Text Recognition and 2D/3D Object Tracking

Este exemplar corresponde à redação final da Tese devidamente defendida e corrigida por Rodrigo Minetto e aprovada pela Banca Examinadora.

This text is the final version of the Thesis defended and properly corrected by Rodrigo Minetto and approved by the Thesis Commitee.

Campinas, 19 de março de 2012.

Jorge Stolfi
Orientador/*Advisor*
UNICAMP

Neucimar J. Leite
Co-orientador/*Co-advisor*
UNICAMP

Tese apresentada ao Instituto de Computação, UNICAMP, e ao Laboratoire d'informatique de Paris 6, UPMC, como requisito parcial para a obtenção do título de Doutor em Ciência da Computação.

This thesis is submitted to the Institute of Computing, UNICAMP, *and to the Laboratoire d'informatique de Paris 6,* UPMC, *in partial fulfilment of the requirements for the degree of Ph.D. in Computer Science.*

FICHA CATALOGRÁFICA ELABORADA POR
MARIA FABIANA BEZERRA MULLER - CRB8/6162
BIBLIOTECA DO INSTITUTO DE MATEMÁTICA, ESTATÍSTICA E
COMPUTAÇÃO CIENTÍFICA - UNICAMP

Minetto, Rodrigo, 1983-
M662r Reconhecimento de texto e rastreamento de objetos 2D/3D / Rodrigo Minetto. – Campinas, SP : [s.n.], 2012.

Orientador: Jorge Stolfi.
Coorientador: Neucimar Jerônimo Leite.
Tese (doutorado) – Universidade Estadual de Campinas, Instituto de Computação.

1. Reconhecimento de texto. 2. Detecção de texto. 3. Descritor de imagem. 4. Rastreamento de texto. 5. Rastreamento tridimensional. I. Stolfi, Jorge, 1950-. II. Leite, Neucimar Jerônimo, 1961-. III. Universidade Estadual de Campinas. Instituto de Computação. IV. Título.

Informações para Biblioteca Digital

Título em inglês: Text recognition and 2D/3D object tracking
Palavras-chave em inglês:
Text recognition
Text detection
Image descriptor
Text tracking
Three-dimensional tracking
Área de concentração: Ciência da Computação
Titulação: Doutor em Ciência da Computação
Banca examinadora:
Jorge Stolfi [Orientador]
Matthieu Cord
Arnaldo de Albuquerque Araújo
Patrick Pérez
Marcin Detyniecki
Data de defesa: 19-03-2012
Programa de Pós-Graduação: Ciência da Computação

TERMO DE APROVAÇÃO

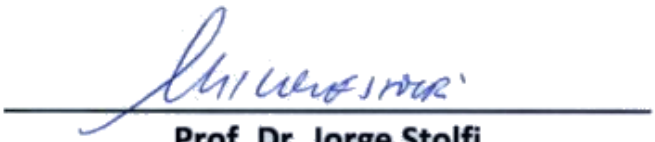
Tese Defendida e Aprovada em 19 de março de 2012, pela Banca examinadora composta pelos Professores Doutores:

Prof. Dr. Marcin Detyniecki
UPMC - FR

Prof. Dr. Patrick Pérez
TRI - FR

Prof. Dr. Arnaldo de Albuquerque Araújo
DCC- UFMG

Prof. Dr. Matthieu Cord
UPMC - FR



Prof. Dr. Jorge Stolfi
IC - UNICAMP

Instituto de Computação
Universidade Estadual de Campinas

Laboratoire d'informatique de Paris 6

Université Pierre et Marie CURIE

Text Recognition and 2D/3D Object Tracking

Rodrigo Minetto[1]

March of 2012

Banca Examinadora (*Thesis Commitee*):

- Jorge Stolfi (Orientador)
- Matthieu Cord
  Universidade Pierre et Marie Curie (UPMC)
- Arnaldo de Albuquerque Araújo
  Universidade Federal de Minas Gerais (UFMG)
- Patrick Pérez
  Pesquisador Sênior na Technicolor Research Innovation
- Marcin Detyniecki
  Universidade Pierre et Marie Curie (UPMC)

[1]Suporte financeiro de: FAPESP (07/54201-6), CNPq (142205/2007-9) e CAPES-COFECUB (592/08). This project was supported by FAPESP (PhD grant 07/54201-6), CNPq (PhD grant 142205/2007-9) and the project CAPES/COFECUB 592/08.

THESE DE DOCTORAT DE
L'UNIVERSITE PIERRE ET MARIE CURIE
ET
UNIVERSIDADE ESTADUAL DE CAMPINAS

Spécialité:
Informatique

présentée par

Rodrigo MINETTO

pour obtenir le grade de
DOCTEUR DE L'UNIVERSITE PIERRE ET MARIE CURIE
et
DOUTOR PELA UNIVERSIDADE ESTADUAL DE CAMPINAS

Reconnaissance de Zones de Texte et Suivi d'Objects dans les Images et les Vidéos

Text Recognition and 2D/3D Object Tracking

soutenue publiquement le 19 mars 2012
devant le jury composé de:

Nicolas THOME	Maitre de Conférences à l'Université Pierre et Marie Curie	*encadrant*
Neucimar J. LEITE	Professeur à l'Université Estadual de Campinas	*encadrant*
Matthieu CORD	Professeur à l'Université Pierre et Marie Curie	*directeur de thèse*
Jorge STOLFI	Professeur à l'Université Estadual de Campinas	*directeur de thèse*
Hélio PEDRINI	Professeur à l'Université Estadual de Campinas	*examinateur*
Marcin DETYNIECKI	CR CNRS à l'Université Pierre et Marie Curie	*examinateur*
Patrick PÉREZ	Senior Researcher à Technicolor Research Innovation	*rapporteur*
Arnaldo de A. ARAÚJO	Professeur à l'Université Feredal de Minas Gerais	*rapporteur*

Resumo

Nesta tese abordamos três problemas de visão computacional: (1) detecção e reconhecimento de objetos de texto planos em imagens de cenas reais; (2) rastreamento destes objetos de texto em vídeos digitais; e (3) o rastreamento de um objeto tridimensional rígido arbitrário com marcas conhecidas em um vídeo digital. Nós desenvolvemos, para cada um dos problemas, algoritmos inovadores, que são pelo menos tão precisos e robustos quanto outros algoritmos estado-da-arte. Especificamente, para reconhecimento de texto nós desenvolvemos (e validamos extensivamente) um novo descritor de imagem baseado em HOG especializado para escrita romana, que denominamos T-HOG, e mostramos sua contribuição como um filtro em um detector de texto (SNOOPERTEXT). Nós também melhoramos o algoritmo SNOOPERTEXT através do uso da técnica multiescala para tratar caracteres de tamanhos bastante variados e limitar a sensibilidade do algoritmo a vários artefatos. Para rastreamento de texto, nós descrevemos quatro estratégias básicas para combinar a detecção e o rastreamento de texto, e desenvolvemos também um rastreador específico baseado em filtro de partículas que explora o uso do reconhecedor T-HOG. Para o rastreamento de objetos rígidos, nós desenvolvemos um novo algoritmo preciso e robusto (AFFTRACK) que combina rastreamento de características por KLT com uma calibração de câmera melhorada. Nós testamos extensivamente nossos algoritmos com diversas bases de dados descritas na literatura. Nós também desenvolvemos algumas bases de dados (publicamente disponíveis) para a validação de algoritmos de detecção e rastreamento de texto e de rastreamento de objetos rígidos em vídeos.

Résumé

Dans cette thèse, nous abordons trois problèmes de vision par ordinateur: (1) la détection et la reconnaissance d'objets de texte dans des images de scènes réelles; (2) le suivi de ces objets de texte dans une vidéo numérique, et (3) le suivi d'objets 3D rigides et arbitraires avec des amers connus dans une vidéo numérique. Pour chaque problème, nous avons développé des algorithmes innovants, qui sont au moins aussi précis et robustes que les algorithmes de l'état de l'art. Plus précisément, pour la reconnaissance de texte, nous avons développé (et largement évalué) un nouveau descripteur basé sur HOG, et dédié au traitement du texte Roman, baptisé T-HOG. Nous avons montré sa valeur en tant que post-filtre pour un détecteur de texte existant (SNOOPERTEXT). Nous avons également amélioré l'algorithme SNOOPERTEXT en développant une approche multi-échelle pour traiter des caractères de taille très différentes tout en limitant la sensibilité de l'algorithme aux différents artéfacts. Pour le suivi des objets de textes, nous avons décrit quatre manières de combiner la détection et le suivi, et nous avons développé un tracker particulier, basé sur un filtre particulaire exploitant le T-HOG. Pour le suivi des objets rigides, nous avons développé un nouvel algorithme précis et robuste (AFFTRACK) qui combine le KLT tracker avec une calibration améliorée de la caméra. Nous avons largement testé nos algorithmes sur plusieurs bases de données de la littérature. Nous avons également créé plusieurs bases de données (publiquement disponibles) pour l'évaluation des algorithmes de détection, suivi de textes et de suivi d'objets rigides dans les vidéos.

Abstract

In this thesis we address three computer vision problems: (1) the detection and recognition of flat text objects in images of real scenes; (2) the tracking of such text objects in a digital video; and (3) the tracking an arbitrary three-dimensional rigid object with known markings in a digital video. For each problem we developed innovative algorithms, which are at least as accurate and robust as other state-of-the-art algorithms. Specifically, for text classification we developed (and extensively evaluated) a new HOG-based descriptor specialized for Roman script, which we call T-HOG, and showed its value as a post-filter for an existing text detector (SNOOPERTEXT). We also improved the SNOOPERTEXT algorithm by using the multi-scale technique to handle widely different letter sizes while limiting the sensitivity of the algorithm to various artifacts. For text tracking, we describe four basic ways of combining a text detector and a text tracker, and we developed a specific tracker based on a particle-filter which exploits the T-HOG recognizer. For rigid object tracking we developed a new accurate and robust algorithm (AFFTRACK) that combines the KLT feature tracker with an improved camera calibration procedure. We extensively tested our algorithms on several benchmarks well-known in the literature. We also created benchmarks (publicly available) for the evaluation of text detection and tracking and rigid object tracking algorithms.

Preface

This thesis was developed under a joint doctoral program of Universidade Estadual de Campinas (UNICAMP) and Université Pierre et Marie Curie (UPMC). I began my Ph. D. studies in 2007 at UNICAMP's Institute of Computing (IC), where I developed part III of this thesis, under the supervision of Professors Jorge Stolfi and Neucimar Jerônimo Leite. In 2009–2010 I spent 18 months at UPMC's Laboratoire d'Informatique de Paris 6 (LIP6), in the Machine Learning and Information REtrieval group (MALIRE), where I started working on parts I and II of this thesis, under the supervision of Professors Matthieu Cord and Nicolas Thome and in collaboration with other French researchers.

At MALIRE I worked on a system to extract textual information from urban images, which may then be integrated with cartographic databases, geo-localization data, business directories, and other information sources. This system was developed within the scope of the iTowns project, which aims to build tools for virtual navigation in urban environments.

The CAPES/Cofecub agreement provided the financial support during my stay in France. In Brazil I was supported by a grant from Fundação de Amparo à Pesquisa do Estado de São Paulo (FAPESP).

Acknowledgements

Agradeço primeiramente a Deus, pela força necessária para a conclusão deste doutorado.

Agradeço muito ao meu orientador o professor Dr. Jorge Stolfi, uma pessoa genial, íntegra e que admiro muito. Agradeço ao meu co-orientador Dr. Neucimar J. Leite por toda a ajuda, pelos bons conselhos, pela assistência em Paris e pela oportunidade de realizar o doutorado cotutela na UPMC.

Agradeço aos meus queridos pais Jovino e Doraci, e aos meus familiares Verediane, Rosane, Gildo, Angelita, Marcos, Jhony, Ariane, Fabrícia, Rodrigo, Sara, e a família de Santi pelo apoio, incentivo, carinho e paciência.

Agradeço em especial a Juliana de Santi ("Juinha") por todo o seu amor, carinho e enorme paciência.

Agradeço aos grandes amigos de tempos antigos: Tiago Sak, Ricardo Dutra da Silva, Leyza B. Dorini, Fábio A. Dorini, Evandro C. Bracht, Renato Neves, Mayza Santos, Hélio Pedrini, William R. Schwartz e Murilo V. G. da Silva.

Agradeço a todos os amigos do Instituto de Computação (LRC, RECOD, LAS, LIV, LSD, LOCO), dos quais não citarei nomes com receio de esquecer alguém.

Agradeço aos grandes amigos da maison du Brésil e de Paris: Paulo Sampaio, Paula Borges Monteiro, Marina Mennucci, Leandro Ferrari Thomaz, Teresa Marques, Greciely Costa, Paula Chiaretti, Rodrigo Ferracine Rodrigues e Solveig Gram Jensen. Agradeço muito aos amigos do 3ème étage, também não citarei nomes com receio de esquecer alguém, por todas as boas conversas durante os bolos feitos pelo grande amigo Igor Teixeira e confraternizações. Agradeço também ao pessoal do 2ème étage, em especial a Pedro Schio por toda a ajuda logo que cheguei em Paris.

Agradeço ao CNPQ, e principalmente ao acordo CAPES-COFECUB e à FAPESP pelo su-

porte financeiro.

Agradeço também a contribuição de todos os professores e funcionários do Instituto de Computação, e todos aqueles que contribuíram para a realização deste projeto.

Je remercie avec beaucoup de gratitude mon directeur de thèse, le professeur Dr. Matthieu Cord, pour m'avoir accueillie au LIP6 et pour sa disponibilité, collaboration, patience, aide scientifique durant ces années de doctorat.

Je remercie également mon encadrement Dr. Nicolas Thome pour la qualité de l'encadrement, patience, et pour l'aide scientifique.

Je remercie toutes les personnes avec qui j'ai eu la chance de collaborer: Jonathan Fabrizio, Jonathan Guyomard, Beatriz Marcotegui Iturmendi e Frédéric Precioso.

Merci à mes amis au LIP6: David Picard, Corina Iovan, Sandra Eliza Fontes de Avila, Carlos Sureda Gutiérrez, Hanlin Goh et Christian Theriault.

Je remercie Dr. Patrick Pérez et Dr. Arnaldo de A. Araújo pour avoir accepté de rapporter ce manuscrit ainsi que Dr. Marcin Detyniecki et Dr. Hélio Pedrini pour leur participation à mon jury de thèse.

Finalement, je tiens à remercier sincèrement l'équipe du LIP6, EDITE et l'accord CAPES/COFECUB pour le soutien financiers pendant mon séjour en France.

Resumo expandido da tese

Visão computacional pode ser definida como a análise de informações visuais, por exemplo em imagens digitais, vídeos, para extrair informações simbólicas e quantitativas; tais como forma, posição e tipo de objetos que aparecem nelas. Nesta tese nós consideramos três problemas de visão computacional:

(1) a detecção e reconhecimento de objetos planos em imagens de cenas reais;

(2) rastreamento desses objetos de texto em vídeos digitais;

(3) rastreamento de um objeto arbitrário 3D com forma conhecida em um vídeo digital.

Estes problemas são os assuntos das Partes I, II e III desta tese, respectivamente.

No problema (1) o objetivo é localizar os objetos físicos de texto na scena; o que não inclui a identificação dos caracteres que compõem aqueles textos, problema este conhecido como *reconhecimento óptico de caracteres* ou OCR. Os objetos rastreados no problema (2) não são conhecidos a priori e são determinados pelo próprio algoritmo de detecção de texto. Como eles são planos, e tem tamanho e proporção desconhecidos, suas posições no espaço 3D não podem ser determinadas a partir de suas projeções; portanto, eles são representados somente como regiões no domínio da imagem. No problema (3), por outro lado, o objeto de interesse é rastreado através de um conjunto de marcas planas não coplanares (não necessariamente textuais) cuja aparência e posição relativa são dadas como entrada. Com essas informações, somos capazes de determinar, para cada quadro, a posição 3D exata do objeto relativo a câmera.

Reconhecimento e detecção de texto

Na Parte I desta tese, estamos interessados principalmente no *problema da classificação texto/não-texto*. Este problema recebe como dado de entrada uma sub-imagem específica de

uma foto digital ou de um quadro de vídeo. A saída é uma decisão binária que idealmente deve ser 'VERDADEIRO' se a sub-imagem contém textos com caracteres latinos (ou similares) e 'FALSO' caso contrário. Veja a Figura 1. Este problema surge em muitas aplicações, como controle e monitoramento de velocidade de carros baseado em placas de tráfego e sinalização e construção de diretórios de lojas e serviços por rua.

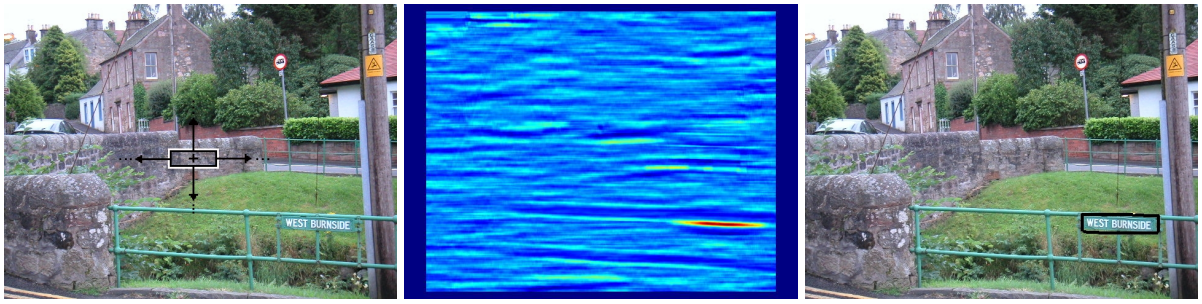


Figura 1: Classificação de texto: à esquerda a imagem de entrada e uma instância da janela (em preto) utilizada para classificar as regiões da imagem. Ao centro, a saída de um detector por janela deslizante usando o classificador T-HOG+SVM, codificado como a cor do pixel central da janela (tons quentes para saídas positivas e tons frios para saídas negativas). À direita, a união das 100 janelas com os maiores scores de classificação.

Especificamente, exploramos o uso do *histograma das orientações dos gradientes* (HOG) [24] para o problema acima descrito. Classificadores baseados em HOG são utilizados para o reconhecimento de pedestres [24], de objetos sólidos [94], e para a detecção de texto [34, 72, 87, 93]. Nesta tese, descrevemos um novo classificador de texto, *Text HOG* (T-HOG) [68], que acuradamente e eficientemente caracteriza textos de uma linha. O *T-HOG* é uma melhoria do HOG padrão, otimizado para a tarefa específica de reconhecimento de linhas de texto. Por meio de uma série extensa de testes, nós mostramos experimentalmente que T-HOG é mais preciso que o HOG padrão, para reconhecimento de linhas de texto, para uma ampla gama de configurações de parâmetros.

Mostramos que a combinação de um detector de texto "permissivo", SNOOPERTEXT [66], com um classificador T-HOG como filtro de saída supera outros detectores de texto estado-da-arte descritos na literatura [27], incluindo os vencedores do desafio ICDAR [55]. Incidentalmente, no decorrer dos testes, detectamos falhas significativas na medida de avaliação utilizada no ICDAR [57], e então desenvolvemos uma medida mais robusta. Também melhoramos a eficiência do detector SNOOPERTEXT através de técnicas multiescala que permitem a detecção de caracteres com tamanhos variados e que o tornam imune à ruídos aleatórios e a variações na

textura.

O classificador T-HOG pode ser utilizado em diversas aplicações tais como a detecção de texto, rastreamento de texto, e reconhecimento por OCR. Em particular, nesta tese mostramos que o T-HOG pode ser utilizado por si só em detecção de texto, através de uma abordagem janela deslizante (veja a Figura 1), e pode também ser utilizado como um componente de um algoritmo para rastreamento de texto.

Rastreamento de texto

Em algumas aplicações de detecção de texto, a entrada é um vídeo ao invés de uma imagem estática. Os exemplos incluem a indexação automática de bibliotecas de vídeos, a construção de diretórios de negócios a partir de vídeos obtidos por câmeras montadas em veículos, a identificação de veículos por suas placas, e a geração de descrições de áudio a partir de imagens do ambiente para ajudar pessoas com problemas de visão. Veja a Figura 2.

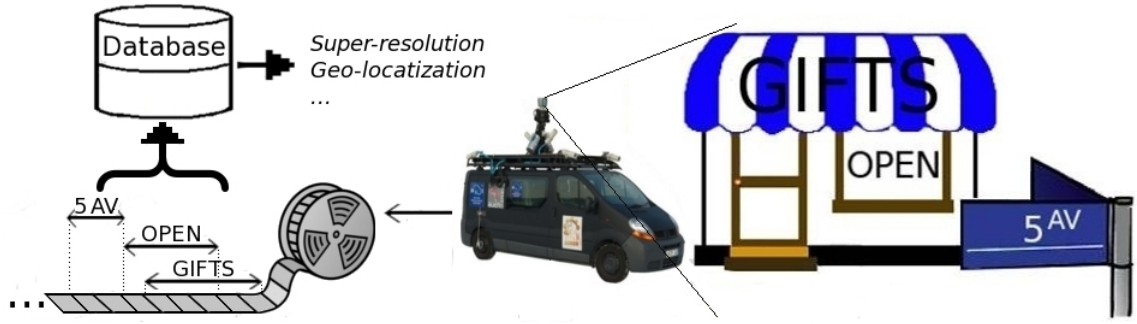


Figura 2: Uma aplicação para rastreamento de texto.

Para estas aplicações, não é suficiente aplicar um detector de texto separadamente em cada quadro. Esta abordagem simplista ignora a coerência temporal entre quadros consecutivos de um mesmo vídeo, e portanto produz milhares de regiões de texto detectadas que nem sempre contém a mesma informação textual. Para evitar esta redundância, é necessário *rastrear* cada objeto físico de texto através de quadros sucessivos. Além de prover uma saída mais sucinta, um algoritmo de rastreamento de texto pode explorar a redundância entre quadros para obter resultados mais precisos e consistentes do que os obtidos por detecção de texto quadro-a-quadro.

Finalmente, um rastreador de texto também pode utilizar técnicas de super-resolução para extrair imagens de texto com uma maior qualidade e resolução que àquelas obtidas a partir de qualquer quadro único.

Com esta motivação, consideramos na Parte II da tese o *problema da detecção de texto e rastreamento*. Este problema recebe como dado de entrada um vídeo digital. A saída consiste em um conjunto de regiões candidatas a texto para cada quadro do vídeo e a identificação das regiões em quadros distintos que supostamente pertencem ao mesmo objeto físico de texto. Veja a Figura 3. Tal como na Parte I, estamos principalmente interessados em textos presentes em

Figura 3: Saída de um algoritmo de rastreamento de texto: regiões de texto detectadas (retângulos) e a informação de rastreamento (setas).

vídeos de cenas urbanas externas ou no interior de prédios.

Nossa contribuição principal para este problema é a descrição e análise de quatro estratégias para rastreamento de texto, que diferem na maneira como a detecção de texto e o rastreamento são intercalados. A maioria das soluções existentes para o problema são variações destes quatro algoritmos.

Outra contribuição nossa é a definição de métricas especiais para avaliar o desempenho de sistemas de rastreamento de texto. Essas métricas avaliam o sistema com base em três aspectos:

(1) *acurácia na detecção de regiões de texto*: a habilidade em detectar objetos de texto visíveis em um quadro;

(2) *acurácia detecção de objetos de texto*: a habilidade em detectar os diferentes objetos de texto ao menos uma vez em algum quadro do vídeo;

(3) *acurácia no rastreamento de objetos de texto*: a habilidade em identificar regiões que pertencem ao mesmo objeto de texto em diferentes quadros.

Nós também desenvolvemos um rastreador especificamente projetado para objetos de texto, baseado em filtro de partículas e no classificador T-HOG. O classificador T-HOG é utilizado duas vezes nesse rastreador: por um lado, como uma assinatura da região da imagem, para avaliar a similaridade entre o conteúdo de duas regiões em quadros sucessivos; e, por outro lado, como um classificador, para determinar se o conteúdo da região extrapolada tem aparência de texto.

Finalmente, como parte deste trabalho, nós também desenvolvemos uma base de dados com seis vídeos reais de cenas urbanas, com as correspondentes anotações XML [62], que esperamos que sejam úteis a outros pesquisadores que trabalham no mesmo problema.

Rastreamento de objetos 3D

O rastreamento do movimento de objetos rígidos no espaço tri-dimensional é um problema geral na análise de vídeos com uma ampla gama de aplicações, que incluem a manipulação e montagem por rôbos, gestão de frotas de veículos e rastreamento, navegação autônoma, e realidade aumentada (a sobreposição de imagens virtuais em imagens do mundo real, tal como na Figura 4).

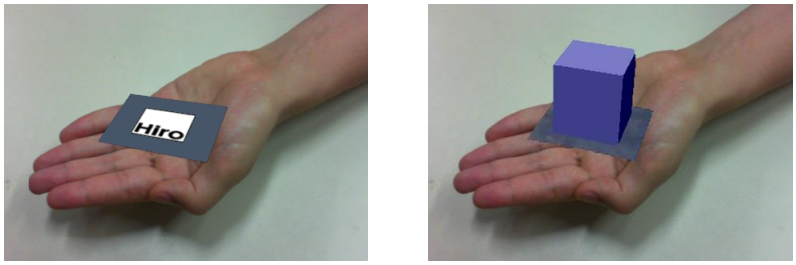


Figura 4: Realidade aumentada.

Para resolver este problema não é necessário um conhecimento completo sobre a forma ou aparência do objeto; basta rastrear um número suficiente de marcas fixadas sobre o mesmo. Portanto, na Parte III da tese, nós consideramos o *rastreamento de objetos rígidos tridimensionais baseado em marcas (características)*; mais precisamente, o rastreamento de objetos sólidos (tais como um caminhão ou um pacote) através de um conjunto de marcas bi-dimensionais conhecidas fixadas sobre a sua superfície. As marcas podem ser intrínsicas ou acidentais (tais

como letras, rótulos, tomadas de parede, logos, *etc*.), ou marcas fiduciais fixadas sobre o objeto especificamente para fins de rastreamento.

Os dados de entrada para este problema são: um vídeo digital e uma lista de marcas a rastrear, que inclui a posição de cada marca no objeto, bem como, a posição aproximada da mesma no primeiro quadro. A saída consiste na posição do objeto no espaço, relativa à câmera, e outros parâmetros de câmera relevantes (tais como o fator de zoom), para cada quadro do vídeo. Em particular, se o objeto rastreado é fixo no ambiente ou outra estrutura estacionária, a saída nesse caso é a posição absoluta da câmera em cada quadro. Veja a Figura 5.

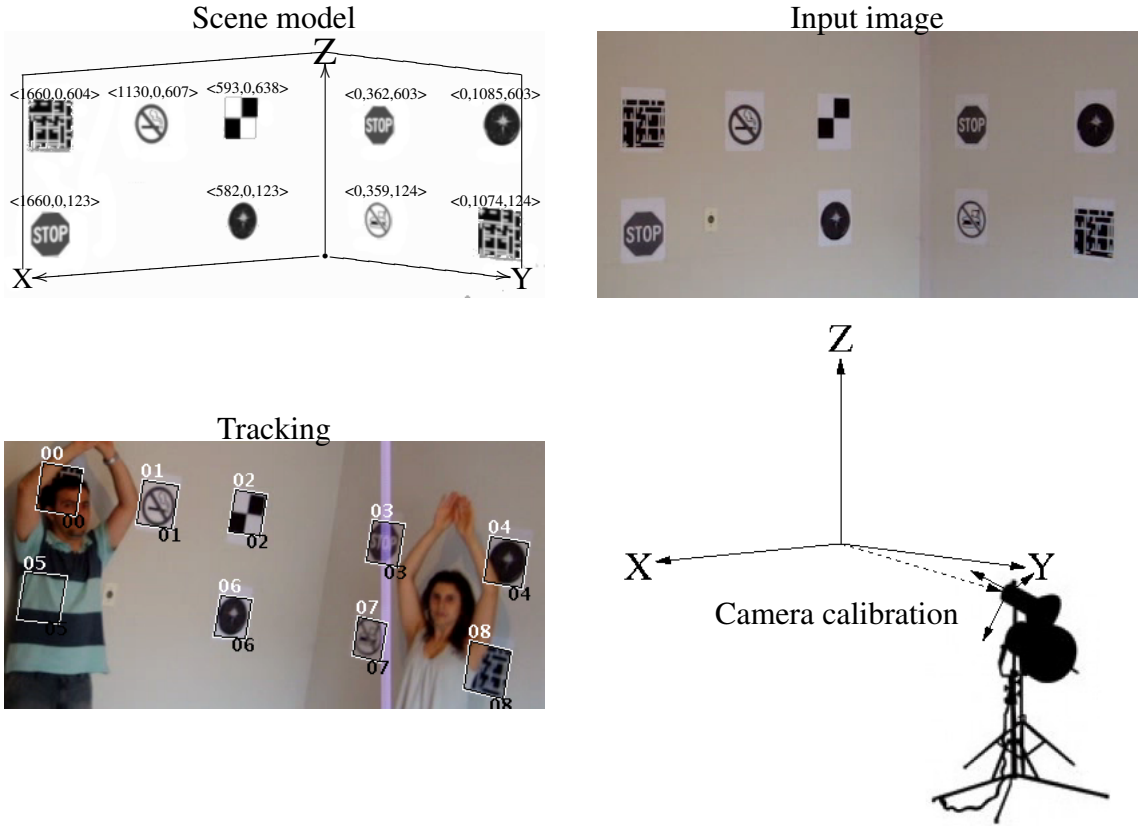


Figura 5: Rastreamento de objetos rígidos através de marcas conhecidas e calibração de câmera.

Nossa principal contribuição a este problema é a descrição de um algoritmo de rastreamento denominado AFFTRACK [65]. Este algoritmo utiliza uma combinação sinérgica de dois módulos principais: *um rastreador de marcas* (FF), e um *calibrador de câmera* (CC). O rastreador determina a posição de cada marca separadamente no próximo quadro, e o calibrador calcula

a posição da câmera a partir dessas posições. Os parâmetros calibrados da câmera para cada quadro do vídeo são então utilizados para prever a posição e aparência de cada marca no próximo quadro. A sinergia entre esses dois módulos permite que AFFTRACK recupere marcas após oclusões de duração arbitrária.

Um aspecto chave do AFFTRACK é a utilização de *pesos de confiança* para expressar a confiabilidade de cada característica em cada quadro. Este peso inicialmente depende da similaridade das marcas identificadas pelo rastreador, e é iterativamente ajustado de acordo com a consistência global das posições das marcas. Essa classificação *"nebulosa"* da confiabilidade das posições das marcas contrasta com a classificação binária *inlier/outlier* utilizada na abordagem RANSAC [30].

AFFTRACK inova em relação a outros rastreadores de objetos 3D descritos na literatura em vários aspectos: ele permite o rastreamento de objetos em vídeos com zoom variável e variação na distorção da lente; ele não requer um modelo geométrico completo do objeto; e ele também não requer a seleção de quadros chave. Em nossos testes, AFFTRACK revelou-se mais robusto e acurado que rastreadores populares como os incluídos no ARToolKit [40] de H. Kato e na biblioteca OpenCV [16] da Intel. Nossa esperança é que AFFTRACK seja útil como uma base para outros algoritmos de visão computacional.

Como sub-produto de nossa avaliação do AFFTRACK, nós também desenvolvemos uma base de dados com oito vídeos reais e nove vídeos sintéticos [61]. Os gabaritos disponíveis para os vídeos sintéticos incluem as posições exatas de cada característica e a calibração da câmera para cada quadro.

Publicações

As publicações diretamente relacionadas a esta tese (em ordem cronológica), são:

- *Integrating Tsai's Camera Calibration Algorithm with KLT Feature Tracking*.
  R. Minetto, N.J. Leite and J. Stolfi.
  Proceedings of IV Workshop de Visão Computacional. 2008.

- AFFTRACK*: Robust Tracking of Features in Variable-Zoom Videos*.
  R. Minetto, N.J. Leite and J. Stolfi.
  IEEE International Conference on Image Processing (ICIP). 2009.

- SNOOPERTEXT*: A Multiresolution System for Text Detection in Complex Visual Scenes*.
  R. Minetto, N. Thome, M. Cord, J. Fabrizio and B. Marcotegui.
  IEEE International Conference on Image Processing (ICIP). 2010.

- SNOOPERTRACK*: Text Detection and Tracking for Outdoor Videos*.
  R. Minetto, N. Thome, M. Cord, N.J. Leite and J. Stolfi.
  IEEE International Conference on Image Processing (ICIP). 2011.

- *Text Detection and Recognition in Urban Scenes*.
  R. Minetto, N. Thome, M. Cord, J. Stolfi, F. Precioso, J. Guyomard and N. J. Leite.
  IEEE/ISPRS Workshop on Computer Vision for Remote Sensing of the Environment CVRS-ICCV. 2011.

Résumé étendu de la thèse

Un champ important de la Vision par ordinateur porte sur l'analyse de l'information visuelle des images numériques ou des vidéos, pour en extraire des informations symboliques et quantitatives, comme la forme, la position ou le type d'objets qui y apparaissent. Dans cette thèse, nous considérons trois problèmes de vision par ordinateur:

(1) la détection et la reconnaissance d'objets de texte dans des images de scènes réelles;
(2) le suivi de ces objets de texte dans les vidéos;
(3) le suivi d'objets tridimensionnels rigides et arbitraires avec des amers connus dans une vidéo numérique.

Ces problèmes sont les sujets des Parties I, II et III de cette thèse, respectivement.

Les objets suivis dans le problème (2) ne sont pas connus à priori et sont déterminés par l'algorithme de détection de texte. Comme ils sont plats mais de taille et de proportion inconnues, leur position dans l'espace 3D ne peut être déterminée à partir de leurs projections. Par conséquent, ils seront représentés comme des régions du domaine image. Dans le problème (3) en revanche, les objets d'intérêt sont suivis en considérant un ensemble d'amers plans connus et non-coplanaires (pas nécessairement textuel) dont les apparitions et les positions relatives sont données en entrée. Avec cette information, nous sommes en mesure de déterminer, pour chaque trame, la position 3D complète de l'objet par rapport à la caméra.

Détection et reconnaissance de texte

Dans la Partie I de la thèse, nous nous intéressons principalement au *problème de la classification texte/non-texte*. Les données d'entrée dans ce cas sont une sous-image d'une photo numérique ou d'une trame d'une vidéo numérique. La sortie de l'algorithme est une décision binaire qui devrait être idéalement 'TRUE' si l'image contient un texte avec des caractères romans et 'FALSE' sinon (voir la Figure 6). Ce problème se pose dans de nombreuses applica-

tions, telles que la surveillance, le contrôle de la vitesse d'un véhicule en fonction des panneaux de signalisation, ou encore la construction de bases de données concernant les information trouvées communément dans des villes (noms de rues, noms de magasins, *etc*.).

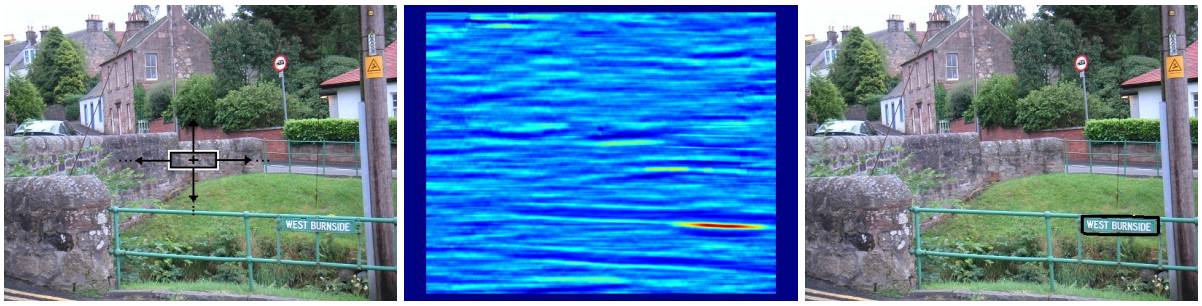


FIG. 6 – Classification de texte : à gauche, l'image d'entrée et la fenêtre (en noir) utilisée pour classer les régions de l'image. Au centre, la sortie de la classification T-HOG+SVM avec une fenêtre glissante sur chaque pixel de l'image. Chaque pixel a été codé avec un code couleur: des couleurs chaudes pour une sortie positive (correspondant à du texte), et des couleurs froides pour une sortie négative (correspondant à du non-texte). À droite, on présente l'union des 100 fenêtres avec les scores les plus élevés.

Plus précisément, pour la reconnaissance de texte, nous explorons l'utilisation des *histogrammes de gradients orientés* (HOG) [24]. Les HOG classificateurs ont été utilisés pour la reconnaissance des piétons [24], des objets rigides [94], et pour la détection de texte [34, 72, 87, 93]. Nous décrivons un classificateur dédié au traitement du texte roman, Texte HOG (T-HOG) [68], qui caractérise efficacement et précisément les lignes de textes en style roman. Le T-HOG est une amélioration du HOG standard, optimisé pour la tâche spécifique de reconnaissance de lignes de texte. Par une série de tests, nous montrons expérimentalement que le T-HOG est plus précis dans la reconnaissance des lignes de texte que le HOG standard, pour une large gamme de paramètres.

Le classificateur T-HOG est utile dans de nombreuses applications comme la détection et le suivi du texte dans des vidéos, et la reconnaissance par OCR. Dans cette thèse, nous montrons en particulier comment le classificateur T-HOG peut être utilisé tout seul comme un détecteur de texte, à travers le concept de fenêtre glissante. Il peut aussi être utilisé comme une composante importante d'un algorithme de suivi de textes. Nous montrons également que la combinaison d'un détecteur de texte, SNOOPERTEXT [66] avec un classificateur T-HOG en post-filtrage, surpasse les algorithmes de l'état de l'art pour la détection de texte décrit dans la littérature [27], y compris ceux de la compétition ICDAR [55]. De plus, lors de l'évaluation de

notre système, nous avons détecté des faiblesses significatives concernant la mesure standard utilisée pour comparer les systèmes dans la compétition ICDAR [57], et nous avons développé une mesure plus robuste.

Suivi du texte

Dans certaines applications de détection de texte, l'entrée de l'algorithme est un flux vidéo au lieu d'une seule image statique. Dans ce cas, les exemples applicatifs concernent l'indexation automatique de bibliothèques vidéo, la construction de bases de données d'adresse de rues à partir de vidéos prises par une caméra installée sur un véhicule, l'identification des véhicules par leurs plaques d'immatriculation, et la génération de descriptions sonores de l'environnement pour aider les personnes mal voyantes (voir la Figure 7).

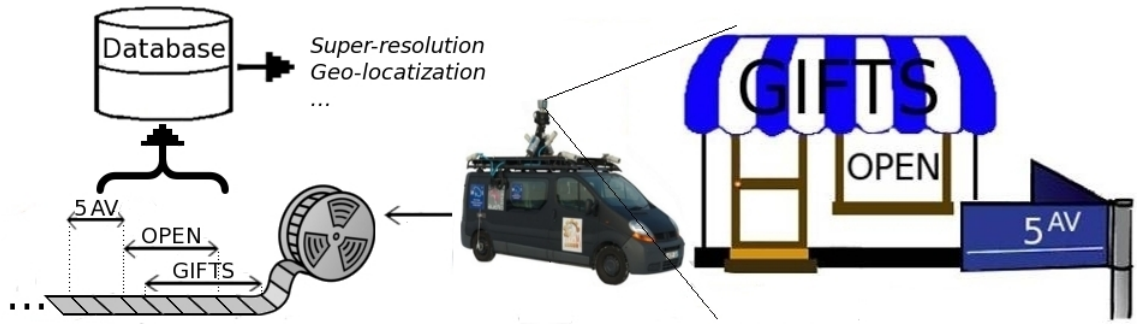


FIG. 7 – Une application pour le suivi de texte.

Pour ces applications, il n'est pas suffisant d'appliquer un détecteur de texte séparément pour chaque trame. Cette approche simpliste qui ignore la cohérence temporelle entre les images de la vidéo, détecte généralement des milliers de zones de texte avec une redondance très importante. Pour éviter ce problème, il est nécessaire de suivre chaque objet physique travers les trames successives de la vidéo. En plus d'offrir une sortie plus succincte, un algorithme de suivi de texte peut exploiter la redondance entre les trames pour surpasser, en termes de précision et rappel, les détecteurs de texte appliqués image par image. Enfin, un tracker de texte peut également employer des techniques de super-résolution pour extraire des composantes de texte avec une meilleure résolution et une meilleure qualité que celles obtenues à partir d'une seule trame.

Fort de cette motivation, nous considérons dans la Partie II de la thèse le problème de détection et de suivi de composantes textuelle. Les données d'entrée pour ce problème sont des vidéos numériques. La sortie est un ensemble de régions, contenant possiblement du texte, dans chaque trame de la vidéo, ainsi que l'identification des régions de texte supposées contenir les mêmes caractéristiques physiques (voir la Figure 8).

FIG. 8 – La sortie d'un algorithme de suivi de texte montrant les zones de texte détectées (rectangles) et les informations de suivi (flèches).

Comme dans la Partie I, nous nous intéressons principalement aux zones de texte intégrées dans les scènes naturelles urbaines ou les scènes d'intérieur de bâtiments. La vidéo est généralement enregistrée par une caméra en mouvement, tenue par la main ou fixée sur un véhicule.

Notre contribution à ce problème est la description et l'analyse de quatre méthodes principales de suivi de textes, qui diffèrent de la façon dont ils entrelacent la détection et le suivi. La plupart des solutions existantes au problème ci-dessus sont des variations de ces quatre algorithmes.

Une autre contribution de ce travail, concerne la définition des mesures spécifiques pour évaluer la performance des systèmes de suivi de texte. Ces mesures évaluent la précision d'un système basé sur trois aspects:

(1) *la détection de zone de texte*: la capacité à détecter des objets de texte visibles dans une trame;

(2) *la détection d'objets de texte*: la capacité à détecter des objets de texte dans n'importe quelque part de la vidéo;

(3) *suivi des objets de textes*: la capacité d'identifier les régions qui appartiennent au même objet de texte dans différentes trames.

Nous avons également développé un tracker spécialement conçu pour les objets de texte, basé sur un filtre particulaire exploitant le T-HOG. Le T-HOG est utilisé deux fois dans ce

tracker: en premier lieu, comme une signature de régions de l'image afin d'évaluer la similitude entre deux régions dans des trames successives et, en deuxième lieu, comme un classificateur pour mesurer combien le contenu d'une région prédite ressemble à des informations textuelles.

Finalement, dans le cadre de ce travail nous avons également créé une base de données de six vidéos de scènes urbaines, avec des annotations XML associées [62], qui, nous l'espérons, seront utiles à d'autres chercheurs travaillant sur ce problème.

Suivi d'objets 3D

Le suivi du mouvement d'objets rigides dans l'espace 3D est un problème général d'analyse des vidéos, avec une large gamme d'applications dans des thématiques diverses : robotique (e.g. navigation autonome, manipulation et assemblage), analyse vidéo (e.g. suivi de véhicules pour la gestion du trafic), et réalité augmentée (superposition d'images virtuelles sur des images du monde réel, comme dans la Figure 9).

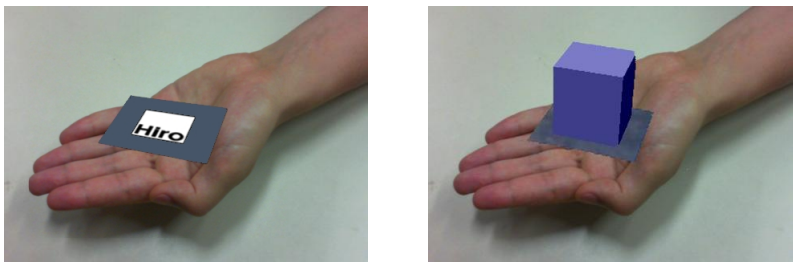


FIG. 9 – Réalité augmentée.

