker feature also belongs to a group. The *Most Likely Group* (MLG) of a feature is the group to which this feature's MLE belongs to. Given this, the tracker's bounding box is then obtained by joining the bounding boxes of its MLGs.

3.2.5 Tracker creation and removal - Trackers are dynamically created when new text is detected, and removed when their associated text entity can no longer be found. After the grouping stage, any text group detected is a potential text entity to be tracked. But some of these groups may belong to text entities already being tracked. The tracking stage identifies the tracked components in the image via the MLE and MLG mechanisms. After the tracking cycle, any unidentified remaining groups are passed to a new tracker. Newly created trackers must continuously track their text for a number of frames to be considered *stable*. Trackers that fail to comply with this are promptly removed. The tracker removal mechanism is very simple. After a consecutive number of frames without a match, the track is considered lost and removed. Should the same text entity then reappear, it will be assigned a new tracker.

4. Results

The system was tested on a variety of typical outdoor and indoor scenarios, e.g. a hand-held camera while passing shops or approaching notices, posters, billboards etc. We present here the results from four typical scenarios. The full video sequences along with other results, including a sequence from [13], are also available online[3].

The results shown are: Figure 7: 'BORDERS' - walking in a busy street with several shop signs overhead, Figure 8: 'UOB' - walking past a signboard including an occlusion in a highly textured scene background, Figure 9'ST. MICHAEL'S HOSPITAL' - a traffic sign with both bright and dark text, complex background and significant perspective change, and Figure 10: 'LORRY' - with text also undergoing viewpoint changes. All sequences were at 640×480 resolution recorded at 15 fps with a consumer grade photo/video camera (Nikon Coolpix P2).

Table 1 shows the performance of the algorithm for the different sample scenes on an Intel Pentium IV 3.2Ghz processor. The results show the performance of the text segmentation and grouping subsystem alone, and the whole tracking process. Text segmentation is very fast. When measured off-line, the system was able to compute the results faster than the actual frame rate of the sequences. With the tracking, the performance of the system is *close to 10 fps on average*, depending on the complexity of the scene, making it promisingly close to realtime. For a simple scene with little background and one 5-character word, the *system could track it effortlessly at 15fps*. While the particle filtering framework is relatively fast, the SIFT matching of features reduces the performance when the number of candidate regions is large, such as in very complex backgrounds, e.g. in Fig. 8. A greater number of false positives (due to the vegetation) produced during segmentation put more stress on the tracking stage, which however rejected these regions due to the instability and lack of longevity of their trackers. Notice also in Fig. 8, the tracker survives the occlusion by the lamppost.

4.1. Discussion

The focus of this paper has been on a framework to track text as robustly and continuously as possible, bearing in

[3] Please see `http://vision.cs.bris.ac.uk/texttrack/`

Figure 7. Example scene 1 - BORDERS - notice several BORDERS signs come along in the sequence.

Figure 8. Example scene 2 - UOB including occlusion, also with much other texture.

Figure 9. Example scene 3 - ST. MICHAEL'S HOSPITAL - two regions, dark over light and vice versa.

Figure 10. Example scene 4 - LORRY

Table 1. Performance of the algorithm in mean frames per second.

	Text segmentation	Full algorithm
Scene 1	31.9 fps	13.2 fps
Scene 2	21.3 fps	4.9 fps
Scene 3	30.7 fps	9.6 fps
Scene 4	32.0 fps	10.6 fps

mind that momentary loss of a text region is not disastrous in terms of recognition. Once stable tracking is obtained after a few frames, the motion information could be used for fronto-parallel recovery as well as generation of a super-resolution representation for better OCR, e.g. as in [12]. In our system, it is more likely that text is missed if it is at sharp perspective viewpoints, than for a non-text region to be tracked with significant stability. We had no such non-text cases, but even if there were, one can assume that OCR would reject it at the next stage.

Some shortcomings of our work are: (1) the robustness of our tracker improves further, in terms of dropping a track only to be picked up again instantly, when we use a more complex motion model, but this means we move further away from a realtime goal, (2) SIFT has limited robustness to viewpoint variations, so big changes of point of view will make the trackers lose the features, and it is by far the slowest part of the system, however we are at the time of writing experimenting with a new method, (3) Our results can not be claimed to be fully realtime, however we are near enough and believe we can achieve it in our future short-term work, (4) even though our few thresholds are fixed they naturally can affect the quality of the results; we aim to address these by applying learning techniques to automate the process where necessary.