Pour résoudre ce problème, il n'est pas nécessaire d'avoir une connaissance complète de la forme et de l'apparence de l'objet. Il est possible, plus simplement, de suivre un nombre suffisant de caractéristiques fixes sur l'objet. Ainsi, nous abordons dans la Partie III de cette thèse le problème du *suivi d'un objet tridimensionnel rigide basé sur le suivi de caractéristiques particulières*. Par exemple, nous proposons une méthode dédiée au suivi d'un objet solide (comme un camion ou un paquet) en suivant un ensemble de points de référence (amers) connus en deux dimensions et fixés à la surface de cet objet. Dans ce cas, les amers peuvent être soit intrinsèques ou accidentelles (des lettres, des étiquettes, des prises murales, des logos, *etc*.), ou des caractéristiques discriminantes propres à l'objet.

Dans ce contexte de suivi d'objets 3d, les données d'entrée sont une vidéo numérique, une liste de caractéristiques qu'on cherche à suivre, y compris leur emplacement sur l'objet, ainsi que la position approximative de la projection de chaque caractéristique dans la première image. La sortie est représentée par la position de l'objet dans l'espace par rapport à la caméra et par le réglage d'autres paramètres de la caméra (tels que le facteur de zoom et la distorsion) pour chaque image. En particulier, si l'objet suivi fait partie d'un bâtiment ou d'une autre structure stationnaire, la sortie est la position absolue de la caméra pour chaque image. Voir la Figure 10.

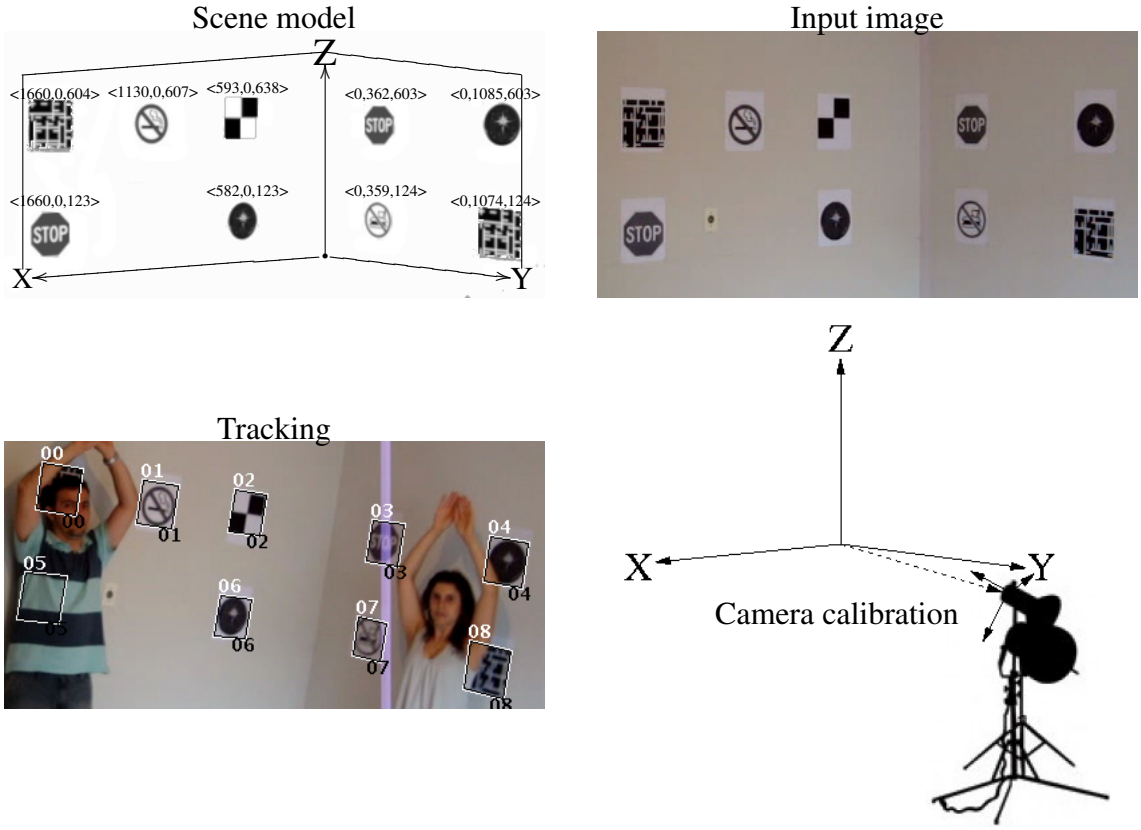


FIG. 10 – Suivi d'objet rigide en utilisant d'amers connus et la calibration de la caméra.

Notre contribution pour résoudre ce problème réside dans la description d'un algorithme baptisé AFFTRACK [65]. Il correspond à une combinaison synergique de deux procédures principales: un *tracker de caractéristiques* multi-échelle (FF) qui localise la position des caractéristiques (amers) sur chaque image, et *une calibration de la caméra* (CC) flexible qui calcule

les paramètres de la caméra à partir de ces positions. Les paramètres de la caméra calibrée pour chaque image de la vidéo sont utilisés pour prédire la position et l'apparence de chaque élément dans l'image suivante. La synergie entre ces deux modules permet à AFFTRACK de récupérer les amers après une occultation de durée arbitraire.

Un aspect clé de AFFTRACK est l'utilisation de *poids de confiance* pour exprimer la fiabilité de chaque caractéristique sur chaque image. Ces poids dépendent d'abord de la qualité de la correspondance obtenue par le tracker, et sont ensuite ajustés itérativement selon la consistance de la position des amers avec le modèle calibré. Ce classement "flou" de la fiabilité de la position constitue une distinction importante par rapport à la classification binaire "inlier/outlier" (caractéristique pertinente/intrus) utilisée par l'algorithme standard RANSAC [30].

AFFTRACK présente de meilleurs résultats que les méthodes publiées dans la littérature, en ce qui concerne la manipulation de vidéos avec un zoom et/ou des distorsions de lentille variables; par ailleurs, il ne demande pas de modèle géométrique complet de l'objet, ni la sélection d'images clés. Lors de nos tests, AFFTRACK a été plus robuste et plus précis que les trackers d'objets populaires inclus dans H. Kato's ARToolKit [40] et dans la bibliothèque OpenCV [16]. Nous espérons qu'AFFTRACK sera utile comme base de mise au point d'autres algorithmes de vision par ordinateur.

Comme un sous-produit de notre évaluation de AFFTRACK, nous avons également créé une base de données avec huit vidéos réelles et neuf vidéos générées par ordinateur [61]. La vérité terrain inclue des coordonnées précises pour chaque caractéristique et les paramètres de calibration de la caméra dans chaque image.

Publications

La liste des publications directement concernées par cette thèse (par ordre chronologique) sont:

- *Integrating Tsai's Camera Calibration Algorithm with KLT Feature Tracking*.
  R. Minetto, N.J. Leite and J. Stolfi.
  Proceedings of IV Workshop de Visão Computacional. 2008.
- AFFTRACK*: Robust Tracking of Features in Variable-Zoom Videos*.
  R. Minetto, N.J. Leite and J. Stolfi.
  IEEE International Conference on Image Processing (ICIP). 2009.

– SNOOPERTEXT*: A Multiresolution System for Text Detection in Complex Visual Scenes*.
R. Minetto, N. Thome, M. Cord, J. Fabrizio and B. Marcotegui.
IEEE International Conference on Image Processing (ICIP). 2010.
– SNOOPERTRACK*: Text Detection and Tracking for Outdoor Videos*.
R. Minetto, N. Thome, M. Cord, N.J. Leite and J. Stolfi.
IEEE International Conference on Image Processing (ICIP). 2011.
– *Text Detection and Recognition in Urban Scenes*.
R. Minetto, N. Thome, M. Cord, J. Stolfi, F. Precioso, J. Guyomard and N.J. Leite.
IEEE/ISPRS Workshop on Computer Vision for Remote Sensing of the Environment CVRS-ICCV. 2011.

Contents

Chapter 1

Introduction

Computer vision can be defined as the parsing of visual information, e.g. in digital images, videos, to extract symbolic and quantitative information; such as the shape, position and type of objects that appear in them. In this thesis we consider three computer vision problems:

(1) the detection and classification of flat text objects in images of real scenes;

(2) tracking of such text objects in a digital video;

(3) tracking an arbitrary 3D object with known shape in a digital video.

These problems are the topics of parts I, II, and III of this thesis, respectively.

In problem (1), the goal is to locate the physical text objects in the scene; it does not include the identification of the characters in those texts, which is a separate problem known as *optical character recognition* or OCR. The text objects tracked in problem (2) are not known a priori and are determined by some text detection algorithm that solves problem (1). Since they are flat, and have unknown size and aspect ratio, their positions in 3D space cannot be determined from their projections; therefore, they are represented only as regions of the image domain. In problem (3), on the other hand, the object of interest is tracked by following a set of non-coplanar flat markings (not necessarily textual) whose appearances and relative positions are given as input. With this information we are able to determine, for each frame, the full 3D position of the object relative to the camera.

1.1 Text detection and classification

In part I of the thesis, we are mainly concerned with the *text/non-text classification problem*. The input data for this problem is a specific sub-image of a digital photo or video frame. The output is a binary decision that ideally should be 'TRUE' if the sub-image contains a text in

Roman-like characters and 'FALSE' otherwise. See Figure 1.1. This problem arises in many applications, such as monitoring and controlling car speed based on traffic signs and constructing street business directories.

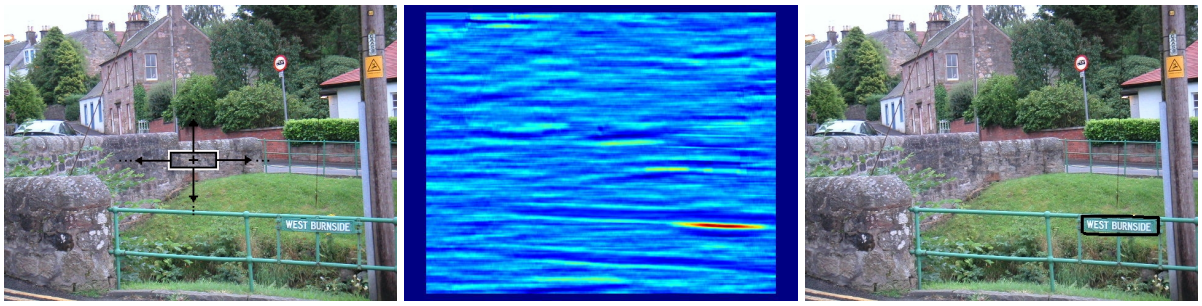


Figure 1.1: Text classification: in left the input image and the window (in black) used to classify the image regions. In center, the output of the T-HOG+SVM sliding window classifier on every pixel of the image, encoded as the color of the central pixel (warm tones for positive output, cold tones for negative). In the right, the union of the 100 windows with highest scores.

Specifically, we explore the use of *histogram of oriented gradients* (HOG) [24] for the above problem. HOG-based classifiers have been used for the recognition of pedestrians [24], solid objects [94], and for text detection [34, 72, 87, 93]. We describe a novel text classifier, *Text HOG* (T-HOG) [68], that efficiently and accurately characterizes single-line texts. The T-HOG is an improvement of the standard HOG, optimized for the specific task of text line classification. By an extensive series of tests, we show experimentally that T-HOG is more accurate, for text line classification, than the standard HOG for a wide range of parameters settings.

We show that the combination of a "permissive" text detector, SNOOPERTEXT [66], with a T-HOG classifier as a post-filter outperforms state-of-the-art text detectors described in the literature [27], including the winners of the ICDAR challenge [55]. Incidentally, while evaluating this system we detected significant weaknesses of the ICDAR scoring method [57], and we developed a more robust one. We also improved the efficiency, accuracy and robustness of the SNOOPERTEXT detector by using the multi-scale technique to handle widely different letter sizes and to render it immune to small-scale noise and texture.

We expect that the T-HOG classifier will be useful for text detection, text tracking, OCR, and related applications. In particular, we show how the T-HOG classifier could be used by itself in a sliding-window text detector, as well as a component of a text tracking algorithm.

1.2 Text tracking

In some text detection applications, the input is a video stream instead of a static image. Examples include the automatic indexing of video libraries, the construction of street directories

from videos taken by car-mounted cameras, the identification of vehicles by their license plates, and the generation of audio descriptions of the environment for blind people. See Figure 1.2.

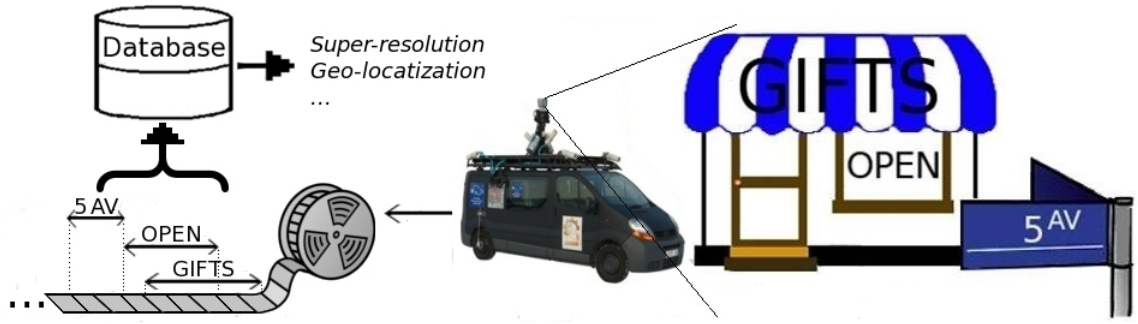


Figure 1.2: An application for text tracking.

For those applications, it is not sufficient to apply a text detector separately to each frame. This simplistic approach ignores the temporal coherence between video frames, and therefore it will usually produce thousands of detected text regions with almost, but not always quite, the same textual information. To avoid this redundancy, it is necessary to *track* each physical text object across successive frames. Besides providing more succinct output, a text tracking algorithm can exploit the redundancy between frames to outperform single-frame text detectors in both precision and recall. Finally, a text tracker can also use super-resolution techniques to extract images of the text with much higher resolution and quality than those of any single frame.

With this motivation, in part II of the thesis we consider the *text detection and tracking problem*. The input data for this problem is a digital video. The output is a set of candidate text regions for each video frame and the identification of the regions which supposedly belong to the same physical text object. See Figure 1.3. As in part I, we are primarily concerned with

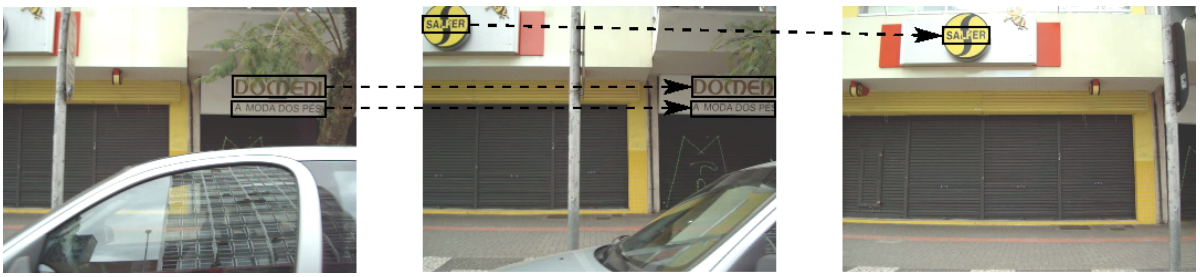


Figure 1.3: The output of a text tracking algorithm, showing the detected text regions (rectangles) and the tracking information (arrows).

text embedded in natural urban scenes or building interiors. The video is typically recorded by a moving camera, either hand-held or fixed on a vehicle.

Our main contribution to this problem is the description and analysis of four basic text tracking methods, which differ on how they interleave text detection and tracking. The most efficient ones incorporate many ideas from SNOOPERTRACK [67], an algorithm for the automatic detection and tracking of text objects — such as store names, traffic signs, license plates, and advertisements — in videos of outdoor scenes. Most of the existing solutions to the above problem are variations of these four algorithms.

Another contribution is the definition of special metrics to evaluate the performance of text tracking systems. These metrics evaluate a system's accuracy based on three aspects: *text region detection*, the ability to detect visible text objects in a frame; *text object detection*, the ability to detect text objects somewhere in the video; and *text object tracking*, the ability to identify regions that belong to the same text object in different frames.

We also developed a tracker specifically designed for text objects, based on particle filtering and the T-HOG classifier. The T-HOG is used twice in this tracker: first, as an image region signature, to evaluate the similarity between the contents of two regions in successive frames; and, second, as a classifier, to measure how much "text-like" is the contents of a predicted region.

Finally, as part of this work we also created a benchmark of six videos of urban scenes, with associated XML annotations [62], which we hope will be useful to other researchers working on this problem.

1.3 Tracking of 3D objects

Tracking the motion of rigid objects in the tri-dimensional space is a general video analysis problem with a wide range of applications, including robotics handling and assembly, vehicle fleet management and tracking, autonomous navigation, and augmented reality (the overlay of virtual imagery on real-world images, as in figure 1.4).

To solve this problem it is not necessary to have full knowledge of the object's shape and appearance; it is enough to track a sufficient number of features fixed on it. Therefore, in part III of this thesis we address the *feature-based tracking of a tri-dimensional rigid object*; namely, the tracking of a solid object (such as a truck or a package) by following a set of known two-dimensional features attached to its surface. The features can be either intrinsic or accidental (such as letters, labels, wall outlets, logos, *etc*.), or fiducial marks attached to the object specifically for the purpose.

The input data for this problem consists of a digital video, a list of the features to be tracked, including their placement on the object, as well as the approximate position of the projection of each feature in the first frame. The output consists of the position of the object in space relative

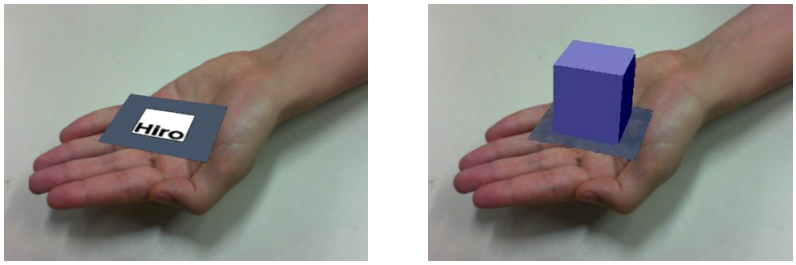


Figure 1.4: Augmented Reality.

to the camera and other relevant camera parameters (such as zoom factor) for each frame. In particular, if the tracked object is part of a building or some other stationary structure, the output is the absolute position of the camera for each frame. See Figure 1.5.

Our major contribution to this problem is the description of an algorithm nicknamed AFFTRACK [65]. It is a synergistic combination of two main procedures: a multi-scale *feature finder* (FF) that locates the positions of the features (markers) on each frame, and a flexible *camera calibrator* (CC) that computes the camera parameters from those positions. The calibrated camera parameters for each video frame are used to predict the position and appearance of each feature in the next frame. The synergy between those two modules allows AFFTRACK to recover features after occlusions of arbitrary duration.

A key aspect of AFFTRACK is the use of a *confidence weight* to express the reliability of each feature on each frame. This weight initially depends on the quality of the match obtained by the finder, and is then adjusted iteratively according to the feature's positional consistency with the calibrated model. This "fuzzy" grading of feature position reliability contrasts with the binary inlier/outlier classification used by RANSAC [30].

AFFTRACK improves on other object trackers described in the literature, in that it can handle videos with variable zoom and variable lens distortion; it does not require a complete geometric model of the object; and it does not require the selection of key frames. In our tests, AFFTRACK was more robust and accurate than the popular object trackers included in H. Kato's ARToolKit [40] and in the OpenCV library [16]. We hope that AFFTRACK may be useful as a building block of other computer vision algorithms.

As a byproduct of our evaluation of AFFTRACK, we also created a benchmark of eight real and nine computer-generated videos [61], with accurate feature positions and camera calibrations provided for each frame.

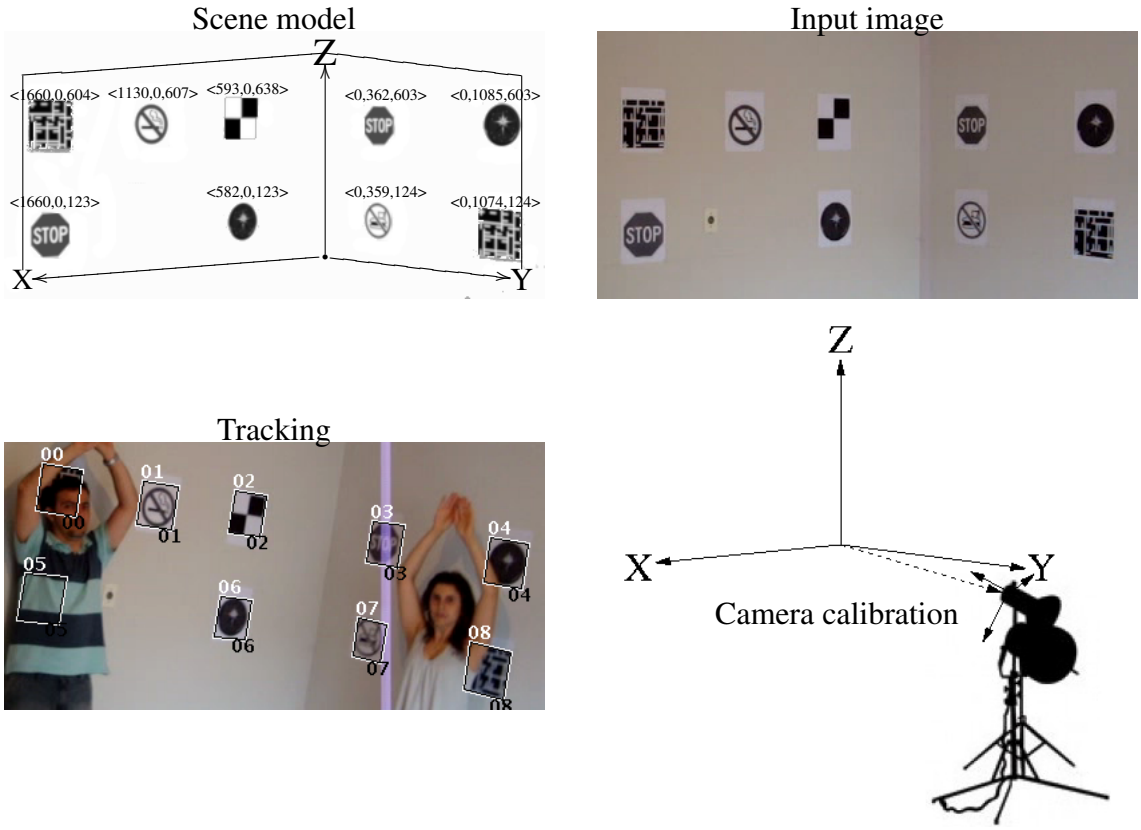


Figure 1.5: Feature-based rigid object tracking for camera calibration.

1.4 Publications

The publications directly related to this thesis (in chronological order), are:

- *Integrating Tsai's Camera Calibration Algorithm with KLT Feature Tracking*.
  R. Minetto, N.J. Leite and J. Stolfi.
  Proceedings of IV Workshop de Visão Computacional. 2008.

- AFFTRACK*: Robust Tracking of Features in Variable-Zoom Videos*.
  R. Minetto, N.J. Leite and J. Stolfi.
  IEEE International Conference on Image Processing (ICIP). 2009.

- SNOOPERTEXT*: A Multiresolution System for Text Detection in Complex Visual Scenes*.
  R. Minetto, N. Thome, M. Cord, J. Fabrizio and B. Marcotegui.
  IEEE International Conference on Image Processing (ICIP). 2010.

- SNOOPERTRACK*: Text Detection and Tracking for Outdoor Videos*.
  R. Minetto, N. Thome, M. Cord, N.J. Leite and J. Stolfi.
  IEEE International Conference on Image Processing (ICIP). 2011.

- *Text Detection and Recognition in Urban Scenes*.
  R. Minetto, N. Thome, M. Cord, J. Stolfi, F. Precioso, J. Guyomard and N. J. Leite.
  IEEE/ISPRS Workshop on Computer Vision for Remote Sensing of the Environment CVRS-ICCV. 2011.

Part I

Text Recognition

Chapter 2

Introduction

In this part of the thesis we consider the *text/non-text classification problem*. The input data for this problem is a specific sub-image of a digital photo or video frame of an arbitrary 3D scene. The output is a binary decision that ideally should be 'TRUE' if the sub-image contains a text in Roman-like characters and 'FALSE' otherwise. Algorithms that solve this problem are important components of other image processing software, such as text detectors and text trackers.

2.1 Motivation

Text detection of arbitrary text (such as store names, traffic signs, and indoor sign) in photographs of unstructured 3D scenes is a challenging task in computer vision, with many potential applications such as traffic monitoring, geographic information systems, road navigation, and scene understanding. However, no efficient solution yet exists. Most Optical Character Recognition (OCR) algorithms are designed for scanned documents, and perform very poorly on photos of 3D scenes [68]. The reasons include extreme text size and font variations, tilted or curved baselines, strong background clutter and difficult illumination conditions. Much better results are obtained by applying a standard OCR algorithm to the output of a generic text detection algorithm [68].

Text detection has been extensively researched in recent years, but most systems described in the literature are dedicated to specific contexts, such as automatic location of license plates [8] or postal addresses in packages [71], or OCR for scanned documents. Relatively few systems consider the extraction of arbitrary text objects in images of outdoor scenes [27, 35, 77, 57, 91]. Only recently, some benchmarks composed only by outdoor images are publicly available for this purpose [1, 25].

2.2 Our solution

Our major contribution in this part is the use of the *histogram of oriented gradients* (HOG) [24] for the text/non-text classification problem. HOG-based classifiers have been used for the recognition of pedestrians [24], solid objects [94], and for text detection [34, 72, 87, 93]. Here we describe a novel text classifier, *Text HOG* (T-HOG) [68], that efficiently and accurately characterizes single-line texts. We expect that the T-HOG descriptor will be useful for text detection, text tracking, OCR, and related applications. In particular, we show that the combination of a "permissive" multi-scale text detector SNOOPERTEXT [66] (described in Section 6.1.1) with a T-HOG based post-filter outperforms state-of-the-art text detectors described in the literature [27]. We also show how the T-HOG could be used by itself in a sliding-window text detector, and as a component of a text tracking algorithm.

Figure 2.1: Image of a urban scene with text objects.

The T-HOG algorithm receives an axis-aligned rectangular sub-image of a digital image and extracts from it a vector of real attributes, *a descriptor* that can be fed to any vector classifier, such as the *support vector machine* (SVM) [21]. The T-HOG descriptor is based on the observation of Chen and Yuille [20] that different parts of Roman-like text have distinctive distributions of edge directions. Dalal and Triggs [24] showed that an effective way of capturing the spatial distribution of gradient orientations for human recognition is to divide the sub-image into a rectangular grid of cells, compute a HOG for each cell, and concatenate those HOGs. They used the term *R-HOG* (Rectangular HOG) for this composite descriptor.

The T-HOG descriptor is an improvement of R-HOG, optimized for the specific task of text line classification. It differs from R-HOG by using modified gradient evaluation and normalization methods, as well as a specific cell layout with blurred boundaries. We determine experimentally the optimal cell tiling for text line classification, which turns out to be a division of the candidate sub-image into horizontal stripes.

The T-HOG and R-HOG descriptors have several parameters that can be tuned to trade accuracy for descriptor length. (Smaller descriptors are interesting, even if less accurate, because

they are more computationally efficient and may help us to identify the aspects of the image that are most relevant for text/non-text discrimination). Finally, we also compare the performances of both classifiers experimentally for a wide range of parameter settings.

2.3 Statement of the problem

2.3.1 Images and pixels

For this thesis, a *digital image* is a 2D array of color values (*pixels*). For monochromatic images, we assume that each pixel is a real number between 0 and 1. For color images, a pixel consists of a separate measurement for each *color channel* (spectral band), e.g. the NTSC standard red (R), green (G) and blue (B) channels. Usually, pixel values are the result of complicated color space transformations and filtering, such as de-Bayering [12]. We assume these corrections have all been applied to the input images.

It is sometimes convenient to convert RGB color images to monochromatic ones. For this purpose we use the CIE 601-1 luminance formula $0.299\ R + 0.587\ G + 0.114\ B$.

The *domain* of an image with n columns and m rows is the axis-aligned rectangle with upper left corner at $(0, 0)$ and lower right corner at (n, m). The position of a point in the image domain can be specified by its *frame* (*or image*) *coordinates* $(x_\mathrm{f}, y_\mathrm{f})$ relative to the upper left corner of the image, with the Y axis pointing down. This assumption implies that the pixel in column j and row i (indexed from 0) can be viewed as a unit square with opposite corners, at coordinates (j, i) and $(j + 1, i + 1)$, and center at $(j + 0.5, i + 0.5)$.

2.3.2 Text objects

For this part of the thesis, a *text object* is any part of the physical scene carrying a string of two or more letters that are easily recognizable and readable in the captured image. See Figure 2.1. We are primarily concerned with texts written in the Roman alphabet or any of its variants, in plain print-style (rather than cursive or ornate) fonts, with fairly "normal" geometry (straight baseline, on smooth surfaces that are flat or moderately curved). We exclude isolated characters since many items of real scenes are quite similar to certain Roman letters like 'I', 'O', *etc*.

We assume that the text consists of a single multi-character line, which can be tightly enclosed by a rectangle in 3D space with a narrow margin all around, as in Figures 2.2(b,c). Consecutive single character regions and multi-line text blocks, as in Figures 2.2(a,d) should be joined or split into separate lines or words, respectively.
Finally, our method is designed for texts that contain a normal mix of characters. It may perform poorly for texts that consist of similar simple letters, such as III.

(a) (b) (c) (d)

Figure 2.2: Alternative segmentations of text objects: (a) characters, (b) words, (c) lines and (d) blocks.

2.3.3 Text regions

We assume that the projection of a text object covers only a small portion of the image's domain. Therefore we can assume that the projection can be approximated by a parallelogram r, which can be completely defined by its center $p_\mathrm{f} \in \mathbb{R}^2$ and two vectors $u_\mathrm{f}, v_\mathrm{f} \in \mathbb{R}^2$ such that $p_\mathrm{f} \pm u_\mathrm{f} \pm v_\mathrm{f}$ are the corners of r. See Figure 2.3 and 2.4(a).

Figure 2.3: The geometry of a text region.

Note that a text is legible in the image only if the angle θ between its normal direction in space and the direction of view is not too large. Moreover it is difficult to distinguish an upright font in oblique projection from a slanted font in a orthogonal projection. For these reasons, we may as well assume that $\theta = 0$, and therefore that the vectors u_f and v_f are perpendicular; meaning that the parallelogram $p_\mathrm{f} \pm u_\mathrm{f} \pm v_\mathrm{f}$ is a rectangle. See Figure 2.4(b).

Moreover, in some applications the vector u_s is known to be parallel to the camera's horizontal axis. In that case, we may also assume that the projected rectangle is aligned with the image axes, i.e. $u_\mathrm{f} = (|u_\mathrm{f}|, 0)$ and $v_\mathrm{f} = (0, |v_\mathrm{f}|)$. Indeed, for computational expediency, one often makes this assumption even in situations which do not quite satisfy this condition. See Figure 2.4 (c).

(a) (b) (c)

Figure 2.4: Modeling projected text line objects: by parallelograms (a), general rectangles (b) and axis aligned rectangles (c).

Chapter 3

Related work

There is an extensive literature on text *detection*. Recently, some benchmarks [1, 3, 25] and challenges [55] were organized to give a clear understanding of the current state-of-the-art of such algorithms. Basically, the algorithms can be classified into top-down and bottom-up approaches. Top-down strategies [57] generally exhaustively sample all possible sub-regions in an image, using a sliding window mechanism, looking first for text regions and then for characters. Bottom-up strategies [19, 57, 27] first identify sub-structures that make up text letters, as connected components or edges and merge these sub-structures to create a text region. However, an exhaustive review of these algorithms is far outside the scope of this thesis, and the reader can refer to the survey of Jung *et al*. [39], that covers some advances in this area.

Comparatively little has been published about text/non-text *classification* algorithms (our primary interest in this thesis), although they are often present as components of text detectors. This problem is often cast as a texture classification problem, and several texture descriptors have been considered in the literature. For instance, in 2004, Kim *et al*. [41] described a text recognizer that decomposes the candidate sub-image into a multi-scale 16×16 cell grid and compute wavelet moments for each block. Then each block is classified as text or not using an SVM. The ratio of text to non-text outcomes is used to decide if the entire sub-region is text or non-text. In 2005, Ye *et al*. [90] described a similar text recognizer with multi-scale wavelet decomposition but they used more elaborate features including moments, energy, entropy, *etc*. In 2004, Chen and Yuille [20] proposed a descriptor that combines several features, including 2D histograms of image intensity and gradient, computed separately for the top, middle and bottom of the text region, as well as for more complex slices subdivisions of the image– 89 features in total. Recently, some text detectors, such as the one described by Anthimopoulos *et al*. in 2010, have used descriptors based on multi-scale *local binary patterns* (LBP) introduced by Ojala *et al*. [70]. Their descriptor has 256 features.

The use of gradient orientation histograms (HOGs) as texture descriptors was introduced by Dalal and Triggs in 2005 [24], for human recognition. HOG descriptors are used in some

recent text recognizers, such as the one proposed in 2008 by Pan *et al*. [72]. They partition the candidate sub-image into 14 cells, as proposed by Chen and Yuille, but compute for each cell a 4-bin HOG complemented by a 2×3 array of LBP features. Their complete descriptor has 140 features.

Other HOG-based text recognizers have been proposed in 2009 by Hanif and Prevost [34] for single-line text, and Wang *et al*. [87] for isolated Chinese and Roman characters as well as single-line text. Hanif and Prevost's descriptor has 151 features (16 cells each with a 8-bin HOG, supplemented by 7 mean difference and 6 standard deviation features over 16 cells). The descriptor of Wang *et al*. has 80 features (8 cells with 8-bin HOG and 1 mean difference feature and 1 standard deviation).

These HOG-based text recognizers, and several others, use vertical cuts as well as horizontal ones when partitioning the candidate region. The use of vertical cuts was apparently borrowed from the Dalal and Triggs paper [24] on pedestrian recognition. They may be justifiable for isolated characters, but they do not appear to be useful for multi-character texts of variable width. In such texts, the gradient distribution is largely independent of horizontal position; therefore, a cell layout with vertical cuts increases the size of the descriptor without providing any additional relevant information.

Chapter 4

The T-HOG descriptor

4.1 Descriptor outline

Dalal and Triggs observed that a particular texture can often be characterized by the distribution of the directions of the image gradient. If the texture consists of simple bi-level shapes (such as Roman letters) then the orientations of the strongest gradients tell the orientations of the edges of those shapes.

To capture the spatial variation of edge orientations, Dalal and Triggs divided the input sub-image into a rectangular grid of (possibly overlapping) cells with n_x columns and n_y rows, which they grouped into 2×2 blocks. Within each cell of each block, they computed a histogram of the gradient directions (HOG), with n_b bins. In these histograms the gradient direction of each pixel is weighted by the gradient's magnitude and by a Gaussian *block weight mask*. Their complete descriptor (R-HOG) is a vector with $n_x n_y n_b$ features that is the concatenation of these $n_x n_y$ HOGs. (Note that up to four overlapping or coincident cells may cover the same set of pixels, and each will generate a separate HOG, with different block weight functions.) To reduce the effects of local contrast and brightness variations, the HOGs in each block are normalized in a specific way.

Our T-HOG descriptor differs from the original R-HOG in some key details. First, we use different methods to extract the candidate text region, to normalize it for size and contrast, and to compute its gradient image. Secondly, the cell grid is simplified to a partition into horizontal stripes (*i.e* we fix $n_x = 1$). Instead of overlapping blocks and block weight functions, in the T-HOG the cells are defined by overlapping *cell weight functions*. As a result, all internal cell boundaries are blurred, unlike those of the R-HOG. See Figure 4.1. As detailed in chapter 5, these changes significantly improved the discriminating power for our target objects—single-line text regions of arbitrary length.

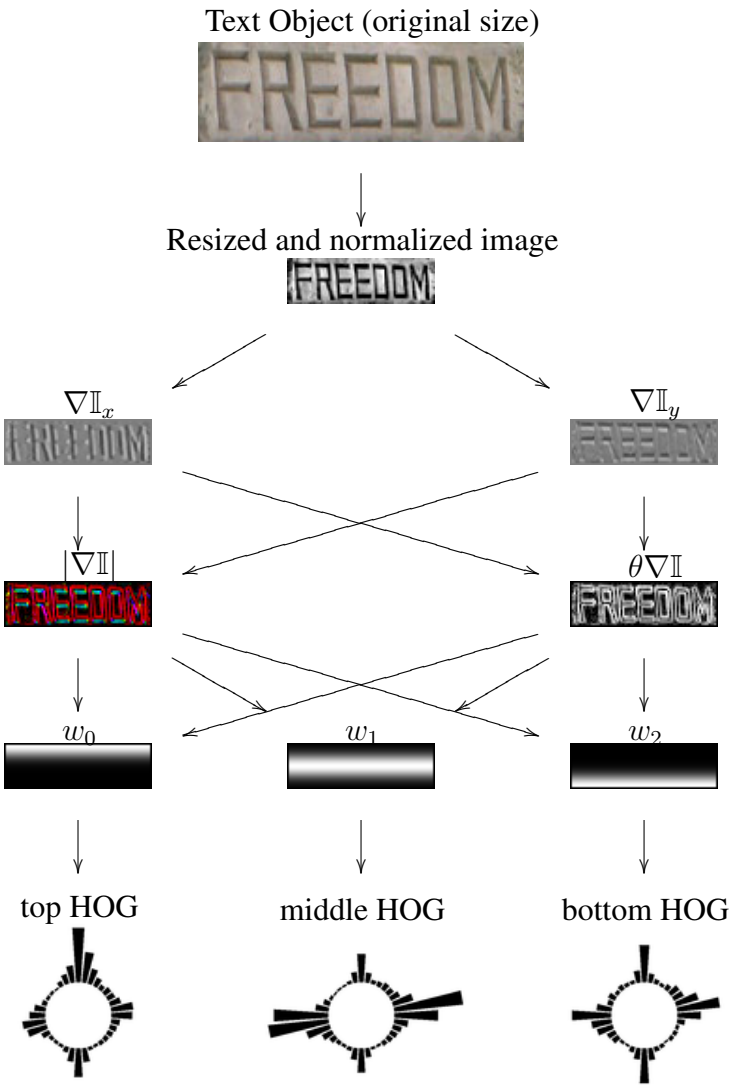


Figure 4.1: Illustration of the T-HOG text descriptor computation, for $n_x = 1$, $n_y = 3$, and $n_b = 24$.

4.2 Size and contrast normalization

The first step of the T-HOG algorithm is to extract the sub-image and scale it to a fixed height H, maintaining its original aspect ratio. Since our method is based on analysis of character edge orientations, the extracted sub-image must include at least one pixel of background all around the text itself. The height H should be large enough for the characters to remain readable, but small enough to eliminate most of the noise and other spurious details. For print-style Roman characters (upper and lower case), we obtained the best results with H between 20 and 25 pixels.

In this step, we also convert the image from color to gray scale, since the human eye-brain uses almost exclusively the brightness channel to recognize character shapes [29]. We observed that objects in urban contexts are often obscured by non-uniform illumination and localized shadows or reflections. To remove these artifacts, we apply to each sample V of the extracted sub-image a contrast normalization procedure $V \leftarrow 0.5 + (V - \mu)/(3\sigma)$, where μ and σ are the local mean and standard deviation, computed with a doubly binomial weight window of width $2H + 1$. (The raw deviation σ is adjusted by $\sigma \leftarrow \sqrt{\sigma^2 + \varepsilon^2}$, where ε is the assumed standard deviation of the image sampling noise.)

4.3 The basic HOG descriptor

By definition, the HOG descriptor of an arbitrary image I is a histogram of the gradient direction $\theta(\nabla\mathbb{I})$, computed at each pixel, quantized into a small number n_b of bins. Each pixel gradient is added to the histogram with "mass" proportional to the gradient magnitude $\rho(\nabla\mathbb{I})$, so as to de-emphasize the random noise-related gradient directions that occur in flat parts of the image. As observed by Dalal and Triggs, if $\theta(\nabla\mathbb{I})$ does not fall at the exact center of a bin, the mass should be distributed between the two nearest bins by a linear splitting criterion. To compute the gradient $\nabla\mathbb{I}$, we use the simple difference schema recommended by Dalal and Triggs, namely

$$\nabla\mathbb{I}(x, y) = \frac{1}{2}(\mathbb{I}(x + 1, y) - \mathbb{I}(x - 1, y),\ \mathbb{I}(x, y + 1) - \mathbb{I}(x, y - 1))$$

For this formula, any non-existing pixel (outside the input sub-image) is assumed to be equal to the nearest existing pixel. Note that we compute the gradient after grayscale conversion and contrast normalization; whereas Dalal and Triggs compute the gradient in each color channel and then pick the vector that has largest norm. The magnitude of the gradient is then estimated by the formula

$$\rho(\nabla\mathbb{I})(x, y) = \sqrt{\max\{0, |\nabla\mathbb{I}(x, y)| - \varepsilon^2\}}$$

Note that this formula is zero if the raw gradient norm $|\nabla\mathbb{I}|$ is smaller than the assumed sampling noise deviation ε.

The gradient direction $\theta(\nabla\mathbb{I})$ is expressed as an angle in the range $[0, 2\pi]$ radians. (Dalal and Triggs found that recognition of some classes of objects (such as humans) was improved when opposite directions were considered equivalent [24], in which case the range of $\theta(\nabla\mathbb{I})$ is $[0, \pi]$ radians. We found that this is not the case for text.)

Figure 4.2 shows the HOGs of a few isolated letters. These HOGs have 16 bins, each $2\pi/16$ radians wide, centered at orientations $2k\pi/16$ for $k = 0, 1, \ldots, 15$. It can be seen that the HOG gives the predominant orientation of the letter strokes. For example, the histogram of a rounded letter like 'O' is almost uniform over the whole range $[0, \pi]$, while that of 'I' has significant spikes at the directions perpendicular to the letter's stem.

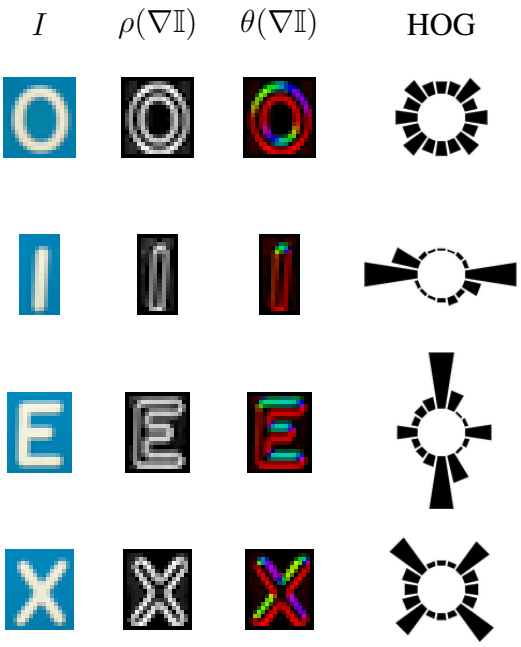


Figure 4.2: HOGs of some isolated letters.

4.4 Multi-cell HOGs

Images of complex objects typically have different HOGs in different parts. Images of humans, for example have different gradient orientation distributions in the head, torso and leg regions. It was this observation that motivated Dalal and Triggs to use a multi-cell HOG (R-HOG) for that application.

This observation is true also for text images. Figure 4.3 shows the distributions of edge directions in the top, middle and bottom parts of an image containing a single-line of text. Note

that the gradient orientations are predominantly 0 (or 180) and 90 (or 270) degrees, reflecting the predominance of vertical and horizontal strokes. Note also that the top and bottom parts contain a larger proportion of horizontal strokes, so that the gradients there are mostly vertical. The middle part, on the other hand, contains a larger proportion of vertical strokes and hence of horizontal gradients. In all three regions there is a small amount of diagonal strokes, due to letters such as 'R' and 'M'; and to the rounded parts of letters such as 'R', 'D', and 'O'. Finally, note that opposite directions tend to be equally represented, due to the fact that the two edges of a letter stroke have opposite gradients.

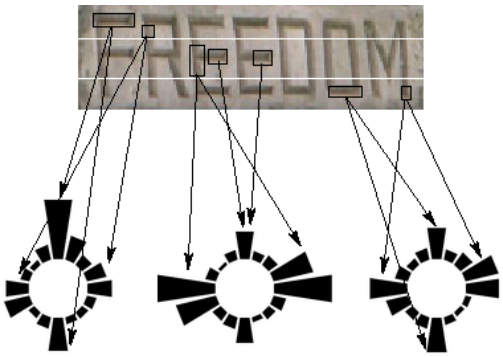


Figure 4.3: From left to right, the 16-bin HOG descriptors of the top, middle and bottom parts of a text sub-image. The arrows indicate the contribution of specific letter strokes to the histogram.

For comparison, Figure 4.4 shows the HOG descriptors of top, middle and bottom regions of some non-text images. Note that several of these HOGs are quite distinct from those of Figure 4.3, and some are significantly unbalanced.

On the other hand, for an image containing an arbitrary single-line, multi-character text, the expected distribution of gradient orientations is largely independent of the horizontal position along the line, as long as the segment analyzed is wide enough to include one whole character. This intuition was confirmed by extensive experimental tests; see Section 5.5.

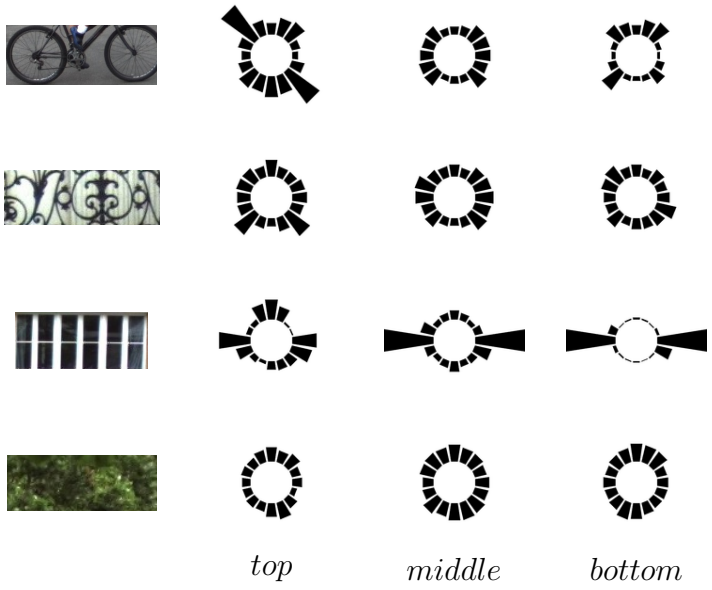


Figure 4.4: Top, middle and bottom 16-bin HOG descriptors of some non-text images.

4.5 Cell weights

If the cells were defined by sharp boundaries, their HOGs would change drastically with small displacements of the text inside the candidate sub-image, as letter strokes would shift from one cell to the next. See Figure 4.5. To reduce this problem, the T-HOG cells are defined by smooth *cell weight functions*. This choice made the T-HOG more robust to such problems. See Figure 4.6.

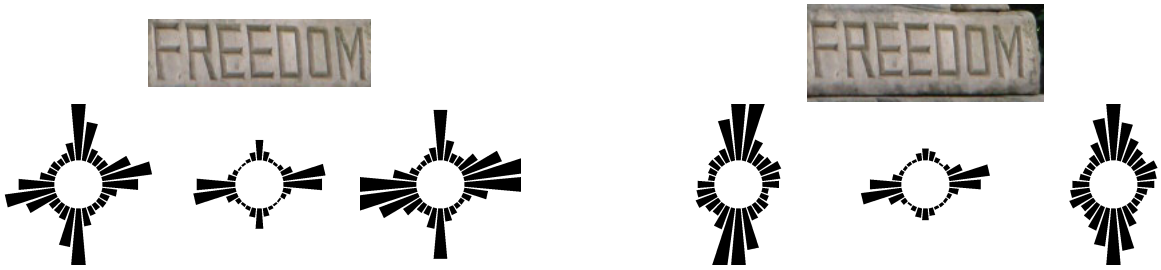


Figure 4.5: Effect of sharp cell boundaries. Two different cropped sub-images of the same text object. The HOG descriptors of the top, middle and bottom parts of each sub-image using sharp cell boundaries.

Namely, let $x_{\min}$, $x_{\max}$, $y_{\min}$, and $y_{\max}$ be the minimum and maximum pixel coordinates

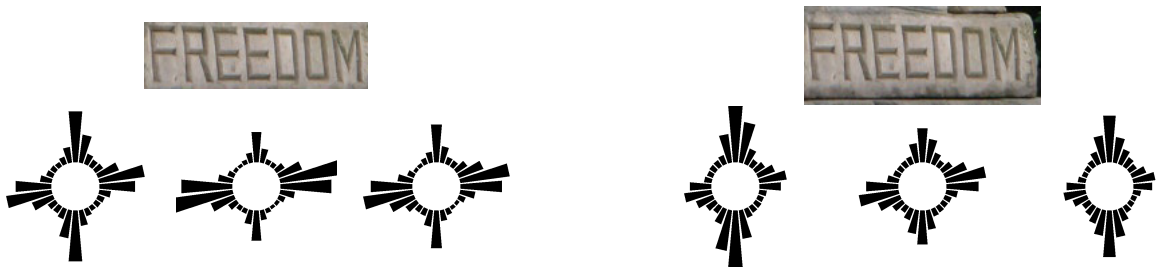


Figure 4.6: Effect of smooth cell boundaries. Two different cropped sub-images of the same text object. The HOG descriptors of the top, middle and bottom parts of each sub-image using smooth cell boundaries.

in the sub-image. For each pixel with center coordinates (x, y), we define the *relative pixel coordinates*

$$X(x) = \frac{x - x_{\min}}{x_{\max} - x_{\min}} \qquad Y(y) = \frac{y - y_{\min}}{y_{\max} - y_{\min}} \tag{4.1}$$

The weight of that pixel relative to a cell C_{ij} in column i and row j of the cell grid is then defined as $w_{ij}(x, y) = u_i(X(x))v_j(Y(y))$, where each function u_i or v_j is 1 at the nominal axis the respective column or row, and falls smoothly to 0 as one moves away from it. The gradient of that pixel contributes to the histogram of cell C_{ij} with mass $\rho(\nabla I)(x, y)w_{ij}(x, y)$, rather than just $\rho(\nabla I)(x, y)$.

4.5.1 Gaussian cell weights

For the one-dimensional weights u_i and v_j, we tested different families of functions (Gaussian bells, Hann windows, Bernstein polynomials, *etc*). In these experiments, the best results were obtained with Gaussian bell functions. Specifically, for $n_y \geq 2$ rows of cells, the vertical weight function of cells in row j is

$$v_j(Y) = \gamma\left(Y,\ -\mu_0 + \frac{1 + 2\mu_0}{(n_y - 1)j},\ \frac{\sigma_0}{n_y}\right) \tag{4.2}$$

where $\mu_0 = 0.01$, $\sigma_0 = 0.5$, and

$$\gamma(z, \mu, \sigma) = \exp\left(-\frac{(z - \mu)^2}{2\sigma^2}\right) \tag{4.3}$$

Figure 4.7 show these weights for $n_y = 3$. As a special case, if $n_y = 1$, the single vertical weight function v_0 is equal to 1 everywhere. Note that the top edges of the topmost cells and the bottom edges of the bottommost cells are still sharp. The horizontal weight functions u_i are defined in the same way.

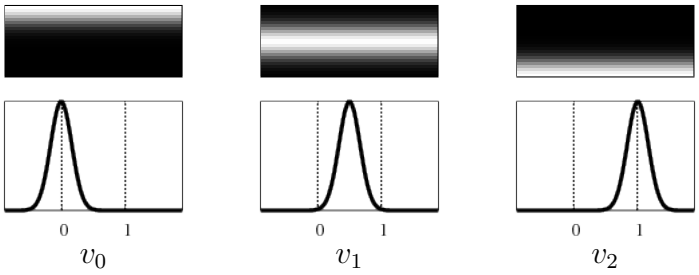


Figure 4.7: The T-HOG vertical cell weight functions v_0, v_1, and v_2 for $n_y = 3$, as a function of the relative coordinate Y(y).

4.5.2 Emulating cells with hard edges

Hard-edged cells can be emulated in the T-HOG by defining each function u_i or v_j to be the appropriate step function. See Figure 4.8.

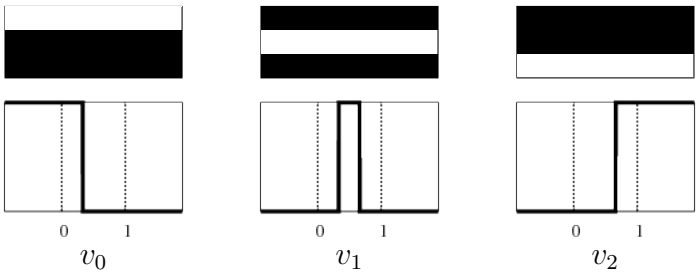


Figure 4.8: Step weight functions v_0, v_1, and v_2 used to emulate hard-edged cells in the T-HOG model.

4.5.3 Relation to R-HOG weight functions

Dalal and Triggs also used Gaussian weight functions, but in a different and more limited way. For one thing, their weight functions were associated to cell blocks (usually containing 2×2 cells) rather than individual cells. With the parameters they used for human recognition, the internal cell boundaries in each block are sharp, while the edges of the sub-image itself fade gradually to zero.

Figure 4.9 shows the effective R-HOG cell weight functions for the best parameter configuration we found using $n_y = 3$ cells: namely, a single block divided into $1 \times n_y$ cells, with a fairly broad block weight function ($\sigma_x = W/2$, $\sigma_y = H/2$, corresponding to setting the `wtscale` parameter to 1 in their implementation). With these parameters, the effective cell weight functions have quite sharp boundaries, as shown in Figure 4.9. Figure 4.10 shows the R-HOG cell weights for the same parameters, but with narrower block weight functions recommended by

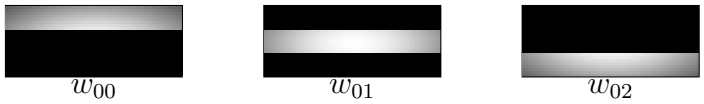


Figure 4.9: The Dalal and Triggs's cell weight functions for a single block of 1×3 cells, with the optimal block weight deviations $\sigma_x = W/2$, $\sigma_y = H/2$.

Dalal and Triggs for human recognition ($\sigma_x = W/4$, $\sigma_y = H/4$, corresponding to the default `wtscale` $= 2$). One can obtain R-HOG weights somewhat similar to the T-HOG weights of

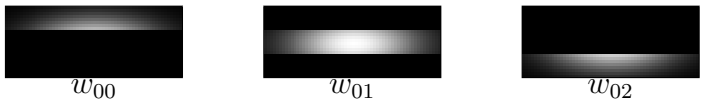


Figure 4.10: The Dalal and Triggs's cell weight functions for a single block of 1×3 cells, with the default block weight deviations $\sigma_x = W/4$, $\sigma_y = H/4$.

Figure 4.7 by using $1 \times n_y$ overlapping blocks with one cell per block, as shown in Figure 4.11. Comparing the cell weights of Figures 4.7 and 4.11, we observe that the latter assign much

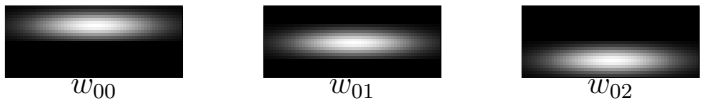


Figure 4.11: The Dalal and Triggs's cell weight functions for 1×3 single-cell blocks, each with height $H/2$ and overlapped with stride $H/4$, with the default block weight deviations $\sigma_x = W/4$, $\sigma_y = H/8$.

lower mass to pixels along the edges of the sub-image (among other differences). Presumably for that reason, the R-HOG classifiers with the weights of Figures 4.10 and 4.11 were less accurate than R-HOG with the weights of Figure 4.9, for the same descriptor size; and all three were worse than T-HOG.

4.6 Normalization

Both algorithms, R-HOG and T-HOG, normalize the resulting descriptor. Dalal and Triggs use a per-block normalization scheme, which is intended to compensate for spatial variations of lighting and contrast over the input image. Since the T-HOG algorithm removes those effects beforehand (see Section 4.2), we simply divide the final descriptor by the sum of all features plus a constant ϵ (L_1 norm).

4.7 Vector classification and thresholding

Like Dalal and Triggs, we use an SVM classifier [21] to turn the descriptor $z \in \mathbb{R}^N$ into a real-valued score $f(z)$, such that positive scores mean 'probably text' and negative scores mean 'probably non-text'. The SVM classifier is defined as

$$f(z) = \sum_{i=1}^{M} \alpha_i K(z_i, z) - b \tag{4.4}$$

where K is the *kernel*, a function from $\mathbb{R}^N \times \mathbb{R}^N$ to $\mathbb{R}$; the z_i are the M fixed *support vectors*; the α_i are real weights; and b is the *bias* or *decision threshold*. Typically the support vectors and weights are determined by a *training* procedure, from given representative samples of text and non-text descriptors.

Chapter 5

Experiments

In this chapter, we describe an extensive set of experiments that we performed in order to determine optimum values for the various parameters of the R-HOG and T-HOG descriptors, and to compare their performance in the basic text/non-text discrimination task. These experiments strongly confirm the advantage of the two main T-HOG innovations, namely the splitting into overlapping horizontal cells (Section 4.4) with blurred boundaries (Section 4.5).

5.1 Image collections

In our tests, we used single-line text samples derived from three image collections:

1. The 2005 ICDAR challenge collection [55], consisting of 499 color images, captured with different digital cameras and resolutions, of book covers, road signs, household objects, posters, *etc*.

2. A subset of the iTowns Project collection [26], consisting of a hundred 1080×1920 color images of Parisian façades taken by a camera-equipped vehicle (similar to Google's Street View images).

3. The Epshtein *et al.* benchmark [27] with 307 color images of urban scenes, ranging from 1024×1360 to 1024×768 pixels, taken with hand-held cameras.

These image collections are suitable benchmarks for text *detectors*, but not for text *classifiers*. Therefore, we extracted from these image collections six sets of candidate sub-images, as follows. We processed each image collection, with SnooperText [66], a state-of-the-art text detector algorithm, tuned for high recall and moderate precision. By visual inspection, we separated the candidate regions returned by SnooperText into a set of text regions X_i, and a set of non-text ('background') regions B_i, for $i = 1, 2, 3$. See Figure 5.1. Table 5.1 gives the number of

sub-images in each set. (For succinctness, we will often omit the index i in the remainder of the chapter.)

i	Image Set	Detected regions	Text $\#X_i$	Non-Text $\#B_i$
1	ICDAR	4961	1727	3234
2	iTowns	2242	714	1528
3	Epshtein	7518	1502	6016

Table 5.1: Text and non-text samples X_i, B_i used in our tests.

Figure 5.1: Samples of text regions (set X_i) and non-text regions (set B_i) extracted by SnooperText from the ICDAR, iTowns and Epshtein image collections.

5.2 Error rate metrics

To quantify the performance of a binary classifier (R-HOG or T-HOG) with a specific set of parameters, we adopted a 'ranking-based' approach. That is, we evaluate the ability of the classifier to score text regions higher than non-text regions, regardless of the absolute value of the SVM score $f(z)$.

Specifically, in our tests we randomly divided the set X (respectively B) into two disjoint sets, each one with 50% of the elements: a 'training' half X' (respectively B') and a 'testing' half X'' (respectively B''). The sets X', B' were used to train the SVM. We then applied the classifier to the complementary sets X'', B''. For several values of the SVM threshold b (see

Section 4.7), we computed the counts TP_b, TN_b (correct decisions, positive and negative) and FP_b, FN_b (incorrect decisions). From these counts we computed the classification success rates for the text and non-text regions on each evaluation dataset, namely

$$\tau_b = \frac{\text{FN}_b}{\#X''} = \frac{\text{FN}_b}{\text{TP}_b + \text{FN}_b} \quad \beta_b = \frac{\text{FP}_b}{\#B''} = \frac{\text{FP}_b}{\text{TN}_b + \text{FP}_b} \tag{5.1}$$

The τ_b metric (*false negative rate*) is the complement of the well-known *recall* metric r; it is the probability of our algorithm incorrectly rejecting a text-containing region. The β_b metric (*false positive rate*) is the probability of incorrectly accepting a non-text region. We choose to use β_b instead of the common *precision* metric because the latter depends strongly on the ratio $\#B''/\#X''$, which is essentially arbitrary.

By adjusting the threshold b, the user can trade one class of errors for the other. In particular, when b is sufficiently small, the classifier accepts all samples, so that $\tau_b = 1$ and $\beta_b = 0$. Conversely, when b is sufficiently large, all samples are rejected, so $\tau_b = 0$ and $\beta_b = 1$.

In order to reduce the sampling error, we repeated the whole procedure $L = 10$ times for each pair of datasets X, B resulting in L different random partitions X', X'' and B', B'' for each set. The raw statistics $\text{TP}_b, \text{TN}_b, \text{FP}_b, \text{FN}_b$ were averaged over these L runs, for each b.

5.3 DET curve and area metric

We compare classifiers by plotting the *decision error trade-off* (DET) *curve* [24, 58], the set of pairs (τ_b, β_b) for $b \in [-\infty, \ldots, +\infty]$. See Figure 5.2. For an ideal classifier, the DET curve lies along the bottom and left edges of the unit square $[0, 1] \times [0, 1]$. The better the classifier, the closer its DET curve should be to this ideal.

In our tests, we observed that, whenever a classifier C_i was significantly better than another classifier C_j for some threshold b, the same usually happened for most other values of b. In other words, the entire curve of C_i was closer to the ideal than that of C_j (below and to the left of it). Therefore, we can use the *decision error area* (DEA), the area A between the DET curve and the ideal curve (the shaded region in Figure 5.2), as a single scalar measure of the performance of a given classifier, independent of the threshold b. The value of A is a monotonically decreasing function of the classifier's accuracy, and is zero if the classifier is perfect (*i.e.*, if one can set the threshold b so that the classifier makes no mistakes). Therefore, we can compare two classifiers C_i and C_j by comparing the respective decision error areas A_i and A_j.

In order to determine whether the difference $A_i - A_j$ is statistically significant, we compute mean values $\boldsymbol{\mu}(A_i)$ and $\boldsymbol{\mu}(A_j)$ and the standard deviations $\boldsymbol{\sigma}(A_i)$ and $\boldsymbol{\sigma}(A_j)$ over the L runs. We then compute Student's test parameter $t(C_i, C_j)$

$$t(C_i, C_j) = \frac{\boldsymbol{\mu}(A_i) - \boldsymbol{\mu}(A_j)}{\sqrt{\frac{S^2}{L} + \frac{S^2}{L}}} \tag{5.2}$$

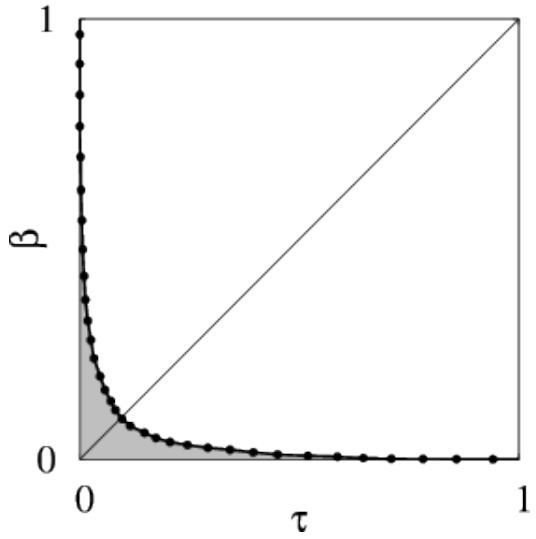


Figure 5.2: Area under curve A in gray.

where

$$S^2 = \frac{(L-1)\cdot\boldsymbol{\sigma}(A_i)^2 + (L-1)\cdot\boldsymbol{\sigma}(A_j)^2}{2L-2} \tag{5.3}$$

The performance variation between C_i and C_j is considered statistically significant at risk level α if $|t(C_i, C_j)|$ is above the corresponding threshold t_α from Student's table.

5.4 General parameter settings

In both the R-HOG and T-HOG algorithms, the sub-images X, B were rescaled during extraction with the Lanczos interpolation filter [84] to the chosen height H. Since the extracted height must be a multiple of the effective number of cell rows, we used $H = 25$ for 5 rows, $H = 21$ for 7 rows, and $H = 24$ for all other tests (with 1, 2, 3, 4, 6, 8 and 12 rows). The rescaled width W was chosen so as to maintain the aspect ratio of the original sub-image, but rounded to the nearest integer multiple of cell columns (which was 1 for most tests). For the mean-variance normalization and for gradient magnitude computation, we assumed a sampling noise with deviation $\varepsilon = 0.02$. In all tests we used a Gaussian χ^2 SVM kernel K, whose standard deviation parameter σ was optimized by cross-validation on the training sets X', B'.

In an extensive series of preliminary tests, we concluded that the best performance of the R-HOG as text classifier, for all three datasets, is achieved with L_1 block histogram normalization (see Figure 5.3), RGB color space with gamma correction 0.5 (RGB_SQRT), oriented gradient directions ranging over $[0, 2\pi]$ instead of $[0, \pi]$, and block mask parameter `wtscale` set to 1

($\sigma_x = W/2$, $\sigma_y = H/2$). In another series of tests, the best T-HOG performance was obtained with L_1 whole-descriptor normalization, and oriented gradient directions ranging over $[0, 2\pi]$. These optimal settings where then used for all subsequent tests.

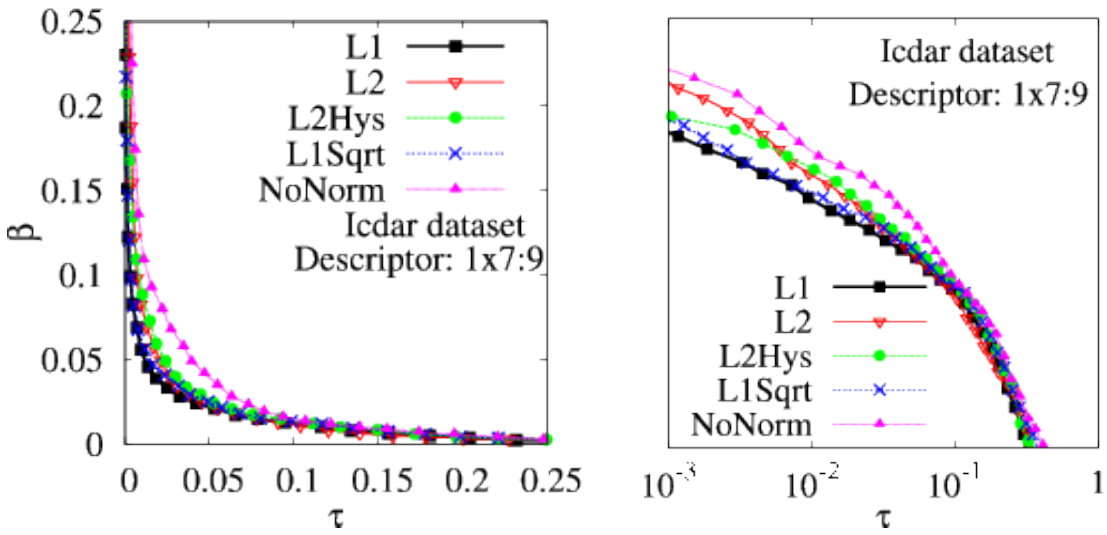


Figure 5.3: Left: DET curves of the R-HOG classifier with $n_x = 1, n_y = 7$ and $n_b = 9$, with various descriptor normalization methods. Right: Same plot in log-log scale.

5.5 Optimal cell arrangements

We next performed a series of tests to determine the optimum cell arrangement for text/non-text classification with the R-HOG algorithm, as a function of the total cell count $n_x n_y$. R-HOG allows the cells to be grouped into blocks, which may partially overlap. The possible arrangements with six cells (counting overlaps) are shown in Figure 5.4. Arrangements (a)–(d) have disjoint, non-overlapping cells, which could be grouped into disjoint blocks in several ways. Arrangements (e) and (h) have two cells per block; note that the two central cells are duplicated in the final descriptor. Arrangements (f) and (g) have single-cell blocks that overlap by half a cell.

We tested many possible cell and block arrangements, with and without overlapping blocks. The DET curves for some combinations of n_x and n_y with $n_b = 12$ are shown in Figure 5.5. Note that the counts n_x and n_y include overlapping cells, so that the descriptor always consists of $n_x n_y$ HOGs.

As mentioned in Section 4.5, we concluded from these experiments that arrangements with two or more blocks, overlapping or not, are not advantageous for R-HOG: for the same de-

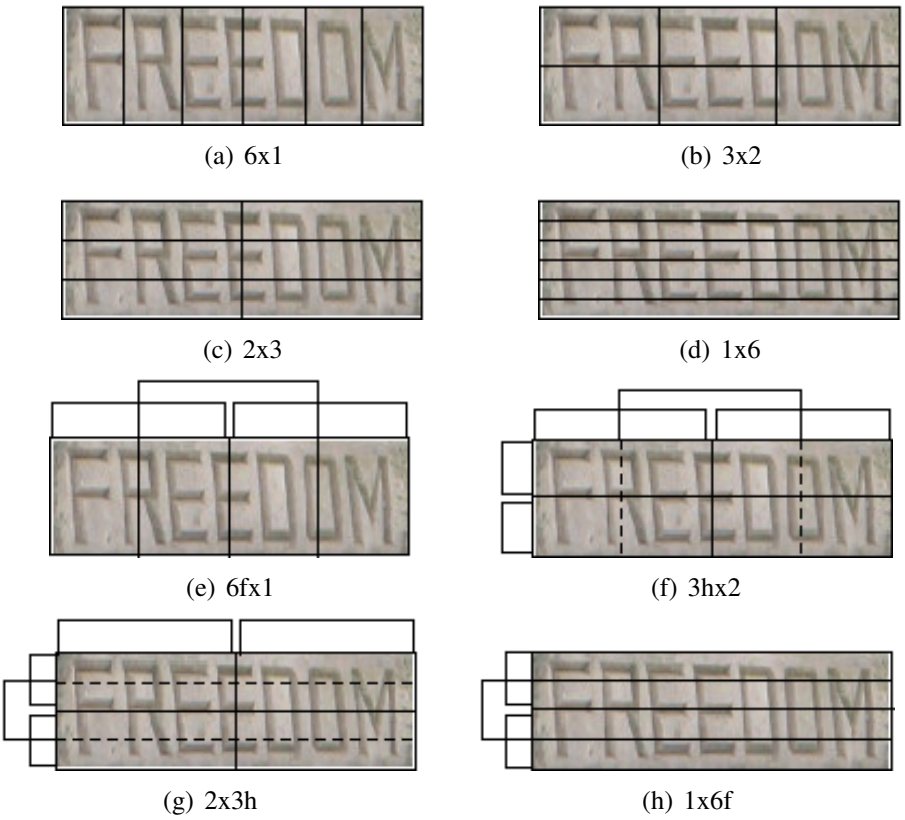


Figure 5.4: Some possible arrangements of blocks and cells that result in an R-HOG descriptor with six HOGs. Solid and dashed lines inside the image are cell boundaries; the external brackets show the blocks.

scriptor size $N = n_x n_y n_b$ and number of bins, a single block is always better. Moreover, we concluded that, for the same descriptor size, the best choice is always $n_x = 1$, that is, a grid of n_y horizontal stripes. These conclusions were confirmed by numerous tests with the other two datasets and with different bin counts ($n_b = 6, 12, 18$ and 36).

A parallel series of tests with our T-HOG classifier gave entirely similar results, confirming that $n_x = 1$ is always the best choice for any descriptor size.

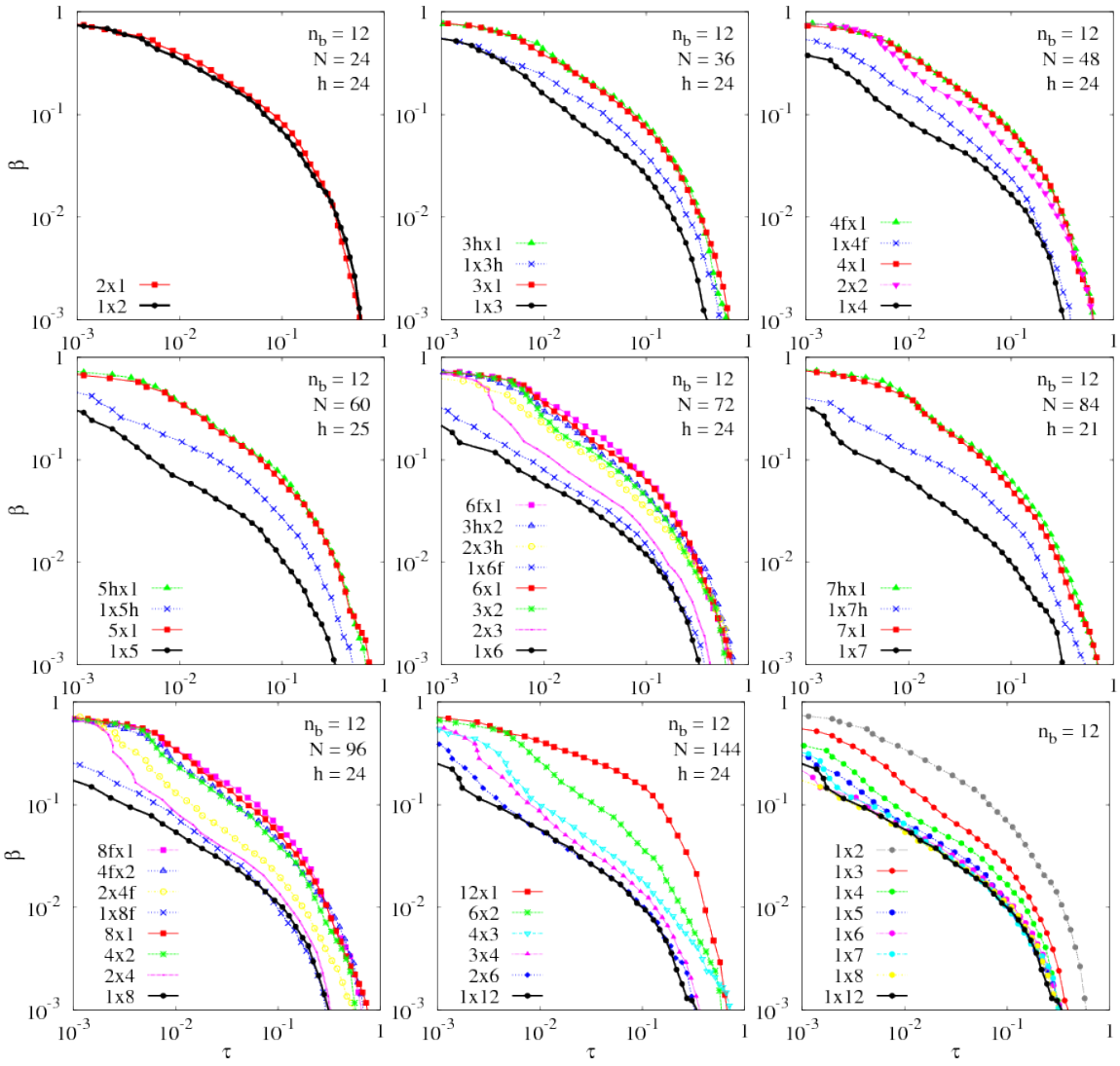


Figure 5.5: DET curves (mean of 10 random partitions) of the R-HOG classifier, for various cell and block arrangements, on the ICDAR-derived dataset X_1, B_1. In each plot, except the last one, all grid configuration give the same descriptor size $N = n_x n_y n_b$. The last plot compares the best combinations of the eight previous plots.

5.6 Performance as function of descriptor size

Having established that the best cell arrangement for R-HOG is always a single block divided into disjoint horizontal stripes, we performed another series of tests to analyze the influence of the number of stripes n_y and the number of bins per stripe n_b on the R-HOG classifier accuracy. Namely, we tested all combinations of $n_y = 1, 2, \ldots, 8, 12$ and $n_b = 4, 5, \ldots, 18, 24, 36$, with n_x fixed at 1. Figure 5.6 shows the results of these experiments for $N \leq 250$. Configurations are identified by the notation $n_x \times n_y : n_b$. From these tests, we concluded that a longer R-HOG descriptor generally gives better results, but the advantage is very small for N greater than 100. In particular, no improvement was seen when N increased beyond 250. We also concluded that R-HOG's accuracy improves dramatically as n_y increases from 1 to 3, improves more gradually until n_y is 7 or so, and is practically the same thereafter. These conclusions were found to hold for all three datasets.

In Figure 5.6 (bottom), the black dots represent the optimal combinations of n_y and n_b, the only ones that are worth using for any specified descriptor size N. Configurations that fall above the solid staircase line (blue dots) are fully dominated by optimal ones, in the sense that the latter provide equal or better performance with equal or smaller N. There appears to be no simple formula for the optimal parameters, partly because n_y and n_b are constrained to be divisors of N. Furthermore, the optimal configurations for the other two datasets are slightly different.

A similar series of tests was performed to determine the best combination of n_y and n_b for the T-HOG classifier. We found that the optimum combinations for each N were generally the same as those of R-HOG (see the next section).

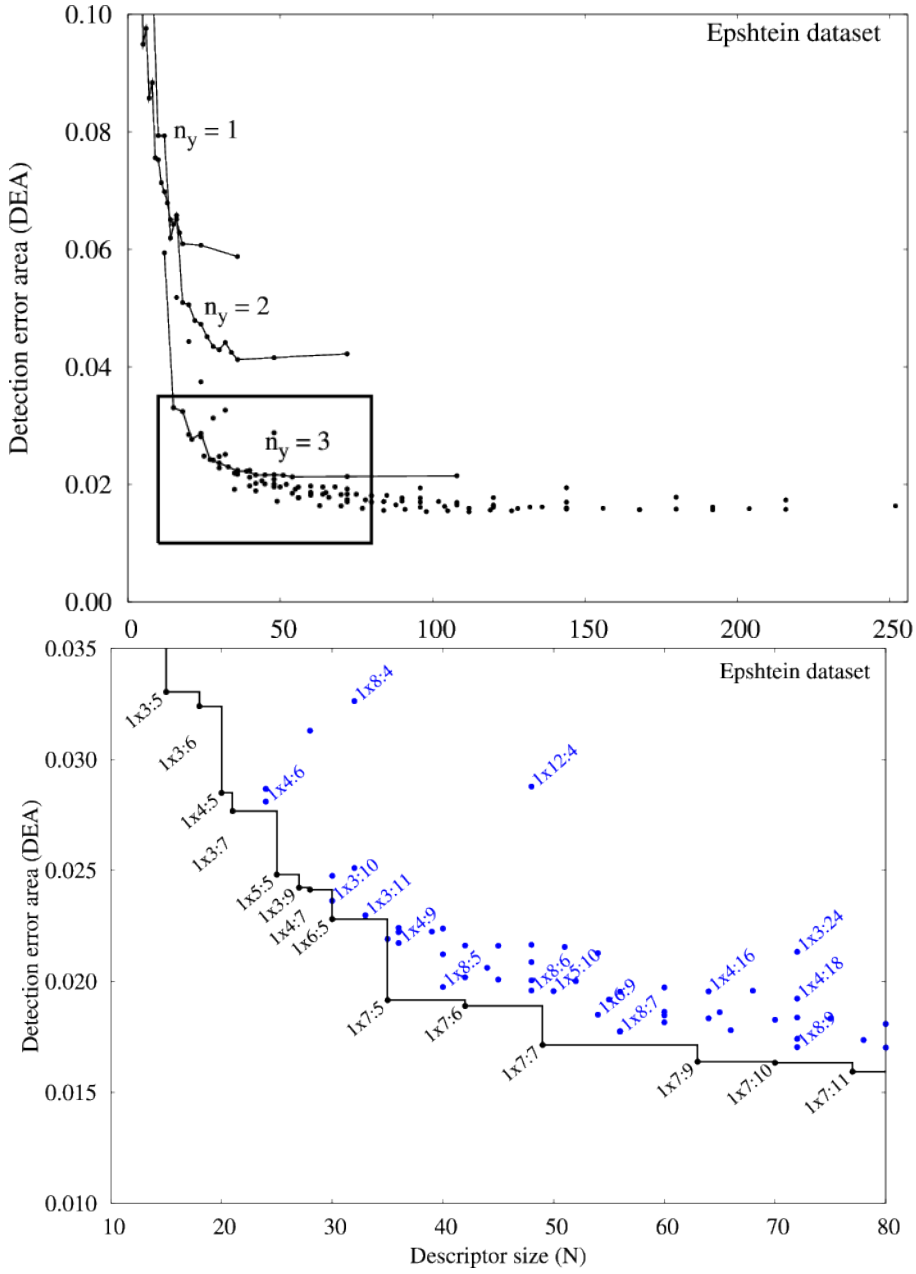


Figure 5.6: Detection error area A (mean of 10 random partitions) of R-HOG as a function of descriptor size $N = n_x n_y n_b$, for various combinations n_y and n_b, with $n_x = 1$. In the top plot, arrangements with the same n_y (1, 2 or 3) and increasing n_b are connected by lines. The outlined region is magnified in the bottom plot. The staircase curve connects the optimal configurations (black dots).

5.7 Comparison of T-HOG *vs.* R-HOG

Figure 5.7 compares the accuracy of the R-HOG and T-HOG classifiers, in the optimal n_y and n_b configurations, for each descriptor size N and for each of the three datasets. As we can see, the T-HOG significantly outperforms R-HOG in all cases. For example, an T-HOG with ≈ 20 features has performance similar to R-HOG with 80 or more features. Table 5.2 gives detailed data for two cell grid and bin count combinations (1×4:5, $N = 20$, and 1×7:9, $N = 63$), selected among the optimum combinations of Figure 5.7. According to Student's

$n_x\times n_y$:n_b	Dataset	R-HOG		T-HOG		T-test
		$\boldsymbol{\mu}(A)$	$\boldsymbol{\sigma}(A)$	$\boldsymbol{\mu}(A)$	$\boldsymbol{\sigma}(A)$	
1×4:5	ICDAR	0.0109	0.0010	0.0054	0.0007	14.73
	iTowns	0.0158	0.0022	0.0082	0.0012	10.19
	Epshtein	0.0285	0.0023	0.0151	0.0016	15.10
1×7:9	ICDAR	0.0042	0.0005	0.0029	0.0005	5.81
	iTowns	0.0087	0.0013	0.0059	0.0010	5.44
	Epshtein	0.0164	0.0022	0.0120	0.0014	6.14

Table 5.2: Statistics of R-HOG and T-HOG classifiers for two optimal cell configurations.

table for $2L - 2 = 18$ degrees of freedom, the smallest t value in Table 5.2, 5.44, corresponds to a risk $\alpha < 10^{-4}$.

Figure 5.7 shows that the *ICDAR-derived* dataset is significantly easier than the other two, for both classifiers. Presumably this is due to the fact that most ICDAR images are digitized 2D documents, whereas the iTowns and Epshtein images are photos of 3D urban scenes.

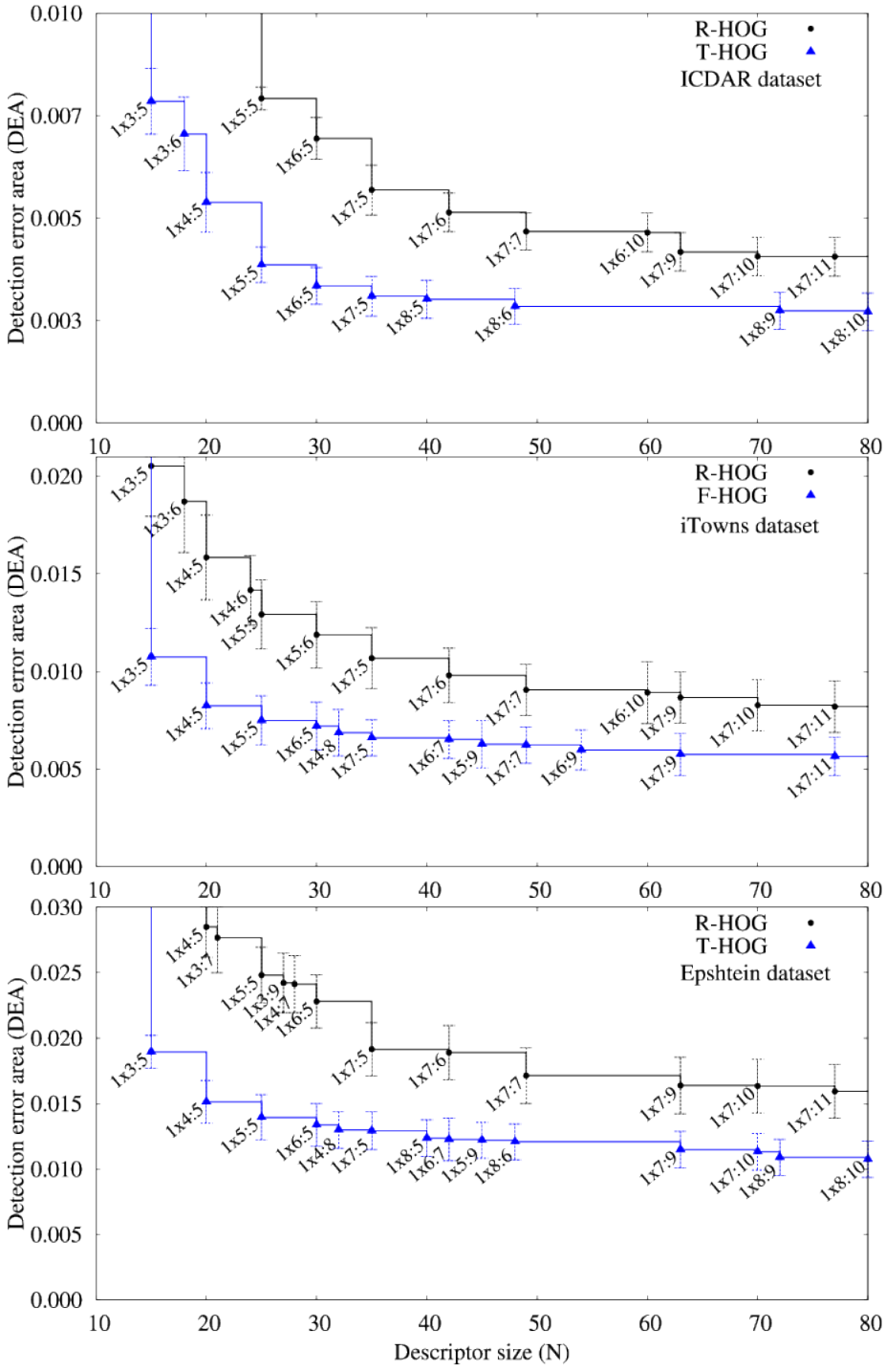


Figure 5.7: Comparison between optimal configurations of R-HOG and T-HOG. The error bars show the standard deviation over 10 random partitions of X_i, B_i. Note that the vertical scale is different in each plot.