5. Conclusion

In this paper we have presented a close to realtime technique to automatically detect and track text in arbitrary natural scenes. To detect the text regions, a depth-first search is applied to a tree representation of the image's connected components, where each leaf in the tree is examined for three criteria. Amongst these criteria is the use of the Eigen-Transform texture measure as an indicator of text. This stage of the algorithm detects both bright and dark text in a single traversal of the tree. Following perceptual grouping of the regions into text entities, particle filtering is applied to track them across sequences involving severe motions and shakes of the camera. We have established a significant framework and can start to improve its individual components in our future work to better our results.

Acknowledgements

Carlos Merino is funded by FIT-350300-2006-92 project from the Spanish Ministerio de Industria, Turismo y Comercio, through the European Regional Development Fund. This work was carried out partly at Bristol University and partly at Universidad de La Laguna and the Instituto Tecnológico de Canarias.

References

[1] *Proc. of the 1st Workshop on Camera Based Document Analysis and Recognition (CBDAR)*, August 2005.
[2] Special issue on camera-based text and document recognition. *International Journal on Document Analysis and Recognition*, 7(2–3), July 2005.
[3] P. Clark and M. Mirmehdi. Recognising text in real scenes. *IJDAR*, 4:243–257, 2002.
[4] P. Clark and M. Mirmehdi. Rectifying perspective views of text in 3d scenes using vanishing points. *Pattern Recognition*, 36(11):2673–2686, 2003.
[5] A. Doucet, J. de Freitas, and N. Gordon. *Sequential Monte Carlo Methods in Practice*. Springer-Verlag, 2001.
[6] R.-J. Jiang, F.-H. Qi, L. Xu, G.-R. Wu, and K.-H. Zhu. A learning-based method to detect and segment text from scene images. *Journal of Zhejiang University*, 8(4):568–574, 2007.
[7] M. León, S. Mallo, and A. Gasull. A tree structured-based caption text detection approach. In *Fifth IASTED VIIP*, 2005.
[8] H. Li, D. Doermann, and O. Kia. Automatic text detection and tracking in digital video. *IEEE-IP*, 9(1):147–156, 2000.
[9] J. Liang, D. Doermann, and H. Li. Camera-based analysis of text and documents: a survey. *IJDAR*, 7(2):84–104, 2005.
[10] R. Lienhart. Indexing & retrieval of digital video sequences based on text recognition. In *ICM*, pages 419–420, 1996.
[11] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *IJCV*, 60(2):91–110, 2004.
[12] C. Mancas-Thillou and M. Mirmehdi. Super-resolution text using the teager filter. In *CBDAR05*, pages 10–16, 2005.
[13] G. K. Myers and B. Burns. A robust method for tracking scene text in video imagery. In *CBDAR05*, 2005.
[14] M. Pilu. Extraction of illusory linear clues in perspectively skewed documents. In *CVPR*, pages 363–368, 2001.
[15] M. Pupilli. *Particle Filtering for Real-time Camera Localisation*. PhD thesis, University of Bristol, October 2006.
[16] H. Shiratori, H. Goto, and H. Kobayashi. An efficient text capture method for moving robots using dct feature and text tracking. In *ICPR*, pages 1050–1053, 2006.
[17] A. Targhi, E. Hayman, J. Eklundh, and M. Shahshahani. The eigen-transform & applications. In *ACCV*, pages I:70–79, 2006.
[18] M. Wienecke, G. A. Fink, and G. Sagerer. Toward automatic video-based whiteboard reading. *IJDAR*, 7(2):188–200, 2005.
[19] L. L. Winger, J. A. Robinson, and M. E. Jernigan. Low-complexity character extraction in low-contrast scene images. *IJPRAI*, 14(2):113–135, 2000.
[20] A. Zandifar, R. Duraiswami, and L. S. Davis. A video-based framework for the analysis of presentations/posters. *IJDAR*, 7(2):178–187, 2005.
[21] Z. Zhu, F. Qi, M. Kimachi, and Y. Wu. Using adaboost to detect & segment characters in natural scenes. In *CBDAR05*, 2005.