5.8 Blurred *vs.* hard-edged cells

Finally, we performed another series of tests to quantify the contribution of blurred cell boundaries to the T-HOG performance. We considered the optimal combinations of n_y and n_b (with $n_x = 1$) identified in the tests of Section 5.5. For each combination, we compared the DEA of T-HOG on the iTowns dataset X_3, B_3, with either the Gaussian or the step weight functions defined in Section 4.5.

The results are shown in Figure 5.8. It can be seen that blurred cells are always better, but the advantage decreases as n_y increases. This is to be expected, since the difference between blurred and sharp cells decreases as the cells become narrower. For $H = 21$ and $n_y = 7$, for example, the effective height of each cell (blurred or sharp) is only three pixels.

Detailed data for two specific configurations ($1\times4{:}5$, $N = 20$, and $1\times7:9$, $N = 63$) are shown in Table 5.3. By Student's t-test the improvement is significant (at risk level $\alpha = 0.05$) for the $1\times4:5$ descriptor ($t = 8.42$), but not for the $1\times7{:}9$ descriptor ($t = 1.47$).

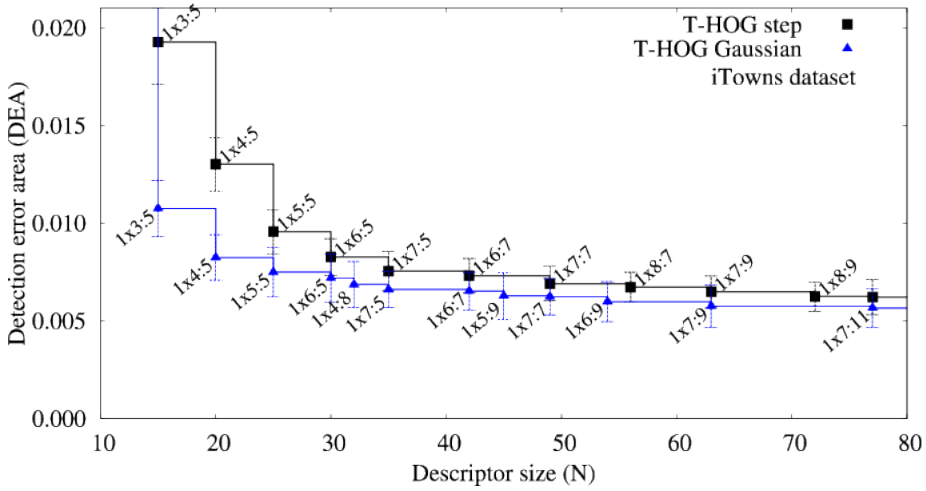


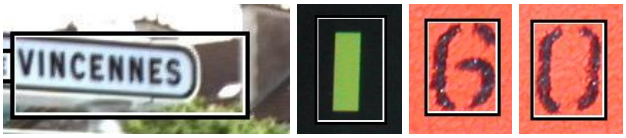
Figure 5.8: DEA plots for optimal T-HOG classifiers with sharp and blurred cells.

$n_x\times n_y{:}n_b$	Dataset	Sharp		Blurred		T-test
		$\boldsymbol{\mu}(A)$	$\boldsymbol{\sigma}(A)$	$\boldsymbol{\mu}(A)$	$\boldsymbol{\sigma}(A)$	
$1\times4{:}5$	iTowns	0.0130	0.0014	0.0082	0.0012	8.42
$1\times7{:}9$	iTowns	0.0065	0.0008	0.0059	0.0001	1.47

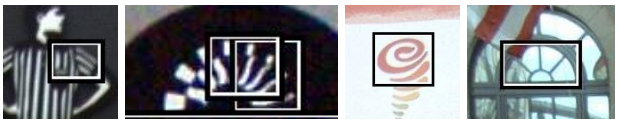
Table 5.3: Statistics for two optimal T-HOG classifiers with sharp and blurred cells.

5.9 Limitations

Figure 5.9 shows some false negatives and false positives reported by the T-HOG classifier (in the 1×7:9 configuration) for the X_i and B_i datasets. False negatives are usually due to incorrect size or position of the candidate sub-image, or to the text being only one or two characters long, to obscured or incomplete characters, or to the use of fonts with peculiar stroke orientation distributions. False positives are typically images with many line-like features in several orientations.



(a) False negatives



(b) False positives

Figure 5.9: Some errors of the T-HOG+SVM recognizer.

Chapter 6

Applications

In this chapter, we describe two applications of the T-HOG text recognizer, namely, as a post-filter for the SNOOPERTEXT detector and as the core of a sliding window text detector. A third application in text tracking will be described in part II of this thesis.

6.1 T-HOG as a post-filter to text detection

The motivating application for text classifiers like T-HOG and R-HOG was the detection of text objects in photos and videos of arbitrary scenes. Specifically, the idea is to use the classifier to filter the output of a fast but "permissive" (high-recall, moderate-precision) detector. To evaluate the suitability of T-HOG in this application, we used the SNOOPERTEXT detector of Minetto *et al.* [66] over the whole image, to generate the candidate sub-images for T-HOG.

6.1.1 SNOOPERTEXT overview

SNOOPERTEXT is an improvement of a text detector described in 2009 by Fabrizio *et al*. [28]. It was developed in the context of the ITOWNS project [26], which aims to build tools for virtual navigation of urban environments. The main raw data for these tools is a collection of GPS-tagged high-resolution digital photos of building façades, taken by a set of car-mounted cameras. A set of images covering a full-hemisphere of 360° is taken periodically as the car travels. The mean viewpoint spacing between sets is about one meter. Each set of images is assembled offline into a complete immersive panorama. See Figure 6.1.

The purpose of SNOOPERTEXT within this project was to extract (offline) any textual information present in the images, such as street and traffic signs, store names, and building numbers (see appendix A); which could then be combined with the GPS coordinates and street maps to support applications such as business finders and urban navigation aids.

Figure 6.1: Panoramic street images generated in the iTowns project.

The structure of the SNOOPERTEXT detector is outlined in Figure 6.2. SNOOPERTEXT is based on the hypothesis generation/validation paradigm. The hypothesis generation is carried out using a bottom-up approach: starting from a character segmentation, classification and grouping. The hypothesis validation is carried out by T-HOG text filtering that works over the text candidates returned by the hypothesis generation. SNOOPERTEXT uses the multi-scale technique to deal with large character size variations.

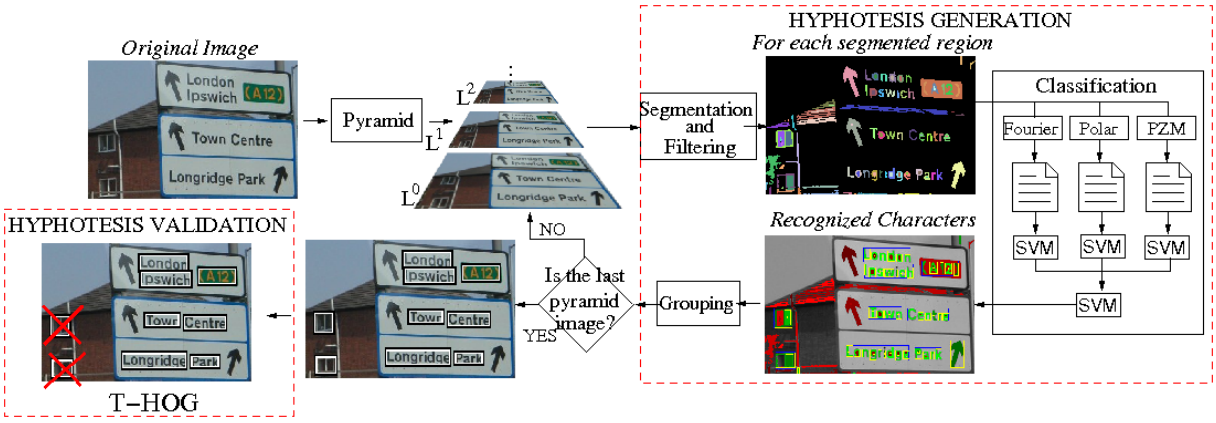


Figure 6.2: Schematic diagram of SNOOPERTEXT with multi-resolution processing and T-HOG post-filtering.

The segmentation algorithm used in SNOOPERTEXT is a modified version of Serra's *toggle mapping* [78], a morphological operator for local contrast enhancement and thresholding, using morphological erosions and dilations to define the local foreground and background levels. The output of this step is an image where every pixel is classified into three types of regions: foreground (relatively dark), background (relatively light) or indeterminate (when the original image is homogeneous in that neighborhood). The thresholding is not symmetrical between dark and light, so, in order to capture light letters on dark background the segmentation is repeated on the negative image. See Figure 6.3 (b,c).

(a) (b) (c)

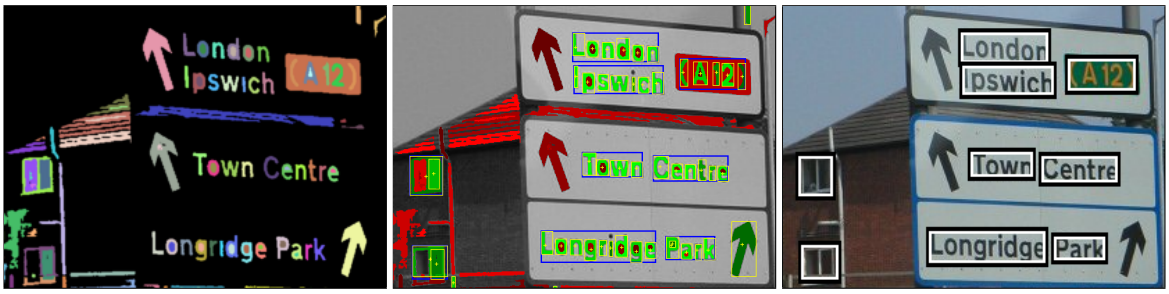
Figure 6.3: (a) An original input image. (b) The toggle segmentation into foreground (dark grey), background (white) and indeterminate (light grey) pixels. (c) The toggle segmentation of the negative image.

The foreground regions from both segmentations are then filtered by simple geometric criteria (based on area, width and height). See Figure 6.4(a). For each surviving region, that pass the geometric filtering, the algorithm extracts three scale- and rotation-invariant shape descriptors: Fourier moments, pseudo-Zernike moments, and an original polar encoding [28]. These descriptors are fed to three separate SVM classifiers, whose numeric outputs are packed as a three-vector and fed to a final SVM classifier [21], which outputs a binary letter/non-letter decision.

The regions classified as letters are then replaced by axis-aligned rectangles, as shown in Figure 6.4(b); which are then grouped into lines according to geometric criteria defined by Retornaz and Marcotegui [77]. These criteria take into account the distance and vertical alignment between neighboring regions relative to their height. In this step, isolated letters are discarded, and each candidate text line is represented by a single axis-aligned rectangle, which is the bounding box of its component letters. Note that the text lines found in both toggle segmentations must be combined to generate the candidate text regions. See Figure 6.4 (c). This grouping step is necessary in order to apply optical character recognition (OCR) to the extracted text, and it also has the benefit of discarding a large fraction of the false positives (non-letter regions

classified as letters).

My first contribution to SNOOPERTEXT was the addition of a *post-filter* to decide whether the contents of each candidate rectangle does look like a text line. However, in the original version of SNOOPERTEXT [66] we used as a text post-filter the R-HOG of Dalal and Triggs, with the parameters values specified by the authors (which were tuned for human recognition). For this thesis, we used our T-HOG and the R-HOG with different parameters, tuned for text classification as described in Chapter 5.



(a) (b) (c)

Figure 6.4: (a) The candidate characters regions detected by segmentation and size criteria. (b) Output of the region classification step with regions classified as letters in green, and non-letters in red. (c) Text lines resulting from character grouping.

My other contribution to SNOOPERTEXT was the use of multi-scale analysis [38] to efficiently deal with the wide variation of font sizes and textures of the texts found in the ITOWNS images. Namely, for each image, SNOOPERTEXT first builds a multi-scale *image pyramid*. The base (level 0) of the pyramid is the original image, and each level l is a copy of the next lower one $l-1$, reduced to half its width and half its height (so that level l has $1/4^l$ as many pixels as level 0).

The basic SNOOPERTEXT algorithm described above is applied separately to each level of the pyramid. At each level, the algorithm only looks for letters with a fixed narrow range $[\alpha \,..\, \alpha+\delta]$ of area sizes, which corresponds to the size range $\left[4^l\alpha \,..\, 4^l(\alpha+\delta)\right]$ in the original image. The parameter δ is chosen so that there is some overlap between two consecutive scales l and $l+1$, namely $\alpha+\delta > 4\alpha$. Segmented regions whose area fall outside the interval $[\alpha \,..\, \alpha+\delta]$ are ignored since they will probably be found in other levels. See Figure 6.5.

The multi-scale approach allows SNOOPERTEXT to use a structuring element of fixed (and modest) size in each morphological operation, with significant speed gains. Note that the cost of processing the whole image pyramid, for letters of any size, is then less than $1+1/4+1/16+1/64+\cdots = 4/3$ times the cost of processing the original frame only for letters within the fixed range $[\alpha \,..\, \alpha+\delta]$.

Figure 6.5: Multi-scale text detection by SNOOPERTEXT on an ICDAR challenge image.

Another advantage of this multi-scale approach is that it makes the segmentation algorithm insensitive to letter texture (high frequency details that are much smaller than the letters) which could break each letter into many separate regions. See Figure 6.6.

6.1.2 Metrics for text detection

The standard metrics to compare text detectors systems used in the literature are based on the ICDAR 2005 measure of similarity [57] between two rectangles r, s, defined as

$$m(r,s) = \frac{S(r \cap s)}{S(r \cup s)} \tag{6.1}$$

Level 0 Level 1 Level 2

Figure 6.6: Letter texture supression by SNOOPERTEXT's multiscale processing. Note that the letters of 'SIGNO' are oversegmented in levels 0 and 1 but correctly segmented in level 2.

where $S(\cdot)$ is the area of the smallest rectangle enclosing the set $\cdot$. The function $m(r, s)$ ranges between 0 (if the rectangles are disjoint) and 1 (if they are identical). The metric m is extended to a set of rectangles Z by the formula

$$m(r, Z) = \max\{m(r, s') : s' \in Z\} \tag{6.2}$$

and the similarity between two sets is given by

$$m(R, Z) = \sum_{r \in R} m(r, Z) \tag{6.3}$$

From this indicator one derives the ICDAR *precision* p and *recall* r scores [57]

$$p = \frac{m(T, G)}{\#T} \qquad r = \frac{m(G, T)}{\#G} \tag{6.4}$$

where G is the set of manually identified text regions in the input images, and T is the set of text regions reported by the detector. For ranking purposes, the ICDAR 2005 committee used the f*-score* [57] which is the harmonic mean of precision and recall,

$$f = 2/(1/p + 1/r) \tag{6.5}$$

Metrics averaging

There are several ways of averaging these metrics over a multi-image database. The approach used by the ICDAR 2005 scoring program (method I) is to evaluate p, r and f separately for each image, and then compute the arithmetic mean of the f-scores over all images. Another approach (II) is to compute p and r for each image, then take the arithmetic means of all p and r values, and compute f from these means. A third approach (III) is to compute p, r globally, as if all images were concatenated in a single big image, and f from the results. These points must be considered when comparing f values reported by different authors.

We note that the approach I, and, to a lesser extent, approach II, suffer from higher sampling noise and negative bias compared to approach III. For instance, if the number of legible text regions varies substantially from frame to frame, the average computed by approaches I and II will be very different from approach III, and therefore, may not measure the real accuracy of the algorithm being tested. See Figure 6.7. The approach III was used to compare text tracking algorithms in part II of this thesis.

6.1.3 Tests and results

We compared the performance of SNOOPERTEXT without post-filter, and with either the T-HOG or the R-HOG as the post-filter. We also compared them with several state-of-the-art detectors described in the literature. Specifically, we compared with the published scores of the detectors that entered the ICDAR 2003 and 2005 Challenges [56] also compared with the detectors of Yi and Tian [91], H. Chen *et al.* [19] and Epshtein *et al.* [27], which had the highest f-scores reported in the literature (as of 2011).

Two critical parameters of SNOOPERTEXT are the minimum size A (in pixels) of the detected character regions in each scale, and the minimum number of characters (GOC) that a text-line must contain. We found that the optimal values of these parameters, when SNOOPERTEXT was used without post-filter, were $A = 50$ and $GOC = 3$ (that is, only lines with 3 or more characters were reported). These settings are denoted "ST3" in what follows.

When SNOOPERTEXT was used in combination with the R-HOG or T-HOG as a post-filter, we found that the optimum parameters were $A = 25$ and $GOC = 2$ (which increase the recall but significantly reduce the precision). We denote these settings by "ST2". For the T-HOG and

(a)

Legible objects: 19
Detected: 19
False positives: 0
$p = 1.0, r = 1.0$ and $f = 1.0$

(b)

Legible objects: 3
Detected: 2
False positives: 4
$p = 0.33, r = 0.66$ and $f = 0.44$

Figure 6.7: Impact of the score averaging method exemplified with a collection consisting of two images (a) and (b) with hypothetical detector outputs shown as rectangular outlines. If the scores are averaged by approach I we get $p = 0.66$, $r = 0.83$ and **f = 0.72**. By approach II we get $p = 0.66$, $r = 0.83$ and **f = 0.74**. However if we use approach III then we have $p = 0.84$ (21/25) and $r = 0.95$ (21/22) and **f = 0.89**. In this example we assumed perfect geometrical matching ($m(r, s) = 1$) for the regions that were detected.

R-HOG, we used the optimal parameters specified in Section 5.4, with the cell arrangement $n_x = 1, n_y = 7$ and $n_b = 9$, resulting in a descriptor of size $N = 63$. See Figure 6.8.

The results, for each of the three image collections, are shown in Tables 6.1–6.3. All p and r scores were computed with the ICDAR scoring program [55]. The scores in the f_{I} and f_{II} columns were averaged by approaches I and II, respectively. We cannot show the f_{III} scores because the per-frame counts needed to compute it were not available for the compared systems.

Note that T-HOG is a more effective post-filter for SNOOPERTEXT than R-HOG; and that the best combination (ST2+T-HOG) is much better than the best results of SNOOPERTEXT without post-filter (ST3). Note also that the combination ST2+T-HOG is at least as effective as the best published methods on the ICDAR dataset, and outperforms the Stroke Width Transform (SWT) [27] of Epshtein *et al.* on their own dataset.

To confirm the results of Section 5.5, we tested the ST2+T-HOG combination with a smaller descriptor ($n_x = 1, n_y = 4$, $n_b = 5$, $N = 20$). The f-score was about 1 to 2% lower, on

ICDAR image collection				
System	p	r	f_{I}	f_{II}
ST2+T-HOG	0.73	0.61	0.65	0.67
ST2+R-HOG	0.70	0.62	0.64	0.66
Yi and Tian [91]	0.71	0.62	0.62	0.66
H. Chen *et al.* [19]	0.73	0.60	—	0.66
Epshtein *et al.* [27]	0.73	0.60	—	0.66
ST3+T-HOG	0.72	0.57	0.62	0.64
ST3+R-HOG	0.72	0.57	0.62	0.64
Hinnerk Becker†	0.62	0.67	0.62	0.64
ST3	0.64	0.59	0.59	0.61
Alex Chen†	0.60	0.60	0.58	0.60
ST2	0.42	0.65	0.47	0.51
Ashida†	0.55	0.46	0.50	0.50
HWDavid†	0.44	0.46	0.45	0.45
Wolf†	0.30	0.44	0.35	0.36
Qiang Zhu†	0.33	0.40	0.33	0.36
Jisoo Kim†	0.22	0.28	0.22	0.25
Nobuo Ezaki†	0.18	0.36	0.22	0.24
Todoran†	0.19	0.18	0.18	0.19
Full†	0.01	0.06	0.08	0.02

Table 6.1: Performances of various text detectors on the "testing" subset of the ICDAR image collection. The competitors of the ICDAR 2003 and 2005 Challenges are marked with †. For this table, the T-HOG and R-HOG classifiers were trained on the output of the ST2 detector applied to the ICDAR "training" subset.

average, for the three image collections. We also tested the ST2+T-HOG combination with the sub-image divided into vertical stripes ($n_x = 7, n_y = 1$, $n_b = 9$, $N = 63$); the f-score was about 5 to 7% lower.

iTowns image collection				
System	p	r	f_{I}	f_{II}
ST2+T-HOG	0.72	0.50	0.56	0.59
ST2+R-HOG	0.70	0.49	0.54	0.58
ST3+T-HOG	0.72	0.43	0.51	0.54
ST3+R-HOG	0.72	0.43	0.51	0.54
ST3	0.49	0.43	0.43	0.46
ST2	0.24	0.53	0.31	0.33

Table 6.2: Performances of the SNOOPERTEXT detector, with and without HOG post-filtering, on the whole iTowns image collection. For this table, the R-HOG and T-HOG classifiers were trained on the $X_1 \cup X_3$ and $B_1 \cup B_3$ datasets.

Epshtein image collection				
System	p	r	f_{I}	f_{II}
ST2+T-HOG	0.59	0.47	0.49	0.52
ST2+R-HOG	0.55	0.44	0.46	0.49
ST3+T-HOG	0.64	0.39	0.46	0.49
Epshtein *et al.* [27]	0.54	0.42	—	0.47
ST3+R-HOG	0.62	0.37	0.43	0.46
ST3	0.46	0.42	0.41	0.44
ST2	0.19	0.54	0.25	0.28

Table 6.3: Performances of the SNOOPERTEXT detector, with and without HOG post-filtering, and of the Epshtein *et al.* detector on the whole Epshtein image collection. For this table, the R-HOG and T-HOG classifiers were trained on the $X_1 \cup X_2$ and $B_1 \cup B_2$ datasets.

Figure 6.8: Output of the SNOOPERTEXT detector with $A = 25$ and $GOC = 2$ (left), and the same output after filtering with the T-HOG recognizer (right) on images from Epshtein and iTowns collections.

6.2 T-HOG as a text detector

Any text recognizer can also be used on its own as a sliding-window text detector. Namely, the recognizer is applied to a sufficiently large set of sub-regions of the input image, and the sub-regions with the largest scores are returned as the output.

Figure 6.9 shows the result of such a text detector, using the T-HOG+SVM recognizer, with a window of fixed size (24 by 72 pixels) sliding over the whole image. Note the high selectivity of recognizer. For this test, we trained the SVM classifier using the set U of positive "ground-truth" sub-regions provided by the ICDAR Challenge team [55], and a set V of negative random sub-regions of the ICDAR images, disjoint from the set U and about three times its size.

Text of variable size can be detected by running this algorithm on several reduced versions of the input image, in multi-scale fashion. See Figure 6.10. However, this brute-force approach to text detection is extremely expensive, since the number of windows that need to be analyzed is very large. For this reason we did not bother to evaluate its accuracy or compare it to other detectors.

6.3 Concluding remarks

The combination SNOOPERTEXT + T-HOG outperformed state-of-the-art text detection systems described in the literature. However, there is still plenty of room for improvement. For example, some failures of our system are due to incorrect placement of the bounding boxes. Therefore, we need to improve the grouping step to only generate text words or text lines outcomes, depending on which metric is used. For example, the ICDAR metric penalizes text line outputs (they must be split into separate words for a higher performance).

Concerning T-HOG as a text detector, there are some ideas to apply it on a regular sparsely grid, spanning the whole image, to save computational time.

Finally, the system SNOOPERTEXT + T-HOG combined with an OCR was successfully applied in a real database of high-resolution geo-referenced street images of the iTowns project [26]. The semantic data, automatically extracted from the images with our system, was used to build a real *Geographic Information System* (GIS) search engine application, that locates streets by textual queries (see appendix A). In this aspect, code optimizations are important to save time because, depending on the complexity of the image, e.g. number of segmented regions, the total execution may take more than one minute.

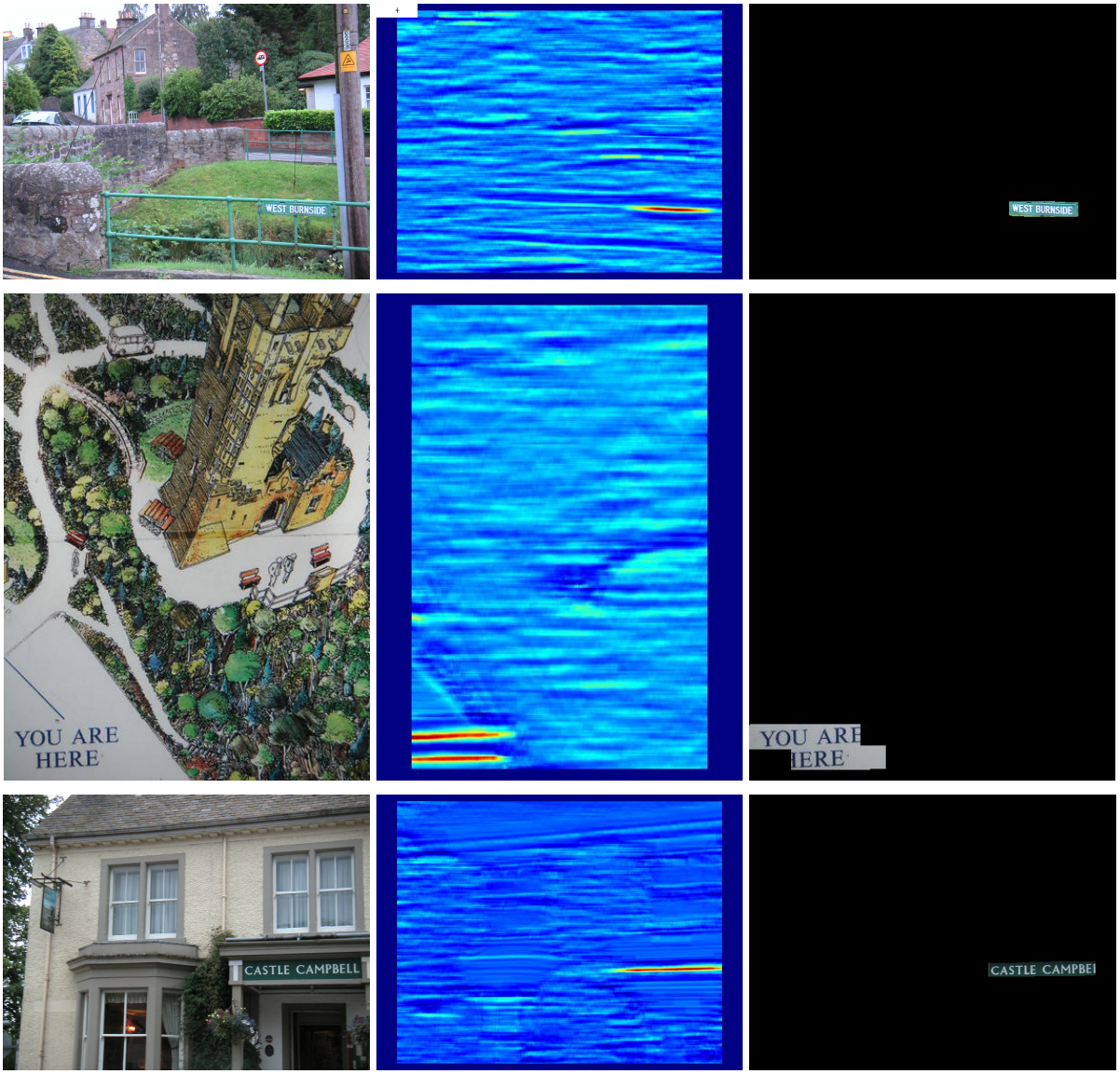


Figure 6.9: Left: Images from ICDAR dataset. Center: the output of the T-HOG+SVM sliding window classifier, encoded as the color of the central pixel (warm tones for positive output, cold tones for negative). The white rectangle at the top left corner shows the size of the sliding window. Right: the union of the 100 windows with highest scores.

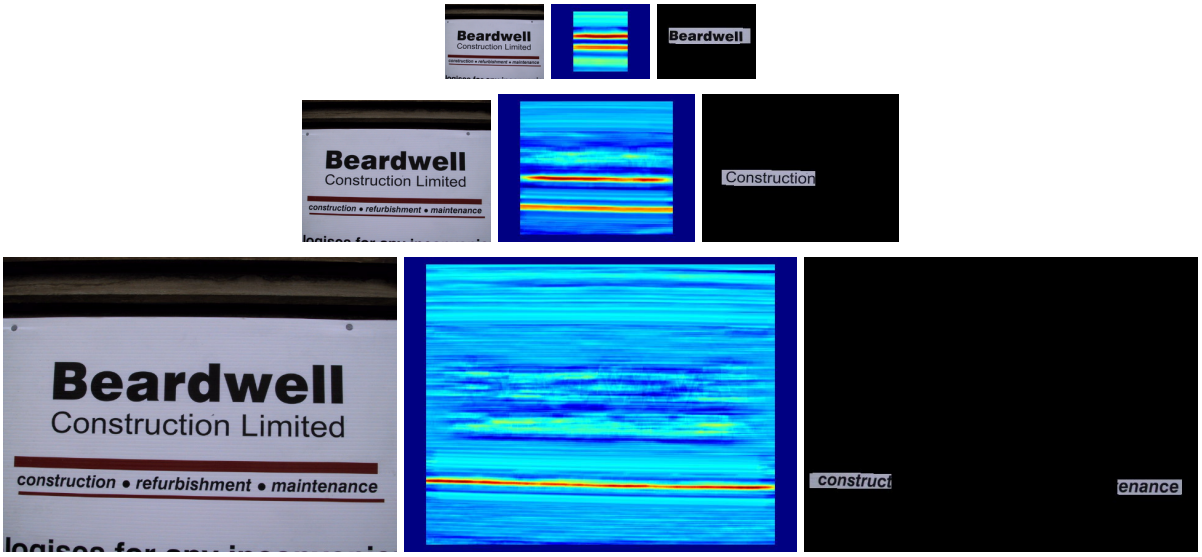


Figure 6.10: Text detection in a multi-scale fashion.

Part II

Text Tracking

Chapter 7

Introduction

In this part of the thesis we consider the *text tracking problem*. Specifically, we consider the integration of text detection and tracking algorithms to identify which text regions in successive frames correspond to the same physical text object.

As in part I, we are primarily concerned with text embedded in natural urban scenes or building interiors. The video is typically recorded by a moving camera, either hand-held or fixed on a vehicle. The input data for this problem is a digital video. The output is a set of candidate text regions for each video frame and the identification of the regions which supposedly belong to the same physical text object.

7.1 Motivation

The automatic identification and tracking of text in videos is essential in many applications, such as the automatic indexing of video libraries, construction of street directories from videos taken by car-mounted cameras, identification of vehicles by their license plates, *etc*.

For those applications, it is not sufficient to apply a text detector separately to each frame. This simplistic approach ignores the temporal coherence between video frames, and therefore it will usually produce thousands of regions with almost, but not always quite, the same textual information. To avoid this redundancy, it is necessary to *track* each physical text object across successive frames. In addition, by exploiting the redundancy between frames, a text tracking algorithm can often be more accurate than a single-frame text detector in both precision and recall. Finally, tracking also allows the use of super-resolution techniques to extract text images with much higher resolution and quality that could be obtained from any single image.

7.2 Our contributions

Our major contribution to this problem is the description of four text tracking algorithms (TRACK1 to TRACK4) and an analysis of their strengths and weaknesses. (The most efficient ones are improvements of the SNOOPERTRACK algorithm, published in 2011 by Minetto *et al* [67]). Such algorithms consist of variations of how detection and tracking can be integrated. Most of the existing solutions to this problem are variations of these four algorithms.

Another contribution is the definition of special metrics to evaluate the performance of text tracking systems. These metrics evaluate a system's accuracy based on three aspects: *text region detection*, the ability to detect visible text objects in a frame; *text object detection*, the ability to detect text objects somewhere in the video; and *text object tracking*, the ability to identify regions that belong to the same text object in different frames.

We also developed a tracker specifically designed for text objects, based on particle filtering and the T-HOG classifier. The T-HOG is used twice in this tracker: first, as an image region signature, to evaluate the similarity between the contents of two regions in successive frames; and, second, as a classifier, to measure how much "text-like" is the contents of a predicted region.

Finally, as part of this work we also created a benchmark of six videos of urban scenes, with associated XML annotations [62], which we hope will be useful to other researchers working on this problem.

7.3 Statement of the problem

We assume that the input video $\mathbb{V}$ is a list of n images (*frames*) $\mathbb{V}^{(0)}, \mathbb{V}^{(1)}, \ldots, \mathbb{V}^{(n-1)}$ with common domain $\mathfrak{D}$. We define a *text region* in a video frame as a region of $\mathfrak{D}$ that contains the projection of a text object. We denote by $r^{(i)}[j]$ the text region number j detected or tracked in frame $\mathbb{V}^{(i)}$. As in part I, we will restrict our research to text lines whose projections are mostly horizontal. Therefore we assume that each text region r is a rectangular axis-aligned box, represented by its center $r.p_{\mathrm{f}}$ and axis vectors $r.u_{\mathrm{f}}$ (horizontal) and $r.v_{\mathrm{f}}$ (vertical). See Section 2.3.

By *text tracking* we mean recover the text regions in each frame by identifying which regions in successive frames correspond to the same physical text object. We represent this information as a list $T^{(0)}, T^{(1)}, \ldots, T^{(m)}$ where each $T^{(i)}$ is a list of presumed text regions in frame $\mathbb{V}^{(i)}$, and a list $\pi^{(1)}, \pi^{(2)}, \ldots, \pi^{(m)}$ where each $\pi^{(i)}$ is a relation between $T^{(i-1)}$ and $T^{(i)}$. Specifically, a pair $(r^{(i-1)}[j], r^{(i)}[k])$ is in the relation $\pi^{(i)}$ iff the regions $r^{(i-1)}[j] \in T^{(i-1)}$ and $r^{(i)}[k] \in T^{(i)}$ are believed to be projections of the same text object. See Figure 7.1.

This output representation is adequate for the algorithms that we discuss here, which can follow a text object only along an unbroken sequence of consecutive frames. A tracker that can

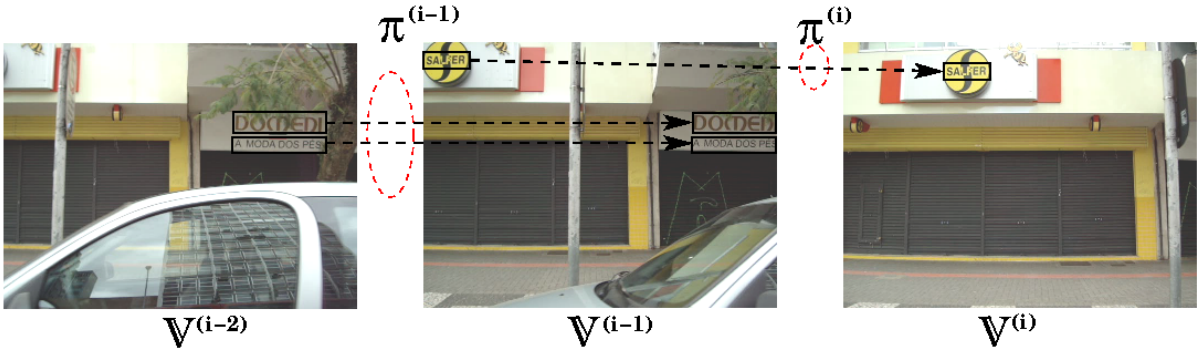


Figure 7.1: The output of a text tracking algorithm, showing the detected text regions $r^{(i)}[j]$ (rectangles) and the tracking relations $\pi^{(i)}$ (arrows).

recover an object after losing it for one or more frames would have to output additional relations $\pi^{(ij)}$, between the sets of regions detected in non-consecutive frames $\mathbb{V}^{(i)}$ and $\mathbb{V}^{(j)}$.

The transitive, reflexive, and symmetric closure of the union of all the relations $\pi^{(i)}$ is an equivalence relation π^* that partitions the set of all text regions in all frames into one or more classes; each class constitutes the presumed trajectory of a physical text object along the video.

The algorithms that we consider in this thesis are such that every relation $\pi^{(i)}$ is one-to-one, that is, each region in one frame is related to at most one region in the next frame. Therefore each class of the π^* relation will be a list of regions, one from each frame, spanning one or more consecutive frames.

7.4 Challenges

The integration of text detection and object tracking algorithms is a non-trivial problem, considering that both algorithms are prone to several failures. Depending on how they are combined, these failures may multiply instead of canceling out.

As we discussed in part I, common failures modes of text detection algorithms include: *false negatives* (failure to detect a text object that is legible in the frame); *false positives* (mistaking a non-text region for a text one); *splitting* of a single text line into several regions; and *joining* of distinct text lines into a single region.

Tracking algorithms are also subjected to several kinds of errors, including *drift*, *jitter* and *loss* which are described below. Tracking generally requires comparing some part of a frame with some *template image* that represents the expected appearance of the text object, as gathered from previously processed frames. The occurrence and severity of tracking errors depend on the approach used for updating the template image. In the variant called *recursive tracking*, the template is the projection of the text object as located in the previous frame. In *fixed-template*

tracking, on the other hand, the same template image is used for all frames of the video. An intermediate strategy is *key frame tracking*, where the template is extracted from a selected *key frame* of the video itself, and used for all subsequent frames until the next key frame.

7.4.1 Tracking jitter

In both recursive and fixed-template trackers, the regions reported for the same text object in successive frames exhibit random-looking variations in size and position, called *jitter*, due to image noise or inaccuracies in the algorithm itself. See Figure 7.2.

Figure 7.2: Tracking jitter in three consecutive frames.

7.4.2 Tracking drift

In addition to jitter, recursive trackers may also incur in substantial *drift*, that is, a cumulative error in the reported size and position of the text object. The reason is that any errors made in one frame are incorporated into the template and therefore affect the tracking in subsequent frames. See Figure 7.3. Eventually the drift may grow to the point that the text object is lost, or the tracker begins to follow an unrelated text object. These errors may be further magnified

Figure 7.3: Tracking drift in recursive tracking.

(and perpetuated) by the motion prediction methods that trackers use to estimate the position of the object on the next frame.

Fixed-template and key-frame trackers are usually free from cumulative drift, because the reported position in each frame is not affected by any errors made in the previous frame – unless those errors are so large that they cause the tracker to entirely miss the text object on the next frame.

7.4.3 Tracking loss

Tracking algorithms can fail completely when the text object's position and appearance change substantially between consecutive frames. Such changes may be due to sudden variation of lighting conditions (see Figure 7.4), or to large and sudden changes in the relative position of camera and object in tri-dimensional space, or excessive perspective deformations.

Figure 7.4: Tracking loss due to lighting changes.

These problems can be alleviated by using template-location algorithms that are insensitive to affine deformations and photometric changes; and/or by extending the search range. However, these changes make the tracker more expensive and may actually increase the occurrence of tracking losses.

However, even with some tracking losses, e.g. due to hard photometric changes, the combination detection and tracking in videos is generally more robust than to apply a text detector separately to each frame. See Figure 7.5. This is why exploiting the temporal coherence between sucessive frames is essential.

Figure 7.5: Text detection loss.

Chapter 8

Related work

There is a vast literature on how to combine detection and tracking. The survey of Yilmaz *et al.* [92] covers several state-of-the-art systems up to 2006. Such systems are tailored for many types of features, such as faces, vehicles, people, *etc*. Several approaches have been considered, a popular one is called *detection-before-tracking*. In this approach the objects of interest are detected at some frame separately; then, the tracked objects in distinct frames are paired with the observations provided by the detector. Bugeau et Pérez in 2007 [9] described some advantages and limitations of track-before-detect algorithms. Another approach is called *tracking-before-detection*. Tracking-before-detection algorithms are specially designed to track targets which are difficult to detect in static images, but can be identified by their motion over several frames. In these algorithms, several candidate trajectories are generated and evaluated over several sucessive frames. Accumulators are often used to detect the existence of the targets, since they would be rarely detected by using fixed thresholds. Such algorithms are useful in various surveillance scenarios such as tracking aircrafs or vehicles in infrared radar images. There is also another class of algorithms based on the *tracking-by-detection* approach. In such algorithms the target is described by a set of features (e.g. SIFT [52] features, HOG [24] features, SURF [11] features, *etc*). Another set of features is used to describe the background, and a binary classifier trained over the chosen features is used to separate the target from background in successive frames.

Concerning *text detection and tracking* in videos little have been published. A survey up to 2004, published by Jung *et al*. [39], cover some of these works. However, many authors who worked on this problem considered mainly the detection of text artificially inserted into videos of real scenes, such as sub-titles, advertisements, movie credits, *etc*. Therefore, they usually made rather restrictive assumptions on the text, such as, constrained position, uniform motion, and known font shape and size [36], which do not hold for images of scene-embedded text objects.

A pioneering system for detection and tracking of text embedded on real scenes was de-

scribed in 1998–2000 by Huiping-Li *et al*. [47, 48]. They used a multi-scale detector that scanned each level of the image pyramid with a sliding window of 16×16 pixels. Each window was classified as text or non-text with a neural network operating on the high-frequency wavelet coefficients. The result was a bitmap with value 1 at every "text-like" position of the window. The regions in the bitmap were then merged, bounded and filtered in order to generate candidate text regions. Tracking was performed by linear motion prediction followed by a translation adjustment to minimize the total squared pixel error relative to the previous frame. Their method allowed gradual changes of the text's projection, but tended to fail when the text object moved in front of complex backgrounds. The text detector was applied only at a selected set of key-frames, with forward tracking used in the intervening frames. The process begins again at each key-frame; they do not attempted to merge the tracked regions with those detected at the next key-frame.

In 2002, Lienhart and Wernicke [50] were apparently the first to use bidirectional tracking for text objects. Like Huiping-Li *et al*. they identified text lines by a multi-scale sliding window detector trained with a neural network, but operating on gradient features rather than wavelet coefficients. Their detector too produced a binary classification map tool, from which the text candidates were extracted. The text detector was applied only to selected key-frames, and forward and backward tracking was used on the intervening frames. In their system, tracking was performed by exhaustive search around the previous region position without motion prediction, to minimize the difference in a specific image descriptor. This descriptor summarizes the spatial distribution of pixel values in the candidate region [50]. The template used during the tracking is changed every few frames, thus allowing for gradual deformation. The text detector was applied only to selected key-frames, and forward and backward tracking was used on the intervening frames. They do not specify how the detected, forward-tracked, and backward-tracked regions were merged.

In 2007, Merino *et al*. [60] apparently introduced particle-filter tracking for text objects. They used a text detector based on the tree of *connected components* of León *et al*. [44]. The character regions are identified in the component tree based on size, energy and texture. They defined a text region as a list of SIFT features [52, 53] obtained from those characters, with specific relative positions apart from a single rotation and translation applied to the entire text region. Tracking was performed by a particle filter using particle weights computed from geometric similarity of the relative features position in the text line complemented by SIFT matching of each single feature.

Another particle-filter based text tracker was described in 2008–2009 by Tanaka and Goto [33, 80]. Their system did not use the multi-scale techniques sliding window approach; instead they partitioned the frame into blocks of 16×16 pixels, and classified each block as text or non-text using discrete-cosine-transform (DCT) features [7]. Text-like blocks were then grouped into text regions. This classification was performed on every frame. The particle-filter

tracker operates on individual text-containing blocks; the motion of each block was then predicted to the next frame and evaluated by the L1 distance between the pixel intensity histograms in each color channel. Each text region was then assumed to be located near the majority of its tracked blocks.

In 2010, Zhi Li *et al*. [49] introduced binary search as a way of further reducing the text detection cost. Their text detector was applied initially at key-frames. Like Huiping-Li *et al*. [47, 48] and Lienhart and Wernicke [50] they used high-frequency wavelet coefficients and and image descriptors to summarize the spatial distribution of pixel values in order to find the text line regions. Instead of tracking each new text region over successive frames, they applied the text detector to a few additional intervening frames between two key-frames, in binary search fashion, to determine the first and the last occurrence of the text line. To compare text candidates in different frames, they used the mean L1 metric over the text region. However, while the range of frames that a text appear may be enough for some applications, this approach may spend the same amount of time to determine all the occurrences of a text region if it was necessary.

Some authors have worked specifically on the text tracking problem, separate from text detection. In 2005, Myers and Burns [31], and in 2010, Na and Wen [69] developed text trackers to handle more complex geometric text transformations such as rotation, perspective and scale changes. They represented a text region as a parallelogram or as a general convex quadrilateral. Myers and Burns described an approach using bi-dimensional affine transformations to track planar text regions that can undergo arbitrary 3-D rigid motion and scale changes. They used normalized correlation to track blocks of 15×15 pixels and RANSAC [30] to ensure tracking consistency. Na and Wen [69] used SIFT features and SIFT matching to handle such transformations.

Finally, several authors considered the problem of text detection and tracking in compressed videos [22, 23, 32, 51, 76, 95] by exploiting characteristics of the MPEG encoding to speed up both tasks. Most of these methods are designed for artificially inserted text. They generally used the discrete-cosine-transform coefficients of the MPEG encoding for text detection, and the MPEG motion vectors for text tracking [23]. Although these methods are faster, the tracking often suffers because MPEG motion vectors are chosen for maximum compression rather than modeling the true motion.

Chapter 9

Tracking Strategies

Text tracking algorithms can be classified according to the way that text detection and tracking are integrated. In this chapter we describe and analyze four increasingly complex strategies TRACK1–TRACK4. See Figure 9.1.

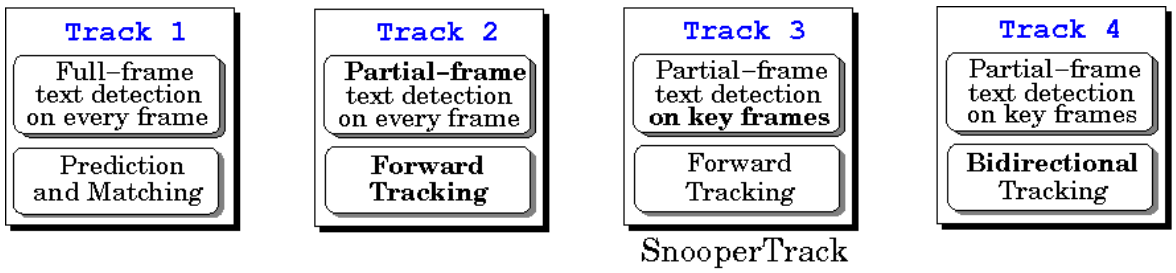


Figure 9.1: Four basic text tracking strategies.

These four algorithms depend on a text detection procedure to find the text objects to be tracked. The detector may be applied to the entire domain $\mathfrak{D}$ of each frame, or to a proper subset of $\mathfrak{D}$, defined by a binary mask (bitmap) with the same domain $\mathfrak{D}$.

These four strategies seem to represent most of the systems described in the literature. One exception is the system of Zhi Li *et al*. [49], which differs from TRACK4 by using binary search instead of bidirectional tracking between the key-frames.

9.1 Detection and matching

The most obvious text tracking strategy is to run a text detection algorithm on each frame of the video, independently, and then run a matching algorithm to find the tracking relations between

the regions detected in successive frames. This strategy is outlined by the TRACK1 algorithm below.

Algorithm 1 TRACK1 $(\mathbb{V}^{(0)}, \mathbb{V}^{(1)}, \ldots, \mathbb{V}^{(n-1)})$

1. $\pi^{(0)} \leftarrow \{\}$;
2. $T^{(0)} \leftarrow$ TEXTDETECT $(\mathbb{V}^{(0)}, \mathfrak{D}(\mathbb{V}^{(0)}))$;
3. *For each* $i \in \{1, 2, \ldots, n-1\}$ *do*
4. $\quad T^{(i)} \leftarrow$ TEXTDETECT $(\mathbb{V}^{(i)}, \mathfrak{D}(\mathbb{V}^{(i)}))$;
5. $\quad (E, \lambda) \leftarrow$ PREDICT $(T^{(i-1)}, \pi^{(i-1)}, \mathfrak{D}(\mathbb{V}^{(i)}))$;
6. $\quad \pi^{(i)} \leftarrow$ MATCH $(T^{(i)}, E, \lambda, \mathbb{V}^{(i-1)}, \mathbb{V}^{(i)})$;
7. *Return* T, π.

The input to the algorithm is a video $\mathbb{V}$ with n frames $\mathbb{V}^{(0)}, \mathbb{V}^{(1)}, \ldots, \mathbb{V}^{(n-1)}$. The outputs of the algorithm are a list of text regions $T^{(i)}$, for each frame $\mathbb{V}^{(i)}$, $i \in \{0, \ldots, n-1\}$; and the inferred tracking relations $\pi^{(i)} \subseteq T^{(i-1)} \times T^{(i)}$ between frames $\mathbb{V}^{(i-1)}$ and $\mathbb{V}^{(i)}$, for each $i \in \{1, \ldots, n-1\}$. In order to simplify the exposition, we assume that $T^{(i)}$ is empty if $i < 0$ or $i \geq n$, and $\pi^{(i)}$ is a empty relation if $i \leq 0$ or $i \geq n$.

The procedure TEXTDETECT is an arbitrary text detection algorithm (such as the sliding T-HOG of Section 6.2, or the SNOOPERTEXT described in Section 6.1.1) that operates only over the domain $\mathfrak{D}$ of the frame. Each call to TEXTDETECT (steps 2 and 4) produces a list of rectangular text regions in the given frame $\mathbb{V}^{(i)}$. The call to PREDICT (step 5) predicts the position of each text region t from the set $T^{(i-1)}$ in the next frame $\mathbb{V}^{(i)}$, given its position $s = [\pi^{(i-1)}]^{-1}(t)$ in the previous frame $\mathbb{V}^{(i-2)}$. It returns a list E of predicted text regions in frame $\mathbb{V}^{(i)}$, and the tracking relation $\lambda \subseteq T^{(i-1)} \times E$. The procedure PREDICT is detailed in Section 9.1.1.

The call to MATCH (step 6) looks for matches between the set E of predicted regions and the set $T^{(i)}$ of regions detected in frame $\mathbb{V}^{(i)}$. It returns the final tracking relation $\pi^{(i)} \subseteq T^{(i-1)} \times T^{(i)}$. The MATCH procedure is detailed in Section 9.1.2.

Observe that in TRACK1 algorithm the final reported regions $T^{(i)}$ are always those regions found by the detector; the predicted regions in E are used only to build the tracking relations $\pi^{(i)}$.

9.1.1 Motion prediction

The procedure PREDICT is given a set F of regions that were detected in some frame $\mathbb{J}$ and predicts their positions in the next frame $\mathbb{I}$. For simplicity we assume that the procedure is a simple linear extrapolation that uses the text regions F and the positions of those text objects in the previous frame $\mathbb{K}$, which are obtained through a tracking relation ξ between frames $\mathbb{K}$ and $\mathbb{J}$

(see chapter 10 for more sophisticated prediction methods). It also requires the domain $\mathfrak{D}$ of the next frame $\mathbb{I}$. The procedure returns the set E of successfully predicted text regions for frame $\mathbb{I}$, and the tracking relation λ between F and E. See Figure 9.2.

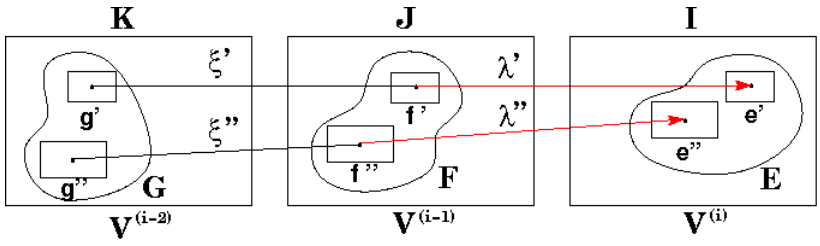


Figure 9.2: Frames, sets and tracking relations.

Algorithm 2 PREDICT $(F, \xi, \mathfrak{D})$

1. $\lambda \leftarrow \{\}$; $E \leftarrow \{\}$;
2. *For each* $f \in F$
3. $\quad g \leftarrow \xi^{-1}(f)$;
4. $\quad$ *If* $g \neq \emptyset$ *then*
5. $\quad\quad e \leftarrow 2f - g$;
6. $\quad$ *else*
7. $\quad\quad e \leftarrow f$;
8. $\quad$ *If* $e \subseteq \mathfrak{D}$ *then*
9. $\quad\quad E \leftarrow E \cup e$;
10. $\quad\quad \lambda \leftarrow \lambda \cup \{(f, e)\}$;
11.
12. *Return* E, λ.

In step 3, the procedure uses the relation ξ to get the region g corresponding to f in the previous frame $\mathbb{K}$. If there is no such region, the predicted region e is the same as f (step 7). Otherwise e (step 5) is obtained from regions f and g. Here we assume a simple linear prediction applied to all text region parameters ($p_{\mathrm{f}}.x,\ p_{\mathrm{f}}.y,\ u_{\mathrm{f}}.x,\ v_{\mathrm{f}}.y$) of g and f disregarding the contents of those regions. If e falls inside the image domain $\mathfrak{D}$ (step 8), it is included in the list E (step 9), and the relation λ is updated (step 10).

In forward tracking, the image $\mathbb{K}, \mathbb{J}$ and $\mathbb{I}$ are consecutive frames $\mathbb{V}^{(i-2)}, \mathbb{V}^{(i-1)}$ and $\mathbb{V}^{(i)}$ of the video.

9.1.2 Matching

The procedure MATCH compares a set of regions D detected in some frame $\mathbb{V}^{(i)}$ ($\mathbb{I}$) with a set of predicted regions E for the same frame. It requires the previous frame

$\mathbb{V}^{(i-1)}$ ($\mathbb{J}$) and a pairing λ between the set F of regions in $\mathbb{J}$ and the set E. It outputs a tracking relation π that pairs the regions of F with regions in D.

The relation π is basically a subset of the given relation λ, with every predicted region replaced by the matching detected region, excluding regions of E that cannot be matched.

Algorithm 3 MATCH $(D, E, \lambda, \mathbb{J}, \mathbb{I})$

1. $\pi \leftarrow \{\}$;
2. $P \leftarrow$ SIMILARITYMATRIX$(D, E, \mathbb{J}, \mathbb{I})$;
3. $\alpha \leftarrow$ HUNGARIANALGORITHM(P);
4. *For each* $(e \in E)$
5. $\quad d \leftarrow \alpha(e)$;
6. $\quad$ *If* $d \neq \emptyset$ *then*
7. $\quad\quad f \leftarrow \lambda^{-1}(e)$;
8. $\quad\quad \pi \leftarrow \pi \cup \{(f, d)\}$;
9.
10. *Return* π;

In step 2, the procedure builds a similarity matrix P with one row for each region of D and one column for each region of E. The entry in row d and column e is a real number between 0 and 1 that measures the likelihood of region d and region e being projections of the same text object. Each value of P is computed according equations (10.4) and (10.17) and takes into account the geometry of the two regions and their contents, respectively.

In step 3, the Hungarian Algorithm [43] is used to compute an optimal partial matching between E and D (using P) returned as a list of pairs $\alpha \subseteq E \times D$. The composition of λ and α gives the desired relation π between regions of F and those of D. See Figure 9.3.

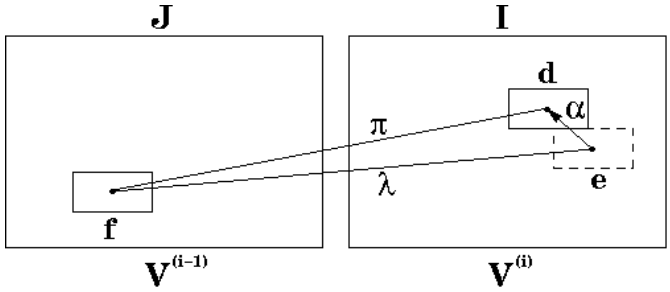


Figure 9.3: Tracking relation composition.

9.1.3 Analysis

The text tracking systems of Merino *et al.* [60] and Tanaka and Goto [33, 80] are similar to this approach, that is, full frame-to-frame detection and matching.

Observe that the procedure PREDICT used by TRACK1 performs a simple linear prediction and does not consider the contents of the predicted regions. For that reason the predicted regions are usually less reliable than the detected regions $T^{(i)}$. That is the reason why the final reported regions $T^{(i)}$ are always the detected ones rather than the predicted ones. Therefore, the accuracy and robustness of TRACK1 are completely dependent on those of the text detector.

The experiments in Chapter 11 shows the most common problem faced in this approach, that continuous detection on every frame is sometimes impossible due to noise, compression artifacts, cluttered background. Whenever the text detector fails in one frame the trajectories are broken. In such case, the TRACK1 strategy starts a new trajectory in the next frame as if the previous text object had disappeared and a new text object had came into view. Such inconvenience was also reported by Merino results [59].

On the other hand, the advantage of this approach is that it detects a text object as soon as it appears, and can track it until it would become unrecognizable or occluded. Another advantage of this algorithm is that it can cope, by setting the MATCH routine according, with large and unpredictable displacements of text regions between frames, which are common in videos taken with hand-held cameras and/or low frame rate. Finally, the cost of TRACK1 is usually dominated by the cost of text detection.

9.2 Complementary detection and forward tracking

In order to reduce the cost of the algorithm, one could replace the PREDICT procedure of TRACK1 by a more sophisticated tracking procedure, that looks at the contents of the predicted region and adjusts it so that it can matches the contents seen in previous frames. In that case, the output of the tracker is usually more accurate and reliable than a detector could provide to the same region. Therefore, it is better to use the successfully predicted regions in $T^{(i)}$, and run the text detector only in those parts of the frame where the prediction produced no results.

This strategy is implemented by the TRACK2 algorithm below.

Algorithm 4 TRACK2 $(\mathbb{V}^{(0)}, \mathbb{V}^{(1)}, \ldots, \mathbb{V}^{(n-1)})$

1. $\pi^{(0)} \leftarrow \{\}$;
2. $T^{(0)} \leftarrow$ TEXTDETECT $(\mathbb{V}^{(0)}, \mathfrak{D}(\mathbb{V}^{(0)}))$;
3. *For each* $i \in \{1, \ldots, n-1\}$ *do*
4. $\quad (E, \pi^{(i)}) \leftarrow$ TRACKREGIONS $(T^{(i-1)}, \mathbb{V}^{(i-1)}, \mathbb{V}^{(i)}, \mathfrak{D}(\mathbb{V}^{(i)}))$;
5. $\quad \mathcal{X}^{(i)} \leftarrow$ UNITEREGIONS (E);
6. $\quad D \leftarrow$ TEXTDETECT $(\mathbb{V}^{(i)}, \mathfrak{D}(\mathbb{V}^{(i)}) \setminus \mathcal{X}^{(i)})$;

7. $\quad \llcorner\ T^{(i)} \leftarrow E \cup D;$
8. *Return* T, π.

The TRACKREGIONS routine, described in Section 10.1, is used in step 4 to find, for each region in $T^{(i-1)}$, its apparent position in frame $\mathbb{V}^{(i)}$. These positions, which are all pairwise disjoint and contained in the frame domain $\mathfrak{D}(\mathbb{V}^{(i)})$, are returned in the set E. The procedure also returns the relation $\pi^{(i)}$ that pairs the regions of E with those in $T^{(i-1)}$.

In step 5, the domain $\mathfrak{D}$ of each frame $\mathbb{V}^{(i)}$ is divided into two regions: the union $\mathcal{X}^{(i)}$ (UNITEREGIONS procedure) of all successfully predicted parallelograms in E with an extra safety margin and its complement ($\mathfrak{D}(\mathbb{V}^{(i)}) \setminus \mathcal{X}^{(i)}$). This subset could be represented for instance by a bitmap with the same domain $\mathfrak{D}$ as the frame.

Except for the first frame $\mathbb{V}^{(0)}$, the text detection (step 6) is applied only to the subset $\mathfrak{D}(\mathbb{V}^{(i)}) \setminus \mathcal{X}^{(i)}$ of $\mathbb{V}^{(i)}$, the ignored region $\mathcal{X}^{(i)}$ is assumed not to contain any new text. We assume that the text detector returns a set of pairwise disjoint regions contained in the subset $\mathfrak{D}(\mathbb{V}^{(i)}) \setminus \mathcal{X}^{(i)}$ of the frame domain.

In this strategy there is no intersection between the successfully predicted regions E and the detected regions D; therefore the final set $T^{(i)}$ is merely the union of both sets D and E (step 7), and the tracking relation $\pi^{(i)}$ is the pairing produced by TRACKREGIONS (step 4).

9.2.1 Analysis

One advantage of TRACK2 is the reduction of computational cost, since the text detection is performed only on some parts of the frame.

The correctness and completeness of the tracking relations obtained by TRACK2 depends strongly on the accuracy of the tracking algorithm. Therefore, the TRACKREGIONS procedure must take into account the contents of the current and the predicted regions, and adjust the position of the latter accordingly. Indeed, the main advantage of this strategy is the robust prediction of those detected regions instead a simple geometric prediction as used by the TRACK1 algorithm.

Likewise TRACK1, this algorithm can also detect objects as soon as they appear, and track them as long as they remain visible.

9.3 Detection on key frames with forward tracking

For many common scenes, the legible text regions cover only a small fraction of each frame, and most of the non-text areas are repeated on several successive frames. Strategies TRACK1 and TRACK2 waste a considerable amount of time running the detector repeatedly over those parts. Thus, for increased efficiency we may consider running the text detection only on a given

set K of *key frames*, and rely exclusively on the tracking procedure for all the other frames. This strategy is implemented by the TRACK3 algorithm below.

Algorithm 5 TRACK3 $(\mathbb{V}^{(0)}, \mathbb{V}^{(1)}, \ldots, \mathbb{V}^{(n-1)}, K)$

1. $\pi^{(0)} \leftarrow \{\}$;
2. $T^{(0)} \leftarrow$ TEXTDETECT $(\mathbb{V}^{(0)}, \mathfrak{D}(\mathbb{V}^{(0)}))$;
3. *For each* $i \in \{1, 2, \ldots, n-1\}$ *do*
4. $\quad (T^{(i)}, \pi^{(i)}) \leftarrow$ TRACKREGIONS $(T^{(i-1)}, \mathbb{V}^{(i-1)}, \mathbb{V}^{(i)}, \mathfrak{D}(\mathbb{V}^{(i)}))$;
5. $\quad$ *If* $i \in K$ *then*
6. $\quad\quad \mathcal{X}^{(i)} \leftarrow$ UNITEREGIONS $(T^{(i)})$;
7. $\quad\quad D \leftarrow$ TEXTDETECT $(\mathbb{V}^{(i)}, \mathfrak{D}(\mathbb{V}^{(i)}) \setminus \mathcal{X}^{(i)})$;
8. $\quad\quad T^{(i)} \leftarrow T^{(i)} \cup D$;
9.
10. *Return* T, π.

In this strategy, text detection is applied in steps 2 and 7 only to the key frames specified by the given set $K \subseteq \{0, \ldots, n-1\}$ of frame indices. At those frames, the TEXTDETECT routine is applied to the area $\mathfrak{D}(\mathbb{V}^{(i)}) \setminus \mathcal{X}^{(i)}$ not covered by the predicted regions (step 7). Note that index 0 is implicitly included in K. The key frames can be chosen by regular sampling with a fixed step, or by more sophisticated heuristics that take into account object and camera motion, frame disparity, or other criteria.

The routine TRACKREGIONS is the same used in TRACK2 algorithm and described in Section 10.1. It is used in every frame except $\mathbb{V}^{(0)}$ (step 4). In step 8 any detected region is added to the set $T^{(i)}$, as new text objects. The procedure also returns the tracking relation $\pi^{(i)}$ that associates regions in frame $\mathbb{V}^{(i-1)}$ and $\mathbb{V}^{(i)}$.

The TRACK3 algorithm is a minor improvement of the original SNOOPERTRACK [67] algorithm of Minetto *et al.*. SNOOPERTRACK, like TRACK3, applied the text detector (SNOOPERTEXT) only on key-frames and used a particle filter (TRACKREGIONS routine) to follow the detected objects. However, it applied the text detector to the whole domain $\mathfrak{D}(\mathbb{V}^{(i)})$ of the key-frame, instead of the subset $\mathfrak{D}(\mathbb{V}^{(i)}) \setminus \mathcal{X}^{(i)}$. It then used the MATCH procedure of TRACK1 to pair up the detected and tracked regions in that key-frame. Whenever it found a matching pair, SNOOPERTRACK would keep in the set $T^{(i)}$ the average of the detected and the tracked region geometries. Unpaired regions of either set were also kept in $T^{(i)}$.

The TRACK3 algorithm turns out to be more accurate than SNOOPERTRACK because (as observed in Section 9.2) our tracker is more accurate and reliable than our detector (see some common text detection failures in Section 11.6.1) at the task of recovering the sucessive occurences of the same text regions across the video frames. Therefore, by skipping the MATCH procedure, the TRACK3 strategy avoided the complexity of the pairing, avoided many errors introduced by the detector and does not waste time computing it.

9.3.1 Analysis

The TRACK3 strategy is similar to the system of Huiping-Li *et al*. [47, 48]. However, unlike us, they do not attempt to merge the tracked regions with those detected at the next key-frame.

The only advantage of this strategy over TRACK2 is speed, since it does not run the detector on frames that are not in K. However its reliability strongly depends on the choice of the key frames. Increasing the step between key frames reduces cost but may result in loss of text objects that appear and disappear in the interval between those frames, or happen to be momentarily undetectable (due to perspective, light reflections, shadows, *etc*) at the key frames. Another major disadvantage is that a text region that becomes detectable between two key frames will not be detected until the next key frame. Because of these delays, the reported trajectories will often be shorter than those that would be determined by TRACK2.

9.4 Detection on key frames with bi-directional tracking

To avoid the late text detection problem of TRACK3, one could apply the tracking procedure backward in time to the new regions detected in a key frame, in order to recover them in the preceding non-key frames. This idea is implemented by the TRACK4 algorithm bellow.

Algorithm 6 TRACK4 $(\mathbb{V}^{(0)}, \mathbb{V}^{(1)}, \ldots, \mathbb{V}^{(n-1)}, K)$

1. $\pi^{(0)} \leftarrow \{\}$;
2. $T^{(0)} \leftarrow$ TEXTDETECT $(\mathbb{V}^{(0)}, \mathfrak{D}(\mathbb{V}^{(0)}))$;
3. *For each* $i \in \{1, 2, \ldots, n-1\}$ *do*
4. $\quad (T^{(i)}, \pi^{(i)}) \leftarrow$ TRACKREGIONS $(T^{(i-1)}, \mathbb{V}^{(i-1)}, \mathbb{V}^{(i)}, \mathfrak{D}(\mathbb{V}^{(i)}))$;
5. $\quad \mathcal{X}^{(i)} \leftarrow$ UNITEREGIONS $(T^{(i)})$;
6. $\quad$ *If* $i \in K$ *then*
7. $\quad\quad D \leftarrow$ TEXTDETECT $(\mathbb{V}^{(i)}, \mathfrak{D}(\mathbb{V}^{(i)}) \setminus \mathcal{X}^{(i)})$;
8. $\quad\quad T^{(i)} \leftarrow T^{(i)} \cup D$;
9. $\quad\quad \pi^{(i+1)} \leftarrow \emptyset$;
10. $\quad\quad$ *For each* $j \in \{i-1, i-2, \ldots, 0\}$ *While* $D \neq \emptyset$ *do*
11. $\quad\quad\quad (D, \lambda) \leftarrow$ TRACKREGIONS $(D, \mathbb{V}^{(j+1)}, \mathbb{V}^{(j)}, \mathfrak{D}(\mathbb{V}^{(j)}) \setminus \mathcal{X}^{(j)})$;
12. $\quad\quad\quad \mathcal{X}^{(j)} \leftarrow \mathcal{X}^{(j)} \cup$ UNITEREGIONS (D);
13. $\quad\quad\quad T^{(j)} \leftarrow T^{(j)} \cup D$;
14. $\quad\quad\quad \pi^{(j+1)} \leftarrow \pi^{(j+1)} \cup \lambda^{-1}$;
15. $\quad\quad$
16. $\quad$
17. *Return* T, π.

In this strategy, whenever the text detection routine is executed in step 7, a tracking backward in time is started (steps 10–14) on the new text objects found. The TRACKREGIONS

routine is the same used in strategies TRACK2 and TRACK3 (described in section 10.1), but using the previously computed bitmaps $\mathfrak{D}(\mathbb{V}^{(i)}) \setminus \mathcal{X}^{(i)}$ to exclude the area covered by previously detected or tracked regions.

In step 8, the new detected regions of frame $\mathbb{V}^{(i)}$ are added to the set $T^{(i)}$. In step 11, we excluded from $\mathfrak{D}(\mathbb{V}^{(j)})$ those frame regions $\mathcal{X}^{(j)}$ where objects were found by the forward tracking. In step 12 we save the regions found by the backward tracking in $\mathcal{X}^{(j)}$. Note that with this step the regions in D will not be considered again in a future backward tracking.

In steps 13 and 14, the set T and the tracking relation π are updated with all successfully regions tracked backwards.

9.4.1 Analysis

This algorithm has most of the efficiency of TRACK3, since the detection is still applied only to a subset of the key frames. Note that the tracking prediction, both forward and backward, usually examines only a small fraction of the frames domain.

The advantage of this strategy is that it can recover the first detectable occurrence of a given text object by tracking it backward in time.

However, it still has the other drawbacks of TRACK3, e.g. the problem of key frame set definition. Another potential problem is the inability of backward tracking in a real time application.

Finally, note that even TRACK2 can have benefits of using backward tracking, since an image region just scanned by a text detector may contain a text region.

Chapter 10

Frame-to-frame tracking of a text object

In this chapter we describe the core of the text tracking routine used in the TRACK2, TRACK3 and TRACK4 algorithms, namely the TRACKREGIONS procedure. It receives a set $T^{(i)}$ of text regions in some frame $\mathbb{V}^{(i)}$, and a subset $\mathfrak{D}$ of the latter's domain. For each region in $T^{(i)}$, it locates its apparent position and shape in $\mathbb{V}^{(i+1)}$, and returns a set $T^{(i+1)}$ of those regions, which will be all pairwise disjoint and contained in $\mathfrak{D}$. It also returns a relation $\pi^{(i)}$ that pairs each region of $T^{(i)}$ with the corresponding one in $T^{(i+1)}$. The simple linear extrapolation is not adequate because it gets easily confused by random camera orientation jitter. Therefore, we have adopted the particle-filter (bootstrap) technique [37, 73].

10.1 Particle filter

The particle-filter tracker maintains a population of candidate sub-regions ("particles") for each text object being tracked. Each particle represents a possible motion state of the object. For each new frame, the particles are propagated independently based on some transition model. The algorithm then evaluates all these candidates, assigning a probability weight for each one based on some observation model (color model, HOG model, etc). Finally it resamples those candidates, replicating each one according its probability. It is expected, that the set of particles contains more weight at locations where the object being tracked is more likely to be. The general scheme of the particle filter technique is illustrated in Figure 10.1.

More precisely, for each text object r^* in $T^{(i)}$, TRACKREGIONS keeps an associated population of particles ("candidates"). Each particle is an object state, that is, a list $\hat{r}^{(i)} = (r^{(i)}, r^{(i-1)}, \ldots, r^{(i-k+1)})$ of k candidate text regions, one in each of the last k frames $\mathbb{V}^{(i)}, \mathbb{V}^{(i-1)}, \ldots, \mathbb{V}^{(i-k+1)}$. The final region $r^{(i)}$ of the state is a region in frame $\mathbb{V}^{(i)}$. For each particle $\hat{r}^{(i)}$, the procedure uses a Kalman-style predictor to compute the probability distribution $\mathcal{P}_\mathcal{K}$ of the corresponding text region in frame $\mathbb{V}^{(i+1)}$ (the *transition model*), and then draws a

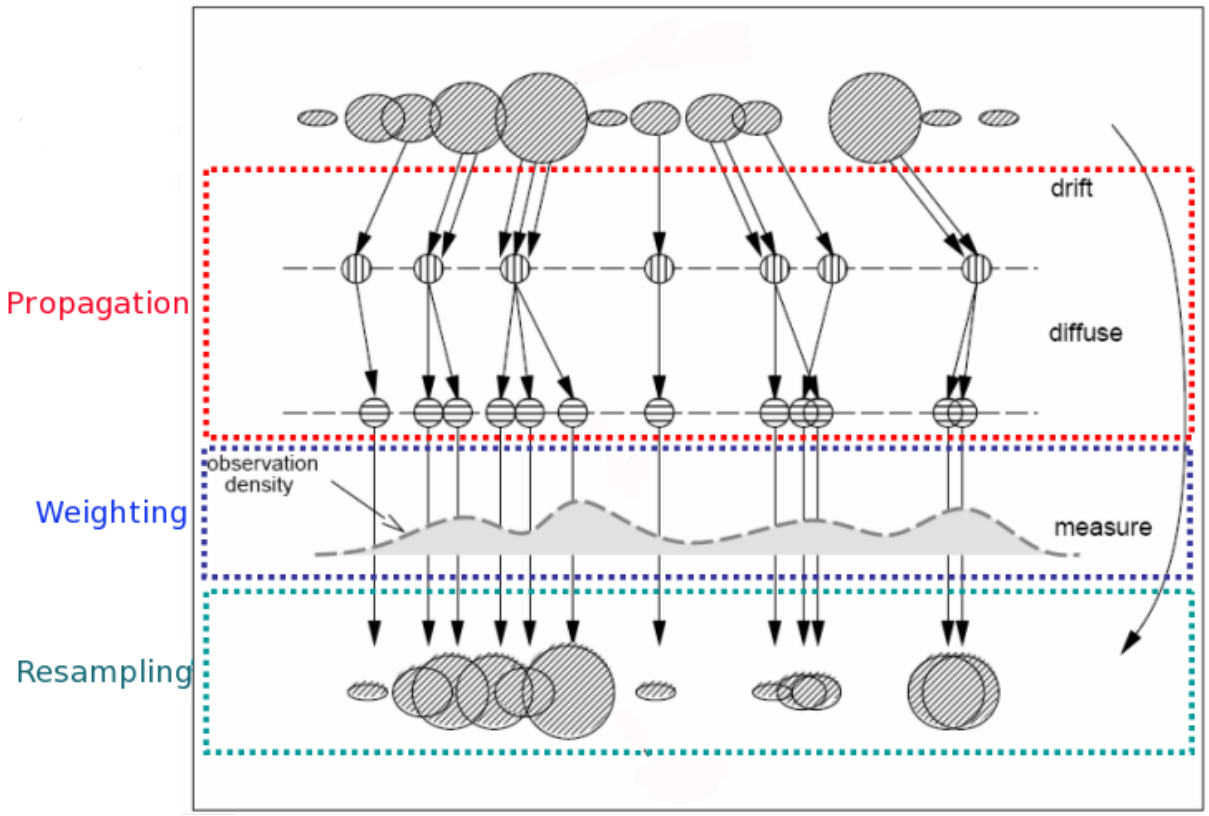


Figure 10.1: General scheme of a particle filter tracking (Isard and Blake [37]).

candidate region $r^{(i+1)}$ from this distribution. This step is called *propagation phase*.

The procedure then evaluates the propagated region $r^{(i+1)}$ of all particles associated to r^*, assigning to each one a conditional probability $\mathcal{P}_{\mathcal{O}}(r^{(i+1)})$ (the *observation model*) based on its contents in frame $\mathbb{V}^{(i+1)}$. This is called the *weighting* or *measurement phase*. Finally, it resamples the particles of the text object r^*, randomly eliminating and replicating each particle according this conditional probability distribution. This is the *resampling phase*. Finally it updates the state of each particle from $\hat{r}^{(i)}$ to $\hat{r}^{(i+1)} = (r^{(i+1)}, r^{(i)}, \ldots, r^{(i-k+2)})$.

10.1.1 Transition model

In the Kalman predictor of order k, the state of a feature r^* after processing frame i consists of the list $\hat{r}^{(i)} = (r^{(i)}, r^{(i-1)}, \ldots, r^{(i-k+1)})$ of its regions in the last k frames $\mathbb{V}^{(i)}, \mathbb{V}^{(i-1)}, \ldots, \mathbb{V}^{(i-k+1)}$.

The Kalman prediction algorithm produces a probability distribution $\mathcal{P}_\mathcal{K}(r^{(i+1)}|\hat{r}^{(i)})$ for the feature's region $r^{(i+1)}$ in the next frame $\mathbb{V}^{(i+1)}$. The distribution $\mathcal{P}_\mathcal{K}$ is a multivariate Gaussian, whose mean is a fixed linear function of the current state $\hat{r}^{(i)}$ and whose covariance matrix is fixed for all frames and features.

That is, let $\breve{r}^{(i)}$ be the geometric parameters of $r^{(i)}$, linearized as a 4-element vector

$$\breve{r}^{(i)} = (r^{(i)}.p_\mathrm{f}.x,\ r^{(i)}.p_\mathrm{f}.y,\ r^{(i)}.u_\mathrm{f}.x,\ r^{(i)}.v_\mathrm{f}.y) = (\breve{r}^{(i)}.x,\ \breve{r}^{(i)}.y,\ \breve{r}^{(i)}.u,\ \breve{r}^{(i)}.v) \tag{10.1}$$

Then, the mean of the distribution $\mathcal{P}_\mathcal{K}(r^{(i+1)}|\hat{r}^{(i)})$ is the 4-vector

$$\breve{\mu}^{(i+1)} = A_0\breve{r}^{(i)} + A_1\breve{r}^{(i-1)} + \cdots + A_k\breve{r}^{(i-k)} \tag{10.2}$$

and

$$\mathcal{P}_\mathcal{K}(r^{(i+1)}|\hat{r}^{(i)}) = \exp[-(\breve{r}^{(i+1)} - \breve{\mu}^{(i+1)})^\top M^{-1}(\breve{r}^{(i+1)} - \breve{\mu}^{(i+1)})] \tag{10.3}$$

where $A_o, A_1, \ldots, A_k$ and M are fixed 4×4 matrices, which are parameters of the model, and should be determined by previous statistical analysis of the correct regions in representative videos.

For this thesis, we assume that each of the parameters $\breve{r}^{(i)}.x, \breve{r}^{(i)}.y, \breve{r}^{(i)}.u$ and $\breve{r}^{(i)}.v$ have independent Gaussian distributions. In other words, we assume that the coefficients matrices $A_0, A_1, \ldots, A_k$ and the covariance matrix M are diagonal matrices. Therefore, we can separate the 4-dimensional Gaussian distribution $\mathcal{P}_\mathcal{K}$ of equation (10.3) into the product of four one-dimensional Gaussians:

$$\begin{aligned}\mathcal{P}_\mathcal{K}(r^{(i+1)}|\hat{r}^{(i)}) = \gamma(&\breve{r}^{(i+1)}.x, \breve{\mu}^{(i+1)}.x, \sigma_x)\\ &\gamma(\breve{r}^{(i+1)}.y, \breve{\mu}^{(i+1)}.y, \sigma_y)\\ &\gamma(\breve{r}^{(i+1)}.u, \breve{\mu}^{(i+1)}.u, \sigma_u)\\ &\gamma(\breve{r}^{(i+1)}.v, \breve{\mu}^{(i+1)}.v, \sigma_v)\end{aligned} \tag{10.4}$$

where γ is the standard Gaussian distribution (equation 4.3) and where each component of the mean $\breve{\mu}^{(i+1)}$ is computed separately by the formula:

$$\breve{\mu}^{(i+1)}.x = A_0.x\ \breve{r}^{(i)}.x + A_1.x\ \breve{r}^{(i-1)}.x + \cdots + A_k.x\ \breve{r}^{(i-k)}.x \tag{10.5}$$

$$\breve{\mu}^{(i+1)}.y = A_0.y\ \breve{r}^{(i)}.y + A_1.y\ \breve{r}^{(i-1)}.y + \cdots + A_k.y\ \breve{r}^{(i-k)}.y \tag{10.6}$$

$$\breve{\mu}^{(i+1)}.u = A_0.u\ \breve{r}^{(i)}.u + A_1.u\ \breve{r}^{(i-1)}.u + \cdots + A_k.u\ \breve{r}^{(i-k)}.u \tag{10.7}$$

$$\breve{\mu}^{(i+1)}.v = A_0.v\ \breve{r}^{(i)}.v + A_1.v\ \breve{r}^{(i-1)}.v + \cdots + A_k.v\ \breve{r}^{(i-k)}.v \tag{10.8}$$

In our experiments we used $k = 2$ (a second-order auto-regressive model), therefore we have

$$\breve{\mu}^{(i+1)}.x = A_0.x\ \breve{r}^{(i)}.x + A_1.x\ \breve{r}^{(i-1)}.x \tag{10.9}$$

$$\breve{\mu}^{(i+1)}.y = A_0.y\ \breve{r}^{(i)}.y + A_1.y\ \breve{r}^{(i-1)}.y \tag{10.10}$$

$$\breve{\mu}^{(i+1)}.u = A_0.u\ \breve{r}^{(i)}.u + A_1.u\ \breve{r}^{(i-1)}.u \tag{10.11}$$

$$\breve{\mu}^{(i+1)}.v = A_0.v\ \breve{r}^{(i)}.v + A_1.v\ \breve{r}^{(i-1)}.v. \tag{10.12}$$

10.1.2 Observation model

In this thesis, the probability $\mathcal{P}_\mathcal{O}$ is assumed to be proportional to an *observation score* $\mathcal{S}_\mathcal{O}$, which is the product of two factors,

$$\mathcal{P}_\mathcal{O}(r^{(i+1)}) \propto \mathcal{S}_\mathcal{O}(r^{(i+1)}) = \mathcal{S}_\mathcal{T}(r^{(i+1)}, \mathbb{V}^{(i+1)})\mathcal{S}_\mathcal{C}(r^{(i+1)}, \mathbb{V}^{(i+1)}, z^*) \tag{10.13}$$

The factor $\mathcal{S}_\mathcal{T}(r^{(i+1)}, \mathbb{V}^{(i+1)})$ measures how much "text-like" is the contents of $r^{(i+1)}$ in the current frame $\mathbb{V}^{(i+1)}$. Similarly, classification scores has also been used in other particle filter based trackers, such as those proposed by Chateau *et al.* [18] in 2007 and Breitenstein *et al.* [17] in 2010. The second factor $\mathcal{S}_\mathcal{C}(r^{(i+1)}, \mathbb{V}^{(i+1)}, z^*)$ measures the similarity between the contents of region $r^{(i+1)}$ in the frame $\mathbb{V}^{(i+1)}$ and that of some "template" z^* associated to the object r^*.

If the maximum score $\mathcal{S}_\mathcal{O}$ of any particle is below a fixed threshold, the procedure assumes that the text object r^* was lost, and drops its from the tracked set. In the published version of SNOOPERTRACK [67] algorithm, we built and updated a spatiotemporal descriptor for each tracked region. That is, only the $\mathcal{P}_\mathcal{C}$ term was used in the observation model. The $\mathcal{P}_\mathcal{T}$ term, averaged over the last few frames (by an affine combination), was used to decide whether the object r^* should continue to be tracked, according to a fixed threshold.

Contents classification

For the classification score $\mathcal{S}_\mathcal{T}(r, \mathbb{I})$ we used an estimate of the probability $\mathcal{P}_\mathcal{T}(r, \mathbb{I})$ of a region r in some frame image $\mathbb{I}$ being a text region, based only on its contents. For that purpose, we use the SVM score $f(z)$ of the T-HOG descriptor $z \in \mathbb{R}^N$ (defined in Section 4.7) extracted from the contents of r in $\mathbb{I}$. There are several ways for mapping the scores $f(z)$ produced by an SVM to probabilities, as discussed by Platt [74]. We have observed that the T-HOG/SVM classification scores $f(z)$ of text and non-text regions have approximately Gaussian distributions with fixed means μ_+, μ_- and deviations σ_+, σ_-, respectively. Therefore we estimate the probability $\mathcal{P}_\mathcal{T}$ by the Bayes formula

$$\mathcal{P}_\mathcal{T}(r, \mathbb{I}) = \frac{p_\mathcal{T}\ \bar{\gamma}(f(z),\ \mu_+,\ \sigma_+)}{p_\mathcal{T}\ \bar{\gamma}(f(z),\ \mu_+,\ \sigma_+) + (1 - p_\mathcal{T})\ \bar{\gamma}(f(z),\ \mu_-,\ \sigma_-)} \tag{10.14}$$

where $p_\mathcal{T}$ is the a priori probability that r is indeed a text region, and $\bar{\gamma}$ is the standard Gaussian probability distribution

$$\bar{\gamma}(z, \mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}}\gamma(z, \mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(z-\mu)^2}{2\sigma^2}\right) \tag{10.15}$$

Statistical analysis of the training set X, B (see Chapter 11) gave the parameters $\mu_+ = 1.88, \mu_- = -2.30, \sigma_+ = 1.17$ and $\sigma_- = 0.93$. The a priori probability $p_\mathcal{T}$ was assumed to be 0.5.

Contents similarity

The score $\mathcal{S}_{\mathcal{C}}(r, \mathbb{I}, z^*)$ measures the similarity between the contents of a region r in frame $\mathbb{I}$ and a "template" z^* for the text object r^*, extracted from the occurrences of r^* in previous frames.

The template could be defined in several ways: it could be simply the content of the text region r^* in the first frame where r^* was detected. In our implementation, the template is a T-HOG descriptor of that first region, which is compared to the T-HOG descriptor of region r in frame $\mathbb{I}$.

To measure the similarity of two descriptors z' and $z''(k) \in \mathbb{R}^N$, we use Bhattacharyya's similarity coefficient

$$d_B(z'(k), z''(k)) = \left[1 - \sum_{k=1}^{N} \sqrt{z'(k)\ z''(k)}\right]^{1/2} \tag{10.16}$$

We are assuming that both descriptors are normalized to unit sum. We also assumed that the square of Battacharyya's distance between the T-HOG descriptors of two projections of the same text has an exponential probability distribution. Therefore, we used

$$\mathcal{S}_{\mathcal{C}}(r, \mathbb{I}, z^*) = \exp\left(-\frac{{d_B}^2(z', z'')}{\sigma^2}\right) \tag{10.17}$$

where z' is the T-HOG of r in $\mathbb{I}$, z'' the T-HOG of z^*, σ^2 is the standard deviation of ${d_B}^2$. In our experiments we set $1/\sigma^2 = 20$.

Chapter 11

Experiments

In this chapter we compare the text tracking schemes described in Chapter 9.

11.1 Video dataset

To evaluate the algorithms we used 6 real videos from urban scenes, listed in Table 11.1. Figures 11.1 and 11.2 show some sample frames. These videos were taken with a hand-held camera under natural lighting, and therefore are affected by natural noise, perspective distortion, blurring, illumination changes, shadows, highlights and occlusions. In videos v3 and v6, some text lines entered the frame gradually across it edges. In the remaining videos, one or more text objects are followed by the camera so that they are visible in all frames, except for temporary occlusions.

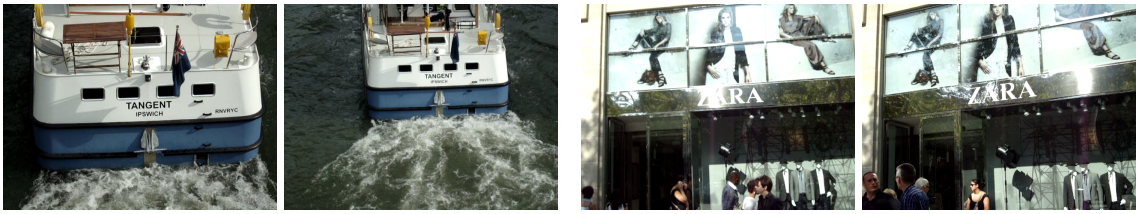
Video	#XMLs	#Text Objs.	Resolution (pixels)
v1	800	2	640×480
v2	1089	2	640×480
v4	400	3	640×480
v5	1250	1	640×480
v3	206	18	640×480
v6	1600	38	640×480

Table 11.1: Text tracking video datasets.

11.2 Output file format

We manually annotated the videos to produce the reference solution ("groundtruth"), consisting of a set of text regions $G^{(i)}$ for each frame $\mathbb{V}^{(i)}$ and a tracking relation $\varrho^{(i)}$ that connects images

(a) v1 - ship identification under perspective changes (b) v2 - store name suffering deformation changes



(c) v4 - boat identification zooming-out (d) v5 - store name going under lighting changes

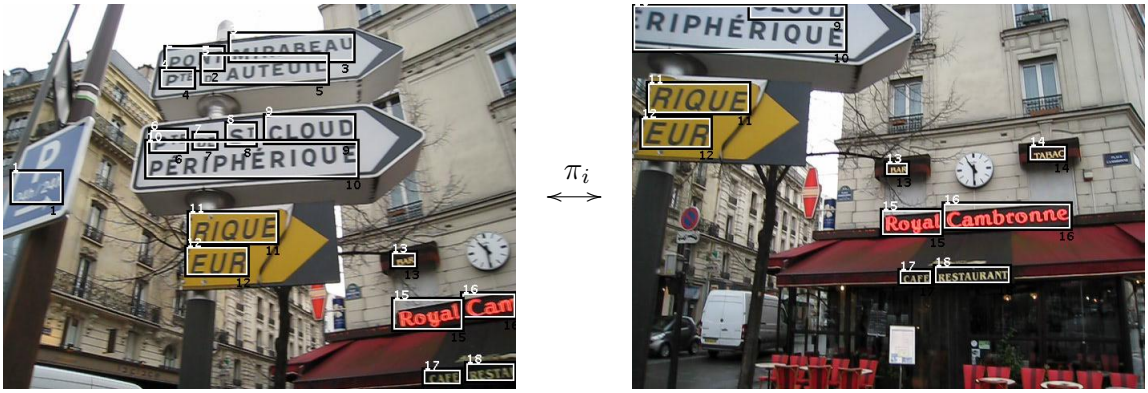
(e) v6 - camera fixed in car recording stores façades

Figure 11.1: Sample frames from our text tracking dataset.

of the same object between successive frames. On each video frame we marked all text lines that could be clearly read and had at least two letters.

For evaluation purposes, the output of each text tracking algorithm, as well as the reference solution, was saved as an XML file, whose structure is similar to that of the ICDAR challenge [55, 57] reference solutions.

Each frame is represented by an <image> field. Each text region in that frame is encoded as a <taggedRectangle> sub-field of <image>. The region's geometry is represented by its parameters **x**, **y** (p_{f}), **w** ($u_{\mathrm{f}}.x$) and **h** ($v_{\mathrm{f}}.y$). Each region also has: an integer parameter **id**, which uniquely identifies the corresponding text object throughout the whole video; a string parameter **text** which contains the part of the text that is visible in the frame; and a fractional parameter **vfr**, between 0.0 and 1.0, which indicates how much of the text line's vertical extent lies between the top and bottom edges of the frame. The **text** and **vfr** fields are non-empty only in the reference file. See Figure 11.2.



π_i

$T_i = 182$

```
<?xml version="1.0" encoding="UTF-8"?>
<tagset>
 <image>
  <imageName>000182</imageName>
  <resolution x="640" y="480"/>
  <taggedRectangles>
    <taggedRectangle x="149.0" y="0.0"   w="118.0" h="19.0" id="9"  text="Cloud"    vfr="0.5"/>
    <taggedRectangle x="4.0"   y="4.0"   w="263.0" h="55.0" id="10" text="RIPHERIQUE" vfr="1"/>
    <taggedRectangle x="22.0"  y="101.0" w="125.0" h="36.0" id="11" text="RIQUE"      vfr="1"/>
    <taggedRectangle x="14.0"  y="144.0" w="85.0"  h="36.0" id="12" text="EUR"        vfr="1"/>
    <taggedRectangle x="319.0" y="200.0" w="26.0"  h="14.0" id="13" text="BAR"        vfr="1"/>
    <taggedRectangle x="498.0" y="179.0" w="44.0"  h="17.0" id="14" text="TABAC"      vfr="1"/>
    <taggedRectangle x="314.0" y="259.0" w="73.0"  h="30.0" id="15" text="Royal"      vfr="1"/>
    <taggedRectangle x="391.0" y="251.0" w="157.0" h="30.0" id="16" text="Cambronne"  vfr="1"/>
    <taggedRectangle x="336.0" y="334.0" w="38.0"  h="16.0" id="17" text="CAFE"       vfr="1"/>
    <taggedRectangle x="380.0" y="329.0" w="92.0"  h="20.0" id="18" text="RESTAURANT" vfr="1"/>
  </taggedRectangles>
 </image>
</tagset>
```

T_i

Figure 11.2: Example of a reference XML tracking output file. Video v3.

11.3 Metrics

We evaluated the accuracy of each text tracking algorithm on three aspects: *text region detection*, *text object detection* and *text object tracking*.

11.3.1 Region detection accuracy

Region detection accuracy refers to the ability of the algorithm to detect the visible text objects *in each frame*. Note that in many applications it is not enough to detect the object once, but

it is necessary to detect it in all frames where it appears. Observe, that for TRACK2, TRACK3 and TRACK4 the region detection accuracy reflects the performance of the TEXTDETECT procedure, as well as that of the PREDICT or TRACKREGIONS.

To measure the region detection accuracy we compare each set $T^{(i)}$ of detected regions with the corresponding reference set $G^{(i)}$, using averaging approach III described in Section 6.1.2. Namely, the precision (p) and recall (r) scores for text region detection are given by

$$p = \frac{\sum_{i=0}^{n-1} m(T^{(i)}, G^{(i)})}{\sum_{i=0}^{n-1} \#T^{(i)}} \qquad r = \frac{\sum_{i=0}^{n-1} m(G^{(i)}, T^{(i)})}{\sum_{i=0}^{n-1} \#G^{(i)}} \tag{11.1}$$

where m is the region set similarity function defined by formula (6.3). The f-score is the harmonic mean of p and r, as in formula (6.5).

Observe that the above metrics do not take into account the possibility of text occlusions and give partly visible texts the same importance as fully visible ones. One could use the fields <text> and <vfr> to reduce the weights of such regions. However, in the videos considered here, the vast majority of the legible texts are totally visible in most frames and therefore the above adjustments would have little effect.

11.3.2 Object detection accuracy

In some applications it is sufficient to detect each text object in at least one frame. For those situations, we define a separate set of metrics to evaluate the accuracy of each text tracking algorithm in detecting the occurrences of the text *objects* (as opposed to text *regions*).

We will denote by G^* the set of all distinct text objects that are listed in the reference file (basically, the set of all distinct **id** parameter values) and by T^* the set of text objects implied by the computed tracking relations $\pi^{(1)}, \pi^{(2)}, \ldots, \pi^{(n)}$. For each object r^* (in G^* or T^*) we denote by r_i the corresponding text region in frame i (or *nil* if the object is not visible in that frame).

An object r^* in one object set R^* (groundtruth or estimated, respectively) is assumed to be present in the region sets $Z^{(0)}, Z^{(1)}, \ldots, Z^{(n-1)}$ (estimated or groundtruth, respectively) if and only if there is some region z in some set $Z^{(i)}$ which matches the corresponding region r_i^*. More precisely

$$m(r^*, Z) = \max\{m(r_i^*, Z^{(i)}) : i = 0, \ldots, n-1\} \tag{11.2}$$

where $m(r_i^*, Z^{(i)})$ is the region matching function defined by formula (6.2). The extension of equation (11.2) to a set R^* is

$$m(R^*, Z) = \sum_{r^* \in R^*} m(r^*, Z) \tag{11.3}$$

Finally, the precision (p) and recall (r) scores for text object detection are given by

$$p = \frac{m(T^*, G)}{\#T^*} \qquad r = \frac{m(G^*, T)}{\#G^*} \tag{11.4}$$

and the f-score is also defined by formula (6.5).

Suppose that only some slice of a text object is visible in some frame, e.g. because it is partially occluded or it will enter in the frame gradually. Also suppose that this slice is perfectly detected. If the reference file contains that partial region annotated, then that object will be considered 100% detected in the video. To avoid this effect, one should exclude partially text occurrences from the reference file. On the other hand, if a text object r^* is always entirely detected into two or more regions, while it is given as a single region in the reference file (or vice-versa) then the recall score $m(r^*, Z)$ for that object will be much less than 1. This last observation is also valid to the region detection accuracy metric and the ICDAR score metric (see Section 6.1.2).

11.3.3 Text tracking accuracy

For text tracking applications, an essential aspect of the algorithm is how accurately it identifies regions in successive frames as belonging to the same object. A tracking failure occurs whenever two regions that belong to the same text object in successive frames $\mathbb{V}^{(i-1)}, \mathbb{V}^{(i)}$ do not appear in the tracking relation $\pi^{(i)}$.

Metrics for tracking accuracy were discussed by Smith *et al.* [79] in 2005. For this thesis, we evaluate the tracking accuracy by comparing each tracking relation $\pi^{(i)}$ to the reference relation $\varrho^{(i)}$. For this purpose, we define a geometric similarity measure $m((r, r'), (s, s'))$ for two pairs of regions, where r, s are supposed to be in one frame and r', s' in the next frame. We use the geometric mean of the region similarities, namely

$$m((r, r'), (s, s')) = \begin{cases} 1 & \text{if } m(r, s) \geq \tau \text{ and } m(r', s') \geq \tau \\ 0 & \text{otherwise} \end{cases} \tag{11.5}$$

where τ is a threshold and $m(r, s)$ and $m(r', s')$ are computed by formula (6.1). As in Section 6.1.2, we extend formula (11.5) to a pair (r, r') and a set of pairs $\lambda \subseteq Z \times Z'$ by

$$m((r, r'), \lambda) = \max\{m((r, r'), (s, s')) : (s, s') \in \lambda\} \tag{11.6}$$

The extension for two sets of pairs $\sigma \subseteq Y \times Y'$ and $\lambda \subseteq Z \times Z'$ (tracking relations), analogous to equation (6.3), is then

$$m(\sigma, \lambda) = \sum_{(r,r') \in \sigma} m((r, r'), \lambda) \tag{11.7}$$

Therefore, the tracking precision (p) and recall (r) scores are given by

$$p = \frac{\sum_{i=1}^{n-1} m(\pi^{(i)}, \varrho^{(i)})}{\sum_{i=1}^{n-1} \#\pi^{(i)}} \qquad r = \frac{\sum_{i=1}^{n-1} m(\varrho^{(i)}, \pi^{(i)})}{\sum_{i=1}^{n-1} \#\varrho^{(i)}} \tag{11.8}$$

The f-score is computed from p and r according to formula (6.5).

11.4 Settings

In all tests with TRACK2, TRACK3 and TRACK4, we used the same particle-filter tracker, described in Section 10.1, with linear Kalman prediction. The number of particles was fixed at 100. The tracker was set to use the T-HOG for content classification and similarity. However, the experimental analysis of Section 5.5, that led us to use $n_x = 1$ for text recognition, is not relevant for this last use of the T-HOG, that is, contents similarity by region signature. Indeed, a division into 6×6 cells was more effective in this role than the 1×7 arrangement that was optimal for text recognition. The latter arrangement yields a descriptor that is very sensitive to vertical displacements of the candidate region, but largely insensitive to horizontal displacements. As a consequence, the path computed by the tracker has substantial jitter in the horizontal direction. By using an equal number of vertical and horizontal cuts, we obtain a descriptor that is sensitive to both vertical and horizontal displacements of the candidate region.

In our experiments, we combined each text tracking algorithm with two text detectors, IDEAL and SNOOPERTEXT [66]. The output of the IDEAL detector is the groundtruth set $G^{(i)}$ restricted to the sub-domain parameter $\mathfrak{D}$. Tests with this detector evaluate the best possible performance of the text tracking algorithm proper. SNOOPERTEXT is the state-of-the-art detector discussed in Section 6.1.1. Tests with this detector illustrate the typical performance and failure modes of each tracking algorithm in real applications. The SNOOPERTEXT algorithm was set to use STC=3 (see Section 6.1).

We did not attempt to tune the parameters for individual videos. The same settings, either in detection and tracking, were used in all strategies and in all videos discussed here.

11.5 Results

Tables 11.2 and 11.3 summarized the results of our four tracking strategies using the IDEAL and SNOOPERTEXT detectors, respectively. The algorithm's accuracy was measured with the metrics described in Section 11.3 and the ICDAR score described in Section 6.1.2.

	TRACK1			TRACK2			TRACK3			TRACK4		
Region detection accuracy												
Video	p	r	f	p	r	f	p	r	f	p	r	f
v1	1.00	1.00	1.00	0.94	0.92	**0.93**	0.93	0.88	0.91	0.94	0.89	0.91
v2	1.00	1.00	1.00	0.83	0.86	0.84	0.84	0.82	0.83	0.88	0.88	**0.88**
v4	1.00	1.00	1.00	0.80	0.86	0.82	0.80	0.86	**0.83**	0.80	0.87	**0.83**
v5	1.00	1.00	1.00	0.92	0.92	0.92	0.93	0.89	0.91	0.93	0.93	**0.93**
v3	1.00	1.00	1.00	0.76	0.79	0.77	0.82	0.64	0.72	0.81	0.75	**0.78**
v6	1.00	1.00	1.00	0.57	0.66	0.61	0.77	0.53	0.63	0.77	0.66	**0.71**
ICDAR detection score												
Video	p	r	f	p	r	f	p	r	f	p	r	f
v1	1.00	1.00	1.00	0.93	0.92	**0.92**	0.89	0.88	0.88	0.92	0.89	0.90
v2	1.00	1.00	1.00	0.85	0.86	0.85	0.82	0.82	0.82	0.88	0.88	**0.88**
v4	1.00	1.00	1.00	0.80	0.85	0.82	0.81	0.86	**0.83**	0.82	0.87	**0.83**
v5	1.00	1.00	1.00	0.93	0.93	**0.93**	0.91	0.90	0.90	0.93	0.93	**0.93**
v3	1.00	1.00	1.00	0.76	0.78	**0.77**	0.80	0.65	0.70	0.81	0.75	**0.77**
v6	1.00	1.00	1.00	0.69	0.73	0.70	0.70	0.61	0.64	0.80	0.73	**0.75**
Object detection accuracy												
Video	p	r	f	p	r	f	p	r	f	p	r	f
v1	1.00	1.00	1.00	0.98	1.00	0.99	1.00	1.00	1.00	1.00	1.00	1.00
v2	1.00	1.00	1.00	0.87	1.00	0.93	0.81	1.00	0.89	0.89	1.00	**0.94**
v4	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
v5	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
v3	1.00	1.00	1.00	0.88	1.00	0.94	0.93	1.00	**0.97**	0.94	1.00	**0.97**
v6	1.00	1.00	1.00	0.75	1.00	0.85	0.93	1.00	**0.96**	0.93	1.00	**0.96**
Text tracking accuracy												
Video	p	r	f	p	r	f	p	r	f	p	r	f
v1	1.00	1.00	1.00	1.00	0.97	**0.98**	1.00	0.94	0.97	1.00	0.95	**0.98**
v2	1.00	1.00	1.00	0.98	1.00	0.99	0.98	0.92	0.95	1.00	0.99	**1.00**
v4	1.00	1.00	1.00	0.93	1.00	**0.96**	0.93	1.00	**0.96**	0.92	1.00	**0.96**
v5	1.00	1.00	1.00	1.00	1.00	**1.00**	1.00	0.95	0.98	1.00	0.99	**1.00**
v3	1.00	1.00	1.00	0.85	0.86	0.85	0.94	0.73	0.83	0.92	0.86	**0.89**
v6	1.00	1.00	1.00	0.52	0.68	0.59	0.91	0.62	0.74	0.89	0.77	**0.83**

Table 11.2: Performance of the tracking algorithms using the IDEAL detector. The boldface values are the maxima in each row among TRACK2, TRACK3 and TRACK4.

	TRACK1			TRACK2			TRACK3			TRACK4		
Region detection accuracy												
Video	p	r	f	p	r	f	p	r	f	p	r	f
v1	0.63	0.49	0.55	0.48	0.79	0.59	0.76	0.78	**0.77**	0.75	0.80	**0.77**
v2	0.64	0.58	0.61	0.72	0.84	**0.77**	0.80	0.74	**0.77**	0.80	0.75	**0.77**
v4	0.77	0.56	0.65	0.83	0.79	0.81	0.84	0.76	0.80	0.86	0.81	**0.84**
v5	0.89	0.37	0.52	0.52	0.65	0.58	0.93	0.75	0.83	0.93	0.84	**0.88**
v3	0.80	0.52	**0.63**	0.64	0.56	0.60	0.72	0.38	0.50	0.72	0.49	0.58
v6	0.67	0.29	0.40	0.46	0.38	0.42	0.62	0.24	0.34	0.66	0.34	**0.45**
ICDAR detection score												
Video	p	r	f	p	r	f	p	r	f	p	r	f
v1	0.51	0.49	0.48	0.48	0.79	0.59	0.73	0.78	**0.75**	0.72	0.80	**0.75**
v2	0.61	0.59	0.60	0.75	0.84	**0.78**	0.75	0.74	0.75	0.76	0.75	0.76
v4	0.67	0.56	0.59	0.77	0.76	0.76	0.73	0.71	0.72	0.82	0.80	**0.81**
v5	0.36	0.36	0.36	0.52	0.65	0.56	0.77	0.76	0.76	0.85	0.84	**0.85**
v3	0.80	0.52	**0.62**	0.64	0.57	0.60	0.72	0.39	0.50	0.72	0.48	0.58
v6	0.49	0.38	0.41	0.50	0.48	**0.47**	0.47	0.35	0.37	0.56	0.42	0.45
Object detection accuracy												
Video	p	r	f	p	r	f	p	r	f	p	r	f
v1	0.61	0.97	**0.75**	0.12	0.97	0.21	0.27	0.94	0.42	0.28	0.98	0.44
v2	0.70	1.00	0.82	0.49	0.98	0.65	0.88	0.97	**0.92**	0.85	0.96	0.90
v4	0.73	0.86	0.79	0.78	0.97	0.87	0.95	0.95	0.95	0.99	0.99	**0.99**
v5	0.85	0.99	0.91	0.15	0.98	0.26	0.98	0.98	0.98	0.99	0.99	**0.99**
v3	0.74	0.90	**0.81**	0.62	0.91	0.74	0.81	0.72	0.76	0.81	0.72	0.76
v6	0.56	0.70	**0.62**	0.45	0.68	0.54	0.67	0.48	0.56	0.69	0.50	0.58
Text tracking accuracy												
Video	p	r	f	p	r	f	p	r	f	p	r	f
v1	0.92	0.32	0.47	0.56	0.92	0.70	0.90	0.92	**0.91**	0.87	0.92	0.90
v2	0.97	0.62	0.76	0.85	0.99	0.91	0.98	0.89	**0.93**	0.96	0.89	**0.93**
v4	0.98	0.66	0.79	0.97	0.92	0.94	0.98	0.89	0.94	0.99	0.93	**0.96**
v5	1.00	0.25	0.40	0.56	0.69	0.62	1.00	0.81	0.89	1.00	0.91	**0.95**
v3	0.98	0.48	0.65	0.79	0.69	**0.74**	0.90	0.46	0.61	0.91	0.60	0.73
v6	0.82	0.27	0.40	0.44	0.41	0.42	0.65	0.26	0.37	0.79	0.40	**0.53**

Table 11.3: Performance of the tracking algorithms using the SNOOPERTEXT detector. The boldface values are the maxima in each row among all four algorithms.

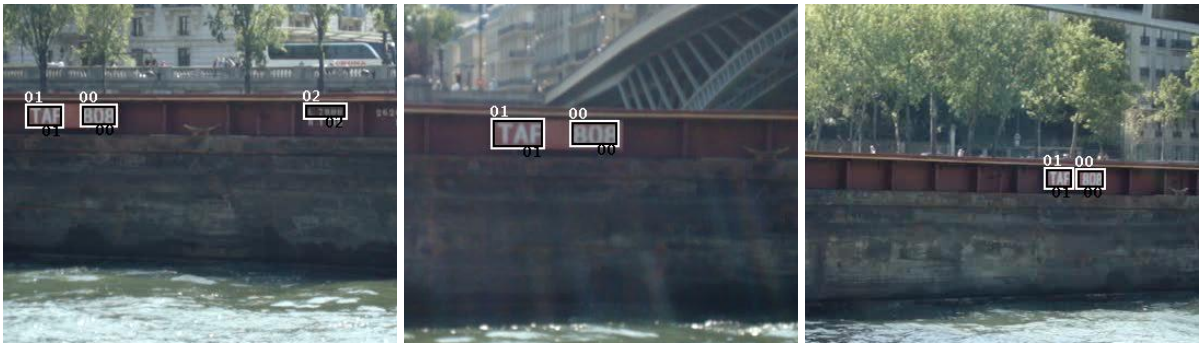
11.6 Discussion

As Table 11.2 shows, if the text detector is fast and very accurate, then TRACK1 is the best of all four strategies. In our test videos, the simple motion prediction we used allowed the text to be correctly matched in all frames. However, this is not true for practical text detectors which are still very far from the ideal, as shown in Table 11.3.

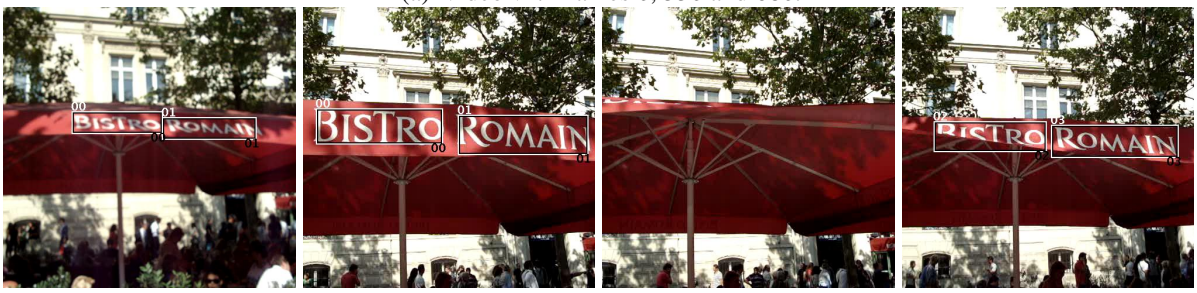
In realistic situations, TRACK4 was the best strategy overall. Some examples of its output are shown in Figures 11.3 and 11.4. Note that TRACK3 also performed very well and is an alternative if we consider real-time applications. Some problems of both strategies in videos v3 and v6 are due to over-segmentation problems (see Section 11.6.3).

The TRACK1 and TRACK2 algorithms are much more computationally expensive than TRACK3 and TRACK4. Typically, running SNOOPERTEXT on a single frame takes 30 times longer than running TRACKREGIONS.

The cost of TRACKREGIONS ($\approx$ 1–2 seconds per object) can be significantly reduced ($\approx$ 0.2–0.9 milliseconds [67] per frame), with small loss of precision, by using the SNOOPERTRACK [67] observation model that compute only one T-HOG classification per object. The observation model used in this section, equation (10.13), compute one T-HOG classification per particle. Moreover, there is still plenty of room for improvement, since our algorithms were coded in JAVA and without any optimization.



(a) Video v1. Frames 0, 330 and 680.



(b) Video v2. Frames 17, 743, 768 and 839.

(c) Video v4. Frames 0, 110 and 280.

(d) Video v6. Frames 1290, 1304, 1310, 1320, 1330 and 1340.

Figure 11.3: Examples of tracking by TRACK4 strategy. The labels indicates the text occurrences found by TRACK4 so far. Note that only at video v2 in (b) the same text projection of 'BISTRO' and 'ROMAIN' received two different labels (due to a total occlusion).

(a) Video v3. Frames 0, 100 and 118.

(b) Video v6. Frames 1107, 1126 and 1130.

Figure 11.4: Examples of tracking failures by TRACK4 strategy. In (a), some text lines were lost due to excessive tilting. In (b), the tracking was able to follow region '20' even while it was partially occluded by the pole, but lost region '19'.

11.6.1 Impact of text detection errors

Text regions are identified in a frame in two ways, either by using the text detector, at least once, or by using a tracker. It turns out that the latter is substantially more accurate than the former both in precision and recall. Therefore, it is not surprising that the region detection score generally improves from TRACK1 to TRACK4.

Observe that the regions missed by the detector (false negatives) directly impact the region recall of TRACK1, whereas the other three algorithms will often correct a detector's failure by tracking the text after it was detected in some previous frame.

On the other hand, the recall score for the object detection accuracy metric is better in TRACK1 and TRACK2, specially in videos v3 and v6, because some objects that were missed by strategies TRACK3 and TRACK4 were found by those two methods. See Figure 11.5. This is explained because TRACK1 and TRACK2 scans most of the pixels in every frame, thus, a region lost by TRACK3 and TRACK4 due to irregular illumination, partial occlusions, or perspective is more likely to be found due to this scanning.

Furthermore, detection errors also impact the tracking accuracy. The low tracking recall scores of TRACK1 are mainly due to the detector returning very different text regions for the same object in successive frames, either by including background pixels, by losing parts of the text, by improper joining and splitting of text regions, or by complete detection failure. See Figure 11.6. Such errors often cause the MATCH procedure to fail, so that a single text trajectory is often split into several separate trajectories. These problems typically occurs in any algorithm that relies on the text detector to define the regions, such as the tracker of Merino *et al.* [59].

Figure 11.5: Effect of irregular illumination: at left, strategies TRACK3 and TRACK4 had only two chances, and failed, to detect the word 'CORREIOS' at key-frame 840 (top) and 870 (bottom), while strategies TRACK1 and TRACK2 detected some part of this word at frame 860 (middle). Note (at right) the variation in the illumination.

Figure 11.6: TRACK1 outputs using the SNOOPERTEXT detector. Examples of tracking failures due to over bounding (top), video v3; improper splitting/joining and under bounding (middle), video v2; and detection failure (bottom), video v5.

11.6.2 Impact of false positive detections

In general, false positives returned by the detector directly affect the region and object detection precision of TRACK1 and TRACK2. Moreover, TRACK2 may start track the false positives, thus magnifying their impact. See Figure 11.7 (a). This explains its lower precision score for region detection. This problem is compensated in TRACK3 and TRACK4, see Figure 11.7 (b), because false positives can only be found on the key frames. As a result, their precision scores for region detection is even higher than those of TRACK1.

(a) TRACK2

(b) TRACK4

Figure 11.7: Effect of false positives: (a) tracking of several false positives by the TRACK2 strategy; (b) strategy TRACK4 found and tracked only a subset of these false positives.

11.6.3 Impact of over-segmentation

The low scores precision scores of TRACK2 in both tables, specially on videos v3 and v6, are partly due to over-segmentation of text regions. Specifically, text lines that enter the frame gradually are often fragmented into separate regions containing 2-3 letters each. This splitting occurs because parts of the text that were detected in previous frames are identified by tracking and hence excluded from the scope of the text detector. Therefore, each group of 2-3 characters is detected as soon it becomes visible, and then is tracked independently of other blocks. See Figure 11.8(c). The same problem occurs when text objects are temporally occluded by foreground objects (tree, poles, *etc*) or obscured by shadows or highlights.

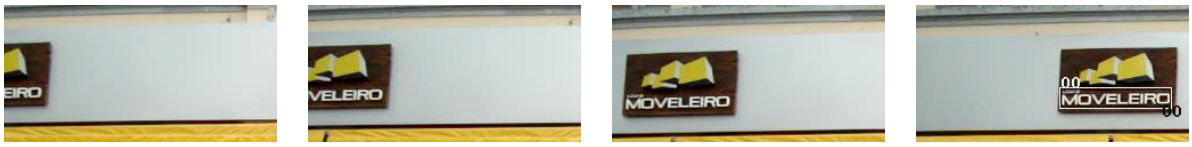
The over-segmentation also affect the tracking, since we have less information available to match the region.

The TRACK3 and TRACK4 strategies are less sensitive to this problem because the detector is run only at key-frames, so new texts are often detected when they are already totally visible. See Figure 11.8.

(a) Key-frame: 990 (b) Key-frame: 1020

Frame: 997 Frame: 1001 Frame: 1006 Frame: 1020

(c) TRACK2



Frame: 997 Frame: 1001 Frame: 1006 Frame: 1020

(d) TRACK3

Frame: 997 Frame: 1001 Frame: 1006 Frame: 1020

(e) TRACK4

Figure 11.8: Impact of over-segmentation in video v6: (c) TRACK2 strategy splits the word "MOVELEIRO" into 2-3 characters slices as it entered in the scene; (d) in the TRACK3 and TRACK4 strategies, the over-segmentation did not occur because the detector was run only on key-frames (a,b). In the intervening frames, TRACK3 missed the text completely (d) while TRACK4 recovered it by tracking backwards (e).

11.6.4 Impact of backward tracking

Tables 11.2 and 11.3 demonstrate the advantages of backward tracking (TRACK4 strategy) over simple forward tracking (TRACK3 strategy). In general, considering both tables, the use of backward tracking improves the tracking accuracy f-score by $\approx 5\%$, the ICDAR f-score by $\approx 5\%$, and the region detection f-score by $\approx 4\%$. In particular, considering only videos v3 and v6 where the text objects frequently enter in the frame gradually, the backward tracking improves the tracking f-score by $\approx 11\%$, the ICDAR f-score by $\approx 9\%$, and the region f-score by $\approx 8\%$. It also improves the detection scores of objects that are temporally occluded/obscured (as in Figure 11.9), and in all cases where a text object has been lost and then found again.

(a) TRACK3

(b) TRACK4

Figure 11.9: Impact of backward tracking in video v5 with temporary text occlusions. Using the SNOOPERTEXT detector, both TRACK3 and TRACK4 found the text 'ZARA' in frame 0 and tracked it forward until frame 951, where it was lost due to glare. The text detector found it again, 219 frames later, in key-frame 1170. The TRACK4 was then able to recover it in frames 1065–1169 by backward tracking from that key-frame.

11.7 Concluding remarks

Apart from the detector's accuracy, there are some key-points to improve in our text tracking strategies: connection of over-segmented text regions; use of general rectangles, instead of only axis aligned rectangles; exploit the spatial coherence of text motions as would be produced by camera motion; and text label unification in situations of total occlusions.

The camera motion, estimated by e.g. optical flow, as described by Minetto *et al.* [63], may also lead to significant reduction in time in TRACK2 strategy by excluding from the set $\mathfrak{D}$ any pixels from the background that were scanned by the detector in the previous frame and remain visible in the current frame.

Part III

Tracking of 3D Rigid Objects

Chapter 12

Introduction

In this part of the thesis we consider the *feature-based tracking problem of a tri-dimensional rigid object*. Namely, the tracking of a solid object (such as a building, truck, package, *etc*.) by following a set of known two-dimensional features attached to its surface. In some cases the entire object of interest may be a single feature. We consider features that are mostly flat, rigid, opaque, and firmly attached to the object of interest along the time.

The features can be either intrinsic or accidental (such as letters, labels, wall outlets, logos, *etc*.), or fiducial marks attached to the object specifically for the purpose. See Figure 12.1.

Figure 12.1: Trackable features (a) accidental and (b) intentional.

The input data for this problem consists of a digital video, a list of the features to be tracked, their positions on the object, and the approximate position of the projection of each feature in the first frame. The output consists of the position of the object in space relative to the camera and other relevant camera parameters (such as zoom factor) for each frame.

12.1 Motivation

Tracking the motion of rigid objects is a general video analysis problem with a wide range of applications, including robotics handling and assembly, vehicle fleet management and tracking, autonomous navigation. For example, the camera calibration of a scene by real time tracking of two-dimensional physical markers is the core of most software for building augmented reality (AR) applications (overlay of virtual imagery on the real world). See Figure 12.2.

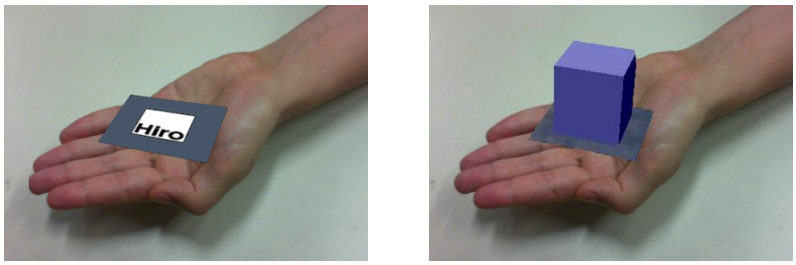


Figure 12.2: Augmented Reality.

12.2 Our contributions

Our major contribution to this problem is the development of an algorithm nicknamed AFFTRACK [65]. AFFTRACK uses a synergistic combination of two main procedures, a multi-scale *feature finder* (FF) that locates the positions of the features (markers) on each frame, and a flexible *camera calibrator* (CC) that computes the camera parameters from those positions.

A key aspect of AFFTRACK is the use of a *confidence weight* to express the reliability of each feature on each frame. These weights are initially computed from the quality of the match returned by the feature finder, and then iteratively adjusted by the calibrator depending on the consistency between the feature's reported position and the position of all other features [64]. Thus, instead of RANSAC's [30] binary accept/reject criterion, our algorithm uses a "fuzzy" classification of outliers similar to that of MLESAC [82].

Another key aspect of AFFTRACK is that the camera parameters calibrated in previous frames are used to guess the location and shape of each feature in the next video frame, as required by the feature finder. Whenever a feature has been successfully located, our algorithm adjusts its expected appearance to account for changes in the lighting and contrast.

AFFTRACK does not require the features to be visible or successfully identified in all frames (not even on the first one), as long as those that are visible are sufficient to determine the camera's unknown parameters. As a result of the two-way interaction between the FF and the

CC, it can usually recover a feature that was occluded or mis-identified by the finder, as soon as it becomes visible again. Increasing the number of tracked features beyond the theoretical minimum generally improves the reliability and accuracy of the calibration.

Our emphasis is on robustness, accuracy, and flexibility within the stated goals, rather than speed. Nevertheless the algorithm is quite efficient, because the accurate guessing of feature positions, provided by the calibrator, allows the finder to use smaller templates and windows. We hope that AFFTRACK will be useful in many real-world applications. Moreover, our weighted camera calibration module can be used in any system where some confidence about the data is available. AFFTRACK may also be useful as a building block of other computer vision algorithms.

Our AFFTRACK implementation (in C) and the test datasets used in this part are available at our site [61].

12.3 Statement of the problem

As in Part II, the input is a video $\mathbb{V}$, assumed to consists of a sequence of n images $\mathbb{V}^{(0)}, \mathbb{V}^{(1)}, \ldots, \mathbb{V}^{(n-1)}$ with common domain $\mathfrak{D}$, equally spaced in time; and a list of m two-dimensional features. Each feature is defined by a canonical image $\mathbb{M}[k]$, a mask $\mathbb{W}[k]$, and the feature's *object geometry*, namely, a point $p_s[k] \in \mathbb{R}^3$, and two orthogonal vectors $u_s[k], v_s[k] \in \mathbb{R}^3$ that define the feature's position, orientation and size on the object; where $k = 0, 1, \ldots, m-1$. See Figure 12.3 (a–c).

(a) (b) (c) (d)

Figure 12.3: The object geometry parameters of a feature in 3D space (a), its canonical image (b), its mask (c), and its image geometry (d).

The parameters $p_s[k], u_s[k], v_s[k]$ are assumed to be relative to the object's own coordinate system (see Section 14.1) and are therefore independent of the camera and object positions in space. The canonical image $\mathbb{M}$ do not need to be particularly accurate; generally it is assumed to be a frontal projection of the feature under ideal lighting conditions. The common domain of $\mathbb{M}$ and $\mathbb{W}$ is assumed to be the rectangle $p_s \pm u_s \pm v_s$ on the object's surface.

We assume that, in each frame $\mathbb{V}^{(i)}$, the domain $p_{\mathrm{s}}[k] \pm u_{\mathrm{s}}[k] \pm v_{\mathrm{s}}[k]$ of each feature is a parellogram, defined by its center ${p_{\mathrm{f}}}^{(i)}[k]$ and ${u_{\mathrm{f}}}^{(i)}[k]$ and ${v_{\mathrm{f}}}^{(i)}[k]$. See Figure 12.3 (d). The user must also provide the approximate position ${p_{\mathrm{f}}}^{(0)}[k]$ of each feature in the first frame and a confidence weight $w^{(0)}[k]$ to be explained later.

The goal is to determine the set $\mathbb{C}^{(i)}$ of parameters that define the position of the object relative to the camera (see Chapter 14) for each frame $\mathbb{V}^{(i)}$, by locating the positions of those features on the frame $\mathbb{V}^{(i)}$.

Note that to solve this tracking problem it is not necessary to have full knowledge of the object's shape and appearance. Note also that the problem of tracking N rigid objects can be treated as N separate instances of this problem.

12.4 Challenges

The discussion of Section 7.4, in Part II, about tracking jitter, drift and loss also applies to this part. See Figures 12.4 and 12.5.

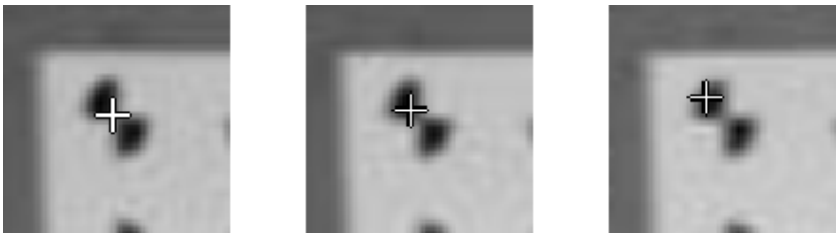


Figure 12.4: Tracking drift in recursive tracking.

Figure 12.5: Tracking loss in fixed template tracking.

12.4.1 Outliers

A problem that arises specifically in camera calibration is the handling of *outliers*, grossly incorrect positions that are occasionally reported by the feature finder. For example, if the feature

becomes occluded at any point, the feature finder may return some meaningless location within its search window. These errors are often very large and do not follow the normal distribution of errors due to camera noise.

In many camera calibration algorithms, all data pairs are assumed to have the same importance, which makes the result very sensitive to outliers in the data set $\mathbb{D}$. A substantial error in the reported position of a single feature may lead to a totally incorrect parameter vector $\mathbb{C}$, or even to failure of the calibration algorithm.

Chapter 13

Related work

The integration of feature tracking and camera calibration for 3D object tracking has been described in several recent articles, some of which are covered in the survey by Lepetit and Fua [45].

We mention in particular the tracker by Vaccheti *et al*. [85] which attempts to solve approximately the same problem as we do here. An advantage of their algorithm is that it automatically identifies trackable features by SIFT-like methods [52, 53] and extracts their template images from the video itself. However, their algorithm requires a complete geometric model of the object. It also requires the selection of certain key frames along the video, where the features to be tracked are all visible and the object's pose is approximately known. AFFTRACK only requires the features positions on the object, and does not require key-frame selection. While our algorithm requires the user to provide canonical images and object positions of the features beforehand, it does not require complete object models, it does not need also more than one camera to treat occlusions as in the work of Black *et al*. [14], or key frame selection, and can track an object in the presence of occlusions, video noise, and finder failures for an indeterminate number of frames, without any long-term drift.

Our algorithm can also handle video shots with variable-zoom and variable radial distortion unlike the work of Koller *et al*. [42] that needs these parameters fixed during a session.

Our algorithm is more general than H. Kato's widely used ARToolKit [40]. Besides allowing changes in focal length, AFFTRACK does not require that the features have a specific and easily-recognized shape as ARToolKit does. In fact, our algorithm can reliably track and use even features that have been reduced to a single-pixel dot. As in the work of Ababsa *et al*. [6], our algorithm also allows features to be located on different planes, with arbitrary orientations. Indeed, it relies on the presence of non-coplanar features to resolve certain calibration ambiguities (such as focal length vs. camera distance) that arise when using a single 2D pattern.

Another popular tracker is the implementation of the KLT algorithm [15] included in the OpenCV library [16] as the routine *cvCalcOpticalFlowPyrLK*. Although it is only a 2D feature

tracker, its output (the locations of the feature on each frame) can be fed to any camera calibrator, such as R. Tsai's algorithm [83], to create a 3D tracker. As we show in Chapter 18, this solution suffers from unavoidable drift that frequently leads to permanent tracking failure of moving objects.

In these solutions outliers are often handled by the RANSAC approach [30, 86]. Namely, the calibration is repeated a certain number M of times, each time using only a randomly chosen small subset of the data and choosing the best one. For each calibrated parameter vector $\mathbb{C}$, the RANSAC method counts the features whose reported positions match the predicted positions within a prescribed tolerance; and chooses the parameter set $\mathbb{C}$ that maximizes this count. While this approach is fairly robust, it may be rather expensive: the number M of trials needed to succeed with probability p is $\log(1-p)/\log(1-(1-\epsilon)^k)$, where ϵ is the probability of a data pair being an outlier, and k is the minimum number of data pairs needed for calibration (which is 7 for the full Tsai algorithm). For $p = 99.5\%$ and $\epsilon = 30\%$, we get $M = 61$. Note that in a video with N frames, we must have $(1-p) < 1/N$ to have a reasonable chance of tracking over the whole video. Moreover, the RANSAC approach gives the same importance to all data pairs which pass the acceptance test, independently of their quality, and ignores completely points that barely fail that test.

Chapter 14

Camera model

For most video and photo cameras, the geometric correspondence between a 3D object and a 2D photograph of it can be approximated quite well by the composition of a rigid *scene-to-camera coordinate transform*, a *perspective projection* onto the plane of the sensor array, a *radial distortion* that is symmetric around the camera's optical axis, and a *sensor sampling* mapping that relates coordinates on the projection plane to indices into the pixel array. For these steps, we use mathematical models and notation similar to those of Roger Tsai's paper [83]. All coordinates are in millimeters unless said otherwise.

14.1 Scene and camera coordinates

We assume a Cartesian *scene coordinate system* that is used to describe both the object of interest and the position of the camera relative to it. Depending on the application, the object of interest may be fixed (e.g. a building) or mobile (e.g. a vehicle); in the second case the scene coordinate system moves with the object.

We also use a *camera coordinate system* where the camera's equivalent pinhole lens is at the origin, the X axis points to the camera's left, the Y axis points to the camera's down, and the Z axis (the optical axis of the lens) points forward. See Figure 14.1.

In both systems, we assume that the coordinates are measured in millimeters. The scene-to-camera mapping is a rigid transformation that takes *scene coordinates* $p_\mathrm{s} = (x_\mathrm{s}, y_\mathrm{s}, z_\mathrm{s})$ of a point to its *camera coordinates* $p_\mathrm{c} = (x_\mathrm{c}, y_\mathrm{c}, z_\mathrm{c})$. It is given by

$$\begin{bmatrix} 1 \\ x_\mathrm{c} \\ y_\mathrm{c} \\ z_\mathrm{c} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ T_\mathrm{x} & R_\mathrm{xx} & R_\mathrm{xy} & R_\mathrm{xz} \\ T_\mathrm{y} & R_\mathrm{yx} & R_\mathrm{yy} & R_\mathrm{yz} \\ T_\mathrm{z} & R_\mathrm{zx} & R_\mathrm{zy} & R_\mathrm{zz} \end{bmatrix} \begin{bmatrix} 1 \\ x_\mathrm{s} \\ y_\mathrm{s} \\ z_\mathrm{s} \end{bmatrix} \tag{14.1}$$

The 4×4 *scene-to-camera matrix* above will be denoted by S in what follows. The lower right

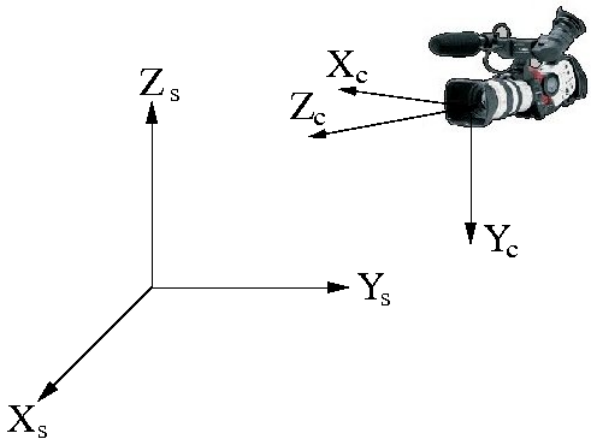


Figure 14.1: The scene coordinate system $(x_\mathrm{s}, y_\mathrm{s}, z_\mathrm{s})$ and the camera coordinate system $(x_\mathrm{c}, y_\mathrm{c}, z_\mathrm{c})$.

3×3 sub-matrix R of S is an orthonormal (rotation) matrix that determines the orientation of the camera axes relative to the scene axes. The column vector $T = (T_\mathrm{x}, T_\mathrm{y}, T_\mathrm{z})$ contains the camera coordinates of the scene system's origin. The corresponding vector $V = (V_\mathrm{x}, V_\mathrm{y}, V_\mathrm{z})$ in the inverse matrix S^{-1} contains the scene coordinates of the camera.

14.2 Projected coordinates

We also use a *projection coordinate system* lying on the *projection plane* that ideally contains the sensor array. Its origin lies on the camera's Z axis and its X and Y axes are parallel to the camera's X and Y axes. The perspective projection that takes camera coordinates $(x_\mathrm{c}, y_\mathrm{c}, z_\mathrm{c})$ to (*undistorted*) *projected coordinates* $p_\mathrm{u} = (x_\mathrm{u}, y_\mathrm{u})$, on the sensor plane, is

$$(x_\mathrm{u}, y_\mathrm{u}) = \frac{f}{z_\mathrm{c}}(x_\mathrm{c}, y_\mathrm{c}) \tag{14.2}$$

where the parameter f is the *focal length* of the lens which determines how the projected size of an object depends on its distance from the camera. A larger value of f is what distinguishes telephoto lenses from macro lenses, or high zoom settings from low zoom settings.

The radial distortion is a common lens artifact that causes lines that are straight in reality appear curved on the projection. It can be viewed as a mapping between the ideal undistorted projection plane coordinates $p_\mathrm{u} = (x_\mathrm{u}, y_\mathrm{u})$, as would be produced by a pinhole camera, and the *distorted projected coordinates* $p_\mathrm{d} = (x_\mathrm{d}, y_\mathrm{d})$ of the point as imaged by the real camera. We use the mapping

$$p_\mathrm{d} = \frac{1}{\sqrt{1 + 2\kappa r_\mathrm{u}^2}} p_\mathrm{u} \qquad p_\mathrm{u} = \frac{1}{\sqrt{1 - 2\kappa r_\mathrm{d}^2}} p_\mathrm{d} \tag{14.3}$$

where $r_{\mathrm{u}}^2 = x_{\mathrm{u}}^2 + y_{\mathrm{u}}^2$, $r_{\mathrm{d}}^2 = x_{\mathrm{d}}^2 + y_{\mathrm{d}}^2$, and κ is a parameter that controls the amount of radial distortion. This mapping agrees to fourth order on r with the widely used Tsai formula, $p_{\mathrm{u}} = (1 + \kappa r_{\mathrm{d}}^2) p_{\mathrm{d}}$, but has the advantage that it can be inverted by negating κ and swapping p_{u}, p_{d}, as shown; whereas in Tsai's model the mapping from p_{u} to p_{d} requires solving a cubic equation. A pinhole camera has $\kappa = 0$; the value of κ is positive for "pincushion" distortion, and negative for "barrel" distortion. Note that the coefficient κ may vary along the video when using lenses with variable zoom and/or variable focus.

Our formulas, like Tsai's, assume that the camera's Z axis is the optical axis of the lens. They can model the actual lens distortion only to the third order on the projected coordinates; that is, the approximation error is proportional to the fourth power of x_u/R_u and y_u/R_u when R_u is the image's radius on the projection plane.

14.3 Sensor coordinates

Each element of the sensor array measures the light that falls on a certain region of the projection plane. The digital image is then the 2D array of light measurements extracted from those sensors.

The *sensor sampling* mapping relates (distorted) projected coordinates $(x_{\mathrm{d}}, y_{\mathrm{d}})$ to image coordinates $(x_{\mathrm{f}}, y_{\mathrm{f}})$. It is determined by the position of the sensor chip relative to the lens's optical axis, by the spacing of the sensor elements, and by the order in which the samples are read out. Typically the sensor sampling mapping can be well described by the formulas

$$(x_{\mathrm{f}}, y_{\mathrm{f}}) = (C_{\mathrm{x}} + \frac{x_{\mathrm{d}}}{d_{\mathrm{x}}},\ C_{\mathrm{y}} + \frac{y_{\mathrm{d}}}{d_{\mathrm{y}}}) \qquad (x_{\mathrm{d}}, y_{\mathrm{d}}) = (d_{\mathrm{x}}(x_{\mathrm{f}} - C_{\mathrm{x}}),\ d_{\mathrm{y}}(y_{\mathrm{f}} - C_{\mathrm{y}})) \tag{14.4}$$

where $(C_{\mathrm{x}}, C_{\mathrm{y}})$, are the image coordinates of the camera's optical axis (in pixels), and $d_{\mathrm{x}}, d_{\mathrm{y}}$, are the effective spacings of the pixels (in millimeters) on the sensor array. In some cameras, one has $C_{\mathrm{x}} = n_{\mathrm{x}}/2$ and $C_{\mathrm{y}} = n_{\mathrm{y}}/2$. (We do not include Tsai's "horizontal uncertainty factor" s_{x} in this mapping, since it is redundant with d_{x} and is largely irrelevant for modern digitally-scanned sensor elements.). Finally, in most cameras, each scanline is captured at a different moment, either sequentially (progressive scanning) or with even or odd scanlines interleaved (interlaced scanning). We assume that the distortions due to those gradual scanning methods have been removed or are negligible.

14.4 Camera parameters

By *camera parameters* we mean the tuple $\mathbb{C}$ of all the coefficients that appear in formulas (14.1–14.4), namely S, p_{f}, κ, C_{x}, C_{y}, d_{x}, and d_{y}. For a given video, some of these parameters (such

as n_x and n_y) are known; some may be unknown, but known to be constant over all frames; and some may vary from frame to frame.

14.5 Feature projection

In general, the appearance of a feature on a digital image $\mathbb{I}$ is distorted by the perspective projection, so that the rectangle $p_s \pm u_s \pm v_s$ becomes a convex quadrilateral. However, assuming that the feature covers only a small fraction of the frame's domain, the perspective projection can be adequately modeled by an affine map A from the domain of $\mathbb{M}$ into the domain of $\mathbb{I}$; which is a first-order local approximation of the object-to-sensor coordinate transformation, defined by equations (14.1–14.4). Therefore, as observed in the statement of the problem, the projection can be approximated by a parallelogram, defined by its center p_f and two vectors u_f, v_f such that p_f, $p_f + u_f$ and $p_f + v_f$ are the projections of p_s, $p_s + u_s$ and $p_s + v_s$ on the image plane. In other words, the domain of $\mathbb{M}$ and $\mathbb{W}$ gets mapped to the parallelogram $p_f \pm u_f \pm v_f$. The parameters p_f, u_f and v_f are expressed in pixels, in the image or frame coordinate system.

14.6 Photometric parameters

The appearance of a feature in an image is also affected by changes in lighting, lens aperture, exposure time, *etc*. For a monochromatic image, these photometric factors can be approximated to first order by the formula $v \approx \alpha u + \beta$ where u is the pixel value in the canonical image $\mathbb{M}$ and v is the corresponding pixel value in the image $\mathbb{I}$. The parameter α is the feature's *relative contrast* and β is its *black level* for that image.

14.7 Blurring

The feature's appearance in the frame is usually blurred due to imperfect focusing, light scattering and diffraction, sensor-to-sensor leakage, de-noising and de-Bayering [12] filtering, *etc*. As a first approximation, we can model these effects by convolution of the ideal (infinitely sharp and detailed) image with a 2D Gaussian kernel of some radius σ, the *blurring* parameter. Note that the local geometric parameters (A), photometric parameters (α, β), and blurring parameter (σ) are usually different for each feature in the same frame, and for the same feature in different frames. In this thesis, we ignore the effect of bluring since it usually has little effect on the feature finder's performance.

Chapter 15

The AFFTRACK algorithm

In this chapter, we describe the AFFTRACK algorithm. Namely, the integration of the Camera Calibration (CC) and Feature Finder (FF) algorithms. The core of AFFTRACK is outlined as the algorithm 7 below. Its steps are described in detail in what follows.

Algorithm 7 AFFTRACK CALIBRATION FOR A FRAME $\mathbb{I}$

1. *Get an approximate camera calibration $\mathbb{C}'$ for this frame.*
2. *For each each feature index k do*
3. | *Get its estimated frame position $p'_{\mathrm{f}}[k]$ and initial visibility weight $w'[k]$.*
4. | *Get provisional photometric parameters $\alpha'[k], \beta'[k]$.*
5. | *Obtain the deformed template $\mathbb{G}'[k]$ and masks $\mathbb{V}'[k]$.*
6. | *If the feature is expected to be visible then*
7. | | *Run the FF to get its adjusted position $p''_{\mathrm{f}}[k]$ and its confidence weight $w''[k]$.*
8. | *else*
9. | | *Otherwise set $w''[k]$ to zero.*
10. |
11. *Run the CC to compute the final parameter tuple $\mathbb{C}$ for this frame.*
12. *For each each feature index k do*
13. | *Compute the final feature position $p_{\mathrm{f}}[k]$ and shape vectors $u_{\mathrm{f}}[k], v_{\mathrm{f}}[k]$.*
14. | *Generate the definitive feature template $\mathbb{G}[k]$ and mask $\mathbb{V}[k]$.*
15. | *Recompute $\alpha[k]$, $\beta[k]$.*
16. *Return $\mathbb{C}$.*

When necessary, we will use the superscript $^{(i)}$ to refer to the values computed by AFFTRACK (Algorithm 7) on frame number $\mathbb{V}^{(i)}$ (counting from 1). We will discuss first the general case $i \geq 3$. The first two frames require special handling, as explained in Sections 15.10 and 15.11.

15.1 Initial camera parameters

In step 1, we obtain an initial estimate $\mathbb{C}'$ of the camera parameters for this frame. Except for the first frame, we set $\mathbb{C}'^{(i)}$ to the calibrated parameters $\mathbb{C}^{(i-1)}$ that were computed in step 11 for the previous frame. (We have tried more sophisticated camera extrapolation algorithms, but found that the advantage of more accurate $\mathbb{C}'$ was negated by increased sensitivity to errors.)

15.2 Initial feature positions

In step 3, we obtain an initial estimate for the position $p'_{\mathrm{f}}[k]$ of each feature. Except for the first two frames, we use linear extrapolation of the feature's motion in the two previous frames: namely, we set ${p_{\mathrm{f}}}'^{(i)}[k]$ to $2{p_{\mathrm{f}}}^{(i-1)}[k] - {p_{\mathrm{f}}}^{(i-2)}[k]$; or, if $i = 2$, to ${p_{\mathrm{f}}}'^{(i-1)}[k]$. This extrapolation is necessary because the feature finder is likely to fail if the initial guess p'_{f} is off from the correct position by more than a fraction of the template's diameter. In typical videos, the position of a feature on successive frames often changes by much more than this tolerance. Therefore, simply setting ${p_{\mathrm{f}}}'^{(i)}[k]$ to ${p_{\mathrm{f}}}^{(i-1)}$ is is not a satisfactory solution. On the other hand, our tests indicate that higher-order extrapolation [88] (using three or more previous frames) is not worth the trouble. Note that this extrapolation uses the final positions ${p_{\mathrm{f}}}^{(j)}$ computed in step 13 from the calibrated parameters $\mathbb{C}^{(j)}$, rather than the positions ${p_{\mathrm{f}}}''^{(j)}[k]$ returned by the feature finder in step 7. Besides making the algorithm less sensitive to finder errors, this detail allows us to provide an estimate ${p_{\mathrm{f}}}'[k]$ even for features that were occluded, out-of-frame, or misplaced by the feature finder on previous frames. In this step, we set the initial confidence weight $w'[k]$ of each feature to 1 if the feature's guessed position $p'_{\mathrm{f}}[k]$ is inside the frame's domain, and 0 otherwise.

15.3 Photometric parameters

In step 4, we obtain estimates $\alpha'[k], \beta'[k]$ for the photometric coefficients of each feature. Except for the first frame, we use the coefficients $\alpha^{(i-1)}[k], \beta^{(i-1)}[k]$ that were computed for the previous frame in step 15.

15.4 Template projection

In step 5, we obtain the template $\mathbb{G}'[k]$ for each feature, deformed according to the estimated camera parameters $\mathbb{C}'$, and the corresponding mask image $\mathbb{V}[k]$. Except for the first frame, here too we use the templates and masks $\mathbb{G}^{(i-1)}$ and $\mathbb{V}^{(i-1)}$ computed for the previous frame in step 14.

15.5 Feature location

In step 7, the feature finder (described in Chapter 16) is called for each feature whose estimated position p'_f is within the frame $\mathbb{I}$ (that is, which has $w'[k] > 0$). The finder is given the frame image $\mathbb{I}$, the geometrically and photometrically corrected template $\alpha'[k]\mathbb{G}'[k] + \beta'[k]$, the corresponding mask $\mathbb{V}'[k]$, and the guessed position $p'_\mathrm{f}[k]$. It returns a found position $p''_\mathrm{f}[k]$, and a weight $w''[k]$ that quantifies the realiability of $p''_\mathrm{f}[k]$.

15.6 Weighted calibration

In step 11, the camera calibrator (described in Chapter 17) is called to compute the final camera parameters $\mathbb{C}$ for the current frame, from the known object positions $p_\mathrm{s}[k]$ of all features and their frame positions $p''_\mathrm{f}[k]$, returned by the finder in step 7. The calibrator also uses the confidence weights $w''[k]$ provided by the finder, so that features that were most clearly identified will have more influence on the calibration. In particular, features that were expected to be invisible get $w''[k] = 0$ and are automatically excluded from the calibration.

15.7 Final feature positions

In step 13, the parameter tuple $\mathbb{C}$ calibrated in step 11 is used to compute the final position $p_\mathrm{f}[k]$ of each feature in the current frame, from its object position $p_\mathrm{s}[k]$. These positions will be used to predict the feature positions in following frames, in step 3. In this step we also re-compute the frame geometry vectors $u_\mathrm{f}[k]$ and $v_\mathrm{f}[k]$ for each feature. We assume that the features are small relative to the scene-to-camera distance, so we compute these vectors from the object shape vectors $u_\mathrm{s}[k]$, $v_\mathrm{s}[k]$, by numerical differentiation of the object-to-sensor map equations (14.1–14.4). Namely, we map the object points $p_\mathrm{s} + u_\mathrm{s}$ and $p_\mathrm{s} - u_\mathrm{s}$ to frame points a_f and b_f, and then we set $u_\mathrm{f} \leftarrow (a_\mathrm{f} - b_\mathrm{f})/2$. Ditto for v_f.

15.8 Recomputing the synthetic templates

In step 14, we recreate the perspective-deformed template $\mathbb{G}[k]$ of each feature and its mask $\mathbb{V}[k]$ from the canonical template $\mathbb{M}[k]$ and its mask $\mathbb{W}[k]$, using the shape vectors $u_\mathrm{f}[k]$ and $v_\mathrm{f}[k]$ obtained in step 13. See Figure 15.1. The affine deformation $A[k]$ usually turns the rectangular domain of $\mathbb{M}[k]$ into a paralellogram. To account for this fact, we also compute a mask $\mathbb{V}[k]$, with the same size as $\mathbb{G}[k]$, that defines the valid pixels of the latter. See Figure 15.1(c). This mask is obtained by deforming a mask $\mathbb{W}$ that is 1 over all valid pixels of the canonical template $\mathbb{M}$.

(a) $\mathbb{M}$ (b) $\mathbb{G}$ (c) $\mathbb{V}$ (d) $\alpha\mathbb{G} + \beta$ (e) $\mathbb{I}$

Figure 15.1: (a) canonical template; (b) deformed template; (c) deformed weight mask; (d) photometrically corrected deformed template; (e) feature occurrence in frame.

15.9 Adaptive photometry correction

Finally, in step 15, we re-estimate the photometric coefficients $\alpha[k]$ and $\beta[k]$ of each feature. If the official position $p_\mathrm{f}[k]$ is inside the frame $\mathbb{I}$, the parameters are obtained by linear regression of the deformed template $\mathbb{G}[k]$, translated to the position $p_\mathrm{f}[k]$, against the frame image $\mathbb{I}$. Specifically, we perform least-squares fitting [75] of the model $\mathbb{I}(x', y') = \alpha\mathbb{G}(x, y) + \beta$, where (x, y) is a point of $\mathbb{G}$, (x', y') is the corresponding point of $\mathbb{I}$, and each data point is weighted by the mask value $\mathbb{V}(x, y)$. If the point $p_\mathrm{f}[k]$ is outside the frame, we simply set $\alpha[k] \leftarrow \alpha'[k]$ and $\beta[k] \leftarrow \beta'[k]$. In either case, these parameters are used to estimate the photometric parameters for the next frame, in step 4.

This method may yield very wrong values of $\alpha[k]$ and $\beta[k]$ if the feature is within the present frame but occluded by some foreground object. These wrong parameters may cause the finder to fail when the feature becomes visible again. At that point, the finder is likely to assign a low confidence for the mark, so that the calibrator will ignore it for that frame. However, the algorithm will then predict the correct position p_f for that feature, and the values of $\alpha[k]$ and $\beta[k]$ will then be correctly obtained, so that the feature will be fully recovered in the next frame.

15.10 Processing of frame 2

The second frame of the video ($i = 2$) is processed as described above, except that in we cannot perform the linear extrapolation of feature positions that is normally done in step 3. In that case, step 3 we simply set the guessed feature positions $p'_\mathrm{f}{}^{(2)}[k]$ to the final positions $p_\mathrm{f}{}^{(1)}[k]$ computed for the first frame. In all the other steps, this case is like the general case.

15.11 Processing of frame 1

For the first frame of the video ($i = 1$), the user must explicitly provide approximate positions $p'_\mathrm{f}[k]$ and the corresponding confidence weights $w'[k]$ for a sufficiently large subset of the fea-

tures. In that case, we first skip steps 1–10, and set $p_{\mathrm{f}}''[k]$ to the user given positions. Next we run steps 11–15 to get a calibrated camera parameter record $\mathbb{C}$ for that frame, as well as official positions $p_{\mathrm{f}}[k]$, template images $\mathbb{G}[k]$, $\mathbb{V}[k]$, and parameters $\alpha[k]$ and $\beta[k]$ for all features that are expected to be visible in the frame. Then we run the whole AFFTRACK algorithm (procedure 7) again on the first frame, using those computed values for $\mathbb{C}'$ in step 1, for $\mathbb{G}'[k]$ and $\mathbb{V}'[k]$ in step 5, for $p'_{\mathrm{f}}[k]$ in step 3, and for $\alpha'[k], \beta'[k]$ in step 4.

15.12 Additional iterations

In theory, after computing the official parameters $p_{\mathrm{f}}[k]$, $\mathbb{G}[k]$, $\mathbb{V}[k]$, $\alpha[k]$ and $\beta[k]$ in steps 13–15, we could repeat step 7 to obtain improved feature positions, and then steps 11–15 to improve the calibration. However, our tests indicate that this iteration does not improve the accuracy to a significant extent.

Chapter 16

The feature finding algorithm

The tracker used in this part of the thesis is an implementation of the Kanade-Lucas-Tomasi (KLT) algorithm [81], applied independently to each feature. Its inputs are three images — the feature template $\mathbb{G}$, the corresponding mask $\mathbb{V}$, and the frame $\mathbb{I}$ — and a guessed position $p'_\mathrm{f} = (x'_\mathrm{f}, y'_\mathrm{f})$ for the center of $\mathbb{G}$ in $\mathbb{I}$. It should return an adjusted position $p''_\mathrm{f} = (x''_\mathrm{f}, y''_\mathrm{f})$ near p'_f such that the sub-image $\mathbb{H}(p''_\mathrm{f})$ of $\mathbb{I}$, with the same size as $\mathbb{G}$ and centered at p''_f, is most similar to the template $\mathbb{G}$. Note that the template $\mathbb{G}$ is assumed to incorporate the necessary geometric and photometric adjustments.

The KLT algorithm works at several scales of resolution [15], where the feature position found at each scale is used as a starting guess for the search at the next finer scale. The Lucas-Kanade (LK) algorithm [54] is used at each scale of the KLT algorithm to compute the precise position p''_f of template $\mathbb{G}$ in the frame $\mathbb{I}$, given its approximate position p'_f.

16.1 Discrepancy function

The quality of the match is measured by the *mean quadratic discrepancy function* $E(p)$. The E function for a given placement $p = (x, y)$ of $\mathbb{G}$ in $\mathbb{I}$ can be defined by

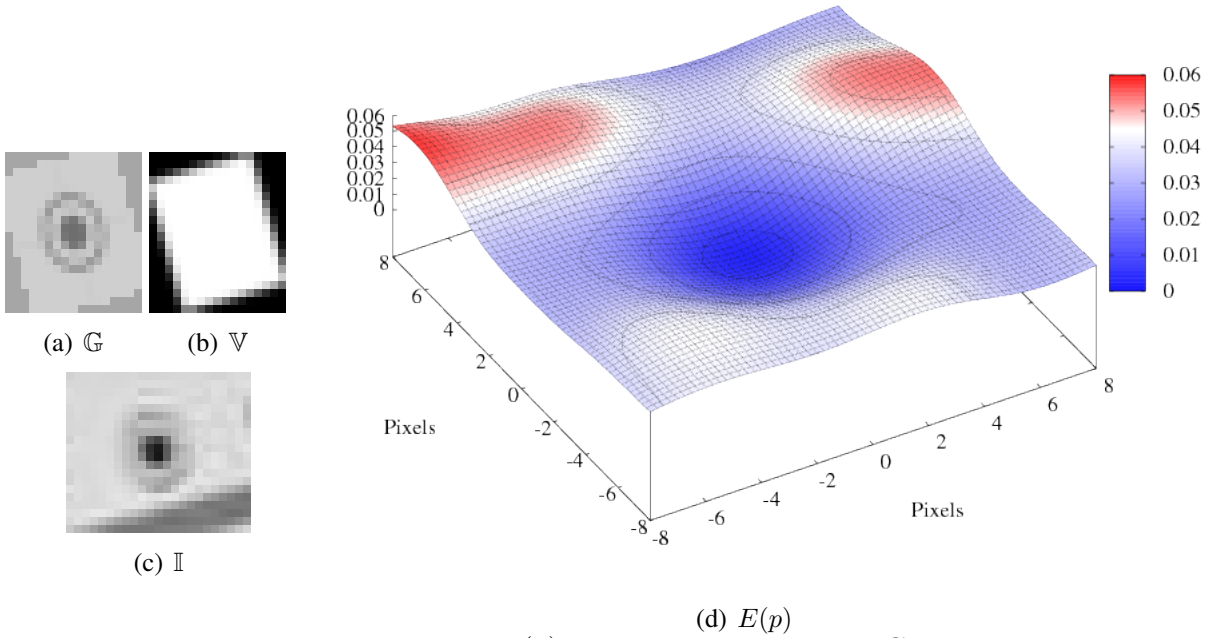
$$E(p) \quad = \quad \int\!\!\int_{\mathfrak{D}} (\mathbb{G}(q) - \mathbb{I}(p+q))^2 \, dq \tag{16.1}$$

where the double integral spans the domain $\mathfrak{D}$ of $\mathbb{G}$.

The abrupt edges of the window $\mathfrak{D}$ have an exaggerated influence in equation (16.1) and tend to increase the number of local minima. To minimize these problems, we used the variant E function defined by Lucas and Kanade [54], which uses a weighting function — in our case the feature mask $\mathbb{V}$. Namely, we replace formula (16.1) by

$$E(p) \quad = \quad \frac{\int\!\!\int_{\mathfrak{D}} \mathbb{V}(q) (\mathbb{G}(q) - \mathbb{I}(p+q))^2 \, dq}{\int\!\!\int_{\mathfrak{D}} \mathbb{V}(q) \, dq} \tag{16.2}$$

To simplify the formulas, we will assume that the integral of $\mathbb{V}$ (the denominator of formula 16.2) is 1. See figure 16.1.



(d) $E(p)$

Figure 16.1: The discrepancy function $E(p)$ for a particular template $\mathbb{G}$, mask $\mathbb{V}$ and image $\mathbb{I}$.

16.2 Discrepancy minimization

The minimization of E can be done by a sequence of steps. We start with a given approximation $p^{(0)} = p'_{\mathrm{f}}$, and at each step we compute a new guess $p^{(i+1)}$ from $p^{(i)}$. The last point $p^{(k)}$ is returned as the presumed feature placement p''_{f}. The iteration is usually stopped after a preset number of steps (30 in our tests), or when the length $|p^{(i+1)} - p^{(i)}|$ of a step is bellow some given threshold $\varkappa$.

For each step we use the Lucas-Kanade formula [10], as implemented by Birchfield [13, 5]. In our tests, we considered the minimization to have failed if the step $p^{(i+1)} - p^{(i)}$ exceeded 2 pixels in either axis. See figure 16.2.

16.3 Image interpolation

In typical applications, the position p''_{f} returned by the finder must have sub-pixel accuracy. Therefore, $E(p)$ must be defined for points p with arbitrary fractional coordinates. Moreover,

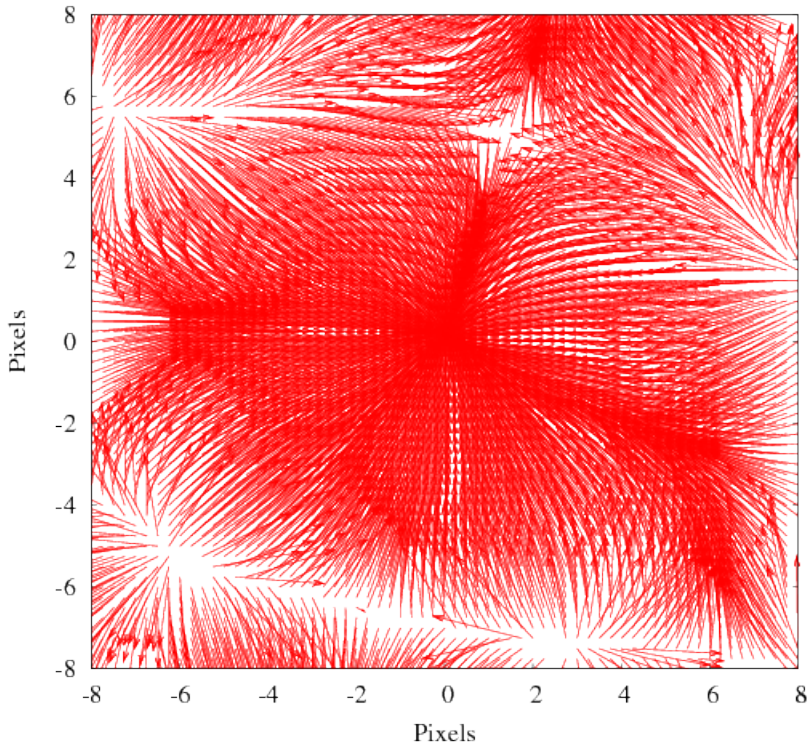


Figure 16.2: The LK step, $p^{(i)} \rightarrow p^{(i+1)}$, for the data of figure 16.1, for various choices of $p^{(i)}$ in the neighborhood of the minimum (at center).

the LK algorithm requires the evaluation of $\nabla\, \mathbb{G}(p)$ and $\nabla\, \mathbb{I}(p)$ at several probe points p. These image gradients are computed by interpolating the pixel values and differentiating the interpolation splines, so that the function and its derivatives are consistent. To ensure that E is smooth enough, we used a biquadratic smoothing spline formula that is C_1 in the position argument. (The cheaper bilinear interpolation, which is only C_0, would cause the LK method to fail.) The integrals are computed as summations over a sufficiently dense grid of sampling points over $\mathfrak{D}$.

16.4 Multiscale search

For the KLT algorithm, we first build a pyramid of images $\mathbb{G}^{(i)}, \mathbb{V}^{(i)}$ and $\mathbb{I}^{(i)}$, for $i = 0, 1, 2, \ldots, m$, where $\mathbb{G}^{(0)}, \mathbb{V}^{(0)}$ and $\mathbb{I}^{(0)}$ are the original (full scale) images, and the images at each successive level are obtained from those of the previous level through reduction by a factor of 2 in each dimension. The final level m is such that the template domain $\mathfrak{D}$ is between 5 and 10 pixels across in the smallest direction. The LK algorithm is applied first at scale m

with initial guess $p'_{\mathrm{f}}/2^m$. Then at each lower level $i = m-1, m-2, \ldots, 0$, the LK algorithm is applied using as initial guess the optimum placement p''_{f} found at the scale $i+1$, multiplied by 2.

16.5 Confidence weight

The finder also returns its estimate w'' of the probability that $p_{\mathrm{f}}{}''$ is a meaningful result — that is, not an outlier. Ideally, $E(p''_{\mathrm{f}})$ is zero for a perfect match. However, image noise and interpolation errors will result in a positive discrepancy $E(p''_{\mathrm{f}})$ even at the correct position. Assuming that these errors can be modeled by additive Gaussian of variance σ^2, the expected value for $E(p''_{\mathrm{f}})$, at the true placement p''_{f}, will be proportional to σ^2. Therefore, we set $w'' = \exp(-r^2/2)$, where $r^2 = (E(p''_{\mathrm{f}}) + \epsilon^2)/\sigma^2/(\alpha^2 + \epsilon^2)$. The denominator α^2 is meant to reduce the confidence when the apparent contrast α of the feature in the image $\mathbb{I}$ is low. The parameter ϵ is a safety term added to avoid division by zero (set to 0.005 in our tests). We also set $p_{\mathrm{f}}{}''$ to zero if the tracker failed to converge at any level.

Chapter 17

The camera calibrator

AFFTRACK's calibrator takes a list $\mathbb{D}$ of m *data pairs* as input, where each pair consists of the world coordinates $p_\mathrm{s}[k]$ of a point on the scene, its apparent coordinates $p''_\mathrm{f}[k]$ in the frame image, and a confidence weight $w''[k]$, for $k = 1, \ldots, M$. It outputs the parameter tuple $\mathbb{C} = (S, f, \kappa, C_\mathrm{x}, C_\mathrm{y}, d_\mathrm{x}, d_\mathrm{y})$ that provides the best fit to that data.

For most cameras, the parameters $C_\mathrm{x}, C_\mathrm{y}, d_\mathrm{x}, d_\mathrm{y}$ are fixed, and usually known from the manufacturer's specifications; although the parameters C_x and C_y may need one-time calibration to account for minor mis-alignments of the lens relative to the sensor array. In cameras with fixed lenses, the values of f and κ are also fixed and need to be calibrated only once for all videos with the same camera. Such cameras have only six degrees of freedom, namely the rotation an translation components of the matrix S; or only three, if the camera's position V is fixed. On the other hand, in movable cameras with variable zoom, both f and κ are variable, so there are eight parameters that need to be determined for each frame.

Besides the calibration data $\mathbb{D}$, the calling program should also specify a range of permitted values for each parameter of $\mathbb{C}$. If the range is trivial (reduced to a single value), the parameter is effectively fixed at that value, and is automatically excluded from the calibration. At present, our procedure requires that the values of n_x, n_y, d_x, and d_y be fixed in this way. The procedure can be used to calibrate any reasonable subset of the remaining 16 parameters (S, f, κ, C_x and C_y). However, the sub-matrix R of S is constrained to be a rotation matrix, so there are only 10 degrees of freedom in $\mathbb{C}$.

There is also a partial redundancy between the focal length f and the mean distance $\bar{z}_\mathrm{c}$ from the camera to the scene: the effect of doubling f while halving $\bar{z}_\mathrm{c}$ can be hardly noticeable, especially if the field of view is very narrow. This is a serious difficulty in the calibration of any camera with a variable zoom lens.

17.1 Rough calibration

The calibration begins by computing the matrix S and focal length f for a camera that is located very far from the scene and which best matches the given data. For such a camera, the scaling factor f/z_{c} in the perspective projection formula (14.2) is essentially the same value μ for all points. Such a camera therefore performs a parallel (rather than conical) projection, so that the undistorted coordinates $p_{\mathrm{u}}[k]$ are affine functions of its object coordinates $p_{\mathrm{s}}[k]$. That is,

$$p_{\mathrm{u}}[k] = Mp_{\mathrm{s}}[k] + b \tag{17.1}$$

where M is some 2×3 matrix and b is some 2-vector. The procedure therefore looks for the matrix M and vector b that best fit the available data in the weighted least square sense. Namely, it computes the M and b that minimize the *quadratic undistorted projection error metric*

$$P(M,b) = \sum_{k=1}^{n} w[k](p_{\mathrm{u}}^{\mathrm{a}}[k] - p_{\mathrm{u}}^{\mathrm{o}}[k])^2 \tag{17.2}$$

where $p_{\mathrm{u}}^{\mathrm{a}}[k]$ is computed from $p_{\mathrm{s}}[k]$ by formula (17.1), while $p_{\mathrm{u}}^{\mathrm{o}}[k]$ is computed from $p_{\mathrm{f}}[k]$ by inverting formula (14.4) and then applying formula (14.3), with C_{x}, C_{y}, and κ fixed at the midpoint of the respective client-specified ranges. The best-fitting M and b can be found by standard linear least-squares fitting. In order to avoid large numbers in the matrix and reduce the order of the system, we replace the object coordinates $p_{\mathrm{s}}[k]$ by $p_{\mathrm{s}}^{*}[k] = p_{\mathrm{s}}[k] - \bar{p}_{\mathrm{s}}$, where $\bar{p}_{\mathrm{s}}$ is the barycenter of all object coordinates $p_{\mathrm{s}}[k]$, weighted by the corresponding $w[k]$. Similarly, we replace the undistorted coordinates $p_{\mathrm{u}}[k]$ by $p_{\mathrm{u}}^{*}[k] = p_{\mathrm{u}}[k] - \bar{p}_{\mathrm{u}}$, where $\bar{p}_{\mathrm{u}}$ is the barycenter of all $p_{\mathrm{u}}[k]$.

Noting that the terms $\sum_{k=1}^{n} w[k]p_{\mathrm{u}}^{*}[k] = 0$ and $\sum_{k=1}^{n} w[k]Mp_{\mathrm{s}}^{*}[k] = 0$, it is easy to check that

$$\begin{aligned} P(M,b) &= \sum_{k=1}^{n} w[k] \mid (p_{\mathrm{u}}^{*}[k] + \bar{p}_{\mathrm{u}}) - A(p_{\mathrm{s}}^{*}[k] + \bar{p}_{\mathrm{s}}) - b \mid^2 && (17.3) \\ &= \sum_{k=1}^{n} w[k] \mid p_{\mathrm{u}}^{*}[k] - Mp_{\mathrm{s}}^{*}[k] \mid^2 + \sum_{k=1}^{n} w[k] \mid \bar{p}_{\mathrm{u}} - M\bar{p}_{\mathrm{s}} - b \mid^2 && (17.4) \end{aligned}$$

Moreover, as M and b minimize $P(M,b)$, we conclude that $b = \bar{p}_{\mathrm{u}} - M\bar{p}_{\mathrm{s}}$, since the first term depends only on M and this choice cancels the second term. Therefore, the problem is reduced to the minimization of the first term and we can determine the latter by solving a 6×6 system instead of an 8×8 system.

If the camera did perform an orthogonal projection with correct $d_{\mathrm{x}}, d_{\mathrm{y}}$ parameters, the two rows r_{a} and s_{a} of the matrix M should be orthogonal vectors with the same length μ. However, since the affine model (17.1) ignores the distortions of conical projection, and may use incorrect

values of $C_\mathrm{x}, C_\mathrm{y}, d_\mathrm{x}, d_\mathrm{y}$, and κ, the two rows of M may not be quite orthogonal and may have different lengths. Therefore, we compute two *orthogonal* unit vectors $X_\mathrm{s}, Y_\mathrm{s}$ and a magnitude μ so that $r_\mathrm{a} \approx \mu X_\mathrm{s}$ and $s_\mathrm{a} \approx \mu Y_\mathrm{s}$. See Figure 17.1.

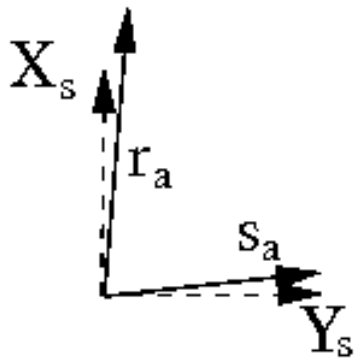


Figure 17.1: Orthogonalization of r_a e s_a.

We then compute a third unit vector $Z_\mathrm{s} = X_\mathrm{s} \times Y_\mathrm{s}$, and take $X_\mathrm{s}, Y_\mathrm{s}, Z_\mathrm{s}$ as the rows of the camera's rotation matrix R. Next we compute the parameters $T_\mathrm{x}, T_\mathrm{y}$ so that the barycenter $\bar{p}_\mathrm{s}$ of the features projects onto the barycenter of the undistorted coordinates, that is

$$T_\mathrm{x} = \frac{\bar{p}_\mathrm{u}[x]}{\mu} - X_\mathrm{s} \cdot \bar{p}_\mathrm{s} \qquad T_\mathrm{y} = \frac{\bar{p}_\mathrm{u}[y]}{\mu} - Y_\mathrm{s} \cdot \bar{p}_\mathrm{s} \tag{17.5}$$

Finally we set T_z to an arbitrary value much larger than the distance $|p_\mathrm{s}[k] - p_\mathrm{s}[j]|$ between any two features, and then set the focal distance f that provides the approximately correct value of μ, namely, $f = \mu(T_\mathrm{z} + R\bar{p}_\mathrm{s})$.

If the data includes at least four non-coplanar points $p_\mathrm{s}[k]$, this method usually yields a reasonable approximation to the camera matrix R, even in the worst case when the camera has a wide field of view and is close to the features (T_z comparable to ρ_s). This method fails when all the feature positions $p_\mathrm{s}[k]$ are coplanar. In that case, there are infinitely many affine models that minimize $P(M, b)$. We will not discuss this case here.

17.2 Calibration refinement

The rough camera parameter vector $\mathbb{C}$, obtained as described in Section 17.1, is then adjusted so as to achieve the best fit between the parameters $\mathbb{C}$ and the data pairs $\mathbb{D}$. The goodness of the fit is quantified by a (*mean quadratic*) *error function* $Q(\mathbb{D}, \mathbb{C})$. This function can be defined in several ways. The *sensor-space error*, for example, measures the discrepancy between the given sensor coordinates p_f of each data point and the sensor coordinates predicted by mapping its object coordinates p_s according to the equations (14.1–14.4) with the proposed parameters $\mathbb{C}$. The

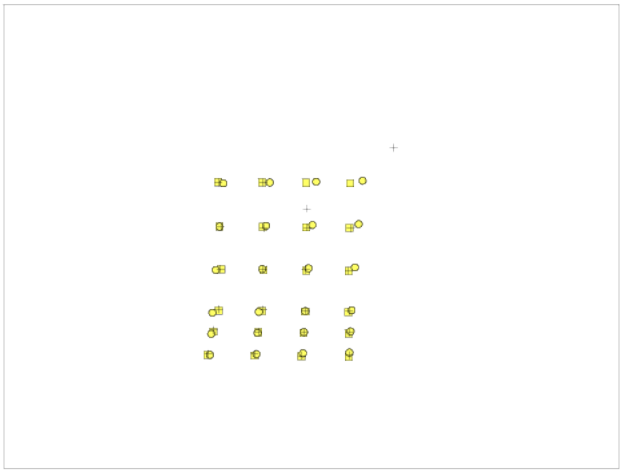


Figure 17.2: Example of rough camera calibration. The crosses are the given apparent positions $p_\mathrm{f}[k]$; the circles are the positions computed from the calibrated $\mathbb{C}$; and the squares are the correct feature positions computed from the world coordinates $p_\mathrm{s}[k]$ with the correct $\mathbb{C}$. Note that we have two obvious outliers in the apparent given positions. This is the reason that the calibrate positions (circles) are shifted from the correct positions (squares).

object-space error proceeds in the inverse direction: it measures the discrepancy between the given object coordinates p_s of each data point, and the ray of all object space points that project to its given sensor position p_f. In both cases, the value of $Q(\mathbb{D}, \mathbb{C})$ is the squared Euclidean distance between the predicted and given coordinates, averaged over all data pairs.

While these two functions may give very different results in some cases, they are practically equivalent when the data set $\mathbb{D}$ is large and fairly accurate. In our implementation, we use the sensor-space metric, defined as

$$Q(\mathbb{D}, \mathbb{C}) = \sum_{k=1}^{n} w''[k](p_\mathrm{f}[k] - p''_\mathrm{f}[k])^2 \tag{17.6}$$

where $p''_\mathrm{f}[k]$ are the frame coordinates of feature k from the input data $\mathbb{D}$, and $p_\mathrm{f}[k]$ are the frame coordinates computed from $p_\mathrm{s}[k]$ and $\mathbb{C}$.

Like R. Willson [89], we use the algorithm `lmdif`, from the Netlib/MINPACK package, which is designed to minimize a real n-variate function that is the sum of many approximately quadratic terms. Note that the metric Q depends on $\mathbb{C}$ in a highly non-linear way. However, for small changes in $\mathbb{C}$, each error term $p_\mathrm{f}[k] - p''_\mathrm{f}[k]$ is approximately an affine function of the parameters, so formula 17.6 is approximately quadratic on them.

In order to improve the numerical stability and reduce the number of variables in the optimization, we replace the parameters $T_\mathrm{x}, T_\mathrm{y}, T_\mathrm{z}$ by the camera's object coordinates $V_\mathrm{x}, V_\mathrm{y}, V_\mathrm{z}$ and the rotation sub-matrix R of S by the equivalent Euler angles R_0, R_1, R_2. We also replace the

focal length f by $\log(f/\bar{z}_c)$ where $\bar{z}_c$ is the mean z coordinate of the features in the camera's system. Note that the ratio $f/\bar{z}_c$ defines the overall spread of the target points on the frame image, and that its effect is roughly orthogonal to that of the camera's position.

17.3 Iterative weight adjustment

After finding the parameters $\mathbb{C}$ that minimize the error function (17.6), we recompute the confidence weights $w[k]$ by Bayesian classification. Specifically, we assume that the frame position errors $e_f[k] = p_f[k] - p''_f[k]$ are two-dimensional Gaussian variables, with different variances σ_i for inliers and σ_o for outliers. The input weights $w''[k]$ are taken to be the a priori probabilities of the points being inliers. We use Bayes's formula to compute $w[k]$ as the a posteriori probability of $p''_f[k]$ being an inlier.

17.4 Effect of input weights

Because the discrepancy function Q incorporates the weights $w[k]$ provided by the feature finder, those data pairs which resulted from more precise template matching will have more influence on the calibrated parameters $\mathbb{C}$. In particular, setting $w[k]$ to zero excludes the data pair $p''_f[k]$ from the calibration. This should be the case, for example, whenever the point $p''_f[k]$ expected to be occluded or outside the current frame.

We have found that the weights provided by the feature finder, based on the match quality, are enough to ensure robust calibration, even without RANSAC-like outlier rejection. Note also that any outlier rejection can be emulated by setting $w[k]$ to 1 for the accepted data pairs, and to 0 for the rejected ones. Compare the calibration results with uniform weights (Figure 17.2) against the calibration with finder confidence weights (Figures 17.3, 17.4, and 17.5) to the same dataset. On the other hand, the weights $w[k]$ allow the calibrator to use all information present in the given apparent positions $p_f[k]$ — even the few extra bits provided by features that were on the borderline between inliers and outliers. Conversely, it makes the calibrator less sensitive to low-quality data pairs that would barely pass the inlier acceptance criterion.

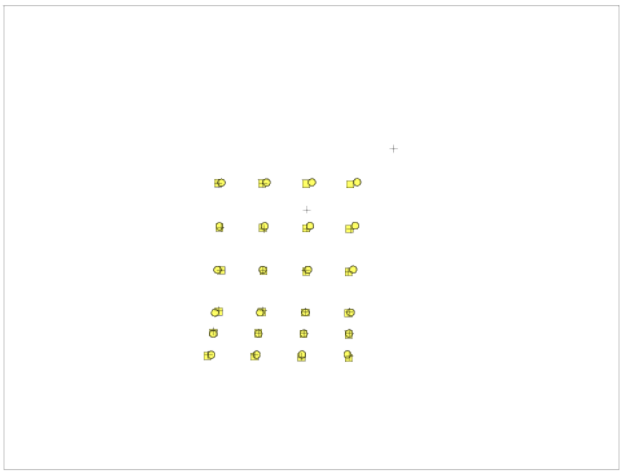


Figure 17.3: Calibration refinement with variable weights: adjusted positions with the rough calibration with the camera at infinity. The initial confidence weights $w[k]$ were 0.9 for all data points, except for the two outliers, which assigned confidence weights 0.30 (upper) and 0.05 (lower).

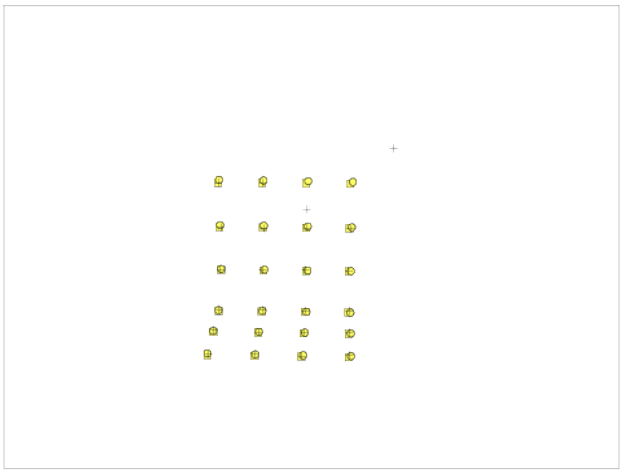


Figure 17.4: Weighted camera calibration: adjusted positions after the refined calibration. The initial positions are the ones given by the rough calibration shown in Figure 17.3.

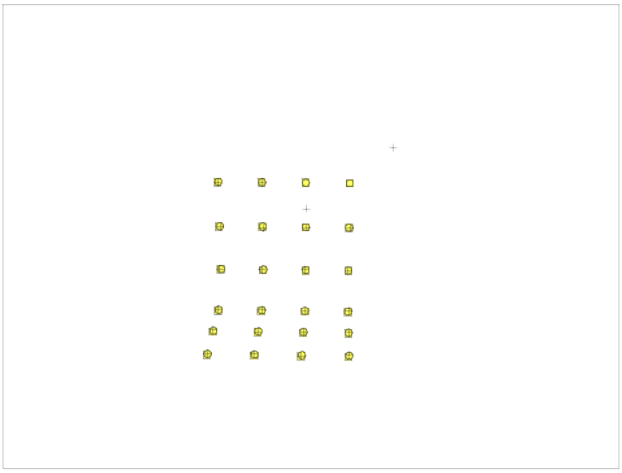


Figure 17.5: Weighted camera calibration: adjusted positions after the iterative weight adjustment. The initial positions and weights are show in Figure 17.4.

Chapter 18

Experiments

18.1 Datasets

We tested AFFTRACK on eight real and nine computer-generated videos [61], summarized in Table 18.1. Note that these videos are different of those used in the text tracking problem (v1-v6) of part II.

18.1.1 Real videos

The real videos `v01`–`v08` were shot with hand-held consumer-grade digital video recorders (Canon Optura 40 or Sony DSR-40) with variable zoom. They were recorded in MPEG format at either 320×240 or 640×480 resolution, 14.98 or 29.99 frames per second, and "standard" image quality setting. The videos were converted to separate PGM frames with `ffmpeg` [2]. In videos `v01`–`v06`, the object to be tracked was a test stage consisting of two perpendicular boards, measuring $285 \times 337 \times 261\,\mathrm{mm}$, each bearing an array of 3×4 "butterfly" fiducial marks measuring $27 \times 38\,\mathrm{mm}$. The camera-to-stage distance varied between 1 and 3 meters. In videos `v06`–`v08`, the object was a room with 12 fiducial marks, measuring $210 \times 297\,\mathrm{mm}$, attached on two perpendicular walls. The camera moved between 3 to 6 meters from the walls. In both cases the fiducial marks were laser-printed patterns on white paper. In some videos, some of the marks were occluded and/or shadowed by moving foreground objects or humans.

18.1.2 Synthetic videos

The synthetic videos `v09`–`v17` were produced with `POV-Ray` [4] from various 3D models (including a virtual version of the stage, above, and a realistic office model `v16` and `v17`), at the same simulated frame rate. The resolution was 640×480 to ARToolKit except for `v09` and `v14` with were 320×240. In some of the videos, the frame images generated by POV-

Video	Source	S	n	T	m	Notes.
v01	Optura	320×240	338	22.5	24	[e, z]
v02	Optura	320×240	270	18.0	8	[a,e,z]
v03	Optura	320×240	301	20.1	8	[a,e,o]
v04	Optura	320×240	244	16.3	24	[e,o,z]
v05	Optura	320×240	130	08.6	24	[e,o,z]
v06	Optura	320×240	233	15.6	28	[a,e]
v07	Sony	640×480	639	20.3	12	[e,o,z]
v08	Sony	640×480	441	14.0	10	[e,o,z]
v09	POV-Ray	320×240	501	33.4	24	[z]
v10	POV-Ray	640×480	501	33.4	8	[a]
v11	POV-Ray	640×480	501	33.4	8	[a,e,o]
v12	POV-Ray	640×480	251	16.8	32	[a,e]
v13	POV-Ray	640×480	251	16.8	28	[a,o]
v14	POV-Ray	320×240	501	33.4	24	[e,o,z]
v15	POV-Ray	640×480	501	33.4	8	[a]
v16	POV-Ray	640×480	501	33.4	26	[o,z]
v17	POV-Ray	640×480	501	33.4	26	[o,z]

Table 18.1: Characteristics of test videos: frame size S in pixels, number of frames n, duration T in seconds, number of features m. Notes: [a] scene includes ARToolKit compatible patterns; [e] frames affected by natural or artificial noise, distortion and blurring; [o] with occlusion of some features; [z] with variable zoom.

Ray were modified by radial distortion, mixing with 5% Gaussian noise, Gaussian blurring, and conversion to and from JPEG format at 85% quality, to simulate the lower image quality of real videos.

18.2 Processing

The videos were processed with (1) our integrated AFFTRACK algorithm (2) with H. Kato's ARToolKit tracker [40], and (3) with the recursive tracker included in the `OpenCV` library [16] followed by R. Willson's implementation of Tsai's camera calibrator [83, 89]. ARToolKit could be used only on those videos which included its special patterns and were shot with fixed focal length (with note [a] but not [z] in Table 18.1). The outer corners of each ARToolKit pattern were treated as four separate features in the other two trackers. For AFFTRACK and OpenCV, the frame coordinates of each feature on the first frame (only) were picked by hand.

18.3 Evaluation

The results of the tests are summarized in Table 18.2. For each program, we classified each

Video	AFFTRACK		ARToolkit		OpenCV	
	P	$\bar{q}$	P	$\bar{q}$	P	$\bar{q}$
v01	100	—	—	—	100	—
v02	100	—	100	—	100	—
v03	100	—	50	—	44	—
v04	100	—	—	—	4	—
v05	100	—	—	—	0	—
v06	100	—	100	—	4	—
v07	100	—	—	—	16	—
v08	100	—	—	—	13	—
v09	100	0.131	—	—	92	2.362
v10	100	0.547	100	0.878	45	2.490
v11	100	0.762	93	0.843	10	3.562
v12	100	0.629	100	1.410	50	4.412
v13	100	0.352	99	1.298	3	4.349
v14	100	0.621	—	—	20	2.024
v15	100	0.647	0	85.935	26	3.716
v16	100	0.625	—	—	99	2.434
v17	100	0.707	—	—	41	2.092

Table 18.2: Calibration and tracking results: percentage P of successfully calibrated frames, and root-mean-square frame position error $\bar{q}$ in the successful frames.

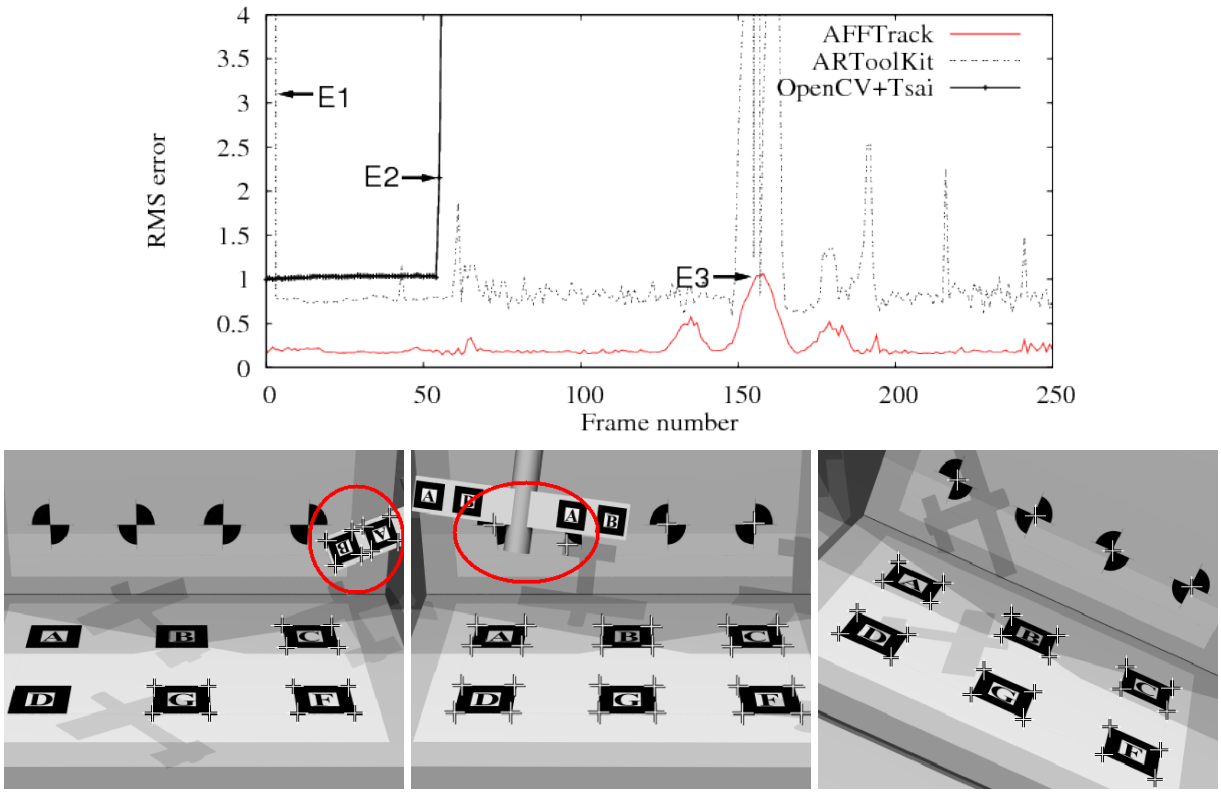
frame as 'success' or 'failure' according to whether the calibration produced minimally usable parameters or not. For the synthetic videos, for which the true camera parameters $\hat{\mathbb{C}}^{(i)}$ were known, we considered frame i to be a success if the root mean square error

$$q^{(i)} = \left[\frac{1}{m} \sum_{1}^{m} w[k] |{p_\mathrm{f}}^{(i)}[k] - {\hat{p}_\mathrm{f}}^{(i)}[k]|^2 \right]^{1/2} \tag{18.1}$$

was less than 8 pixels, where $\hat{p_\mathrm{f}}[k]$ is the position of feature k implied by $p_\mathrm{s}[k]$ and $\hat{\mathbb{C}}^{(i)}$. For the real videos, we visually classified as 'failure' every frame for which the image positions computed by any two of the programs disagreed by more than 8 pixels in the RMS sense, and counted all other frames as 'successes'.

18.4 Discussion

These and other tests confirmed that AFFTRACK is free from long-term drift, even though it carries information from one frame to the next. AFFTRACK also has consistently less jitter than the other two trackers. See Figure 18.1.



(a) Event **E1** (b) Event **E2** (c) Event **E3**

Figure 18.1: RMS feature position error $q^{(i)}$ along video v13. (a) event **E1**: frame 02 showing failure of the ARToolKit's finder due to feature confusion ($\bar{q} = 154.240$); (b) event **E2**: frame 56 showing OpenCV tracker drift due to the airplane feature occlusion ($\bar{q} = 2.155$); (c) event **E3**: frame 155 showing the maximum error of AFFTRACK due to camera roll motion ($\bar{q} = 0.975$).

ARToolkit is also free from long-term drift, although its RMS frame position error is about twice as large as AFFTRACK's. On the other hand the OpenCV+Tsai combination often exhibits runaway drift in fast-moving videos.

As shown in Table 18.2, our algorithm successfully calibrated 100% of the frames in all videos, whereas the other two trackers had significant failure rates, sometimes up to 100%. Be-

sides being prone to runaway drift, the OpenCV tracker also tends to fails permanently whenever any feature becomes occluded.

The ARToolKit tracker is also sensitive to occlusion, but usually recovers once the features become visible again. On the other hand, ARToolKit locates each target independently so it often matches the wrong pattern (see Figure 18.1(a)), and/or the correct pattern in the wrong orientation. Such pattern orientation errors are the reason for ARToolKit's large $\bar{q}$ on video `v15`.

In video `v04`, as the ruler swept over each feature, the AFFTRACK finder began returning incorrect positions $p''_f[k]$. However, the large discrepancy $E[k]$ of these false matches resulted in a low confidence weight $w''[k]$, so that data pair was effectively ignored by the calibrator. In the next frame, the initial guess for the feature's position was computed from the TCC-calibrated parameter $\mathbb{C}$, rather than from the KLT output, so it was essentially correct, even though the feature was not visible. So, AFFTRACK is resilient to occlusions as long as a sufficient number of features remain visible and can promptly recovered them once the features became visible again. See Figure 18.2 (top). This is true even when the occluding object has markings identical to the tracked features, as in Figure 18.2 (bottom).

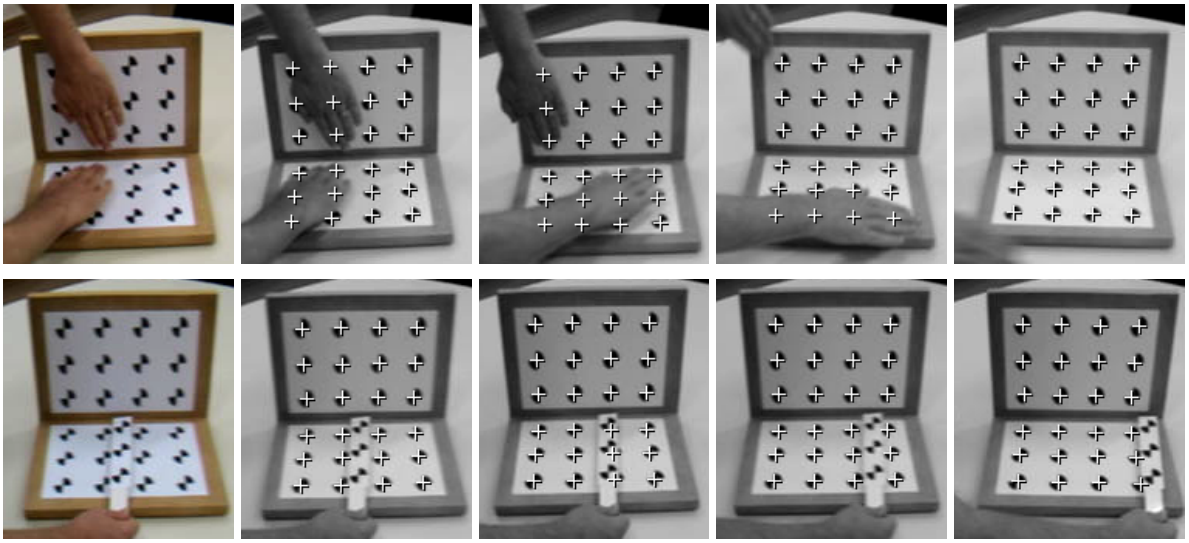


Figure 18.2: Frames from video v05 showing AFFTRACK's robustness to occlusion (top) and frames from video v04 showing AFFTRACK's resistance to feature confusion (bottom).

We also tested the performance of AFFTRACK under common problems as lens distortion and image noise. As shown in Figure 18.3, even with small fiducial marks and partial object occlusion, our algorithm was able to calibrate the whole video without failures.

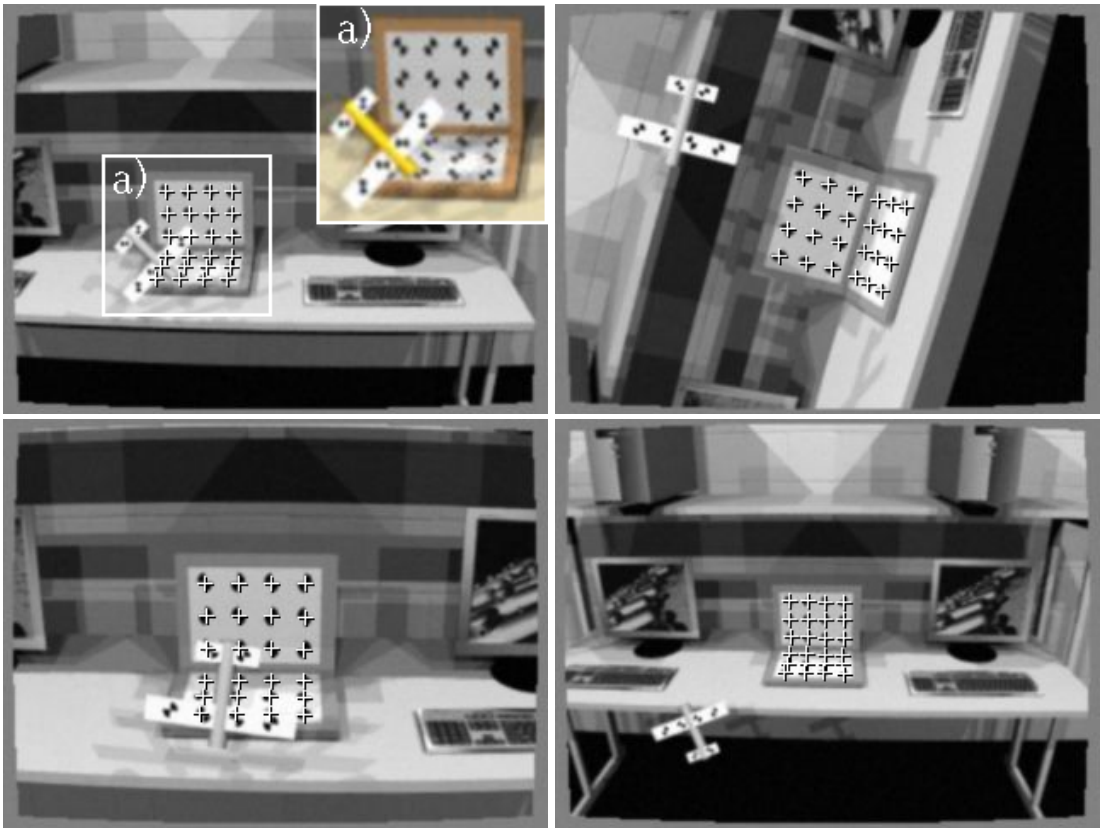


Figure 18.3: Frames from the synthetic video v13 showing occlusion with similar marks over the airplane in addition with camera distortion effect (see the table lines) and artificial noise to simulate a real scenario.

Finally, we used the AFFTRACK's algorithm to calibrate a real scenario with human occlusions, see Figure 18.4(top). As in Augmented Reality applications, we artificially inserted a virtual lamp in the roof of the room. See Figure 18.4(bottom). Note that the lamp positions are changed according to the camera position.

Figure 18.4: Augmented reality: AFFTRACK's tracking and calibration (crosses) in a real scenario with human occlusions and illumination variation (top). A virtual lamp (bottom) was inserted in the real scenario using the camera calibration.

Chapter 19

Conclusions

In this thesis, we worked on three computer vision problems: the detection and classification of flat text objects in images of real scenes; the tracking of such text objects in a digital video; and the tracking an arbitrary rigid 3D object with known shape in a digital video.

19.1 Text detection and classification

In Part I of this thesis, we investigated the use of the histogram of oriented gradients (HOG) for text classification. In this part, we reported extensive experiments with the Dalal and Triggs's multiple HOG descriptor (R-HOG) and SVM classification for the text/non-text discrimination problem. These experiments showed that the optimum cell configuration, for any descriptor size, consists of non-overlapping horizontal bands, in a single weighting and normalization block. Splitting the sub-image by vertical cuts is never cost-effective. In retrospect, this conclusion makes sense, given the nature of the 'object' to be recognized – a single line of text of arbitrary contents and length. The experiments also showed that conflating opposite gradient orientations slightly degrades the R-HOG performance for text classification.

We then defined another multiple HOG descriptor, the T-HOG, whose cells have fuzzy boundaries defined by overlapping Gaussian weight functions. With another exhaustive series of experiments, we confirmed that the best cell arrangement for the T-HOG+SVM text recognizer is also a stack of horizontal bands. In these experiments we also determined the best values for the number of cells n_y and the number of bins n_b, for each descriptor size $N = n_y \times n_b$. In particular, we found that increasing N beyond 100 has practically no effect on recognition accuracy. Those tests also showed that the T-HOG+SVM classifier consistently outperforms R-HOG+SVM at text/non-text discrimination, for any descriptor size N.

In the last chapter of Part I, we described the use of T-HOG+SVM in two text-related applications. First, we described and improved a text detector (SNOOPERTEXT) by using the multi-scale technique to handle widely different letter sizes and to render it immune to small-

scale noise and texture. Second, we investigated the use of T-HOG as a post-filter for a high-recall, low-precision version of the SNOOPERTEXT; we showed that T-HOG is better than R-HOG for this application, and that the combination T-HOG+SVM is at least as good as the best text detectors reported in the literature (an application of SNOOPERTEXT + T-HOG to locate streets by textual queries in a real geographic information system is discussed in appendix A). Third, we described a sliding-window text detector based on the T-HOG classifier, and we gave anecdotal evidence of its accuracy.

19.2 Text tracking

In Part II of this thesis, we considered the problem of tracking the projected image of a text object over successive frames of a video. We described and discussed four text tracking algorithms which represent most of the solutions to this problem. Many ideas of these four algorithms are from our SNOOPERTRACK, an algorithm for the automatic detection and tracking of text objects in videos of outdoor scenes. We evaluated the performance of these four strategies on six videos from real urban scenes, which are available for download [62]. For these tests, we developed new separate metrics for detection and tracking accuracy.

The four algorithms differ on how the text detector is combined with the text tracking proper. If one had a fast and completely accurate text detector, the best scheme would be to run it on every frame, and then find the best partial pairing among the regions found in successive frames. However, the detection accuracy f-score of existing text detectors on outdoor videos is still between 50-70%. It that case, we found that the best strategy was to run the detector only at some key frames (about once per second) and use bi-directional tracking to locate the texts in the intervening frames.

For this part of the thesis, we developed a text tracking procedure that uses the particle-filter approach. We used the T-HOG descriptor to evaluate each candidate region (particle), both to compute a "text-like" score via SVM and as an image fingerprint to evaluate the similarity of region contents across frames.

19.3 Feature-based rigid object tracking

Finally, in Part III of this thesis, we considered the problem of tracking a rigid tri-dimensional object by following a set of non-coplanar bi-dimensional features attached to it at known relative positions. Our main contribution in this part was an algorithm, which we called AFFTRACK, that appears to be significantly more accurate and robust than other published methods.

AFFTRACK is a synergistic combination of the KLT feature tracker with an improved version of Tsai's camera calibration algorithm. The two procedures are integrated so that infor-

mation flows in both directions: the calibrator provides reliable predictions of feature positions and deformations to the feature finder, while the latter provides to the former the adjusted feature positions and estimates of their reliability. As a result, AFFTRACK can track an object in the presence of occlusions, video noise, and tracking failures for an indeterminate number of frames, without any long-term drift. AFFTRACK can also handle video shots with variable zoom (and therefore with variable radial distortion).

Our camera calibration procedure uses real-valued confidence weights to allow a robust calibration even in the presence of outliers (occasional widely incorrect feature positions). These weights initially measure the quality of the match returned by the feature finder, and are then iteratively adjusted according to the consistency between the feature's reported position and the position of all other features. This "gradual" approach to outlier handling contrasts with the sharp inlier/outlier classification used by RANSAC.

To test the AFFTRACK algorithm we compiled a benchmark of 17 videos, annotated with the true feature positions and (when available) the true camera position and orientation for each frame. This benchmark is also available for public download [61].

Appendix A

Keyword Search

The SNOOPERTEXT detector was developed in the context of the ITOWNS project [26], which aims to build tools for virtual navigation of urban environments. The main raw data for these tools is a collection of GPS-tagged high-resolution digital photos of building façades. The purpose of SNOOPERTEXT within this project is to extract (offline) any textual information present in the images, such as street and traffic signs, store names, and building numbers. Therefore, we run SNOOPERTEXT on each image, and process each extracted text box with a publicly available OCR (Tesseract). The user can then retrieve city location images by textual queries on the resulting strings. The text query is matched against each word of the database by computing the Edit distance [46]. Each image containing a matching word is considered as relevant to the query. Figures A.1 and A.2 show examples of querying the database with the word “sushi”.

Figure A.1: Example of keyword search for the query "sushi".

Figure A.2: Example of keyword search for the query "sushi".

Bibliography

[1] Epshtein et al. Text Detection Database. `http://research.microsoft.com/enus/um/people/eyalofek/text_detection_da%tabase.zip`.

[2] FFmpeg. `http://ffmpeg.mplayerhq.hu/`. 2009.

[3] International conference on document analysis and recognition (icdar competition (2003-2005). `http://algoval.essex.ac.uk/icdar/Datasets.html`.

[4] The Persistence of Vision raytracer. `http://www.povray.org`. 2010.

[5] KLT: An implementation of the Kanade-Lucas-Tomasi feature tracker, 2009. `http://www.ces.clemson.edu/~stb/klt/`. Accessed November, 2010.

[6] Fakhr-eddine Ababsa and Malik Mallem. Robust camera pose estimation using 2d fiducials tracking for real-time augmented reality systems. In *ACM SIGGRAPH International Conference on Virtual Reality Continuum and Its Applications in Industry (VRCAI)*, pages 431–435, New York, NY, USA, 2004. ACM.

[7] N. Ahmed, T. Natarajan, and K.R. Rao. Discrete cosine transform. *IEEE Transactions on Computers*, C-23(1):90 –93, jan. 1974.

[8] C. Arth, F. Limberger, and H. Bischof. Real-time license plate recognition on an embedded DSP-platform. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–8, 2007.

[9] Patrick Pérez Aurélie Bugeau. Track and cut: simultaneous tracking and segmentation of multiple objects with graph cuts. Technical report, IRISA, Campus universitaire de Beaulieu, 35042 Rennes Cedex (France), 2007.

[10] Simon Baker and Iain Matthews. Lucas-kanade 20 years on: A unifying framework. *Int. J. Comput. Vision (IJCV)*, 56(3):221–255, February 2004.

[11] Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool. Speeded-up robust features (surf). *Computer Vision Image Understading - Elsevier*, 110:346–359, June 2008.

[12] Bryce Bayer. An optimum method for two-level rendition of continuous-tone pictures. *IEEE International Conference on Communications*, pages 11–15, 1973.

[13] Stan Birchfield. Derivation of kanade-lucas-tomasi tracking equation. Unpublished manuscript. `http://www.ces.clemson.edu/~stb/klt/birchfield-klt-derivation.pdf`, 1997.

[14] James Black and Tim Ellis. Multi camera image tracking. *Image and Vision Computing*, 24(11):1256 – 1267, 2006. Performance Evaluation of Tracking and Surveillance (PETS).

[15] Jean-Yves Bouguet. Pyramidal implementation of the Lucas Kanade feature tracker. description of the algorithm, 2000.

[16] G. R. Bradski and V. Pisarevsky. Intel's Computer Vision Library: Applications in calibration, stereo, segmentation, tracking, gesture, face and object recognition. 2000.

[17] Michael D. Breitenstein, Fabian Reichlin, Bastian Leibe, Esther Koller-Meier, and Luc Van Gool. Online multiperson tracking-by-detection from a single, uncalibrated camera. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33:1820–1833, 2011.

[18] Thierry Chateau, Vincent Gay-Belille, Frederic Chausse, and Jean-Thierry Lapresté. Real-time tracking with classifiers. In *Proceedings of the international conference on dynamical vision*, WDV'05/WDV'06/ICCV'05/ECCV'06, pages 218–231, Berlin, Heidelberg, 2007. Springer-Verlag.

[19] H. Chen, S. Tsai, G. Schroth, D. Chen, R. Grzeszczuk, and B. Girod. Robust text detection in natural images with edge-enhanced maximally stable extremal regions. *IEEE International Conference on Image Processing (ICIP)*, pages 1–4, 2011.

[20] Xiangrong Chen and Alan L. Yuille. Detecting and reading text in natural scenes. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2:366–373, 2004.

[21] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20:273–297, September 1995.

[22] D. Crandall and R. Kasturi. Robust detection of stylized text events in digital video. In *International Conference on Document Analysis and Recognition (ICDAR)*, pages 865–869, 2001.

[23] David J. Crandall, Sameer Antani, and Rangachar Kasturi. Extraction of special effects caption text events from digital video. *International Journal on Document Analysis and Recognition (IJDAR)*, 5(2-3):138–157, 2003.

[24] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 886–893, 2005.

[25] Agencie Nationale de Recherche. The iTowns Dataset (annotated), 2009. `www.itowns.fr/benchmarking.html`.

[26] Agencie Nationale de Recherche. The iTowns project, 2009. `http://www.itowns.fr`.

[27] Boris Epshtein, Eyal Ofek, and Yonatan Wexler. Detecting text in natural scenes with stroke width transform. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 886–893, 2010.

[28] Jonathan Fabrizio, Matthieu Cord, and Beatriz Marcotegui. Text extraction from street level images. *ISPRS Workshop City Models, Roads and Traffic (CMRT)*, 2009.

[29] Mark D. Fairchild. *Color Appearance Models, Second Edition*. Wiley-IST Series in Imaging Science and Technology, Chichester, UK, 2005.

[30] M. A. Fischler and R. C. Bolles. Random Sample Consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Readings in Computer Vision: Issues, Problems, Principles, and Paradigms. Morgan Kaufmann Publishers Inc.*, pages 726–740, 1987.

[31] Myers G. and J. B. Burns. A robust method for tracking scene text in video imagery. In Koichi Kise and David S. Doermann, editors, *International Workshop on Camera-Based Document Analysis and Recognition*, volume 1, Seoul, Korea, August 2005.

[32] J. Gllavata, R. Ewerth, and B. Freisleben. Tracking text in mpeg videos. In *ACM international conference on Multimedia*, pages 240–243, 2004.

[33] Hideaki Goto and Makoto Tanaka. Text-tracking wearable camera system for the blind. In *International Conference on Document Analysis and Recognition (ICDAR)*, pages 141 –145, july 2009.

[34] Shehzad Muhammad Hanif and Lionel Prevost. Text detection and localization in complex scene images using constrained adaboost algorithm. *International Conference on Document Analysis and Recognition (ICDAR)*, pages 1–5, 2009.

[35] S.M. Hanif, L. Prevost, and P.A. Negri. A cascade detector for text detection in natural scene images. In *International Conference on Pattern Recognition (ICPR)*, pages 1–4, Dec. 2008.

[36] Weihua Huang, P. Shivakumara, and Chew Lim Tan. Detecting moving text in video using temporal information. In *International Conference on Pattern Recognition (ICPR)*, pages 1 –4, dec. 2008.

[37] Michael Isard and Andrew Blake. Condensation - conditional density propagation for visual tracking. *International Journal of Computer Vision (IJCV)*, 29:5–28, 1998.

[38] Jean-Michel Jolion and Azriel Rosenfeld. *A Pyramid Framework for Early Vision: Multiresolutional Computer Vision*. 1994.

[39] K. Jung, K. Kim, and A. Jain. Text information extraction in images and video: a survey. *Pattern Recognition (PR) - Elsevier*, 37(5):977–997, May 2004.

[40] H. Kato and M. Billinghurst. Developing AR applications with ARToolKit. In *International Symposium on Mixed and Augmented Reality (ISMAR)*, page 305, 2004.

[41] K. C. Kim, H. R. Byun, Y. J. Song, Y. W. Choi, S. Y. Chi, K. K. Kim, and Y. K. Chung. Scene text extraction in natural scene images using hierarchical feature combining and verification. In *International Conference on Pattern Recognition (ICPR)*, volume 2, pages 679–682, 2004.

[42] Dieter Koller, Gudrun Klinker, Eric Rose, David Breen, Ross Whitaker, and Mihran Tuceryan. Real-time vision-based camera tracking for augmented reality applications. In *ACM Symposium on Virtual Reality Software and Technology (VRST)*, pages 87–94, New York, NY, USA, 1997. ACM.

[43] H. W. Kuhn. The Hungarian method for the assignment problem. *Naval Research Logistic Quarterly*, 2:83–97, 1955.

[44] Miriam Leon, Sergio Mallo, and Antoni Gasull. A tree structured-based caption text detection approach. In *IASTED International Conference on Visualization, Imaging and Image Processing (VIIP)*, 2005.

[45] V. Lepetit and P. Fua. Monocular model-based 3d tracking of rigid objects: A survey. *Foundations and Trends® in Computer Graphics and Vision*, 1(1), 2005.

[46] V. I. Levenshtein. Binary codes capable of correcting deletions, insertions and reversals. *Soviet Physics Doklady.*, 10(8):707–710, February 1966.

[47] Huiping Li and D. Doermann. Automatic text tracking in digital videos. In *IEEE Second Workshop on Multimedia Signal Processing*, pages 21 –26, dec 1998.

[48] Huiping Li, D. Doermann, and O. Kia. Automatic text detection and tracking in digital video. *IEEE Transactions on Image Processing (TIP)*, 9(1):147 –156, jan 2000.

[49] Zhi Li, Guizhong Liu, Xueming Qian, Chen Wang, Yana Ma, and Yang Yang. A video text detection method based on key text points. In *Pacific Rim Conference on Advances in Multimedia Information Processing: Part I*, PCM'10, pages 284–295, Berlin, Heidelberg, 2010. Springer-Verlag.

[50] R. Lienhart and A. Wernicke. Localizing and segmenting text in images and videos. *IEEE Transactions on Circuits and Systems for Video Technology (TCSVT)*, 12(4):256 –268, apr 2002.

[51] Young-Kyu Lim, Song-Ha Choi, and Seong-Whan Lee. Text extraction in mpeg compressed video for content-based indexing. In *International Conference on Pattern Recognition (ICPR)*, volume 4, pages 409 –412 vol.4, 2000.

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

[53] David G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision (IJCV)*, 60:91–110, 2004. 10.1023/B:VISI.0000029664.99615.94.

[54] Bruce D. Lucas and Takeo Kanade. An iterative image registration technique with an application to stereo vision. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 674–679, April 1981.

[55] Simon M. Lucas. Text locating competition - international conference on document analysis and recognition (icdar) (2003-2005). `http://algoval.essex.ac.uk:8080/icdar2005/`.

[56] Simon M. Lucas. Text locating competition results. In *International Conference on Document Analysis and Recognition (ICDAR)*, pages 80–85, 2005.

[57] S.M. Lucas. Icdar 2005 text locating competition results. In *International Conference on Document Analysis and Recognition (ICDAR)*, pages 80–84 Vol. 1, 2005.

[58] A. Martin, G. Doddington, T. Kamm, M. Ordowski, and M. Przybocki. The det curve in assessment of detection task performance. In *Proc. Eurospeech'97*, pages 1895–1898, 1997.

[59] Carlos Merino and Majid Mirmehdi. A framework towards realtime detection and tracking of text. http://www.cs.bris.ac.uk/Research/Vision/texttrack/.

[60] Carlos Merino and Majid Mirmehdi. A framework towards realtime detection and tracking of text. In *International Workshop on Camera-Based Document Analysis and Recognition (CBDAR)*, pages 10–17, 2007.

[61] Rodrigo Minetto. AFFTRACK Dataset: Robust Tracking of Features. `http://www.liv.ic.unicamp.br/~minetto/afftrack`. 2010.

[62] Rodrigo Minetto. Urban Scenes Text Tracking Dataset. `http://www.liv.ic.unicamp.br/~minetto/datasets/text/`. 2011.

[63] Rodrigo Minetto. Detecção robusta de movimento de câmera em vídeos por análise de fluxo ótico ponderado. Master's thesis, Universidade Estadual de Campinas UNICAMP, Instituto de Computação, 2007.

[64] Rodrigo Minetto, Neucimar J. Leite, and Jorge Stolfi. Integrating Tsai's camera calibration algorithm with KLT feature tracking. *Workshop de Visão Computacional (WVC)*, pages 1–6, november 2008.

[65] Rodrigo Minetto, Neucimar J. Leite, and Jorge Stolfi. Afftrack: Robust tracking of features in variable-zoom videos. *IEEE International Conference on Image Processing (ICIP)*, pages 4285–4288, 2009.

[66] Rodrigo Minetto, Nicolas Thome, Matthieu Cord, Jonathan Fabrizio, and Beatriz Marcotegui. Snoopertext: A multiresolution system for text detection in complex visual scenes. *IEEE International Conference on Image Processing (ICIP)*, pages 3861–3864, 2010.

[67] Rodrigo Minetto, Nicolas Thome, Matthieu Cord, Neucimar J. Leite, and Jorge Stolfi. Snoopertrack: Text detection and tracking for outdoor videos. *IEEE International Conference on Image Processing (ICIP)*, pages 505–508, 2011.

[68] Rodrigo Minetto, Nicolas Thome, Matthieu Cord, Jorge Stolfi, Frederic Precioso, Jonathan Guyomard, and Neucimar J. Leite. Text detection and recognition in urban scenes. *IEEE/ISPRS Workshop on Computer Vision for Remote Sensing of the Environment CVRS-ICCV*, pages 227–234, 2011.

[69] Yinan Na and Di Wen. An effective video text tracking algorithm based on sift feature and geometric constraint. In *Advances in Multimedia Information Processing - PCM 2010*, volume 6297 of *Lecture Notes in Computer Science*, pages 392–403. Springer Berlin / Heidelberg, 2010.

[70] Timo Ojala, Matti Pietikäinen, and Topi Mäenpää. Multiresolution gray-scale and rotation invariant texture classification with local binary patterns. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 24:971–987, July 2002.

[71] Paul W. Palumbo, Sargur N. Srihari, Jung Soh, Ramalingam Sridhar, and Victor Demjanenko. Postal address block location in real time. *IEEE Computer*, 25(7):34–42, 1992.

[72] Yi-Feng Pan, Xinwen Hou, and Cheng-Lin Liu. A robust system to detect and localize texts in natural scene images. *IAPR International Workshop on Document Analysis Systems*, pages 35–42, 2008.

[73] Patrick Pérez, Carine Hue, Jaco Vermaak, and Michel Gangnet. Color-based probabilistic tracking. In *European Conference on Computer Vision (ECCV)*, pages 661–675, London, UK, 2002. Springer-Verlag.

[74] John C. Platt. Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. In *Advances in Large Margin Classifiers*, pages 61–74. MIT Press, 1999.

[75] William H. Press, Brian P. Flannery, Saul A. Teukolsky, and William T. Vetterling. *Numerical Recipes: The Art of Scientific Computing*. Cambridge University Press, 1986.

[76] Xueming Qian and Guizhong Liu. Text detection, localization and segmentation in compressed videos. In *International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, volume 2, page II, may 2006.

[77] Thomas Retornaz and Beatriz Marcotegui. Scene text localization based on the ultimate opening. *International Symposium on Mathematical Morphology (ISMM)*, 1:177–188, 2007.

[78] Jean Serra. Toggle mappings. *From pixels to features*, pages 61–72, 1989. J.C. Simon (ed.), Elsevier.

[79] K. Smith, D. Gatica-Perez, J. Odobez, and Sileye Ba. Evaluating multi-object tracking. In *Computer Vision and Pattern Recognition (CVPR) Workshops*, page 36, june 2005.

[80] Makoto Tanaka and Hideaki Goto. Text-tracking wearable camera system for visually-impaired people. In *International Conference on Pattern Recognition (ICPR)*, pages 1–4, dec. 2008.

[81] C. Tomasi and T. Kanade. Detection and tracking of point features. Technical Report CMU-CS-91-132, Carnegie Mellon University, April 1991.

[82] P. H. S. Torr and A. Zisserman. MLESAC: A new robust estimator with application to estimating image geometry. *Computer Vision and Image Understanding (CVIU)*, 78(1):138–156, 2000.

[83] R. Y. Tsai. A versatile camera calibration technique for high-accuracy 3D machine vision metrology using off-the-shelf TV cameras and lenses. *IEEE Journal of Robotics and Automation*, 3(4):323–344, 1987.

[84] Ken Turkowski. Filters for common resampling tasks. In Andrew S. Glassner, editor, *Graphics gems*, pages 147–165. Academic Press Professional, Inc., San Diego, CA, USA, 1990.

[85] L. Vacchetti and V. Lepetit. Stable real-time 3d tracking using online and offline information. *IEEE Trans. Pattern Anal. Mach. Intell. (TPAMI)*, 26(10):1385–1391, 2004.

[86] Christian Vogler, Siome Goldenstein, Jorge Stolfi, Vladimir Pavlovic, and Dimitris Metaxas. Outlier rejection in high-dimensional deformable models. *Image and Vision Computing (IVC) - Elsevier*, 25(3):274–284, 2007.

[87] Xiufei Wang, Lei Huang, and Changping Liu. A new block partitioned text feature for text verification. *International Conf. on Document Analysis and Recognition (ICDAR)*, 0:366–370, 2009.

[88] Greg Welch and Gary Bishop. An Introduction to the Kalman Filter. Technical report, Chapel Hill, NC, USA, 1995.

[89] R. Willson. Tsai camera calibration software. `http://www.cs.cmu.edu/~rgw/TsaiCode.html`. 2010.

[90] Qixiang Ye, Qingming Huang, Wen Gao, and Debin Zhao. Fast and robust text detection in images and video frames. *Image and Vision Computing (IVC) - Elsevier*, 23:565–576, June 2005.

[91] C. Yi and Y. Tian. Text string detection from natural scenes by structure-based partition and grouping. *IEEE Transactions on Image Processing (TIP)*, PP(99):1, 2011.

[92] Alper Yilmaz, Omar Javed, and Mubarak Shah. Object tracking: A survey. *ACM Comput. Surv.*, 38(4):13, 2006.

[93] Jing Zhang and Rangachar Kasturi. Text detection using edge gradient and graph spectrum. *International Conference on Pattern Recognition (ICPR)*, 0:3979–3982, 2010.

[94] Wei Zhang, G. Zelinsky, and D. Samaras. Real-time accurate object detection using multiple resolutions. In *IEEE International Conference on Computer Vision (ICCV)*, pages 1–8, October 2007.

[95] Yu Zhong, Hongjiang Zhang, and Anil K. Jain. Automatic caption localization in compressed video. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 22:385–392, 2000.